Techniken zur schnelleren Berechnung von Gomory-Hu-Bäumen

- Verfahren und experimentelle Analyse -

vorgelegt von
Sascha Grau
Matr.-Nr. 33690

07. September 2007

Betreuer:
Dr. rer. nat. Michael Brinkmeier

Verantwortlicher Hochschullehrer:
Prof. Dr. rer. nat. habil. Manfred Kunde

Fachgebiet Automaten und Formale Sprachen
Institut für Theoretische Informatik
Fakultät Informatik und Automatisierung
Technische Universität Ilmenau

Inventarisierungsnummer: 2007-09-07/114/IN01/2241

Inhaltsverzeichnis

1. Einleitung

Gomory-Hu-Bäume sind Datenstrukturen, um die Lösung des ALL-PAIRS-MINIMUM-CUT-Problems für ungerichtete Graphen zu repräsentieren. Als solche wurden sie 1961, zusammen mit einem wegweisenden Algorithmus für ihre Berechnung, von Ralph E. Gomory und T. C. Hu eingeführt ([GH61]).

Sie enthalten eine kompakte Darstellung der Zusammenhangsverhältnisse eines Graphen und finden heute in sehr verschiedenen Domänen ihre Anwendung. Dazu zählen das Design digitaler Schaltungen ebenso wie die Clusterung von Daten. Außerdem ist ihr Einsatz zur Berechnung anderer graphentheoretischer Problemstellungen interessant. So werden sie in verschiedenen Implementierungen zur Approximation des Traveling Salesman Problems genutzt.

Die Entwicklung neuer Algorithmen zur Berechnung von Gomory-Hu-Bäumen schritt in den vergangenen Jahren vergleichsweise langsam voran. Der überwiegende Großteil der bekannten Verfahren beruht auf Heuristiken und Modifikationen des ursprünglichen Gomory-Hu-Algorithmus. Eine grundlegende Abweichung von diesem zeigte sich 1990 im Algorithmus von Gusfield ([Gus90]). Außerdem stellte [BHPT07] für ungewichtete Eingabegraphen einen Algorithmus auf Basis von *Tree Packing* vor.

Wird das Problem auf gewichteten Graphen mit n Knoten betrachtet, so greifen bisher prinzipiell alle bekannten Algorithmen auf $(n-1)$ Maximalflussberechnungen zwischen bestimmten Knotenpaaren zurück. Diese dominieren im Wesentlichen die für die Berechnung benötigte Zeit.

Im Rahmen dieser Arbeit sollen zunächst die Einsatzmöglichkeiten Maximaler Adjazenzordnungen untersucht werden, um die aufwändigen ersten Maximalflussberechnungen zu ersetzen.

Des Weiteren wird mit der Flusskomponentenzerlegung eine neue Technik ausgearbeitet und analysiert, um die notwendigen Maximalen Flüsse auf möglichst stark verkleinerten Versionen des Eingabegraphen bestimmen zu können und deren Laufzeit somit zu verkürzen. Durch Wiederverwendung des aus bereits getätigten Flussberechnungen gewonnenen Wissens kann sie ebenso Berechnungen völlig einsparen.

Die eingeführten Verfahren wurden implementiert und ihre Eigenschaften auf bekannten Graphenklassen mit denen des Gomory-Hu-Algorithmus verglichen. Die gewonnenen Ergebnisse werden vorgestellt und erläutert.

2. Problemstellung & Definitionen

Um eine Grundlage für die folgenden Kapitel zu schaffen, sollen nun zuerst essentielle Begriffe definiert und danach die bei der Berechnung eines Gomory-Hu-Baumes zu lösende Problemstellung eingeführt werden.

Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit der Knotenmenge V und der Kantenmenge E. Zusätzlich sei mit $w_G : E \to \mathbf{R}^+$ eine *Kantengewichtsfunktion* definiert.

Jede Bipartition von V in $P \subset V$ und $\overline{P} := V \setminus P$ mit $s \in P$ und $t \in \overline{P}$ ist ein s-t-Schnitt. P und $\overline{P}$ heißen Schnittmengen und $C(P) = \{e = \{u, v\} | e \in E; u \in P; v \in \overline{P}\}$ ist die Menge der geschnittenen Kanten oder auch der *Schnittverlauf*. Ein s-t-Schnitt $\{P, \overline{P}\}$ ist bereits durch eine seiner Schnittmengen eindeutig bestimmt, weshalb die Bezeichnung P ausreichend ist.

Im weiteren Verlauf dieser Arbeit wird vor allem die *Kreuzung von Schnittverläufen* immer wieder einer Rolle spielen:

Definition 2.1 ***Schnittkreuzung***
Ein Schnitt P bzw. dessen Schnittverlauf *kreuzt* eine Knotenmenge X, falls dieser zwei Knoten $x, y \in X$ trennt, also $x \in P$ und $y \notin P$ gilt.

Ein Schnitt P bzw. dessen Schnittverlauf *kreuzt* einen anderen Schnitt Q, falls er *beide* Schnittmengen Q und $\overline{Q}$ kreuzt, d.h. keine der Mengen $P \cap Q$, $P \cap \overline{Q}$, $\overline{P} \cap Q$ und $\overline{P} \cap \overline{Q}$ leer ist.

Die Funktion $c_G : \mathcal{P}(V) \to \mathbf{R}^+$ mit $c_G(P) = \sum_{e \in C(P)} w_G(e)$ bestimmt die *Kosten* des Schnittes. Da $c_G(P) = c_G(\overline{P})$ gilt, gehört sie zu den symmetrischen Mengenfunktionen.

Definition 2.2 ***Minimaler s-t-Schnitt***
Ein s-t-Schnitt $\{P, \overline{P}\}$ ist genau dann *minimal*, wenn in G kein s-t-Schnitt $\{Q, \overline{Q}\}$ mit $c_G(Q) < c_G(P)$ existiert. Er ist damit für sein Knotenpaar nicht zwangsweise eindeutig.

Als Sonderform eines Minimalschnittverlaufs, sei weiterhin der Blattschnitt definiert:

Definition 2.3 ***Blattschnitt***
Ein *Blattschnitt* für den Knoten $v \in V$ ist ein Minimaler Schnitt zwischen v und einem weiteren Knoten $x \in V$, mit den Schnittmengen $\{v\}$ und $(V \setminus \{v\})$.

Für einen per Blattschnitt getrennten Knoten $v \in V$ existiert ein Gomory-Hu-Baum des entsprechenden Graphen in dem v als Blatt auftritt.

Mit diesem Grundgerüst, lässt sich nun das während der Berechnung eines Gomory-Hu-Baums eigentlich zu lösende Problem einführen.

Definition 2.4 ***ALL-PAIRS-MINIMUM-CUT***
Die Feststellung von Wert und Verlauf eines Minimalen Schnittes für jedes Knotenpaar des Graphen G bildet das ALL-PAIRS-MINIMUM-CUT Problem.

Zur Repräsentation der Ergebnisse des ALL-PAIRS-MINIMUM-CUT Problems führten Gomory und Hu in ihrer Veröffentlichung [GH61] die Struktur des Gomory-Hu-Baums ein:

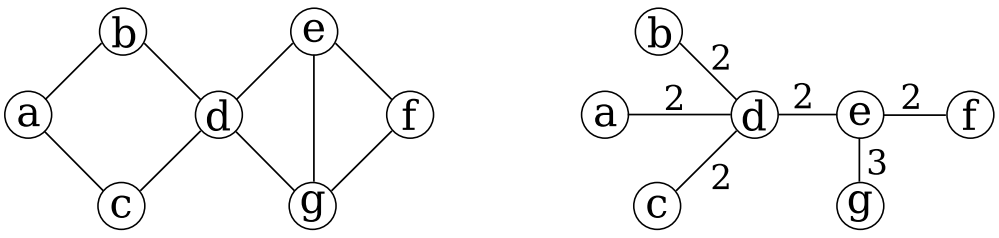
Definition 2.5 ***Gomory-Hu-Baum***
Der Baum $T = (V, A)$ mit der Kantengewichtsfunktion $w_T : A \to \mathbf{R}^+$ ist genau dann ein *Gomory-Hu-Baum* oder auch *Schnittbaum* für den gewichteten, ungerichteten Graphen $G = (V, E)$, wenn er folgende Bedingungen erfüllt:

1. $\forall a \in A$: Sind T_1 und T_2 die Knotenmengen der durch Löschung von a aus T entstehenden Teilbäume, so gilt $w_T(a) := c_G(T_1)$.
2. Die minimal gewichtete Kante auf jedem Pfad zwischen zwei Knoten $s, t \in V$ in T, ist mit dem Minimalen s-t-Schnittwert in G gewertet.

Ein Gomory-Hu-Baum verfügt offensichtlich über nur $n = |V|$ Knoten und damit $n - 1$ Kanten. Es handelt sich nicht um einen Teilgraphen von G, da auch nicht adjazente Knoten aus G in T benachbart sein können.

Wie die in den Abschnitten 3 und 5.3 vorgestellten Algorithmen zeigen, ist der Gomory-Hu-Baum zu einem Graphen G, und damit das ALL-PAIRS-MINIMUM-CUT Problem, in polynomieller Zeit berechenbar.

Das Ziel dieser Arbeit ist es, neue Methoden einzuführen, deren Nutzung es erlaubt, die von den bisher etablierten Algorithmen erreichten Laufzeiten nochmals abzusenken.



(a) Graph G (b) Gomory-Hu-Baum für G

Abbildung 1: Beispiel Gomory-Hu-Baum

3. Bekannte Algorithmen

3.1. Der Algorithmus von Gomory und Hu

Mit ihrer Veröffentlichung [GH61] führten Gomory und Hu bereits 1961 Ideen und Datenstrukturen ein, welche bis heute die Betrachtung des ALL-PAIRS-MINIMUM-CUT Problems bestimmen. Sie ersetzten die bis dahin vorherrschende Matrix-Repräsentation durch einen Schnittbaum, welcher später nach beiden Autoren benannt wurde.

Dies gelang durch die Beobachtung, dass es möglich ist, die $\binom{n}{2}$ Schnittbeziehungen zwischen Knoten des Graphen G, bereits durch $(n-1)$ Schnittverläufe zu charakterisieren. Insbesondere zeigt sich, dass damit bereits $(n-1)$ Schnittberechnungen ausreichend sind, um einen Gomory-Hu-Baum zu berechnen.

Dem Algorithmus liegt folgende Beobachtung zu Grunde:

Lemma 3.1 ***Gomory und Hu, 1961***
Seien X und $\overline{X}$ die beiden Schnittmengen eines Minimalen s-t-Schnittes mit $s \in X$ und $t \in \overline{X}$. Existieren weiterhin zwei Knoten $a, b \in X$ (alternativ $\overline{X}$), so gibt es mindestens einen Minimalen a-b-Schnitt $\{Y, \overline{Y}\}$, welcher die Schnittmenge $\overline{X}$ (alternativ X) nicht kreuzt.

In einem solchen Fall, ist also $\overline{X}$ (alternativ X) vollständig als Teilmenge einer der beiden Schnittmengen Y und $\overline{Y}$ vertreten.

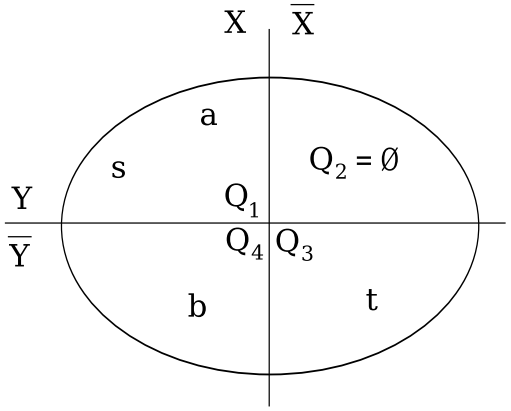


Abbildung 2: Möglicher Schnittverlauf nach Gomory und Hu. Existenz eines a-b-Schnittes Y mit $Q_2 = \emptyset$ garantiert

Abbildung 2 zeigt einen solchen Schnittverlauf. Die Gesamtknotenmenge ist als Oval schematisiert, welches durch den s-t-Schnittverlauf senkrecht geteilt wird. Ein Minimaler a-b-Schnitt verläuft horizontal, wobei hier beispielhaft $s \in Y$ und $t \in \overline{Y}$ festgelegt wurde. Gemäß dem eingeführten Lemma, muss in einer solchen Situation der a-b-Schnitt Y existieren, in welchem der Quadrant Q_2 keine Knoten enthält.

Erst die garantierte Existenz solcher kreuzungsfreien Schnittverläufe macht eine vollständige Zerlegung des Graphen durch $(n-1)$ geschachtelte Schnittverläufe, und damit die Darstellung in Baumform, erst möglich.

Der Algorithmus von Gomory und Hu nutzt Lemma 3.1 nun einerseits zur Begrenzung der Anzahl von ihm eingesetzter Schnittoperationen und zur geschickten Beschleunigung der von ihm als Teilalgorithmus getätigten Flussberechnungen.

Verlauf

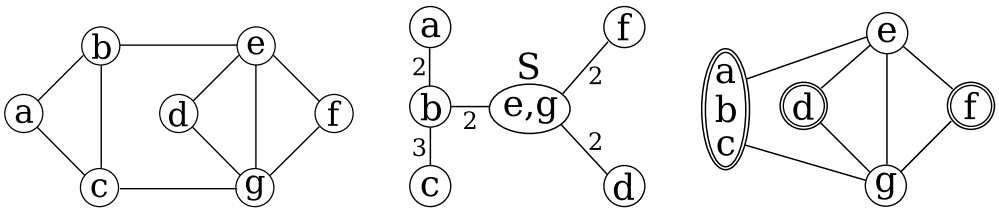
Während der gesamten Berechnung verwaltet der Algorithmus den Baum T von *Superknoten*, welche Teilmengen der Knotenmenge V von G repräsentieren. Die in diesem Baum verwalteten Teilmengen sind dabei ohne Überschneidung und ergeben in ihrer Gesamtheit zu jedem Zeitpunkt die Menge V. Initial besteht T aus einem einzelnen Knoten, welcher ganz V repräsentiert.

Solange der Baum T nun noch über Superknoten verfügt, deren Knotenmenge mindestens zwei Elemente enthält, wählt der Algorithmus einen solchen Knoten S aus und erstellt für diesen einen *Kontraktionsgraphen*. Dabei handelt es sich um eine vereinfachte Version des Graphen G, in welcher Knotenmengen, deren Trennung durch den folgenden Schnitt ausgeschlossen werden kann, zu einem *Kontraktionsknoten* zusammengefasst worden sind. Der Kontraktionsknoten beerbt die in ihm versammelten Originalknoten in allen Kantenbeziehungen. Entstehende Eigenschleifen werden aus dem Kontraktionsgraphen entfernt.

Um die zu kontrahierenden Knotenmengen zu bestimmen, wird der gewählte Superknoten S als Wurzel des Baums T definiert. Für jeden von S abgehenden Teilbaum werden nun die in dessen Superknoten verwalteten Originalknoten aus G zusammengefasst und kontrahiert. Die mit S assoziierte Knotenteilmenge verbleibt unkontrahiert im Graphen. Abbildung 3 zeigt den Vorgang beispielhaft.

Aus diesen unkontrahierten Knoten wird nun ein beliebiges Schnittpaar ausgewählt und der Minimale Schnitt zwischen beiden Knoten berechnet. Offensichtlich bipartitionieren die entstehenden Schnittmengen die Knotenmenge des Kontraktionsgraphen.

Der Superknoten S in T wird nun aufgespalten und durch die Knoten S_1 und S_2 ersetzt, auf welche die mit S assoziierte Originalknotenmenge entsprechend der Bipartition aufgeteilt wird. Zwischen S_1 und S_2 wird eine neue Kante mit dem festgestellten



(a) Graph G (b) Baum T (c) Kontraktionsgraph G'_S

Abbildung 3: Bildung eines Kontraktionsgraphen G'_S für den Knoten S des Baums T

Minimalen Schnittwert eingefügt. Zuletzt werden die ehemals von S abzweigenden Teilbäume von T gemäß der Zuordnung ihrer Kontraktionsknoten zu S_1 bzw. S_2 mit den neuen Repräsentanten verbunden.

Ist der Vorgang abgeschlossen, kann mit dem nächsten Superknoten mit nichttrivialer Knotenmenge fortgefahren werden. Auf diese Weise, verfeinert der Algorithmus den Baum T zusehends zu einem Gomory-Hu-Baum des Graphen G. Der Verfahren terminiert, sobald jeder Superknoten nur noch einen einzelnen Originalknoten aus V vertritt. Es bleibt nur noch, die Superknoten durch die vertretenen Knoten selbst zu ersetzen.

Motivation der Kontraktion Durch die Baumrepräsentation aller bisherigen Schnitte und die Bildung von Kontraktionsgraphen vor jedem neuen Schnitt, wird zu jeder Zeit sichergestellt, dass neu gefundene Schnittverläufe die bisherigen nicht kreuzen. Dies ist zulässig, da laut Lemma 3.1 weiterhin auch im unkontrahierten Graphen G gültige, Minimale Schnittverläufe zwischen den unkontrahierten Knoten im Kontraktionsgraphen existieren. Die kontrahierten Knotenmengen stellen jeweils die gegenüberliegende Schnittmenge eines vorherig getätigten Schnittes dar, in dem das aktuelle Schnittpaar stets in der gleichen Schnittmenge platziert wurde.

Neben der Einschränkung der auffindbaren Schnittverläufe, dient die Kontraktion ebenso der Beschleunigung der Minimalschnitt-Algorithmen, da die zu berücksichtigende Knoten- und Kantenzahl in den Kontraktionsgraphen gegenüber dem Originalgraphen stark abweichen kann (vgl. Abschnitt 8.4).

Komplexität Das Laufzeitverhalten des Algorithmus von Gomory und Hu ist entscheidend geprägt von der Komplexität des eingesetzten Algorithmus für gezielte Minimale s-t-Schnitte. Aufgrund der Gültigkeit des Max-Flow-Min-Cut-Theorems, werden hier üblicherweise leistungsfähige Maximalflussalgorithmen eingesetzt.

Für die im Rahmen dieser Arbeit erfolgte Implementierung (vgl. Abschnitt 7.2.2) wurde auf den Preflow-Push-Algorithmus von Goldberg und Tarjan zurückgegriffen. Er besitzt in der gewählten Variante worst-case Laufzeiten von $\mathcal{O}(n^3)$, für Graphen mit n Knoten. Die im Durchschnittsfall erreichten Werte liegen noch einmal sehr deutlich darunter (vgl. Abschnitt 8.4.1).

Die schnellsten dem Autor bekannten Maximalflussalgorithmen besitzen bereits eine worst-case Abschätzung von $\mathcal{O}(nm \log n)$ bzw. $\mathcal{O}(\min(n^{\frac{2}{3}}, m^{\frac{1}{2}})m \log(n^2/m) \log(U))$ ([GR98]) falls die Kantengewichte auf natürliche Zahlen im Intervall von $[0, U]$ beschränkt sind und m die Kantenzahl des Graphen darstellt.

In die Laufzeitabschätzung des Gomory-Hu-Algorithmus, gehe die Komplexität des Maximalflussalgorithmus mit dem Faktor $\mathfrak{M}(n, m)$ ein.

Durch die fortgesetzte Aufspaltung der Originalknotenmenge V ist die Zahl der benötigten Schnittberechnungen natürlich auf $(n-1)$ Ausführungen begrenzt, was zu einer Gesamtkomplexität von $\mathcal{O}(n\mathfrak{M}(m, n))$ führt.

Algorithmus 1: Der Algorithmus von Gomory und Hu

```
Data: Ungerichteter, gewichteter Graph G = (V, E)
Result: Gomory-Hu-Baum für G

// Initialisierung
S := new_supernode();
S.set := V;
T := ({S}, ∅);
Queue Q;
Q.enqueue(S);

// Kern-Schleife
while ¬Q.empty() do
    S := Q.dequeue();
    G' := G.contract_nodes_per_subtree_of(T,S);
    s := v ∈ S.set;
    t := v ∈ S.set \{s};
    λ := G'.calculate_min_cut(s, t);

    // Superknotenspaltung
    S_1 := new_supernode(); S_1.set := ∅;
    S_2 := new_supernode(); S_2.set := ∅;
    Nimm S_1 und S_2 in Knotenmenge von T auf;
    foreach v ∈ S.set do
        if v liegt in G' in einer Schnittmenge mit s then
            S_1.set.add(v);
        else
            S_2.set.add(v);

    // Baummanipulation
    Verbinde S_1 und S_2 mit λ-wertiger Kante;
    foreach X ∈ neighbors(S) do
        x := Kontraktionsknoten in G' welcher die Knoten aus X.set beinhaltet;
        if x liegt in G' in einer Schnittmenge mit s then
            Ersetze S durch S_1 in T-Kante zu X;
        else
            Ersetze S durch S_2 in T-Kante zu X;
    Lösche S aus T;

    // Queue-Pflege
    if |S_1.set| > 1 then Q.enqueue(S_1) ;
    if |S_2.set| > 1 then Q.enqueue(S_2) ;
return( T);
```

3.2. Der Algorithmus von Gusfield

Nach der Einführung des Gomory-Hu-Algorithmus 1961 vergingen 29 Jahre, bis von Daniel M. Gusfield ein neuer konkurenzfähiger Algorithmus für das ALL-PAIRS-MINIMUM-CUT-Problem vorgestellt wurde ([Gus90]). Dessen Entwicklung war motiviert von der ausgiebigen Nutzung von Knotenkontraktionen durch den Algorithmus von Gomory und Hu.

Dort besitzen sie eine zentrale Rolle in der Verhinderung sich kreuzender Schnitte und tragen ebenfalls zur Geschwindigkeit der eingesetzten Schnittalgorithmen bei. Die Graphkontraktion ist jedoch schwierig zu implementieren und verbraucht sowohl zusätzliche Speicher- als auch Rechenresourcen.

Als Alternative stellt Gusfield einen Algorithmus vor, welcher vollkommen ohne Kontraktionen auskommt und sämtliche Schnittberechnungen auf dem Originalgraphen durchführt. Er ist äußerst kompakt und einfach implementierbar.

Algorithmus 2: Der Algorithmus von Gusfield

Data: Ungerichteter, gewichteter Graph $G = (V, E)$
Result: Gomory-Hu-Baum für G

```
// Initialisierung
x := v ∈ V;
T := (V, E_T) mit E_T = {{x, y, −1}|y ∈ V, x ≠ y} ;          // Gewichtung −1
// Kern-Schleife
while ∃e = {s, t, −1} ∈ E_T do
    if degree(s) = 1 then
        leaf := s;
        center := t;
    else
        leaf := t;
        center := s;
    λ = G.calculate_min_cut(s, t);
    foreach v ∈neighbors( center) do
        if G.in_same_cut_set( v, leaf) then
            v.replace_neighbor( center, leaf) ;
    e := {s, t, λ} ;                                          // Gewichtung λ
return(T);
```

Verlauf Auch Gusfields Algorithmus verwaltet einen gewichteten Baumgraphen T, welcher im weiteren Verlauf die Form des Gomory-Hu-Baums für den Eingabegraphen $G = (V, E)$ annimmt. T ist hier von Beginn an über den Knoten der Menge V definiert. Vor dem Start des Algorithmus wird er sternförmig initialisiert, o.B.d.A sei also der Knoten mit der kleinsten ID im Zentrum und alle weiteren Knoten als Blätter

an diesem positioniert. Die Gewichtsmarken sämtlicher Kanten sind zunächst noch uninitialisiert.

Während des gesamten Verlaufs gilt die Invariante, dass jede noch nicht gewichtete Kante einen Blattknoten an einen Zentrumsknoten anbindet.

Solange noch ungewichtete Kanten in T existieren, wird nun wiederholt eine solche gewählt, und ihre inzidenten Knoten als nächstes Kandidatenpaar für eine Minimalschnittberechnung auf dem unkontrahierten Graphen G festgelegt. Dabei sei s der Nicht-Blattknoten und t der Blattknoten des Paares. Steht das Ergebnis fest, ist die Knotenmenge des Graphen durch die Schnittmengen wiederum bipartitioniert.

Sämtliche Knoten, welche in T zum Nicht-Blattknoten s adjazent sind, jedoch durch die Schnittberechnung in der gleichen Schnittmenge wie Knoten t positioniert wurden, verlieren ihre Verbindung zu s und werden stattdessen an t angehangen. Die modifizierten Kanten behalten dabei ein eventuell bereits vorhandenes Gewicht bei.

Als letzter Schritt kann die s-t-Kante in T mit dem errechneten Schnittwert versehen werden und die Iteration beginnt von Neuem.

Der entstehende Schnittbaum Durch den Verzicht auf Kontraktionen und damit die Einschränkung möglicher Schnittverläufe in G sind die aus den Schnittberechnungen resultierenden Verläufe nur in seltenen Fällen kreuzungsfrei.

Die gewählte Umformung des Baumes T nach der Schnittoperation führt jedoch dazu, dass dieser stets gültige, kreuzungsfreie Schnittverläufe enthält:

Lemma 3.2 ***[CH90] Fact 2***
Es sind die Knoten i und j mit einer ungewichteten Kante in T verbunden, j ein Blatt und es existiert eine bereits gewichtete Kante zwischen i und k. Weiter seien die Knoten des auf der k-Seite durch Entfernung der i-k Kante entstehenden Teilbaums mit K zusammengefasst. Dann existiert ein Minimaler i-j-Schnitt $(X,\overline{X})$ in G, bei welchem K vollständig in entweder X oder $\overline{X}$ enthalten ist.

Beweis: [CH90] Jede bereits gewichtete Kante in T repräsentiert einen Minimalen Schnitt in G. Seien $(Y,\overline{Y})$ die Schnittmengen des Minimalen i-k-Schnittes, mit $i, j \in Y$ und $k \in \overline{Y}$. Da bekannt ist (siehe Lemma 3.1), dass eine Version von $(X,\overline{X})$ existiert, welche $(Y,\overline{Y})$ nicht kreuzt, muss K in diesem Schnitt vollständig in X oder $\overline{X}$ enthalten sein.

□

Die Verschiebung ganzer Teilbäume aufgrund der Schnittmengenzuteilung ihrer Wurzelknoten, ist also korrekt und führt zu den gewünschten kreuzungsfreien Schnittverläufen. Damit wird T zum Gomory-Hu-Baum.

Zu bemerken ist, dass die darin angegebenen Schnittverläufe selten den Verläufen der eigentlich durchgeführten Schnittberechnungen entsprechen.

Komplexität Die Laufzeitkomplexität des Gusfield-Algorithmus entspricht durch die $(n-1)$-fache Ausführung eines Algorithmus für gezielte Minimale s-t-Schnitte, der des Gomory-Hu-Algorithmus und liegt bei Nutzung eines Maximalflussalgorithmus mit Komplexität $\mathfrak{M}(n,m)$ bei $\mathcal{O}(n\mathfrak{M}(n,m))$.

3.3. Vergleich und Einschätzung

Nach der Vorstellung des Gusfield-Algorithmus, ist dieser insbesondere in [GGP+99] ausführlich mit dem klassischen Gomory-Hu-Algorithmus verglichen worden. Besonderer Schwerpunkt lag dabei auf den real erzielbaren Laufzeiten einer Implementierung und der Möglichkeit den Algorithmus um weitere Heuristiken zu erweitern.

Prinzipiell verspricht Gusfields Version Vorteile durch seine bestechende Einfachheit und den Wegfall der sonst für Kontraktionen anfallenden Arbeit. Der klassische Algorithmus punktet dagegen mit stark verkleinerten Probleminstanzen für die Minimalschnittalgorithmen, einer freieren Auswahlmöglichkeit der Schnittpartner und damit mehr Flexibilität und Ansatzpunkte für weitere Heuristiken.

In den resultierenden Laufzeiten zeigt sich, dass die Vorteile der Graphkontraktion ihre Nachteile aufwiegen und der klassische Algorithmus im Durchschnittsfall stabiler und schneller arbeitet. Zusätzlich lässt sich der Algorithmus über verschiedenere Heuristiken beschleunigen.

Aus diesen Gründen wurde für die in Abschnitt 8 vorgestellten Experimente der Gomory-Hu-Algorithmus als Referenzalgorithmus verwendet.

4. Einsatz Maximaler Adjazenzordnungen

4.1. Definition, Eigenschaften und Berechnung

Die Technik der Erstellung einer Maximalen Adjazenzordnung basiert auf Grundideen von Nagamochi und Ibaraki ([NI92a], [NI92b]) und ist in der hier vorgestellten Form zuerst in [SW97] von Stoer und Wagner eingeführt und genauer betrachtet worden. In der Folgezeit wurde sie von vielen Algorithmen bei der Analyse des Zusammenhangs von Graphen aufgegriffen. Später entstanden als Abwandlung die Laxen Adjazenzordnungen, welche erfolgreich im Minimalschnitt-Algorithmus nach [Bri05] eingesetzt werden.

Um eine Maximale Adjazenzordnung zu definieren, sei zunächst auf Basis der Kantengewichtungsfunktion w des Graphen G die *Adjazenz* $d(X, v)$ eingeführt:

Definition 4.1 ***Adjazenz eines Knotens zu einer Knotenmenge***
Ist $G = (V, E)$ ein ungerichteter Graph mit Kantengewichtsfunktion w, so ist die *Adjazenz eines Knotens v zu einer Knotenmenge $X \subseteq V$* definiert als:

$$d(X, v) = \sum_{x \in X, \{v,x\} \in E} w(\{v, x\})$$

Damit ergibt sich bereits die Maximale Adjazenzordnung:

Definition 4.2 ***Maximale Adjazenzordnung***
Eine *Maximale Adjazenzordnung* über einem Graphen $G = (V, E)$ mit Knotenmenge $V = \{v_1, \cdots, v_n\}$ ist eine Knotenordnung $V_n = (v_1, \cdots, v_n)$ mit beliebigem *Startknoten* $v_1 \in V$, in der für jeden Knoten v_i mit $i > 1$ und die Teilordnungen $V_j = (v_1, \cdots, v_j)$ mit $j < n$ gilt:

$$v_i = \underset{x \in V \setminus V_{i-1}}{\arg\max}\, d(V_{i-1}, x)$$

Über den Elementen einer solchen Ordnung können nun interessante Eigenschaften nachgewiesen werden. Die im Folgenden mit Abstand wichtigste ist dabei der garantierte Minimale Blattschnitt zwischen v_n und v_{n-1}.

Lemma 4.1 ***Stoer und Wagner, 1997***
Ist $V_n = (v_1, \cdots, v_{n-1}, v_n)$ eine Maximale Adjazenzordnung für den Graphen G, so existiert in G ein Minimaler v_{n-1}-v_n-Schnitt mit den Schnittmengen $\{v_n\}$ und $V \setminus \{v_n\}$.

Offensichtlich liegt der brisante Punkt während der Berechnung einer Adjazenzordnung in der Bestimmung des als nächsten auszuwählenden, höchstadjazenten Knotens. Effizient lässt sich dies mit einer Max-Heap-Datenstruktur implementieren. Hierzu werden während der Algorithmusinitialisierung sämtliche Knoten außer des Startknotens

mit einem Adjazenzwert von Null in den Heap eingefügt und folgend für jede zum Startknoten inzidente Kante, der Heap-Schlüssel des Kantennachbarn um das Kantengewicht erhöht. Zuletzt wird der Startknoten als besucht markiert.

Solange nun noch nicht alle Knoten des Graphen besucht worden sind, ist der nächste höchstadjazente Knoten durch das maximale Heap-Element bestimmt. Der Knoten wird entnommen, als besucht markiert und die Heap-Schlüssel all seiner unbesuchten Nachbarn wiederum um den sie anbindenden Kantenwert erhöht.

Dieses Vorgehen führt auf dem Heap zu $(n-1)$ `insert` sowie `extractMax` Operationen und m `increaseKey` Aufrufen. Wird für die Heap-Implementierung ein Fibonacci-Heap verwendet, so resultiert eine Gesamtlaufzeit von $\mathcal{O}(n \log(n) + m)$.

Algorithmus 3: Bildung einer Maximalen Adjazenzordnung (abstrakt)

Data: Ungerichteter, gewichteter Graph $G = (V, E)$, Startknoten x
Result: Maximale Adjazenzordnung order

$X := \emptyset$;
while $X \neq V$ **do**
  $v := \arg\max_{x \in V \setminus X} d(X, x)$;
  $X := X \cup \{v\}$;
  order $[|X|] := v$;
`return(order);`

Diese Laufzeitabschätzung liegt deutlich unter den für die besten s-t-Maximalflussalgorithmen im worst-case etablierten $\mathcal{O}(nm \log(n))$.

Sollen Maximale Adjazenzordnungen für die Bestimmung möglichst vieler Minimaler s-t-Schnitte in einem Graphen verwendet werden, so erweist sich als schwierig, dass auf die Schnittpartner des je Ordnungserstellung festgestellten Minimalen Schnitts äußerst wenig Einfluss genommen werden kann. Das einzige verfügbare Steuerungsinstrument liegt im verwendeten Startknoten. Im schlechtesten Fall, kann jedoch auch das Durchprobieren sämtlicher Knoten des Graphen als Startknoten zu genau zwei festgestellten Minimalschnitten führen.

Die Ordnung zeigt sich dabei sensibel für einzelne Knoten welche in ihrer Adjazenz zum Restgraphen stark nach unten abweichen. Zusätzlich spielt die Gradverteilung und damit das Layout des Graphen eine große Rolle, was sich auch in den in Abschnitt 8 gezeigten Experimenten niederschlägt.

4.2. Ableitung eines maximalen Flusses

Das direkte Ergebnis der Bildung einer Maximalen Adjazenzordnung ist - neben der Ordnung selbst - zunächst nur der Minimale Schnitt zwischen den beiden letzten Knoten v_{n-1} und v_n. Wie Arikati und Mehlhorn in [AM99] zeigen, ist es mit einem Zusatzaufwand von $\mathcal{O}(m)$, jedoch ebenso leicht möglich, einen entsprechenden Maximalen v_{n-1}-v_n-Fluss in G abzuleiten.

Der Algorithmus profitiert, unter anderem, davon, dass mit dem Blattschitt von v_n bereits eine vollständige Schnittmenge und so auch der Minimale Schnittwert bzw.

Maximale Flusswert λ bekannt ist. Während der Berechnung *schiebt* der Algorithmus negativen Fluss im Wert von λ aus v_n und positiven Fluss aus v_{n-1} schrittweise in Richtung niedriger eingeordneter Knoten. Dabei treffen die entstehenden Flusspfade zusehends aufeinander und neutralisieren sich schließlich. Das Ergebnis ist ein korrekter Maximaler v_{n-1}-v_n-Fluss in G.

Algorithmus 4: Maximaler Fluss aus Maximaler Adjazenzordnung

Data: Graph $G = (V, E)$, Maximale Adjazenzordnung order
Result: Ein Maximaler v_{n-1}-v_n-Fluss in G

$\lambda := c_G(\{v_n\})$;
Schiebe Flusswert $-\lambda$ heraus aus v_n;
open $:= \lambda - w(\{v_{n-1}, v_n\})$; /* $\{v_{n-1}, v_n\} \notin E \Rightarrow w(\{v_{n-1}, v_n\}) = 0$ */
Schiebe Flusswert open aus v_{n-1} zu Knoten $\{v_1, \cdots, v_{n-2}\}$,
beginnend mit hoch eingeordneten Knoten;

$i := n - 2$;
while open $\neq 0$ **do**
  v_i :=order $[i]$;
  δ :=inflow(v_i) - outflow(v_i) ; /* Kann negativ werden. */
  Schiebe δ-Fluss aus v_i zu Knoten $\{v_1, \cdots, v_{i-1}\}$,
  beginnend mit Knoten hoher Ordnung;
  open := open $-$ min(inflow(v_i),outflow(v_i));
  $i := i - 1$;
return()

Während der wiederholten Flussverschiebungen garantiert die Maximale Adjazenzordnung, dass das summierte Gewicht der Kanten eines Knotens v_i zu den Knoten $\{v_1, \cdots, v_{i-1}\}$ grundsätzlich ausreicht um den Überschuss des von den Knoten $\{v_{i+1}, \cdots, v_n\}$ erhaltenen Flusses weiterzuleiten.

4.3. Einsatz im Algorithmus

Der Einsatz Maximaler Adjazenzordnungen, wurde, neben weiteren Heuristiken, bereits in [THS99] für die Berechnung von Gomory-Hu-Bäumen vorgeschlagen. Dort wurde jedoch nur ihre generelle Eignung als Ersatz zur Anwendung eines Maximalflussalgorithmus untersucht und aufgrund der fehlenden Steuerbarkeit des Ordnungsergebnisses schließlich verworfen.

Im Gegensatz dazu, sollen Maximale Adjazenzordnung hier den Maximalflussalgorithmus nicht komplett ersetzen, sondern nur in erfolgsversprechenden Fällen als preiswertes Orakel zur Anwendung kommen. Liefert dies kein für den Algorithmus neues Ergebnis, wird dennoch auf den Flussalgorithmus zurückgegriffen. Auf die in der Implementierung verwendete Dosierung ihres Einsatzes, geht Abschnitt 7.2.3 gesondert ein.

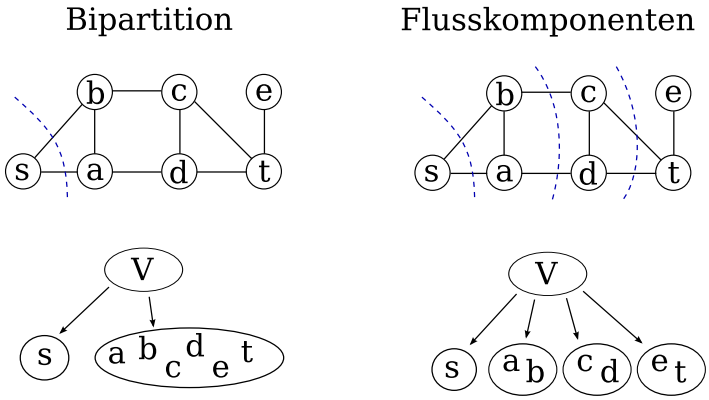


Abbildung 4: Mehrfachsplit durch Zerlegung eines Maximalen s-t-Flusses

5. Die Flusskomponentenzerlegung

5.1. Idee

Als weiterer Ansatz zur Beschleunigung der Berechnung von Gomory-Hu-Bäumen, soll die *Flusskomponentenzerlegung* eingeführt werden. Sie basiert auf der einfachen Feststellung, dass die im Zuge der Berechnung gebildeten Maximalen Flüsse, nicht nur *einen* Minimalen Schnitt repräsentieren, sondern dass durch einen Maximalen s-t-Fluss sämtliche Minimalen s-t-Schnitte in G aufzählbar sind ([PQ80]).

