Approximationsalgorithmen

Seminar im Sommersemester 2008

Sebastian Bauer, Wei Cheng und David Münch

Herausgegeben von

Martin Nöllenburg, Ignaz Rutter und Alexander Wolff

Institut für Theoretische Informatik

Fakultät für Informatik
Universität Karlsruhe

Inhaltsverzeichnis

Kapitel 1

SET COVER

David Münch

Zusammenfassung

In dieser Ausarbeitung möchte ich schrittweise vorstellen, wie man zu dem Problem Set Cover gute Approximationsalgorithmen, besonders mit Hilfe linearer Programme, konstruieren kann. Set Cover ist eines der 21 klassischen NP-vollständigen Probleme, die Karp 1972 definiert hat. Dabei geht es darum, eine günstigste Menge von Teilmengen zu finden, welche alle Elemente eines gegebenen Universums abdecken. Zuerst wird ein Greedyalgorithmus konstruiert. Anschließend wird Set Cover als lineares Programm modelliert und dessen Lösung nach geeigneten Verfahren gerundet. Im letzten Abschnitt wird das Primal-Duale Schema auf Set Cover angewendet, d.h. es wird eine gültige Lösung des linearen Programms gesucht, ohne dieses explizit zu lösen. Als Referenz dient hauptsächlich [Vaz01, Kapitel 2,13,14,15].

1.1 Einführung

Das dieser Ausarbeitung zugrunde liegende Problem ist Set Cover und wird hier zuerst formal definiert.

Problem 1.1. Set Cover

Gegeben: *Ein Universum U mit n Elementen, eine Menge $\mathcal{S} = \{S_1, ..., S_k\}$ von k Teilmengen $S_i \subseteq U, i = 1, ..., k$ und eine Kostenfunktion $c : \mathcal{S} \to \mathbb{Q}^+$.*

Gesucht: *Eine Teilmenge $C \subseteq \mathcal{S}$, die alle Elemente von U überdeckt, d.h. $\bigcup_{S \in C} S = U$, und dabei minimale Kosten $c(C) = \sum_{S \in C} c(S)$ hat.*

Beginnen wir mit einem Beispiel: Angenommen, es sind p Computerviren und q Teilzeichenketten von mehr als zwanzig aufeinanderfolgenden

Bytes aus schadhaftem Code bekannt, die nicht in gutartigem Code auftauchen. Muss dann nach allen q Teilzeichenketten gesucht werden, um einen Virenbefall nachzuweisen?
Die Antwort ist nein, denn definieren wir die p Viren als Elemente des Universums U und die q Teilzeichenketten als Teilmengen von U, so dass jede Teilmenge diejenigen Viren enthält, in denen die Teilzeichenkette vorkommt, dann hat man ein Set Cover Problem. Goldberg et al. [GGPS98] zeigten für ein Beispiel mit $p = 5000$ und $q = 9000$, dass es ausreicht nach 180 von 9000 Teilzeichenketten zu suchen, um Virenbefall nachzuweisen.

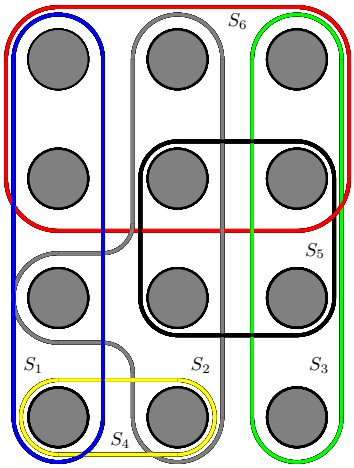

Abbildung 1.1: Instanz eines Set Cover Problems. Elemente des Universums sind als Kreise dargestellt, Teilmengen als Regionen.

Beispiel 1.1. *In Abbildung 1.1 decken die Mengen S_1, S_2 und S_3 alle Elemente des Universums ab und bilden bei konstanten Kosten ein minimales* Set Cover.

Das Problem Set Cover wurde bereits 1972 von Karp als $\mathcal{NP}$-vollständig nachgewiesen [Kar72]. Er reduzierte dabei nach folgendem Schema:

$$\text{SAT} \propto \text{CLIQUE} \propto \text{NODE (VERTEX) COVER} \propto \text{SET COVER}$$

Falls $\mathcal{P} \neq \mathcal{NP}$, sind $\mathcal{NP}$-vollständige Probleme nicht effizient und exakt lösbar. Das stellt uns wiederum vor die Herausforderung, wie wir diese Probleme, wenn nicht effizient und korrekt, dann zumindest effizient und fast korrekt lösen können. Ein genereller Lösungsansatz sind hier Approximationsalgorithmen, die hier ihre Anwendung finden. Ein erster Versuch soll der folgende Greedyalgorithmus sein.

1.2 Der Greedyalgorithmus

Ein einfacher Ansatz zur Lösung von Set Cover ist ein Greedyalgorithmus, der nacheinander die Mengen mit der geringsten Kosteneffizienz $\alpha(S)$ auswählt. Sie wird definiert durch die Kosten der Teilmenge, geteilt durch die Anzahl der noch nicht überdeckten Elemente von S, wobei C alle bereits überdeckten Elemente enthält, konkret: $\alpha(S) = \frac{c(S)}{|S-C|}$.

Algorithmus 1 : Greedy Set Cover Algorithmus

$C \leftarrow \emptyset$
while $C \neq U$ **do**
  $S_{min} \leftarrow \text{argmin}_S \alpha(S)$.
  $\forall e \in S_{min} - C$, setze $\text{price}(e) = \alpha(S_{min})$.
  $C \leftarrow C \cup S_{min}$.
Ausgabe der gewählten Mengen.

Beispiel 1.2. *Algorithmus 1 wählt nun im Beispiel in Abbildung 1.1, bei konstanten Kosten $c(S) \equiv 1$, immer die Teilmenge mit den meisten noch nicht überdeckten Elementen aus. D.h. hier nacheinander die Mengen S_6, S_2, S_3 und S_4. Diese Teilmengen decken zwar alle Elemente des Universums ab, bilden aber kein minimales* Set Cover *(vgl. Beispiel 1.1).*

Wir nummerieren die ausgewählten Elemente $e_m, m \in \{1, ..., n\}$ in der Reihenfolge, in der sie Algorithmus 1 zu C hinzufügt mit $e_1, ..., e_n$. Dann gilt folgendes Lemma:

Lemma 1.1. *Für jedes $m \in \{1, ..., n\}$ ist $price(e_m) \leq \frac{OPT}{n-m+1}$, wobei OPT die Kosten einer optimalen Lösung bezeichnet.*

Beweis. In jedem Schleifendurchlauf haben die übrig gebliebenen Mengen der optimalen Lösung maximal OPT Kosten. Darunter gibt es nach dem Schubfachprinzip mindestens eine Menge mit der Kosteneffizienz $\frac{\text{OPT}}{|U-C|}$. Im Schleifendurchlauf m, bei dem e_m hinzugenommen wird, enthält die Menge $U-C$ mindestens $n-m+1$ Elemente. Daraus folgt direkt das Lemma 1.1. □

Satz 1.1. *Algorithmus 1 ist ein Faktor-H_n-Approximationsalgorithmus für das* Set Cover *Problem, wobei $H_n = 1 + \frac{1}{2} + ... + \frac{1}{n} \leq \ln(n) + 1$. Er kann mit einer Laufzeit in $\mathcal{O}(\sum_{S \in \mathcal{S}} |S|)$ implementiert werden.*

Beweis. Die Gesamtkosten des Set Covers entsprechen $\sum_{k=1}^{n} price(e_k)$. Somit sind diese nach Lemma 1.1 maximal $(1 + \frac{1}{2} + ... + \frac{1}{n}) \cdot \text{OPT}$. Nach [CLR05, 35.3-3] ergibt sich die Laufzeit. □

1.3 Set Cover als lineares Programm

Bisher wurde Set Cover mit Mengen definiert. Nun wollen wir Set Cover zuerst als ganzzahliges lineares Programm (ILP) formulieren, dann die Ganzzahligkeit aufheben und das ILP zu einem linearen Programm (LP), dem primalen LP, umformen. Zu dem primalen LP wird das duale LP konstruiert und schließlich werden die Beziehungen untereinander aufgezeigt.

1.3.1 Ganzzahliges und primales lineares Programm

Beginnen wir mit dem ganzzahligen linearen Programm:

Minimiere

$$\sum_{S \in \mathcal{S}} c(S) x_S \tag{1.1}$$

unter den Nebenbedingungen

$$\sum_{S: e \in S} x_S \geq 1, \quad e \in U \tag{1.2}$$

$$x_S \in \{0,1\}, \quad S \in \mathcal{S} \tag{1.3}$$

Der Vektor $\mathbf{x} = (x_{S_1}, ..., x_{S_k})$ kann aufgrund Nebenbedingung (1.3) komponentenweise die Werte Null (d.h. Teilmenge nicht enthalten) oder Eins (d.h. Teilmenge enthalten) annehmen. Mit den Einzelkosten $c(S)$ multipliziert und über alle ausgewählten Teilmengen aufsummiert, sollen diese Gesamtkosten minimiert werden. Dabei gilt es die Nebenbedingung (1.2), dass jedes Element in einer gültigen Lösung einer Set Cover Instanz überdeckt wird, zu beachten.
Die Einschränkung $x_S \in \{0,1\}$ kann gelockert werden zu $x_S \geq 0$. Dann liegt kein ILP mehr vor, sondern ein lineares Programm. Das Aufheben der Ganzzahligkeitsbedingung nennt man Relaxation des ILP. Dieses so genannte primale lineare Programm lautet:

Minimiere

$$\sum_{S \in \mathcal{S}} c(S) x_S \tag{1.4}$$

unter den Nebenbedingungen

$$\sum_{S: e \in S} x_S \geq 1, \quad e \in U \tag{1.5}$$

$$x_S \geq 0, \quad S \in \mathcal{S} \tag{1.6}$$

Beispiel 1.3. *Mit der Kostenfunktion $c(S) = |S|$ erhält man zur Instanz aus Abbildung 1.2 folgendes LP:*

Minimiere

$$2x_1 + 2x_2 + 2x_3 \tag{1.7}$$

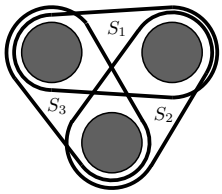

Abbildung 1.2: Instanz eines Set Cover Problems.

unter den Nebenbedingungen

$$\begin{aligned} x_1 + x_2 &\geq 1 && (1.8)\\ x_2 + x_3 &\geq 1 && (1.9)\\ x_1 + x_3 &\geq 1 && (1.10)\\ x_1, x_2, x_3 &\geq 0 && (1.11) \end{aligned}$$

Die optimale Lösung für diese Instanz ist: $x_1 = x_2 = x_3 = 0.5$.

Wie kann man die Lösung von Beispiel 1.3 nun interpretieren? Wie kann man eine halbe Teilmenge zu einem Set Cover hinzufügen? Diese Frage werden wir in Abschnitt 1.4 näher betrachten.

1.3.2 Duales Programm

Das Dualitätsprinzip von linearen Programmen erlaubt es uns, wie folgt zu dem primalen LP (1.4) – (1.6) ein duales LP aufzustellen, siehe [Vaz01, Kapitel 12].

