Vorlesung

Approximationsalgorithmen

WS 2013/2014

Prof. Dr. Georg Schnitger

21. November 2013

Anregungen und Fehler im Skript bitte an die Adresse

besser@thi.cs.uni-frankfurt.de

schicken.

Inhaltsverzeichnis

1 Einführung **7**
- 1.1 Optimierung und Approximation . . . 7
- 1.2 Parametrisierte Optimierung . . . 11
- 1.3 Grundlagen aus der Stochastik . . . 13
  - 1.3.1 Abweichungen vom Erwartungswert . . . 19

I Entwurfsmethoden **23**

2 Greedy-Algorithmen und Heuristiken **25**
- 2.1 INTERVALL-SCHEDULING . . . 26
- 2.2 HUFFMAN CODE . . . 27
- 2.3 KÜRZESTE WEGE . . . 31
- 2.4 MINIMALER SPANNBAUM und STEINER BAUM . . . 33
- 2.5 Matroide . . . 34
- 2.6 k-Matroide . . . 38
- 2.7 Sequenzierung . . . 41
  - 2.7.1 Shotgun-Sequenzierung und Fragment Assembly . . . 42
  - 2.7.2 Sequenzierung der Drosophila-DNA . . . 43
  - 2.7.3 SHORTEST COMMON SUPERSTRING . . . 44
  - 2.7.4 ZYKLUS-ÜBERDECKUNG . . . 49
- 2.8 Heuristiken für das Metrische TSP . . . 51

3 Dynamische Programmierung **57**
- 3.1 Das RUCKSACK-Problem . . . 58
- 3.2 BIN PACKING . . . 61
  - 3.2.1 Greedy Algorithmen . . . 61
  - 3.2.2 Ein polynomielles Approximationsschema . . . 64
- 3.3 MINIMUM MAKESPAN SCHEDULING . . . 67
  - 3.3.1 Greedy Algorithmen . . . 67
  - 3.3.2 Ein polynomielles Approximationsschema . . . 69
  - 3.3.3 MINIMUM MAKESPAN mit Prioritäten . . . 72
- 3.4 TSP und Dynamische Programmierung . . . 73

3.4.1 Ein exakter Algorithmus . . . 73
3.4.2 Das Euklidsche TSP . . . 73

4 Lokale Suche **79**
4.1 PLS: Polynomielle Suchprobleme . . . 82
4.2 FACILITY LOCATION und Cluster Probleme . . . 87
4.2.1 Lokale und Globale Minima . . . 88
4.2.2 Wieviele Iterationen werden benötigt? . . . 91
4.2.3 k-MEDIAN und k-CENTER . . . 92
4.3 Lokale Suche für Spannbaumprobleme . . . 93
4.3.1 MAX LEAF SPANNING TREE . . . 93
4.3.2 MIN DEGREE SPANNING TREE . . . 95
4.4 Lokale Suche in variabler Tiefe . . . 98
4.5 Der Metropolis Algorithmus und Simulated Annealing . . . 99
4.6 Evolutionäre Algorithmen . . . 103

5 Local Ratio **107**
5.1 VERTEX COVER . . . 107
5.2 Das Local Ratio Theorem . . . 109
5.3 FEEDBACK VERTEX SET für ungerichtete Graphen . . . 111
5.3.1 Eine 3-approximative Lösung . . . 114
5.3.2 Eine 2-approximative Lösung . . . 115

6 Branch & Bound **117**
6.1 Das RUCKSACK-Problem . . . 118
6.2 TSP: Branch & Bound mit 1-Bäumen . . . 119

II Lineare Programmierung **123**

7 Der Simplex-Algorithmus **133**
7.1 Vollständig unimodulare Programme . . . 143
7.2 Dualität und komplementäre Slackness . . . 146
7.3 Primal-duale Algorithmen . . . 155
7.3.1 VERTEX COVER . . . 156
7.4 SET COVER . . . 157

8 Die Interior-Point-Methode von Ye **163**
8.1 Bestimmung einer Anfangslösung . . . 165
8.2 Die Iteration . . . 166
8.2.1 Gradientenabstieg . . . 169
8.2.2 Der primale Schritt . . . 170
8.2.3 Der duale Schritt . . . 172
8.3 Zusammenfassung . . . 175

9 Lineare Programmierung und Approximation **177**
9.1 Scheduling für nicht-identische Maschinen 178
9.2 Approximation durch Rundung 181
9.2.1 VERTEX COVER 181
9.2.2 SET COVER 183
9.2.3 Ein Routing Problem 184
9.2.4 MAX-SAT 187
9.3 Die Held-Karp Schranke für TSP 190
9.4 Branch & Cut 194

III Schwierige Optimierungsprobleme **199**

10 Probabilistisch überprüfbare Beweise **207**
10.1 $\mathcal{PCP}$ und Approximierbarkeit 210
10.2 VERTEX COVER, INDEPENDENT SET und CLIQUE 212
10.3 Hastad's 3-Bit $\mathcal{PCP}$ 218
10.4 Parallel Repetition 220
10.4.1 2-Prover Spiele 221
10.4.2 LABEL COVER 223
10.4.3 SET COVER 224
10.4.4 Die Unique Games Vermutung 226

11 $\mathcal{NP} \subseteq \mathcal{PCP}(\text{poly}(n), O(1))$ **229**

Literaturverzeichnis **235**

Kapitel 1

Einführung

Die folgenden Quellen ergänzen und vertiefen das Skript:

- Für das Gebiet der Approximationsalgorithmen das Textbuch [Vaz].
  (Einige Kapitel des Skriptes folgen der Darstellung in [Vaz].)
- Für die lineare Programmierung die Texte [KV05] oder [Van].
- Für Teil III das Textbuch [AB].

Der Text [ACGK] sowie die Webseite `http://www.nada.kth.se/~viggo/wwwcompendium` charakterisieren die Approximationskomplexität einer Vielzahl wichtiger diskreter Optimierungsprobleme.

1.1 Optimierung und Approximation

Wir beginnen mit einer Bemerkung von Vasek Chvatal: „In den kommunistischen Ländern des Ostblocks in den 60'er und 70'er Jahren war es möglich, intelligent, ehrenhaft und ein Mitglied der kommunistischen Partei zu sein, aber es war nicht möglich alle drei Eigenschaften gleichzeitig zu verkörpern."
Algorithmen für NP-harte Optimierungsprobleme treffen eine vergleichbare Situation vor: Wir verlangen, dass sie

1. optimale Lösungen bestimmen,
2. in polynomieller Zeit rechnen
3. und dies für jede Instanz tun.

Hier gibt es nur einen Ausweg, wir müssen zulassen, dass approximitive Lösungen berechnet werden.
Wir betrachten Optimierungsprobleme der Form

$$\mathrm{opt}_x f(x_0, x) \text{ so dass } \text{Lösung}(x_0, x).$$

Mit anderen Worten, die Zielfunktion $f(x, x_0)$ ist bei gegebener Instanz x_0 über alle Lösungen x zu x_0 zu optimieren. Wir unterscheiden also zwischen der Instanz x_0 und einer Lösung x zu vorgegebener Instanz x_0. Wir nehmen an, dass opt $\in \{\min,\ \max\}$.

Beispiel 1.1 CLIQUE besitzt Graphen als Instanzen. Für einen gegebenen Graphen $x_0 = G$ sind all die Knotenmengen U Lösungen zu x_0, für die je zwei Knoten in U durch eine Kante verbunden sind. Die zu *maximierende* Zielfunktion ist die Größe einer Knotenmenge.

Wir beschränken uns auf „einigermaßen anständige“ Optimierungsprobleme, nämliche auf solche Optimierungsprobleme, für die das Lösungsprädikat und die Zielfunktion effizient berechenbar sind.

Definition 1.1
Ein $\mathcal{NP}$-Optimierungsproblem wird durch das Tripel $P = (\text{opt}, f, L)$ beschrieben.

(1) Es ist entweder „opt = min“ oder „opt = max“.

(2) Die Zielfunktion $f(x_0, x)$ ist durch einen deterministischen Algorithmus in polynomieller Zeit (in $|x_0| + |x|$) berechenbar.

(3) Es gibt einen effizienten Algorithmus, der entscheidet, ob x_0 eine Instanz für P ist.

(4) Zu einer Instanz x_0 heisst x eine Lösung, falls das Prädikat $L(x_0, x)$ wahr ist. Die folgenden Eigenschaften müssen erfüllt sein:

(4a) L besitzt nur polynomiell lange Lösungen, das heißt es muss

$$L(x_0, x) \Rightarrow |x| \leq p(|x_0|)$$

für ein Polynom p gelten.

(4b) Das Prädikat $L(x_0, x)$ ist in polynomieller Zeit (in $|x_0| + |x|$) durch einen deterministischen Algorithmus auswertbar.

$P = (\text{opt}, f, L)$ heißt ein $\mathcal{NP}$-Optimierungsproblemen, da die Sprachenversion

$$\text{Decision}_P = \{(x_0, k) \mid \text{es gibt } x \text{ mit } L(x_0, x) \text{ und } f(x_0, x) \begin{matrix}\leq \\ = \\ \geq\end{matrix} k\ \}$$

zur Klasse $\mathcal{NP}$ gehört.

Definition 1.2 **(a)** $\mathcal{NPO}$ ist die Klasse aller $\mathcal{NP}$-Optimerungsprobleme.
(b) Ein $\mathcal{NP}$-Optimierungsproblem heißt polynomiell beschränkt, wenn es ein Polynom q mit $|f(x_0, x)| \leq q(|x_0|)$ für jede Instanz x_0 und jede Lösung x gibt. $\mathcal{NPO} - \mathcal{PB}$ ist die Klasse aller polynomiell beschränkten $\mathcal{NP}$-Optimerungsprobleme.

Offensichtlich gehört CLIQUE zur Klasse $\mathcal{NPO} - \mathcal{PB}$, während das im nächsten Beispiel eingeführte RUCKSACK Problem keine polynomielle Beschränkung aufweist.

Beispiel 1.2 Im RUCKSACK Problem sind n Objekte mit Gewichten g_i und Werten w_i $(1 \leq i \leq n)$ vorgegeben ebenso wie eine Gewichtsschranke G. Der Rucksack ist mit einer Auswahl von Objekten so zu bepacken, dass einerseits die Gewichtsschranke nicht überschritten wird und dass andererseits der Gesamtwert maximal ist.
Die Instanz x_0 spezifiziert also die Gewichtsschranke G sowie das Gewicht und den Wert eines jeden Objekts. Die Lösungen entsprechen genau denjenigen Vektoren $x \in \{0,1\}^n$ mit $\sum_{i=1}^{n} g_i \cdot x_i \leq G$, es ist also

$$\text{Lösung}(x_0, x) \Leftrightarrow x \in \{0,1\}^n \text{ und } \sum_{i=1}^{n} g_i \cdot x_i \leq G.$$

Die Zielfunktion

$$f(x_0, x) = \sum_{i=1}^{n} w_i \cdot x_i$$

ist zu maximieren.