Wird aus diesen eine maximale Anzahl von $k \geq 1$ zueinander kreuzungsfreiher Minimaler s-t-Schnittverläufe ausgewählt, so zerlegen diese die Knotenmenge V in $(k+1)$ Differenzmengen, anstatt der bisher konstanten Bipartition. Wie in Abschnitt 6.1 ausführlich gezeigt wird, lassen sich diese Differenzmengen ähnlich den klassischen Schnittmengen von Gomory und Hu verwenden und garantieren die Existenz global gültiger Minimalschnitte in auf ihrer Basis kontrahierten Versionen des Originalgraphen G.

Der Einsatz dieser Technik verspricht ein zügigeres Absinken der Knotenzahl eines durchschnittlichen Kontraktionsgraphen und damit eine Beschleunigung aller folgenden Berechnungen auf diesem. Die Flusskomponentenzerlegung bedingt eine zusätzliche Buchhaltung über die Ergebnisse der durchgeführten Flussoperationen. Mit ihrer Hilfe ist jedoch auch die gänzliche Einsparung von Flussberechnungen möglich. So kann im Einzelfall die ürsprünglich feste Zahl von $(n-1)$ Maximalflussberechnungen beinahe halbiert werden.

Die vorgestellte Technik hat weiterhin Potential Graphkontraktionen einzusparen und lässt sich positiv mit dem in Abschnitt 4 vorgeschlagenen Einsatz Maximaler Adjazenzordnungen kombinieren.

5.2. Relaxierte Gomory-Hu-Bäume

Um die Idee der Flusskomponentenzerlegung in einem Algorithmus umzusetzen, muss zunächst das Konzept des Gomory-Hu-Baums verallgemeinert werden:

Definition 5.1 ***Relaxierter Gomory-Hu-Baum***
Der Baum $T = (V \cup Z, A)$ mit der Kantengewichtsfunktion $w_T : A \to \mathbf{R}^+$ ist genau dann ein *Relaxierter Gomory-Hu-Baum* für den gewichteten, ungerichteten Graphen $G = (V, E)$, wenn er folgende Bedingungen erfüllt:

1. $\forall a \in A$: Sind T_1 und T_2 die Knotenmengen, der durch Löschung von a aus T entstehenden Teilbäume, so gilt $w_T(a) := c_G(T_1 \cap V)$.
2. Die minimal gewichtete Kante auf jedem Pfad zwischen zwei Knoten $s, t \in V$ in T, ist mit dem Minimalen s-t-Schnittwert in G gewertet.

Ähnlich einem Gomory-Hu-Baum, herrscht also auch im Relaxierten Gomory-Hu-Baum Flussäquivalenz zu G und für jedes Knotenpaar $s, t \in V$ sind sowohl Wert als auch Verlauf eines Minimalen s-t-Schnittes in G über die niedrigwertigste Kante des s-t-Pfades im T ermittelbar.

Ein Relaxierter Gomory-Hu-Baum kann, wie in Abschnitt 5.5 gezeigt wird, mit einem maximalen Zeitaufwand von $\mathcal{O}(n^2)$ in einen Gomory-Hu-Baum überführt werden.

5.3. Algorithmus

5.3.1. Überblick

Wie das klassische Gomory-Hu-Schema folgt der vorgeschlagene Algorithmus dem *Divide & Conquer*-Funktionsprinzip. Iterativ wird die Knotenmenge des Originalgraphen V in immer feinere Teilmengen zerlegt, zwischen deren Elementen in G gültige Maximale Flüsse auf einem kontrahierten Graphen berechnet werden können, in welchem die restlichen Teilmengen in Kontrationsknoten zusammengefasst sind.

Zu diesem Zweck verwaltet auch der vorgeschlagene Algorithmus die einzelnen Teilmengen von V in einem Baum T, welcher mit zunehmendem Fortschreiten der Berechnungen die Gestalt eines Relaxierten Gomory-Hu-Baums annimmt (abgesehen von der Tatsache, dass die Knoten aus V nicht direkt in diesem vorkommen, sondern am Ende für jedes $v \in V$ genau ein Baumknoten existiert, mit welchem die Knotenmenge $\{v\}$ assoziiert ist).

Für die Aufteilung der Knotenmenge V wird hierbei im Anschluss an die Identifikation eines Maximalen s-t-Flusses nicht länger eine reine Bipartition auf Basis eines Minimalen s-t-Schnittes vorgenommen. Stattdessen folgt eine Analyse der durch den Fluss aufzählbaren kreuzungsfreien Minimalschnittverläufe. Auf ihrer Basis wird die Knotenmenge anschließend in Flusskomponenten zerlegt.

Die durch die Kontraktion erzwungene Kreuzungsfreiheit der gefundenen Schnittverläufe zu sämtlichen vorher getätigten Aufspaltungen (hier aller weiteren im Baum

T verwalteten Teilmengen von V) gewährleistet fortlaufend, dass die durch einen Maximalen Fluss erzielten Ergebnisse stets durch *rein lokale Änderungen* in den sich entwickelnden Baum integriert werden können.

Eine deutliche Abweichung vom Gomory-Hu-Algorithmus muss vorgenommen werden, sobald die Kardinalität aller verwalteten Knotenteilmengen auf Eins gesunken ist. Im Gegensatz zum klassischen Algorithmus, kann dies hier bereits eintreten, bevor alle zur korrekten Erfassung der Schnittverhältnisse in G notwendigen Maximalflussoperationen ausgeführt worden sind (siehe Abschnitt 6.3).

Um dieses Problem zu behandeln, erhält der Algorithmus mit dem *Blattschnittbeziehungsgraphen* L eine weitere Buchführungskomponente, in der die bisher bereits durchgeführten Schnittoperationen festgehalten und noch notwendige Operationen identifizierbar sind.

Der resultierende Algorithmus lässt sich somit in zwei Phasen gliedern:

Phase 1 verläuft ähnlich dem Gomory-Hu-Algorithmus, bildet für die Knoten des verwalteten Baums T entsprechende Kontraktionsgraphen, berechnet dort mittels *Flusskomponentenzerlegungen* Partitionen der Originalknotenmenge V von G und spaltet den betrachteten Superknoten des Baumes T zu einer Kette von Superknoten auf. In dieser Phase wird der Blattschnittbeziehungsgraph L ausschließlich gepflegt, muss jedoch noch nicht zur Identifikation geeigneter Schnittpartner eingesetzt werden, da sich diese aus den unkontrahierten Knoten jedes Kontraktionsgraphen direkt ergeben.

Phase 2 beginnt sobald alle in T verwalteten Knotenmengen eine Kardinalität von Eins erreicht haben. Nun werden mit Hilfe der im Graphen L gespeicherten Schnittinformationen, die zur Bildung eines vollständigen Relaxierten Gomory-Hu-Baums noch ausstehenden Schnittberechnungen bestimmt und auf stark kontrahierten Versionen des Graphen G durchgeführt.

5.3.2. Datenstrukturen

Die im Algorithmus verwendeten Datenstrukturen lassen sich wie folgt zusammenfassen.

Der Graph G Durch den Aufrufer gegeben, ist der mit nicht-negativen Kantengewichten versehene, ungerichtete Graph $G = (V, E)$, dessen Relaxierter Gomory-Hu-Baum zu berechnen ist.

Der Baum T Als Prototypen hierfür, verwaltet der Algorithmus den gewichteten, ungerichteten Baum T, dessen Knoten sich in *Superknoten* und *Leere Knoten* kategorisieren lassen. Erstere korrespondieren zu disjunkten, nicht-leeren Teilmengen der Knotenmenge V, deren Vereinigung jederzeit wieder V ergibt. Mit Leeren Knoten sind dagegen nur leere Mengen assoziiert. Durch fortgesetzte Teilung von Superknoten und lokale Verfeinerung, wird aus diesem Baum im weiteren Verlauf der Relaxierte Gomory-Hu-Baum zu G.

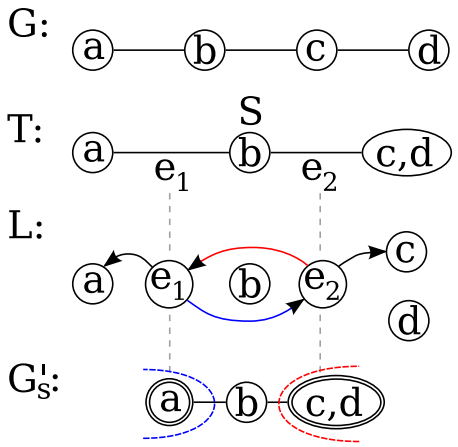


Abbildung 5: Aufbau und Informationsgehalt des *Blattschnittbeziehungsgraphen* L. Bekannte minimale $[a]$-$[c,d]$-Schnitte in G'_S der Form $\{[a]\}$ und $\{[c,d]\}$.

Auftragsqueue Q Des Weiteren existiert eine Auftragsqueue Q, deren Elemente auf Knoten des Baumes T verweisen. Sie dient der Identifikation des als nächstes zu bearbeitenden *Kontraktionsgraphen* G'.

Die Kontraktionsgraphen G' Ein solcher entsteht, indem ein vorgegebener Knoten als Wurzel des Baums T definiert wird. Die Knotenmengen aller in davon abgehenden Teilbäumen enthaltenen Superknoten, werden je Teilbaum verschmolzen, und die in ihnen enthaltenen Originalknoten aus G jeweils zu einem *Kontraktionsknoten* zusammengefasst.

Dieser erbt die Position der in ihm kontrahierten Knoten in sämtlichen Kanten, welche die durch ihn repräsentierte Knotenmenge an einem Endpunkt berühren. Entstehende Eigenschleifen an Kontraktionsknoten werden aus dem Graphen entfernt.

Der Blattschnittbeziehungsgraph L Der Blattschnittbeziehungsgraph bewahrt Informationen über bereits getätigte Maximalflussberechnungen und die folgenden Zerlegungen der Knotenmenge. Damit enthält er zusätzlich Wissen über sämtliche in einem Kontraktionsgraphen bereits bekannten Schnittbeziehungen. Mit seiner Hilfe können jederzeit die noch nicht im Baum T festgehaltenen Schnitte identifiziert werden. Zusätzlich ermöglicht er in bestimmten Situationen (siehe Abschnitt 6.4) die Einsparung von Schnittoperationen.

Die Knotenmenge des gerichteten Graphen $L = (V \cup A, C)$, setzt sich aus den Knoten in V und *Kantenknoten* A zusammen. Dabei existiert für jede Kante des Baumes $T = (V \cup Z, A)$ ein eineindeutiger Kantenknoten in L.

Da L über Originalknoten aus G und Kantenknoten definiert ist, die Kontraktiongra-

phen jedoch aus Originalknoten und Kontraktionsknoten bestehen, ist eine Abbildung zwischen beiden notwendig:

Wird der Kontraktionsgraph G'_S für einen T-Knoten S erzeugt, so leitet jede der zu S in T inzidenten Kanten einen der kontrahierten Teilbäume ein. Deren Kantenknoten in L lassen sich eineindeutig den aus den jeweiligen Teilbäumen entstehenden Kontraktionsknoten in G'_S zuordnen.

Für die Auswertung der bekannten Schnittbeziehungen in G'_S ist also ausschließlich der Teilgraph von L interessant, welcher von der mit S assoziierten Teilmenge von V und den Kantenknoten der zu S inzidenten Kanten induziert wird. Nur für diese Knoten sind Vetreter im Kontraktionsgraphen zu S vorhanden.

Informationsgehalt von L-Kanten Für jede Kante dieses L-Teilgraphen existiert in G'_S ein bekannter und bereits in T repräsentierter Minimaler Schnitt. Die L-Kanten beginnen dabei grundsätzlich in Kantenknoten und können sowohl auf Kantenknoten als auch auf Knoten aus V verweisen.

Seien e_1 und e_2 zwei zu S inzidente T-Kanten, welche durch die Kontraktionsknoten $[e_1]$ und $[e_2]$ in G'_S vertreten sind und der Knoten $b \in V$ Element der mit S assoziierten Knotenmenge (siehe Situation in Abbildung 5). Existiert nun in L eine Kante $e_1 \to b$, so stellt $\{[e_1]\}$ in G'_S einen bekannten Minimalen Schnitt zwischen $[e_1]$ und b dar. Entsprechend folgert aus der Existenz einer L-Kante $e_1 \to e_2$ ein Minimaler Schnitt der Form $\{[e_1]\}$ zwischen $[e_1]$ und $[e_2]$ in G'_S.

Prinzipiell ist sichergestellt, dass für jeden Kontraktionsknoten eines Kontraktionsgraphen G' mindestens ein Minimaler Schnitt (in Form seines Blattschnitts) zu einem weiteren Knoten in G' bekannt ist (Details siehe Lemma 6.7).

Während des gesamten Algorithmusverlaufs werden alle genutzten Minimalen Schnitte in L festgehalten und diese Repräsentation immer wieder aktualisiert, so dass die bekannten Schnitte jeweils auf die aktuell für die Knoten des sich verändernden Baums T bildbaren Kontraktionsgraphen übertragbar bleiben.

5.3.3. Initialisierung

Zu Beginn des Algorithmus besteht T aus einem einzelnen Blatt, dem Superknoten der Knotenmenge V. Um dessen Bearbeitung anzustoßen, enthält Q als einziges Element einen Verweis auf diesen Superknoten. Der Graph L enthält zu Beginn exakt die Knoten $v \in V$ und ist kantenlos.

5.3.4. Phase 1: Superknotenspaltung

Diese Phase dauert an, solange die Auftragsqueue gefüllt ist. Dies trifft zu, falls noch mindestens ein Superknoten in T existiert, dessen assoziierte Knotenmenge eine Kardinalität größer Eins besitzt.

Erstellung des Kontraktionsgraphen Am Beginn jeder neuen Iteration, wird der Queue ein Superknoten S entnommen und, wie bereits in Abschnitt 5.3.2 beschrieben, der entsprechende Kontraktionsgraph G'_S für ihn gebildet.

Algorithmus 5: Algorithmus mit Flusskomponentenzerlegung

Data: Ungerichteter, gewichteter Graph $G = (V, E)$
Result: Gomory-Hu-Baum für G

```
// Initialisierung
S := new_supernode(); S.set := V;
T := ({S}, ∅);
L := (V, ∅);
Queue Q;
if |V| > 1 then Q.enqueue(S);

// Phase 1
while ¬Q.empty() do
    S := Q.dequeue();
    G' := G.contract_nodes_per_subtree_of(T,S);
    Berechne gültigen s-t-Maximalfluss auf G';          // PushRelabel oder MAO
    Berechne Flusskomponenten {C_1, ··· , C_k} aus Residualgraph von G';
    // Pflege T
    Spalte S zu {S_1, ··· , S_k} gemäß {C_1, ··· , C_k};
    Verkette {S_1, ··· , S_k} mit Flusswert tragenden Kanten {e_1, ··· , e_{k-1}};
    Adaptierte ehemals zu S inzidente T-Kanten gemäß {C_1, ··· , C_k};
    // Pflege L
    Aufnahme von {e_1, ··· , e_{k-1}} in Knotenmenge von L;
    Erzeuge neue L-Kanten für berechnete Flusszerlegung;
    Adaptiere L-Kanten mit in T nicht länger benachbarten Endpunkten;
    foreach S_i ∈ {S_1, ··· , S_k} ∧ (|S_i.set| > 1) do Q.enqueue(S_i);

// Phase 2
foreach X ∈ Nodes_of(T) ∧ (degree(X) > 1) do Q.enqueue(X);
while ¬Q.empty() do
    S := Q.dequeue();
    if S-induzierter Teilgraph von L hat keinen Spannbaum then
        G' := G.contract_nodes_per_subtree_of(T, S);
        Identifiziere Schnittpartner s,t aus Wurzelknoten des L-Spannwalds;
        Berechne Minimalen s-t-Schnitt in G';
        // Pflege T (s. Abbildung 9)
        Klassifiziere Schnittverlauf und adaptiere T entsprechend;
        // Pflege L (gemäß Klassifikation)
        if Neue T-Kante e_n erzeugt then
            Aufnahme von Kantenknoten e_n in L;
            Bildung neuer L-Kanten e_n → s und e_n → t;
            Richte L-Kanten mit in T getrennten Endpunkten auf e_n;
        else Bildung einer L-Kante für gefundenen Blattschnitt;
        Füge S und evtl. entstandenen Leeren T-Knoten in Q ein;

return(RGHTtoGHT(T));
```

Maximalflussberechnung In G'_S wird im nächsten Schritt ein Maximaler s-t-Fluss berechnet. Dies kann entweder durch einen expliziten s-t-Maximalfluss-Algorithmus geschehen, in welchem Fall sowohl Quelle als auch Senke aus der Menge der nicht kontrahierten Knoten gewählt werden. Alternativ ist die Berechnung über die Bildung einer Maximalen Adjazenzordnung und eine anschließende Flussverfolgung (siehe Abschnitt 4) möglich.

Die zweitgenannte Methode besitzt eine bessere worst-case Laufzeitabschätzung, gewährt jedoch kaum Einfluss auf die Endpunkte des Flusses, so dass sie unter Umständen bereits aus vorherigen Berechnungen bekannte Maximale Flüsse in G'_S ein weiteres Mal identifiziert. In einem solchen Fall schlägt die später folgende Gültigkeitsprüfung fehl und die Maximalflussberechnung muss erneut ausgeführt werden. Es ist zu erwarten, dass ihre Anwendung in großen Graphen und besonders zu Beginn des Algorithmus dennoch zu Laufzeitvorteilen führen kann. Diese Hypothese ist in Experimenten später zu überprüfen.

Identifikation von Flusskomponenten Mit Hilfe des Residualgraphen des so gefundenen Maximalen s-t-Flusses, können nun mit einem Verfahren nach [PQ80] sämtliche Minimalen s-t-Schnittverläufe aufgezählt werden. Hierfür werden zunächst die Starken Zusammenhangskomponenten des Residualgraphen bestimmt und deren Knoten je Komponente kontrahiert. Die Kantenrichtungen im resultierenden kreisfreien Graphen definieren nun eine Halbordnung R, deren transitive Hüllen die s-t-Schnittmengen in G'_S bilden.

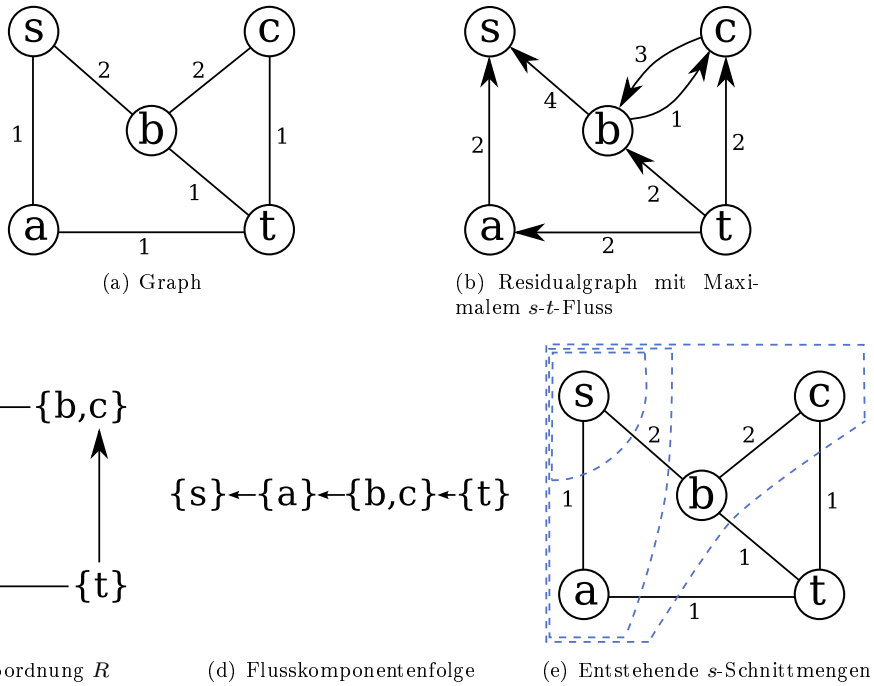
Die kleinste s enthaltende Schnittmenge lässt sich damit leicht als die Vereinigung der s enthaltenden Zusammenhangskomponente und all ihrer transitiven Nachfolger in R bestimmen. Gleiches gilt für die kleinste t enthaltende Schnittmenge, wobei die Vorgänger der Komponente von t hinzugezählt werden.

Anstatt nun sämtliche (unter Umständen exponential viele) Minimale s-t-Schnitte in G'_S aufzuzählen, besteht die Zielstellung jedoch in der Identifikation einer maximalen Zahl zueinander kreuzungsfreier s-t-Schnittverläufe.

Sind die zur kleinsten s- und t-Schnittmenge gehörenden Komponenten jeweils zusammengezogen worden, ist dies durch die Festlegung einer Totalordnung über den verbliebenen Zusammenhangskomponenten möglich. Eine topologische Sortierung auf Basis der bestehenden Halbordnung R, wobei für die t-Komponente die oberste Position erzwungen wird, resultiert in einer geordneten Folge von Komponenten, in der jedes Element zusammen mit seinen Vorgängern unter Vereinigung eine s enthaltende Schnittmenge (s-Schnittmenge) eines Minimalen s-t-Schnitts in G'_S bildet.

Die für jedes Element entstehende s-Schnittmenge umschließt dabei diejenige des Vorgängerelements vollkommen. Die jeweiligen Schnittverläufe trennen grundsätzlich entweder nur Knoten der s-Schnittmenge oder der t-Schnittmenge aller anderen Verläufe der Folge und sind somit *kreuzungsfrei*.

Dies ist eine **Folge von Flusskomponenten** des Maximalen s-t-Flusses. Ein beispielhafter Ablauf für den beschriebenen Prozess ist in Abbildung 6 zu sehen.



(a) Graph

(b) Residualgraph mit Maximalem s-t-Fluss

(c) Halbordnung R

(d) Flusskomponentenfolge

(e) Entstehende s-Schnittmengen

Abbildung 6: Festlegung einer Flusskomponentenfolge am Beispiel

Gültigkeit Der Graph in dem diese Flusskomponenten aufgefunden worden sind, besteht aus Originalknoten $v \in V$ und Kontraktionsknoten. Da die Zielstellung in einer feineren Zerlegung der mit dem Superknoten S assoziierten Knotenmengen aus V besteht, sind nur solche Flusskomponenten interessant, welche mindestens einen unkontrahierten Knoten aus V enthalten. Komponenten, welche ausschließlich Kontraktionsknoten beinhalten, werden mit einer Nachbarkomponente der Folge verschmolzen.

Verbleiben nach diesem Vorgehen noch mindestens zwei Flusskomponenten, so sind s-t-Fluss und Flusskomponentenzerlegung *gültig*. Mindestens ein Minimaler s-t-Schnitt trennt zwei Originalknoten des betrachteten Kontraktionsgraphen G'_S. In T kann der Superknoten S, welcher zur Bildung des Kontraktionsgraphen geführt hat, weiter aufgespalten werden.

Anderenfalls muss ein Minimaler Schnitt für eine alternative Knotenpaarungen gefunden werden. Dieser Fall kann jedoch ausschließlich dann eintreten, falls Kontraktionsknoten als Flussendpunkte dienen, was nur beim Einsatz von Maximalen Adjazenzordnungen zur Flussbestimmung unbeabsichtigt geschehen kann.

Superknotenzerlegung in T In T teilt sich der Superknoten S entsprechend der von den Flusskomponenten $C_1, ..., C_k$ vorgegebenen Partition der mit S assoziierten Teil-

menge von V auf. Die neu entstehenden Superknoten $S_1, ..., S_k$ werden in T gemäß der Ordnung der Flusskomponentenfolge jeweils mit dem Superknoten ihrer Nachfolgerkomponente verkettet. Die eingefügten Kanten $\{e_1, \cdots, e_{k-1}\}$ tragen den Flusswert $\lambda(s,t)$.

Die ehemals von S abgehenden Teilbäume von T werden, gemäß der Aufteilung der mit ihnen assoziierten Kontraktionsknoten zu Flusskomponenten in G'_S, einem der neuen Superknoten angefügt.

Adaption von L Im Graphen L werden zunächst für die neu in T eingefügten Kanten entsprechende Kantenknoten in die Knotenmenge aufgenommen.

Schnittinformationen für neue Kantenknoten: Im nächsten Schritt werden für jedes Paar e_{i-1}, e_i frisch eingefügter, zum selben neuen Superknoten S_i inzidenter T-Kanten zwei gegenläufig gerichtete Kanten zwischen deren Kantenknoten in L eingefügt. Zusätzlich verweist eine neue L-Kante vom Kantenknoten der zu S_1 inzidenten T-Kante e_1 auf den Quellknoten s und eine weitere neue L-Kante vom Kantenknoten der zu S_k inzidenten T-Kante e_{k-1} auf den Senkenknoten t.

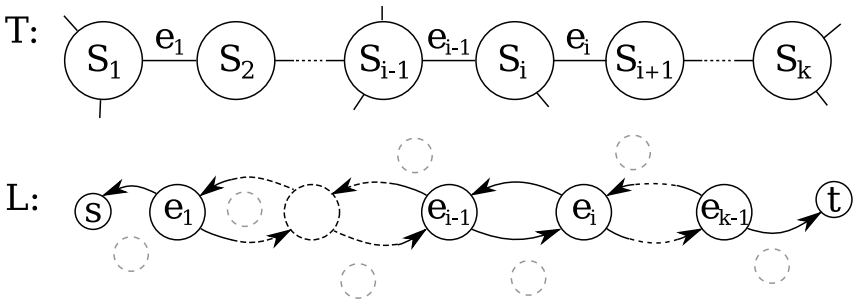


Abbildung 7: Verbindung neu eingefügter Kantenknoten in L

Dieses Vorgehen hält Informationen über die gemeinsame Entstehung der neuen T-Kanten $\{e_1, \cdots, e_{k-1}\}$ und die Endpunkte s und t des zu ihrer Bildung führenden Flusses fest. Um diese später einzusetzen, lässt sich folgende Beobachtung machen:

Jede Kante e_i kodiert eine Bipartition der Knotenmenge V entsprechend eines der ausgewählten Minimalen s-t-Schnitte. In dem später für den neu geschaffenen Superknoten S_i erzeugbaren Kontraktionsgraphen G'_{S_i} existieren für die maximal zwei zu S_i inzidenten neuen T-Kanten e_{i-1} und e_i Kontraktionsknoten $[e_{i-1}]$, $[e_i]$ welche jeweils eine dieser Partitionen zusammenfassen. Dabei enthält (falls e_{i-1} existiert) $[e_{i-1}]$ den Knoten s und (falls e_i existiert) $[e_i]$ den Knoten t. Da die Kontraktion die Minimalität der Schnitte nicht verändert (siehe Abschnitt 6.3.1), ist in diesem Kontraktionsgraphen also ein Minimaler Blattschnitt für beide Kontraktionsknoten bekannt. Der Schnittpartner für $[e_{i-1}]$ ist dabei entweder $[e_i]$ oder der Knoten t, falls dieser unkontrahiert im Kontraktionsgraphen vorkommt. Selbiges gilt für $[e_i]$ und dessen Blattschnittbeziehung zu $[e_{i-1}]$ bzw. den Knoten s.

Jede der in L eingefügten Kanten speichert also einen neuen bekannten Blattschnitt ein einem aus T bildbaren Kontraktionsgraphen.

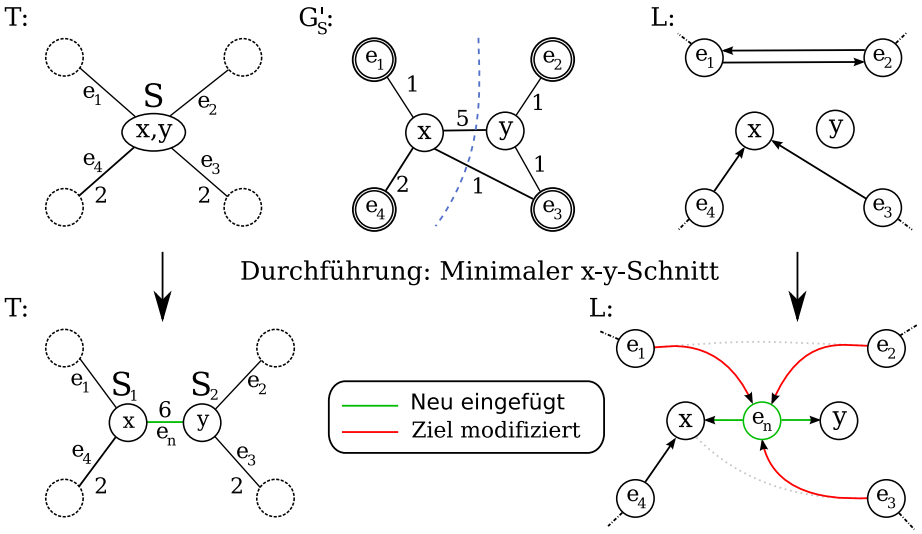


Abbildung 8: Adaption des Graphen L während Algorithmusphase 1

Anpassung der Kantenverbindungen getrennter Kantenknoten: Durch Betrachtung des L-Teilgraphen über den Kantenknoten zu S inzidenter T-Kanten und den zu S gehörenden Knoten $v \in V$ sind die für G'_S bereits bekannten Schnittbeziehungen bestimmbar. Um diese Information für die zukünftigen Kontraktionsgraphen von S_1 bis S_k anzupassen, muss L modifiziert werden.

Durch die Zerlegung des Superknotens S in S_1 bis S_k sind viele der ehemals zum selben Superknoten S inzidenten Kanten nun an verschiedenen Superknoten verankert. In Abbildung 8 trifft dies auf die T-Kanten e_1 und e_2 zu. Da Kontraktionsknoten für diese Kanten nicht länger gemeinsam in einem Kontraktionsgraphen auftreten können, die Blattschnittbeziehung dort aber nun auf einen neuen Partner verweist, müssen zwischen e_1 und e_2 in L verlaufende Kanten adaptiert werden.

Neues Ziel der Kante $e_1 \rightarrow e_2$ wird mit e_n der Kantenknoten der ersten *neu eingefügten* Kante in T, welche dort auf dem Weg von e_1 zu e_2 liegt. Für diesen existiert im Kontraktionsgraphen G'_{S_1} zum neu gebildeten und zu e_1 sowie e_n inzidenten Superknoten S_1 ein Kontraktionsknoten $[e_n]$, für welchen $\{[e_1]\}$ ein Minimaler $\{[e_1]\}$-$[e_n]$-Schnitt ist. Dies gilt, da $[e_n]$ den ehemaligen Blattschnittpartner enthält (siehe Abschnitt 6.3.1).

Neben zwischen den Kantenknoten nicht länger adjazenter T-Kanten verlaufenden L-Kanten, müssen noch weitere adaptiert werden. Dabei handelt es sich um L-Kanten deren Ziel ein Knoten $x \in V$ ist, der sich nach der Zerlegung nicht mehr in einem Superknoten befindet, der zur T-Kante des Kantenausgangspunkts inzident ist. In Abbildung 8 gilt dies für die L-Kante $e_3 \rightarrow x$.

Hier wird der Kantenknoten e_n der ersten *neu eingefügten* T-Kante auf dem Weg

von T-Kante e_3 zum neuen Superknoten von x das neue Ziel der L-Kante.

Der als neues Ziel eingesetzte Kantenknoten ist jeweils leicht bestimmbar: Je neuem Superknoten S_i existieren maximal zwei neue T-Kanten. Die Richtige ergiebt sich durch Bestimmung, ob der dem alten Zielknoten zugeordnete Knoten in G'_S in einer Flusskomponente liegt, welche in der Flusskomponentenordnung höher oder niedriger positioniert wurde, als die Komponente des dem Kantenausgangspunkt zugeordneten Knoten in G'_S.

Abbildung 8 zeigt die in Phase 1 notwendig werdenden Adaptionen an L kombiniert für einen beispielhaften x-y-Schnitt. Dieser führt zur Spaltung des Superknotens S und dem Hinzufügen der Kante e_n in T. Ein entsprechender Kantenknoten und die grün dargestellten Kanten werden in L aufgenommen um die aus diesem Schnitt gewonnenen Informationen zu speichern. Die Zerlegung trennt in G'_S mehrere Knoten, für die in L Blattschnittbeziehungen gespeichert waren. Für diese L-Kanten muss der neu erzeugte Kantenknoten e_n als Zielpunkt eingesetzt werden. Die entsprechend modifizierten Kanten sind rot dargestellt.

Queue-Management Jeder neu geschaffene Superknoten dessen Knotenmenge eine Kardinalität größer Eins aufweist, wird am Queueende angefügt und somit für eine weitere Teilung vorgesehen.

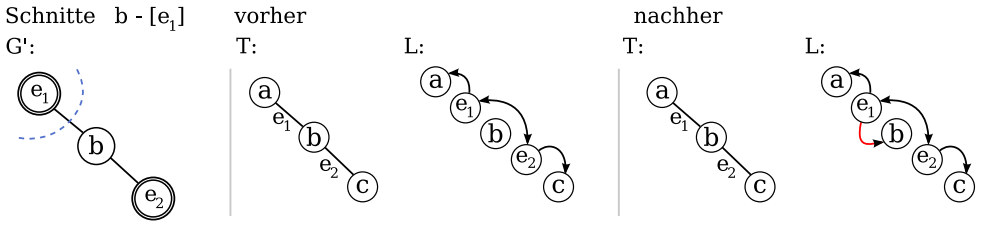
5.3.5. Phase 2: Identifikation und Durchführung ausstehender Schnittoperationen

Die zweite Phase beginnt, sobald die Queue in Phase eins leer läuft, dass heißt, sobald jeder Superknoten in T nur noch einen einzigen Knoten $v \in V$ repräsentiert. Vor dem eigentlichen Ablauf werden sämtliche Superknoten, welche in T nicht als Blatt auftreten, wieder in die Queue eingestellt.

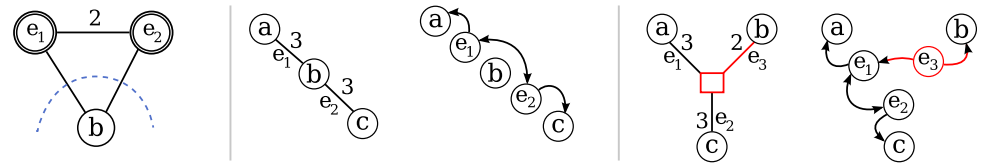
Erstellung des Kontraktionsgraphen Solange die Auftragsqueue gefüllt ist, wird nun jeweils das erste Element S entnommen und der dazugehörige Kontraktionsgraph G'_S erstellt.

Im Gegensatz zur Phase 1 kann es sich beim Queue-Knoten S nun nicht nur um Superknoten sondern auch um Leere Knoten handeln. An der Konstruktionmethodik von G'_S ändert dies nichts, es werden wiederum alle von S abgehenden Teilbäume von T in Kontraktionsknoten zusammengefasst. Da mit dem Leeren Knoten selbst keine Knoten $v \in V$ assoziiert sind, wird der Kontraktionsgraph vollständig aus Kontraktionsknoten bestehen.

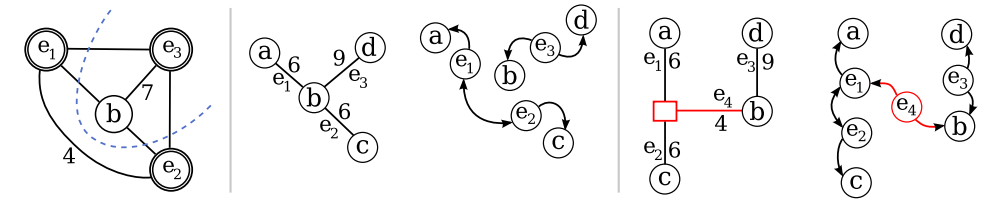
Bestimmung des Schnittpaars Um die Knotenpaare für im Weiteren notwendige Schnitte zu bestimmen, werden die Knoten des Kontraktionsgraphen G'_S erneut Knoten aus L zugeordnet. Das Vorgehen ist identisch mit Phase 1: Jeder Knoten $b \in V$ welcher im Kontraktionsgraphen vorkommt, existiert auch in L. Jedem Kontraktionsknoten wird der Kantenknoten der zum Queue-Knoten S in T inzidenten Kante seines T-Teilbaums zugeordnet.



(a) Blattschnitt eines Kontraktionsknotens



(b) Blattschnitt des Knotens $b \in V$



(c) Nichttrivialer Schnittverlauf

Abbildung 9: T- und L-Adaptionen in Phase 2 (jeweils nach Min. b-$[e_1]$-Schnitt) (Kantengewichte von 1, falls nicht dargstellt)

Die in diesen Abbildungen zusammengefassten Knoten induzieren einen Teilgraphen in L, über welchem nun ein *wurzelknotengerichteter* Spannwald berechnet wird. Hat er die Form eines Spannbaums, so ist für den aktuellen Queue-Knoten S kein weiterer Aufwand notwendig (siehe Abschnitt 6.3.2) und der nächste Queue-Knoten kann entnommen werden. Enthält der Spannwald jedoch mindestens zwei Bäume, so werden deren Wurzelknoten bestimmt, ein Paar dieser Knoten ausgewählt und der Minimale Schnitt zwischen ihren Abbildungen in G'_S berechnet.

Existiert ein Schnittverlauf, welcher in beiden Schnittmengen mehr als einen Knoten besitzt, kann dieser bevorzugt werden, um weitere Schnittberechnungen einsparen zu können (Details siehe Abschnitt 6.4). Bei kleinen Kontraktionsgraphen kann dies jedoch auch zu unverhältnismäßigem Verwaltungsaufwand führen.

Adaption von T und L Für den resultierenden Schnittverlauf sind drei Fälle zu unterscheiden. Abbildung 9 stellt sie und die notwendigen Adaptionen dar. Für eine weitergehende Begründung der vorgenommenen Änderungen siehe Abschnitt 6.3.3.