Maximiere

$$\sum_{e \in U} y_e \qquad (1.12)$$

unter den Nebenbedingungen

$$\begin{aligned} \sum_{e:e \in S} y_e &\leq c(S), \quad S \in \mathcal{S} && (1.13)\\ y_e &\geq 0, \quad e \in U && (1.14) \end{aligned}$$

Dieses duale LP kann wie folgt veranschaulicht werden: Für jedes Element $e \in U$ hat man eine nicht negative Variable y_e eingeführt. Die Summe über alle y_e soll maximiert werden. Dabei gilt es die Nebenbedingung (1.13) zu beachten, die verhindert, dass einzelne Teilmengen S *überpackt* werden, d.h. die Kosten der einzelnen Elemente übersteigen die Gesamtkosten der

Teilmenge. Insbesondere ist das von Algorithmus 1 ausgewählte Set Cover mit jeweils Kosten price(e) von der dualen Lösung vollständig *bezahlt*, d.h. $\sum_{e \in U} price(e) \geq \sum_{S \in \mathcal{S}} c(S) x_S$. Im Allgemeinen ist Bedingung (1.13) für $y_e = \text{price}(e)$ nicht erfüllt.

1.3.3 Zusammenhang zwischen primalem und dualem LP

Bis jetzt haben wir ILP, primales und duales LP kennen gelernt. Ihre Beziehungen untereinander sind in Abbildung 1.3 dargestellt. Die Kosten einer optimalen Lösung des primalen und des dualen Programms stimmen überein. Jede duale Lösung ist eine untere Schranke für jede primale Lösung, insbesondere auch für die beste ganzzahlige Lösung. Das duale LP wird maximiert und dessen Lösung hat als Wert OPT_f, was auch der Wert der Lösung des zu minimierenden primalen LP ist. Die optimale Lösung des ILP hat den Wert OPT, siehe Abbildung 1.3. Diese Lücke $\frac{\text{OPT}}{\text{OPT}_f}$ [Vaz01, Kapitel 12] wird Ganzzahligkeitslücke genannt und trennt eine optimale ganzzahlige von einer optimalen gebrochenen Lösung einer Set Cover Instanz ab. Die Schwierigkeit besteht darin, aus einer gebrochenen optimalen Lösung eine gültige und möglichst gute ganzzahlige Lösung zu erhalten. Dieses Problem wird im nächsten Abschnitt behandelt.
Eine andere Möglichkeit, die Beziehung zwischen primalem und dualem LP

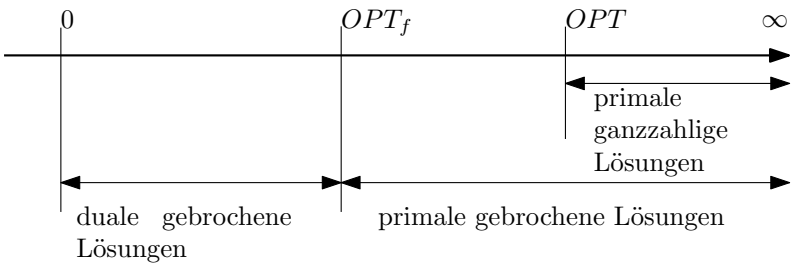

Abbildung 1.3: Zusammenhang von ILP, primalem und dualem LP

auszunutzen ist, das Set Cover des Greedy-Algorithmus 1 als Lösung des primalen und dualen LP zu interpretieren. Eine Methode die hier ihre Anwendung findet wird Dual Fitting gennant und bedeutet in unserem Fall, dass der Greedy-Algorithmus eine duale Lösung mitberechnet, nämlich durch die price(e)-Werte. Allerdings ist diese duale Lösung nicht gültig, deshalb skaliert Dual Fitting diese Werte der Variablen um einen Faktor H_n, so dass die Lösung gültig wird (d.h. erfüllt Bedingungen (1.5),(1.6)). Dieser Ansatz ist eine weitere Möglichkeit den Approximationsfaktor des Greedy-Algorithmus zu bestimmen.

1.4 Runden

Weil wir in diesem Abschnitt einige lineare Programme effizient lösen wollen, benötigen wir dazu das Innere-Punkte-Verfahren und dessen Laufzeit.

Satz 1.2. *[Kar84] Die Laufzeit zum Lösen eines LP liegt mit dem Innere-Punkte-Verfahren in $\mathcal{O}(s^{3,5}L^2 \log L \log\log L)$ mit s Variablen und L Eingabebits.*

Die Relaxation (1.4) – (1.6) der ILP Formulierung von Set Cover erlaubt es uns nach Satz 1.2 effizient eine gebrochene Lösung zu bestimmen. Eine nahe liegende Möglichkeit aus einer gebrochenen eine ganzzahlige Lösung zu erhalten, ist das Runden der gebrochenen Werte.

1.4.1 Set Cover Algorithmus mit einfachem Runden

Eine triviale Möglichkeit wäre, alle Werte größer Null auf Eins zu runden. So erhält man in jedem Fall ein gültiges Set Cover. Wir verbessern unsere Strategie des Rundens, um sowohl den Beweis der Approximationsgüte einfacher zu machen, als auch um weniger Mengen auszuwählen. Die Frequenz f ist die maximale Anzahl von Mengen, die ein Element $e \in U$ überdecken, also $f = \max_e |\{S \in \mathcal{S} | e \in S\}|$. Wir werden zeigen, dass der folgende Algorithmus eine Faktor-f-Approximation liefert.

Algorithmus 2 : Set Cover mit LP-Runden

Finde eine optimale Lösung für die LP-Relaxation (1.4) – (1.6).
Nimm alle Mengen S mit $x_S \geq 1/f$ in die Lösungsmenge, d.h. setze $x_S = 1$, andernfalls $x_S = 0$.

Beispiel 1.4. *Einfaches Runden angewandt auf Beispiel 1.3, liefert eine gültige, aber keine optimale Lösung $x_1 = x_2 = x_3 = 1$.*

Satz 1.3. *Algorithmus 2 ist ein Faktor-f-Approximationsalgorithmus für das Set Cover Problem. Das Lösen des LP ist nach Satz 1.2 effizient möglich. Das anschließende Runden benötigt $\mathcal{O}(n)$ Zeit.*

Beweis. Sei C die Menge der von Algorithmus 2 ausgewählten Teilmengen. Ein Element e ist nach Definition in maximal f verschiedenen Teilmengen enthalten. Mindestens eine dieser Mengen muss aufgrund der Bedingung (1.5) mit mindestens der Größe $1/f$ in der gebrochenen Lösung ausgewählt sein. Somit wird e von C überdeckt und man hat ein gültiges Set Cover. Beim Runden wird x_S für jede Teilmenge $S \in C$ maximal um einen Faktor f erhöht. Somit sind die Kosten von C maximal f mal größer als die einer optimalen gebrochenen Lösung. □

1.4.2 Randomisiertes Runden bei Set Cover

Eine weitere Möglichkeit ist, die optimale, gebrochene Lösung als Wahrscheinlichkeitsvektor aufzufassen, anhand dessen die Mengen wie folgt in das Set Cover genommen werden.

Algorithmus 3 : Randomisiertes Runden bei Set Cover

Finde eine optimale Lösung $\mathbf{x} = (x_{S_1}, ..., x_{S_k})$ für die LP-Relaxation (1.4) – (1.6).
repeat
  $C \leftarrow \emptyset$
  for $i = 1$ **to** $d \log n$ **do**
    for $j = 1$ **to** k **do**
      Werfe Münze für jede Menge S_j mit der Wahrscheinlichkeit x_{S_j}.
      Nimm alle Mengen S_j, die ausgewählt wurden, in die bisherige Lösungsmenge C auf, also $C \leftarrow C \cup \{S_j\}$
until C *ist gültiges* Set Cover.

Satz 1.4. *Algorithmus 3 ist ein $\mathcal{O}(\log n)$-Approximationsalgorithmus für das* Set Cover *Problem. Das Lösen des LP ist nach Satz 1.2 effizient möglich. Das anschließende Runden benötigt bei erwartet $\mathcal{O}(1)$ Durchläufen $\mathcal{O}(n \log n)$ Zeit.*

Beweis. Skizze: Es ist d eine Konstante, die folgende Ungleichung erfüllen muss, um eine gültige Lösung zu erhalten: $(\frac{1}{e})^{d \log n} \leq \frac{1}{4n}$. Weil die Schleife in Algorithmus 3 $d \log n$-mal ausgeführt wird bekommt man über elementare stochastische Überlegungen und Anwendung der Markov-Ungleichung, dass die Wahrscheinlichkeit $\mathbb{P}[\mathbf{x}$ ist eine gültige Lösung und hat Kosten $\leq \mathrm{OPT}_f \cdot 4d \log n] \geq \frac{1}{2}$ ist. Die erwartete Anzahl an Durchläufen ist daher höchstens 2. Für den vollständigen Beweis siehe [Vaz01, Kapitel 14]. □

In diesem Abschnitt wurde das Lineare Programm exakt gelöst und dann das Ergebnis weiterverarbeitet. Ein LP kann man nach Satz 1.2 in polynomieller Zeit lösen. Das einfache Runden geht in linearer Zeit und Algorithmus 3 muss erwartungsgemäß nur $\mathcal{O}(1)$ mal zu $\mathcal{O}(n \log n)$ Zeit durchlaufen werden.

1.5 Primal-Duales Schema

Wie man diese Laufzeit aus Abschnitt 1.4 noch unterbietet, wird in diesem Abschnitt aufgezeigt. Es wird nicht mehr das primale bzw. duale LP gelöst, sondern es wird mit Bedingungen für eine gültige Lösung an diesen LP gearbeitet. Dadurch erhält man eine erhebliche Laufzeitverbesserung.

Primales LP	**Duales LP**
minimiere $\sum_{j=1}^{n} c_j x_j$	maximiere $\sum_{j=1}^{n} b_i y_i$
unter den Nebenbedingungen	
$\sum_{j=1}^{n} a_{ij} x_j \geq b_i,\ i = 1, .., m$	$\sum_{i=1}^{m} a_{ij} y_i \leq c_j,\ j = 1, .., n$
$x_j \geq 0,\ j = 1, ..., n$	$y_i \geq 0,\ i = 1, ..., m$

1.5.1 Primal-Duales Schema allgemein

Um das Primal-Duale Schema herzuleiten, benötigt man die Standardform für ein primales und duales LP. An das primale LP kann man die folgende primale komplementäre Schlupfbedingung stellen, dabei sei $\alpha \geq 1$.

$$\forall j : 1 \leq j \leq n : \text{ entweder } x_j = 0 \text{ oder } c_j/\alpha \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j \tag{1.15}$$

An das duale LP kann man die folgende duale komplementäre Schlupfbedingung stellen, dafür sei $\beta \geq 1$.

$$\forall i : 1 \leq i \leq m : \text{ entweder } y_i = 0 \text{ oder } b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \beta b_i \tag{1.16}$$

Die Erfüllung dieser beiden Schlupfbedingungen ist Voraussetzung dafür, dass das Primal-Duale Schema angewendet werden kann. Prinzipiell beginnt man beim Primal-Dualen Schema mit einer ungültigen primalen Lösung und einer gültigen dualen Lösung. Trivialerweise ist das $\mathbf{x} = 0, \mathbf{y} = 0$. Solange die Lösung noch ungültig ist, wird die Gültigkeit der primalen und der Wert der dualen Lösung abwechselnd verbessert. Die Ausgabe ist eine gültige ganzzahlige Lösung des primalen LP. Setzt man in Bedingung (1.15) und (1.16), $\alpha = 1$ und $\beta = 1$, dann ist eine gültige Lösung nach [Vaz01, Theorem 12.3] bereits eine optimale Lösung.