Beispiel 1.3 In LINEARE PROGRAMMIERUNG beschreibt die Instanz x_0 eine Matrix A sowie Vektoren b und c. x ist eine Lösung zur Instanz x_0 genau dann, wenn

$$A \cdot x = b \text{ und } x \geq 0$$

gilt. Die Zielfunktion

$$f(x, x_0) = c^T \cdot x$$

ist zu maximieren. LINEARE PROGRAMMIERUNG kann als ein $\mathcal{NPO}$ Problem aufgefasst werden, wenn wir nur Lösungen zulassen, deren Beschreibungslänge durch die Beschreibungslänge der Instanz polynomiell beschränkt sind. Dieser Ansatz ist auch vernünftig, da ansonsten Lösungen nicht repräsentierbar sind.
In GANZZAHLIGE LINEARE PROGRAMMIERUNG wird zusätzlich noch gefordert, dass jede Komponente einer Lösung ganzzahlig ist. In 0-1 PROGRAMMIERUNG werden schließlich nur binäre Vektoren als Lösungen zugelassen

Definition 1.3 Sei $P = (\text{opt}, \text{f}, \text{L})$ ein Optimierungsproblem und $\varepsilon : \mathbb{N} \to \mathbb{R}$ sei eine Funktion.
(a) x^* heißt eine optimale Lösung für die Instanz x_0, falls

$$f(x_0, x^*) = \text{opt } \{f(x_0, x) \mid x \text{ erfüllt das Prädikat } L(x_0, x) \}.$$

Wir setzen $\text{opt}_P(x_0) := f(x_0, x^*)$.
(b) Eine Lösung x von P für Eingabe x_0 heißt eine ε-approximative Lösung, falls

$$\varepsilon(|x_0|) \geq \max \left\{ \frac{\text{opt}_P(x_0)}{f(x_0, x)}, \frac{f(x_0, x)}{\text{opt}_P(x_0)} \right\}.$$

(c) Ein Algorithmus A heißt Approximationsalgorithmus für P, falls A für jede Instanz x_0 eine Lösung $A(x_0)$ berechnet. Wir sagen:

- A ist ε-approximativ (oder A ist ein ε-Approximationsalgorithmus), falls $A(x_0)$ für jede Instanz x_0 eine ε-approximative Lösung ist.

- A besitzt die Laufzeit $t : \mathbb{N} \to \mathbb{N}$, falls A für jede Instanz x_0 eine Lösung in Zeit $t(|x_0|)$ berechnet.

Wir unterscheiden Probleme nach der erreichbaren Approximationsqualität effizienter Algorithmen. Die nächste Definition führt die $\mathcal{APX}$-Klassen ein, $\mathcal{APX}$ steht abkürzend für „approximierbar“.

Definition 1.4 $\varepsilon : \mathbb{N} \to \mathbb{R}$ sei eine Funktion.

- $\varepsilon - \mathcal{APX}$ ist die Klasse aller Optimierungsprobleme in $\mathcal{NPO}$, für die es $O(\varepsilon)$-approximative Algorithmen gibt, die in polynomieller Zeit laufen.

- Wir kürzen $1 - \mathcal{APX}$ durch $\mathcal{APX}$ ab. $\mathcal{APX}$ ist also die Klasse aller Probleme, die effiziente c-approximative Algorithmen für eine natürliche Zahl c besitzen.

- Eine weitere wichtige Klasse ist log-$\mathcal{APX}$ und poly(log)-$\mathcal{APX}$: Hier sind also jeweils $O(\log n)$-approximative bzw. poly(log(n))-approximative Lösungen erforderlich, wobei n für die Beschreibungslänge der Instanz steht.

- Schließlich ist poly-$\mathcal{APX}$ die Klasse aller Probleme, für die poly(n)-approximative Lösungen effizient berechnet werden können.

Natürlich sind auch solche Optimierungsprobleme interessant, die beliebig scharf approximierbar sind.

Definition 1.5 Ein Optimierungsproblem besitzt ein polynomielles Approximationsschema, wenn es einen Approximationsalgorithmus A gibt, der für jede Instanz und für jedes $\delta > 1$ eine δ-approximative Lösung bestimmt; bei *fixiertem* δ muss die Laufzeit polynomiell in der Beschreibungslänge der Instanz sein.
$\mathcal{PTAS}$ (polynomial time approximation scheme) ist die Klasse aller Optimierungsprobleme mit einem polynomiellen Approximationsschema.

Ein polynomielles Approximationsschema erreicht zwar beliebig scharfe Approximationen, muss dafür mitunter aber enorm mit Laufzeit bezahlen. Man beachte, dass zum Beispiel Algorithmen mit einer Laufzeit von $n^{1/(\delta-1)}$ als polynomielle Approximationsschemata gelten: Für kleine Werte von δ sind solche Algorithmen nicht mehr praktikabel.
Die nächste Definition führt deshalb eine Problemklasse mit scharfen *und* schnellen Approximationsalgorithmen ein.

Definition 1.6 Ein Optimierungsproblem besitzt ein volles Approximationsschema, wenn es einen Approximationsalgorithmus A gibt, der für jede Instanz und für jedes $\delta > 1$ eine δ-approximative Lösung bestimmt. Die Laufzeit von A muss polynomiell in n und $\frac{1}{\delta-1}$ sein.
$\mathcal{FPTAS}$ (fully polynomial time approximation scheme) ist die Klasse aller Optimierungsprobleme mit einem vollen polynomiellen Approximationsschema.

Die folgende Aufgabe zeigt, dass volle polynomielle Approximationsschema selten sind.

Aufgabe 1

Wir nehmen an, dass $\mathcal{P} \neq \mathcal{NP}$ gilt.

(a) Sei $P = (\max, \text{f}, \text{Lösung})$ ein $\mathcal{NPO}$ Problem, wobei die Zielfunktion nur *ganzzahlige* Werte annimmt. Wir betrachten die unäre Sprachenversion L_P^* von P, also

$$L_P^* = \{(x_0, 1^k) \mid \text{es gibt } x \text{ mit Lösung}(x_0, x) \text{ und } f(x_0, x) \geq k \}.$$

Die Instanz x_0 ist binär kodiert, während der numerische Parameter k unär kodiert ist.

Zeige: Wenn L_P^* $\mathcal{NP}$-vollständig ist, dann besitzt P *kein* volles polynomielles Approximationsschema.

(b) **Zeige**, dass ein $\mathcal{NPO} - \mathcal{PB}$ Problem $P = (\max, \text{f}, \text{Lösung})$ *kein* volles polynomielles Approximationsschema besitzt, wenn

$$L_P = \{(x_0, k) \mid \text{es gibt } x \text{ mit Lösung}(x_0, x) \text{ und } f(x_0, x) \geq k \}$$

$\mathcal{NP}$-vollständig ist. Hier sind die Instanz x_0 wie auch der numerische Parameter k binär kodiert.

Schließlich bezeichne $\mathcal{PO}$ die Klasse aller Optimierungsprobleme in $\mathcal{NPO}$, die in polynomieller Zeit exakt gelöst werden können. Beachte, dass wir jetzt die Hierarchie

$$\mathcal{PO} \subseteq \mathcal{FPTAS} \subseteq \mathcal{PTAS} \subseteq \mathcal{APX} \subseteq \text{log-}\mathcal{APX} \subseteq \text{poly(log)-}\mathcal{APX} \subseteq \text{poly-}\mathcal{APX} \subseteq \mathcal{NPO}$$

erhalten haben.

Aufgabe 2
Wir nehmen an, dass $\mathcal{P} \neq \mathcal{NP}$. Zeige, dass $\mathcal{APX}$ eine echte Teilklasse von $\mathcal{NPO}$ ist.
Hinweis: Zeige, dass die 0-1 Programmierung nicht in $\mathcal{APX}$ liegt.

Wir werden die Klasse PO studieren und feststellen, dass das wichtige Problem der linearen Programmierung in PO liegt. Danach werden wir versuchen, (volle) polynomielle Approximationsschemata für wichtige Optimierungsprobleme zu entwerfen. Dabei wird sich herausstellen, dass der Nachweis, dass ein polynomielles Approximationsschema nicht existiert, völlig neue Methoden erfordert: Wir werden deshalb das neue Konzept der probabilistically checkable proofs (PCP) entwickeln. Beachte, dass der Nachweis, dass kein **volles** polynomielles Approximationsschema vorliegt hingegen oft sehr einfach ist.
Von großem Interesse sind auch die Klassen $\mathcal{APX}$ und $\log - \mathcal{APX}$, die eine Vielzahl wichtiger Optimierungsprobleme enthalten. Unser Ziel wird es dann sein, fundamentale Optimierungsprobleme zu studieren und „ihre“ jeweilige Klasse zu bestimmen.
Wir schließen das Kapitel mit einer Fragestellung der parametrisierten Optimierung.

1.2 Parametrisierte Optimierung

Wir führen n Tests aus, wobei die Ergebnisse einiger weniger Tests stark verfälscht sind. Leider wissen wir nicht welche Testergebnisse verfälscht sind, sondern nur, dass von einigen wenigen falschen Ergebnissen auszugehen ist. Um diese Ausreißer zu entdecken, wenden wir

das VERTEX COVER Problem an: Wir bauen einen ungerichteten Graphen mit n Knoten und setzen eine Kante $\{i, j\}$ ein, wenn die Ergebnisse des $i$ten und $j$ten Test zu stark voneinander abweichen. Die wenigen stark verfälschten Tests definieren die kleine Knotenmenge X, wobei alle Kanten mindestens einen Endpunkt in X besitzen. Wir nennen X eine Knotenüberdeckung. Wenn wir Glück haben, dann ist X eine kleinste Knotenüberdeckung, und die Bestimmung einer optimalen Lösung von VERTEX COVER ist ausreichend.
Haben wir einen Vorteil in der exakten Lösung von VERTEX COVER, wenn wir wissen, dass ein Graph $G = (V, E)$ kleine Überdeckungen besitzt? Die Antwort ist erfreulicherweise positiv. Wir nehmen an, dass G eine Knotenüberdeckung der (relativ kleinen) Größe höchstens K besitzt und suchen nach exakten Algorithmen mit einer Laufzeit der Form $f(K) \cdot \text{poly}(n)$.
Um das Optimierungsproblem zu lösen, genügt eine Lösung des Entscheidungsproblems

Hat G eine Knotenüberdeckung der Größe k?

Durch Binärsuche können wir dann das Optimum mit $O(\log_2 K)$-maliger Lösung des Entscheidungsproblems berechnen, wenn wir wissen, dass G eine Knotenüberdeckung der Größe K besitzt. Wann hat G eine Knotenüberdeckung der Größe k? Nur dann, wenn der Graph nicht zu viele Kanten besitzt.

Lemma 1.7 *Der Graph G habe n Knoten und eine Knotenüberdeckung der Größe k. Dann hat G höchstens $k \cdot n$ Kanten.*

Beweis: Eine Knotenüberdeckung $C \subseteq V$ muss einen Endpunkt für jede Kante des Graphen besitzen. Ein Knoten kann aber nur Endpunkt von höchstens $n - 1$ Kanten sein und deshalb hat G höchstens $k \cdot (n - 1) \leq k \cdot n$ Kanten. □

Wir haben natürlich die Möglichkeit, alle k-elementigen Teilmengen $U \subseteq V$ zu durchlaufen und zu überprüfen, ob U eine Knotenüberdeckung ist. Aber dies bedeutet die Inspektion von $\binom{n}{k}$ Teilmengen. Hier ist ein weitaus schnelleres Verfahren. ($G - w$ ist der Graph G nach Herausnahme von Knoten w.)

Algorithmus 1.8 **VC(G, k)**

(1) Setze $U = \emptyset$.

/* U soll eine Knotenüberdeckung der Größe höchstens k speichern. */

(2) Wenn G mehr als $k \cdot n$ Kanten hat, dann hat G nach Lemma 9.4 keine Knotenüberdeckung der Größe k. Der Algorithmus bricht mit einer „erfolglos“ Meldung ab.

/* Wir können ab jetzt annehmen, dass G höchstens $k \cdot n$ Kanten besitzt. */

Wenn $k = 0$, dann brich mit einer „erfolgreich“ Meldung ab.

/* Für $k = 0$ erfolgt also stets ein Abbruch. */

(3) Wähle eine beliebige Kante $\{u, v\} \in E$.

- Rufe $\mathbf{VC(G - u, k - 1)}$ auf. Wenn die Antwort „erfolgreich“ ist, dann setze $U = U \cup \{u\}$ und brich mit der Nachricht „erfolgreich“ ab.
- Ansonsten rufe $\mathbf{VC(G - v, k - 1)}$ auf. Wenn die Antwort „erfolgreich“ ist, dann setze $U = U \cup \{v\}$ und brich mit der Nachricht „erfolgreich“ ab.
- Ansonsten sind beide Aufrufe erfolglos. Brich mit der Nachricht „erfolglos“ ab.