Blattschnitt eines Kontraktionsknotens: Zum Einen kann eine der Schnittmengen vollständig von genau einem Kontraktionsknoten gebildet werden. In diesem Fall, wird in L eine zusätzliche Kante eingefügt (in Abbildung 9(a) die Kante $e_1 \rightarrow b$). Sie verweist vom Kantenknoten, welcher mit dem per Blattschnitt getrennten Kontraktionsknoten assoziiert ist, zum L-Knoten des Schnittpartners aus G'_S. Nun kann erneut mit der Spannbaumberechnung in L überprüft werden, ob weitere Schnitte notwendig sind, und diese auf dem bereits kontrahierten Graphen G'_S durchgeführt werden.

Blattschnitt von $b \in V$***:*** Alternativ kann eine der Schnittmengen nur aus dem einzigen Originalknoten b bestehen (vgl. Abbildung 9(b)). Hier wird in T an der bisherigen Stelle des Superknotens von b ein Leerer Knoten eingesetzt und der Superknoten über eine Kante mit dem festgestellten Schnittgewicht als Blatt an diesen angebunden.

Der Kantenknoten für die neue T-Kante wird in L eingefügt und erbt dort die Position des Knotens b in sämtlichen auf diesen verweisenden Kanten. Zusätzlich erhält er Auswärtskanten zum Kantenknoten des Schnittpartners von b (im Beispiel der Knoten e_1) und zu b selbst. Da der Superknoten von b nun jedoch ein Blatt in T ist, und somit kein Kontraktionsknoten mehr für ihn gebildet werden muss, ist letztere Kante von einer Implementierung ebenso vernachlässigbar.

Anschließend wird der neue Leere Knoten zur späteren Bearbeitung am Ende der Auftragsqueue eingefügt.

Nichttrivialer Schnittverlauf: Bestehen beide Schnittmengen aus mehr als einem Knoten, so wird wiederum ein Leerer Knoten in T eingefügt.

Handelte es sich beim Queue-Knoten S um einen Superknoten, so ersetzt ihn der Leere Knoten als Wurzel all der T-Teilbäume, deren Kontraktionsknoten nicht mit dem zu S gehörenden Originalknoten $b \in V$ in der selben Schnittmenge liegen. Dies trifft im Beispiel in Abbildung 9(c) zu, so dass dort die von e_1 und e_2 eingeleiteten Teilbäume an den Leeren Knoten angehangen werden, während e_3 am alten Superknoten verbleibt.

Ist S dagegen selbst ein Leerer Knoten, so bildet der neue Leere Knoten die Wurzel der Teilbäume von Kontraktionsknoten der kleineren Schnittmenge. Die obersten Kanten der Teilbäume in T werden dabei jeweils nur in einem Endpunkt modifiziert.

Als nächstes wird zwischen S und dem neuen Leeren Knoten eine mit dem Schnittgewicht bewertete Kante eingefügt.

Der zu ihr gehörende neue Kantenknoten in L erhält zwei Auswärtskanten in Richtung der Schnittpartner. Handelt es sich bei einem der Endpunkte des soeben berechneten Minimalen Schnitts um den (einzigen) unkontrahierten Knoten in G'_S, so ist dieser Ziel einer der Kanten (im Beispiel die Kante $e_4 \rightarrow b$). Handelt es sich bei einem Endpunkt um einen Kontraktionsknoten, wird dessen zugehöriger Kantenknoten in L als Kantenziel verwendet (im Beispiel $e_4 \rightarrow e_1$).

Wie bereits in Phase 1 können durch das Einfügen des neuen Knotens und der neuen Kante in T dort vormals zum selben Knoten inzidente Kanten nun getrennt worden sein. In diesem Fall treten deren Kontraktionsknoten nicht mehr gemeinsam in einem Kontraktionsgraphen auf. Bestanden zwischen beiden bekannte Blattschnittbeziehungen in L, beerbt sie in zukünftigen Kontraktionsgraphen jeweils ein anderer Kontraktionsknoten in ihrer Rolle als Schnittziel.

Gleiches gilt für L-Kanten zwischen den Kantenknoten nun nicht mehr zu S inzidenter T-Kanten und dem unkontrahierten Knoten b. Auch hier muss ein neues Ziel eingesetzt werden um die Schnittbeziehung späteren Kontraktionsgraphen nutzen zu können.

Um dem Rechnung zu tragen, werden die im für G'_S relevanten Teilgraphen von L verlaufenden Kanten überprüft und ihr Ziel auf den neu eingesetzten Kantenknoten adaptiert, falls ihre Quelle und ihr Ziel nun in T nicht mehr direkt benachbart sind.

Abschließend werden sowohl der alte Queue-Knoten, als auch der neue Leere Knoten wieder am Queue-Ende eingefügt und der nächste Knoten entnommen.

Terminierung Wird die Queue in Phase 2 vollständig geleert, so ist die Bildung des Relaxierten Gomory-Hu-Baums abgeschlossen. Falls dies gewünscht ist, kann nun, den Erläuterungen des Abschnitts 5.5 folgend, mit quadratischem worst-case Zeitaufwand der Gomory-Hu-Baum von G abgeleitet werden.

5.4. Beispielverlauf

An einem Beispiel soll der vollständige Ablauf noch einmal verdeutlicht werden. Der hierbei verwendete Graph ist in Abbildung 10 zu sehen. Werden im Folgenden Kontraktionsknoten textuell erwähnt, so sind sie durch eckige Klammern dargestellt, welche die enthaltenen Knoten aus V umfassen (z.B. $[abc]$). Eine Darstellung in geschweiften Klammern bezeichnet, je nach Kontext, Knotenmengen, Flusskomponenten oder Superknoten.

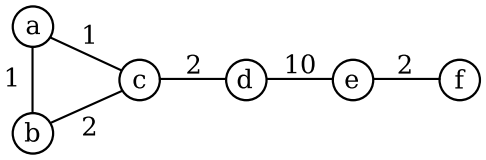


Abbildung 10: Beispielgraph

Initialisierung Die Initialisierung erfolgt wie beschrieben: Der einzige Superknoten in T fasst sämtliche Knoten des Graphen G zusammen und bildet den einzigen Eintrag in der Auftragsqueue Q. L besteht aus der Knotenmenge von G und enthält weder Kantenknoten noch Kanten.

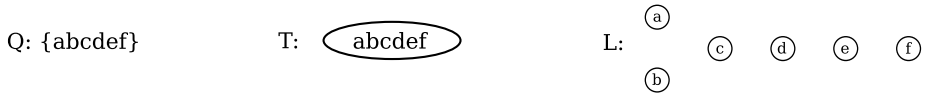


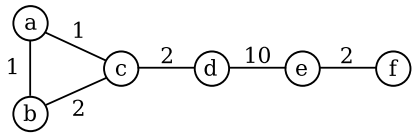
Abbildung 11: Initialisierung von Q, T und L vor Algorithmusbeginn

Phase 1 Es folgt die erste Phase des Algorithmus'. Ihre Auswirkungen auf die mitgeführten Datenstrukturen werden in Abbildungen 12, 13 und 14 verdeutlicht. Die nach jeder Flusskomponentenzerlegung modifizierten Kanten werden darin rot hervorgehoben.

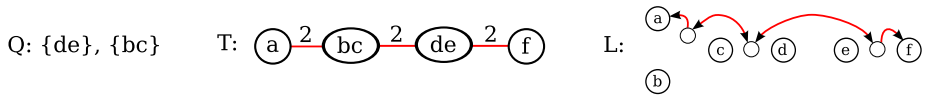
Im angenommenen Algorithmusverlauf werden die Knoten a und f als erstes Flusspaar ausgewählt und ein Maximaler Fluss zwischen ihnen berechnet. Der zugehörige Residualgraph besitzt die starken Zusammenhangskomponenten $\{a\}, \{b, c\}, \{d, e\}$ und $\{f\}$, welche in dieser Reihenfolge auch eine korrekte Folge von Flusskomponentn darstellen. Die Zerlegung ist als erste Zerlegung trivialerweise *gültig*.

Der Superknoten in T kann also in je einen Superknoten pro Flusskomponente aufgespalten werden. Zwischen den neuen Superknoten verlaufen drei Kanten, für welche korrespondierende Kantenknoten in L eingefügt werden. Zwischen den Kantenknoten der neu enstandenden T-Kanten welche zum selben Superknoten inzident sind, werden beidseitig gerichtete Kanten in L eingefügt. Dazu kommt eine gerichtete Kante welche vom Kantenknoten der T-Kante die zu a's Superknoten inzident ist, in Richtung a verläuft. Außerdem wird eine entsprechende Kante für f eingefügt.

Abschließend werden mit $\{d, e\}$ und $\{b, c\}$ die neuen Superknoten mit mehr als zwei enthaltenen Knoten wieder in die Queue eingefügt. Es kommt zur in Abbildung 12(b) dargestellten Situation.



(a) $G'_{\{abcdef\}}$



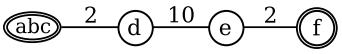
(b) Q, T und L nach Flusskomponentenzerlegung für Maximalen a-f-Fluss in $G'_{\{abcdef\}}$

Abbildung 12: Datenstrukturen zur ersten Maximalflussberechnung

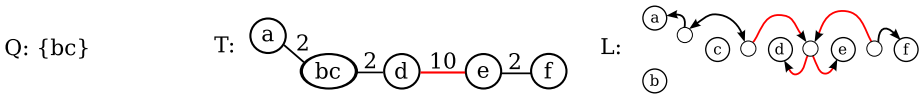
Im nächsten Schritt wird $\{d, e\}$ entnommen und der, in Abbildung 13(a) dargestellte, Kontraktionsgraph $G'_{\{de\}}$ gebildet. In diesem kommt es zur Berechnung eines Maximalen d-e-Flusses. Die resultierende Flusskomponentenfolge ist $\{[abc], d\}, \{e, [f]\}$. Die Folge ist *gültig* und der Superknoten $\{d, e\}$ kann zerlegt werden.

Die Nachbarschaftszuordnung der Superknoten in T erfolgt entsprechend der Einteilung in Flusskomponenten. So wird der bestehende Teilbaum mit den Superknoten $\{a\}$ und $\{b, c\}$ an den neuen Knoten $\{d\}$ angehangen und der den Knoten $\{f\}$ enthaltende Teilbaum ist $\{e\}$ zugeordnet.

Für die neue T-Kante wird ein Kantenknoten in L aufgenommen und von ihm zu Quelle d und Senke e des Flusses führende Kanten eingefügt. Weitere *neue* Kanten



(a) $G'_{\{de\}}$



(b) Q, T und L nach Flusskomponentenzerlegung für Maximalen d-e-Fluss in $G'_{\{de\}}$

Abbildung 13: Datenstrukturen zur zweiten Maximalflussberechnung

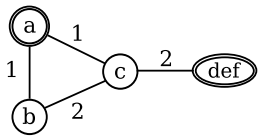
sind aufgrund der Aufteilung in nur zwei Flusskomponenten nicht nötig.

Für die Kontraktionsknoten des Kontraktionsgraphen G' wird nun zuerst die ihren T-Teilbaum einleitende Kante und dann deren Kantenknoten in L bestimmt. Diese Kantenknotenmenge ist entsprechend der Zuteilung der Kontraktionsknoten zu den Flusskomponenten partitioniert.

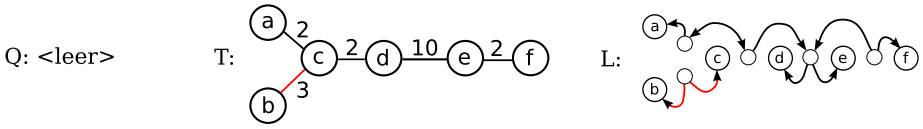
Die Zielknoten über die Partitionsgrenzen hinweg verlaufender L-Kanten, werden auf den neu angelegten Kantenknoten angepasst, dessen Kante in T zum Superknoten der Flusskomponente des Kantenstarts inzident ist und in Richtung des Superknoten der Flusskomponente des ursprünglichen Kantenziels führt.

Im Beispiel betrifft dies beide ehemals zu $\{d, e\}$ inzidenten Kanten. Die Zuteilung der durch sie eingeleiteten Kontraktionsknoten $[abc]$ und $[f]$ in verschiedene Flusskomponenten führt zur Modifikation der zwischen ihren Kantenknoten verlaufenden Kanten. Neues Ziel beider Kanten ist der neu eingefügte Kantenknoten.

Da keiner der neuen Superknoten mindestens zwei Knoten beinhaltet, erhält die Auftragsqueue keine weiteren Elemente. Die obige Abbildung 13(b) zeigt den nun eingenommenen Zustand der Datenstrukturen.



(a) $G'_{\{bc\}}$



(b) Q, T und L nach Flusskomponentenzerlegung für Maximalen b-c-Fluss in $G'_{\{bc\}}$

Abbildung 14: Datenstrukturen zur dritten Maximalflussberechnung

In der nächsten Iteration wird ihr der Superknoten $\{b, c\}$ entnommen und wieder der passende Kontraktionsgraph $G'_{\{bc\}}$ gebildet. Als Flussendpunkte werden b und c gewählt und der Maximale Fluss bestimmt. Die Flusskomponentenfolge ist $\{b\}, \{[a]\}, \{c, [def]\}$. Da die mittlere Flusskomponente ausschließlich Kontraktionsknoten enthält, wird sie mit einem Nachbarn, in diesem Fall $\{c, [def]\}$ vereinigt.

Es folgt die Zerlegung des Superknotens in T sowie das Einfügen des neuen Kantenknotens und der benötigten neuen Kanten in L (vgl. Abbildung 14). Eine Überprüfung der Beziehungen der bestehenden Kantenknoten in L zeigt, dass keine Kanten adaptiert werden müssen.

Wiederum ist kein Superknoten mit mehr als einem vertretenen Knoten entstanden, so dass keine neuen Aufträge in die Queue eingefügt werden. Sie läuft damit leer und Phase 2 beginnt.

Phase 2 Zu Beginn dieses Abschnitts werden zunächst sämtliche nicht als Blatt auftretenden Superknoten aus T in die Auftragsqueue eingestellt. An erster Stelle steht der Superknoten $\{c\}$, welcher entnommen wird, um den zugehörigen Kontraktionsgraphen $G'_{\{c\}}$ zu bilden. Die an seiner Bearbeitung beteiligten Datenstrukturen sind in Abbildung 15 genauer dargestellt.

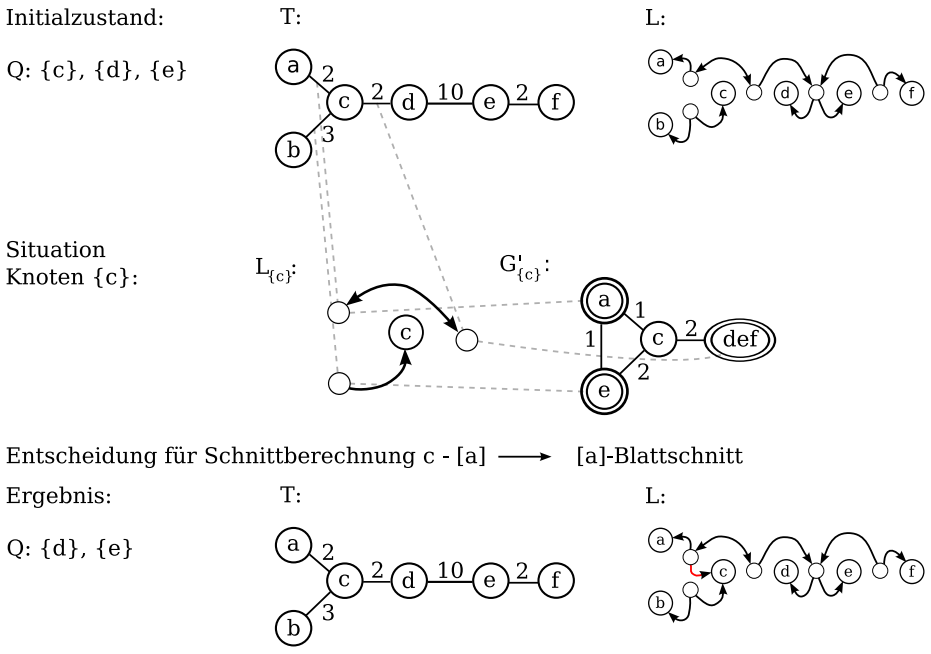


Abbildung 15: Bearbeitung des Superknotens {c} in Phase 2

Um die noch notwendigen Schnittberechnungen zu analysieren, werden in L die Kantenknoten für die zu $\{c\}$ in T inzidenten Kanten bestimmt. Diese induzieren zusammen mit dem Knoten c in L einen Teilgraphen, welcher hier $L_{\{c\}}$ genannt werden soll. Jeder der beteiligten Kantenknoten, steht für eine T-Kante, die den Teilbaum eines Kontraktionsknoten aus $G'_{\{c\}}$ einleitet und lässt sich so auf genau einen Kontraktionsknoten abbilden.

Nun wird ein Spannwald in $L_{\{c\}}$ berechnet. Die Wurzelknoten der enthaltenen Bäume sind c und der Kantenknoten für den Kontraktionknoten $[a]$. Dies führt zur Auswahl der Knoten c und $[a]$ als Schnittpartner für die nächste durchzuführende Flussberechnung.

Das Ergebnis des Minimalen c-$[a]$-Schnitts in $G'_{\{c\}}$ ist ein $[a]$-Blattschnitt. Der bestehende Baum T muss somit nicht verändert werden, da er die Schnittbeziehung bereits korrekt wiedergibt (siehe Abschnitt 6.3.3). In L wird für den gefundenen Blattschnitt eine Kante vom mit dem Kontraktionsknoten $[a]$ verbundenen Kantenknoten zu c eingefügt, um den Schnitt in allen späteren Kontraktionsgraphen von c zu repräsentieren.

Da die Schnittberechnung zu einem Blattschnitt geführt hat und somit kein Umbau von T notwendig war, kann direkt mit $G'_{\{c\}}$ weitergearbeitet werden. Wiederum wird der Spannwald für $L_{\{c\}}$ berechnet, welcher nun einem Spannbaum entspricht. Daraus folgt, dass für den Superknoten $\{c\}$ keine weitere Arbeit zu verrichten ist.

In der Folge werden die nächsten Knoten aus der Auftragsqueue Q entnommen und für diese jeweils der Kontraktionsgraph und der entsprechende Teilgraph von L gebildet. Da auch hier jeweils bereits ein Spannbaum auffindbar ist, sind keine weiteren Schnittberechnungen notwendig.

Die Queue läuft leer und der Algorithmus terminiert. T und L befinden sich in dem Zustand, in welchen sie die zuletzt durchgeführte Schnittberechnung versetzt hat. Dieser ist Abbildung 15 zu entnehmen.

5.5. Ableitung eines Gomory-Hu-Baumes

Aus dem relaxierten Gomory-Hu-Baum $T = (V \cup Z, A)$ für den Graphen $G = (V, E)$ lässt sich ein Gomory-Hu-Baum ableiten. Für die weitere Schilderung sei die Knotenmenge Z als die Menge der Leeren Knoten bezeichnet.

Die Umwandlung geschieht durch eine Kontraktionsoperationen je Leerem Knoten $l \in Z$. Dabei wird l in seinen höchst adjazenten Nachbarknoten h aufgenommen. Der Leere Knoten l und die schwerste seiner inzidenten Kanten werden aus dem Baum gelöscht und der Kontraktionsknoten h nimmt dessen Platz in sämtlichen verbliebenen, voher zu l inzidenten Kanten ein.

Der Relaxierte Gomory-Hu-Baum für G besitzt maximal $2n-2$ Knoten, und somit $2n-3$ Kanten (siehe Abschnitt 6.5). Der maximale Vergleichs- und Adaptionsaufwand pro ausgeführter Kontraktion richtet sich nach der Kantenzahl, ist also in $\mathcal{O}(|V|)$, womit der Gesamtaufwand in $\mathcal{O}(|V|^2)$ liegt.

Lemma 5.1
Ist $T = (V \cup Z, A)$ ein Relaxierter Gomory-Hu-Baum, $l \in Z$ ein Leerer Knoten und entsteht der Baum B aus T durch die Kontraktion von l in seinen höchstadjazenten Nachbarknoten h, dann gilt für zwei Knoten $u, v \in (V \cup Z \setminus \{l\})$: $\lambda_T(u, v) = \lambda_B(u, v)$.

Beweis: Aufgrund des azyklischen Charakters des Relaxierten Gomory-Hu-Baumes, existiert zwischen jedem Paar von Knoten nur ein einziger Pfad. Der maximale Fluss zwischen diesem Knotenpaar muss an ihm entlang verlaufen und schöpft dabei den Flusswert der niedrigsten Kante des Pfades vollständig aus. Diese Kante besitzt den Wert des Minimalen Schnittes.

Für alle über den zu kontrahierenden Leeren Knoten l in T verlaufenden Pfade, bis auf jene in welchen er einen Pfadendpunkt darstellt, gilt, dass diese exakt zwei zu l inzidente Kanten beinhalten müssen. Wird l mit seinem höchstadjazenten Nachbarknoten kontrahiert, so vernichtet dies die höchstwertige inzidente Kante. Jeder Pfad in T zwischen zwei Knoten $u, v \in (V \cup Z \setminus \{l\})$ erfüllt nun einen der drei folgenden Fälle:

1. Der u,v-Pfad verläuft nicht über den Knoten l, enthält somit keine dessen inzidenter Kanten und wird von der Operation nicht beeinflusst.
2. Der u,v-Pfad verläuft über den Knoten l, enthält jedoch dessen schwerste Kante nicht. Da sämtliche anderen Kanten bestehen bleiben, wird auch dieser Pfad nicht beeinflusst.
3. Der u,v-Pfad verläuft über den Knoten l und über dessen schwerste Kante. Da er jedoch mindestens eine weitere (nicht schwerere) zu l inzidente Kante enthält, wird der Minimalwert seiner Kanten nicht beeinflusst.

Die Kontraktionsoperation verändert die Kantenfolge eines Pfades also entweder gar nicht oder sie entfernt nicht seine einzige Minimalkante.

□

Damit bleibt also insbesondere auch der Minimale Schnittwert zwischen allen Knoten $v, w \in V$ im entstehenden Baum erhalten.

Lemma 5.2
Entsteht der Baum B aus dem Relaxierten Gomory-Hu-Baum $T = (V \cup Z, A)$ durch die Kontraktion eines Leeren Knotens $l \in Z$ in seinen höchstadjazenten Nachbarknoten h und verändert dies eine bestehende Kante $e = \{x, l\}$ aus T mit $x \in (V \cup Z) \setminus \{l, h\}$ zu $e' = \{x, h\}$ in B, so zerlegt die Herausnahme von e bzw. e' den jeweiligen Baum in Teilbäume mit den Knotenmengen T_1, T_2 bzw. B_1, B_2.
Für $x \in B_1$ und $x \in T_1$ gilt dann $B_1 \cap V = T_1 \cap V$ sowie $B_2 \cap V = T_2 \cap V$.

Beweis: Es werden die durch Löschung von e im unveränderten und e' im veränderten Graphen entstehenden Teilbäume betrachet. Der x enthaltende Teilbaum stimmt in

beiden Fällen vollständig überein. Dem h enthaltenden Teilbaum fehlt im kontrahierten Fall ausschließlich der Leere Knoten l. Da dieser kein Element aus V war, bleibt die Menge der in beiden Teilbäumen verwalteten Originalknoten identisch.

□

Nach der Kontraktion sämtlicher Leerer Knoten eines Relaxierten Gomory-Hu-Baums T bleiben im resultierenden Baum B trivialerweise nur noch die Knoten aus V zurück. Zusätzlich ist gemäß Lemma 5.1 gesichert, dass für jedes Knotenpaar $u, v \in V$ die geringwertigste Kante des u-v-Pfades in T auch in B weiterhin Teil des u-v-Pfades ist und laut Lemma 5.2 ihre Entnahme die Knotenmenge V in gleicher Weise teilt, wie sie dies in T getan hätte.

Ist T also ein gültiger Relaxierter Gomory-Hu-Baum, so erfüllt B die Definition eines gültigen Gomory-Hu-Baums.

5.6. Komplexitäten der Teilschritte

Um die Zeitkomplexität des vorgestellten Algorithmus zu beurteilen, sollen zunächst die Äufwände für die eingesetzten Teilalgorithmen betrachtet werden.

Wie im klassischen Gomory-Hu-Algorithmus wird der Großteil der Laufzeit von wiederholten gezielten Maximalflussberechnungen eingenommen. Für diese ist, wie bereits bei der Schilderung des Gomory-Hu-Algorithmus erwähnt, derzeit eine worst-case Schranke von $\mathcal{O}(nm \log n)$ erreichbar (siehe auch Abschnitt 7.2.2). Zusätzlich setzt der Algorithmus auf den Einsatz Maximaler Adjazenzordnungen als Alternative zur Flussberechnung. Deren Bildung verläuft in $\mathcal{O}(n \log n + m)$ mit zusätzlichem Aufwand von $\mathcal{O}(m)$ zur Ableitung des enthaltenen Flusses.

Im Anschluss an die Bestimmung des Maximalflusses folgt die Flusskomponentenzerlegung, deren größter Kostenfaktor in der Bestimmung der Starken Zusammenhangskomponenten des Residualgraphen besteht. Diese Aufgabe lässt sich mittels des Algorithmus von Tarjan (Bsp. [Pea05]) in $\mathcal{O}(n+m)$ lösen. Es folgen topologische Sortierung, Kontraktion und Baummanipulation, welche zusammen ebenso mit $\mathcal{O}(n+m)$ begrenzt werden können ([TS92]), wobei dieser worst-case Wert in der Praxis nur im Ausnahmefall erreicht wird.

In Phase 2 sinkt der Aufwand während der Nachverarbeitung, dafür wird hier zusätzlich die Identifikation noch notwendiger Schnittberechnungen notwendig. Im Maximum verursacht sie im Gesamtverlauf $(2n-2)$ Spannwaldbildungen, alias Tiefensuchen, auf Teilgraphen von L. Diese sind im schlechtesten Fall wiederum jeweils mit $\mathcal{O}(n+m)$ abzuschätzen. Dieses Vorgehen ist dabei vergleichsweise grob, da Maximalzahl und Maximalaufwand nicht gleichzeitig eintreten.

Die Komplexität einer Iteration, egal welcher Phase, ist also bestimmt vom Aufwand $\mathfrak{M}(n,m) = \mathcal{O}(nm \log n)$ der Flussberechnung. Wie in Abschnitt 6.4 gezeigt wird, sind diese auf maximal $(n-1)$ Durchläufe beschränkt. Damit liegt die Gesamtkomplexität, wie die des klassischen Gomory-Hu-Algorithmus, im Moment bei $\mathcal{O}(n\mathfrak{M}(n,m))$.

6. Theoretische Motivation der Flusskomponentenzerlegung

Durch den Einsatz von Flusskomponentenzerlegungen lässt sich die Zahl der durch einen Superknoten vertretenen Originalknoten aus V schnell absenken. Damit können auf einer gleichen Problemstellung durchschnittlich kleinere Kontraktiongraphen erreicht werden, als mit der klassischen Bipartition nach Gomory-Hu.

Im ersten Teil dieses Abschnitts soll gezeigt werden, dass auch bei einer Flusskomponentenzerlegung der Knotenmenge V, die in den resultierenden Kontraktionsgraphen G' berechenbaren Minimalen Schnittverläufe global auf G gültig sind.

Da die zur Bildung des Kontraktionsgraphen G' durchgeführte Knotenkontraktion die Menge der in G' auffindbaren, zwischen zwei Originalknoten aus V verlaufenden und global gültigen Minimalschnitte begrenzt, muss garantiert sein, dass für jedes Paar am Kontraktionsgraphen G' beteiligter Originalknoten mindestens ein auch in G minimaler Schnittverlauf existiert. Der Schlüssel hierzu liegt wie bei Gomory und Hu in der garantierten Existenz kreuzungsfreier Minimaler Schnitte innerhalb der gewählten Komponentenmengen.

Weiterhin zeigt der Abschnitt, dass es die Kontraktionsgraphen in ihrer Gesamtheit ebenso ermöglichen Minimale Schnitte zwischen *jedem* Knotenpaar von G korrekt aufzufinden.

Als nächstes werden die Eigenschaften von Maximalen Flüssen in einem Kontraktionsgraphen G' untersucht, welche einen Kontraktionsknoten als Quelle oder Senke besitzen. Bilden auch diese Maximale Flüsse zwischen zwei Knoten aus G, so lassen sich ihre Ergebnisse auf gleiche Art und Weise berücksichtigen wie Schnitte zwischen unkontrahierten Knoten. Dies gewährt erst die Nutzung in Phase 2 berechneter Schnittverläufe und ist ebenso hilfreich beim Einsatz von s-t-Schnittalgorithmen, in denen die Auswahl der Schnittpartner nicht gesteuert werden kann. Beispiele hierfür sind Maximale Adjazenzordnungen.

Anschließend folgen Erläuterungen zur Notwendigkeit weiterer Schnittberechnungen, nach Abschluss der Phase 1. Nach der Einführung wird zunächst auf die Wiederverwendbarkeit bereits bekannter Schnittverläufe und deren bestehende Gültigkeit in stärker kontrahierten Graphen eingegangen. Es folgt die Vorstellung und Begründung der Methodik zur Auswahl von Zielknoten für die noch ausstehenden Schnittberechnungen und anschließend notwendig werdende Modifikationen an den mitgeführten Datenstrukturen.

Der Abschnitt schließt mit Einschätzungen der insgesamt notwendigen Schnittberechnungen und der Maximalzahl von Knoten, eines mittels des vorgestellten Algorithmus konstruierten Relaxierten Gomory-Hu-Baums.

6.1. Auffindbarkeit und korrekte Repräsentation Minimaler Schnitte

Ziel dieses Abschnitts ist es zu zeigen, dass es durch Flusskomponentenzerlegungen gelingt, Knotenmengen auszumachen, welche durch weitere Minimale Schnitte nicht

getrennt werden müssen. Diese können für spätere Berechnungen kontrahiert werden, ohne Einfluss auf die Auffindbarkeit Minimaler Schnitte zwischen den verbliebenen Knoten zu haben. Weiterhin wird belegt, dass der vom Algorithmus während Phase 1 konstruierte Baum T die Schnittbeziehungen korrekt widerspiegelt und die Erstellung geeigneter Kontraktionsgraphen ermöglicht.

Wie in Abschnitt 5.3.4 beschrieben wurde, bildet der Algorithmus zu jedem Element einer gültigen Folge von Flusskomponenten einen Superknoten in T. Diese sind entsprechend der Folge linear miteinander verkettet. Jeder dieser Superknoten und die direkt von ihm abgehenden Teilbäume, welche nicht mit einem aus der gleichen Zerlegung stammenden Superknoten beginnen, beinhalten alle Originalknoten die - unkontrahiert oder kontrahiert - an seiner Flusskomponente beteiligt waren.

Im ersten Schritt, wird nun nachgewiesen, dass nach der Zerlegung eines Maximalen s-t-Flusses in seine Flusskomponenten, die Schnitte zwischen Knoten innerhalb einer Flusskomponente auch dann noch korrekt auffindbar sind, falls durch Bildung des Kontraktionsgraphen für einen der erzeugten Superknoten bestimmte Schnittverläufe ausgeschlossen werden. Diese würden Knotenpaare trennen, welche in der Flusskomponentenordnung entweder höher oder niedriger eingeordneten Komponenten zugewiesen sind.

In einem zweiten Schritt wird dann gezeigt, dass Minimale Schnitte zwischen Knoten, die in verschiedenen Flusskomponenten platziert wurden, weiterhin auffindbar sind. Sie lassen sich nach der Flusskomponentenzerlegung in einem der beiden Kontraktionsgraphen bestimmen, in welchen ein solcher Knoten unkontrahiert auftritt.

Weiterhin wird verdeutlicht, dass die gefundenen Schnitt korrekt im Baum T repräsentiert werden.

6.1.1. Kreuzungsfreiheit Minimaler Schnitte zwischen Knoten einer Flusskomponente zu Vereinigungen anderer Komponenten

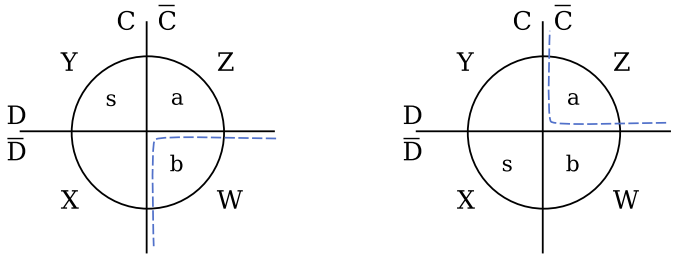
Um die späteren Ergebnisse zu begründen, muss zunächst das folgende Lemma über das Kreuzungsverhalten Minimaler Schnitte eingeführt werden:

Lemma 6.1
Sei $\{C, \overline{C}\}$ ein Minimaler s-t-Schnitt in G mit $s \in C$ und den Knoten $a, b \in \overline{C}$. Zusätzlich sei $\{D, \overline{D}\}$ ein Minimaler a-b-Schnitt, welcher $\{C, \overline{C}\}$ kreuzt.
Liegt s in $C \cap D$, dann ist $\overline{C} \cap \overline{D}$ ein Minimaler a-b-Schnitt in G. Liegt stattdessen s in $C \cap \overline{D}$ so ist $\overline{C} \cap D$ ein Minimaler a-b-Schnitt.

Beweis: Die garantiert nicht-leeren Überschneidungen der Schnittmengen, seien wie in Abbildung 16 mit $W = \overline{C} \cap \overline{D}$, $X = C \cap \overline{D}$, $Y = C \cap D$ und $Z = \overline{C} \cap D$ bezeichnet.

Zusätzlich sei die Definition der Schnittkostenfunktion c_G wie folgt auf zwei Mengen $Q, R \subseteq V$ erweitert:

$$c_G(Q, R) := \sum_{\{x,y\} \in E : x \in Q, y \in R} w_G(\{x, y\})$$



(a) Fall $s \in Y$ (b) Fall $s \in X$

Abbildung 16: Knotenlage und garantierter Minimaler a-b-Schnitt in Lemma 6.1.

. Die bisherige, einstellige Definition ergibt sich als $c_G(P) = c_G(P, \overline{P})$.

Da die Kosten des Minimalen s-t-Schnittes $\{C, \overline{C}\}$ der Kostensumme der über ihn verlaufenden Kanten entsprechen und $C = X \cup Y$ sowie $\overline{C} = W \cup Z$ offensichtlich zutrifft, gilt:

$$c_G(C, \overline{C}) = c_G(Y, Z) + c_G(Y, W) + c_G(X, Z) + c_G(X, W) \tag{1}$$

Zunächst werde nun $s \in Y = C \cap D$ angenommen. Da der s-t-Schnitt $\{C, \overline{C}\}$ minimal ist, gilt:

$$c_G(C, \overline{C}) \leq c_G(Y, Z) + c_G(Y, W) + c_G(Y, X)$$

Durch Gleichsetzen mit Formel (1) und einseitigen Abzug von $c_G(X, Z)$ folgt daraus:

$$c_G(X, W) \leq c_G(Y, X)$$

Somit sind die Schnittkosten von $W = \overline{C} \cap \overline{D}$ garantiert nicht größer als der bekannte Minimale a-b-Schnitt $\{D, \overline{D}\}$:

$$c_G(Y, W) + c_G(Z, W) + c_G(X, W) \leq c_G(Y, W) + c_G(Z, W) + c_G(X, Z) + c_G(Y, X)$$

$$c_G(W) \leq c_G(D, \overline{D})$$

Gilt $s \in X = C \cap \overline{D}$, so ist die Beweisführung ähnlich. Wieder folgt aus der Minimalität des s-t-Schnitts $\{C, \overline{C}\}$:

$$c_G(C, \overline{C}) \leq c_G(Y, X) + c_G(X, Z) + c_G(X, W)$$

Zusammen mit Gleichung (1) gilt damit:

$$c_G(Y, Z) \leq c_G(Y, X)$$

Somit sind in diesem Fall die Schnittkosten von $Z = \overline{C} \cap \overline{D}$ garantiert nicht größer als der bekannte Minimale a-b-Schnitt $\{D, \overline{D}\}$:

$$c_G(Y,Z) + c_G(X,Z) + c_G(W,Z) \leq c_G(Y,X) + c_G(X,Z) + c_G(W,Z) + c_G(Y,W)$$

$$c_G(Z) \leq c_G(D, \overline{D})$$

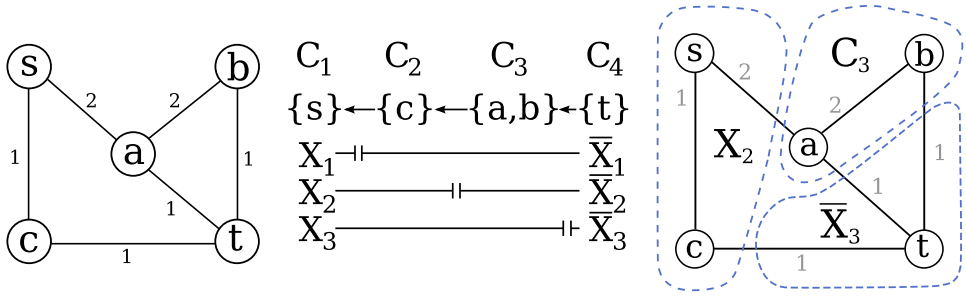
□

Mit diesem Ergebnis lassen sich nun die Minimalen Schnitte zwischen Knoten einer Flusskomponente analysieren.

Aus dem Maximalen Fluss zwischen zwei Knoten s und t des Graphen G sei gemäß Abschnitt 5.3.4 eine Folge von Flusskomponenten $C_1, ..., C_k$ konstruiert worden. Im Weiteren gelte $s \in C_1$ und $t \in C_k$.

Aus diesen $k \geq 2$ Flusskomponenten entsteht eine Folge von $(k-1)$ s-Schnittmengen Minimaler s-t-Schnitte wie folgt:

Die erste s-Schnittmenge X_1 bildet mit C_1 diejenige Komponente, welche die Quelle s enthält. Jede weitere Schnittmenge X_i entstehe aus $X_{i-1} \cup C_i$. Es gilt $\overline{X}_i := V \setminus X_i$. Abbildung 17 zeigt ein Beispiel einer solchen Konstruktion.