Satz 1.5. *Wenn* $\mathbf{x}$ *und* $\mathbf{y}$ *gültige Lösungen für das primale und duale LP sind und die beiden komplementären Schlupfbedingungen (1.15) und (1.16) erfüllt sind, dann gilt:*

$$\sum_{j=1}^{n} c_j x_j \leq \alpha \cdot \beta \cdot \sum_{i=1}^{m} b_i y_i \tag{1.17}$$

Satz 1.5 sagt aus, dass eine gültige Lösung des primalen LP, die die beiden Schlupfbedingungen erfüllt, um maximal $\alpha \cdot \beta$ schlechter ist als die optimale Lösung des LP, da $\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$ [Vaz01, Theorem 12.2] erfüllt ist. Folglich hat ein Algorithmus für das Primal-Duale Schema einen Approximationsfaktor von $\alpha \cdot \beta$.

Der Vorteil dieses Verfahrens gegenüber rundungsbasierten Verfahren liegt darin, dass nicht mehr das LP gelöst werden muss, sondern nur noch abwechselnd mit (1.15) und (1.16) gearbeitet wird. Dadurch ergibt sich eine große Laufzeitverbesserung.

1.5.2 Primal-Duales Schema für Set Cover

Hier wenden wir das Primal-Duale Schema auf Set Cover an. Dabei wählt man $\alpha = 1$ und $\beta = f$ und erhält damit nach Satz 1.5 einen Approximationsfaktor von f. Die primale komplementäre Schlupfbedingung lautet nun

$$\forall S \in \mathcal{S} : x_S \neq 0 \Rightarrow \sum_{e:e \in S} y_e = c(S). \tag{1.18}$$

Die Gleichheit gilt deshalb, weil in 1.15 $\alpha = 1$ gesetzt wurde. Die duale komplementäre Schlupfbedingung lautet

$$\forall e : y_e \neq 0 \Rightarrow \sum_{S:e \in S} x_S \leq f. \tag{1.19}$$

Ein Element kann maximal f-mal nach Definition überdeckt werden. Somit ist die duale komplementäre Schlupfbedingung trivialerweise erfüllt.

Algorithmus 4 : Primal-Dualer Algorithmus für Set Cover

Beginne mit einer ungültigen primalen Lösung $C = \emptyset$ und einer gültigen dualen Lösung $\mathbf{y} = 0$.
while *Noch nicht alle Elemente überdeckt* **do**
  Wähle ein noch nicht überdecktes Element e, erhöhe y_e bis einige Mengen gesättigt sind, d.h. Gleichung (1.18) erfüllt wird.
  Nimm alle gesättigten Mengen in das Cover C.
  Alle Elemente in diesen Mengen sind nun überdeckt.
Ausgabe: gültige ganzzahlige Lösung: Set Cover C.

Satz 1.6. *Algorithmus 4 ist ein Faktor-f-Approximationsalgorithmus für* Set Cover *und hat eine Laufzeit von $\mathcal{O}(\sum_{S \in \mathcal{S}} |S|)$.*

Beweis. Algorithmus 4 stellt sicher, dass alle Elemente überdeckt werden und keine Menge überpackt wird. Somit sind Bedingungen (1.18) und (1.19) erfüllt und daher ist sowohl die primale als auch die duale Lösung gültig. Nach Satz 1.5 und Schlupfbedingungen (1.18) und (1.19) hat man hier mit $\alpha = 1$ und $\beta = f$ einen Approximationsfaktor von f.

In jeder Iteration von Algorithmus 4 wird ein Element e ausgewählt und y_e erhöht bis mindestens eine Menge S_j gesättigt ist. Dabei müssen alle Mengen, in denen e enthalten ist überprüft werden. Eine Menge S wird dementsprechend maximal $|S|$-mal überprüft. Dies ergibt die Laufzeit von $\mathcal{O}(\sum_{S\in\mathcal{S}} |S|)$. □

1.6 Fazit

In dieser Ausarbeitung wurden vier verschiedene Approximationsalgorithmen betrachtet um das Problem Set Cover zu lösen. Angefangen bei einem einfachen Greedyalgorithmus über zwei rundungsbasierte Verfahren wurde schließlich das Primal-Duale Schema vorgestellt. Zusammenfassend sind die Algorithmen und ihre Eigenschaften in Tabelle 1.1 dargestellt.

Verfahren	Laufzeit	Approximations-faktor
Greedy	$\mathcal{O}(\sum_{S\in\mathcal{S}} \|S\|)$	H_n
einfaches Runden	LP: $\mathcal{O}(s^{3,5}L^2 \ln L \ln\ln L)$ Runden $\mathcal{O}(n)$	f
randomisiertes Runden	LP: $\mathcal{O}(s^{3,5}L^2 \ln L \ln\ln L)$ Erwartet $\mathcal{O}(n \log n)$	$\mathcal{O}(\log n)$
Primal-Duales Schema	$\mathcal{O}(\sum_{S\in\mathcal{S}} \|S\|)$	f

Tabelle 1.1: Vergleich der betrachteten Verfahren bezogen auf Set Cover

Mit einer Laufzeit von $\mathcal{O}(\sum_{S\in\mathcal{S}} |S|)$ und einem Approximationsfaktor f ist das Primal-Duale Schema die Methode der Wahl zum Lösen von Set Cover Instanzen, solange $f \leq \log n$. Sobald $f \geq \log n$, kann man mit dem Greedyalgorithmus mit gleichem Zeitaufwand und einem Approximationsfaktor von H_n bessere Ergebnisse erzielen. Die rundungsbasierten Verfahren haben den Nachteil, dass das LP gelöst werden muss.

Kapitel 2

Facility-Location und K-Median

Wei Cheng

Zusammenfassung

In diesem Kapitel werden Approximationsalgorithmen für die Probleme *Facility-Location* und *K-Median* vorgestellt. Grundlage für die Ausarbeitung sind Kapitel 24 und 25 des Buches *Approximation Algorithms* von Vazirani [Vaz01]. Es wird ein primal-dual basierter Algorithmus für Facility-Location vorgestellt, der eine 3-Approximation erreicht. Außerdem wird basierend auf dem ersten Algorithmus eine 6-Approximation für das K-Median-Problem angegeben, die ein randomisiertes Rundungsverfahren benutzt.

2.1 Facility-Location

Das Problem *Facility-Location* spielt eine wichtige Rolle im Fachbereich Operations Research. Man kann mit Facility-Location beispielsweise den Ort von Fabriken, Schulen und Krankenhäusern so wählen, dass die Gesamtkosten minimal sind. Beispielsweise soll es einige Krankenhäuser in einer Stadt geben. In einigen Gebieten ist es teuer, ein Krankenhaus zu bauen, dafür liegen sie im Stadtzentrum. Das bedeutet diese Krankenhäuser haben hohe Eröffnungskosten aber niedrige Verbindungskosten. Andere Krankenhäuser haben niedrige Eröffnungskosten aber hohe Verbindungskosten. Das Ziel der Stadt ist, die Eröffnungskosten sowie Verbindungskosten zu minimieren, und die Anforderung von Krankenhäusern zu erfüllen. Facility-Location ist leider ein NP-schweres Problem. Daher ist es vermutlich nicht effizient lösbar und es ist sinnvoll, einen Approximationsalgorithmus zu finden. In diesem Kapitel wird ein primal-dual basierter Algorithmus erläutert, der eine 3-Approximation für Facility-Location erreicht.
Zuerst gebe ich eine formale Definition von dem Problem Facility-Location an, anschließed wird das Problem als ILP(ganzzahliges lineares Programm) formuliert. Danach wird ein primal-dual basierter Algorithmus angegeben und analysiert.

Problem 2.1. *Facility-Location*
Sei $F = \{F_1, F_2, \ldots, F_k\}$ eine Menge von möglichen Standorten, sowie $C = \{C_1, C_2, \ldots, C_l\}$ eine Menge von Städten. Weiter seien f_i die Eröffnungskosten des Standortes F_i und c_{ij} bezeichne die Verbindungskosten von C_j zu F_i, wobei diese die Dreieckungleichung erfüllen.
Gesucht ist eine Teilmenge $I \subseteq F$ von eröffneten Standorten, sowie eine Funktion $\phi : C \to I$, die jeder Stadt einen offenen Standort zuordnet. Das Ziel ist, die Summe aus Eröffnungskosten und Verbindungskosten zu minimieren.

2.1.1 Lineare Programme

In Folgenden wird ein ganzzahliges lineares Programm für Facility-Location angegeben.

Primales Problem:
Minimiere:

$$\sum_{i \in F, j \in C} c_{ij} x_{ij} + \sum_{i \in F} f_i y_i \tag{2.1}$$

unter den Nebenbedingungen:

$$\sum_{i \in F} x_{ij} \geq 1, \quad j \in C \tag{2.2}$$
$$y_i - x_{ij} \geq 0, \quad i \in F, \quad j \in C \tag{2.3}$$
$$x_{ij} \in \{0,1\}, \quad i \in F, \quad j \in C \tag{2.4}$$
$$y_i \in \{0,1\}, \quad i \in F \tag{2.5}$$

Das LP hat für jeden Standort eine Variable y_i, die angibt ob Standort i geöffnet ist. Die Variable x_{ij} besagt, ob Standort i mit Stadt j verbunden ist. Nebenbedingung 2.2 bedeutet, dass jede Stadt mit einem Standort verbunden werden muss. Nebenbedingung 2.3 besagt, dass Standorte, zu denen sich eine Stadt verbindet, eröffnet sein müssen. Eine ganzzahlige Lösung des LPs entspricht einer Lösung von Facility-Location gemäß $y_i = 1 \Leftrightarrow i \in I$ und $x_{ij} = 1 \Leftrightarrow i = \phi(j)$.
Um die Dualitätstheorie linearer Programme anwenden zu können, relaxieren wir das Problem, indem wir die Nebenbedingungen $x_{ij} \in \{0,1\}$ und $y_i \in \{0,1\}$ durch $x_{ij} \geq 0$ und $y_i \geq 0$ ersetzen. Das zugehörige duale Problem lautet wie folgt.

Duales Problem:
Maximiere:

$$\sum_{j \in C} \alpha_j \tag{2.6}$$

unter den Nebenbedingungen:

$$\alpha_j - \beta_{ij} \leq c_{ij}, \quad i \in F, \quad j \in C \tag{2.7}$$
$$\sum_{j:\phi(j)=i} \beta_{ij} \leq f_i, \quad i \in F \tag{2.8}$$
$$\alpha_j \geq 0, \quad j \in C \tag{2.9}$$
$$\beta_{ij} \geq 0, \quad i \in F, \quad j \in C \tag{2.10}$$

Jetzt gebe ich eine anschauliche Bedeutung für die duale Lösung (mehr Information über Dualität kann man im Kapitel 12 des Buches *Approximation Algorithms* von Vazirani [Vaz01] finden). Die Variable α_j ist das gesamte Geld, das die Stadt j bezahlt, c_{ij} sind die Verbindenskosten. Und β_{ij} ist der Beitrag, den Stadt i zur Eröffnung von Standort j bezahlt.
Seien (x, y) eine optimale primale Lösung, und (α, β) eine optimale Lösung

des dualen Programmes. Dann gelten folgende Schlupfbedingungen:

$$\forall i \in F, j \in C : x_{ij} > 0 \Rightarrow \alpha_j - \beta_{ij} = c_{ij} \tag{2.11}$$

$$\forall i \in F : y_i > 0 \Rightarrow \sum_{j:\phi(j)=i} \beta_{ij} = f_i \tag{2.12}$$

$$\forall j \in C : \alpha_j > 0 \Rightarrow \sum_{i \in F} x_{ij} = 1 \tag{2.13}$$

$$\forall i \in F, j \in C : \beta_{ij} > 0 \Rightarrow y_i = x_{ij} \tag{2.14}$$

Daraus folgen zwei wichtige Eigenschaften, die wir im nächsten Abschnitt für den Entwurf des Algorithmus benutzen werden: Wenn $\phi(j) = i$, dann gilt $\alpha_j - \beta_{ij} = c_{ij}$ und $\sum_{j:\phi(j)=i} \beta_{ij} = f_i$, d.h. wenn eine Stadt mit einem Standort verbunden ist, sind ihre Gesamtkosten gleich der Summe aus den Verbindungskosten und dem Beitrag der Stadt zum Standort, und die Eröfnungskosten eines geöffneten Standorts werden vollständig durch die dort angebundenen Städte bezahlt.