/* Die Knotenüberdeckung U muss einen Endpunkt der Kante $\{u, v\}$ besitzen. Algorithmus 1.8 untersucht zuerst rekursiv die Option $u \in U$ und bei Mißerfolg auch die Option $v \in U$. */

Aufgabe 3
Ist die Abfrage auf die Kantenzahl in Schritt (2) notwendig oder genügt die Abfrage, ob Kanten für $k = 0$ existieren?

Unser Algorithmus scheint nur mäßig intelligent zu sein, aber seine Laufzeit ist dennoch wesentlich schneller als die Überprüfung aller möglichen k-elementigen Teilmengen. Warum? Algorithmus 1.8 erzeugt mit seinem ersten Aufruf $\mathrm{VC}(G, k)$ einen Rekursionsbaum. Der Rekursionsbaum ist binär und hat Tiefe k. Also haben wir höchstens 2^k rekursive Aufrufe, wobei ein Aufruf höchstens Zeit $O(n)$ benötigt.

Satz 1.9 *Algorithmus 1.8 überprüft in Zeit $O(2^k \cdot n)$, ob ein Graph mit n Knoten eine Knotenüberdeckung der Größe höchstens k besitzt.*

Wir können also das $\mathcal{NP}$-vollständige VERTEX COVER Problem effizient lösen, wenn k höchstens ein (nicht zu großes) Vielfaches von $\log_2 n$ ist.
Man bezeichnet das Forschungsgebiet von Problemen mit fixierten Parametern auch als parametrisierte Komplexität.

1.3 Grundlagen aus der Stochastik

Definition 1.10 Sei U eine endliche Menge.

(a) Eine Wahrscheinlichkeitsverteilung über U ist eine Funktion $\pi : U \to [0, 1]$, so dass $\sum_{u \in U} \pi(u) = 1$.

(b) Die Elemente von U heißen Elementarereignisse, Teilmengen von U heißen Ereignisse. Die Wahrscheinlichkeit eines Ereignisses $V \subseteq U$ wird durch

$$\mathrm{prob}[V] = \sum_{u \in V} \pi(u)$$

definiert.

(c) Es seien V_1 und V_2 zwei Ereignisse. Dann ist

$$\mathrm{prob}[\ V_1 \mid V_2\ ] = \frac{\mathrm{prob}[V_1 \cap V_2]}{\mathrm{prob}[V_2]}$$

die bedingte Wahrscheinlichkeit von V_1 bezüglich V_2.

(d) Eine Zufallsvariable über U ist eine Funktion $X : U \to \mathbb{R}$. Der Erwartungswert von X wird durch

$$\mathbb{E}[X] = \sum_{u \in U} \pi(u) \cdot X(u)$$

und die Varianz durch

$$V[X] = \mathbb{E}[\ X^2 - \mathbb{E}[X]^2\ ]$$

definiert.

Beispiel 1.4 Die **Gleichverteilung** oder **uniforme Verteilung** auf U weist jedem Elementarereignis $u \in U$ die Wahrscheinlichkeit $\frac{1}{|U|}$ zu.
Wir werfen eine Münze n-mal und definieren die Zufallsvariable X als die Anzahl der Versuche mit Ausgang Wappen. (X ist also über $U = \{\text{Wappen, Zahl}\}^n$ definiert.) Wenn p die Wahrscheinlichkeit für Wappen ist, dann ist

$$p_k = \text{prob}[\ X = k\ ] = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}.$$

$p = (p_0, \ldots, p_n)$ ist die **Binomialverteilung** mit den Parametern n und p. Beachte, dass $\mathbb{E}[\ X\ ] = n \cdot p$ und $V[\ X\ ] = n \cdot p \cdot (1-p)$.
Wir werfen solange eine Münze, bis wir den Ausgang Wappen erhalten. Die Zufallsvariable X halte diese Anzahl der Versuche fest. (Die Zufallsvariable ist über $U = \{\ \text{Zahl}^n \cdot \text{Wappen} \mid n \in \mathbb{N}\}$ definiert. Diesmal sind also abzählbar unendlich viele Elementarereignisse gegeben.) Wenn p wiederum die Wahrscheinlichkeit für Wappen ist, dann ist

$$p_k = \text{prob}[\ X = k\ ] = (1-p)^{k-1} \cdot p.$$

$p = (p_0, \ldots)$ ist eine **geometrische Verteilung**. Beachte, dass

$$\mathbb{E}[X] = \sum_{k=1}^{\infty} k \cdot (1-p)^{k-1} \cdot p = \frac{1}{p}.$$

Aufgabe 4
Uns steht eine stochastische Zufallsquelle zur Verfügung, die das Bit 0 (bzw. 1) mit Wahrscheinlichkeit $\frac{1}{2}$ ausgibt.

- Gegeben ist eine Zielverteilung $(p_1, \ldots, p_n)$. Entwirf einen Algorithmus, der mit Wahrscheinlichkeit p_i den Wert i ausgibt und dazu in der Erwartung nur $O(log(n))$ Bits aus der Zufallsquelle benötigt. Beachte, dass $p_1, \ldots, p_n$ beliebige reelle Zahlen sind.
- Bestimme die worst-case Zahl von Zufallsbits, mit der das obige Problem für $p_1 = p_2 = p_3 = \frac{1}{3}$ gelöst werden kann.

Aufgabe 5
Wir spielen ein Spiel gegen einen Gegner. Der Gegner denkt sich zwei Zahlen aus und schreibt sie für uns nicht sichtbar auf je einen Zettel. Wir wählen *zufällig* einen Zettel und lesen die darauf stehende Zahl. Sodann haben wir die Möglichkeit, diese Zahl zu behalten oder sie gegen die uns unbekannt gebliebene Zahl zu tauschen. Sei x die Zahl, die wir am Ende haben, und y die andere. Dann ist unser (möglicherweise negativer) Gewinn $x - y$.

- Wir betrachten Strategien S_t der Form *„Gib Zahlen $< t$ zurück und behalte diejenigen $\geq t$“*. Analysiere den Erwartungswert $E_{x,y}(Gewinn(S_t))$ des Gewinns dieser Strategie in Abhängigkeit von t, x und y.
- Gib eine randomisierte Strategie an, die für beliebige $x \neq y$ einen positiven erwarteten Gewinn für uns aufweist.

Beispiel 1.5 **Das Sekretär Problem**.
Wir möchten eine Stelle mit einem möglichst kompetenten Bewerber besetzen. Wir haben allerdings ein Problem: Wann immer sich ein Bewerber vorstellt, müssen wir am Ende des Bewerbungsgesprächs dem Bewerber absagen oder aber zusagen (und damit den Bewerbungsprozess beenden). Gibt es eine Strategie, die es uns erlaubt mit guter Wahrscheinlichkeit die beste Bewerbung auszuwählen?
Im allgemeinen Fall ist die Antwort sicherlich negativ: Wenn die erste Bewerbung bereits die beste ist, dann wird es uns schwerfallen sofort „zuzuschlagen“, da ja noch viele andere, möglicherweise bessere Bewerbungen anstehen. Erstaunlicherweise ist die Antwort aber positiv, wenn wir fordern, dass

- die Zahl n der Bewerber von vornherein bekannt ist und
- dass die Bewerber in zufälliger Reihenfolge zum Bewerbungsgespräch erscheinen.

n Bewerber $B_1, \ldots, B_n$ werden also eingeladen und ihre Reihenfolge π wird zufällig nach der Gleichverteilung ausgewürfelt. Nach dem Gespräch mit Bewerber B_i vergeben wir eine Note N_i. Wie können wir einen Bewerber mit höchster Note N_i auswählen, obwohl wir nach jedem Gespräch zu- oder absagen müssen? Wir betrachten die folgende Strategie für einen Parameter r mit $1 \leq r \leq n$:

(1) Wir sagen den ersten r Bewerbern ab, bestimmen aber die Höchstnote N, die von einem dieser Bewerber erreicht wird.

(2) Wir sagen dem ersten verbleibenden Bewerber zu, der eine bessere Note als N erhält. Wird die Höchstnote nicht mehr erreicht, dann sagen wir dem letzten Bewerber zu.

Wie gut ist diese Strategie und insbesondere, wie sollte r gewählt werden? Wir nehmen an, dass der beste Bewerber an Position opt erscheint. Wenn opt $\leq r$, dann haben wir verloren, denn wir haben dem besten Bewerber abgesagt.
Nehmen wir also an, dass opt $> r$ gilt, und der beste Bewerber bleibt also im Rennen. Beachte, dass wir genau dann erfolgreich sind, wenn der **zweitbeste unter den ersten opt Bewerbern** zu den ersten r Bewerbern gehört, also automatisch abgewiesen wird. Warum?

- Wenn der zweitbeste zu den ersten r Bewerbern gehört, der beste aber nicht, dann wird unsere Strategie den besten Bewerber auswählen.
- Wenn der zweitbeste nicht zu den ersten r Bewerbern gehört, dann wird unsere Strategie den Bewerbungsprozess vor dem besten Bewerber abbrechen, denn der zweitbeste erscheint ja vor Position opt.

Der zweitbeste kann an jeder der opt-1 Positionen erscheinen. Wenn also opt $> r$ gilt, dann erscheint der zweitbeste mit Wahrscheinlichkeit $r/(\text{opt}-1)$ unter den ersten r Bewerbern und damit ist $r/(\text{opt}-1)$ die Erfolgswahrscheinlichkeit unserer Strategie. Der beste Bewerber erscheint mit Wahrscheinlichkeit $1/n$ auf irgendeiner fixierten Position und die Erfolgswahrscheinlichkeit p unserer Strategie ist

$$p = \sum_{\text{opt}=r+1}^{n} \frac{1}{n} \cdot \frac{r}{\text{opt}-1}.$$

Wir approximieren die Summe durch das entsprechende Integral und erhalten

$$p = \sum_{\text{opt}=r+1}^{n} \frac{1}{n} \cdot \frac{r}{\text{opt}-1} \approx \frac{r}{n} \cdot \int_{x=r}^{n} \frac{1}{x}\, dx = \frac{r}{n} \cdot (\ln(n) - \ln(r)) = -\frac{r}{n} \cdot \ln(\frac{r}{n}).$$

Um die Erfolgswahrscheinlichkeit zu maximieren, müssen r so bestimmen, dass

$$f(r) = -\frac{r}{n} \cdot \ln(\frac{r}{n})$$

maximiert wird. Alternativ müssen wir das Maximum von $g(x) = -x \cdot \ln(x)$ über dem Intervall $[0, 1]$ maximieren. Wir differenzieren und erhalten

$$g'(x) = -\ln(x) - x \cdot \frac{1}{x} = -\ln(x) - 1.$$

Wenn wir $g'(x) = 0$ fordern, werden wir auf

$$\ln(x) = -1$$

und damit auf $x = 1/e$ geführt.

Satz 1.11 *Betrachte die Strategie, die die ersten $r = n/e$ Bewerber ablehnt und den ersten Bewerber akzeptiert, dessen Note mindestens so hoch ist wie die der ersten r Bewerber. Dann wird der beste Bewerber mit einer Wahrscheinlichkeit von ungefähr $\frac{1}{e}$ bestimmt.*

Als Nächstes beschreiben wir das „Monty Hall Problem“. In einer Game Show ist hinter einer von drei Türen ein Preis verborgen. Ein Zuschauer rät eine der drei Türen und der Showmaster Monty Hall wird daraufhin eine weitere Tür öffnen, hinter der sich aber kein Preis verbirgt. Der Zuschauer erhält jetzt die Möglichkeit, seine Wahl zu ändern. Sollte er dies tun? Wir betrachten die Ereignisse

- P_i, dass sich der Preis hinter Tür i befindet,
- Z_i, dass der Zuschauer zuerst Tür i wählt und
- M_i, dass Monty Hall Tür i nach der ersten Wahl des Zuschauers öffnet.