(a) Beispielgraph

(b) s-t-Flusskomponentenfolge und ableitbare s-Schnittmengen X_i

(c) Situation C_3

Abbildung 17: Flusskomponenten C_i und umschließende Schnittmengen X_{i-1} und $\overline{X}_i$

Lemma 6.2
Für jedes Knotenpaar $a, b \in C_i$ existiert ein Minimaler a-b-Schnitt in G, welcher weder X_{i-1} noch $\overline{X}_i$ kreuzt.

Beweis: Jedes Paar $\{X_i, \overline{X}_i\}$ entspricht einem Minimalen s-t-Schnitt in G. Da $a, b \in C_i$ und $C_i \subseteq X_i$ gilt, existiert somit gemäß Lemma 3.1 ein Minimaler a-b-Schnitt, welcher $\overline{X}_i$ nicht kreuzt. Ebenso gibt es einen weiteren Minimalen a-b-Schnitt, welcher X_{i-1} nicht kreuzt, denn es gilt auch $C_i \subseteq \overline{X}_{i-1}$.

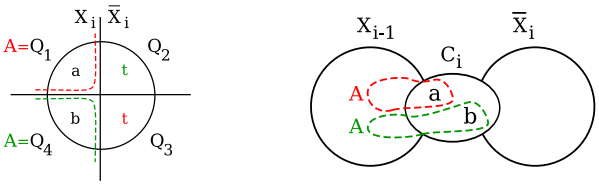


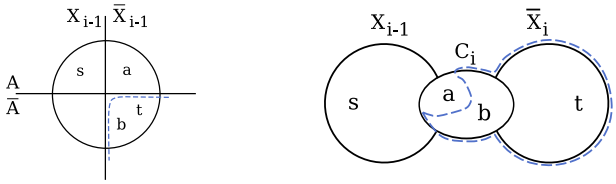
Abbildung 18: Minimaler a-b-Schnitt $A \subset X_i$ in Abhängigkeit der Lage von t in $\overline{X}_i$.

Handelt es sich bei C_i um C_1 oder C_k, gilt Lemma 6.2 also bereits nach Gomory und Hu, da für C_1 keine Menge X_{i-1} und für C_k kein $\overline{X}_k$ existiert.

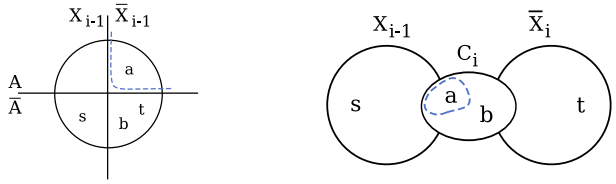
Im Weiteren gelte $1 < i < k$. Gemäß Lemma 6.1 existiert in X_i der Minimale a-b-Schnitt A mit $A \subset X_i$. In der Darstellung 18 ist dies, abhängig von der Lage von t, der Quadrant Q_1 oder Q_2.

Ohne Beschränkung der Allgemeinheit, gelte für die weitere Schilderung $a \in A$ und $t \notin A$. Wird nun die Kreuzung der Schnitte $\{A, \overline{A}\}$ und $\{X_{i-1}, \overline{X}_{i-1}\}$ betrachtet, so können, je nach Lage des Knotens s zwei Fälle unterschieden werden:

Liegt s in A, so kommt es zur in Abbildung 19(a) dargestellten Situation und gemäß Lemma 6.1 ist $\{\overline{A} \cap \overline{X}_{i-1}, A \cup X_{i-1}\}$ ein Minimaler a-b-Schnitt. Die Knotenmenge X_{i-1} liegt offensichtlich vollständig in $A \cup X_{i-1}$. Für $\overline{X}_i$ gilt $A \cap \overline{X}_i = \emptyset$ und $X_{i-1} \cap \overline{X}_i = \emptyset$, weshalb es vollständig in der Schnittmenge $\overline{A} \cap \overline{X}_{i-1}$ liegen muss. Somit wird weder X_{i-1} noch $\overline{X}_i$ vom Minimalen a-b-Schnitt gekreuzt.



(a) Minimaler a-b-Schnitt im Fall $s \in A$, kreuzungsfrei zu X_{i-1} und $\overline{X}_i$



(b) Minimaler a-b-Schnitt im Fall $s \notin A$, kreuzungsfrei zu X_{i-1} und $\overline{X}_i$

Abbildung 19: Minimale a-b-Schnitte in $\overline{X}_{i-1}$ in Abhängigeit der Lage von s

Ähnlich verläuft die Argumentation für $s \notin A$. Dieser Fall ist in Abbildung 19(b) dargestellt. Hier existiert nach Lemma 6.1 der Minimale a-b-Schnitt $\{A \cap \overline{X}_{i-1}, \overline{A} \cup X_{i-1}\}$. Dabei gilt $A \cap \overline{X}_{i-1} \subseteq X_i \cap \overline{X}_{i-1} = C_i$. Damit sind sowohl X_{i-1} als auch $\overline{X}_i$ vollständig in der konträren Schnittmenge enthalten und werden insbesondere nicht gekreuzt.

□

Global gültige Minimale a-b-Schnitte können also in einem kontrahierten Graphen G'_i gefunden werden. Gilt $1 < i < k$, so entsteht dieser Graph durch Kontraktion der Knotenmengen X_{i-1} und $\overline{X}_i$ zu je einem Kontraktionsknoten. Für die Komponenten C_1 und C_k können in G' sämtliche weiteren Komponenten kontrahiert werden.

Eine im Weiteren auf die kontrahierten Graphen für Komponenten eines Maximalen s-t-Flusses beschränkte Verarbeitung ermöglicht weiterhin die korrekte Bestimmung der Schnittbeziehungen zwischen allen Knoten der betrachteten Komponente.

6.1.2. Korrektheit für Schnitte mit Endknoten in verschiedenen Flusskomponenten

Um die Korrektheit der Repräsentation in Baumform zu zeigen, soll zunächst ein Ergebnis über die im Baum T zwischen nicht-adjazenten Superknoten der gleichen Flusszerlegung verlaufenden Pfade aufgestellt werden.

Seien X_{i-1}, C_i und $\overline{X}_i$ wie bisher durch einen s-t-Fluss definierte Schnittmengen, bzw. Flusskomponenten. Im Baum T wurde für jede Flusskomponente C_j ein Superknoten S_j gebildet. Sei e_s die vom Algorithmus im Baum T zwischen dem Superknoten S_{i-1} (dem ersten Vertreter des Teilbaums der Knotenmenge von X_{i-1}) und dem Superknoten S_i eingefügte Kante und sei e_t die vom Algorithmus zwischen den Superknoten S_i und S_{i+1} (als erstem Vertreter von $\overline{X}_i$) eingefügte Kante.

Lemma 6.3
Im vom Algorithmus konstruierten Baum gilt für jede Kante e auf dem Pfad zwischen e_s und e_t: $w_T(e) \geq \lambda(s,t)$.

Beweis: Die Kanten e_s und e_t selbst tragen beide Flusswert $\lambda(s,t)$.

Seien $[s]$ und $[t]$ die Kontraktionsknoten, welche im Kontraktionsgraphen G' zu S_i die in T durch e_s und e_t angebundenen Teilbäume repräsentieren.

Aufgrund der Maximalität des s-t-Flusses kann in G' kein $[s]$ und $[t]$ trennender Schnitt mit einem Gewicht $< \lambda(s,t)$ existieren. Für alle Maximalen Flüsse in G' mit einem Wert $< \lambda(s,t)$ gilt also, dass sie $[s]$ und $[t]$ in der selben Flusskomponente gruppieren, wodurch der vorgestellte Algorithmus alle restlichen Flusskomponenten im konstruierten Baum T als vom e_s-e_t-Pfad abgehende Teilbäumen anordnet.

□

Um die Auffindbarkeit von Schnitten zu analysieren, deren Endpunkte sich in verschiedenen Flusskomponenten befinden, hilft das folgende Lemma.

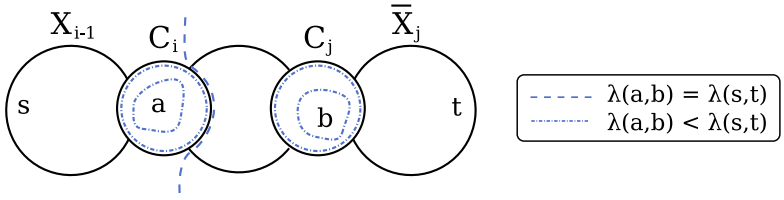


Abbildung 20: Minimale a-b-Schnittverläufe nach Lemma 6.4.

Lemma 6.4
Gegeben sei eine Sequenz von Flusskomponenten $C_1, \cdots, C_k$ eines Maximalen s-t-Flusses und die daraus konstruierbaren s-Schnittmengen $X_i = \bigcup_{r=1..i} C_r$. Seien $a, b \in V$ und gelte $a \in C_i$, $b \in C_j$ und $1 \leq i < j \leq k$, so treffen die folgenden Aussagen zu:

1. Es existiert kein Minimaler a-b-Schnitt mit $\lambda(a,b) > \lambda(s,t)$.
2. Gilt $\lambda(a,b) = \lambda(s,t)$, so ist X_i ein Minimaler a-b-Schnitt.
3. Gilt $\lambda(a,b) < \lambda(s,t)$, so existiert ein Minimaler a-b-Schnitt A mit $s, t \in \overline{A}$ und:
   a) $A \subseteq C_i$, falls $a \in A$.
   b) $A \subseteq C_j$, falls $b \in A$.

Beweis: Die Aussagen 1 und 2 ergeben sich trivial aus der Minimalität des s-t-Schnitts X_i und der Tatsache, dass dieser ebenso a und b trennt.

Aussage 3 setzt $\lambda(a,b) < \lambda(s,t)$ voraus. Ein solcher a-b-Schnitt kann s und t nicht trennen, da dies der Minimalität des bekannten s-t-Schnitts widersprechen würden. Somit muss dieser entweder a von s oder b von t separieren. Ohne Beschränkung der Allgemeinheit sei dies für a und s zutreffend, für b und t gelten symmetrische Betrachtungen. Dann gilt also $\lambda(a,s) \leq \lambda(a,b) < \lambda(s,t)$.

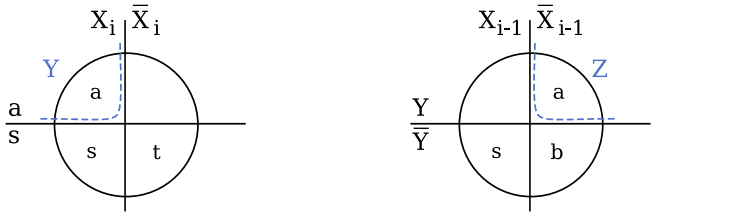
Wird das in Abbildung 21(a) gezeigte Kreuzungsschema des Minimalen a-s-Schnittes betrachtet, folgt gemäß Lemma 6.1, dass eine Schnittversion Y existiert, welche vollständig in X_i verläuft sowie a und s trennt.

Dieser Minimale a-s-Schnitt trennt ebenso die Knoten a und b. Damit gilt auch $\lambda(a,b) \leq \lambda(a,s)$ und in Kombination mit dem bereits festgestellten $\lambda(a,s) \leq \lambda(a,b)$ folgt $\lambda(a,s) = \lambda(a,b)$. Y ist also ein Minimaler a-b-Schnitt in X_i mit $a \in Y$ und $s \notin Y$.

Handelt es sich bei C_i um C_1 so ist der Beweis bereits abgeschlossen. Für $1 < i$ wird das in Abbildung 21(b) gezeigte Kreuzungsschema von $\{Y, \overline{Y}\}$ und $\{X_{i-1}, \overline{X}_{i-1}\}$ betrachtet. Wieder folgt nach Lemma 6.1, dass ebenso ein Minimaler a-b-Schnitt $Z = Y \cap \overline{X}_{i-1}$ mit $a \in Z$ existiert.

Laut Korollar 3 in [PQ80] (Beweis in [Hu70]) entspricht nun auch die Menge $A = Y \cap Z$ einem Minimalen a-b-Schnitt. Weiterhin gilt $A \subseteq X_i \cap \overline{X}_{i-1} = C_i$.

□



(a) Kreuzungsschema Minimaler a-s-Schnitt mit $\{X_i, \overline{X}_i\}$

(b) Kreuzungsschema $\{Y, \overline{Y}\}$ mit $\{X_{i-1}, \overline{X}_{i-1}\}$

Abbildung 21: Schnittkreuzungen im Beweis zu Lemma 6.4.

Aus Lemma 6.4 folgt direkt, dass für die Knoten $a \in C_i$ und $b \in C_j$ mit $i < j$ ein Minimaler Schnitt in einem der Kontraktionsgraphen zu S_i oder S_j bestimmbar ist.

Ebenso ist die Repräsentation im Baum T korrekt: Der Minimale s-t-Schnitt ist mit den zwischen $S_1, \cdots, S_k$ eingefügten Knoten vollständig erfasst. Für die Knoten a und b gilt entweder $\lambda(a,b) = \lambda(s,t)$ oder $\lambda(a,b) < \lambda(s,t)$. Im ersteren Fall kodiert jede der zwischen $S_i, \cdots, S_j$ eingefügten neuen T-Kanten einen Minimalen a-b-Schnitt X_r für $i \leq r < j$. Lemma 6.3 zeigt, dass auch durch spätere Schnitte keine größeren Kanten auf diesem Pfad eingefügt werden können.

Gilt $\lambda(a,b) < \lambda(s,t)$, ist der Minimale a-b-Schnitt in G'_{S_i} oder G'_{S_j} auffindbar und wird $X_{i-1} \cup \overline{X}_i$ oder $X_{j-1} \cup \overline{X}_j$ nicht kreuzen. Die ihn repräsentierende Kante wird dann in einem vom gerade eingefügten Kantenpfad abweichenden Teilbaum eingefügt werden. Wiederum ist nach Lemma 6.3 gesichert, dass der s-t-Pfad auch zu diesem Zeitpunkt keine geringwertigere Kante enthält.

Es bleibt zu bemerken, dass die Minimalen a-b-Schnitte mit $\lambda(a,b) < \lambda(s,t)$, $a \in C_i$, $b \in C_j$, $i < j$ und $i \neq 1$ oder $j \neq k$ die Motivation für die Relaxierung des Gomory-Hu-Baums darstellen. Werden sie in Phase 2 des Algorithmus aufgefunden, so lassen sie sich unter Umständen nur noch mit Einsatz Leerer Knoten repräsentieren. Abschnitt 6.3 widmet sich einer ausführlichen Beschreibung der Problematik.

6.2. Globalität Minimaler Schnitte mit kontrahierten Schnittpartnern

Als Nächstes soll noch einmal verdeutlicht werden, inwieweit Minimale Schnitte in einem Kontraktionsgraphen G', welche mindestens einen kontrahierten Schnittpartner besitzen, zu Aussagen über Minimale Schnitte in G führen können. Diese Ergebnisse gewährleisten die Verwendbarkeit der in Phase 2 berechneten Schnitte, an welchen grundsätzlich kontrahierte Knoten beteiligt sind. Außerdem sind sie beim Einsatz Maximaler Adjazenzordnungen interessant, da hier nicht garantiert werden kann, dass der gefundene Maximalfluss nur zwischen unkontrahierten Knoten verläuft.

Lemma 6.5
Ein Minimaler x-$[s]$-Schnitt in einem Kontraktionsgraphen G', zwischen einem Originalknoten $x \in V$ und einem Kontraktionsknoten $[s]$, welcher genau die s-Schnittmenge eines Minimalen s-t-Schnitts enthält, entspricht einem Minimalen s-x-Schnitt in G.

Beweis: Die im Kontraktionsknoten $[s]$ zusammengefassten Knoten sind eine Schnittmenge X_{i-1} mit $1 < i \leq k$ eines Maximalen s-t-Flusses. Da $s \in C_1$ und $x \in C_i$ gilt, existiert nach Lemma 6.4 ein den Maximalen s-x-Fluss begrenzender Schnitt, welcher X_{i-1} und $\overline{X}_i$ nicht kreuzt. Damit ist er in G' berechenbar; auch falls der Kontraktionsknotens $[t]$ für die eventuell vorkommende Knotenmenge $\overline{X}_i$ existiert (im Fall $i < k$). Jeder Minimale x-$[s]$-Schnitt in G' entspricht nun der günstigsten Möglichkeit X_{i-1} von x zu trennen und kann $\overline{X}_i$ nicht kreuzen.

□

Besitzt ein Maximaler x-$[s]$-Fluss in G' mindestens zwei Flusskomponenten, welche unkontrahierte Knoten aus V enthalten, so ist auch dieser Fluss *gültig*, denn es wurde ein bisher unbekannter Minimaler s-x-Schnitt in G gefunden. Der die unkontrahierten Knoten von G' repräsentierende Superknoten S in T kann also, gemäß den in Abschnitt 5.3.4 gezeigten Prinzipien, aufgespalten werden.

Lemma 6.6
Ist G'_{sa} ein Kontraktionsgraph mit den Kontraktionsknoten $[s]$ und $[a]$, wobei $[s]$ genau eine s-Schnittmenge aus der Flusskomponentenzerlegung eines Maximalen s-t-Flusses zusammenfasst und $[a]$ genau eine a-Schnittmenge aus der Flusskomponentenzerlegung eines Maximalen a-b-Flusses vetritt, so ist ein Minimaler $[s]$-$[a]$-Schnitt in G'_{as} ein Minimaler s-a-Schnitt in G.

Beweis: Jeder Minimale $[s]$-$[a]$-Schnitt in G'_{sa} trennt offensichtlich s von a. Kann gewährleistet werden, dass ein Minimaler s-a-Schnitt existiert, der keine der in $[s]$,$[a]$ und den eventuell vorhandenen Knoten $[t]$,$[b]$ jeweils zusammengefassten Originalknotenmengen kreuzt, so ist der Minimale $[s]$-$[a]$-Schnitt in G'_{sa} auch für s und a in G minimal.

Da die bekannten s-t- und a-b-Schnitte bereits jeweils s und a trennen, muss also $\lambda(s,a) \leq \min(\lambda(a,b), \lambda(s,t))$ gelten.

Gilt $\lambda(s,a) = \min(\lambda(a,b), \lambda(s,t))$ so entspricht die in $[s]$ oder die in $[a]$ zusammengefasste Knotenmenge einem Minimalen s-a-Schnitt in G. Dieser ist jeweils kreuzungsfrei zu den von $[a]$, $[s]$ und den eventuell vorhandenen Kontraktionsknoten $[b]$,$[t]$ vertetenen Knotenmengen. Damit ist er in G'_{as} als Minimaler $[s]$-$[a]$-Schnitt bestimmbar.

Im Weiteren gelte $\lambda(s,a) < \min(\lambda(a,b), \lambda(s,t))$.

Um die Kontraktionsknoten $[s]$ und $[a]$ zu bilden sind bereits die Flusskomponentenzerlegungen für den Maximalen s-t-Fluss und den Maximalen a-b-Fluss durchgeführt worden.

Wird o.B.d.A. zuerst der Maximale s-t-Fluss in G gebildet und in seine Komponenten zerlegt, so gibt es eine Komponente C_s mit $a \in C_s$ und $s \notin C_s$ sowie die benachbarte s-Schnittmenge X_s, und eventuell die t-Schnittmenge $\overline{X}_s = V \setminus (X_s \cup C_s)$. Die Knoten s und a müssen getrennt worden sein, denn sonst würde s in G'_{sa} unkontrahiert auftreten. Es folgt nach Lemma 6.4, dass ein Minimaler a-s-Schnitt $A \subseteq C_s$ mit $a \in A$ existiert. Dieser kreuzt weder X_s noch $\overline{X}_s$.

Wird der Kontraktionsgraph für den Superknoten zu C_s gebildet, tritt darin der Kontraktionsknoten $[s]$ für die Menge X_s und - wenn $t \notin C_a$ galt - $[t]$ für $\overline{X}_s$ auf. Wird der Maximale a-b-Fluss in diesem Kontraktionsgraphen bestimmt und in seine Flusskomponenten zerlegt, gibt es eine Komponente C_a mit $[s] \in C_a$ und $a \notin C_a$ sowie die benachbarte a-Schnittmenge X_a, und eventuell die b-Schnittmenge $\overline{X}_a = V \setminus (X_a \cup C_a)$. Die Trennung von $[s]$ und a folgt, da a sonst in G'_{as} unkontrahiert auftreten würde. Nach Lemma 6.4 gibt es den Minimalen a-$[s]$-Schnitt $B \subseteq C_s$ mit $[s] \in B$. Dieser entspricht gemäß Lemma 6.5 einem in G gültigen Minimalen a-s-Schnitt.

Er kreuzt weder X_a, die Knotenmenge in $[a]$, noch $\overline{X}_a$, die Knotenmenge im eventuell in G'_{sa} auftretenden $[b]$. Zusätzlich ist die Kreuzung der Knotenmengen in $[s]$ und $[t]$ durch deren Kontraktion ausgeschlossen.

Der Schnitt B ist also offensichtlich in G'_{sa} berechenbar und trennt dort $[s]$ und $[a]$ mit Minimalen Kosten.

□

Somit können also auch Minimalschnitte, welche aus gültigen Maximalen Flüssen zwischen Kontraktionsknoten ableitbar sind, zur Zerlegung von Superknoten in T verwendet werden.

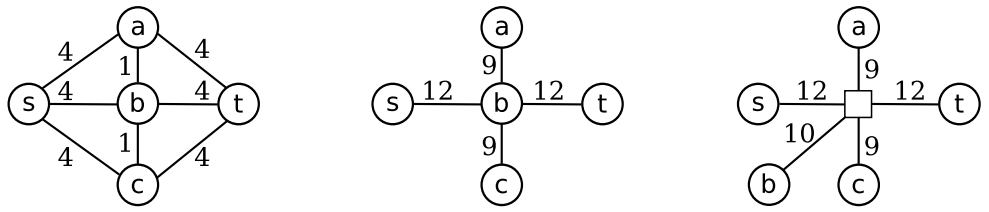
Zusätzlich ist das Schnittergebnis zwischen zwei Kontraktionsknoten für die Gesamtberechnung auch dann von Nutzen, falls kein Paar unkontrahierter Knoten getrennt wird, jedoch neue Erkenntnisse über die Schnittbeziehung der beteiligten Knoten folgern. So können auf diese Weise aufgefundene Blattschnitte eines Kontraktionsknoten direkt als Kante in L aufgenommen werden. Dies kann die Durchführung eines zusätzlichen Schnittes in Algorithmusphase 2 einsparen.

Eine gezielte Berechnung von Flüssen zwischen Kontraktionsknoten ist jedoch besser auf Phase 2 zu verschieben, da sie, im Gegensatz zu gezielten Flussberechnungen zwischen unkontrahierten Knoten eines Kontraktionsgraphen G', seltener zur Aufspaltung von Superknoten und zur Verkleinerung der Konstraktionsgraphen für die folgenden Schnitte führen.

6.3. Notwendigkeit weiterer Schnittbetrachtungen nach Phase 1

Die vorgestellte Technik der Zerlegung der Knotenmenge eines Kontraktionsgraphen G' in Flusskomponenten, führt gegenüber der klassischen Gomory-Hu-Zerlegung in ausschließlich zwei Schnittmengen pro Schnittberechnung, zu einer schnelleren Verkleinerung der in den Superknoten von T verwalteten Knotenmengen.

Phase 1 des Algorithmus ist abgeschlossen, sobald jeder Superknoten S des bisher konstruierten Baums T nur noch einen einzigen Knoten $v \in V$ repräsentiert. Die



(a) Beispielgraph

(b) T nach Phase 1 mit Flüssen s-t und a-c

(c) Fertiger relaxierter Gomory-Hu-Baum

Abbildung 22: Notwendigkeit weiterer Berechnungen nach Phase 1 (Fehlrepräsentation von b-t-/b-s-Schnitten)

Kontraktionsgraphen zu diesen einelementigen Superknoten werden im Weiteren auch als *elementare Kontraktionsgraphen* bezeichnet.

Der zu diesem Zeitpunkt bestehende Baum T, muss noch nicht dem Relaxierten Gomory-Hu-Baum entsprechen, da durch die bisher durchgeführten Schnittberechnungen unter Umständen noch nicht sämtliche Minimalschnittbeziehungen der Knotenpaare aus V erfasst sind.

Wird eine s-t-Flusskomponentenzerlegung $\{C_1, C_2, \cdots, C_k\}$ in mehr als 2 Komponenten betrachtet, so handelt es sich bei den noch nicht in T repräsentierten Schnitten um solche zwischen Knoten aus verschiedenen Komponenten C_i und C_j, von denen höchstens eine einen Flussendpunkt enthält. Für die weitere Argumentation sei C_i eine 'Mittelkomponente' mit $1 < i < k$.

Mit dem zur Zerlegung führenden Maximalen s-t-Fluss, sind die Maximalflusswerte der Knoten aus C_i zu den Knoten der anderen Komponenten keineswegs genau bestimmt. Vielmehr entspricht der Flusswert $\lambda(s,t)$ nur einer oberen Schranke für diese weiteren Flüsse. Sind deren Flusswerte sämtlich kleiner als $\lambda(s,t)$, so sind sie nach Abschluss von Phase 1 in T noch fehlrepräsentiert. In einem solchen Fall existiert ein Superknoten S für ein $v \in C_i$ an welchem zwei Teilbäume hängen, die genau die Knotenmengen X_{i-1} und $\overline{X}_i$ in sich vereinen. Beide sind über eine Kante des Werts $\lambda(s,t)$ angebunden, welche keinem Minimalschnitt zwischen v und einem Knoten dieser Teilbäume entspricht. Abbildung 22 zeigt ein entsprechendes Beispiel.

Die für eine solche Fehlrepräsentation in Frage kommenden Schnitte, sind in der Argumentation des Abschnitts 6.1.2 die Minimalen a-b-Schnitte mit $\lambda(a,b) < \lambda(s,t)$, $a \in C_i$, $b \in C_j$, $i < j$ und $i \neq 1$ oder $j \neq k$. Dort wurde auch gezeigt, dass für jedes solche Knotenpaar weiterhin ein in G gültiger Minimaler Schnitt in mindestens einem der aus den Knoten von T bildbaren Kontraktionsgraphen berechnet werden kann.

Um die noch nicht in T repräsentierten Schnitte aufzufinden, ist es hinreichend die noch unbekannten Maximalflussverhältnisse zwischen jedem Knoten der n elementaren Kontraktionsgraphen festzustellen. Da sämtliche Informationen der bereits durchgeführten Schnitte im Blattschnittbeziehungsgraphen L gespeichert sind, lassen sich die ausstehenden Schnitte einfach identifizieren. Zusätzlich müssen die neu gewonnen In-

formationen derart im entstehenden Relaxierten Gomory-Hu-Baum repräsentiert werden, dass der Umbau die dort bereits vorhandenen Informationen nicht verfälscht.

6.3.1. Bekannte Eigenschaften der Schnittbeziehungen zwischen den Knoten eines Kontraktionsgraphen

Betrachtet man den Kontraktionsgraphen für einen beliebigen Superknoten, so treten darin verschiedene Schnittbeziehungen auf, welche bereits durch früher ausgeführte Schnittberechnungen bekannt sind und nicht aufwändig neu festgestellt werden müssen.

Lemma 6.7
Für jeden Kontraktionsknoten $[z]$, welcher durch einen Maximalen a-z-Fluss in G entstanden ist und der den Knoten z enthält, existiert mindestens ein Knoten y (beliebigen Typs) in G', für welchen der $[z]$-Blattschnitt dem Minimalen y-$[z]$-Schnitt in G' entspricht.

Beweis: Handelt es sich beim außerhalb von $[z]$ liegenden Flussendpunkt a um einen unkontrahierten Knoten in G', so ist die Behauptung mit $y = a$ trivialerweise korrekt, da $[z]$ die z-Schnittmenge eines Minimalen a-z-Schnittes in sich vereint.

Ist der Flussendpunkt a in G' nur noch in Form eines Kontraktionsknotens $[a]$ vertreten, so kann es keinen günstigeren Schnittverlauf als den $[z]$-Blattschnitt zwischen $[a]$ und $[z]$ geben, da dieser sonst auch ein kleinerer Minimaler a-z-Schnitt gewesen wäre. Somit gilt $y = [a]$ und der $[z]$-Blattschnitt ist ein Minimaler y-$[z]$-Schnitt in G'.

□

Im Spezialfall durch den selben Maximalen Fluss gebildeter Kontraktionsknoten, stellen natürlich beide Blattschnitte Minimale Schnitte in einem gemeinsamen Kontraktionsgraphen dar.

Für die Kontraktionsgraphen zu Superknoten welche im Baum T als Blatt auftreten, sind die Schnittbeziehungen bereits vollständig bekannt. Alle Minimalen Schnitte sind jeweils Blattschnitte, was aufgrund ihrer Knotenzahl von zwei allerdings auch nicht weiter verwunderlich ist.

Besonders interessant ist der zweite Teil des obigen Beweises, sobald er auf bereits mehrfach kontrahierten Graphenversionen betrachtet wird. So ist in einem Kontraktionsgraphen G' zwischen zwei durch einen s-t-Fluss entstandenen Kontraktionsknoten $[s]$ und $[t]$ bereits ein Maximaler Fluss bekannt, dessen Engstellen in den Blattschnitten von $[s]$ und $[t]$ liegen. Werden diese nun durch einen höherwertigen, nichttrivialen Schnittverlauf zwischen einem anderen Knotenpaar getrennt, so behält jeder von ihnen in seinem erneut kontrahierten Graphen G'' seine Blattschnittbeziehung zum neu entstandenen Kontraktionsknoten, welcher sowohl den ehemaligen Flusspartner, als auch weitere Knoten enthält. Dies bildet die Grundlage sämtlicher Adaptionen *bestehender Kanten* des Graphen L.

Es bleibt, die Beziehungen zwischen den verbliebenen Knotenpaarungen zu untersuchen. Dabei hilft das folgende Lemma:

Lemma 6.8
In einem Kontraktionsgraphen G' mit zwei durch den selben Maximalen s-t-Fluss entstandenen Kontraktionsknoten $[s]$ und $[t]$ gilt für jeden weiteren Knoten b aus G' $\lambda(b,[s]) = \lambda(b,[t])$.

Beweis: Jeder Minimale b-$[t]$-Schnitt in G' mit $\lambda(b,[t]) < \lambda(s,t)$ separiert ebenfalls b und $[s]$, da der bekannte s-t-Schnitt sonst nicht minimal gewesen wäre. Gleiches gilt für jeden Minimalen b-$[s]$-Schnitt mit $\lambda(b,[s]) < \lambda(s,t)$, der ebenfalls b und $[s]$ trennt. Somit ist in diesem Fall jeder Minimale b-$[s]$-Schnitt auch ein Minimaler b-$[t]$-Schnitt und es folgt $\lambda(b,[t]) = \lambda(b,[t]) < \lambda(s,t)$.

Minimale Schnittwerte größer als $\lambda(s,t)$ können nicht eintreten, da mit den $[s]$- und $[t]$-Blattschnitten, jeweils ein Schnitt mit Wert $\lambda(s,t)$ bekannt ist. Nach Ausschluss aller anderen Fälle verbleibenden nur noch die Schnitte mit $\lambda(b,[s]) = \lambda(s,t)$ und $\lambda(b,[s]) = \lambda(s,t)$ womit auch hier offensichtlich $\lambda(b,[s]) = \lambda(b,[t])$ gilt.

□

Korollar 6.1
In einem Kontraktionsgraphen G' gilt für einen Knoten b und die Kontraktionsknoten $[s]$ und $[t]$ eines bekannten Maximalen s-t-Flusses:

- Gilt $\lambda(b,[s]) = \lambda(b,[t]) < \lambda(s,t)$, sind die Flusskomponenten eines Maximalen b-$[s]$- bzw. b-$[t]$-Flusses in G' identisch.
- Gilt $\lambda(b,[s]) = \lambda(b,[t]) = \lambda(s,t)$, so sind mit den Blattschnitten von $[s]$ und $[t]$ bereits für beide b involvierenden Minimalen Schnitte Schnittverläufe bekannt.

Es folgt also, dass sich aus der Berechnung *eines* Minimalen Schnittes zwischen dem Knoten b und einem Kontraktionsknoten $[s]$ im Kontraktionsgraphen, direkt der Schnittverlauf und -wert des Minimalen Schnittes zwischen b und dem mit $[s]$ durch einen gemeinsamen Maximalen Fluss entstandenen Kontraktionsknoten $[t]$ ergibt.

6.3.2. Auswahl expliziter Schnittberechnungen in Kontraktionsgraphen

Es bleibt zu klären, wie die noch ausstehenden Schnittberechnungen nun identifiziert werden können. Während des Verlaufs von Phase 1 und ebenso im klassischen Gomory-Hu-Algorithmus ist dies trivial, da jeder zwei unkontrahierte Knoten in einem Kontraktionsgraphen G' trennende Schnitt bisher offensichtlich noch nicht in T festgehalten worden ist.

In den Kontraktionsgraphen der Phase 2 tritt jedoch maximal noch ein einziger unkontrahierter Knoten auf, so dass ein alternativer Weg für ihre Bestimmung gefunden werden muss.

Da laut Abschnitt 6.1 alle noch ausstehenden Schnitte in den Kontraktionsgraphen zum Baum T bestimmbar sind, würde die Lösung des ALL-PAIRS-MINIMUM-CUT-Problems für jeden Kontraktionsgraphen ebenso das Vorhandensein aller für T notwendigen Schnittinformationen des Originalgraphen G bedeuten.

Alle bereits durchgeführten und in T verzeichneten Minimalschnitte sind ebenso in den Kontraktionsgraphen gültig (siehe Lemma 6.7). Der Graph L zeigt zwischen welchen Knoten sie dort verlaufen. Umgekehrt, sind die in T noch nicht repräsentierten Schnitte offensichtlich auch in den Kontraktionsgraphen noch unbekannt.

Für den Kontraktionsgraphen G'_S zu einem Knoten S aus T sind die bekannten Schnittbeziehungen, wie schon in Abschnitt 5.3.2 erläutert, in Form gerichteter Kanten in einem Teilgraphen von L gespeichert. Dieser wird von der dem Superknoten S zugehörigen Originalknotenmenge und den Kantenknoten der zu S inzidenten T-Kanten induziert. Für jede dieser T-Kanten existiert in G'_S ein Kontraktionsknoten, welcher die Knotenmengen des durch sie von S ausgehenden Teilbaums zusammenfasst. Die zu S gehörenden Originalknoten treten in G'_S unkontrahiert auf.

Für die folgenden Betrachtungen soll der Graph $L'_{G'_S}$ verwendet werden, welcher exakt diesem Teilgraphen entspricht, wobei dessen Kantenknoten bereits durch die zugehörigen Kontraktionsknoten in G'_S ersetzt wurden, um die Schilderung zu verkürzen. So verfügt $L'_{G'_S}$ über die selbe Knotenmenge wie G'_S.

Der in einer Kante von $L'_{G'_S}$ enthaltene Informationswert bleibt gegenüber L unverändert und wird durch die Abbildung direkt auf G'_S übertragen: Ist $G'_S = (V', E')$ und existiert für $x, y \in V'$ eine Kante $x \to y$ in $L'_{G'_S}$, so entspricht $\{x\}$ in G'_S einem Minimalen x-y-Schnitt. Dieses Wissen wurde durch die vorausgegangenen Flusskomponentenzerlegungen erschlossen.

Aus diesen bekannten Blattschnittbeziehungen lassen sich nun Informationen über einen möglichen Gomory-Hu-Baum für G'_S ableiten und ebenso die für dessen Erstellung noch notwendigen Schnittberechnungen feststellen:

Lemma 6.9
Existiert in $L'_{G'_S}$ ein Wald wurzelknotengerichteter Bäume, so gibt es einen Gomory-Hu-Baum für G'_S, in dem sämtliche Nicht-Wurzelknoten dieser Bäume als Blatt auftreten.

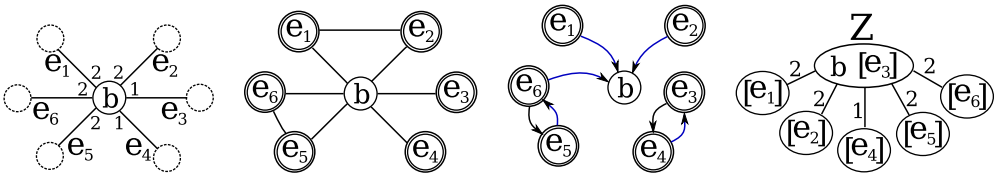
Beweis: Um den Rahmen dieser Arbeit zu begrenzen, soll der Beweis hier nur skizziert werden.

Es werden nacheinander für jeden der gewählten Bäume in $L'_{G'_S}$ die mit seinen Kanten assoziierten Schnitte, in absteigender Reihenfolge ihrer Entfernung zur Baumwurzel, auf G'_S ausgeführt. Von jedem Schnitt wird, wie ja bereits im Vorfeld bekannt ist, jeweils nur ein einzelner Knoten vom Rest getrennt. Durch die Kreisfreiheit des Waldes und die Ausführungsreihenfolge ist gewährleistet, dass beide Schnittpartner grundsätzlich der Menge von Knoten angehören, welche noch nicht als Blatt abgeschert wurden.

Eine solche Schnittfolge ist gemäß der klassischen Gomory-Hu Methodik geeignet, den Gomory-Hu-Baum für G'_S aufzubauen. Nach der Ausführung aller durch Baumkanten repräsentierten Schnitte, besitzt der in der Entstehung begriffene Gomory-Hu-Baum einen zentralen Superknoten Z, welcher aus den Wurzelknoten des $L'_{G'_S}$-Waldes besteht und an diesem sind sämtliche Nicht-Wurzelknoten direkt als Blätter positioniert. Jeder weitere Verlauf des Gomory-Hu-Algorithmus kann deren Blattstatus nicht

mehr ändern, so dass dieser auch im schließlich entstehenden Gomory-Hu-Baum erhalten bleiben muss.