2.1.2 Primal-Dual basierter Algorithmus

Der Algorithmus besteht aus zwei Phasen. In Phase 1 werden mehrere Standorte eröffnet, und jede Stadt mit einem oder mehrerem Standorten verbunden. In Phase 2 werden unnötige Standorte geschlossen und sichergestellt, dass jede Stadt mit genau einem Standort verbunden ist.

Primal-Dual basierter Algorithmus - Phase 1: Wir möchten die duale Lösung so groß wie möglich finden. Dazu erhöht jede Stadt ihre duale Variable α_j. Da die Schlupfbedingungen erfüllt werden müssen, zieht dies ggf. auch Erhöhungen von β_{ij} nach sich, gemäß der Bedingung 2.11.
Am Anfang (Zeitpunkt 0) sind alle Städte *unverbunden*, und alle Variablen sind gleich 0. In jedem Zeitschritt erhöhen wir die α_j der unverbundenen Städte j um 1.
Sobald $\alpha_j = c_{ij}$ gilt, ist die Kante (i, j) *bezahlt*. Danach wird die Duale Variable β_{ij} auch erhöht, gemäß der Schlupfbedingung 2.11, $\alpha_j - \beta_{ij} = c_{ij}$. Die Kanten (i, j) mit $\beta_{ij} > 0$ heißen *besonders*. Besonders bedeutet, die Stadt hat schon die Kosten für die Verbindung aufgebracht und beteiligt sich an der Eröffung von Standort i.
Sobald $\sum \beta_{ij} = f_i$ gilt, ist der Standort i *bezahlt*. Der Algorithmus definiert den Standort i dann als *temporär geöffnet*. Städte mit bezahlter Kante zu einem eröffneten Standort F_i sind *verbunden*, und Standort i ist *Verbindungszeuge* dieser Städte. Die duale Variable α_j einer verbundenen Stadt wird nicht mehr erhöht. Sobald eine unverbundene Stadt j eine bezahlte Kante zu einem eröffneten Standort i hat, ist Stadt j ebenfalls mit Standort i verbunden, und Standort i ist Verbindungszeuge von j. In diesem Fall ist $\beta_{ij} = 0$, und die Kante (i, j) ist *nicht besonders*.

Wenn alle Städte verbunden sind, endet Phase 1. Wenn zu einem Zeitpunkt mehr als ein Ereignis gleichzeitig stattfindet, werden sie in beliebige Reihenfolge behandelt.
Am Ende der Phase 1, gibt es noch zwei Probleme: Zu viele Standorte sind geöffnet und eine Stadt kann zu mehreren Standorten verbunden sein.

Primal-Dual basierter Algorithmus - Phase 2: In Phase 2 müssen wir natürlich die Probleme von der Phase 1 lösen. Daher schließen wir unnötige Standorte, und verbinden alle Städte mit nur einem Standort. Vor dem Anfang von Phase 2 möchte ich einige Variablen definieren:
Sei F_t die Menge des temporär geöffneten Standorte. Wir definieren nun den Graph H auf diesen Standorten. Eine Kante zwischen zwei Standorten in H bedeutet, dass es eine Stadt gibt, die gleichzeitig mit den beiden Standorten verbunden ist. In diesem Fall ist es unnötig, beide Standorte zu eröffnen, einer davon genügt. Von zwei verbundenen Standorten soll also stets höchstens einer geöffnet werden. Andererseits soll jeder Standort mindestens einen eröffnete Nachbarn in H besitzen. Dies entspricht genau einer *maximalen unabhängigen Menge* in H.
Sei I eine beliebige maximale unabhängige Menge in H. Wir eröffnen alle Standorte in I. Nun benötigen wir noch eine Zuordnungsfunktion $\phi : C \to I$. Dazu betrachten wir für jede Stadt j die Menge $\mathcal{F}_j = \{i \in F_t \mid (i, j) \text{ ist besonders}\}$ der Standorte, zu diesem j mit einer besonderen Kante verbunden ist. Weil I eine unabhängige Menge ist, gibt es höchstens einen geöffnenten Standort in $\mathcal{F}_j$. Sei nun j eine feste Stadt. Wir unterschieden zwei Fälle.
Im ersten Fall ist ein Standort $i \in \mathcal{F}_j$ *geöffnet*. Dann können wir j mit i verbinden, also $\phi(j) := i$ setzen, und die Stadt j ist *direkt verbunden*. Im zweiten Fall ist kein Standort in $\mathcal{F}_j$ geöffnet. Sei I' die Menge der Standorte mit bezahlter Verbindung zu j, die aber nicht besonders ist, also für alle $i' \in I'$, gilt $\beta_{i'j} = 0$. Die Menge I' ist nicht leer, da nach Phase 1 alle Städte verbunden sind. Die Standorte $i' \in I'$ sind hier Verbindungszeuge von j. Dann gibt es zwei Unterfälle. Wenn ein Standort $i' \in I'$ eröffnet ist, so verbinde Stadt j mit i', setze also $\phi(j) := i'$. Stadt j ist dann auch direkt verbunden. Wenn kein $i' \in I'$ geöffnet, wähle ein beliebiges $i' \in I'$, dann finde einen Standort i, der eröffneter Nachbar von i' im Graph H ist. Dieser existiert, da I eine maximale unabhängige Menge ist. Verbinde Stadt j mit i, setze also $\phi(j) := i$. Die Stadt j ist *indirekt verbunden*. Die Menge I und die Funktion ϕ definieren eine Lösung des primalen Problems.

2.1.3 Analyse

Im Folgenden zeigen wir, dass der Algorithmus eine 3-Approximation liefert. Zuerst möchten wir genauer beschreiben, wie die dualen Variablen α_j für die Kosten bezahlen. Dazu unterscheiden wir zwischen den Kosten für Verbin-

dung α_j^e und den Beiträgen zur Eröffung von Standorten α_j^f. Es gilt stets $\alpha_j = \alpha_j^f + \alpha_j^e$.
Die Kosten α_j^f sind die Eröffnungskosten eines Standorts, die von Stadt j bezahlt werden. Die Kosten α_j^e sind die Kosten, die für die Verbindung bezahlt werden. Ist j indirekt zu Standort i verbunden, dann ist $\alpha_j^f = 0$ und $\alpha_j^e = \alpha_j$. Wenn j direkt verbunden ist, dann gilt $\alpha_j^f = \beta_{ij}$ und $\alpha_j^e = c_{ij}$.
Das folgende Lemma besagt, dass die Eröffnungskosten eines geöffneten Standorts vollständig durch die dort angebunden Städte bezahlt werden.

Lemma 2.1. *Für alle $i \in I$ gilt,*

$$\sum_{j:\phi(j)=i} \alpha_j^f = f_i. \tag{2.15}$$

Beweis: Weil Standort i am Ende von Phase 1 temporär geöffnet ist, ist er bezahlt:

$$\sum_{j:(i,j)\text{besonders}} \beta_{ij} = f_i \tag{2.16}$$

Jede Stadt j, die für f_i bezahlt, muss mit i in Phase 2 direkt verbunden sein. Für diese Städte gilt $\alpha_j^f = \beta_{ij}$. Für andere Städte j', die mit i verbunden sind, muss $\alpha_{j'}^f = 0$ gelten, also auch $\beta_{ij'} = 0$
Hieraus folgt direkt, dass die Städte die Kosten für alle Eröffnungen tragen.

Korollar 2.1. *Es gilt $\sum_{i\in I} f_i = \sum_{j\in C} \alpha_j^f$.*

Beweis: Summiere über Lemma 2.1 und benutze die Tatsache, dass nur direkt verbundene Städte Eröffnungskosten bezahlen.
Da die Kosten für alle direkt verbunden Städte optimal sind, bleibt nur noch, die Verbindungskosten der indirekt verbunden Städte gegen die duale Lösung abzuschätzen. Dabei geht im wesentlichen ein, dass die Verbindugskosten die Dreiecksungleichung erfüllen.

Lemma 2.2. *Für jede indirekt verbundene Stadt j mit $i = \phi(j)$ gilt $c_{ij} \leq 3\alpha_j^e$.*

Beweis: Sei i' ein Verbindungszeuge von Stadt j. Weil j indirekt verbunden mit i ist, muss es eine Stadt j' geben, mit (i, j') und (i', j') sind beide besondere Kanten. Abbildung 2.1 zeigt die Situation. Seien t_1 und t_2 die Zeitpunkte, zu denen i und i' in Phase 1 temporär geöffnet werden. Weil die Kante (i', j) bezahlt ist, gilt $\alpha_j \geq c_{i'j}$. Wir möchten zeigen, dass $\alpha_j \geq c_{ij'}$ und $\alpha_j \geq c_{i'j'}$. Die Aussage folgt dann aus der Dreiecksungleichung. Weil die Kanten (i', j) und (i, j') bezahlt sind, gilt $\alpha_{j'} \geq c_{i'j}$ und $\alpha_{j'} \geq c_{ij'}$. Denn beide Kanten sind besonders, müssen also beide gezahlt werden, bevor i oder

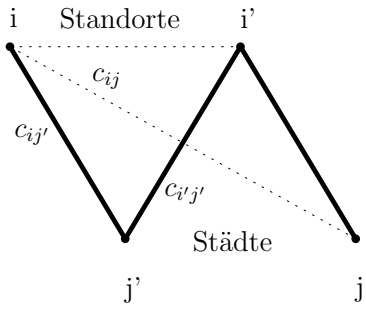

Abbildung 2.1: Stadt j ist indirekt mit Standort i verbunden.

i' geöffnet wird. Die Variable $\alpha_{j'}$ kann nicht mehr nach $\min(t_1, t_2)$ erhöht werden. Deshalb gilt $\alpha_{j'} \leq \min(t_1, t_2)$. Weil i' der Verbindungszeuge von j ist, gilt $\alpha_j \geq t_2$, und deshalb $\alpha_j \geq \alpha_{j'}$.

Satz 2.1. *Für die vom Algorithmus konstruierte primale und duale Lösung gilt:*

$$\sum_{i \in F, j \in C} c_{ij} x_{ij} + 3 \sum_{i \in F} f_i y_i \leq 3 \sum_{j \in C} \alpha_j \tag{2.17}$$

Beweis: Für eine direkt verbunde Stadt j gilt $c_{ij} = \alpha_j^e \leq 3\alpha_j^e$, wenn $\phi(j) = i$. Mit Lemma 2.2 bekommen wir:

$$\sum_{i \in F, j \in C} c_{ij} x_{ij} \leq 3 \sum_{j \in C} \alpha_j^e \tag{2.18}$$

Zusammen mit Korollar 2.1 multipliziert mit 3, folgt die Aussage des Satzes. Gemäß dem primal-dualen Schema liefert der Algorithmus also eine 3-Approximation.