Wir nehmen o.B.d.A. an, dass der Zuschauer zuerst Tür 1 wählt und dass Monty Hall daraufhin Tür 2 öffnet; desweiteren nehmen wir an, dass der Zuschauer wie auch Monty Hall seine Wahl jeweils nach der Gleichverteilung trifft. Wir müssen die bedingten Wahrscheinlichkeiten prob[$P_1 \mid Z_1, M_2$] und prob[$P_3 \mid Z_1, M_2$] berechnen und beachten, dass prob[$P_1 \mid Z_1, M_2$] + prob[$P_3 \mid Z_1, M_2$] = 1 gilt, denn der Preis befindet sich nicht hinter der geöffneten Tür 2. Nach Definition der bedingten Wahrscheinlichkeiten ist

$$\begin{aligned} \text{prob}[\ P_1 \mid Z_1, M_2\ ] \cdot \text{prob}[\ Z_1, M_2] &= \text{prob}[\ Z_1, M_2 \mid P_1\ ] \cdot \text{prob}[\ P_1\ ] \text{ und} \\ \text{prob}[\ P_3 \mid Z_1, M_2\ ] \cdot \text{prob}[\ Z_1, M_2] &= \text{prob}[\ Z_1, M_2 \mid P_3\ ] \cdot \text{prob}[\ P_3\ ]. \end{aligned}$$

Wir bestimmen jeweils die rechten Seiten und erhalten

$$\text{prob}[\ P_1 \mid Z_1, M_2\ ] \cdot \text{prob}[\ Z_1, M_2] = (\frac{1}{3} \cdot \frac{1}{2}) \cdot \frac{1}{3},$$

denn Monty Hall kann die Türen 2 und 3 öffnen. Andererseits ist

$$\text{prob}[\ P_3 \mid Z_1, M_2\ ] \cdot \text{prob}[\ Z_1, M_2] = (\frac{1}{3} \cdot 1) \cdot \frac{1}{3},$$

denn Monty Hall kann nur Tür 2 öffnen. Also ist

$$\text{prob}[\ P_3 \mid Z_1, M_2\ ] \cdot \text{prob}[\ Z_1, M_2] = 2 \cdot \text{prob}[\ P_1 \mid Z_1, M_2\ ] \cdot \text{prob}[\ Z_1, M_2]$$

und wir erhalten prob[$P_3 \mid Z_1, M_2$] = $\frac{2}{3}$ und prob[$P_1 \mid Z_1, M_2$] = $\frac{1}{3}$: Der Zuschauer sollte seine Wahl stets ändern!
Eine kompaktere Argumentation ist wie folgt: Sei W die *Zufallsvariable*, die die erste Wahl des Zuschauers beschreibt und sei T die Zufallsvariable, die die richtige Tür beschreibt. Dann findet die Änderungsstrategie genau dann die richtige Tür, wenn W und T unterschiedliche Werte annehmen und dieses *Ereignis* hat die Wahrscheinlichkeit $\frac{2}{3}$. Fazit: Mit höherer Wahrscheinlichkeit war die erste Wahl schlecht und wird durch die veränderte Wahl richtig.

Definition 1.12 Zufallsvariablen $X_1, \ldots, X_n$ heißen unabhängig genau dann, wenn für alle $x_1, \ldots, x_n \in \mathbb{R}$

$$\text{prob}[\ X_1 = x_1 \wedge \cdots \wedge X_n = x_n\ ] = \Pi_{i=1}^{n} \text{prob}[\ X_i = x_i\ ].$$

Die Zufallsvariablen $X_1, \ldots, X_n$ heißen k-fach unabhängig, wenn $X_{i_1}, \ldots, X_{i_k}$, für jede Teilmenge $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, n\}$ von k Elementen, unabhängig sind.

Beachte, dass wiederholte Experimente unabhängigen Zufallsvariablen entsprechen. Unabhängige Zufallsvariablen $X_1, \ldots, X_n$ sind k-fach unabhängig für jedes $k < n$. Die Umkehrung gilt im Allgemeinen nicht.

Aufgabe 6
Konstruiere Zufallsvariablen X_1, X_2, X_3, die 2-fach unabhängig, aber nicht unabhängig sind.

Aufgabe 7
Eine *stochastische Zufallsquelle* liefert Nullen und Einsen, wobei jedes Zeichen unabhängig von der Vorgeschichte ist und die Wahrscheinlichkeit für eine Eins konstant p ist. Wir haben Zugang zu einer solchen Zufallsquelle Q, kennen aber die Wahrscheinlichkeit p nicht. Unsere Aufgabe ist es aus dieser Zufallsquelle eine stochastische Zufallsquelle zu konstruieren, die eine 1 mit Wahrscheinlichkeit $\frac{1}{2}$ erzeugt. Wir erhalten also den Ausgabestrom von Q und müssen aus den Bits neue Zufallsbits generieren. Dabei soll die erwartete Zahl der zwischen zwei neu generierten Bits verarbeiteten Zeichen aus Q den Wert $\frac{1}{p\cdot(1-p)}$ nicht überschreiten.
Wie kann diese Konstruktion gelingen?
Hinweis: Betrachte zwei aufeinanderfolgende Bits von Q.

Beispiel 1.6 **Bestimmung des durchschnittlichen Gehalts ohne das eigene Gehalt preiszugeben.**
n Personen $1, \ldots, n$ wollen ihr durchschnittliche Gehalt berechnen, aber keiner möchte dabei sein eigenes Gehalt mitteilen. Wenn alle Personen ehrlich sind, dann ist das durchschnittliche Gehalt korrekt zu bestimmen; wenn hingegen irgendwelche k Personen unehrlich sind, dann sollten sie nicht mehr als die Summe der Gehälter der restlichen $n-k$ Personen in Erfahrung bringen können.

Algorithmus 1.13 **Ein Protokoll**

(1) M sei eine Zahl, von der bekannt ist, dass M größer als die Summe der Gehälter ist. Jede Person i wählt zufällig Zahlen $X_{i,1}, \ldots, X_{i,i-1}, X_{i,i+1}, \ldots X_{i,n} \in \{0, \ldots, M-1\}$ und gibt $X_{i,j}$ an Person j weiter.

(2) Wenn G_i das Gehalt von Person i ist, dann berechnet i die Restklasse

$$S_i = G_i + \sum_{j,j\neq i}^{n} X_{j,i} - \sum_{j,j\neq i}^{n} X_{i,j} \bmod M.$$

Schließlich wird S_i bekannt gegeben.

(3) Jede Person i gibt S_i bekannt und berechnet dann $\sum_i S_i \bmod M$.

Kommentar: Wenn alle Personen ehrlich sind, dann ist

$$\sum_i S_i \bmod M \equiv \sum_i G_i + \sum_{i,j,j\neq i}^{n} X_{j,i} - \sum_{i,j,j\neq i}^{n} X_{i,j} \bmod M \equiv \sum_i G_i \bmod M = \sum_i G_i.$$

Also ist $\frac{1}{n} \cdot \sum_j S_j$ das gewünschte durchschnittliche Gehalt.

Warum ist dieses Protokoll sicher? Angenommen, die letzten k Personen sind Betrüger. Wir setzen $S_i^* = G_i + \sum_{j=1,j\neq i}^{n-k} X_{j,i} - \sum_{j=1,j\neq i}^{n-k} X_{i,j} \bmod M$. Eine ehrliche Person i veröffentlicht

$$S_i = S_i^* + \sum_{j=n-k+1}^{n} X_{j,i} - \sum_{j=n-k+1}^{n} X_{i,j} \bmod M.$$

Beachte, dass $\sum_{i=1}^{n-k} S_i^* = \sum_{i=1}^{n-k} G_i$.

Aufgabe 8
Zeige: Jede Wertekombination $(s_2^*, \ldots, s_{n-k}^*)$ der Zufallsvariablen $S_2^*, \ldots, S_{n-k}^*$ wird mit Wahrscheinlichkeit $M^{-(n-k-1)}$ ausgegeben. Also sind die Zufallsvariablen $S_2^*, \ldots, S_{n-k}^*$ unabhängig.

Als Konsequenz der Übungsaufgabe sind die Zufallsvariablen $S_2^*, \ldots, S_{n-k}^*$ unabhängig und gleichverteilt in $\{0, \ldots, M-1\}$. Man mache sich klar, dass diese Aussage auch für $S_2, \ldots, S_{n-k}$ gilt und die Betrüger lernen nichts, da jede Folge von $n-k-1$ Werten mit Wahrscheinlichkeit $\frac{1}{M^{n-k-1}}$ auftritt. Nur durch die Hinzunahme von S_1 können die Betrüger mit Hilfe der Beziehung $\sum_{i=1}^{n-k} S_i^* = \sum_{i=1}^{n-k} G_i$ das Gesamtgehalt der ehrlichen Personen bestimmen.

Lemma 1.14 *Seien X und Y Zufallsvariablen.*

(a) Es ist stets $\mathbb{E}[\ X+Y\ ] = \mathbb{E}[X] + \mathbb{E}[Y]$. Wenn X und Y unabhängig sind, dann gilt $\mathbb{E}[\ X \cdot Y\ ] = \mathbb{E}[X] \cdot \mathbb{E}[Y]$.

(b) Es ist stets $V[\ a \cdot X + b\ ] = a^2 \cdot V[\ X\ ]$.

Wenn X und Y unabhängig sind, dann ist $V[\ X+Y\ ] = V[\ X\ ] + V[\ Y\ ]$.

(c) Es seien V_1 und V_2 zwei Ereignisse. Dann ist stets

$$prob[\ V_1 \cup V_2\ ] \leq prob[V_1] + prob[V_2].$$

Wenn V_1 und V_2 unabhängige Ereignisse sind, dann ist

$$prob[\ V_1 \cap V_2\ ] = prob[V_1] \cdot prob[V_2].$$

Aufgabe 9
Zeige $E(X \cdot Y) = E(X) \cdot E(Y)$ für unabhängige diskrete Zufallsvariablen X und Y.

Aufgabe 10
Wir nehmen in einem Casino an einem Spiel mit Gewinnwahrscheinlichkeit $p = 1/2$ teil. Wir können einen beliebigen Betrag einsetzen. Geht das Spiel zu unseren Gunsten aus, erhalten wir den Einsatz zurück und zusätzlich denselben Betrag aus der Bank. Endet das Spiel ungünstig, verfällt unser Einsatz. Wir betrachten die folgende Strategie: `i:=0`

```
REPEAT
    setze 2^i$
    i:=i+1
UNTIL(ich gewinne zum ersten mal)
```

Bestimme den erwarteten Gewinn dieser Strategie und die erwartete notwendige Liquidität (also den Geldbetrag, den man zur Verfügung haben muss, um diese Strategie ausführen zu können).

1.3.1 Abweichungen vom Erwartungswert

Die folgenden Abschätzungen sind besonders hilfreich, wenn Abweichungen vom Erwartungswert untersucht werden.

Lemma 1.15 *Sei X eine Zufallsvariable.*

(a) Die Markoff-Ungleichung für eine Zufallsvariable $X \geq 0$ and $a \in \mathbb{R}_{>0}$:

$$\text{prob}[\ X > a\ ] \leq \frac{\mathbb{E}[X]}{a}.$$

(b) Die Tschebyscheff Ungleichung:

$$\text{prob}[\ |X - \mathbb{E}[X]| > t\ ] \leq \frac{V[X]}{t^2}.$$

(c) Die Chernoff Ungleichungen: $X_1, \ldots, X_k$ seien unabhängige binäre Zufallsvariablen, wobei $p_i = prob[\ X_i = 1\ ]$ die Erfolgswahrscheinlichkeit von X_i ist. Dann ist $E = \sum_{i=1}^{k} p_i$ die erwartete Anzahl von Erfolgen und es gilt

$$\begin{aligned}
\text{prob}[\ \sum_{i=1}^{k} X_i > (1+\beta)\cdot E\ ] &\leq \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{E} \leq e^{-E\cdot\beta^2/3} \\
\text{prob}[\ \sum_{i=1}^{k} X_i < (1-\beta)\cdot E\ ] &\leq \left(\frac{e^{-\beta}}{(1-\beta)^{1-\beta}}\right)^{E} \leq e^{-E\cdot\beta^2/2}
\end{aligned}$$

für jedes $\beta > 0$ (bzw. $0 < \beta \leq 1$ im zweiten Fall).