□



(a) Ausschnitt aus T (b) G'_S (c) $L'_{G'_S}$ (d) Schnittbaum für G'_S

Abbildung 23: Schnittbaum des klassischen Gomory-Hu-Algorithmus nach Ausführung einen Spannwald in $L'_{G'_S}$ bildender Schnitte ($[e_5]$-$[e_6]$, $[e_6]$-b, $[e_1]$-b,$[e_2]$-b, $[e_4]$-$[e_3]$)

Existiert in $L'_{G'_S}$ ein wurzelgerichteter *Spannbaum*, so entspricht der Gomory-Hu-Baum von G'_S einem Stern, mit dem Wurzelknoten als Mittelpunkt. Die Flusswerte aller Pfade in diesem, decken sich mit dem Layout in T. Die Flüsse zwischen Kontraktionsknoten aus G'_S entsprechen denen zwischen den jeweiligen von S abgehenden Teilbäumen. Die Flüsse zwischen Kontraktionsknoten und Originalknoten aus V entsprechen dem Wert der zwischen dem zum Kontraktionsknoten gehörenden Teilbaum und dem T-Knoten S verlaufenden T-Kante. Sämtliche in G'_S verlaufenden Minimalschnittbeziehungen der Knoten aus G sind bereits in T erfasst und für G'_S muss keine weitere Arbeit mehr verrichtet werden.

Ein ungünstigerer Fall tritt ein, falls G'_S zwei Kontraktionsknoten $[s]$ und $[t]$ enthält, welche durch den gleichen Maximalen s-t-Fluss entstanden sind. Ihre Blattschnittbeziehungen bilden in $L'_{G'_S}$ einen minimalen Kreis. Da durch die Flusskomponentenzerlegungen der Phase 1 minimal $|V'|-1$ Blattschnitte bekannt sind, reicht die Kantenzahl in diesem Fall nicht für einen Spannbaum aus. Mit der Spannwaldberechnung ergeben sich jedoch direkt Kandidaten für passende, unbekannte Schnitte, deren Ergebnis die Baumanzahl im Spannwald sinken lässt:

Lemma 6.10
Die zur Bestimmung aller Minimalschnittverhältnisse in G'_S ausstehenden Schnittberechnungen verlaufen zwischen den Wurzelknoten eines Spannwaldes von $L'_{G'_S}$. Besitzt der Spannwald r Wurzelknoten, so sind $(r-1)$ Schnittberechnungen ausreichend.

Beweis: Dies wird wiederum deutlich, indem der mittels der klassischen Gomory-Hu Methodik partiell konstruierbare Gomory-Hu-Baum betrachtet wird, welcher durch

die Ausführung der den Spannwald in $L'_{G'_S}$ konstituierenden Schnitte gebildet werden kann.

Die zur Bildung des vollständigen Gomory-Hu-Baums noch ausstehenden Schnitte sind solche, welche garantiert Knoten des zentralen, alle Wurzelknoten enthaltenden, Superknotens Z trennen. Diesen Anspruch erfüllen gezielte Minimalschnitte zwischen diesen Knoten aus Z trivialerweise. Bei Weiterverfolgung der Gomory-Hu-Methode beträgt die Zahl der notwendigen Schnitte dabei offensichtlich $(|Z| - 1) = (r - 1)$. Da die Bildung des Spannwaldes über $L'_{G'_S}$ die Zahl der Bäume und damit die Zahl der Wurzelknoten minimiert, minimiert das Vorgehen auch die Zahl der durchgeführten Schnitte.

□

Der notwendige Spannwald in $L'_{G'_S}$ lässt sich dabei mit einer Tiefensuche in linearer Zeit berechnen.

Für jeden so zwischen zwei Knoten aus G'_S berechneten Minimalen Schnitt muss nun der Baum T angepasst werden um das gewonnene Ergebnis aufzunehmen. Der Umbau muss zudem sicherstellen, dass keine bereits vorhandenen Ergebnisse verfälscht werden und der gerade festgestellte Schnittverlauf in den zukünftig entstehenden Kontraktionsgraphen ebenso als Blattschnitt auftritt.

6.3.3. Korrekte Umsetzung gewonnener Schnittinformation im zu konstruierenden Relaxierten Gomory-Hu-Baum

Mit den gezeigten Mitteln lassen sich also für jeden Kontraktionsgraphen G' die noch benötigten s-t-Schnitte feststellen, um den Gomory-Hu-Baum für G' zu bilden. Von Interesse ist jedoch die Fertigstellung des globalen Baums T zum relaxierten Gomory-Hu-Baum für G. Dieser repräsentiert alle bekannten, in L gespeicherten, und zur Bestimmung der noch benötigten Schnitte eingesetzten globalen Schnittverläufe bereits korrekt.

Unter möglichst minimalen Änderungen muss die nun neu erhaltene Information einbezogen werden, ohne die bereits darin wiedergespiegelten Schnitte zu verändern.

Würde es sich bei T um einen klassischen Gomory-Hu-Baum für G handeln, wären umfangreichere Architekturänderungen notwendig, welche Wissen über sämtliche Schnittverhältnisse in G vorraussetzen. Stattdessen ist die Modifikation in einem relaxierten Gomory-Hu-Baum durch den Einsatz Leerer Knoten allein unter Einbezug einer lokalen Umgebung in T möglich.

Die Kontraktionsknoten für einen Kontraktionsgraphen G'_S, welcher auf Basis eines T-Knotens S gebildet wurde, repräsentieren die Originalknotenmengen der in T sternförmig von S abgehenden Teilbäume. Direkt im Anschluss an Phase 1 des Algorithmus besteht T noch vollständig aus Superknoten. Später kann S ebenso ein Leerer Knoten sein.

In den weiteren Ausführungen wird zur Anschauung wiederum $L'_{G'_S}$ als auf G'_S übertragener, *relevanter Teilgraph von* L verwendet. Sämtliche Modifikationen werden vom Algorithmus natürlich auf dem Originalgraphen L durchgeführt.

Wird vorausgesetzt, dass G'_S maximal einen Originalknoten b enthält, so sind für den Schnittverlauf zwischen Wurzelknoten eines Spannwalds aus $L'_{G'_S}$ verschiedene Ausprägungen möglich:

Behandlung neu entdeckter Blattschnitte

Zum Einen, kann für einen im Spannwald von $L'_{G'_S}$ als Baumwurzel fungierenden Kontraktionsknoten $[e_1]$ und einen zweiten Baumwurzelknoten b aus G'_S ein Minimaler $[e_1]$-b-Schnitt der Form $\{[e_1]\}$ gefunden werden. In diesem Fall ist keine Adaption von T notwendig.

Korrektheit: Die bereits vorhandene T-Kante zwischen dem in $[e_1]$ zusammengefassten Teilbaum und dem T-Knoten S trägt bereits den Wert $c_{G'_S}([e_1])$ und stellt sowohl den Schnittverlauf, als auch den Schnittwert vollständig dar. Die zusätzliche Information wird lediglich als gerichtete Kante in L aufgenommen und ermöglicht im auf G'_S abgebildeten Teilgraphen $L'_{G'_S}$ das Verschmelzen zweier Bäume des Spannwalds.

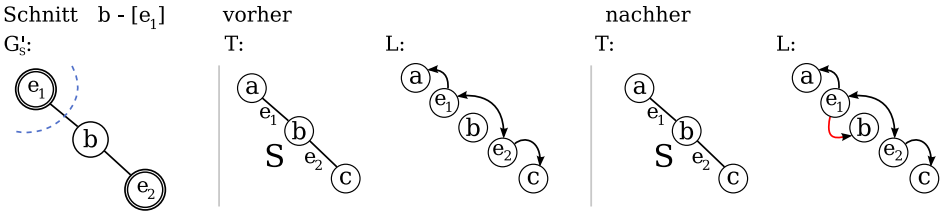


Abbildung 24: Umsetzung von Kontraktionsknoten-Blattschnitten

Anders liegt der Fall, falls für den unkontrahierten Knoten b mit $\{b\}$ ein Minimaler Schnitt zu einem anderen Knoten $[e_1]$ in G'_S gefunden wird. Da S vor dieser Feststellung als Mittelpunkt einer Sternstruktur im Baum T fungiert, der alle restlichen Knoten des Graphen in einer Schnittmenge zusammenfassende Schnitt $(\{b\}, V \setminus \{b\})$ so aber nicht, wie gefordert, durch eine einzige Kante repräsentiert werden kann, muss T adaptiert werden.

Ein korrektes Vorgehen besteht im Einfügen eines Leeren Knotens O an Stelle von S, welches als weiteres Blatt an O angefügt wird. Die verbindende Kante (in Abbildung 25 die Kante e_3) trägt den Wert des b-Blattschnitts.

Korrektheit: Im resultierenden Baum sind nur solche Pfade von und zu S, dem Superknoten von b, verändert worden. Da diese nur um die neue Kante e_3 verlängert wurden, sind durch bestehende Kanten kodierte, eventuell vorhandene kleinere Schnitte zwischen b und anderen Knoten in G weiterhin auf diesen Pfaden minimal.

Durch den Umbau, verändert sich der für den Zentrumsknoten der T-Verzweigung (vormals S nun O) bildbare Kontraktionsgraph. In diesem tritt b fortan nur noch im Kontraktionsknoten $[b]$ kontrahiert auf (welcher jedoch ausschließlich b enthält).

Dessen soeben festgestellte Blattschnittbeziehung wird durch eine neue Kante $e_3 \rightarrow e_1$ in L festgehalten. Sie verbindet die beiden Spannwaldwurzeln und führt zur Abnahme

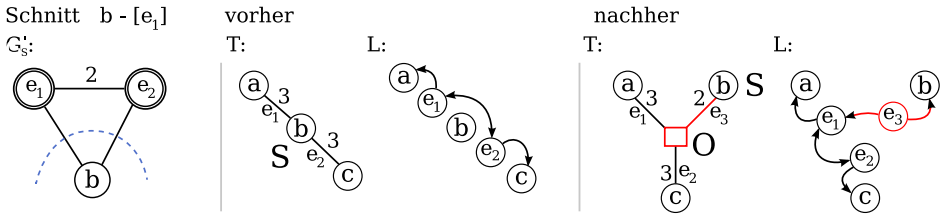


Abbildung 25: Umsetzung von Blattschnitten des unkontrahierten Knotens

der Baumzahl des Spannwaldes in $L'_{G'_O}$ für den zukünftigen Kontraktionsgraphen G'_O. Ähnliches gilt für die Kante $e_3 \rightarrow b$, ist jedoch für die Berechnung unnötig, da der Knoten S nun in T ein Blatt ist und kein Kontraktionsgraph mehr für ihn gebildet wird.

In L vorher auf b verweisende Kanten, müssen nun auf den Kantenknoten der neu eingefügten T-Kante e_3 adaptiert werden, da in G'_O der Kontraktionsknoten $[b]$ den in ihn aufgenommenen Knoten b als Zielpunkt jeder vorher auf b verweisenden Blattschnittbeziehung beerbt (siehe Abschnitt 6.3.1).

Behandlung nicht-trivialer Schnittverläufe

Der komplizierteste Fall liegt vor, falls der gefundene Minimale Schnitt in G'_S für keinen der beteiligten Knoten einem Blattschnitt entspricht. Der T-Knoten S, bildet zu diesem Zeitpunkt die gemeinsame Wurzel aller von ihm ausgehenden Teilbäume. Dieses Verhältnis kann nun nicht weiter aufrecht erhalten werden, da *eine* Kante eingefügt werden soll, deren Entnahme zu einer Partition führt, deren Partitionsmengen beide mehr als einen der Teilbäume umfassen.

Als Lösung wird in T erneut ein Leerer Knoten O eingefügt, die Knoten S und T mit einer den Schnittwert tragenden Kante verbunden und den ehemals sämtlich von S ausgehenden Teilbäumen entweder S oder O als neuer Wurzelknoten zugeteilt.

Die Zuweisung erfolgt gemäß des Schnittverlaufs in G'_S. Dieser gliedert die Kontraktionsknoten von G'_S in zwei Mengen. Ist S kein Leerer Knoten, so enthält eine der

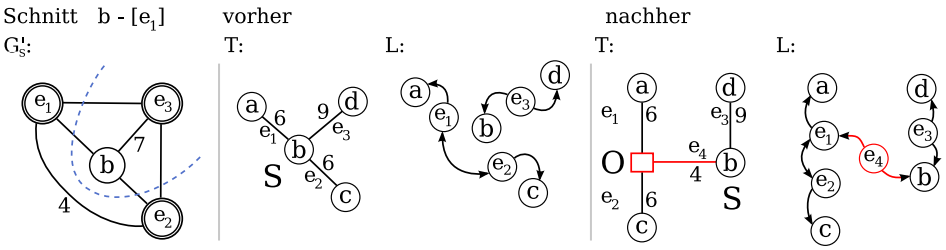


Abbildung 26: Umsetzung nicht-trivialer Schnittverläufe

Schnittmengen den zu S gehörenden unkontrahierten Knoten $b \in V$. In diesem Fall verbleiben die von den Kontraktionsknoten dieser Schnittmenge vertretenen T-Teilbäume an S, alle anderen werden an O angehangen.

Ist S ein Leerer Knoten, so ist frei wählbar welcher der beiden T-Knoten welche Schnittmenge vertreten soll. Um die notwendigen Adaptionen zu minimieren ist es vorteilhaft in diesem Fall die zur kleineren Schnittmenge gehörenden Teilbäume O zuzuweisen.

Korrektheit: Dieses Vorgehen spiegelt die neu gewonnene Schnittinformation korrekt im relaxierten Gomory-Hu-Baum wieder. Für jede der Kanten auf einem Pfad zwischen, durch den auf G projezierten Schnitt, nicht getrennten Knoten, sind Wert und über V induzierte Partition unverändert geblieben. Ausschließlich die Pfade zwischen getrennten Knoten wurden durch die neue T-Kante verlängert. Durch den Weiterbestand aller vorher vorhandenen Kanten, ist die weitere Repräsentation eventuell vorhandener kleinerer Schnitte zwischen Knoten beider Seiten gewährleistet.

Es bleibt zu klären, wie sich der Fund eines nicht-trivialen Schnittverlaufs auf die Auswahl weiterer Schnittpartner auswirkt. Durch das Einfügen des Leeren Knotens O und die Aufteilung der T-Teilbäume zu nun zwei verschiedenen (miteinander verbundenen) Wurzelknoten in T, existieren für S und O natürlich auch vom ursprünglichen G'_S abweichende Kontraktionsgraphen. Diese sollen G''_S und G''_O heißen.

In beiden ist die jeweils andere Schnittmenge von G'_S zu einem neuen Kontraktionsknoten $[O]$ bzw. $[S]$ zusammengefasst, dessen Teilbaum von der zwischen S und O eingefügten T-Kante eingeleitet wird. Jeder besitzt eine minimale Blattschnittbeziehung zum in seinem Kontraktiongraphen verbliebenen Endpunkt des nicht-trivialen Minimalen Schnitts. Zusätzlich beerbt er gemäß Abschnitt 6.3.1, alle in ihm kontrahierten Knoten in ihrer Funktion als Zielpunkt bereits bekannter Blattschnitte.

Die Übertragung dieser neuen Situation auf den Blattschnittbeziehungsgraphen L entspricht den bereits während jeder Superknotenteilung in Phase 1 angewandten Schritten:

Für die neu entstandene T-Kante wird ein Kantenknoten aufgenommen und von diesem in Richtung der Endpunkte des Minimalen Schnitts verweisende L-Kanten hinzugefügt. Schließlich wird für die bestehenden Kanten im für G'_S relevanten Teilgraphen von L überprüft, ob die Vertreter der beiden inzidenten Knoten durch den Minimalen Schnitt in G'_S getrennt wurden. In diesem Fall ersetzt der neue Kantenknoten das ursprüngliche Ziel der L-Kante.

Mit Hilfe der auf G''_S und G''_O übertragenen Teilgraphen $L'_{G''_S}$ und $L'_{G''_O}$ von L können die weiter ausstehenden Schnittberechnungen in beiden neuen Kontraktionsgraphen identifiziert werden. Die dabei auftretenden Szenarien und deren Behandlung decken sich mit den in diesem Abschnitt bereits vorgestellten Fällen für G'_S.

Zu bemerken bleibt noch, dass für jeden in Algorithmusphase 2 notwendigen, explizit berechneten Minimalen s-t-Schnitt, maximal ein weiterer Leerer Knoten und eine weitere Kante in den relaxierten Gomory-Hu-Baum eingefügt werden müssen.

6.4. Gesamtzahl notwendiger Schnittberechnungen

Offen ist noch immer die Frage, wie viele Schnitte nun in Phase 2 noch explizit bestimmt werden müssen und zu welcher Gesamtanzahl von Maximalflussberechnungen sich dies über beide Algorithmusphasen summiert.

Um dies zu analysieren ist u.A. folgendes Lemma hilfreich:

Lemma 6.11
Wird während oder direkt nach Abschluss von Phase 1 des vorgestellten Algorithmus, der Kontraktionsgraph G'_S für einen beliebigen Knoten S aus T gebildet und der für G'_S relevante - durch die S zugeordneten Koten aus V und die Kantenknoten der zu S inzidenten T-Kanten induzierte - Teilgraph von L analysiert, so existieren dort keine Kreise mit mehr als 2 Kanten.

Beweis: Zu Beginn der Phase 1 ist L noch vollständig kantenlos und das Lemma gilt offensichtlich.

Mit jeder Flusskomponentenzerlegung werden nun die folgenden Schritte ausgeführt:

1. Es werden neue Kanten in T eingefügt und somit neue Kantenknoten in L aufgenommen.
2. Bestehende L-Kanten, deren Quelle und Ziel durch die Zerlegung getrennt wurden, werden modifiziert. Neues Ziel ist einer der *neu* eingefügten Kantenknoten.
3. Um die im Fluss gefundenen Schnitte zu repräsentieren, werden von den neuen Kantenknoten ausgehende Kanten in L eingefügt. Sie können verschiedene Zielknoten besitzen:
   a) Einen anderen *neuen* Kantenknoten, dessen Kante in T zum selben Superknoten inzident ist. Je inzidentem Superknoten kann es höchstens einen solchen Partner geben. Gleichzeitig gibt es von dort eine weitere in die Gegenrichtung weisende Kante, so dass ein Minimalkreis entsteht.
   b) Einem Endpunkt $s \in V$ des Maximalen Flusses (in Phase 1 werden ausschließlich Flüsse zwischen unkontrahierten Knoten berechnet). Die Knoten der Menge V besitzen keine Auswärtskanten in L.

Da maximal zwei der neuen T-Kanten zu einem Knoten S inzident seien können, kann zwischen den Kantenknoten des für G'_S relevanten Teilgraphen durch neu eingefügte Kanten höchstens ein Minimalkreis erzeugt werden (Punkt 3a). Neue L-Kanten zu Knoten aus V können keinen Kreis schließen, da von ihrem Zielpunkt keine weiteren Kanten ausgehen (Punkt 3b).

Schließlich führen modifizierte L-Kanten ausschließlich von bestehenden zu neuen Kantenknoten (Punkt 2), es werden jedoch niemals Kanten von neuen zu bestehenden Kantenknoten eingefügt. Somit kann sich auch hier kein Kreis schließen.

□

Nun lässt sich folgende Aussage zur maximalen Anzahl notwendiger Berechnungen treffen:

Lemma 6.12
Mit der vorgestellten Methodik sind zur Berechnung aller Minimalschnittbeziehungen zwischen Knoten des Graphen G höchstens $(n-1)$ Maximalflussberechnungen notwendig.

Beweis: Prinzipiell ist die Gesamtberechnung abgeschlossen, sobald jeder Knoten S in T nur noch maximal einen Originalknoten aus V vertritt und in den bekannten Blattschnittbeziehungen des Kontraktionsgraphen zu S ein vollständiger Spannbaum identifizierbar ist.

Die vollständige Aufspaltung der Knotenmenge V geschieht in Phase 1 mittels $1 \leq z \leq (n-1)$ Maximalflussberechnungen. Dabei muss es $(n-1)-z$ *Mittelkomponenten* gegeben haben, welche weder Quell- noch Zielknoten des Flusses beinhalteten.

Sind während Phase 1 des Algorithmus Maximalflussberechnungen ausschließlich zwischen Originalknoten aus V durchgeführt worden, so war die Gültigkeit dieser Zerlegungen grundsätzlich garantiert.

Wird für einen Knoten S aus Baum T der Kontraktionsgraph $G'_S = (V', E')$ gebildet und die in diesem bereits bekannten Blattschnittbeziehungen mittels $L'_{G'_S}$ analysiert, so verfügt jeder Kontraktionsknoten über *genau eine* auf einen anderen Knoten des Kontraktionsgraphen verweisende Blattschnittbeziehung.

In diesen Beziehungen bestehen nur für sehr bestimmte Kontraktionsknoten Kreise. Dabei handelt es sich um Kontraktionsknoten für von S ausgehende Teilbäume von T, deren einleitende T-Kanten durch die gleiche Flusskomponentenzerlegung entstanden sind. Es muss sich also bei allen nicht in diesen beiden Kontraktionsknoten zusammengefassten Originalknoten von V um eine Mittelkomponente der entsprechenden Flusskomponentenzerlegung gehandelt haben.

Gemäß Lemma 6.11 kann es direkt im Anschluss an Phase 1 in einem Kontraktionsgraphen keine anderen als solche, durch die Bildung von Mittelkomponenten hervorgerufene, minimalen Kreise in den Blattschnittbeziehungen geben.

Existiert zu Beginn von Phase 2 in den Blattschnittbeziehungen von G'_S keine solche Minimalkreisbeziehung, so muss ein Spannbaum in $L'_{G'_S}$ vorhanden sein, da G'_S nur noch maximal einen Originalknoten enthält und für sämtliche restlichen Knoten (es handelt sich nur noch um Kontraktionsknoten) jeweils mindestens eine Auswärtskante in $L'_{G'_S}$ bekannt ist.

Je existierendem Minimalkreis erhöht sich die Zahl der Bäume im Spannwald von $L'_{G'_S}$ um maximal einen, da die minimal zur Verfügung stehende Kantenzahl von $|V'|-1$ in diesem Fall nicht ausreichen kann um Zusammenhang herzustellen.

Wird angenommen, dass nach Abschluss von Phase 1 sämtliche gemeinsam entstandenen und bei ihrer Entstehung zu einem gemeinsamen Superknoten inzidenten T-Kantenpaare weiterhin zueinander adjazent sind, gibt es je gebildeter Mittelkomponenten einen Kreis in den Blattschnittbeziehungen eines Kontraktionsgraphen. Insgesamt gibt es so also maximal $((n-1)-z)$ Kreise. In Phase 2 können keine weiteren Mittelkomponenten entstehen, da hier auf Mehrfachzerlegung verzichtet wird.

Jede Schnittberechnung in Phase 2 senkt nun die Gesamtzahl von Spannwaldbäumen um eins ab:

Ist aufgrund des Schnittverlaufs keine Adaption von T nötig, so wird einzig eine neue L-Kante eingefügt, welche zwei Baumwurzeln im Spannwald von $L'_{G'_S}$ verbindet.

Musste ein neuer Leerer Knoten O in T eingesetzt werden, existieren statt G'_S in Zukunft zwei verschiedene Kontraktionsgraphen G'_S und G'_O, in denen ein neuer Kontraktionsknoten die jeweils gegenüberliegende Schnittmenge zusammenfasst. Vom zur S und O verbindenden T-Kante e_n gehörenden Kantenknoten in L werden L-Kanten zu den vorher als Schnittpartner genutzten Wurzelknoten eingefügt. Damit sinkt die summierte Zahl der Bäume für die Spannwälder beider relevanten L-Teilgraphen $L'_{G'_S}$ und $L'_{G'_O}$ um eins gegenüber der Baumzahl im ursprünglichen $L'_{G'_S}$.

Phase 2 verlangt somit im Maximum $((n-1)-z)$ Maximalflussberechnungen, wodurch sich insgesamte eine maximale Ausführungszahl von $z+((n-1)-z)$ ergiebt.

□

Dies entspricht dem von Gomory und Hu festgestellten Wert. Gegenüber dem traditionellen Gomory-Hu Verfahren sind jedoch viele dieser Schnitte auf kleineren Probleminstanzen bestimmt worden.

Die Maximalzahl von $(n-1)$ Schnittberechnungen muss jedoch nicht immer vollständig ausgeschöpft werden.

Einsparung von Schnittoperationen Es seien e_i und e_{i+1} zwei durch die gleiche Maximalflussberechnung in Phase 1 entstandene T-Kanten, welche zum selben T-Knoten S inzident sind und sei G'_S der aus S bildbare Kontraktionsgraph. Werden die Kontraktionsknoten $[e_i]$ und $[e_{i+1}]$, welche die durch e_i bzw. e_{i+1} eingeleiteten Teilbäume in G'_S repräsentieren, durch eine gültige nachfolgende Maximalflussoperation in verschiedenen Flusskomponenten positioniert, so wird eine explizite Schnittberechnung zwischen $[e_i]$ oder $[e_{i+1}]$ und einem dritten Knoten eingespart (vgl. Abbildung 27).

In G'_S war für $[e_i]$ und $[e_{i+1}]$ in beide Richtungen eine minimale Blattschnittbeziehung (und somit ein Kreis in $L'_{G'_S}$) bekannt. Nach der nun anstehenden Adaption von T, insbesondere der Tatsache, dass e_i und e_{i+1} nun keinen gemeinsamen T-Knoten mehr besitzen, existiert kein Kontraktionsgraph mehr, in welchem durch beide Kanten ein Kontraktionsknoten gebildet wird.

Seien G'_i und G'_{i+1} die Kontraktionsgraphen, welche nach der Modifikation durch die beiden in T näher zusammenliegenden Endpunkte von e_i und e_{i+1} gebildet werden können. In G'_i existiert noch immer $[e_i]$ und in G'_{i+1} existiert weiter $[e_{i+1}]$. Der ehemalige Schnittpartner ist jedoch in beiden Graphen durch einen Kontraktionsknoten $[e_c]$ vertreten, der wesentlich mehr Knoten in sich vereint. Dessen Blattschnittbeziehung verweist garantiert auf einen dritten Knoten, welcher im $[e_i]$ und $[e_{i+1}]$ in G' trennenden Minimalen Schnitt als Endpunkt diente. Zusätzlich beerbt $[e_c]$ in G'_i und G'_{i+1} jeweils den ehamaligen Schnittpartner als Zielpunkt der Blattschnittrelation von $[e_i]$ und $[e_{i+1}]$ (siehe zweiter Teil des Beweises zu Lemma 6.7).

Bei der Einschätzung der Schnittbeziehungen für G'_i und G'_{i+1} können $[e_i]$ und $[e_{i+1}]$ nun beide als garantierte Blätter eines Gomory-Hu-Baums angenommen werden und

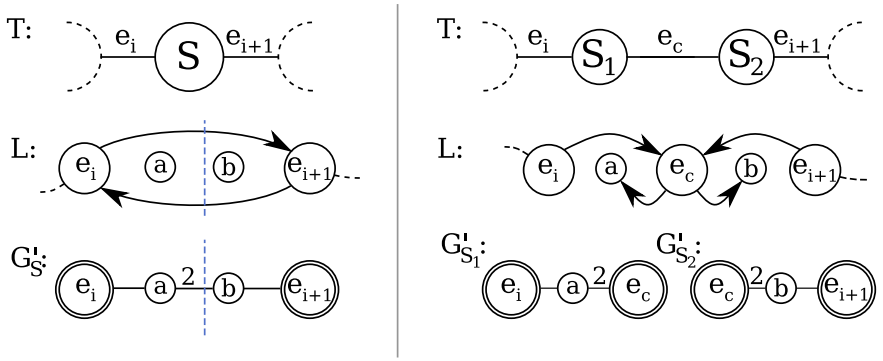


Abbildung 27: Einsparung einer Schnittoperation: a-b-Schnitt trennt e_i,e_{i+1} und ermöglicht gleichzeitige Nutzung beider vorher bekannten Blattschnittbeziehungen.

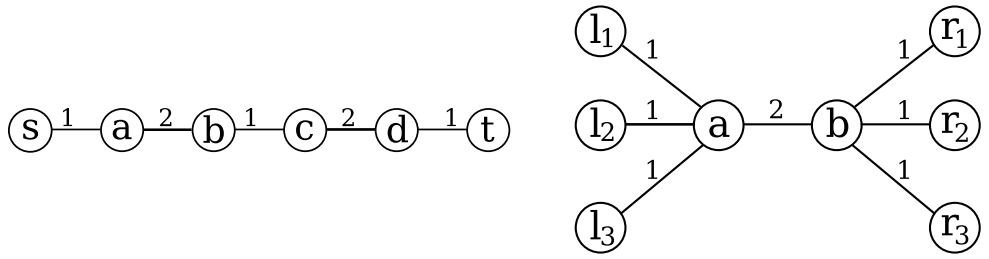
die sonst in einem gemeinsamen Kontraktionsgraphen zusätzlich notwendige Schnittberechnung kann entfallen.

Auftreten des Einsparungseffekts Ein solcher vorteilhafter Nebeneffekt, kann in beiden Phasen des Algorithmus auftreten. Während der Flusskomponentenzerlegungen, geschieht dies, sobald zwei durch die selbe Zerlegung entstandene Kontraktionsknoten in verschiedenen Komponenten eines nachfolgend berechneten, mindestens gleichteuren Flusses enden.

Während der Durchführung verbliebener Schnittoperationen in Phase 2, können zwei zusammengehörende Kontraktionsknoten durch einen nichttrivialen Schnitt zwischen einem anderen Knotenpaar des Kontraktionsgraphen getrennt werden. Künftige Berechnungen werden auch in diesem Fall auf zwei weiter kontrahierten Versionen durchgeführt, wobei die nun getrennten Kontraktionsknoten in beiden als Blatt eines Gomory-Hu-Baums für den Kontraktionsgraphen feststehen.

Lemma 6.13
Zur Feststellung von Minimalschnittwert und -verlauf zwischen jedem Knotenpaar des Graphen G, sind mit Hilfe der vorgestellten Methodik im Minimum $(1 + \lceil \frac{n-2}{2} \rceil)$ Maximalflussberechnungen ausreichend.

Beweis: Zwei durch die selbe Flusskomponentenzerlegung entstandene und zum selben Knoten inzidente Kanten des Baums T seien im Nachfolgenden als *Kantenpaar* bezeichnet. Die im Kontraktionsgraphen des gemeinsamen T-Knotens auftretenden Kontraktionsknoten, welche die mit den jeweiligen Kanten beginnenden Teilbäume von T in sich vereinen, seien ein *Paar von Kontraktionsknoten*. Eine Komponente einer Flusskomponentenzerlegung, welche weder Quell- noch Zielknoten enthält ist eine *Mittelkomponente*.



(a) Optimalfolge: s-t, a-b, c-d (b) Optimalfolge: l_i-r_i, a-b

Abbildung 28: Beispielgraphen optimaler Schnittfolgen in Phase 1

Während der Algorithmusphase 1 besteht das Einsparungspotential in Kantenpaaren, deren gemeinsamer Superknoten S noch zwei oder mehr Knoten aus V vertritt, da ein Maximalfluss zwischen diesen im Kontraktionsgraphen zu S auch die Kontraktionsknoten des Kantenpaars trennen könnte. In diesem Fall würde sich eine spätere Maximalflussoperation erübrigen.

Es ist möglich, dass direkt mit der ersten Flussberechnung der Phase 1 die Maximalzahl von $\lfloor \frac{n-2}{2} \rfloor$ Mittelkomponenten mit exakt zwei Knoten auffinbar sind ist. Der Quell- und der Senkenknoten sind dabei in ihrer Komponente als einziger Knoten vertreten und bei ungerader Knotenzahl n existiert noch eine dritte einelementige Komponente. Im Optimum sind in einem solchen Fall nachfolgend nur $\lceil \frac{n-2}{2} \rceil$ Schnittberechnungen notwendig, jeweils eine pro Mittelkomponente. Bei ungerader Knotenzahl n findet eine der Berechnungen in Phase 2 für die einelementige Mittelkomponente statt. Die Schnittverläufe der Übrigen müssen jeweils die Kontraktionsknoten aus der ersten Zerlegung trennen. Summiert verlangt ein solches Szenario nach minimal $1 + \lceil \frac{n-2}{2} \rceil$ Maximalflussoperationen. Abbildung 28(a) zeigt einen Beispielgraphen mit zugehöriger Schnittfolge.

Durch Phase 1-Verläufe die von diesem Extrem abweichen, lassen sich keine zusätzlichen Schnitte einsparen:

- Sind mehr als ein Knoten in einer Quell- oder Zielkomponente versammelt, so bietet sich kein Einsparungspotential, da ein Fluss zwischen diesen Knoten später keine Kontraktionsknoten für ein Kantenpaar des aktuellen Flusses trennen kann; trivialerweise deswegen, weil für die Quell- und Zielkomponente keine zweite Kante existiert.

- Jede entstehende einelementige Mittelkomponente zieht eine weitere Schnittberechnung in Phase 2 nach sich.

- Jede entstehende Mittelkomponente mit mehr als zwei Knoten aus V, erfordert mehr als eine zusätzliche Minimalschnittberechnung um weiter zerlegt zu werden. Allerdings ist es unter Umständen möglich dort das gerade entstandene Kontraktionsknotenpaar mit denen weiterer Zerlegungen im selben Kontraktionsgraphen

zu positionieren und später gleichzeitig trennen zu lassen. Der zusätzliche Schnitt wird dann durch die Einsparung eines anderen Schnittes aufgewogen, das erreichbare Minimum wird jedoch nicht gesenkt.

Der Extremfall eines solchen Verlaufs tritt ein, falls vor der letzten Schnittberechnung in Phase 1 sämtliche Kanten von T jeweils paarweise entstanden und zu einem gemeinsamen Superknoten inzident sind. Dieser enthält als Einziger noch zwei Knoten aus V. Einen solchen Graphen zeigt Abbildung 28(b). Um dieses Szenario zu erzeugen, sind vorher $\lceil \frac{n-2}{2} \rceil$ einzelne Flussberechnungen notwendig. Diese besitzen möglichst drei Komponenten, wobei Quell- und Zielknoten jeweils als Blattschnitt abgetrennt werden. Tritt der Optimalfall ein und lassen sich die beiden im Superknoten verbliebenen Knoten auf eine solche Art und Weise trennen, dass auch sämtliche Paare von Kontraktionsknoten separiert werden, so wird nur noch diese eine Zerlegung benötigt und es stehen keine weiteren Schnittoperationen aus. Auch hier beträgt die Gesamtzahl an Flussoperationen $1 + \lceil \frac{n-2}{2} \rceil$.

Um Schnittberechnungen während Phase 2 des Algorithmus' einsparen zu können, ist es offensichtlich ebenso wichtig, während Phase 1 mehrere Kantenpaare am selben Superknoten zu benachbarn. Da zur Bildung jedes Paares eine Maximalflussberechnung zwingend erforderlich ist, entspricht die Anzahl der Paare aber auch der Anzahl der in jedem Fall durchzuführenden Operationen. Andererseits werden, bei Vorhandensein von i Paaren, dann im Optimum durch einen nicht-trivialen Schnitt auch $i-1$ weitere Berechnungen unnötig.

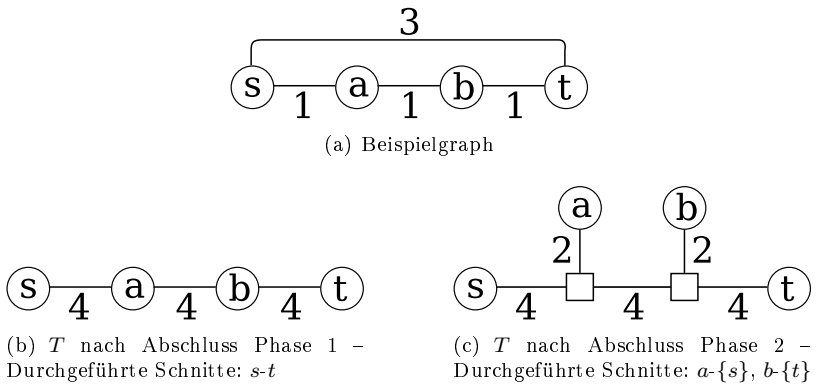
Für maximale Einsparungen ist also wieder ein Baum T geeignet, in dem sämtliche Kantenpaare zum selben Superknoten inzident sind, so dass danach im Optimum nur noch eine einzige Folgeoperation notwendig ist. Jede Zerlegung in Phase 1 muss möglichst (bei ungeradem n garantiert) 3 Komponenten besitzen, von denen es sich bei zweien immer um Blattschnitte des Quell- und des Senkenknotens handeln.

Jeder Schnitt trennt also zwei Knoten aus dem zu Beginn alle Knoten aus V enthaltenden Superknoten. Zusätzlich ist bekannt, dass der Superknoten am Ende der Flusskomponentenzerlegung exakt einen Knoten enthält. Die Menge von $n = |V|$ Knoten lässt sich so in minimal $\lceil \frac{n-1}{2} \rceil$ Operationen zerlegen; insgesamt sind in diesem Fall also $1 + \lceil \frac{n-1}{2} \rceil$ Berechnungen notwendig.

□

Es ist anzumerken, dass ein Algorithmusverlauf, welcher die volle Anzahl von $(n-1)$ Maximalflussberechnungen ausschöpft, die Existenz einer alternativen Fluss- beziehungsweise Schnittfolge nicht ausschließt, durch die Berechnungen hätten eingespart werden können. Im schlechtesten Fall, ist für den Graphen in Abbildung 28(a) ebenso eine Schnittfolge *a*-*b*, *c*-*d*, *s*-*a*, *b*-*c*, *d*-*t* möglich.

Um die spätere Trennung von Kontraktionsknoten durch nicht-kleinere Schnitte zu ermöglichen, wäre es insbesondere von Vorteil zuerst solche Flüsse mit kleinem Wert und vielen Fluskomponenten zu berechnen zu können. Die Tatsache, dass die hauptsächlich in frühen Stadien des Algorithmus zur Flussberechnung eingesetzten Maximalen Adjazenzordnung vor allem schwach zusammenhängende Knotenpaare identifizieren, kommt dieser Anforderung entgegen. Der Effekt ist gut in den durchgeführten Experimenten auf Zufallsgraphen zu beobachten (vgl. Abschnitt 8.4.2).



(a) Beispielgraph

(b) T nach Abschluss Phase 1 – Durchgeführte Schnitte: s-t

(c) T nach Abschluss Phase 2 – Durchgeführte Schnitte: a-$\{s\}$, b-$\{t\}$

Abbildung 29: Entstehung eines Relaxierten Gomory-Hu-Baums mit max. Knotenzahl

6.5. Knotenzahl des Relaxierten Gomory-Hu-Baums

Lemma 6.14
Ein mittels der vorgestellten Methodik berechneter Relaxierter Gomory-Hu-Baum besitzt zwischen n und $2n-2$ Knoten und demnach zwischen $n-1$ und $2n-3$ Kanten.