2.1.4 Laufzeit

Phase 1 des Algorithmus kann mit einer Prioritätswarteschlange implementiert werden. Jede Kante wird dabei maximal zweimal betrachtet. Erst wenn sie bezahlt ist, und dann wenn sie eine Stadt zu einem Standort verbindet. Die Laufzeit ist daher in $O(m \log m)$, wobei m die Anzahl der Kanten ist. In Phase 2 muss eine maximale unabhängige Menge gefunden werden, was mit einem Greedy-Algorithmus in linearer Zeit möglich ist. Die Zuordnung der Städte zu den eröffnete Standorte benötigt $O(m)$ Zeit. Insgesamt ergibt sich somit eine Laufzeit von $O(m \log m)$.

Der folgende Satz fasst das Ergebnis des Abschnitts zusammen.

Satz 2.2. *Der Algorithmus berechnet in $O(m \log m)$ Zeit eine 3-Approximation für Facility-Location.*

2.2 K-Median

Das Problem *K-Median* ähnelt dem Problem Facility-Location. Sie Unterscheiden sich an zwei Stellen: beim K-Median-Problem sind die Eröffnungskosten für alle Standorte 0, dafür dürfen aber nur maximal k Standorte eröffnet werden. In diesem Abschnitt wird eine 6-Approximation für das K-Median-Problem vorgestellt. Diese verwendet den Algorithmus aus dem vorherigen Abschnitt als Subroutine.
Zuerst gebe ich ein lineares Programm für K-Median an, dann zeige ich den Zusammenhang zwischen den beiden Problemen. Am Ende steht der Algorithmus und seine Analyse.

Problem 2.2. *K-Median*
Die Mengen F, C und Verbindungskosten c_{ij} seien wie in Problem 2.1 definiert. Die Eröffnungskosten f_i aller Standorte seien 0.
Gesucht ist eine Teilmenge $I \subseteq F$ mit $|I| \leq k$ von eröffneten Standorten, sowie eine Funktion ϕ, die jeder Stadt einen Standort aus I zuweist. Das Ziel ist die Summe der Verbindungskosten zu minimieren.

2.2.1 Lineares Programm

Das folgende ganzzahlige lineare Programm ist für *K-Median*.
Primales Problem:
Minimiere:

$$\sum_{i \in F, j \in C} c_{ij} x_{ij} \tag{2.19}$$

unter den Nebenbedingungen:

$$\sum_{i \in F} x_{ij} \geq 1, \quad j \in C \tag{2.20}$$

$$y_i - x_{ij} \geq 0, \quad i \in F, \quad j \in C \tag{2.21}$$

$$\sum_{i \in F} -y_i \geq -k \tag{2.22}$$

$$x_{ij} \in \{0,1\}, \quad i \in F, \quad j \in C \tag{2.23}$$

$$y_i \in \{0,1\}, \quad i \in F \tag{2.24}$$

Die Variablen y_i und x_{ij} spielen die gleiche Rolle wie in LP 2.1. Die Bedingungen 2.20 und 2.21 bleiben auch wie in LP 2.1. Die Nebenbedingung 2.22 besagt, dass maximal k Standorte eröffnet werden.
Wiederum relaxieren wir das Problem, indem wir die Nebenbedingungen $x_{ij} \in \{0,1\}$ und $y_i \in \{0,1\}$ durch $x_{ij} \geq 0$ und $y_i \geq 0$ ersetzen. Das zugehörige duale Program lautet wie folgt:

Duales Problem:
Maximiere:

$$\sum_{j \in C} \alpha_j - zk \tag{2.25}$$

unter den Nebenbedingungen:

$$\begin{aligned} \alpha_j - \beta_{ij} &\leq c_{ij}, \quad i \in F, \quad j \in C && (2.26)\\ \sum_{j:\phi(j)=i} \beta_{ij} &\leq z, \quad i \in F && (2.27)\\ \alpha_j &\geq 0, \quad j \in C && (2.28)\\ \beta_{ij} &\geq 0, \quad i \in F, \quad j \in C && (2.29)\\ z &\geq 0 && (2.30) \end{aligned}$$

Die dualen Variablen α_j und β_{ij} sind auch gleich wie LP 2.6. Ein sehr wichtiger Unterschied hier ist die Variable z, die nicht in LP 2.6 vorkommt. Sie modelliert eine Art Eröffnungskosten für die Standorte.

2.2.2 Grundidee

Wegen der Ähnlichkeit zwischen den beiden Problemen, können wir K-Median mit Hilfe des Algorithmus von Facility-Location lösen.
Wichtig ist, einen guten Wert für die Variable z zu finden, weil diese eine Art Eröffnungskosten repräsentiert. Setzt man etwa $z = 0$, so werden alle Standorte geöffnet. Aber wenn z sehr groß ist, wird nur ein einziger Standort geöffnet. Die Variable z spielt also eine ähnliche Rolle wie die Eröffnungskosten in LP 2.1. Wir suchen nun ein geeignetes z, um genau k Standorte zu öffnen, und benutzen anschließend den Algorithmus für Facility-Location. Wir könnten also mit einer binären Suche ein geeignetes $z \in [0, nc_{\max}]$ suchen, wobei $c_{\max}$ die schwerste Kante ist.
Aber es ist zu aufwändig ein solches z zu finden. Daher finden wir stattdessen z_1 und z_2, mit z_1 öffnet $k_1 < k$ Standorte, z_2 öffnet $k_2 > k$ Standorte, so dass z_1 und z_2 kleinen Abstand haben, d.h. $z_1 - z_2 \leq c_{\min}/(12n_c^2)$. Dabei bezeichnet $c_{\min}$ das Gewicht der leichtesten Kante und n_c die Anzahl der Städte. Seien (x^s, y^s) und (x^l, y^l) die zu den Eröffnungskosten z_1 und z_2 gehörigen Lösungen des Facility-Location-Problems, weiter seien (α^s, β^s) und (α^l, β^l), die entsprechenden dualen Lösungen. Wir definieren eine fraktionale Lösung des K-Medien-Problems durch Konvexkombination von (x^s, y^s) und (x^l, y^l), so dass k Standorte eröffnet werden. Seien $(x, y) = a(x^s, y^s) + b(x^l, y^l)$, mit $k = ak_1 + bk_2$, wobei $a = (k_2 - k)/(k_2 - k_1)$ und $b = (k - k_1)/(k_2 - k_1)$ gilt. Im Folgenden zeige ich erst, dass (x, y) nicht zu weit von den optimale Kosten entfernt ist. Anschließend werden wir sehen, dass sich die Qualität der Lösung durch randomisiertes Runden höchstens um Faktor 2 verschlechtert.

Lemma 2.3. *Die Kosten von (x, y) sind höchstens $(3 + 1/n_c)$ mal so groß wie die optimalen Lösung des K-Median-Problems.*

Beweis: Nach Satz 2.1 gilt:

$$\sum_{i\in F,j\in C} c_{ij}x^s_{ij} \leq 3(\sum_{j\in C}\alpha^s_j - z_1k_1) \tag{2.31}$$

und

$$\sum_{i\in F,j\in C} c_{ij}x^l_{ij} \leq 3(\sum_{j\in C}\alpha^l_j - z_2k_2) \tag{2.32}$$

Weil $z_1 > z_2$, ist (α^l, β^l) auch eine duale Lösung des Facility-Location-Problems, wenn die Eröffnungskosten der Standorte z_1 ist. Wir möchten z_2 in der zweiten Ungleichung durch z_1 ersetzen. Dazu brauchen wir auch die obere Grenze von $z_1 - z_2 \leq c_{\min}/(12n_c^2)$ und $\sum_{i\in F,j\in C} c_{ij}x^l_{ij} \geq c_{\min}$. Dann haben wir:

$$\sum_{i\in F,j\in C} c_{ij}x^l_{ij} \leq (3+\frac{1}{n_c})(\sum_{j\in C}\alpha^l_j - z_1k_2) \tag{2.33}$$

Multiplikation mit b und a-fache Addition der ersten Ungleichung ergibt:

$$\sum_{i\in F,j\in C} c_{ij}x_{ij} \leq (3+\frac{1}{n_c})(\sum_{j\in C}\alpha_j - z_1k) \tag{2.34}$$

wobei $\alpha = a\alpha^s + b\alpha^l$, $\beta = a\beta^s + b\beta^l$.
Im nächsten Abschnitt werden wir zeigen, dass die Lösung (x, y) ganzzahlig gemacht werden kann, ohne die Lösungsqualität zu stark zu verschlechtern.

2.2.3 Randomisiertes Runden

Das Problem ist, dass die Lösung nicht ganzzahlig ist. Wir geben ein randomisiertes Rundungsverfahren mit Kosten von Faktor $1 + \max(a, b)$ an.
Seien A und B die Menge von Standorten, die in den beiden Lösungen eröffnet werden. Es gilt $|A| = k_1$ und $|B| = k_2$. Für jeden Standort in A, suche den nächsten Standort in B (die Standorte können gleich sein). Seien $B' \subset B$ diese Standorte. Wenn $|B'| < k_1$ ist, füge $k_1 - |B'|$ zufällige Standorte von $B - B'$ zu B' hinzu.
Das randomisierte Runden funktioniert wie folgt. Mit Wahrscheinlichkeit a werden alle Standorte in A geöffnet, mit Wahrscheinlichkeit $b = 1 - a$ alle Standorte in B'. Zusätzlich eröffnen wir $k-k_1$ zufällige Standorte aus $B-B'$. Sei I die Menge von eröffneten Standorten. Es gilt $|I| = k$ nach Konstruktion.
Jetzt definieren wir die Funktion $\phi : C \to I$. Sei die Stadt j verbunden mit $i_1 \in A$ und $i_2 \in B$ in den zwei Lösungen von Facility-Location. Dann definieren wir cost(j) durch cost$(j) = ac_{i_1j} + bc_{i_2j}$. Liegt i_2 in B', so ist in jedem Fall einer der beiden Standorte in der Lösung enthalten und die erwarteten Verbindenskosten sind cost(j). Anderenfalls sei i_3 in B' der nächste Standort von i_1. Die Stadt j kann mit i_1, i_2 oder i_3 verbunden werden, siehe Abbildung

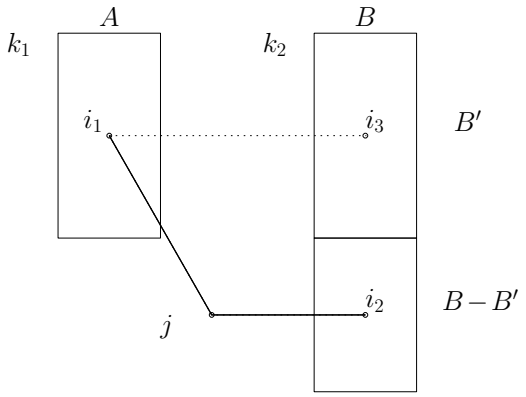

Abbildung 2.2: Mögliche Standorte für die Stadt j. Standort $i_2 \in B - B'$ und $i_3 \in B'$ ist der nächsten Standort von i_1.