Beweis (a): Nach Definition des Erwartungswerts ist

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{u\in U} \pi(u)\cdot X(u) \\
&\geq \sum_{u\in U, X(u)>a} \pi(u)\cdot X(u) \\
&\geq \text{prob}[\ X > a\ ]\cdot a.
\end{aligned}$$

(b) Wir wenden die Markoff-Ungleichung mit $a = t^2$ für die Zufallsvariable $(X - \mathbb{E}[X])^2$ an und erhalten

$$\text{prob}[\ (X - \mathbb{E}[X])^2 > t^2\ ] \leq \frac{\mathbb{E}[(X - \mathbb{E}[X])^2]}{t^2}.$$

Offensichtlich ist $|X - \mathbb{E}[X]| > t \Leftrightarrow (X - \mathbb{E}[X])^2 > t^2$ und die Behauptung folgt, wenn wir beachten, dass

$$\begin{aligned}
\mathbb{E}[\ (X - \mathbb{E}[X])^2\ ] &= \mathbb{E}[\ X^2 - 2X\cdot\mathbb{E}[X] + \mathbb{E}[X]^2\ ] = \mathbb{E}[X^2] - 2\mathbb{E}[X]^2 + \mathbb{E}[X]^2 \\
&= \mathbb{E}[X^2] - \mathbb{E}[X]^2 = V[X].
\end{aligned}$$

(c) Wir zeigen nur, dass die erste Abschätzung gilt und stellen die zweite Abschätzung als Übungsaufgabe. Wir erhalten mit der Markoff-Ungleichung für beliebiges $\alpha > 0$

$$\begin{aligned}
\text{prob}[\ \sum_{i=1}^{k} X_i > t\ ] &= \text{prob}[\ e^{\alpha\cdot\sum_{i=1}^{k} X_i} > e^{\alpha\cdot t}\ ] \\
&\leq e^{-\alpha\cdot t}\cdot\mathbb{E}[e^{\alpha\cdot\sum_{i=1}^{k} X_i}] \\
&= e^{-\alpha\cdot t}\cdot\Pi_{i=1}^{k}\mathbb{E}[e^{\alpha\cdot X_i}].
\end{aligned}$$

In der letzten Gleichung haben wir benutzt, dass $\mathbb{E}[Y_1 \cdot Y_2] = \mathbb{E}[Y_1] \cdot \mathbb{E}[Y_2]$ gilt, wenn Y_1 und Y_2 unabhängige Zufallsvariablen sind. Wir ersetzen t durch $(1+\beta) \cdot E$, α durch $\ln(1+\beta)$ und erhalten

$$\begin{aligned} \text{prob}[\ \sum_{i=1}^{k} X_i > (1+\beta) \cdot E\ ] &\leq e^{-\alpha \cdot (1+\beta) \cdot E} \cdot \Pi_{i=1}^{k} \mathbb{E}[e^{\alpha \cdot X_i}] \\ &= (1+\beta)^{-(1+\beta) \cdot E} \cdot \Pi_{i=1}^{k} \mathbb{E}[(1+\beta)^{X_i}]. \end{aligned}$$

Die Behauptung folgt, da $\mathbb{E}[(1+\beta)^{X_i}] = p_i(1+\beta) + (1-p_i) = 1 + \beta \cdot p_i \leq e^{\beta \cdot p_i}$. □

Aufgabe 11
Für $\beta \leq 1$ gilt stets $\frac{e^{\beta}}{(1+\beta)^{1+\beta}} = e^{\beta - (1+\beta)\ln(1+\beta)} \leq e^{-\beta^2/3}$.

Aufgabe 12
$X_1, \ldots, X_k$ seien unabhängige binäre Zufallsvariablen mit $p_i = \text{prob}[\ X_i = 1\ ]$. Es sei $E^* \geq \sum_{i=1}^{k} p_i$. Zeige, dass

$$\text{prob}[\ \sum_{i=1}^{k} X_i > (1+\beta) \cdot E^*\ ] \leq \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}} \right)^{E^*} \leq$$

für jedes $\beta > 0$ gilt.

Aufgabe 13
Zeige die zweite Chernoff Ungleichung.

Beispiel 1.7 Wir nehmen an, dass ein Zufallsexperiment X vorliegt, dessen Erwartungswert wir experimentell messen möchten. Das Experiment ist aber instabil, d.h. die Varianz von X ist groß.
Wir „boosten“, wiederholen also das Experiment k mal. Wenn X_i das Ergebnis des $i$ten Experiments ist, dann setzen wir $Y = \frac{1}{k} \cdot \sum_{i=1}^{k} X_i$ und beachten, dass die Zufallsvariablen $X_1, \ldots, X_k$ unabhängig sind: Es gilt also

$$V[\ Y\ ] = \frac{1}{k^2} \cdot V[\ \sum_{i=1}^{k} X_i\ ] = \frac{1}{k^2} \cdot \sum_{i=1}^{k} V[\ X_i\ ] = \frac{1}{k} \cdot V[\ X\ ].$$

Wir haben die Varianz um den Faktor k gesenkt, aber den Erwartungswert unverändert gelassen, denn $\mathbb{E}[\ Y\ ] = \mathbb{E}[\ \frac{1}{k} \cdot \sum_{i=1}^{k} X_i\ ] = \frac{1}{k} \cdot \sum_{i=1}^{k} \mathbb{E}[\ X\ ] = \mathbb{E}[\ X\ ]$. Die Tschebyscheff Ungleichung liefert jetzt das Ergebnis

$$\text{prob}[\ |Y - \mathbb{E}[X]| > t\ ] = \text{prob}[\ |Y - \mathbb{E}[Y]| > t\ ] \leq \frac{V[Y]}{t^2} = \frac{V[X]}{k \cdot t^2}$$

und große Abweichungen vom Erwartungswert sind unwahrscheinlicher geworden.
Angenommen, wir haben erreicht, dass Y mit Wahrscheinlichkeit mindestens $p = 1 - \varepsilon$ in ein „Toleranzintervall“ $T = [\mathbb{E}[\ X\ ] - \delta, \mathbb{E}[\ X\ ] + \delta\ ]$ fällt. Können wir p „schnell gegen 1 treiben“? Wir wiederholen diesmal das Experiment Y und zwar m mal und erhalten wiederum unabhängige Zufallsvariablen $Y_1, \ldots, Y_m$. Als Schätzung des Erwartungswerts geben wir jetzt den *Median* M von $Y_1, \ldots, Y_m$ aus. Wie groß ist die Wahrscheinlichkeit, dass M nicht im Toleranzintervall T liegt?

Y liegt mit Wahrscheinlichkeit mindestens p in T. Wenn also der Median außerhalb des Toleranzintervalls liegt, dann liegen mindestens $\frac{m}{2}$ Einzelschätzungen außerhalb, während nur $(1-p)\cdot m = \varepsilon \cdot m$ außerhalb liegende Einzelschätzungen zu erwarten sind. Wir wenden die Chernoff Ungleichung an und beachten, dass $(1+\frac{1-2\cdot\varepsilon}{2\cdot\varepsilon})\cdot\varepsilon\cdot m = \frac{m}{2}$. Also erhalten wir mit $\beta = \frac{1-2\cdot\varepsilon}{2\cdot\varepsilon}$

$$\text{prob}[\ M \notin T\ ] \leq e^{-\varepsilon\cdot m\cdot\beta^2/3} = e^{-(1-2\cdot\varepsilon)^2\cdot m/(12\cdot\varepsilon)}$$

und die Fehlerwahrscheinlichkeit fällt negativ exponentiell, falls $\varepsilon < \frac{1}{2}$.

Aufgabe 14
Gegeben ist eine Menge S mit n paarweise verschiedenen Elementen. Der folgende randomisierte Algorithmus findet das k-kleinste Element.
Eingabe: k, S.

(1) Wähle zufällig ein Element y aus S gemäß der Gleichverteilung.

(2) Partitioniere $S \setminus \{y\}$ in zwei Mengen S_1 und S_2, so dass jedes Element in S_1 kleiner als y ist, und jedes Element in S_2 größer als y ist.

(3)
- Wenn $|S_1| = k-1$, dann gib y aus.
- Wenn $|S_1| \geq k$, dann $S := S_1$ und gehe zu (1).
- Ansonsten setze $S := S_2$, $k := k - |S_1| - 1$ und gehe zu (1).

Zeige, dass die erwartete Laufzeit $O(n)$ ist.
Beachte, dass die Partitionierung in (2) $|S|$ Schritte benötigt. Wir nehmen an, dass für (1) ein geeigneter Zufallsgenerator vorhanden ist, der in konstanter Zeit ein Element auswählt.

Teil I

Entwurfsmethoden

Kapitel 2

Greedy-Algorithmen und Heuristiken

In diesem Kapitel behandeln wir Greedy-Verfahren, eine wichtige Methode für den Entwurf von effizienten Approximationsalgorithmen.
Greedy-Algorithmen bestimmen eine Lösung iterativ, wobei jedesmal eine Entscheidung getroffen wird, die „lokal“ am vielversprechendsten ist. Getroffene Entscheidungen werden nicht revidiert.

Bemerkung 2.1 Diese „Definition“ ist alles Andere als präzise und erlaubt zum Beispiel nicht den Nachweis, dass Greedy-Algorithmen für ein bestimmtes Optimierungsproblem *keine* scharfen Approximationen erlauben.
Wir geben jetzt eine präzisere Definition, müssen aber mit dem Begriff eines *Datenelements* arbeiten. Beispiele von Datenelementen sind

- Jobs mit Startzeiten, Verarbeitungszeiten, Fristen und Strafen für Scheduling Probleme,
- Kanten und ihre Gewichte, bzw Knoten, ihre Gewichte und Nachbarn für Graph Probleme.

Wir stellen uns vor, dass eine Eingabe aus einer Menge von *tatsächlichen Datenelementen* besteht. Ein Greedy-Algorithmus G muss in jedem Schritt eine vollständige Ordnung auf *allen möglichen Datenelementen* bestimmen ohne die Eingabe zu kennen. G erhält dann das Datenelement der Eingabe mit höchster Priorität und muss dann eine nicht revidierbare Entscheidung für dieses Datenelement treffen: Die Art dieser Entscheidung ist problemabhängig.
Wenn der Begriff des Datenelements ebenso festliegt wie die Art der zu treffenden Entscheidung, dann erhalten wir die Klasse der *Priority-Algorithmen*. Man sagt, dass ein Priority-Algorithmus adaptiv ist, wenn die berechnete Ordnung nach einer getroffenen Entscheidung neu berechnet werden darf.

Wir lernen verschiedenste Greedy-Algorithmen kennen, einige Algorithmen berechnen exakte Lösungen, andere Algorithmen spielen die Rolle von Heuristiken und erlauben nur die Berechnung approximtiver Lösungen.

2.1 INTERVALL-SCHEDULING

In INTERVALL SCHEDULING sind Aufgaben $A_1, \ldots, A_n$ auf einem Prozessor auszuführen. Aufgabe A_i hat den Startpunkt s_i und den Endpunkt e_i. Das Ziel ist die Ausführung möglichst vieler Aufgaben $A_{i_1}, A_{i_2}, \ldots, A_{i_m}$, so dass sich die Zeitintervalle verschiedener Aufgaben nicht überlappen. Es wird also

$$[s_{i_k}, e_{i_k}) \cap [s_{i_l}, e_{i_l}) = \emptyset$$

gefordert, wann immer $k \neq l$. Der Ansatz, zuerst eine Aufgabe A_i mit kürzester Dauer zu wählen (also mit minimalem $s_i - e_i$), führt nicht zu einer optimalen Lösung. Betrachten wir folgendes Beispiel mit drei Aufgaben:

Aufgabe	Start	Ende	Dauer
A_1	$s_1 = 1$	$e_1 = 5$	$e_1 - s_1 = 4$
A_2	$s_2 = 5$	$e_2 = 10$	$e_2 - s_2 = 5$
A_3	$s_3 = 4$	$e_3 = 6$	$e_3 - s_3 = 2$

Die optimale Lösung führt A_1 und A_2 aus, bei unserem Ansatz wählen wir A_3. Auch das Kriterium, zuerst eine Aufgabe A_i mit minimalem s_i zu wählen, führt nicht zu einer optimalen Lösung. Betrachten wir folgendes Beispiel mit drei Aufgaben:

Aufgabe	Start	Ende
A_1	$s_1 = 1$	$e_1 = 10$
A_2	$s_2 = 2$	$e_2 = 3$
A_3	$s_3 = 3$	$e_3 = 4$

Die optimale Lösung führt A_2 und A_3 aus, bei unserem Ansatz wählen wir A_1. Ist dieses Problem doch nicht so einfach?