Beweis: Am Ende der Algorithmusphase 1 besteht der Baum T aus n Superknoten welche jeweils genau einen Knoten aus V vertreten. Je Schnittoperation der Phase 2 kann hier höchstens ein Leerer Knoten hinzukommen (vgl. Abschnitt 6.3.3).

Das Maximum in der in Phase 2 auszuführender Schnittberechnungen wird erreicht, falls in Phase 1 die erste (und einzige) Maximalflussberechnung zu einer Zerlegung in n Flusskomponenten mit je einem Knoten führt und es während der folgenden Schnittberechnungen zu keiner der im vorigen Abschnitt beschriebenen Einsparungen kommt. Ein solcher Verlauf benötigt noch einmal $(n-2)$ Schnitte und ist beispielhaft in Abbildung 29 dargestellt.

Damit ergibt sich die angegebene Maximalzahl der Knoten. Die Kantenzahl folgt trivial gemäß der Baumtopologie.

□

7. Implementierung

7.1. Überblick

Für die praktische Erprobung der in den vorangegangenen Abschnitten aufgestellten theoretischen Ergebnisse, wurde eine Implementierung in der Programmiersprache C++ unter GNU/Linux vorgenommen. Diese wurde arbeitsbegleitend mittels Doxygen dokumentiert.

Bestandteil der Implementierung sind zum Einen die benötigten Teilalgorithmen und Datenstrukturen für den klassischen Gomory-Hu-Algorithmus und die besprochenen neuen Kandidaten, zum Anderen eine Auswahl von Formatkonvertern für gespeicherte Graphen und mehrere Graphgeneratoren, auf die Abschnitt in 8.2.2 noch ausführlicher eingegangen werden soll.

Es wurde großer Wert darauf gelegt, keinen der Algorithmen unfair zu benachteiligen. Insbesondere sind die verwendeten Unteralgorithmen identisch und Detailoptimierungen jeweils für sämtliche Kandidaten durchgeführt worden.

Die Implementierungergebnisse liegen dieser Arbeit in elektronischer Form bei. Im Weiteren sollen nun Lösungen für wichtige Teilprobleme herausgegriffen und gesondert vorgestellt werden.

7.2. Implementierung wichtiger Teilprobleme

7.2.1. Kontraktion / Vollkontraktion

Die Graphkontraktion spielt sowohl im klassischen Gomory-Hu-Algorithmus als auch in der in Abschnitt 5.3 vorgestellten Modifikation eine zentrale Rolle. Wie bereits die Tatsache der Entwicklung des Gusfield-Agorithmus zeigt, verkompliziert sie jedoch ebenso die Implementierung.

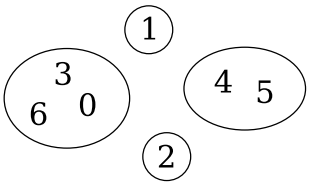
Um die kontrahierten Knoten eines Graphen zu verwalten, wird dabei prinzipiell eine *Union-Find*-Datenstruktur eingesetzt. Jeder Knoten besitzt eine eindeutige ID im Intervall 0 bis $(n-1)$. Desweiteren existieren zwei Felder `rep` und `member` mit Dimension n.

Die in einem gemeinsamen Kontraktionsknoten zusammengefassten Originalknoten besitzen jeweils einen gemeinsamen eindeutigen Repräsentantenknoten aus ihrer Menge. Im Feld `rep` sind entsprechende Verweise auf die ID des Repräsentanten, baumförmig organsiert, gespeichert. Der Repräsentant des Knoten mit ID i findet sich so durch Zugriff auf Position i des Feldes, was einen Knoten liefert welcher in der Baumstruktur näher am Repräsentanten liegt. Ist durch wiederholten Zugriff ein Knoten gefunden worden, dessen `rep`-Eintrag seiner eigenen ID entspricht, so handelt es sich um den Repräsentantenknoten seiner Knotenmenge. Dies trifft in einem unkontrahierten Graphen für jeden Knoten zu.

Wie von einer Union-Find-Struktur gewohnt kann bei der Rückverfolgung eines solchen Weges von einem Knoten zu seinem Repräsentanten eine Pfadverkürzung vorgenommen werden, um weitere Zugriffe zu beschleunigen und Kosten amortisieren zu können. Die gewöhnliche Pfadtiefe liegt jedoch bei Eins, worauf folgend noch eingegangen wird.

Index	0	1	2	3	4	5	6
rep	3	1	2	3	4	4	3
member	0	1	2	6	5	5	0

(a) `rep` und `member` Arrays



(b) Kontraktionsgraph

Abbildung 30: Datenstrukturen Knotenkontraktion

Um alle Knoten, welche durch einen gemeinsamen Kontraktionsknoten vertreten werden, aufzählen zu können, sind die Mitglieder einer solchen Menge über Verweise im `member` Array wie in einer linearen Liste organisiert. Vom Repräsentanten der Menge ausgehend verweist das Feld jeweils auf die ID des nächsten Mengenangehörigen bzw. auf den aktuellen Knoten selbst, falls kein weiterer Angehöriger existiert.

Ein Kontraktionsvorgang von zwei Knoten a und b besteht in einer einfachen Zuweisung der ID von a an die entsprechende `rep`-Position des Knoten b und invalidiert das `member` Array. Da ein Neuaufbau Gesamtkosten von $\mathcal{O}(n)$ verursacht, sind Kontraktionsanweisungen zu bündeln, bevor neue Mitgliedschaftabfragen vorgenommen werden. Auf die sonst üblichen Zähler für die Mitgliederzahl einer Menge wurde verzichtet, da Einzelknotenkontraktion in der vorliegenden Implementierung äußerst selten eingesetzt wird und der sonst nötige Speicherplatz so eingespart werden kann.

Im Gegensatz zur in den Implementierungen anderer Graphenalgorithmen vorkommenden häufigen und inkrementellen Einzelknotenkontraktion, wird für die vorliegenden Gomory-Hu-Baum-Algorithmen jedoch größtenteils der gesamte Graph auf Basis des [Relaxierten] Gomory-Hu-Baums und eines Superknotens S kontrahiert. Dabei werden die in den Superknoten des Baums verwalteten Knotenmengen einmalig durchlaufen und je von S abgehendem Teilbaum kontrahiert. Eine maximale Pfadtiefe im `rep`-Feld von Eins, kann trivial gewährleistet werden, indem der zuerst auftretende Originalknoten je Teilbaum direkt beim Mengendurchlauf für alle weiteren Knoten als Repräsentant eingesetzt wird. Nach dieser Mengenkontraktion ist wiederum ein Neuaufbau des `member` Arrays nötig, insgesamt liegen die Laufzeitkosten in $\mathcal{O}(n)$. Nach jeder solchen Kontraktionsoperation bleibt der Graph für längere Zeit stabil und eine Reihe von Arbeitsschritten werden auf ihm ausgeführt.

Im Verlaufe der folgenden Berechnung kommt es auf dem kontrahierten Graphen sehr häufig zur Iteration über alle Kanten eines Kontraktionsgraphen. Ein solcher Zugriff kann während der Programmierung leicht mittels eines Iterators abstrahiert werden, intern muss ein solcher jedoch sämtliche Knoten einer Kontraktionsknotenmenge durchlaufen und für jede der im Originalgraphen vorhandenen Kanten entscheiden ob diese im Kontraktionsgraphen sichtbar ist oder nicht. Zusätzlich verwandelt die Kontraktion den Graphen in einen Multigraphen mit parallelen Kanten, da diese während der Kontraktion bisher noch nicht zusammengefasst wurden.

Ist dieser Aufwand bei gering kontrahierten Graphen noch minimal, macht er auf

stark kontrahierten Versionen den eigentlichen Einsparungseffekt für nun unnötige Kantenbesuche zu nichte. Zu diesem Zweck wurde der Implementierung eine zweite Kontraktionsstufe, die *Vollkontraktion*, hinzugefügt, welche ab einem parametrisierbaren Kontraktionsfaktor für die im Graphen verbliebenen Knoten (Originalknoten und Repräsentanten) Ersatz-Inzidenzlisten anlegt, in welchen sämtliche Multikanten zusammengelegt und Eigenschleifen entfernt wurden. Der hierfür einmalig anfallende, zusätzliche Aufwand von $\mathcal{O}(n+m)$ Schritten amortisiert sich durch die im Anschluss erzielten Einsparungen.

7.2.2. Implementierung Maximalflussalgorithmus

Für die Implementierung des eingesetzten Maximalflussalgorithmus, wurde der Push-Relabel-Algorithmus nach Goldberg und Tarjan [GT88] ausgewählt. Dieser liegt in der verwendeten Variante mit einer worst-case Laufzeitabschätzung von $\mathcal{O}(n^3)$ zwar deutlich über den in [GR98] gezeigten möglichen Grenzen von $\mathcal{O}(nm \log n)$ beziehungsweise $\mathcal{O}(\min(n^{\frac{2}{3}}, m^{\frac{1}{2}})m \log(n^2/m) \log(U))$ für ganzzahlige Kantengewichte bis U, ist jedoch vergleichsweise einfach zu implementieren.

Zusätzlich zeigen die in [AKMO97] gemachten Erfahrungen, dass die zum Erreichen einer besseren Laufzeitabschätzung eingeführten Datenstrukturen häufig zu einer Erhöhung der durchschnittlichen Laufzeiten führen. Die Veröffentlichung gibt neben einem umfassenden Überblick zum Entwicklungsstand der Maximalflussalgorithmen auch aufschlussreiche empirisch ermittelte Laufzeittrends.

Die im Rahmen dieser Arbeit erstellte Implementierung verwendet mit der in [DM89] eingeführten Lückensuche (*Gap-Heuristik*) und einer genauen Entfernungsmarkierung zu Beginn des Algorithmus (*Exact Initial Labeling*), zwei einfache Heuristiken, die die reale Laufzeitentwicklung auf vielen der während der Experimente verwendeten Testgraphenklassen auf pseudo-lineares Niveau sinken lassen. Entsprechende Ergebnisse zeigt Abschnitt 8.4.1.

Der Push-Relabel-Algorithmus Ein Push-Relabel-Algorithmus (auch Preflow-Push-Algorithmus) verwaltet während seiner gesamten Laufzeit einen *Präfluss* (Preflow) auf dem Graphen G. Dies ist eine Flusswertzuordnung $f_p : e \in E \rightarrow [-w(e), w(e)]$ für die (zu diesem Zweck als gerichtet aufgefassten) Kanten von G, welche für jeden Knoten außer Quelle s und Ziel t des gewünschten Flusses sicherstellt, dass sein Zuflusswert nie unter dessen Abflusswert liegt. Die Differenz von Zufluss- und Abflusswert eines Knotens, heißt *Überschuss*. Liegt der Überschuss für jeden Knoten außer s und t bei Null, so ist der Präfluss ein Fluss.

Der Algorithmus erstellt zunächst einen Maximalen Präfluss, indem der maximal mögliche Flusswert über alle zum Quellknoten inzidenten Kanten aus diesem heraus geleitet wird. Durch fortgesetzte Verschiebungsoperationen auf Basis des Residualgraphen von G wird dieser im Anschluss in Richtung des Zielknotens weitergeleitet. Flussanteile welche einen möglicherweise vorhandenen Engpass im Graphen nicht überqueren können, werden schließlich zur Quelle zurück geleitet um aus dem Präfluss einen Fluss zu erstellen.

Um die Fortleitungsrichtung des Präflusses zu bestimmen, verwaltet der Algorithmus eine Entfernungsmarkierung $d : V \to \{0, \cdots, 2 \cdot |V| - 1\}$. Diese ist *gültig*, falls sie $d(s) = |V|$, $d(t) = 0$ und $d(x) \leq d(y) + 1$ für jede existierende Residualkante (x, y) erfüllt. Gilt $d(v) < |V|$, so bildet sie eine unter Schranke für die Entfernung von v zum Zielknoten t im Residualgraphen von G. Gilt $d(v) \geq |V|$, so ist $d(v) - |V|$ eine untere Abschätzung der Entfernung zum Knoten s.

Ein Knoten heißt *aktiv*, falls er weder Quell- noch Zielknoten ist und einen Flussüberschuss besitzt. Eine Residualkante (x, y) ist *zulässig*, falls $d(x) = d(y) + 1$ gilt.

Der allgemeine Push-Relabel-Algorithmus besteht nun aus einer wiederholten Ausführung von Push/Relabel-Operationen, bis der Überschuss in allen Knoten außer s und t getilgt ist.

Algorithmus 6: Abstrakter Push-Relabel Algorithmus

Data: $G = (V, E)$, Residualgraph $G_f = (V, E_f)$, Quelle s, Ziel t
Result: Maximaler s-t-Fluss in G

forall $v \in V$ **do** $d(v) := 0$;
Leite max. möglichen Fluss über alle Kanten heraus aus s;
$d(s) = |V|$;

while $\exists$ *aktiver Knoten* v *in* G **do**
  `push_relabel(`v`);`
`return();`

Procedure `push_relabel(`v`)`

Data: aktiver Knoten $v \in V$, Residualgraph $G_f = (V, E_f)$

if v *besitzt zulässige inzidente Residualkante* (v, w) **then**
  `// Push-Operation`
  $\delta :=$ min(Überschuss in v, Kantengewicht (v, w));
  Schiebe Flusswert δ von v nach w;
else
  `// Relabel-Operation`
  $d(v) := \min_{(v,w) \in E_f, c((v,w)) > 0)}(d(w) + 1)$;

Eine reale Implementierung kombiniert dabei üblicherweise sämtliche für einen gewählten aktiven Knoten möglichen Push-Operationen, bis entweder dessen Überschuss gesättigt ist oder es zu einer Relabel-Operation kommen muss.

Kritisch für die Laufzeit des Push-Relabel-Algorithmus, zeigt sich die Auswahl und Verwaltung der aktiven Knoten. Werden sie in einer einfachen FIFO-Datenstruktur vorgehalten, führt dies auf eine Komplexität von $\mathcal{O}(n^3)$. Alternativ lässt sich mit Hilfe einer Bucket-Datenstruktur für die Bearbeitung jeweils ein aktiver Knoten mit höchster Entfernungsmarkierung auswählen, was zu einer Abschätzung von $\mathcal{O}(n^2\sqrt{m})$ führt (jeweils [GT88]).

Beide Varianten wurden implementiert. Für die finale Version fiel die Entscheidung schließlich auf den FIFO-Algorithmus, da dieser zum Einen in Experimenten bessere Ergebnisse zeigte und seine Abschätzung sich außerdem unabhängig von der Kantenzahl zeigt. Zu Grunde liegt hier die Überlegung, dass sich die Kontraktionsgraphen mit stärkerer Kontraktion durch die verwendete Kontraktionstechnik vermehrt zu Multigraphen wandeln, während ihre Knotenzahl fortlaufend abnimmt (vgl. Abschnitt 7.2.1).

Heuristiken Für die vorliegende Implementierung wurde der Algorithmus von Goldberg und Tarjan wie in [DM89] um zwei verbreitete Heuristiken erweitert, durch deren Kombination das Laufzeitverhalten entscheidend verändert werden konnte.

Die ***Genaue Initiale Entfernungsmarkierung*** (Exact Initial Labeling) verwendet anstatt der im vorausgehenden Abschnitt gezeigten trivialen Initialmarkierung von $d(s) = |V|$, $d(t) = 0$ und $d(v) = 0$ für alle $v \in V, v \notin \{s, t\}$ exakte Entfernungswerte, welche vor Algorithmusbeginn durch eine vom Zielknoten t ausgehende Breitensuche ermittelt werden. Dies ermöglicht eine sofortige, gezielte Ableitung des Flussüberschusses in Richtung des Zielknotens und spart eine Vielzahl von Relabel-Schritten ein, in denen die Entfernungslabels der Zwischenknoten sonst erst inkrementell auf den entsprechenden Wert erhöht würden. Zusätzlich verhindert es die Benutzung von Beginn an bestehender 'Sackgassen'.

Auf die häufig anzutreffende Erweiterung, ein solches exaktes Entfernungsmaß nach Ausführung einer festen Anzahl von Algorithmusschritten wiederholt aufzubauen, wurde verzichtet, um die Zahl empirisch festgelegter Parameter gering zu halten.

Wurden durch die Einführung der Genauen Initialen Entfernungsmarkierung bereits Laufzeitverbesserungen erreicht, bezieht die Implementierung einen Großteil ihrer Einsparungen aus der ***Lückensuche*** (Gap-Heuristik). Dieses Verfahren basiert auf der einfachen Beobachtung, dass sobald in den vergebenen Entferungsmarkierungen eine Lücke entsteht, der Flussüberschuss aller höher markierten Knoten den Zielknoten nicht mehr erreichen kann und zur Quelle zurückkehren muss. Werden für jede Markierung die mit ihr versehenen Knoten gespeichert, so lässt sich im Anschluss an jede Relabel-Operation leicht überprüfen, ob noch weitere Knoten mit dem alten Label existieren. Ist dies nicht der Fall, so können sämtliche aktiven Knoten mit höherem Label aus der Berechnung herausgenommen werden. Sie werden erst in einer zweiten Phase wieder berücksichtigt, in der ihr Überschuss zum Quellknoten zurückgeleitet wird.

Durch dieses Verfahren werden 'Sackgassen' frühzeitig erkannt und große Flusswerte aus der Berechnung genommen, ohne dass sie aufwändig und sinnlos in im Residualgraphen bereits abgetrennten Teilgraphen kreisen bis dort sämtliche Entfernungsmarken auf $|V|$ angestiegen sind (ab welchem Zeitpunkt der Rückfluss zur Quelle beginnt). Der Algorithmus erhält dadurch einen sehr lokalen Charakter, was sich unter anderem darin äußert, dass sich die Zahl der besuchten aktiven Knoten in den durchgeführten Experimenten sehr häufig linear entwickelt und regelmäßig auch unter der Knotenzahl des Graphen lag.

Die Wirkung dieser beiden Heuristiken zeigen beeindruckend, die in Abschnitt 8.4.1

zusammengefassten Experimente.

7.2.3. Einsatzvorgaben für Maximale Adjazenzordnungen

Als schwierig erwies sich die Entscheidung über Einsatzzeitpunkte und -häufigkeiten für Maximale Adjazenzordnungen während des Algorithmusverlaufes. Prinzipiell sind die Erfolgsaussichten, d.h. die Wahrscheinlichkeit des Auffindens eines bisher unbekannten Minimalschnitts, und auch die erwarteten erzielbaren Laufzeitvorteile gegenüber einem Maximalflussalgorithmus auf den großen, unkontrahierten Graphen zu Beginn des Algorithmus besonders groß, so dass sie besonders in dieser frühen Phase eingesetzt werden sollten.

In den Experimenten zeigte sich, dass Anzahl der mittels Maximaler Adjazenzordnungen auffindbaren Schnitte sehr stark vom Layout und der Kantengewichtsverteilung im Graphen abhängt. Äußerst kontraproduktiv sind dabei einzelne Knoten, welche in ihrer Adjazenz stark nach unten abweichen und so in sämtlichen Berechnungen als Schlussknoten der Ordnung auftauchen. Diese bleiben durch den aus der Ordnung resultierenden Blattschnitt ebenso in den folgenden Kontraktionsgraphen erhalten, insofern nicht mehr als zwei Flusskomponenten aus dem Schnitt ableitbar sind.

Liefert bereits die erste auf einem Kontraktionsgraphen gebildete Ordnung keinen unbekannten Schnittverlauf, so zeigte sich auch die Erfolgswahrscheinlichkeit eines zweiten Versuchs gering. Trotz ihrer Überlegenheit in der worst-case Laufzeitabschätzung, müssen Fehlversuche möglichst vermieden werden, um die Gesamtlaufzeit gering zu halten.

Während der Implementierung entstanden drei Ansätze für die Steuerung der Dosierung des Einsatzes Maximaler Adjazenzordnungen:

Limit Kontraktionsgraphgröße Das erste verwendete Limit, bestand in der Anweisung in Phase 1 für jeden gebildeten Kontraktionsgraphen, welcher eine prozentual zum Originalgraphen angegebene Knotenanzahl überschreitet, vor der Verwendung des Maximalflussalgorithmus eine Maximale Adjazenzordnung zu erstellen. Dieser Ansatz erweist sich jedoch sehr schnell als ungenügend, da vielzählige Graphklassen existieren, deren Gomory-Hu-Baum zu einer Sternform mit Tiefe Eins neigt (vgl. Abschnitt 8.4.2 und 8.4.3). Bei diesen weicht die durchschnittliche Kontraktionsgraphengröße nicht von der des Originalgraphen ab. Hierdurch werden während des gesamten Algorithmusverlaufs Adjazenzordnungen gebildet und ihr Einsatz bleibt nicht, wie beabsichtigt, auf den Abschnitt beschränkt indem sie hohe Erfolgknoten versprechen.

Statische Aufrufanzahl Eine Alternative ist die statische Angabe auf den wieviel ersten erstellten Kontraktionsgraphen jeweils eine Adjazenzordnung gebildet werden soll. Diese Lösung überträgt die Dosierung dem Nutzer und verlangt von ihm hohe Erfahrung und Kenntnis der eingesetzten Graphklasse. Sie ist unflexibel und nicht adaptiv zur Veränderung der Kontraktionsgraphen im Berechnungsverlauf.

Limit Anzahl unkontrahierter Knoten Dies führte schließlich zur Einführung der letztlich verwendeten Dosierungsanweisung. Diese ist abhängig von der Anzahl der

unkontrahierten Knoten des Kontraktionsgraphen, also der Kardinalität der Knotenmenge des Superknotens, für welchen der Kontraktionsgraph erstellt wurde. Kontraktionsknoten welche nur einen Originalknoten enthalten, gehören hier ausdrücklich nicht dazu.

Dieses Limit sinkt im Gegensatz zur reinen Kontraktionsgraphengröße mindestens linear und kann gleichzeitig auf starke Abfälle der Kontraktionsgraphgrößen während der Berechnung reagieren. In der verwendeten Version besteht es aus zwei Parametern. Zum Einen die Anzahl der je Kontraktionsgraph auszuführenden Versuche, zum Anderen das angesprochene Limit, ab welchem auf eine reine Verwendung der Maximalflussalgorithmen übergegangen wird.

8. Experimentelle Ergebnisse

8.1. Überblick

Für die Durchführung der folgenden Experimente wurden die in Abschnitt 7 umrissenen Programme mit der GNU Compiler Collection in Version 4.1.2 bei höchster automatischer Optimierungsstufe (`-O3`) übersetzt.

Der Großteil der Experimente wurde auf zwei Rechnern mit Intel Pentium 4 Prozessoren von 2,93 bzw. 3 GHz durchgeführt, welche über 1 bzw. 2 GB Arbeitsspeicher verfügten. Die in Abschnitt 8.4.1 gezeigten Ergebnisse, entstanden auf einem System mit auf 1 GHz getaktetem AMD Athlon XP mobile Prozessor und 512 MB Arbeitsspeicher.

Die Laufzeitentwicklung der einzelnen Algorithmen ist grundsätzlich von den zur Erzeugung des Testgraphen gewählten Randparametern abhängig. Hier wurde jeweils versucht eine repräsentative Auswahl zu treffen.

Die verwendeten Programme und Skripte, sowie die erzielten Messreihen sind zur leichten Nachvollziehbarkeit in elektronischer Form dieser Arbeit beigefügt.

8.2. Graphgeneratoren & Problemauswahl

Graphen lassen sich in bestimmte topologische Klassen einteilen. Um die Leistungsfähikeit der Algorithmen zu vergleichen, ist es nun zum Einen interessant, wie sie sich auf häufig eingesetzten Graphklassen von theoretischem (Zufallsgraphen) und praktischem (Power-Law-Graphen) Wert verhalten. Um des Weiteren möglichst aussagekräftige Ergebnisse zu erreichen, ist es ebenso wichtig Klassen und Instanzen auszuwählen, für welche bestimmte Merkmale der getesteten Algorithmen besonders hervortreten.

Eine wichtige Rolle spielen dabei auch die Eigenschaften der für die Generierung der Testgraphen verantwortlichen Programme. Zeichnen sich die Knoten der erzeugten Graphen, zum Beispiel, durch eine charakteristische Speicherungs- oder Nummerierungsreihenfolge aus, die auf die topologischen Rolle bestimmter Knoten hinweist, so kann dies das Verhalten einer Implementierung beeinflussen. Um solche Effekte zu beheben, wurden die Knotenbezeichnungen solcher Graphen nach ihrer Erzeugung noch einmal randomisiert.

Die beiden folgenden Abschnitte stellen die betrachteten, wie auch die schließlich intensiver verwendeten, Graphengeneratoren vor.

8.2.1. Goldberg-Generatoren

Die erste Gruppe der zur Verfügung stehenden Graphengeneratoren, wurde von Andrew Goldberg und Mathew S. Levine erstellt um u.A. in [GGP+99] verschiedene Implementierungen des Gomory-Hu- und des Gusfield-Algorithmus zu vergleichen. Ihr C-Quellcode ist heute unter [GLG] frei erhältlich.

Die Sammlung umfasst zwölf verschiedene Generatoren, welche vor allem Zufallselemente, Baum- und Kreistopologien miteinander verbinden. Tabelle 1 stellt die verfügbaren Graphklassen vor.

Generator	Erzeugte Graphen
`irregulargen`	Graph mit x kantendisjunkten Hamiltonkreisen plus y Kanten zwischen zufälligen, überschneidungsfreien Knotenpaaren
`cyclegen`	Graph aus Hamiltonkreis mit potentiell sehr schweren Kantengewichten, plus x 'leichter', zufällig eingefügter Querkanten.
`bikewheelgen`	Graph mit einem großen, alle bis auf zwei Knoten umfassenden Kreis. Zusätzlich sind die Kreisknoten alternierend jeweils mit einem der beiden ausgesparten Knoten verbunden, zwischen welchen eine weitere Verbindung besteht. Maximaldistanz zweier Knoten: 3 Hops
`dblcyclegen`	Graph mit einem Hamiltonkreis, mit konstant schwerem Kantengewicht, über nach ID aufsteigend geordneten Knoten. Dazu Verbindung jeweils drei Hops in Kreis 1 entfernt liegender Knoten mit leichten Kanten. Falls Knotenzahl nicht durch 3 teilbar ist, besteht das Resultat in zwei *regelmäßig verkreuzten* Hamiltonkreisen. Anderenfalls, bilden die leichten Kanten drei knotendisjunkte Kreise. In Kreis 1 weichen zwei Kanten mit ihrem Gewicht nach unten ab, in Kreis 2 ist das Gewicht von vier Kanten leicht gesteigert.
`pathgen`	Graph mit Pfad der Länge k dessen Gewichte sehr hohe Werte erreichen können. Zusätzlich werden alle nicht auf dem Pfad liegenden Knoten mit einer schweren Kante an Pfadknoten angebunden. Zuletzt erfolgt das Hinzufügen von Zufallskanten mit niedrigen Gewichtswerten, bis die gewünschte Dichte erreicht ist.
`noigen`	Zufällige Einteilung der Graphknoten in k Gruppen, deren Kantengewichte untereinander potentiell weit höher liegen, als zu Knoten anderer Gruppen. Sicherstellung des Graphzusammenhangs durch nach Knoten-IDs geordnetem Kreis. Zusätzlich Einfügen von Zufallskanten, bis gewünschte Dichte erreicht ist.
`prgen`	Padberg-Rinaldi-Generator
`randomgen`	Randomisierte, nahezu reguläre Graphen (Bug)
`regulargen`	Reguläre Graphen durch Verschmelzung von x randomisierten Hamiltonkreisen oder Paarungen
`treegen`	Baum mit potentiell schweren Kanten, dazu Zufallskanten bis gewünschte Kantenzahl erreicht ist.
`unbalanced`	Unbalancierter Baum mit bestimmbarer Balanceabweichung. Kantengewichte entsprechen Entfernung von Wurzelknoten.
`wheelgen`	Großer Kreisgraph mit Zentralknoten, welcher mit allen anderen Knoten verbunden ist

Tabelle 1: Übersicht Goldberg-Generatoren

Für die Erzeugnisse aller Generatoren wurden einschätzende Versuche durchgeführt. Aufgrund ihrer Aussagekraft über spezifische Stärken der Algorithmen mit Flusskomponentenzerlegung, wurden mit `cyclegen` und `dblcyclegen` schließlich zwei Programme für eine ausführliche Schilderung ausgewählt, mit denen sich Kreisgraphen und bestimmte Eulergraphen erzeugen lassen. Die von den restlichen Generatoren gebildeten Graphklassen ähneln entweder den beiden gewählten oder werden durch später vorgestellte Generatoren mit abgedeckt. So besitzen viele der Graphen starke Zufallskomponenten, so dass die getesteten Algorithmen ein sehr ähnliches Verhalten wie auf reinen Zufallsgraphen zeigten.

Im folgenden soll jedoch auch auf die Ergbnisse der restlichen Goldberg-Generatoren Bezug genommen werden, sobald ein Vergleich interessante Einsichten bietet.

8.2.2. Eigenimplementierungen

Um zusätzliche, interessante Graphklassen untersuchen zu können und größeren Einfluss auf die erzeugten Graphen zu besitzen, wurden weiterhin eine Auswahl von Graphgeneratoren selbst entwickelt.

Dabei handelt es sich zunächst um einen Generator für klassische **Zufallsgraphen**. Dieser unterstützt sowohl die Erzeugung von Multigraphen, bei Angabe gewünschter Knoten- und Kantenzahlen, als auch von einfachen Graphen. Für letztere lassen sich die gewünschte Knotenzahl und Kantenwahrscheinlichkeit bestimmen. In den Experimenten wurden aussschließlich Zufallsgraphen ohne Multikanten verwendet. Sind für Zufallsgraphen Kantenzahlen angegeben, so wurden diese zur Erzeugung in zugehörige Kantenwahrscheinlichkeiten umgerechnet. Sie entsprechen daher nur dem Erwartungswert der Kantenzahl des Graphen.

Aufgrund ihrer Bedeutung in natürlich vorkommenden Netzwerken, sollten außerdem **Power-Law-Graphen** erzeugt werden. Der hierfür implementierte Generator arbeitet gemäß dem Albert-Barabási-Modell ([BA99]) mit *Preferential Attachment* und *Initial Attractiveness*. Dabei wächst der Graph ausgehend von einem initialen Kern. Jeder neue Knoten erhält eine vorgegebene, feste Anzahl von Kanten zu bereits im Graphen vorhandenen Knoten. Die Verbindungswahrscheinlichkeit ist proportional zum bereits erreichten Knotengrad des Kantenpartners. Zusätzlich wird jedoch sichergestellt, dass auch Knoten ohne Nachbarn eine, wenn auch niedrige, Grundwahrscheinlichkeit besitzen.

Da viele der bisher vorgestellten Graphklassen dazu neigen Gomory-Hu-Bäume mit sehr geringem Durchmesser zu entwickeln, wurde zusätzlich ein Generator für **Tree-Seeded-Graphen** implementiert. Dieser erstellt Graphen auf Basis eines als Eingabeparameter bestimmten Baums. Solange der Baumgraph noch über zusammenhängende Komponenten verfügt, wird aus einer Komponente ein zufälliges Knotenpaar gewählt und dieses im zu erzeugenden Graphen verbunden. Im nächsten Schritt wird das Gewicht der niedrigwertigsten Kante auf dem Baumpfad zwischen beiden Knoten dekrementiert und die Kante bei Erreichen des Gewichts Null gelöscht. Anschließend beginnt die Iteration von Neuem.

Der vorgegebene Baumgraph ist dabei für gewöhnlich kein Gomory-Hu-Baum für den entstehenden Graphen. Besitzt er jedoch geringe Knotengrade für seine inneren

Knoten, so gilt dies ebenso für die Gomory-Hu-Bäume des erzeugten Graphen. Sie haben nicht-trivialen Durchmesser und weite Verästelungen. Diese Eigenschaft ist für die schnelle Absenkung der Knotenzahl je Kontraktionsgraph äußerst vorteilhaft. Die Wahrscheinlichkeit einer Kante nimmt zwischen Knoten mit kurzen Distanzen und hohen Wegkapazitäten im Eingabebaum stark zu. So entstehen zusammenhängende Graphen mit stark lokaler Vermaschung, welche weitläufig konstant abnimmt. Dies ist zum Beispiel ein großer Unterschied zu den Zufallsgraphen, deren Kantenbeziehung global sind, dass heißt, zwischen jedem Knotenpaar eine gleiche Verbindungswahrscheinlichkeit besteht.

Um Graphen mit regelmäßigeren Bestandteilen zu untersuchen, wurde schließlich ein Generator für **Cluster-Grid-Graphen** entwickelt. Die hiervon gebildeten Graphen besitzen eine mehrdimensionale Gitterstruktur, deren Knotenpunkte durch gleich große, zufällige Teilgraphen mit wählbarer Dichte gebildet werden. Die Gitterdimensionen können dabei wahlweise an ihren Enden kurzgeschlossen werden um Ringe und Tori zu produzieren.

8.3. Zielstellungen der vorgestellten Algorithmen

Sowohl der Einsatz Maximaler Adjazenzordnungen, als auch die Technik der Flusskomponentenzerlegung sind angedacht, um Flussberechnungen einsparen zu können oder zu erleichtern.

Dabei sollen die Maximalen Adjazenzordnungen besonders zu Beginn des Algorithmusverlaufs zur schnellen Bestimmung erster Maximaler Flüsse zwischen beliebigen Knotenpaaren eingesetzt werden. In diesem Stadium sind die betrachteten Kontraktionsgraphen noch sehr groß, so dass ihre bessere Laufzeitabschätzung einen Vorteil gegenüber der Flussberechnung mittels Push-Relabel-Algorithmus verspricht. Zusätzlich ist zu diesem Zeitpunkt noch mit wenig Kollisionen zu bereits bekannten Schnittverläufen zu rechnen. Wie in Abschnitt 6.2 gezeigt wurde, sind *gültige* Flüsse mit einem Kontraktionsknoten als Endpunkt verwertbar, was die Verwendbarkeit der Ergebnisse Maximaler Adjazenzordnungen steigert.

Durch die Flusskomponentenzerlegung, sollen für die, während des gesamten Verlaufs gefundenen, Flüsse alle realisierbaren Mehrfachaufteilungen der Knotenmenge genutzt werden. Hierdurch bietet sich die Möglichkeit die zur Flussberechnung herangezogenen Kontraktionsgraphen zu verkleinern und damit die Berechnung selbst zu beschleunigen. Weiterhin besteht die Chance durch eine passende Abfolge von Zerlegungen Schnittberechnungen einsparen zu können.

Ein kritischer Faktor in den folgenden Experimenten wird sein, ob die dadurch erzielten Laufzeiteinsparungen ausreichen, um den unweigerlich höheren Verwaltungsaufwand aufzuwiegen bzw. vernachlässigbar zu machen. Wie sich zeigen wird, ist dies überaus abhängig von der Topologie des Eingabegraphen und trifft leider nur auf einigen Klassen zu. Eine wichtige Rolle hierfür spielt die festgestellte Realleistung der Maximalflussalgorithmen auf den getesteten Graphen.

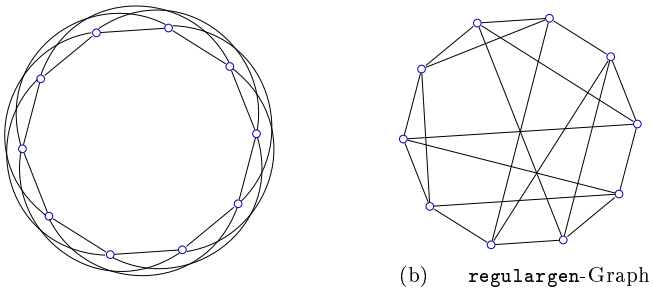
8.4. Ergebnisse

8.4.1. Realleistung der PushRelabel-Implementierung

Um die Ergebnisse der folgenden Experimente zu begründen, soll zuerst die von der Implementierung des Push-Relabel-Algorithmus (siehe auch Abschnitt 7.2.2) erzielte Laufzeitentwicklung vorgestellt werden. Diese zeigt sich naturgemäß graphenabhängig, bestätigt jedoch den auch in [AKMO97] festgestellten Sachverhalt, dass die mittlere Laufzeit mit Einführung der Lückensuche und der Genauen Initialen Entfernungsmarkierung markant von ihrer worst-case-Abschätzung $\mathcal{O}(n^3)$ abweichen kann.

Insbesondere zeigt sich in vielen Messungen eine gänzlich andere Verlaufscharakteristik mit pseudo-linearem Anstieg. In der zitierten Veröffentlichung konnte für viele Graphklassen eine empirische obere Laufzeitgrenze von $\mathcal{O}(n^{1,5})$ festgestellt werden, die auch dort bis zu sehr hohen Knotenzahlen, visuell lineares Verhalten zeigt.

Eine deutlich der worst-case-Abschätzung angenäherte Entwicklung besitzen als Einzige der überprüften Graphklassen die Doppel-Kreis-Graphen des `dblcyclegen`-Generators. Auffällig ist hier besonders der Unterschied zu den prinzipiell verwandten Graphen des `regulargen`-Generators. Werden auch dort zwei Hamiltonkreise zu einem regulären Graphen kombiniert, diesmal jedoch randomisiert, resultiert wieder eine pseudo-lineare Laufzeitentwicklung. Das verschiedene Verhalten muss in der systematischen Kreuzungscharakteristik der Hamiltonkreise und dem größeren Durchmesser der Graphen des `dblcyclegen`-Generators begründet sein.