2.2. Die Wahrscheinlichkeit, dass i_2 geöffnet wird, ist b. Die Wahrscheinlichkeit, dass Standort i_2 ungeöffnet und i_1 geöffnet ist, ist $(1-b)a = a^2$. Die Wahrscheinlichkeit, dass beide ungeöffnet sind, ist $(1-b)(1-a) = ab$. Dann haben wir:

$$\mathbf{E}[c_{\phi(j)j}] = bc_{i_2j} + a^2c_{i_1j} + abc_{i_3j} \tag{2.35}$$

Weil i_3 der nächsten Standort von i_1 ist, gilt $c_{i_1i_3} \leq c_{i_1i_2} \leq c_{i_1j} + c_{i_2j}$. Wegen der Dreiecksungleichung, gilt $c_{i_3j} \leq c_{i_1j} + c_{i_1i_3} \leq 2c_{i_1j} + c_{i_2j}$. Dann folgt:

$$\mathbf{E}[c_{\phi(j)j}] \leq bc_{i_2j} + a^2c_{i_1j} + ab(2c_{i_1j} + c_{i_2}j) \tag{2.36}$$

Wegen $a^2c_{i_1j} + abc_{i_1j} = ac_{i_1j}$, folgt:

$$\begin{aligned} \mathbf{E}[c_{\phi(j)j}] &\leq (ac_{i_2j} + bc_{i_2j}) + ab(c_{i_1j} + c_{i_2}j) \\ &\leq (1 + \max(a,b))\text{cost}(j) \end{aligned}$$

Insgesamt ergibt sich folgendes Lemma:

Lemma 2.4. *Für die erwarteten Verbindenskosten von Stadt j gilt*

$$\boldsymbol{E}[c_{\phi(j)j}] \leq (1 + \max(a,b))\text{cost}(j) \tag{2.37}$$

2.2.4 Analyse

In diesem Abschnitt zeigen wir die Approximationsgarantie und die Laufzeit. Für die Variablen a, b, gilt $a \leq 1 - 1/n_c$ (Gleichheit liegt nur bei $k_1 = k-1, k_2 = n_c$ vor) und $b \leq 1 - 1/k$ (Gleichheit nur bei $k_1 = 1$, $k_2 = k+1$). Deshalb gilt $1 + \max(a,b) \leq 2 - 1/n_c$. Zusammen mit Lemma 2.3 ist die Approximationsgarantie $(2 - 1/n_c)(3 + 1/n_c) < 6$.
Jetzt zeigen wir die Laufzeit. Der Bereich der binären Suche ist $[0, nc_{\max}]$ bis $z_1 - z_2 \leq c_{\min}/(12n_c^2)$. Deshalb brauchen wir insgesamt $O(\log(c_{\max}/c_{\min}) +$

$\log n$) Proben. Jede Probe braucht $O(m \log m)$ Zeit mittels Facility-Location. Das Bilden der Konvexkombination und das randomisierte Runden lässt sich mit linearer Laufzeit implementieren. Damit ergibt sich eine Gesamtlaufzeit von $O(m \log m(\log(c_{\max}/c_{\min}) + \log n))$.

Der folgende Satz fasst die Ergebnisse dieses Abschnitts zusammen.

Satz 2.3. *Der Algorithmus berechnet in $O(m \log m(\log(c_{\max}/c_{\min}) + \log n))$ Zeit eine 6-Approximation für K-Median.*

2.3 Zusammenfassung

In der Ausarbeitung wurden Algorithmen für Facility-Location und K-Median beschrieben. Es wurde zuerst ein primal-dual basierter Algorithmus für Facility-Location vorgestellt, der in $O(m \log m)$ Zeit eine 3-Approximation erreicht. Anschließend wurde basierend auf dem ersten Algorithmus eine 6-Approximation für das K-Median-Problem angegeben.
Bemerkenswert ist, dass sich durch geeignete Wahl der Eröffungskosten (so dass genau k Standorte eröffnet werden) das K-Median-Problem auf Facility-Location reduzieren lässt. Da sich ein geeigneter Wert leider nicht effizient bestimmen lässt, wurde mit Hilfe eines randomisierten Rundungsverfahrens aus der Lösung zweier Facility-Location-Instanzen eine Lösung des K-Median-Problems gewonnen. Diese Konstruktion hat den Vorteil, dass eine bessere Approximation von Facility-Location auch direkt eine Verbesserung der Approximation von K-Median nach sich zieht.

Kapitel 3

Steinerwald & Steinernetzwerk

Sebastian Bauer

Zusammenfassung

Dieses Kapitel basiert auf dem Buch von Vazirani [Vaz01] und behandelt jeweils einen Approximationsalgorithmus für das Steinerwald- und das Steinernetzwerk-Problem auf Basis eines linearen Programms. Dabei wird das lineare Programm jedoch nicht mit einem allgemein gültigen Verfahren, wie dem Simplexverfahren, gelöst, da diese auf der einen Seite zu langsam oder auf der anderen Seite keine ganzzahlige Lösung liefern. Vielmehr wird jeweils ein eigenständiger Faktor-2 Approximationsalgorithmus entwickelt, welcher iterativ mit Hilfe eines linearen Programms eine Lösung approximiert. Die Technik des iterativen Lösens des linearen Programms wird hierbei zusätzlich mit anderen Techniken kombiniert werden, die in früheren Kapiteln schon vorgestellt worden sind.

3.1 Steinerwald

In Peer-to-Peer-Netzwerken, wie sie heute zum Beispiel bei Bittorrent eingesetzt werden, sind meist hohe Auslastungen der Netzwerkverbindungen vorzufinden [Ell06]. Besonders hierbei ist, dass eine bestimmte Gruppe von Teilnehmern exakt dieselben Daten verteilen und empfangen. Um die Netzwerkauslastung zu minimieren, damit auch noch andere Dienste Daten über dasselbe Netzwerk übertragen können, gilt es, den Daten möglichst so den Weg zu weisen, dass redundante Datenübermittlung minimiert oder im optimalen Fall sogar ausgeschlossen werden kann.

Ein solches Problem – das *Steinerbaumproblem* – hat Jakob Steiner [Wik08] schon im 19. Jahrhundert formuliert. Das hier behandelte *Steinerwaldproblem* ist eine Erweiterung des Steinerbaumproblems, bei dem nicht nur bestimmte Knoten eines Netzwerkes, sondern jeweils Teilmengen zu einem Baum und somit zu einem Wald zusammengeführt werden sollen. Da Steinerwald jedoch NP-schwer ist, werden wir, um es effizient und, zum Beispiel für obig genannte Netzwerke, akzeptabel gut lösen zu können, ein lineares Programm als Hilfsmittel verwenden, um einen Approximationsalgorithmus entwickeln, wie es Vaziranifür Steinerwälder ausgearbeitet hat ([Vaz01], Kapitel 22). Zuletzt werden wir den Approximationsalgorithmus bezüglich Güte und Laufzeit analysieren.

Problem 3.1 (Steinerwald)**.** *Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit Kostenfunktion $c : E \to \mathbb{Q}^+$ sowie eine Sammlung disjunkter Teilmengen $S_1, \ldots, S_k$ von V.*

Gesucht ist ein Teilgraph G' von G mit minimalen Kosten, in welchem jeweils alle Knotenpaare aus S_i miteinander verbunden sind.

3.1.1 Primales und duales LP

Nachdem wir das Problem formalisiert haben, wollen wir ein ILP dazu angeben. Dazu muss aber zuvor der Umstand, dass Knoten derselben Teilmenge S_i verbunden werden müssen, in einen mathematischen Rahmen gepackt werden.

Beginnen wir mit der *Verbindungsbedarfsfunktion*. Diese ordnet zwei Knoten u und v den Wert 1 zu, wenn sie verbunden werden müssen:

$$r(u, v) = \begin{cases} 1, \text{ wenn } u \text{ und } v \in S_i \\ 0, \text{ sonst} \end{cases}$$

Als nächstes definieren wir eine *Schnittbedarfsfunktion* $f : 2^V \to \{0, 1\}$, die die minimale Anzahl an Kanten festlegt, welche einen Schnitt $(S, \overline{S})$ kreuzen muss:

$$f(S) = \begin{cases} 1, \text{ wenn } \exists u \in S_i \text{ und } v \in \overline{S_i} \text{ mit } r(u, v) = 1 \\ 0, \text{ sonst} \end{cases}$$

Ein Beispiel, wie sich diese Funktion in einem Graphen ablesen lässt, ist durch die Abbildung 3.1 gegeben.

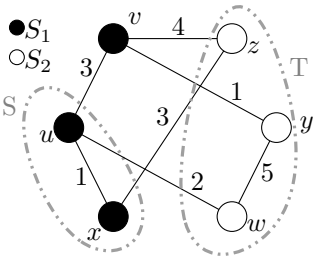

Abbildung 3.1: Hier ist $f(S) = 1$, da v noch außerhalb der Menge S liegt und $f(T) = 0$, da alle zu verbindenden Knoten aus S_2 schon innerhalb der Menge T liegen.

Mit diesen Funktionen können wir nun das primale LP aufstellen:

Minimiere

$$\sum_{e \in E} c_e x_e \tag{3.1}$$

unter den Nebenbedingungen

$$\sum_{e : e \in \delta(S)} x_e \geq f(S), \quad S \subseteq V \tag{3.2}$$

$$x_e \geq 0, \quad e \in E \tag{3.3}$$

wobei $\delta(S)$ die Menge an Kanten ist, die den Schnitt $(S, \overline{S})$ kreuzen. Die Variable x_e sei genau dann 1, wenn die Kante verwendet wird, ansonsten 0. Die Nebenbedingungen 3.2 stellen die Erfüllung des Schnittbedarfes sicher, denn sobald für eine Menge $f(S) = 1$gilt, muss mindestens eine Kante e aus $\delta(S)$ verwendet werden und damit muss x_e gleich 1 sein. Der Approximationsalgorithmus wird zwar ausschließlich ganze Zahlen verwenden, doch für das duale Programm benötigen wir das oben beschriebene relaxierte lineare Programm 3.1. Das Duale lautet folgendermaßen:

Maximiere

$$\sum_{S \subseteq V} f(S) \cdot y_S \tag{3.4}$$

unter den Nebenbedingungen

$$\sum_{S : e \in \delta(S)} y_S \leq c_e, \quad e \in E \tag{3.5}$$

$$y_S \geq 0, \quad S \subseteq V \tag{3.6}$$

Es fällt auf, dass wir für jede Teilmenge S von V nicht nur eine Nebenbedingung im primalen sowie im dualen Programm erhalten, sondern auch eine neue Variable y_S. Wie mit den exponentiell vielen Werten und Bedingungen umgegangen wird, zeigt das folgende Kapitel. Zunächst wollen wir uns der Bedeutung desdualen Programms klar werden.

Anschaulich sagt das duale Programm aus, dass wir Mengen S im Graphen suchen, bei denen in der Menge selbst wie auch im Komplement $\overline{S}$ zu verbindende Knoten sind, und eine Verbindung dahin uns möglichst wenig kostet. Dabei muss stets gelten, dass die Werte y_S aller Mengen, die eine Kante e imSchnitt haben, die Kosten c_e der Kante nicht übersteigen.

3.1.2 Approximationsalgorithmus

Der Algorithmus, den wir in diesem Abschnitt vorstellen werden startet bei einer primalen und dualen Lösung von 0. Er wird mit Hilfe des *primal-dualen Schemas* von jeweils allen Knoten einen Baum wachsen lassen, die mit anderen Knoten verbunden werden müssen. Dies geschieht, indem die y_S dieser Knoten, die im ersten Iterationsschritt eine ein-elementige Menge S darstellen, solange erhöht werden, bis eine der Nebenbedingungen $\sum_{S:e\in\delta(S)} y_S \leq c_e$ mit Gleichheit erfüllt ist. Damit erfüllen wir die primale Schlupfbedingung für die Kante e und können, um eine optimale Lösung zu erhalten, deren x_e auf 1 setzen.

Nun wird also diese Kante e ausgewählt und wir beginnen in der nächsten Iteration eine neue Menge S', die die Vereinigung von S mit dem Knoten inzident zur gewählten Kante e darstellt, zu erhöhen. Um die Nebenbedingungen nicht zu verletzen, wird die ursprüngliche Menge S nicht mehr erhöht. Das duale Programm bestimmt also die Kanten, die ausgewählt werden sollen.