Algorithmus 2.1 **Ein Greedy-Algorithmus**

(1) Die Eingabe besteht aus den Aufgaben $A_1, \ldots, A_n$, den Startpunkten $s_1, \ldots s_n$ und den Endpunkten $e_1, \ldots, e_n$.

(2) Sortiere die Aufgabe gemäß aufsteigenden Endpunkten. Die sortierte Reihenfolge sei $e_{i_1} \leq e_{i_2} \leq \ldots \leq e_{i_n}$. Setze $E = 0$.

(3) FOR $k = 1$ TO n DO

IF $E < s_{i_k}$ THEN führe A_{i_k} aus und setze $E = e_{i_k}$.

Warum ist Algorithmus 2.1 korrekt? Sei

$$(A_{j_1}, A_{j_2}, \ldots, A_{j_m})$$

eine legale Ausführung einer maximalen Anzahl von Aufgaben. Wenn $e_{i_1} \leq e_{i_2} \leq \ldots \leq e_{i_m}$ die sortierte Folge der Endpunkte ist, dann führt der Greedy-Algorithmus zuerst Aufgabe A_{i_1} aus. Da $e_{i_1} \leq e_{j_1}$, ist auch

$$(A_{i_1}, A_{j_2}, A_{j_3}, \ldots, A_{j_m})$$

eine legale Ausführung. Den Korrektheitsbeweis vervollständigt man jetzt durch Induktion.

Bemerkung 2.2 Wir wählen Aufgaben als Datenelemente und repräsentieren ein Datenelement durch den Start- und Endpunkt seiner Aufgabe. Algorithmus 2.1 ist ein nicht-adaptiver Priority-Algorithmus, der alle möglichen Aufgaben nach ihrem Endpunkt vollständig ordnet: Jedes erhaltene Datenelement wird entweder akzeptiert oder verworfen, abhängig davon ob sein Startpunkt mit dem Endpunkt bereits ausgeführter Aufgaben kollidiert.

Aufgabe 15
Wir betrachten INTERVALL SCHEDULING, aber geben diesmal auch eine nicht-negative Gewichtung der Aufgaben an. Jetzt ist nicht die Anzahl ausgeführter Aufgaben zu maximieren, sondern die Summe der Gewichte ausgeführter Aufgaben.
Beschreibe einen effizienten Algorithmus für diese Variante.

Aufgabe 16
Eine Färbung der Kanten in einem ungerichteten Graphen $G = (V, E)$ ist *legal*, falls *keine* zwei Nachbarn (d.h. Kanten, die einen gemeinsamen Endpunkt besitzen) dieselbe Farbe tragen. Es ist klar, dass jeder Graph mindestens Δ Farben braucht, wobei Δ der maximale Grad von G ist. Andererseits kann man zeigen, dass $\Delta + 1$ Farben immer ausreichen!
Betrachte den folgenden *Greedy-Algorithmus*. Zuerst fixieren wir eine beliebige Reihenfolge der Kanten $e_1, \ldots, e_m$. Dann färbt der Algorithmus die Kanten gemäß dieser Reihenfolge. Kante e_1 erhält die Farbe 1. Wenn $e_1, \ldots, e_k$ gefärbt sind, dann erhält e_{k+1} die *kleinste* natürliche Zahl j als Farbe, die noch nicht an einen Nachbarn von e_{k+1} vergeben wurde.

(a) **Zeige**, dass dieser Algorithmus für jeden Graphen eine legale Färbung mit höchstens $2\Delta - 1$ Farben liefert.
Fazit: Dieser einfache Ansatz liefert für einen Online Algorithmus keine schlechte Approximationsleistung.

(b) **Zeige**, dass die obere Schranke $2\Delta - 1$ auch optimal ist: Für jedes Δ gibt es einen Graphen mit maximalem Grad Δ, so dass der Algorithmus $2\Delta - 1$ Farben benötigt. Beachte, dass der Graph und die Kantenreihenfolge bösartig gewählt werden können.

2.2 HUFFMAN CODE

Gegeben ist eine Datei bestehend aus Zeichen über einem endlichen Alphabet Σ. Für jedes Zeichen $a \in \Sigma$ sei $H(a)$ die Häufigkeit von a in der Datei. Unser Ziel ist eine Komprimierung der Datei. Dazu kodieren wir die Buchstaben aus Σ durch binäre Worte. Präfixcodes erlauben eine einfache Dekodierung:

Definition 2.2 Eine Funktion code : $\Sigma \to \{0,1\}^*$ heißt Präfixcode, falls für alle $a, b \in \Sigma$ mit $a \neq b$ gilt, dass code(a) kein Präfix von code(b) ist.

In HUFFMAN CODE ist ein Präfixcode zu bestimmen, der zu einer minimal langen Kodierung der Datei führt. Präfixcodes lassen sich durch binäre Bäume, die *Kodierbäume*, veranschaulichen. Wir geben eine induktive Beschreibung des Kodierbaums für einen Präfixcode „code". Anfänglich sei $v = \varepsilon$ die Wurzel des Kodierbaums.

- Ist die Konstruktion am Knoten v angelangt, sind alle Codewörter code(a), die mit $v0$ (bzw. $v1$) beginnen, im linken (bzw. rechten) Teilbaum der Wurzel abzuspeichern. Dementsprechend wird die Kante zum linken Kind $v0$ (bzw. zum rechten Kind $v1$) mit 0 (bzw. 1) beschriftet.

- Gibt es nur ein mit v beginnendes Codewort code(a), dann wird v zu einem Blatt und v wird mit dem Buchstaben a markiert.

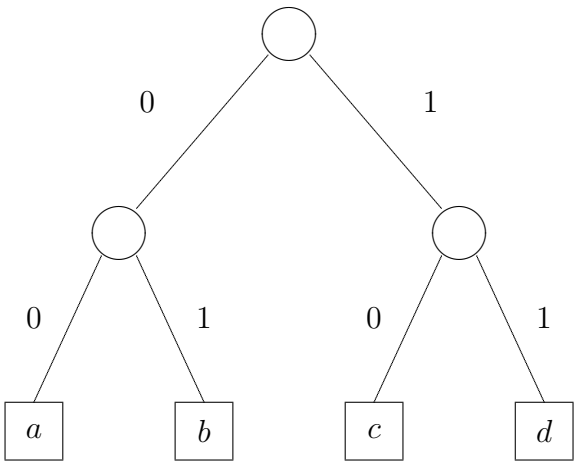
- Gibt es kein mit v beginnendes Codewort, dann wird v abgehangen.

Da ein Präfixcode vorliegt, werden sämtliche Buchstaben durch Blätter kodiert. Wird ein Blatt mit dem Buchstaben a markiert, dann definiert die Konkatenation der Markierungen auf dem Weg von der Wurzel zum Blatt das Codewort code(a).
Das Dekodieren einer kodierten Datei D gelingt problemlos: Lese die Bits von D von links nach rechts und durchlaufe den Kodierbaum entsprechend bis ein Blatt errreicht wird. Dekodiere den bisher gelesenen Präfix von D durch den Buchstaben des Blatts, gehe zurück zur Wurzel und wiederhole das Verfahren.
Wie lang ist die kodierte Datei D? Ihre Länge stimmt überein mit

$$||\text{code}|| = \sum_{a \in \Sigma} H(a) \cdot |\text{code}(a)|,$$

wobei $H(a)$ die Häufigkeit des Buchstaben $a \in \Sigma$ in der ursprünglichen Datei und $|\text{code}(a)|$ die Länge des Codewortes ist.

Beispiel 2.1 Sei $\Sigma := \{a, b, c, d\}$ sowie $H(a) = 1$, $H(b) = 2$, $H(c) = 4$ und $H(d) = 8$ vorgegeben. Dann beschreibt der Kodierbaum



den Präfixcode

$$code(a) := 00,\ code(b) := 01,\ code(c) := 10;\ code(d) := 11.$$

Die Länge des Codes ist $2 \cdot (1 + 2 + 4 + 8) = 30$. Wir werden sehen, dass dieser Präfixcode nicht optimal ist.

Huffman's Algorithmus verfolgt den Ansatz einer kurzen Kodierung für häufig auftretende Buchstaben. Anfänglich wird jedem Buchstaben ein Einzelknoten zugewiesen. Dieser Wald von Einzelknoten wird dann in einer Folge von Greedy-Schritten in einem Kodierbaum zusammengefasst.

Algorithmus 2.3 **Der Huffman Algorithmus.**

(1) Die Eingabe besteht aus dem Alphabet Σ und der Häufigkeitsfunktion $H : \Sigma \to \mathbb{N}$ für die Buchstaben der Datei.

(2) Erzeuge anfänglich zu jedem Buchstaben einen Einzelknoten.

(3) Füge die Buchstaben in einen gemäß aufsteigender Häufigkeit organisierten Heap ein.

(4) WHILE (Heap enthält noch mindestens zwei Buchstaben) DO

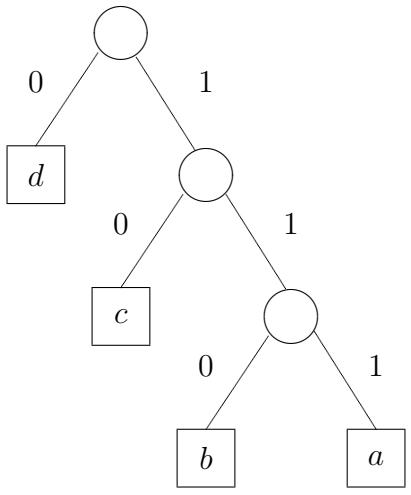
(4a) Entferne die beiden Buchstaben a und b minimaler Häufigkeit aus dem Heap.

(4b) Ein neuer Knoten ab wird erzeugt, der Knoten von a wird linkes Kind von ab und der Knoten von b wird rechtes Kind von ab im Baum.
Kommentar: Damit wird code(a) =code(ab)0 und code(b) =code(ab)1.

(4c) Der neue Buchstabe ab wird mit Häufigkeit $H(ab) := H(a) + H(b)$ in den Heap eingefügt.

(5) Berechne den Präfixcode code aus dem konstruierten Baum.