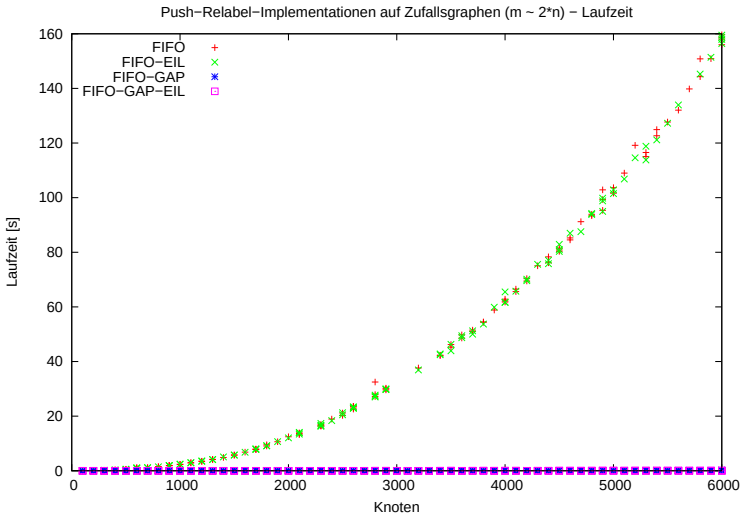


(a) Doppel-Kreis-Graph

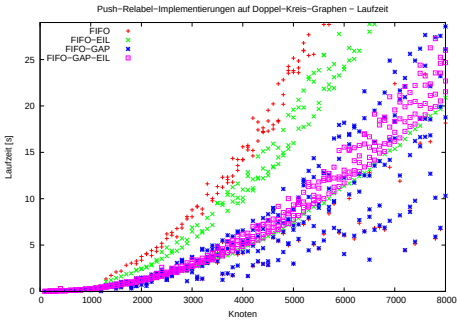
(b) `regulargen`-Graph aus zwei Hamiltonkreisen

Abbildung 31: Unterschied Doppel-Kreis-Graphen und `regulargen`-Graphen

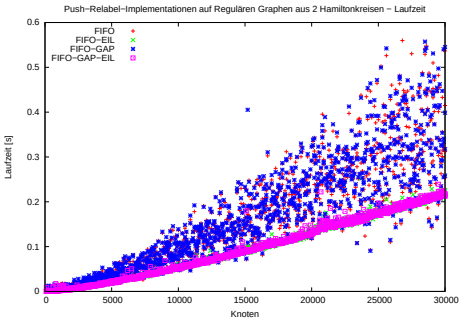
Abbildung 32 zeigt beispielhafte Ergebnisse für Zufallsgraphen, Doppel-Kreis-Graphen und Reguläre Graphen. Für die Messungen wurde nach Erzeugung des Graphen ein zufälliges Knotenpaar ausgewählt und mit verschiedenen Implementierungsvarianten je eine Maximalflussberechnung zwischen diesem durchgeführt. Da der Berechnungsaufwand abhängig von der Graphtopologie und Lage des gewählten Paares ist, können sich auch bei gleicher Graphengröße starke Schwankungen zeigen. Dargestellt sind die Laufzeitentwicklungen für den reinen Push-Relabel-Algorithmus mit FIFO-Strategie nach Goldberg/Tarjan und jeweils dessen Kombination mit Genauer Initialer Entfernungsmarkierung (_EIL) und Lückensuche (_GAP).



(a) Zufallsgraphen ($m = 2n$)



(b) Doppel-Kreis-Graphen



(c) Reguläre Graphen

Abbildung 32: Laufzeitentwicklung von Push-Relabel-Implementierungen auf verschiedenen Graphklassen

Vergleich mit Laufzeiten Maximaler Adjazenzordnungen Nach den obigen Ergebnissen, ist es nun interessant eine Vergleich zwischen den Laufzeiten der Maximalen Adjazenzordnungen und der Push-Relabel-Implementierung mitsamt der vorgestellten Heuristiken durchzuführen. Dafür wurde von den getesteten Graphklassen jeweils eine Instanz erzeugt, auf dieser eine Maximale Adjazenzordnung aufgebaut und der Maximale Fluss zwischen ihren letzten beiden Knoten abgeleitet. Im nächsten Schritt, wurde der Maximalfluss zwischen dem gleichen Knotenpaar mit dem Push-Relabel-Algorithmus berechnet und beide Laufzeiten aufgezeichnet.

Die dabei zu Stande kommenden Ergebnisse zeigen sich unvorteilhaft für die Maximalen Adjazenzordnungen, denn gerade Flüsse zwischen den von ihnen identifizierten schwach zusammenhängenden Knotenpaaren, können vom Push-Relabel-Algorithmus häufig sehr lokal berechnet werden. Eine Auswertung der während ihrer Berechnung besuchten Knoten zeigt, dass diese meist weit unter der Knotenzahl des Graphen liegen. Zum Einen liegt dies an der Tatsache, dass die Adjazenzordnung bevorzugt unzusammenhängende Teile des Graphen und speziell einzelne abgetrennte Knoten auffindet. Eine Push-Relabel-Flussberechnung zwischen den Knoten verschiedener Komponenten, ist dann bereits nach der Exakten Initialen Entfernungsmarkierung sofort beendet, da eine Unerreichbarkeit festgestellt werden kann. Die Lokalität kann jedoch ebenso für zusammenhängende Graphen beobachtet werden. Hier hilft die Tatsache, dass der Flusswert im Vergleich zum Zusammenhangswert des restlichen Graphen relativ klein ist und schnell in die richtige Richtung fortgeleitet werden kann, ohne auf Engstellen im Graphen zu stoßen.

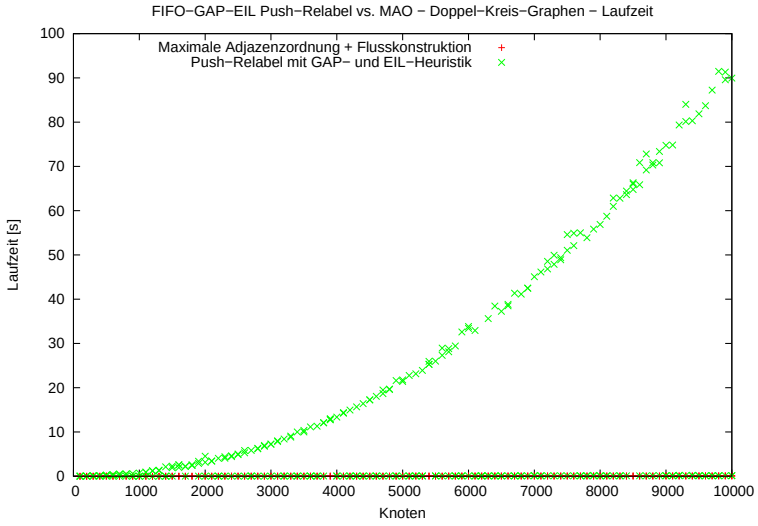
Beim erfolgreichen Einsatz von Maximalen Adjazenzordnungen in anderen Algorithmen, wie solchen für Minimale Schnitte, kann dieser Malus ausgeglichen werden, da durch Knotenkontraktionen später auch aufwändigere Flussberechnungen durch die Erstellung von Adjazenzordnungen ersetzt werden können. Dies trifft hier nicht zu.

Weiterhin kann nicht argumentiert werden, dass der getroffene Vergleich die Maximalen Adjazenzordnungen benachteiligt, da sie das schwach zusammenhängende Knotenpaar für die Push-Relabel-Berechnung erst identifizieren. Diese Feststellung ist zwar prinzipiell korrekt, auch bei völligem Verzicht auf den Einsatz von Adjazenzordnungen, würde der Schnittverlauf zwischen den beiden Knoten im fortschreitenden Algorithmusverlauf jedoch berechnet werden müssen, wenn auch nicht zwangsweise durch einen Fluss mit den gleichen Endpunkten.

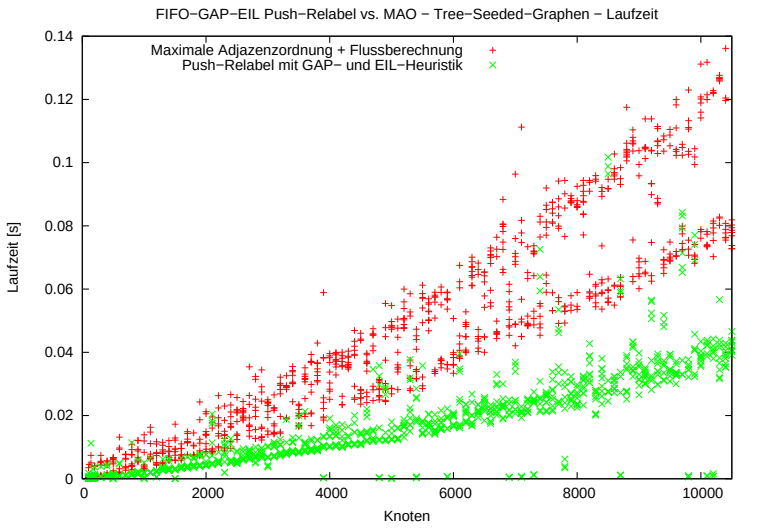
Abbildung 33 zeigt Laufzeitvergleiche für verschiedene Graphenklassen. Speziell ist zuerst das für die Maximalen Adjazenzordnungen vorteilhafte Beispiel der Doppel-Kreis-Graphen dargestellt. Das schlechte Abschneiden des Push-Relabel-Algorithmus auf diesen Graphen wurde bereits weiter oben festgestellt. Weiterhin ist der Verlauf für Tree-Seeded-Graphen abgebildet, welche sich (gegenüber Zufallsgraphen) durch garantierten Zusammenhang auszeichnen.

8.4.2. Experimente Zufallgraphen

Für Zufallsgraphen wurden zwei verschiedene Experimente ausgeführt. Zuerst wurde das Knoten-Kanten-Verhältnis konstant festgelegt und die Knotenzahl der untersuchten Graphen kontinuierlich erhöht. Anschließend wurden in einem zweiten Experiment



(a) Doppel-Kreis-Graphen



(b) Tree-Seeded-Graphen

Abbildung 33: Laufzeitvergleich Maximaler Adjazenzordnungen und Push-Relabel-Berechnung des gleichen Flusses

Graphen mit fester Knotenzahl aber zunehmender Anzahl von Kanten betrachtet, um die Auswirkungen der zunehmenden Vermaschung zu beobachten.

Der Laufzeitvergleich zwischen den neu vorgestellten Algorithmen und dem klassischen Gomory-Hu-Algorithmus zeigt sich dabei in beiden Fällen unbefriedigend.

Das Aussehen von Zufallsgraphen ist stark abhängig von der eingesetzten Kantenwahrscheinlichkeit. Beginnend mit einem unzusammenhängenden Graphen aus sehr vielen einzelnen Komponenten entwickeln sie sich mit steigender Kantenwahrscheinlichkeit zu dichten Graphen mit sehr homogoner Knotengradverteilung und beinahe regulärem Charakter. Insbesondere dichte Zufallsgraphen besitzen sehr kurze mittlere Knotendistanzen und verfügen über keine natürliche Clusterung oder Lokalität in den Knotenverbindungen. Insgesamt äußern sich diese Eigenschaften in einem sehr flachen, zur Sternförmigkeit neigenden Gomory-Hu-Baum.

Mehrfachzerlegungen sind häufig auf maximal drei Komponenten begrenzt, von denen Quell- und Zielkomponente des Flusses üblicherweise nur einen einzigen Knoten enthalten. Der Kontraktionsgraph der Mittelkomponente enthält somit die gleiche Anzahl an Knoten, wie der ursprüngliche Graph und die Vorteile der durch Mehrkomponentenzerlegung möglichen stärkeren Kontraktion können nicht genutzt werden. Insbesondere besitzen selbst die Kontraktionsgraphen in einem sehr späten Stadium des Algorithmus (zum Beispiel in Phase 2) noch Knotenzahlen, welche nur minimal vom Originalgraphen abweichen. Das Streben zu sternförmigen Gomory-Hu-Bäumen verhindert also die durch die Flusskomponentenzerlegung angestrebten Einsparungen. Dieser Fakt wird auch noch in anderen Graphenklassen zu beobachten sein.

Für den Einsatz Maximaler Adjazenzordnungen zeigt sich als generelles Problem von dichten Zufallsgraphen, dass meist einige wenige Knoten existieren, welche in ihrem Knotengrad vom Erwartungswert stark nach unten abweichen und somit extrem häufig im Ergebnis der Maximalen Adjazenzordnungen als Flusspartner auftauchen. Auf einem solchen Graphen sind mitteles Maximaler Adjazenzordnungen dann nur sehr wenige Flüsse auffindbar, da sehr schnell immer wieder die bereits bekannten Ergebnisse geliefert werden.

Entwicklung der Zielparameter Werden die als Zielstellung gesetzten Randarameter betrachtet, ist vor allem das Experiment mit zunehmender Graphendichte interessant. In den dargestellten Verläufen zeigen sich sehr gut die verschiedenen Charakteristika von Zufallsgraphen mit zunehmender Dichte und Einblicke in das Verhalten der Algorithmen.

Wie in Abbildung 36 gezeigt, gelingt es durch die Flusskomponentenzerlegung zunächst erfolgreich die mittlere Größe der Kontraktionsgraphen abzusenken. Auffällig ist, der durch die größer werdenden unzusammenhängenden Komponenten bedingte Abfall der Kontraktionsgraphengröße auch für den Gomory-Hu-Algorithmus. Bei den vorher niedrigeren Kantenwahrscheinlichkeiten ist eine der gefundenen Schnittmengen gewöhnlich nur sehr klein, was wieder zu sternförmigen Gomory-Hu-Graphen und großen Kontraktiongraphen führt. Die Verfahren mit Flusskomponentenanalyse identifizieren in diesem Stadium bereits mit der ersten Zerlegung sämtliche unzusammenhängenden Komponenten. Es ist jedoch nicht möglich aus diesem Vorteil größeren Nutzen

zu ziehen, da die Flussberechnungen in derart unzusammenhängenden Graphen hauptsächlich von der Größe der größten zusammenhängenden Komponente abhängen und beinahe konstante Zeit benötigen.

Abbildung 37 zeigt die durch Maximale Adjazenzordnungen gelieferten, verwertbaren Maximalflüsse im Verhältnis zur Zahl der ingesamt gebildeten Adjazenzordnungen. Dabei ist die Mehrfachzerlegung zunächst so erfolgreich, dass das gesetzte Ausführungslimit für noch vorhandene unkontrahierte Knoten bereits nach dem ersten Durchlauf unterschritten wurde. Mit steigender Mittlerer Kontraktionsknotengröße wächst dann auch die Zahl eingesetzter und erfolgreicher Adjazenzordnungsberechnungen. Durch die steigende Kontraktionsgraphengröße und den zunehmenden Sterncharakter des Gomory-Hu-Baums, kommen viele durch Adjazenzordnungen gefundene Blattschnitte bei Zufallsgraphen mit hoher Dichte in jedem Kontraktionsgraphen als Kontraktionsknoten vor, wodurch die Berechnungen dort immer wieder die selben Ergebnisse liefern. Die Folge ist ein Absinken der Zahl erfolgreicher Verläufe bei steigender Graphendichte.

Abbildung 38 zeigt den in Abschnitt 6.4 vorhergesagten Effekt einer deutlichen Steigerung der eingesparten Schnittberechnungen beim erfolgreichen Einsatz Maximaler Adjazenzordnungen. Dies war bedingt durch die Tatsache, dass schwach zusammenhängende Knotenpaare hier zuerst separiert werden und durch spätere, sie erneut trennende, höherwertige Schnitte Einsparungen möglich gemacht werden. Die absolute Zahl der erzielten Einsparungen bleibt mit 3% jedoch niedrig.

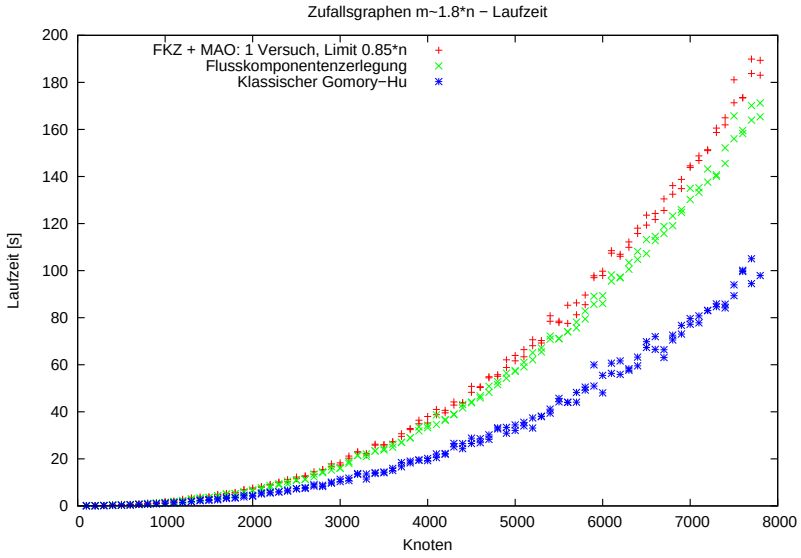


Abbildung 34: Laufzeitentwicklung auf wachsenden Zufallsgraphen

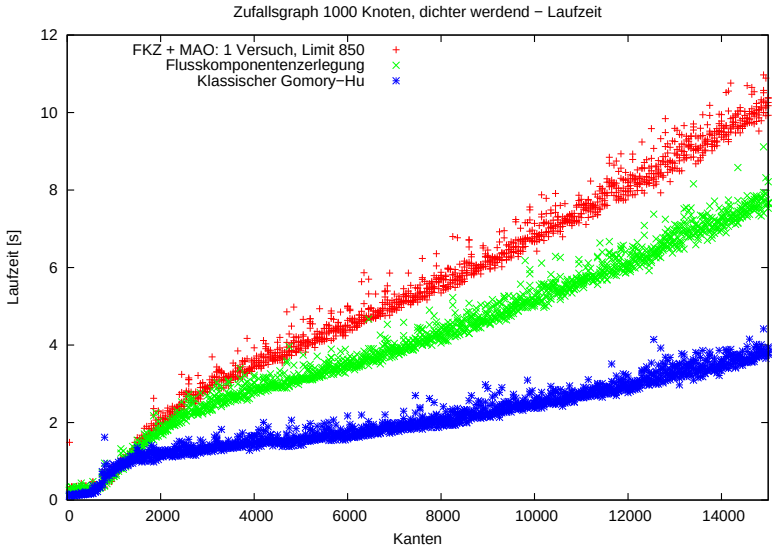


Abbildung 35: Laufzeiten auf Zufallsgraphen mit wachsender Dichte

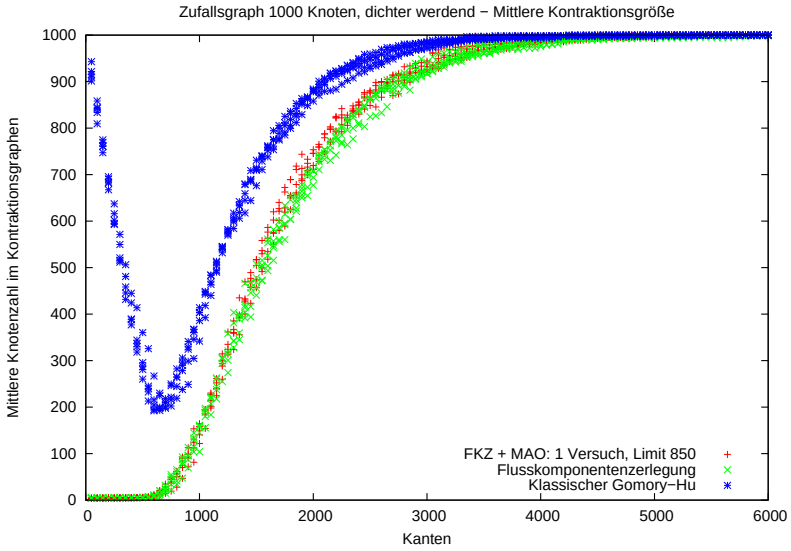


Abbildung 36: Mittlere Kontraktionsgraphgröße in Zufallsgraphen mit wachsender Dichte

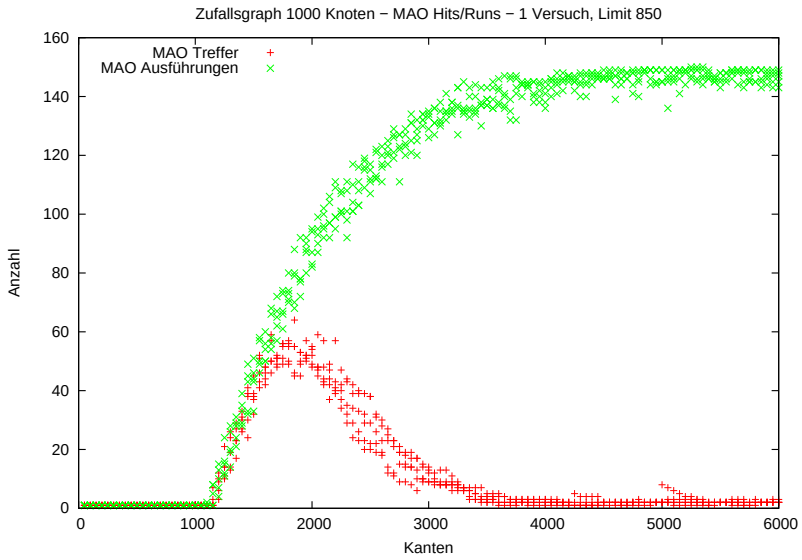


Abbildung 37: Trefferquote Maximaler Adjazenzordnungen in Zufallsgraphen mit wachsender Dichte

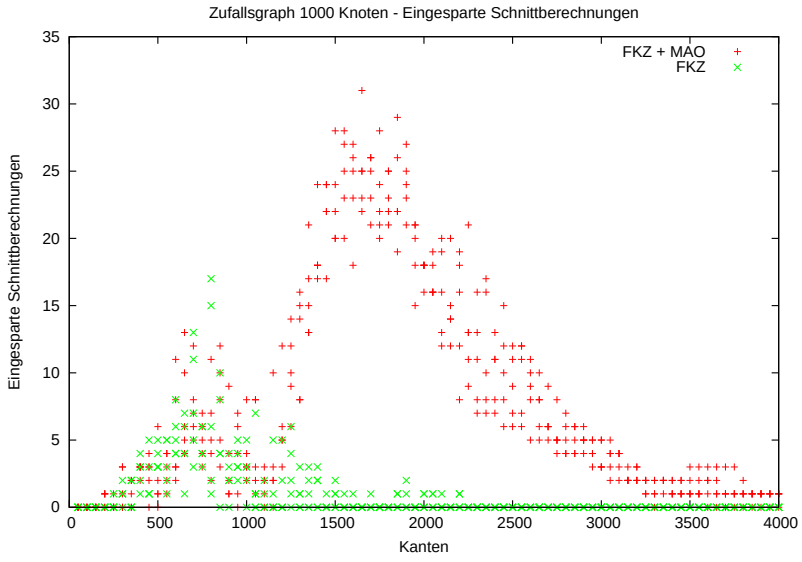


Abbildung 38: Eingesparte Schnittberechnungen gegenüber Gomory-Hu in Zufallsgraphen mit wachsender Dichte

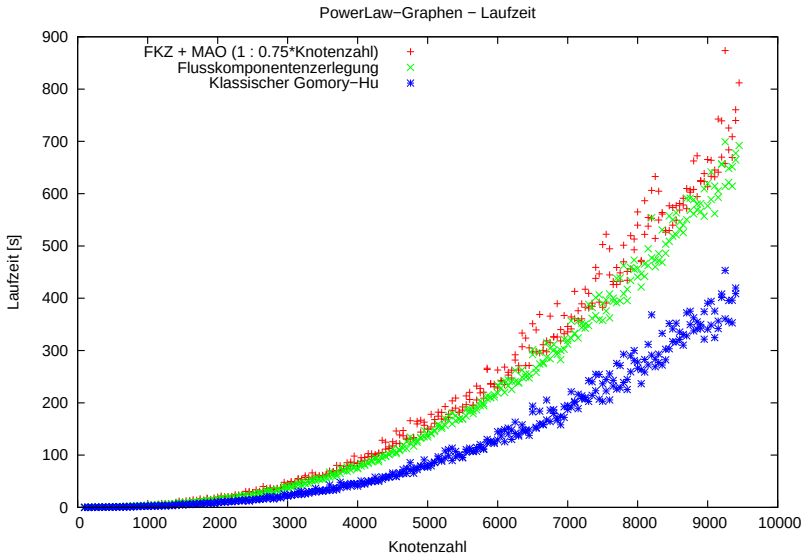


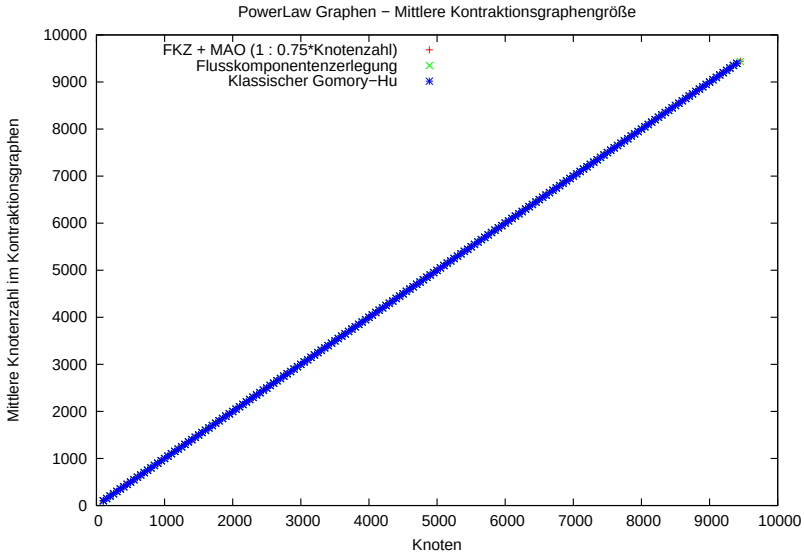
Abbildung 39: Laufzeitentwicklung auf PowerLaw-Graphen

8.4.3. Experimente Power-Law-Graphen

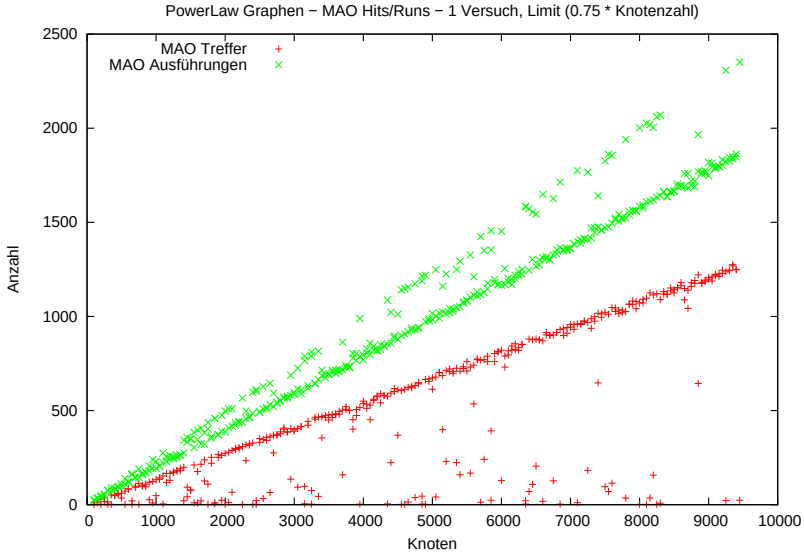
Auch die Power-Law-Graphen tendieren, aufgrund ihrer Gradverteilung, zu sternförmigen Gomory-Hu-Bäumen. Die Algorithmen mit Flusskomponentenzerlegung können hier ihre Vorteile nicht ausspielen und benötigen durch den von ihnen betriebenen Mehraufwand wesentlich mehr Zeit als der Gomory-Hu-Algorithmus. Dabei bleibt das Verhältnis der Laufzeiten im untersuchten Bereich konstant. Dies zeigt, dass der Aufwand je Flussberechnung proportional zum (jeweils linearen) Zerlegungs- und Verwaltunsaufwand der Flusskomponentenzerlegung wächst.

Existieren einzelne vom Rest getrennte Knoten, so können diese den Einsatz Maximaler Adjazenzordnungen stören und sämtliche Berechnungen mit einem Knotenpaar aus dieser sehr kleinen Knotenmenge enden. Gibt es jedoch keine einzelnen Abweicher, so ist es durch die Gradverteilung der Power-Law-Graphen möglich, relativ viele Maximale Flüsse mittels Maximaler Adjazenzordnungen zu berechnen, da die Zahl der Knoten mit minimalem Grad sehr groß ist.

Abbildung 39 zeigt die Laufzeitentwicklung der Algorithmen auf Power-Law-Graphen mit einer Kerngröße von 3 und jeweils 3 neuen Kanten pro hinzugefügtem Knoten (vgl. Abschnitt 8.2.2). Die völlige Sternform der entstehenden Gomory-Hu-Bäume ist sehr gut in Abbildung 40(a) erkennbar, in der die Mittlere Kontraktionsgraphengröße für alle getesteten Algorithmen jeweils exakt der Größe des Ausgangsgraphen entspricht. Ein solcher Verlauf ist nur durch ausnahmlose Berechnung von Gomory-Hu-Bäumen der Tiefe Eins erreichbar. Abbildung 40(b) zeigt die vergleichsweise hohen Verwertungsmöglichkeiten von Ergebnissen aus Maximalen Adjazenzordnungen.



(a) Mittlere Knotenzahl Kontraktionsgraph



(b) MAO-Erfolgsquote

Abbildung 40: Verhalten auf PowerLaw-Graphen

8.4.4. Experimente Kreisgraphen

Im nächsten Experiment sollten die Algorithmen auf verschiedenen Kreisgraphen getestet werden. Dazu wurden sie zuerst auf Kreisen mit uniformem und randomisiertem Gewicht ausgeführt. Den Erwartungen entsprechend, kommt ein uniform gewichteter Kreis dabei der Implementierung mit Flusskomponentenzerlegung sehr entgegen. Hier muss jeder Maximalfluss sämtliche Kanten überqueren und zerlegt den Graphen in Flusskomponenten der Größe Eins. In Phase 1 des Algorithmus wird somit nur eine einzige Berechnung durchgeführt und sämtliche weiteren Kontraktionsgraphen besitzen nur noch exakt drei Knoten. Abbildung 41(a) zeigt die Laufzeitentwicklung.

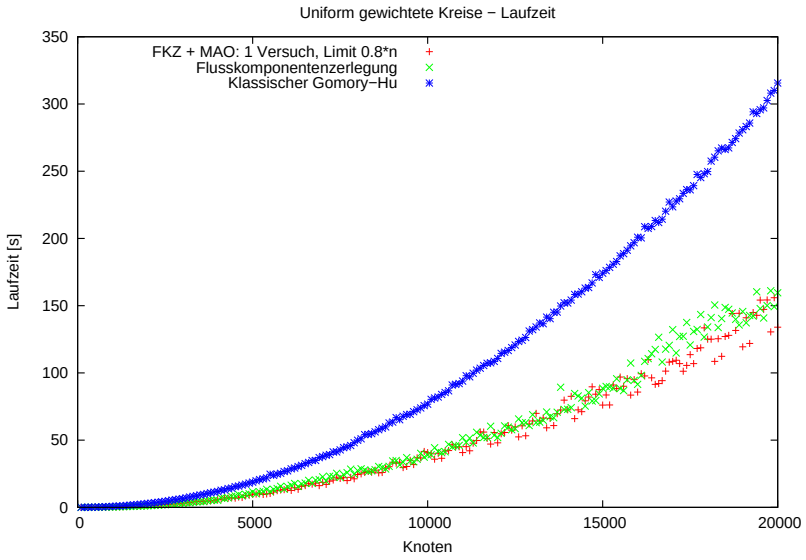
Die randomisierte Gewichtung der Kreiskanten wirkt diesem Verlauf entgegen. Eine Mehrfachzerlegung ist nun nur noch möglich, falls mehrere Kanten den selben Gewichtswert erhalten haben. Abbildung 41(b) demonstriert die Auswirkungen auf die Laufzeitentwicklung.

Interessant ist der Vergleich der Mittleren Kontraktionsgraphengröße beider Experimente in Abbildung 42. Auffällig ist zunächst die erwartete, beinahe konstante mittlere Knotenzahl für die Implementierung mit Flusskomponentenzerlegung auf den uniform gewichteten Graphen. Im starken Kontrast dazu steht die der Größe des Eingabegraphen entsprechende Entwicklung für den Gomory-Hu-Algorithmus. Werden diese Ergebnisse mit denen auf randomisiert gewichteten Kreisen verglichen, fällt auf, dass dort nicht nur die Flusskomponentenalgorithmen sondern auch die Werte des Gomory-Hu-Algorithmus einen anderen Verlauf zeigen. Erstere leiden unter der nun nur noch selten gegebenen Möglichkeit der Mehrfachzerlegung von Flüssen.

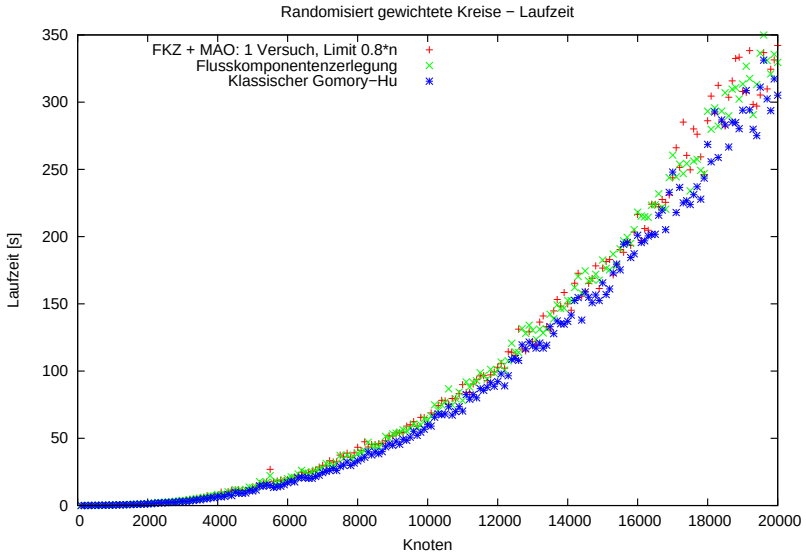
Das abweichende Verhalten des Gomory-Hu-Algorithmus liegt an der Eigenart der Implementierung für die Bipartition die erste gefundene Schnittmenge auszuwählen. Diese entspricht bei den uniformen Kreisen grundsätzlich dem Blattschnitt des Quellknotens, wodurch hier ein sternförmiger Gomory-Hu-Baum entsteht (erkennbar an der Äquivalenz der Größen von Eingabegraph und Kontrationsgraphen). Bei randomisiert gewichteten Kreisen ist ein solcher Blattschnitt nur noch zufällig möglich, wodurch balanciertere Bipartitionen entstehen. Dies führt zu einem größeren Durchmesser des Gomory-Hu-Baums und kleineren Kontraktionsgraphen.

Auch wenn die Auswahl der erstbesten Schnittmenge in diesem Fall nicht zum bestmöglichen Verlauf geführt hat, gehört es doch zu den großen Vorteilen der Gomory-Hu-Implementierung, nicht wie die Algorithmen der Flusskomponentenzerlegung den gesamten Flussverlauf analysieren zu müssen.

Abschließend lassen sich die bei der Kontraktionsgraphengröße unterschiedlichen Verläufe der Varianten mit und ohne die Bildung Maximaler Adjazenzordnungen betrachten. Erstere neigt zu deutlich größeren Kontraktionsgraphen. Dieses Phänomen ist der Tatsache geschuldet, dass Maximale Adjazenzordnungen in erster Linie Minimale Blattschnitte liefern. Lässt sich der abgeleitete Fluss nicht noch weiter aufteilen, hält dies den Durchmesser des Gomory-Hu-Baums klein und führt zu größeren Kontraktionsgraphen. Der Effekt ist auch in anderen Experimenten sichtbar.

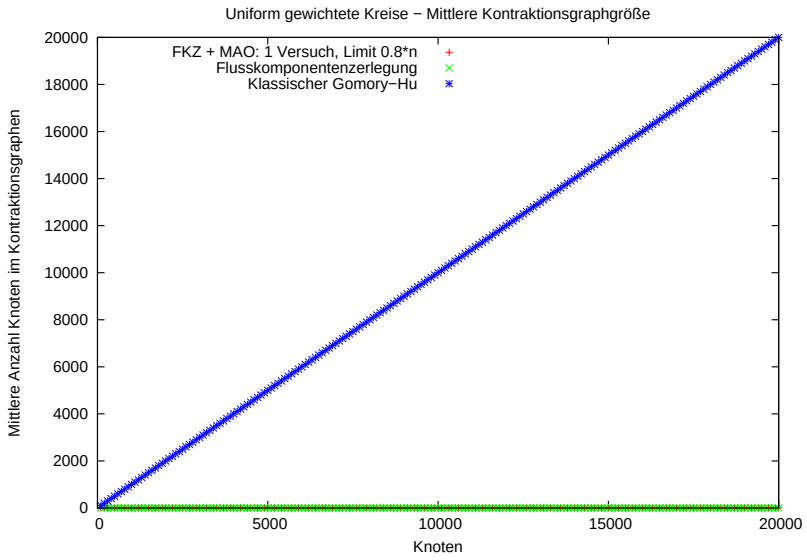


(a) uniform gewichtet

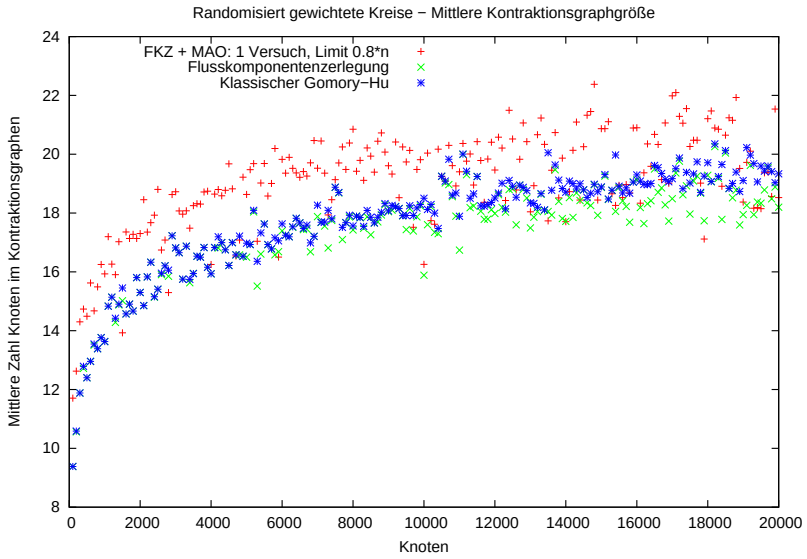


(b) randomisiert gewichtet

Abbildung 41: Laufzeitentwicklung auf uniform und randomisiert gewichteten Kreisen



(a) uniform gewichtet



(b) randomisiert gewichtet

Abbildung 42: Mittlere Kontraktionsgraphengröße auf uniform und randomisiert gewichteten Kreisen

Der mit wachsender Kreisgröße zunehmende Vorteil der nur Flusskomponentenzerlegung nutzenden Implementierung kann von einer Intervall-Begrenzung der als Kantengewicht eingesetzten Ganzzahlwerte herrühren. Mit wachsender Kantenzahl steigt so die Wahrscheinlichkeit von Wiederholungen und damit einer möglichen Mehrfachzerlegung.