Da wir also nur die y_S-Variablen von Mengen beachten müssen, die wir verwenden – alle anderen sind gleich 0 – reduzieren wir durch schlichtes Weglassen aller nicht verwendeten Mengen exponentiell vielen Nebenbedingungen und y_S-Variablen auf linear viele. Befinden wir uns in einer Folgeiteration, so erstellen wir neue Variablen und Nebenbedinungen für hinzugekommene Mengen S'.

Wird dieses Verfahren für alle zu verbindenden Knoten durch- und fortgeführt, so entsteht ein Wald wachsender Bäume, wobei vorzugsweise die Kanten gewählt werden, die zwei Bäume kostengünstig miteinander verbinden.

Um den Algorithmus aufstellen zu können, fügen wir zunächst Bezeichnungen für besondere Mengen ein. Hierzu klassifizieren wir eine Teilmenge S von V als *unbefriedigt*, wenn $f(S) = 1$ ist und keine Kante den Schnitt $(S, \overline{S})$ kreuzt. Des weiteren heißt eine unbefriedigte Menge *aktiv*, wenn sie minimal unter Inklusion ist. Diese Menge ist dann genau die Einschließende eines bisher gewachsenen Baumes, deren Wert y_S aktuell erhöht werden kann und

soll. Damit haben wir alle Begriffen eingeführt, um den Algorithmus formal aufzuschreiben:

Algorithmus 5 : Steinerwald

output : Restkantenmenge F'
input : Graph $G = (V, E)$
1. Initialisierung:
$F \leftarrow \emptyset$
foreach $S \subseteq V$ **do**
  $y_S \leftarrow 0$
2. Kantenzuwachs:
while $\exists$ *unbefriedigte Menge* **do**
  repeat
    foreach *aktiver Schnitt* y_S **do**
      erhöhe y_S
  until *Kante e berührt die Grenze*
  $F \leftarrow F \cup \{e\}$
3. Säuberung:
$F' = \{e \in F \mid F - \{e\}$ ist primal zulässig $\}$

Schritt 3 wurde eingeführt, um überflüssige Kanten wieder zu entfernen. Betrachten wir den Ergebnisgraphen als Wald und davon einen beliebigen Baum. Als Wurzel des Baumes wählen wir einen Knoten, der in einer der Mengen S_i, der jeweils zu verbindenden Knoten, und selbstverständlich im gewählten Baum liegt.Redundante Kanten sind nun solche, deren Teilbaum unter ihnen nur noch Knoten enthält, die nicht verbunden werden müssen. Anschaulich wären dies genau die Äste, die in die „falsche“ Richtung gewachsen sind. Abbildung 3.2 veranschaulicht den Sachverhalt.

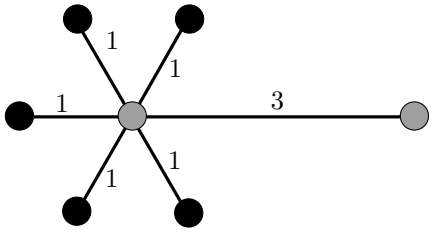

Abbildung 3.2: Bei diesem Stern müssen die Knoten inzident zur Kante mit dem Gewicht 3 verbunden werden. Die restlichen Kantengewichte sind 1. Dann wählt Algorithmus 5 zuerst die redundanten, mit 1 gewichteten Kanten aus.

3.1.3 Analyse

Bis zu diesem Zeitpunkt haben wir einen Algorithmus angegeben und behauptet, er approximiere das Steinerwaldproblem. Nun wollen wir eine Aus-

sage darüber treffen, ob er wirklich eine mögliche Lösung für Steinerwald liefert, wie nah sie an der optimalen Lösung kommt und wie schnell der Algorithmus diese Lösung findet.

Beginnen wir damit zu zeigen, dass Algorithmus 5 tatsächlich eine gültige Lösung findet. Um dies sicherzustellen, müssen wir die Anforderungen, die das Problem Steinerwald stellt abhandeln.

Das wäre als erstes, dass wir einen Teilgraphen von G suchen, was jedoch schon aufgrund der Konstruktion des Algorithmus gegeben ist. Die zweite Bedingung, minimale Kosten, haben wir auf Grund der Komplexität des Problems aufgegeben und zeigen folglich nur noch, dass alle Knotenpaare verbunden sind.

Nach Schritt 2 sind alle zu verbindenden Knoten verbunden, sonst gäbe es noch einen unbefriedigten Schnitt und Schritt 2 wäre noch nicht beendet. Stellen wir also sicher, dass Schritt 3 keine Kanten löscht, durch welche eine bestehende Verbindung getrennt werden könnte. Dies könnte geschehen, wenn es mehrere mögliche Pfade zwischen zwei Knoten u und v gibt. Entfernt man gleichzeitig alle nicht-redundanten Kanten, so bleibt die Lösung immer noch zulässig, da sie zu jeder Zeit ein Wald von Bäumen, insbesondere also azyklisch, ist. Dieser Sachverhalt folgt, da nur *aktive* Mengen S erhöht werden und neue Kanten nur aus dem Schnitt $(S, \overline{S})$ dieser Mengen hinzugenommen werden, niemals aber innerhalb einer solchen. Somit gibt es von zu verbindenden Knoten u, v immer nur genau ein Pfad von u nach v. Alle Kanten darauf sind nicht-redundant und werden nicht gelöscht.

Nach dem Löschen von redundanten Kanten erreichen wir eine Approximationsgüte von 2, was über das duale Programm gezeigt werden kann, denn es gilt stets:

$$\sum_{e \in F'} c_e \leq 2 \cdot \sum_{S \subseteq V} y_S$$

Beweisskizze. Weil alle Nebenbedinungen aus dem dualen Programm 3.4 mit Gleichheit erfüllt werden, gilt:

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \left(\sum_{S : e \in \delta(S)} y_S \right)$$

Durch Vertauschen der Reihenfolge in der Summierung ergibt sich:

$$\sum_{e \in F'} c_e = \sum_{S \subseteq V} \left(\sum_{e \in \delta(S) \cap F'} y_S \right) = \sum_{S \subseteq V} \deg_{F'}(S) \cdot y_S \leq 2 \cdot \sum_{S \subseteq V} y_S$$

wobei $\deg_{F'}(S)$ die Anzahl an Kanten in F' ist, die den Schnitt $(S, \overline{S})$ kreuzen. Der letzte Schritt muss noch gezeigt werden. Er bedeutet jedoch anschaulich, dass im Durschnitt, der Grad aller aktiver Mengen höchstens 2 ist. Dies ist sowohl in Bäumen als auch in Wäldern gegeben [Vaz01]. □

Zuletzt wollen wir noch eine Aussage über die Laufzeit des Algorithmus machen. Dazu betrachten wir ihn Schritt für Schritt und setzen zum Schluss die einzelnen Ergebnisse wieder zusammen.

1. Schritt: $\mathcal{O}(1)$.

2. Schritt: betrachten wir die verschachtelten Schleifen jeweils getrennt:

   - die **foreach**-Schleife wird maximal s mal durchlaufen, wobei gilt: $s = |\bigcup S_i|$. Da aktive Mengen nur vereinigt werden und initial s davon vorhanden sind $\Rightarrow \mathcal{O}(s)$ Iterationen
   - abhängig vom maximalen Kantengewicht wird die **repeat**-Schleife wiederholt $\Rightarrow \mathcal{O}(c_{\max})$ Iterationen
   - im schlechtesten Falle wird die äußerste **while**-Schleife $|V|$ mal durchlaufen, da die Schleife spätestens dann endet, wenn alle Knoten verbunden sind $\Rightarrow \mathcal{O}(|V|)$ Iterationen

   Insgesamt egibt sich damit eine Laufzeit von $\mathcal{O}(s \cdot c_{\max} \cdot |V|)$

3. Schritt: $\mathcal{O}(|V|)$ Zeit, da wir nach dem 2. Schritt auf einem Baum, operieren und nicht mehr auf dem ganzen Graphen

Schätzen wir das Kalkül noch etwas nach oben ($s \sim |V|$) ab, so erhalten wir eine Laufzeit in $\mathcal{O}(c_{\max} \cdot |V|^2)$.

3.1.4 Zusammenfassung

Wir haben also eine Faktor-2-Approximation für ein wichtiges Netzwerkproblem mit einer Laufzeit von $\mathcal{O}(c_{\max} \cdot |V|^2)$. Ist dies gut für unser Beispiel aus der Einführung? In kleinen Netzen ja, denn der Algorithmus ist hier ausreichend schnell und garantiert ein sehr gutes Ergebnis. Ingroßen, wie dem Internet, eher nicht, da für den Algorithmus die gesamte Netztopologie bekannt sein muss und die Routingalgorithmen hier nur konstante Zeit benötigen sollten aber dafür ein unter Umständen ein deutlich schlechteres Ergebnis liefern dürfen.

Eine Erkenntnis, die aus diesem Abschnitt gewonnen werden soll, ist, dass man manche LPs gut approximieren kann, wenn man bestimmte Werte des primalen oder hier dualen Programmes synchron erhöht und iteriert, sobald Nebenbedingungen verletzt werden.

3.2 Steinernetzwerk

In diesem Abschnitt greifen wir das Netzwerkproblem aus Abschnitt 3.1 auf und erweitern die Aufgabenstellung darum, dass wir von nun an regeln wollen, welche Menge an Daten an eine andere Station übertragen werden soll.

Leider verursacht eine Verbindung sehr viel Rechenlast auf einem Host, wodurch man nur einen bestimmten Kredit an Last hat, den man auf die angeforderten Verbindungen verteilen muss. Aus Sicherheitsgründen bezüglich der Abhörbarkeit müssen die Daten jedoch durch verschiedene Kabel transportiert werden, was selbstverständlich Geld und Raum kostet, sodass in manchen Gegenden nur begrenzt viele Verbindungen angelegt werden können.

Nennen wir diese Erweiterung *Steinernetzwerk*. Im folgenden werden wir nach Vazirani [Vaz01] Kapitel 23 für das Steinernetzwerk ein Lineares Programm (LP) aufstellen und einen Algorithmus angeben, der das Problem Steinernetzwerk mit dem Faktor 2 approximiert. Dazu wird einen Teil der Technik aus Steinerwald – iteratives Vorgehen – mit einer weiteren Technik zur Annäherung optimaler Lösungen von LPs, dem *iterierten Runden*, kombiniert werden.sebas

Problem 3.2 (Steinernetzwerk)**.** *Gegeben sei ein ungerichteter Graph $G = (V, E)$, eine Kostenfunktion $c : E \to \mathbb{Q}^+$, die nicht der Dreiecksungleichung genügen muss, eine Verbindungsvoraussetzung $r : (V \times V) \to \mathbb{Z}^+$ und eine Kapazitätsfunktion $u : E \to \mathbb{Z}^+ \cup \{\infty\}$.*

Gesucht ist Multigraph G' über V, der durch herauskopieren von Kanten aus G entsteht. Er soll minimale Kosten haben und alle Knotenpaare u, v je $r(u, v)$ disjunkte Verbindungen aufweisen. Eine Kopie einer Kante e hat Kosten von $c(e)$ und darf zusätzlich höchstens $u(e)$ mal kopiert werden.