Für das obige Beispiel konstruiert Algorithmus 2.3 den Kodierbaum



Die Länge der komprimierten Daten ist

$$H(a) \cdot 3 + H(b) \cdot 3 + H(c) \cdot 2 + H(d) \cdot 1 = 3 + 6 + 8 + 8 = 25.$$

Warum funktioniert Huffman's Algorithmus? Sei code ein beliebiger optimaler Präfixcode, den wir durch seinen Kodierbaum T repräsentieren. Seien a, b die beiden Buchstaben geringster Häufigkeit. Der Huffman-Baum H, also der Kodierbaum des Huffman-Algorithmus, besitzt a und b als Markierungen der tiefsten Geschwisterblätter. Können wir T geringfügig, ohne Verlust der Optimalität, ändern, so dass auch T die Buchstaben a und b als Beschriftung der tiefsten Geschwisterblätter erhält?
Seien x, y Geschwisterblätter in T maximaler Tiefe. O.B.d.A. sei $H(a) \leq H(b)$ und $H(x) \leq H(y)$. Wenn wir in T die Blattmarkierungen a und x sowie b und y vertauschen, dann wird die Länge

$$\sum_{\sigma \in \Sigma} H(\sigma) \cdot ||\text{code}(\sigma)|| = \sum_{\sigma \in \Sigma} H(\sigma) \cdot \text{Tiefe}(\sigma)$$

der komprimierten Datei nicht ansteigen. (Tiefe(σ) ist die Tiefe des Blatts in T, das mit σ markiert ist.)
Wir möchten dieses Argument wiederholt anwenden. Dazu genügt es zu zeigen, dass jeder optimale Baum mit Geschwisterblättern $a, b \in \Sigma$ einen optimalen Code code$'$ für das kleinere Kodierproblem

$$\Sigma' := (\Sigma \cup \{ab\}) \setminus \{a, b\}$$

mit den Häufigkeiten

$$H'(\sigma) := \begin{cases} H(\sigma) & \sigma \neq ab \\ H(a) + H(b) & \text{sonst} \end{cases}$$

berechnet. Dies ist klar, da stets gilt

$$\begin{aligned} ||\text{code}|| &= \textstyle\sum_{\sigma \in \Sigma} H(\sigma) \cdot \text{Tiefe}(\sigma) \\ &= \textstyle\sum_{\sigma \in \Sigma \setminus \{a,b\}} H(\sigma) \cdot \text{Tiefe}(\sigma) + [H(a) + H(b)] \cdot [\text{Tiefe}(ab) + 1] \\ &= \underbrace{\sum_{\sigma \in \Sigma \setminus \{a,b\}} H(\sigma) \cdot \text{Tiefe}(\sigma) + [H(a) + H(b)] \cdot \text{Tiefe}(ab)}_{= ||\text{code}'||} + H(a) + H(b). \end{aligned}$$

Jeder optimale Präfixcode ist also insbesondere optimal auf Σ' und H'. Man zeigt jetzt durch Induktion über die Größe von Σ, dass Huffman's Algorithmus einen optimalen Präfixcode berechnet.
Betrachten wir die Laufzeit von Algorithmus 2.3. Der Aufbau des Heaps gelingt in Zeit $O(|\Sigma|)$. In jeder Iteration führen wir zwei DeleteMin- und eine Insert-Operation aus, wobei darauffolgend die Heapgröße um eins abnimmt. Wir erhalten also

$$O(|\Sigma|) + O(|\Sigma| \cdot \log_2 |\Sigma|) = O(|\Sigma| \cdot \log_2 |\Sigma|)$$

als Laufzeit, da jede DeleteMin- und Insert-Operation in logarithmischer Zeit gelingt.

Satz 2.4 *Für ein Alphabet Σ und vorgegebene Häufigkeiten bestimmt der Algorithmus von Huffman einen optimalen Präfixcode in Zeit $O(|\Sigma| \cdot \log_2 |\Sigma|)$.*

Bemerkung 2.3 Wir können den Algorithmus von Huffman als einen nicht-adaptiven Priority-Algorithmus interpretieren. Dazu wählen wir Paare (Buchstaben, Häufigkeit) als Datenelemente. Der Algorithmus von Huffman ordnet diese Paare nach steigender Häufigkeit und baut einen Kodierbaum ohne irgendwann getroffene Entscheidungen zurückzunehmen.

2.3 KÜRZESTE WEGE

Sei $G = (V, E)$ ein gerichteter Graph mit Quelle $s \in V$ und Kantengewichtung Länge : $E \to \mathbb{R}$, wobei nur nicht-negative Gewichte auftreten. In KÜRZESTE WEGE sind kürzeste Wege von s nach v für alle Knoten $v \in V$ zu bestimmen.

Algorithmus 2.5 **Dijkstras Algorithmus**

(0) Die Eingabe besteht aus einem gerichteten Graphen $G = (V, E)$, einer Quelle $s \in V$ und Kantenlängen Länge : $E \to \mathbb{R}$.

Erweitere die Definition der Kantenlängen um Länge$(u, v) = \infty$ für $(u, v) \in V^2 \setminus E$.

(1) Setze $S := \{s\}$ und $D_S(v) := \begin{cases} \text{Länge}(s, v) & v \neq s \\ 0 & \text{sonst.} \end{cases}$

(2) WHILE $(S \neq V)$ DO

(2a) Wähle $w \in V \setminus S$ mit $D_S(w) = \min\{D_S(v) \mid v \in V \setminus S\}$ und setze $S := S \cup \{w\}$.

(2b) Berechne $D_S(v) := \min\{D_S(v), D_S(w) + \text{Länge}(w, v)\}$ für jeden Nachfolger v von w.

Kommentar: $D_S(v)$ stimmt also überein mit der Länge eines kürzesten S-Weges von s nach v, wobei ein S-Weg, mit Ausnahme seines Endpunkts, nur Knoten in S besuchen darf.

(Algorithmus 2.5 kann so modifiziert werden, dass nicht nur die minimale Länge, sondern auch der zugehörige Weg ausgegeben wird. Wie?) Algorithmus 2.5 berechnet Teilmengen $S \subseteq V$ sowie ein Distanzarray D_S mit den folgenden Eigenschaften.

1.) Für jeden Knoten $v \in S$ verläuft der kürzeste Weg von s nach v vollständig in S und besitzt die Länge $D_S(v)$,

2.) Für jeden Knoten $v \in V \setminus S$ ist $D_S(v)$ die minimale Länge eines S-Weges von s nach v.

Der Algorithmus erweitert S beginnend mit $S = \{s\}$ iterativ zu $S = V$ und aktualisiert jeweils D_S. Durch Induktion über die Größe von S zeigt man, dass die beiden obigen Eigenschaften stets erfüllt sind. Betrachten wir zum Beispiel den Induktionsschritt für $S_{\text{neu}} = S_{\text{alt}} \cup \{w\}$.
Angenommen, der kürzeste Weg von s nach w verläuft nicht vollständig in S_{neu}. Seien $s = u_1, u_2, \ldots, u_k = w$ die Knoten des Weges und sei u_i der erste Knoten, der nicht in S_{neu} liegt. Aber der kürzeste Weg von s nach u_i, mit inneren Knoten nur in S_{alt}, hat nach Induktionsvoraussetzung die Länge $D_{S_{\text{alt}}}(u_i)$, und es gilt

$$D_{S_{\text{alt}}}(u_i) + \underbrace{\sum_{j=i}^{k-1} \text{Länge}(u_j, u_{j+1})}_{>0} \leq D_{S_{\text{alt}}}(w).$$

Dies ist ein Widerspruch zur Wahl von w, denn wir hätten statt w den Knoten u_i wählen müssen. Wir haben also Eigenschaft 1.) nachgewiesen. Der Nachweis der Eigenschaft 2.) ist als Übungsaufgabe gestellt.

Wir betrachten schließlich die laufzeitbestimmenden Operationen:

- Am Anfang fügen wir $O(|V|)$ Werte ein.
- Wir löschen $|V|$-mal das Minimum und
- aktualisieren insgesamt $|E|$-viele Distanzwerte.

Mit Heaps als Datenstruktur hat Dijkstras Algorithmus für kürzeste Wege im Graphen $G = (V, E)$ die Laufzeit $O((|E|+|V|)\cdot\log_2|V|)$. Verwenden wir statt Heaps die mächtigeren Fibonacci-Heaps[1], dann kann die Laufzeit auf $O(|E| + |V| \cdot \log_2|V|)$ gedrückt werden.

Satz 2.6 *Der Algorithmus von Dijkstra löst das Single-Source-Shortest-Path Problem für einen Graphen $G = (V, E)$ in Zeit $O(|E| + |V| \cdot \log_2|V|)$.*

Bemerkung 2.4 Dijkstra's Algorithmus ist ein adaptiver Priority-Algorithmus, wenn wir

einen Knoten v, zusammen mit seinen Nachbarn und den Distanzen zu den Nachbarn

als das Datenelement von v wählen. Im ersten Schritt fordert Dijkstra's Algorithmus das Datenelement von s an und bestimmt die Distanzen zwischen s und seinen Nachbarn. (Wie kann der Knoten s angefordert werden? Indem alle möglichen Datenelemente zur Quelle s höhere Priorität als alle anderen Datenelemente erhalten).
In nachfolgenden Schritten wird die Ordnung geändert. Wenn der Knoten $v \notin S$ die kleinstmögliche Distanz $D_S(v)$ unter allen Knoten in $V \setminus S$ besitzt, dann erhalten alle möglichen Datenelemente von v die höchste Priorität.

[1]Ein Fibonacci-Heap führt k Einfüge- und Aktualisierungsoperationen in Zeit $O(k)$ und m DeleteMin-Operationen in Zeit $m \cdot \log(k + m)$ aus.

2.4 MINIMALER SPANNBAUM und STEINER BAUM

Sei G ein ungerichteter, zusammenhängender Graph und w_e eine Gewichtung der Kanten $e \in E$. In MINIMALER SPANNBAUM suchen wir einen leichtesten Spannbaum.
Algorithmus 2.7 baut einen Spannbaum iterativ, indem jeweils eine Kante e mit minimalem Gewicht betrachtet wird. Wenn durch Hinzunahme von e zum gegenwärtig konstruierten Spannwald kein Kreis entsteht, wird diese vielversprechende Kante hinzugenommen. Wir werden den Korrektheitsbeweis als Konsequenz eines allgemeineren Ergebnisses später erhalten.

Algorithmus 2.7 **Der Algorithmus von Kruskal**

(1) Die Eingabe besteht aus einem ungerichteten, zusammenhängenden Graphen $G = (V, E)$ und einer Kantengewichtung.

(2) Setze $K = \emptyset$. (K wird letztlich die Kantenmenge eines minimalen Spannbaums sein.)

(3) WHILE ($E \neq \emptyset$) DO

(3a) Bestimme eine Kante $e \in E$ mit minimalem Gewicht und setze $E = E \setminus \{e\}$.

(3b) Wenn $(V, K \cup \{e\})$ keinen Kreis besitzt, dann setze $K = K \cup \{e\}$.

Bemerkung 2.5 Kruskal's Algorithmus ist ein nicht-adaptiver Priority Algorithmus, wenn wir Kanten und ihre Gewichte als Datenelemente benutzen: Der Algorithmus fordert Kanten nach aufsteigendem Gewicht an und nimmt eine Kante genau dann auf, wenn kein Kreis geschlossen wird.

Im Problem STEINER BAUM ist ein ungerichteter Graph $G = (V, E)$ und eine Teilmenge TĄ $\subseteq V$ von „Terminals“ gegeben, ebenso wie eine Gewichtung $w_e \geq 0$ der Kanten $e \in E$. Das Ziel ist die Bestimmung eines Teilbaums S von G, so dass

(1) S ein Steinerbaum ist, also ein Baum, der alle Terminals in T von S überdeckt und

(2) S ein leichtester Baum mit der Eigenschaft (1) ist.

Wir betrachten den Graphen $G' = (T, E')$ auf den Terminals, wobei E' aus allen Kanten $\{t_1, t_2\} \subseteq T$ besteht. Die Kante $e = \{t_1, t_2\}$ erhält das Gewicht

$$w_e = \text{ Länge eines kürzesten Weges von } t_1 \text{ nach } t_2.$$

Aufgabe 17
(a) Zeige, dass das Gewicht eines minimalen Spannbaums für G' höchstens doppelt so groß wie das Gewicht eines optimalen Steinerbaums für G ist.
(b) Entwerfe einen effizienten 2-approximativen Algorithmus für STEINER BAUM.

2.5 Matroide

Wir geben jetzt eine formale Definition von *eingeschränkten* Greedy-Algorithmen und werden die Problemklassen, zu denen es eingeschränkte Greedy-Algorithmen gibt, genau charakterisieren.

Definition 2.8 Ein monotones Teilmengensystem über der endlichen Menge X ist ein Paar $(\mathcal{P}, X)$ mit den Eigenschaften

- $\mathcal{P} \subseteq \mathcal{P}(X)$, wobei $\mathcal{P}(X)$ die Potenzmenge von X bezeichnet.
- Aus $Y_1 \in \mathcal{P}$ und $Y_2 \subseteq Y_1$ folgt $Y_2 \in \mathcal{P}$.