Experiment Kreiskurzschlüsse Nach den Experimenten zu wachsenden Kreisgraphen, wurden einem bestehenden, randomisiert gewichteten Kreis eine steigende Zahl von Kurzschlüssen hinzugefügt. In diesem Experiment liegen alle Implementierungen nah beinander. Nach einem zunächst steilen Anstieg, nehmen die Laufzeiten mit wachsender Dichte später nur noch langsam und linear zu. Dabei sind die Ergebnisse aller Implementierungen starken Schwankungen unterworfen.

Auffällig ist hier die, im Gegensatz zu den verwandten Zufalls- oder `regulargen`-Graphen, sehr ähnliche Laufzeitentwicklung der verschiedenen Implementierungen. Dabei konnten jedoch weder zahlreiche Schnitte eingespart, noch erfolgreich Maximale Adjazenzordnungen eingesetzt werden. Auch die Leistung des Push-Relabel-Algorithmus unterscheidet sich prinzipiell nicht von den Vergleichsklassen.

8.4.5. Experimente Doppel-Kreis-Graphen

Wie sich in Abschnitt 8.4.1 bereits angedeutet hat, führen die auf Doppel-Kreis-Graphen durchgeführten Experimente, zu den besten festgestellten Ergebnissen für die in dieser Arbeit eingeführten Algorithmen. Insbesondere sind beim Einsatz Maximaler Adjazenzordnungen zusätzlich große Einsparungen feststellbar.

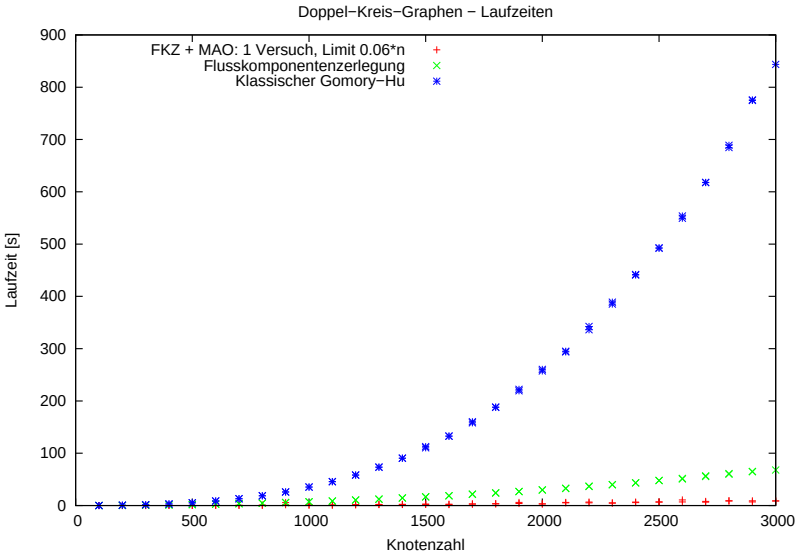
Ein eindeutiger Grund hierfür, ist die auf dieser Graphenklasse auch real angenommene worst-case-Laufzeitentwicklung des eingesetzten Push-Relabel-Algorithmus. Dadurch tritt das Einsparungspotential durch die neuen Algorithmen deutlich hervor und die zusätzliche Verwaltungsarbeit kann vernachlässigt werden.

Zusätzlich fallen die Doppel-Kreis-Graphen durch sehr gute Zerlegungseigenschaften mit vielen Komponenten je Flussberechnung auf.

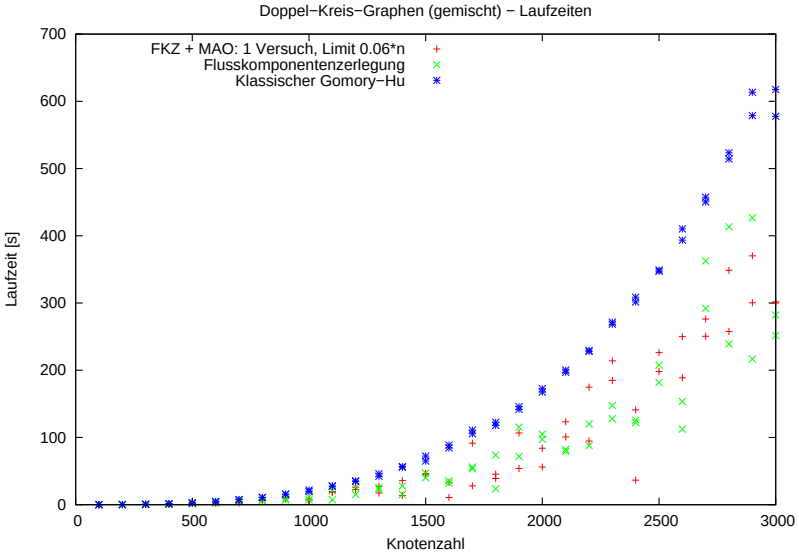
Auffällig ist eine Abhängigkeit der Laufzeiten von der Knotennummerierung während der Speicherung der erzeugten Graphen. Diese beeinflusst in den Implementierungen implizit die Auswahl der nächsten Flusspartner im Graphen. Auf den direkt aus dem Generator hervorgehenden Graphen, ist dabei ein wesentlich optimaleres und schwankungsfreieres Laufzeitverhalten feststellbar, als auf den ebenfalls getesteten Versionen mit nachträglich randomisierter Knotennummerierung. In beiden Fällen sind die Algorithmen mit Flusskomponentenzerlegung jedoch deutlich im Vorteil (vgl. Abbildung 43).

Beim Einsatz Maximaler Adjazenzordnungen wird das Ausführungslimit durch die sehr guten Zerlegungseigenschaften schnell unterschritten. Die wenigen gebildeten Adjazenzordnungen ersetzen jedoch gerade die aufwändigsten ersten Flussberechnungen und erreichen so das gesteckte Ziel.

Für die bei einer durch 3 teilbaren Knotenzahl aus dem `dblcyclegen` entstehenden 'anomalen' Graphen (siehe Abschnitt 8.2.1) konnte kein abweichendes Verhalten festgestellt werden.

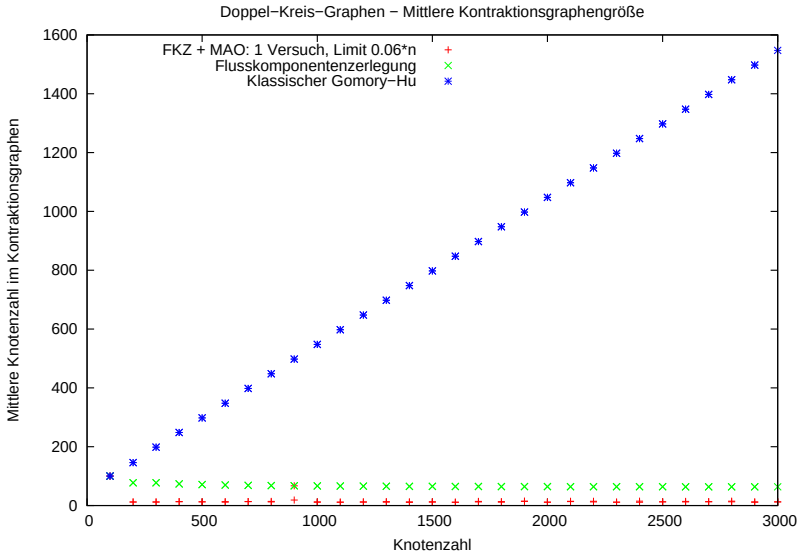


(a) Generatorausgabe

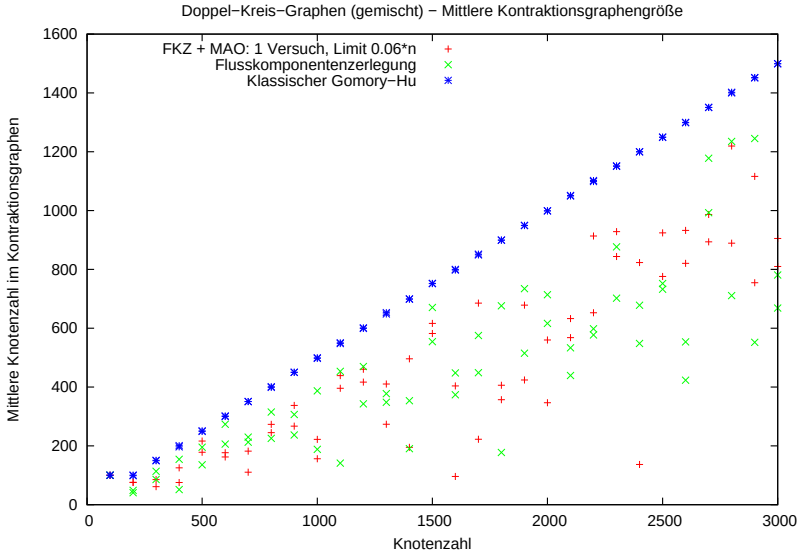


(b) Knoten-IDs gemischt

Abbildung 43: Laufzeitentwicklung auf Doppel-Kreis-Graphen



(a) Generatorausgabe



(b) Knoten-IDs gemischt

Abbildung 44: Mittlere Kontraktionsgraphengröße auf Doppel-Kreis-Graphen

Werden die Ergebnisse mit denen von Graphen des `regulargen` Generators oder zweidimensionalen Torus-Gittern verglichen, welche ebenso aus zwei disjunkten Hamiltonkreisen bestehen, so ist keine Ähnlichkeit feststellbar. Bei beiden besteht eine starke Tendenz zu Dreifachzerlegungen in eine Mittelkomponente und Blattschnitte für den Quell- und Zielknoten eines Flusses, welche wiederum in sternförmigen Gomory-Hu-Bäumen münden. Eine solche Entwicklung wird bei den Doppel-Kreis-Graphen durch die kleine, aber sehr wirkungsvolle, Anzahl von Kanten mit abweichendem Gewicht verhindert. Zusätzlich wächst die mittlere Push-Relabel-Laufzeit auf den Graphen der Vergleichsklassen wesentlich langsamer (vgl. Abschnitt 8.4.1). Die Kombination dieser Faktoren führt dort zu einem weit besseren Abschneiden des klassischen Gomory-Hu-Algorithmus.

Die Kombination aus großem Durchmesser des Eingabegraphen und einer sehr gleichförmigen, jedoch von einigen wenigen Abweichern geprägten, Kantengewichtung scheint die das Abschneiden der auf Flusskomponentenzerlegung basierenden Implementierungen sehr zu fördern.

8.4.6. Experimente Tree-Seeded-Graphen

In den Experimenten auf Tree-Seeded-Graphen liegen die Laufzeiten aller getesteten Implementierungen eng zusammen. Sowohl die Flusskomponentenzerlegung einsetzenden Kandidaten als auch der klassische Gomory-Hu-Algorithmus können vom Layout des Graphen profitieren.

Abbildung 47(a) stellt die Größe des durchschnittlichen Kontraktionsgraphen dar. Bei Verzicht auf Maximale Adjazenzordnungen, kann hier ein wachsender Abstand zwischen dem Gomory-Hu-Algorithmus und der Implementierung mit Flusskomponentenzerlegung festgestellt werden. Dieser beträgt zuletzt mehr als ein Drittel des Wertes des Gomory-Hu-Algorithmus.

Werden Maximale Adjazenzordnungen verwendet, zieht dies eine deutlich gesteigerte und stark schwankende durchschnittliche Knotenzahl in den Kontraktionsgraphen nach sich. Ein ähnlicher Effekt konnte bereits bei den randomisierten Kreisgraphen beobachtet werden. Gleichzeitig erhöht sich jedoch ebenso die Zahl eingesparter Schnittberechnungen (vgl. Abbildung 47(b)). Dies deckt sich mit den in Abschnitt 6.4 aufgestellten Erwartungen und trat auch bereits im Experiment auf dichter werdenden Zufallsgraphen auf.

8.4.7. Experimente Cluster-Grid-Graphen

Für Cluster-Grid-Graphen wurden zum Einen Experimente mit einer reinen Gitter-Struktur, ohne Zufallseinfluss, durchgeführt und in einem weiteren Experiment an den Kreuzungspunkten des Gitters Cluster aus kleinen Zufallsgraphen eingesetzt deren Knotenzahl kontinuierlich erhöht wurde.

Die reinen Gridstrukturen, besonders in geschlossener Form wie Ringe und Tori, sind für Mehrfachzerlegung prinzipiell sehr gut geeignet, besitzen beim Einsatz der Flusskomponentenzerlegung jedoch einen sehr starken Hang zu Blattschnitten von sowohl Quell- als auch Zielknoten und streben somit zu sternförmigen Gomory-Hu-Bäumen.

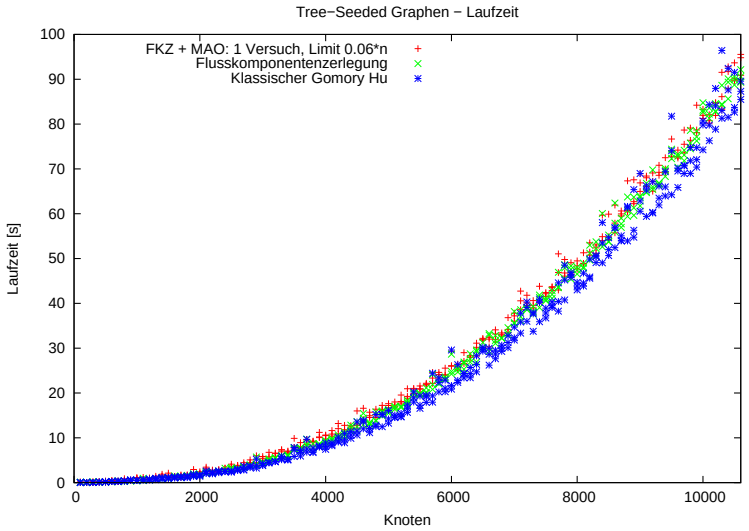


Abbildung 45: Laufzeitentwicklung auf Tree-Seeded-Graphen

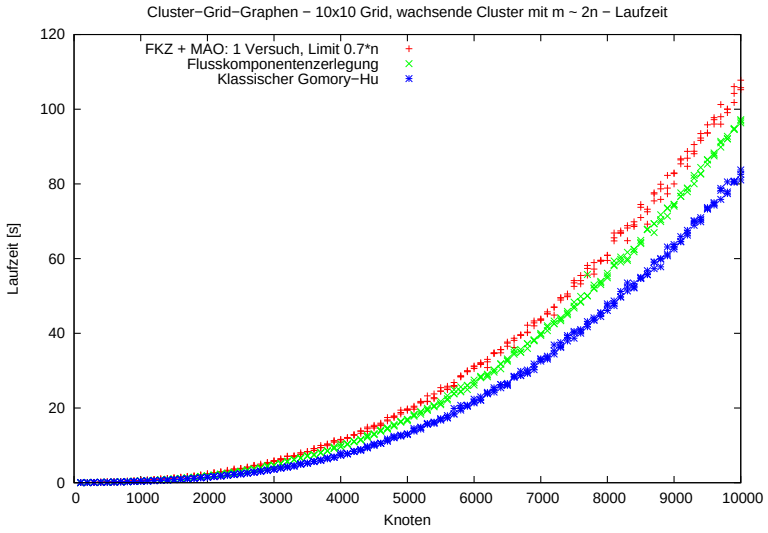
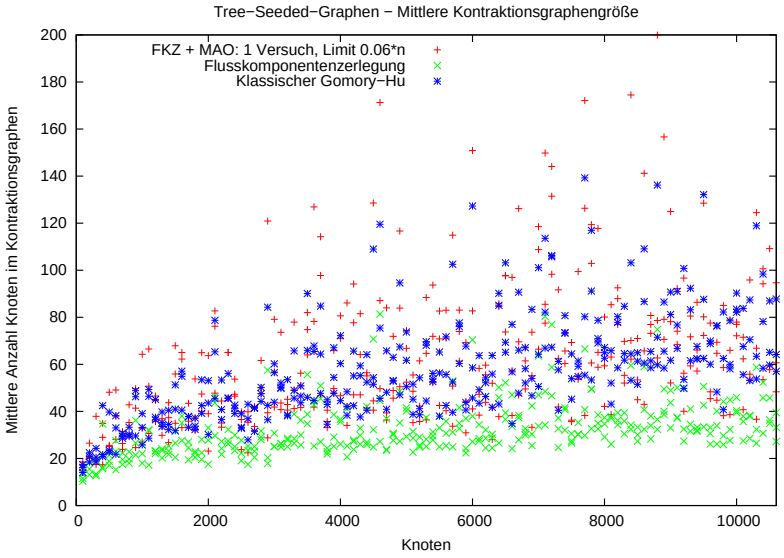
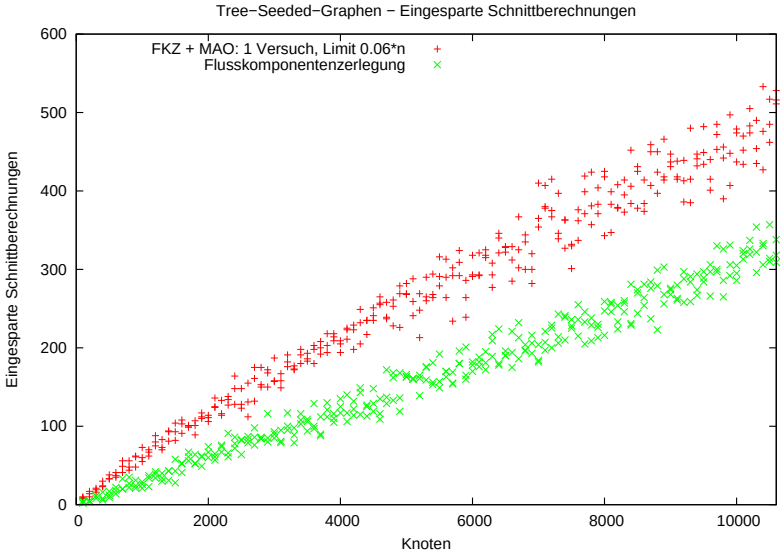


Abbildung 46: Laufzeitentwicklung auf Cluster-Grid-Graphen bei wachsenden Clustern

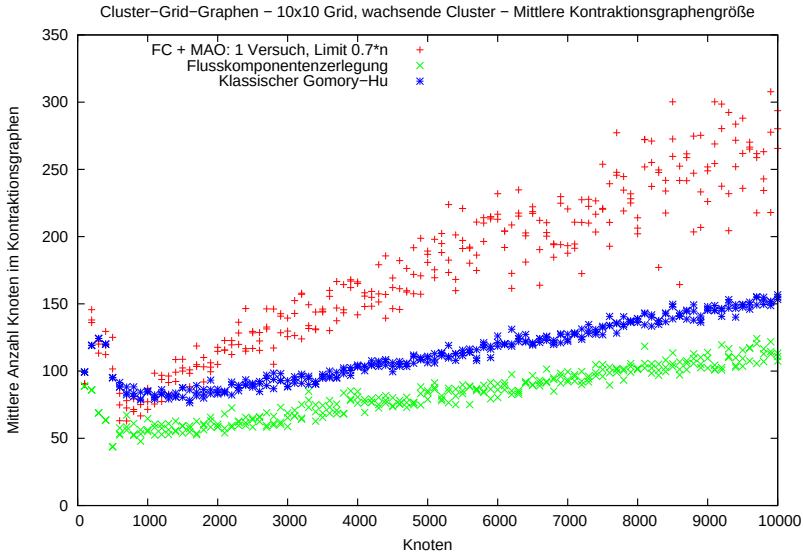


(a) Mittlere Kontraktionsgraphengröße

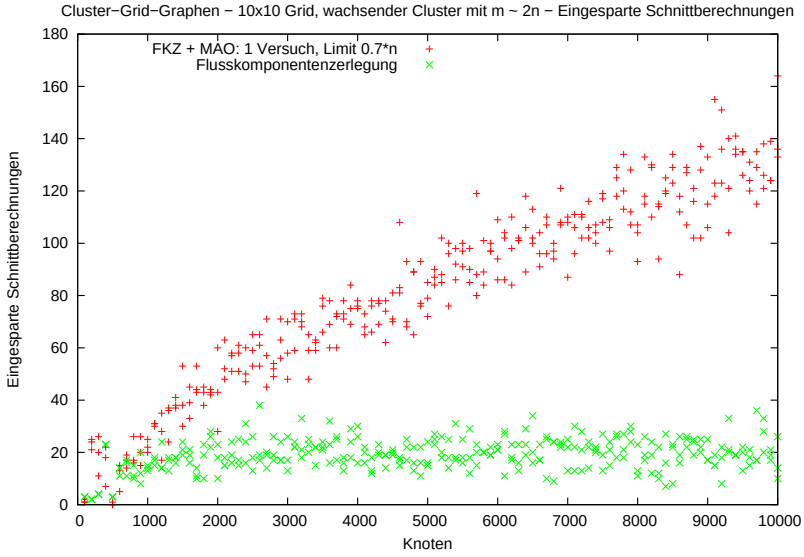


(b) Eingesparte Schnittberechnungen

Abbildung 47: Verhalten auf Tree-Seeded-Graphen



(a) Mittlere Kontraktionsgraphengröße



(b) Eingesparte Schnittberechnungen

Abbildung 48: Verhalten auf Cluster-Grid-Graphen bei wachsenden Clustern

Dies verhindet somit ein Schrumpfen der Kontraktionsgraphen. Zusätzlich erleichtern die Blattschnitte dem klassischen Gomory-Hu-Algorithmus die schnelle Identifikation geeigneter Bipartitionsmengen, wogegen die Flusskomponentenzerlegung grundsätzlich den Fluss über den gesamten Graphen auswerten muss. Die Folge ist ein äußerst schlechtes Abschneiden letzterer, mit stets mehr als doppelter Laufzeit des Gomory-Hu-Algorithmus.

Das Experiment zu wachsenden Clustergrößen bestätigt größtenteils die bereits in anderen Versuchen gemachten Beobachtungen. Abbildung 46 zeigt die Laufzeitentwicklung aller Algorithmen. Der Mehraufwand der Flusskomponentenzerlegung fällt hier nicht so stark aus, wie auf anderen Graphen. Wird das Verhalten mit und ohne Einsatz Maximaler Adjazenzordnungen verglichen, so zeigt sich erneut eine Steigerung der durchschnittlichen Kontraktionsgraphengröße, bei gleichzeitiger Erhöhung der Anzahl eingesparter Schnittberechnungen (vgl. Abbildung 48).

8.5. Abschließende Auswertung

Zusammengefasst betrachtet, sind die erzielten Ergebnisse ernüchternd. Auf den getesteten Graphen zeigt sich, durch die unerwartet gute Entwicklung der Laufzeiten der Maximalflussalgorithmen, wenig Sparpotential durch verkleinerte Kontraktionsgraphen. Ebenso wenig wirken sich somit die erzielten Einsparungen von Flussberechnungen aus, deren Anzahl in den Experimenten graphenabhängig zwischen nur 0 bis 5% schwankte. Stattdessen fallen die notwendigen Mehrkosten der Flusskomponentenzerlegung ins Gewicht. Somit wird die Algorithmusleistung sehr abhängig vom Aufteilungsprozess des Graphen während der Berechnung. Besonders kritisch ist ein Laufzeitvergleich auf Graphklassen, welche zu Gomory-Hu-Bäumen mit geringem Durchmesser tendieren, da hier wenig bis keine Absenkung der Knotenzahl in den Kontraktionsgraphen möglich ist und auch keine Berechnungen überflüssig werden.

Wie erwartet, ist ebenso eine starke Variabilität in den Kantengewichten des Eingabegraphen einer Mehrfachzerlegung abträglich. Es zeigte sich allerdings zusätzlich, dass eine völlige Gleichgewichtung sehr häufig zu Szenarien führt, in welchen vor allem Blattschnitte für den Quell- und den Zielknoten eines Flusses identifiziert werden, wodurch der Gomory-Hu-Baum vermehrt zur Sternform tendiert. Positiv wirkte hier die Präsenz einiger weniger im Gewicht abweichender Kanten. Solche Graphen zeigten eine höhere Zahl an Komponenten je Maximalflussberechnung und größere Durchmesser im entstehenden Gomory-Hu-Baum.

Beim Einsatz Maximaler Adjazenzordnungen konnten auch auf für diese Form der Flussberechnung vergleichweise gut geeigneten Graphen, recht geringe Verwertungsquoten der gefundenen Flüsse erreicht werden. Zusätzlich zeigten sich die Gesamtlaufzeiten auch bei erfolgreicher Verwendung Maximaler Adjazenzordnungen meist erhöht. Eine Erklärung hierfür liegt unter Anderem im in Abschnitt 8.4.1 getroffenen Laufzeitvergleich zur Push-Relabel-Implementierung. Des Weiteren kann durch ihre Blattschnittergebnisse eine Tendenz zu im Mittel größeren Kontraktionsgraphen beobachtet werden. Positiv zeigte sich jedoch ihre Wirkung auf die Anzahl verhinderbarer Maximalflussberechnungen, da durch sie die schwach zusammenhängenden Knoten des Graphen bereits zu Beginn des Algorithmusverlaufs identifiziert werden können.

9. Zusammenfassung und Ausblick

Nach einer Einführung in bekannte Methoden zur Berechnung von Gomory-Hu-Bäumen, wurden in dieser Arbeit verschiedene Techniken vorgeschlagen, um ihre Bestimmung zu beschleunigen. Da die Gesamtlaufzeit entscheidend vom Aufwand für die eingesetzten Maximalflussberechnungen abhängt, sind diese besonderes Ziel der Optimierungsansätze.

Abschnitt 4 stellt Maximale Adjazenzordnungen und ihre Anwendungsmöglichkeit zur Berechnung des Maximalen Flusses zwischen einem nicht frei wählbaren Knotenpaar vor. Diese sollten eingesetzt werden, um besonders zu Beginn der Algorithmen Maximalflussberechnungen zu ersetzen. Zu diesem Zeitpunkt ist die Problemgröße für die Maximalflussalgorithmen noch besonders umfangreich und gleichzeitig eine hohe Verwertbarkeit der durch die Adjazenzordnungen gefundenen Flüsse wahrscheinlich.

Als weiterer Vorschlag wurde in Abschnitt 5 mit der Flusskomponentenzerlegung ein Verfahren eingeführt, welches darauf abzielt, nicht länger nur eine sondern sämtliche durch einen Maximalen Fluss gegebenen Aufspaltungsmöglichkeiten der Knotenmenge des betrachteten Graphen auszunutzen. Durch ein solches Vorgehen lässt sich die durchschnittliche Größe der während des Algorithmenverlaufs gebildeten Kontraktionsgraphen senken, und somit die auf ihnen durchzuführenden Berechnungen beschleunigen.

Wie in Abschnitt 6.4 gezeigt wurde, ist es, unter Ausnutzung der für dieses Verfahren anzusammelnden Zusatzinformationen, ebenso möglich bestimmte Berechnungen als unnötig zu identifizieren. Das Ausmaß dieser Einsparungen ist jedoch abhängig vom Eingabegraphen und individuellen Algorithmusverlauf.

Für die Flusskomponentenzerlegung wurden Korrektheit, Anwendbarkeit und genauere Eigenschaften in Abschnitt 6 eingehend untersucht.

Um die Wirkung der vorgestellten Verfahren zu analysieren, wurden sie implementiert und ihr Verhalten auf einer Auswahl verschiedener Graphklassen mit dem ebenfalls implementierten Gomory-Hu-Algorithmus verglichen. Dabei zeigte sich, dass die als Zielsetzung zu beeinflussenden Parameter der durchschnittlichen Knotenzahl von Kontraktionsgraphen und insgesamt durchgeführten Maximalflussberechnungen meist erfolgreich beeinflusst werden konnten, dies für gewöhnlich jedoch nicht in Laufzeitvorteile umsetzbar war.

Als primärer Grund hierfür, erwies sich die starke Abweichung zwischen worst-case- und average-case-Laufzeiten der eingesetzten gezielten Maximalflussalgorithmen. Auf den getesteten Graphklassen konnte eine äußerst gutmütige Laufzeitentwicklung beobachtet werden, welche sich häufig proportional zum für die in dieser Arbeit vorgestellten Methoden notwendigen Zusatzaufwand zeigte.

Eine detaillierte Auswertung des Verhaltens der verschiedenen Implementierungen auf den ausgewählten Graphenklassen zeigt Abschnitt 8.4. Besondere Einflussfaktoren werden in Abschnitt 8.5 zusammengefasst.

In Anbetracht der experimentell festgestellten Ergebnisse, ist ein Einsatz der vorgestellten Methoden in ihrer jetzigen Form nur dann empfehlenswert, wenn vorteilhafte Grapheneigenschaften oder hohe Laufzeiten bei der Berechnung Maximaler Flüsse festgestellt werden können.

Das von der Flusskomponentenzerlegung eingesetzte Schema zur Identifikation unabhängiger Komponenten bietet wenig weiteres Optimierungspotential. Wird auf die Identifikation einer größtmöglichen Zahl von Komponenten verzichtet, existieren geringfügig schnellere Algorithmen zur Aufzählung nur einiger kreuzungsfreier Minimaler s-t-Schnitte aus einem Maximalen s-t-Fluss. Da die Zahl der ausgenutzten Komponenten je Flussberechnung jedoch einen Schlüsselfaktor in der durch die Fluskomponentenzerlegung erhofften Beschleunigung darstellt, ist hier mit keiner Verbesserung der Ergebnisse zu rechnen.

Schließlich ist denkbar die Zweiphasigkeit des Algorithmus aufzuheben und insbesondere den Blattschnittbeziehungsgraphen L einzusparen.

Nach der Identifikation der Flusskomponenten eines Maximalen s-t-Flusses, werden diese dann nicht wie im vorgestellten Algorithmus statisch und linear im Relaxierten Gomory-Hu-Baum angeordnet. Stattdessen wird ein Komponentengraph konstruiert in dem die Flusskomponenten jeweils kontrahiert auftreten. Aus dem Gomory-Hu-Baum für diesen Ersatzgraphen lassen sich Informationen über die endgültige Positionierung der einzelnen Flusskomponenten im Relaxierten Gomory-Hu-Baum für G ableiten. Sie müssen somit nicht mehr später über den Einsatz Leerer Knoten bestimmt werden.

Da die übliche Anzahl an Flusskomponenten je Maximalflussberechnung vergleichsweise niedrig ist, kann der Ersatzgraph besonders leicht analysiert werden. Bei den in den Experimenten häufig auftretenden Zerlegungen in drei Komponenten lässt sich die einzige noch durchzuführende Maximalflussberechnung zum Beispiel durch einen simplen Vergleich von $\lambda(s,t)$ mit dem Grad des die Mittelkomponente zusammenfassenden Knotens ersetzen.

A. Literaturverzeichnis

[AKMO97] R. K. Ahuja et al. "Computational investigations of maximum flow algorithms". *European Journal of Operational Research*, 97(3):S. 509–542 (1997).

[AM99] S. R. Arikati und K. Mehlhorn. "A correctness certificate for the Stoer-Wagner min-cut algorithm". *Inf. Process. Lett.*, 70(5):S. 251–254 (1999).

[BA99] A.-L. Barabási und R. Albert. "Emergence of Scaling in Random Networks". *Science*, 286(5439):S. 509–512 (1999).

[BHPT07] A. Bhalgat et al. "An $\tilde{O}(mn)$ Gomory-Hu tree construction algorithm for unweighted graphs". *eingereicht für 39th ACM Symposium on Theory of Computing (STOC 2007)* (2007). URL `http://people.csa.iisc.ernet.in/debmalya/papers/ghtree.pdf`.

[Bri05] M. Brinkmeier. "A simple and fast Min-Cut Algorithm". *Lecture Notes on Computer Science*, 3623:S. 317–328 (2005). URL `http://www.tu-ilmenau.de/fakia/fileadmin/template/startIA/afs/publications/mbrinkme/mincut.pdf`.

[CH90] C.-K. Cheng und T. C. Hu. "Ancestor Tree for Arbitrary Multi-Terminal Cut Functions". In *Proceedings of the 1st Integer Programming and Combinatorial Optimization Conference*, S. 115–127. University of Waterloo Press, Waterloo, Ont., Canada, Canada (1990). ISBN 0-88898-099-X.

[DM89] U. Dergis und W. Meier. "Implementing Goldberg's Max–Flow–Algorithm – A Computational Investigation". *Methods and Models of Operations Reseach*, (33):S. 383–403 (1989). URL `http://www.springerlink.com/index/U5226522323V5868.pdf`.

[GGP+99] A. V. Goldberg et al. "Cut Tree Algorithms". In *Symposium on Discrete Algorithms*, S. 376–385 (1999). URL `http://citeseer.ist.psu.edu/295887.html`.

[GH61] R. E. Gomory und T. C. Hu. "Multi-Terminal Network Flows". *Journal of the Society for Industrial and Applied Mathematics*, 9(4):S. 551–570 (1961). URL `http://links.jstor.org/sici?sici=0368-4245%28196112%299%3A4%3C551%3AMNF%3E2.0.CO%3B2-9`.

[GLG] "Graphengeneratoren von Andrew Goldberg". URL `http://www.cs.princeton.edu/~kt/cut-tree/experiments/fam.tar.gz`.

[GR98] A. V. Goldberg und S. Rao. "Beyond the flow decomposition barrier". *Journal of the ACM*, 45(5):S. 783–797 (1998).

[GT88] A. V. Goldberg und R. E. Tarjan. "A New Approach to the Maximum-Flow Problem". *Journal of the ACM*, 35(4):S. 921–940 (1988).

[Gus90] D. M. Gusfield. "Very Simple Methods for All Pairs Network Flow Analysis". *SIAM Journal on Computing*, (19):S. 143–155 (1990).

[Hu70] T. C. Hu. *Integer programming and network flows*. Addison-Wesley (1970).

[NI92a] H. Nagamochi und T. Ibaraki. "Computing edge connectivity in multigraphs and capacitated graphs.". *SIAM Journal on Discrete Mathematics*, 5(1):S. 54–66 (1992).

[NI92b] H. Nagamochi und T. Ibaraki. "A linear time algorithm for finding a sparse k-connected spanning sub-graph of a k-connected graph.". *Algorithmica*, 7(5&6):S. 583–596 (1992).

[Pea05] D. J. Pearce. "An Improved Algorithm for Finding the Strongly Connected Components of a Directed Graph". Technischer Bericht (2005). URL `http://www.mcs.vuw.ac.nz/~djp/files/P05.pdf`.

[PQ80] J.-C. Picard und M. Queyranne. "On the structure of all minimum cuts in a network and applications". *Mathematical Programming Study*, 13:S. 8–16 (1980).

[SW97] M. Stoer und F. Wagner. "A simple min-cut algorithm". *Journal of the ACM*, 44(4):S. 585–591 (1997). URL `http://www.cs.dartmouth.edu/~rahul/Teaching/stoerwagner-mincut.pdf`.

[THS99] P. A. Tucker, T. C. Hu und M. T. Shing. "Min Cuts Without Path Packing". Technischer Bericht, University of California, San Diego (1999). URL `http://www.cse.ucsd.edu/Dienst/UI/2.0/Describe/ncstrl.ucsd_cse/CS1999-0625`.

[TS92] K. Thulasiraman und M. N. S. Swamy. *Graphs: Theory and Algorithms*. John Wiley & Sons, Inc., New York, NY, USA (1992). ISBN 0-471-51356-3.

B. Abbildungsverzeichnis

C. Algorithmenverzeichnis

D. Thesen

1. Die vorgestellte Methode der Flusskomponentenzerlegung arbeitet korrekt und liefert Gomory-Hu-Bäume für beliebige Eingabegraphen.

2. Gegenüber dem klassischen Gomory-Hu-Algorithmus können sowohl Schnittberechnungen als auch Kontraktionen eingespart und die mittlere Kontraktionsgraphengröße gesenkt werden. Die Ausprägung dieser Einsparungen ist stark abhängig vom gewählten Eingabegraphen und dem Algorithmusverlauf.

3. Maximale Adjazenzordnungen können im frühen Stadium des Algorithmus eingesetzt werden, um Maximalflussberechnungen zu ersetzen.

4. Aufgrund schwieriger Dosierbarkeit, kaum bzw. nicht vorhandenen Laufzeitvorteilen zu modernen Maximalflussalgorithmen und der nicht sichergestellten Verwertbarkeit ihrer Ergebnisse, sind sie jedoch nur in speziellen Fällen erfolgsversprechend.

5. Die Anwendung Maximaler Adjazenzordnungen kann die durchschnittliche Größe der Kontraktionsgraphen erhöhen.

6. Wird der Einsatz Maximaler Adjazenzordnungen mit der Flusskomponentenzerlegung kombinert, kann dies die Zahl eingesparter Schnittberechnungen steigern.

7. Die durch beide Neuerungen erzielbaren wirklichen Laufzeitvorteile gegenüber dem klassischen Gomory-Hu-Algorithmus sind äußerst abhängig von der mittleren Leistung des eingesetzten Maximalflussalgorithmus und konnten in Experimenten nur für wenige, ausgewählte Graphklassen gezeigt werden.

8. Als vorteilhaft zeigten sich Graphen mit hohem Durchmesser und Kreisen. Ihre Kantengewichtung sollte wenig variieren, jedoch nicht uniform sein.

9. Der Gomory-Hu-Algorithmus ist besonders dann im Vorteil, falls der entstehende Gomory-Hu-Baum einen geringen Durchmesser annimmt.

Ilmenau, den 07. September 2007

E. Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Die Arbeit wurde bisher weder einer anderen Prüfungsbehörde vorgelegt, noch veröffentlicht.

Ilmenau, den 07. September 2007