3.2.1 LP-Relaxierung

Bevor wir das LP aufstellen können, definieren wir uns wie schon für das Steinerwaldproblem 3.1.1 eine *Schnittbedarfsfunktion* $f : 2^V \to \mathbb{Z}^+$, die uns hier jedoch die größte Verbindungsvoraussetzung nennen soll, die den Schnitt $(S, \overline{S})$ kreuzt:

$$f(S) = \max\{r(u, v) \mid u \in S \text{ und } v \in \overline{S}\}.$$

Hiermit ist es uns nun möglich, das LP analog zu Steinerwald aufzustellen. Geändert hat sich nur, dass x_e auch größer als 1 werden kann und die Begrenzungen für die maximale Anzahl an Kopien pro Kante in die Nebenbedingungen aufgenommen wurden:

Minimiere

$$\sum_{e \in E} c_e x_e \tag{3.7}$$

unter den Nebenbedingungen

$$\sum_{e : e \in \delta(S)} x_e \geq f(S), \qquad S \subseteq V \tag{3.8}$$

$$x_e \geq 0, \qquad e \in E \text{ falls } u_e = \infty \tag{3.9}$$

$$u_e \geq x_e \geq 0, \qquad e \in E \text{ falls } u_e \neq \infty \tag{3.10}$$

3.2.2 Lösungsansatz

Nachdem wir das LP formuliert haben, stellt sich unmittelbar die Frage, wie wir es lösen. Der Ansatz, das primale-duale Schema mittels Synchronisation wie bei Steinerwald zu verwenden funktioniert hier nicht, da dieses Schema Kreise verhindert, die hier zu einer korrekten Lösung notwendig sein können.

Eine andere Möglichkeit wäre, wie zum Beispiel bei Vertex Cover, auf Ganzzahligkeit zu verzichten, das relaxierte LP zu lösen und anschließend Kanten aufzurunden, die größer sind als 1/2.

Leider liefert uns eine gebrochene optimale Lösung nicht unbedingt ein Ergebnis, das durch Runden eine ganzzahlige zulässige Lösung darstellt. Manche Instanzen des Problems mögen dies erfüllen, doch der *Petersen-Graph* (siehe Abbildung 3.3) tut dies nicht.

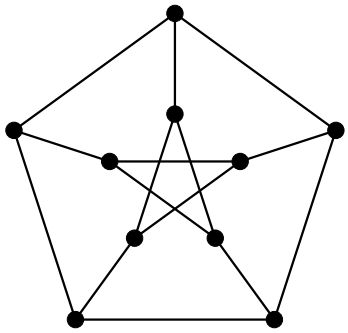

Abbildung 3.3: Petersen-Graph. Hier haben sowohl alle Verbindungsvoraussetzungen als auch alle Kantengewichte den Wert 1.

Die optimale gebrochene Lösung für die Instanz aus der Abbildung 3.3 ist 1/3 für jede Kante e, Kosten entstehen in der Höhe von 5. Anschaulich bedeutet dies, dass jede Kante zu einem Drittel ausgewählt wird. Damit haben wir Kosten von 15 Kanten mal 1/3 - ergibt 5. Die Kosten an einem Knoten entsprechen 1, wenn man alle inzidenten Kanten aufaddiert. Rundet man diese Lösung ab, so werden keine Kanten gewählt, rundet man sie auf, alle. Im ersten Fall haben wir keine zulässige Lösung, im zweiten eine, die von der Optimalen um den Faktor 3 abweicht.

Nehmen wir an, es gibt eine andere, *halbzahlige* und optimale Lösung mit Kosten von 5. Eine halbzahlige Lösung, wie man sie zum Beispiel bei Vertex Cover erhält, besteht nur aus den Werten 0, 1/2 und 1. In einer solchen Lösung kann es jedoch im Petersen-Graph keine Kanten geben, die mit $x_e = 1$ ausgewählt werden. Zusätzliche Kanten wären notwendig, um die Endpunkte dieser Kante mit dem Rest des Graphen zu verbinden, wodurch die Summe aller inzidenten Kanten dieser Endpunkte 1 übersteigt und damit auch die Gesamtkosten von 5 nicht erreicht werden können. Dies wäre keine optimale Lösung mit minimalen Kosten.

Bleiben Kanten mit x_e aus den Werten $\{0, 1/2\}$. Eine optimale Lösung sähe also so aus, dass an jedem Knoten zwei Kanten ausgewählt werden, jeweils zu 1/2. Das Problem dabei ist jedoch, dass eine solche Lösung einen Hamiltonkreis darstellt. Der Petersen-Graph besitzt aber keinen Hamiltonkreis. Folglich kann es keine optimale halbzahlige Lösung von Steinernetzwerk geben und wir müssen einen anderen Ansatz wählen.

Für die Approximation behelfen wir uns mit Extrempunktlösungen. Extrempunktlösungen liegen im Lösungspolyeder in einer Ecke. Sie sind nicht notwendigerweisehalbzahlig und jeweils affin unabhängig zu weiteren Extrempunktlösungen. In diesem speziellen Problem Steinernetzwerk erfüllt jedoch immer mindestens eine ihrer Komponenten die Eigenschaft $x_e \geq 1/2$, womit wir den Approximationsalgorithmus aufstellen können ([Vaz01], Abschnitte 23.3 und 23.4).

3.2.3 Approximationsalgorithmus

Wir wollen uns nun dem Algorithmus mittels iteriertem Runden widmen:

Algorithmus 6 : Steinernetzwerk

input : Graph $G = (V, E)$
output : Graph H
$H \leftarrow \emptyset$, $f' \leftarrow f$.
while $f' \not\equiv \mathbf{0}$ **do**
  finde Extrempunktlösung $\boldsymbol{x}$ für das LP mit
    Schnittbedarfsfunktion f'
  foreach *Ecke* e *mit* $x_e \geq \frac{1}{2}$ **do**
    $H = H \cup \lceil x_e \rceil \cdot \{e\}$
    $u_e = u_e - \lceil x_e \rceil$
  Aktualisiere f': **for** $S \subseteq V$ **do**
    $f'(S) \leftarrow f(S) - |\delta_H(S)|$

Wie beim linearen Programm für Steinerwald, haben wir hier auch das Problem exponentiell vieler Nebenbedinungen. Hier behelfen wir uns mit einem *Separationsorakel*, welches bei einer gegeben Lösung die Zulässigkeit sicherstellt oder eine verletzte Nebenbedingung nennt. Damit das Separationsorakel bei der Validierung der gefundenen Lösung $\boldsymbol{x}$ nicht alle Nebenbedingungen überprüft, konstruieren wir einen Hilfsgraphen über der Knotenmenge V mit Kantengewichten x_e für alle Kanten e. Darin wird nun mit einer Maximaler-Fluss-Routine sichergestellt, dass zwischen je zwei Knoten u und v ein Fluss von mindestens $r(u, v)$ besteht. Ist dies nicht erfüllt, so kann leicht ein verletzter Schnitt angegeben werden. Dies erfolgt in polynomieller Zeit, sodass wir mit Hilfe des Separationsorakels das lineare Programm in polynomieller Zeit lösen können [Lü04].

Haben wir nun eine Iteration berechnet, so erstellen wir eine neue Instanz des Problems, indem wir gerundete Kanten aus dem Graphen G entfernen,

die nächste Extrempunktlösung $\boldsymbol{x}'$ unter der aktualisierten Schnittbedarfsfunktion f' mit dem Separationsorakel suchen und wiederum Kanten runden, deren Wert $1/2$ übersteigt. Dieses Verfahren nennt man *iteriertes Runden*.

Bisher noch nicht erwähnt wurde, wie man mit dem Aktualisieren des Schnittbedarfes f' umgeht – schließlich müssen wir hier ebenso exponentiell viele Werte in nur polynomieller Zeit erneuern, um effizient zu bleiben. Das Aktualisieren kann jedoch schlicht weggelassen werden, da eine Nebenbedingung für eine Lösung $\boldsymbol{x}'$ unter der Schnittbedarfsfunktion f' nach dem Aktualisieren genau dann nicht erfüllt ist, wenn sie für eine Lösung $\boldsymbol{x}$ unter der Schnittbedarfsfunktion f vor dem Aktualisieren schon nicht erfüllt war.

Dass Algorithmus 6 eine Approximationsgarantie von 2 erreicht, kann mittels Induktion über die Iterationstiefe gezeigt werden. Hierbei werden dieKosten in der ersten Iteration, die durch das Runden maximal doppelt so groß wie die optimalen sind, mit den Kosten einer Folgeiteration verglichen. Da nur Kanten mit einem Wert größer als $1/2$ gerundet werden, werden nach der ersten Iteration maximal doppelte Kosten verursacht, was sich in Folgeiterationen nicht mehr ändert.

In jeder Iteration ein lineares Programm zu lösen, spiegelt sich auch in der Laufzeit unseres Approximationsalgorithmus wieder. Sei $M(m, n)$ die Zeit, die für die Berechnung eines maximalen Flusses in einem Graphen mit n Knoten und m Kanten benötigt wird, sowie T die maximale Stellenanzahl von allen involvierten Zahlen in Binärrepresentation. Dann benötigt die Implementierung von Jain $O(m^2n(T + \log m)M(m, n) + m^2(T + \log m)P(m))$ Zeit, wobei $P(m)$ die Zeit ist, die benötigt wird, um eine Multiplikation von zwei $m \times m$-Matrizen durchzuführen [Jai01].

3.2.4 Zusammenfassung

Wieder erhalten wir einen Algorithmus, der ein ganzzahliges lineares Programm in polynomieller Zeit approximiert und für unser – wenn auch etwas fiktives – Beispiel ein Faktor-2 Ergebnis liefert. Was ist nun anders, als bei Steinerwald? Iterativ gehen wir in beiden Problemen vor. Auch errechnen wir eine Teillösung und verbessern diese. Während wir bei Steinerwald jedoch deterministisch jeweils abwechselnd die primale und duale Lösung durch synchrone Erhöhung dualer Werte verbessert haben, lösen wir hier in jeder Iteration ein lineares Programm, dessen Ergebnis dann gerundet wird. Auch wenn die behandelten Probleme undder Weg, eine Lösung zu bekommen, sich ähneln, so gibt es doch Unterschiede, die sich gerade beim Steinernetzwerkproblem in einer deutlich höheren Laufzeit nierderschlagen.

Literaturverzeichnis

[CLR05] Cormen, Thomas H., Charles E. Leiserson und Ronald L. Rivest: *Introduction to algorithms*. MIT Press [u.a.], 2nd.ed. Auflage, 2005. 5

[Ell06] Ellis, Leslie: *BitTorrent's Swarms Have a Deadly Bite On Broadband Nets*. http://www.multichannel.com/article/CA6332098.html, 2006. Online. 28

[GGPS98] Goldberg, L.A., P.W. Goldberg, C.A. Phillips und G.B. Sorkin: *Constructing Computer Virus Phylogenies*. Journal of Algorithms 26(1), Seiten 188–208, 1998. 4

[Jai01] Jain, Kamal: *A Factor 2 Approximation Algorithm for the Generalized Steiner Network Problem*. Combinatorica. Springer-Verlag, Januar 2001. 37

[Kar72] Karp, Richard M.: *Reducibility Among Combinatorial Problems*. In: *In Complexity of Computer Computations, Proc. Sympos. IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y.*., Seiten 85–103, 1972. 4

[Kar84] Karmarkar, N.: *A new polynomial-time algorithm for linear programming*. 1984. 9

[Lü04] Lübbecke, Marco: *Approximationsalgorithmen Sommersemester 04*. http://www.math.tu-berlin.de/~luebbeck/teaching/approx06/approx04.pdf, Juli 2004. Online. 36

[Vaz01] Vazirani, Vijay V.: *Approximation Algorithms*. Springer-Verlag, 2001. 3, 7, 8, 10, 11, 12, 15, 17, 27, 28, 32, 34, 36

[Wik08] WikipediA: *Jakob Steiner*. http://de.wikipedia.org/w/index.php?title=Jakob_Steiner&oldid=46954673, 2008. Online, Version vom 6. Juni 2008. 28