Wir nennen die Mengen $Y \in \mathcal{P}$ unabhängig. Das Optimierungsproblem für $(\mathcal{P}, X)$ mit den Gewichten $w_x \geq 0$ für $x \in X$ ist die Bestimmung einer Menge $Y_{\max} \in \mathcal{P}$ mit maximalem Gewicht, d.h. es muss

$$\sum_{x \in Y_{\max}} w_x = \max\{\sum_{x \in Y} w_x \mid Y \in \mathcal{P}\}$$

gelten. Wir suchen also eine unabhängige Menge mit maximalem Gewicht.

Wir erhalten aus Kruskal's Algorithmus 2.7 für minimale Spannbäume den allgemeinen Greedy-Algorithmus 2.9, wenn wir eine unabhängige Menge mit maximalem Gewicht suchen.

Algorithmus 2.9 **Der Greedy-Algorithmus für monotone Mengensysteme**

(1) Die Eingabe besteht aus einem monotonen Teilmengensystem $(\mathcal{P}, X)$ über der Menge X und den Gewichten $w_x \geq 0$ für $x \in X$.

(2) Setze $Y = \emptyset$.

(3) WHILE $(X \neq \emptyset)$ DO

  (3a) Bestimme das Element $x \in X$ mit größtem Gewicht und setze $X = X \setminus \{x\}$.

  (3b) IF $(Y \cup \{x\} \in \mathcal{P})$ THEN $Y = Y \cup \{x\}$.

Bemerkung 2.6 Algorithmus 2.9 ist ein nicht-adaptiver Priority-Algorithmus, wenn wir Elemente und ihre Gewichte als Datenelemente auffassen: Elemente werden nach fallendem Gewicht geordnet.

Wann bestimmt Algorithmus 2.9 optimale Lösungen?

Definition 2.10 Ein monotones Teilmengensystem $(\mathcal{P}, X)$ heißt genau dann ein Matroid, wenn Algorithmus 2.9 eine optimale Lösung des zugehörige Optimierungsproblem für jede Gewichtung bestimmt.

Satz 2.11 gibt mehrere äquivalente Charakterisierungen eines Matroids an.

Satz 2.11 *Für ein monotones Teilmengensystem $(\mathcal{P}, X)$ sind äquivalent:*

(a) $(\mathcal{P}, X)$ ist ein Matroid.

(b) Die Ergänzungseigenschaft gilt: Für je zwei unabhängige Mengen Y_1, Y_2 mit $|Y_1| = |Y_2| - 1$ gibt es ein $x \in Y_2 \setminus Y_1$, so dass $Y_1 \cup \{x\}$ unabhängig ist.

(c) Die Maximalitätseigenschaft gilt: Alle nicht vergrößerbaren unabhängigen Teilmengen einer beliebigen Menge $Z \subseteq X$ besitzen dieselbe Größe.

Beweis: Wir führen einen Ringbeweis.
(a)→(b) Angenommen, es gibt zwei unabhängige Mengen Y_1 und Y_2 mit $p := |Y_1| = |Y_2| - 1$, aber $Y_1 \cup \{x\} \notin \mathcal{P}$ für jedes $x \in Y_2 \setminus Y_1$. Wir können den Greedy-Algorithmus 2.9 falsifizieren, wenn wir die Gewichte gemäß

$$w_x := \begin{cases} p+2 & \text{falls } x \in Y_1 \\ p+1 & \text{falls } x \in Y_2 \setminus Y_1 \\ 0 & \text{sonst.} \end{cases}$$

wählen. Algorithmus 2.9 wählt zuerst die p Elemente aus Y_1. Elemente aus Y_2 werden nicht mehr gewählt, da nach Voraussetzung $Y_1 \cup \{x\} \notin \mathcal{P}$ für jedes $x \in Y_2 \setminus Y_1$ gilt. Anschließend werden möglicherweise zusätzliche Elemente mit Gewicht 0 gewählt. Der Algorithmus erreicht das Gesamtgewicht

$$|Y_1| \cdot (p+2) = p \cdot (p+2) = p^2 + 2p,$$

während die unabhängige Menge Y_2 das größere Gewicht von mindestens

$$(p+1)^2 = p^2 + 2p + 1$$

besitzt — Widerspruch.
(b)→(c) Für $Z \subseteq X$ seien $Y_1, Y_2 \subseteq Z$ zwei größte, unabhängige Teilmengen mit $|Y_1| < |Y_2|$. Sei $Y_2^* \subseteq Y_2$ eine beliebige Teilmenge mit der Eigenschaft $|Y_1| = |Y_2^*| - 1$. Gemäß der vorausgesetzten Ergänzungseigenschaft können wir Y_1 um ein Element aus $Y_2^* \subseteq Y_2$ vergrößern und Y_1 ist nicht größtmöglich — Widerspruch.
(c)→(a) Sei eine beliebige Gewichtung $w_x \geq 0$ für $x \in X$ gegeben. Der Greedy-Algorithmus bestimmt eine unabhängige Menge Y. Sei Y^* eine beliebige andere, maximale unabhängige Menge. Da auch der Greedy-Algorithmus eine nicht-vergrößerbare unabhängige Menge konstruiert, folgt $|Y| = |Y^*|$ aus der Voraussetzung (c) mit $Z := X$. Wir nummerieren die Elemente von Y und Y^* gemäß absteigendem Gewicht und erhalten

$$Y = \{y_1, \ldots, y_m\}, \quad Y^* = \{y_1^*, y_2^*, \ldots, y_m^*\}.$$

Wir wollen zeigen, dass die Ausgabe Y von Algorithmus 2.9 eine optimale Lösung ist. Es genügt der Nachweis, dass $w_{y_i} \geq w_{y_i^*}$ für $i = 1, \ldots m$ gilt. Da Algorithmus 2.9 im

Schritt (3a) ein Element mit maximalem Gewicht wählt, gilt die Behauptung für $i = 1$. Im Induktionsschritt auf i wählen wir

$$Z^* := \{x \in X \mid w_x \geq w_{y_i^*}\}.$$

Wenn $w_{y_i} < w_{y_i^*}$, ist $\{y_1, \ldots, y_{i-1}\}$ eine nicht vergrößerbare unabhängige Teilmenge von Z^*, während die Teilmenge $\{y_1^*, y_2^*, \ldots, y_i^*\} \subseteq Z^*$ um ein Element größer ist. Wir haben einen Widerspruch zur Voraussetzung erhalten, dass alle nicht vergrößerbaren unabhängigen Teilmengen von $Z^* \subseteq X$ dieselbe Größe besitzen. □

Wir betrachten eine Reihe wichtiger Matroide beginnend mit der mächtigen Klasse der Matrix-Matroide. (Viele Begriffe dieses Kapitels sind durch Matrix-Matroide motiviert.)

Beispiel 2.2 **Matrix-Matroid**
Seien $v_1, \ldots v_m \in \mathbb{R}^n$ Vektoren und sei $X := \{v_1, \ldots v_m\}$. Gesucht ist eine Menge linear unabhängiger Vektoren aus X mit größtem Gewicht für eine gegebene Gewichtung $w_{v_i} \geq 0$. Sei $\mathcal{L}$ die Klasse aller linear unabhängigen Teilmengen von X. Das Paar $(\mathcal{L}, X)$ ist ein Matroid, das Matrix-Matroid genannt wird. Um dies zu belegen, wende Kriterium (c) von Satz 2.11 an.

Beispiel 2.3 **Das Graph-Matroid**
Sei $G := (V, E)$ ein ungerichteter, zusammenhängender Graph. Wir sagen, dass eine Menge $E' \subseteq E$ unabhängig ist, wenn die Kanten in E' einen Wald definieren.
Dann ist $(\mathcal{W}, E)$ ein Matroid, das man Graph-Matroid nennt. Um die Matroid-Eigenschaft nachzuweisen, wende Kriterium (c) von Satz 2.11 an.
Der Greedy-Algorithmus berechnet also Spannbäume mit maximalem Gewicht. Wenn wir die Kantengewichte w_e durch

$$w'_e := \max\{w_k \mid k \in E\} - w_e$$

ersetzen, wählt der Greedy-Algorithmus Kanten gemäß aufsteigender alter Gewichtung und wir haben somit Kruskalt's Algorithmus erhalten. Da ein Spannbaum von maximalem Gesamtgewicht in der neuen Gewichtung einem minimalen Spannbaum in der alten Gewichtung entspricht, haben wir Kruskals Algorithmus verifiziert.

Aufgabe 18

Es sei X eine endliche Menge, $A = \{S_1, \ldots, S_m\}$ eine Menge von Teilmengen von X und $T = \{x_1, \ldots, x_t\} \subseteq X$. Dann heißt T ein Transversal von A, wenn es paarweise verschiedene Zahlen $j(1), \ldots, j(t)$ gibt, so dass $x_i \in S_{j(i)}$ für $i = 1, \ldots, t$. Sei $\mathcal{P}_A$ die Menge aller Transversals von A.
Zeige, dass $M = (\mathcal{P}_A, X)$ ein Matroid ist.

Aufgabe 19

(a) Zeige, dass ein Transversal-Matroid auch als Matrix-Matroid darstellbar ist.
(b) Zeige, dass ein Graph-Matroid auch als Matrix-Matroid darstellbar ist.
HINWEIS: Wähle $\mathbb{Z}_2$ als Körper.

Aufgabe 20

(a) Sei X eine endliche Menge, und seien $X_1, \dots X_k$ eine Zerlegung von X in disjunkte Teilmengen. Zeige, dass das Teilmengensystem $(\mathcal{P}, X)$ mit

$$\mathcal{P} = \{A \subseteq X \mid |A \cap X_i| \leq 1 \text{ für } i = 1, \dots, k\}$$

ein Matroid ist.
(b) Sei $(\mathcal{P}, X)$ ein Matroid. Zeige, dass dann auch $(\mathcal{P}', X)$ mit

$$\mathcal{P}' = \{A' \subseteq X \mid X \setminus A' \text{ enthält eine nicht vergrößerbare Menge } A \in \mathcal{P}\}$$

ein Matroid ist.

Beispiel 2.4 **MATROID SCHEDULING**
Wir betrachten ein weiteres Scheduling Problem. Die Aufgaben $A_1, \dots, A_n$ sind auf einem Prozessor auszuführen, wobei die Ausführung jeweils in einem Schritt gelinge. Jeder Aufgabe A_i weisen wir eine Frist f_i und eine Strafe b_i zu, die anfällt, wenn die Aufgabe zum Zeitpunkt f_i nicht abgearbeitet ist. Die Aufgaben sind so auszuführen, dass die gesamte Strafe minimal ist.
Sei $X := \{A_1, \dots A_n\}$ und $\mathcal{P}$ bestehe aus allen Teilmengen von X, die ohne Fristüberschreitung ausführbar sind. Insbesondere gilt für jedes $Y \in \mathcal{P}$, dass es für jeden Zeitpunkt t in der Menge Y höchstens t Aufgaben A_i mit Frist $f_i \leq t$ gibt, denn sonst gäbe es eine Fristüberschreitung. Wir wollen mit Kriterium (b) von Satz 2.11, also der Ergänzungseigenschaft, zeigen, dass $(\mathcal{P}, X)$ ein Matroid ist.
Die Mengen $Y_1, Y_2 \in \mathcal{P}$ mit $|Y_1| = |Y_2| - 1$ seien beliebig gewählt. Für die Ergänzungseigenschaft müssen wir ein $A_i \in Y_2 \setminus Y_1$ mit $Y_1 \cup \{A_i\} \in \mathcal{P}$ angeben, also $A_i \in Y_2 \setminus Y_1$ so wählen, dass die Aufgaben aus Y_1 zusammen mit A_i ohne Fristüberschreitung ausführbar sind. Wähle $t \geq 0$ maximal mit

$$|\{A_j \in Y_2 \mid f_j \leq t\}| \leq |\{A_j \in Y_1 \mid f_j \leq t\}|.$$

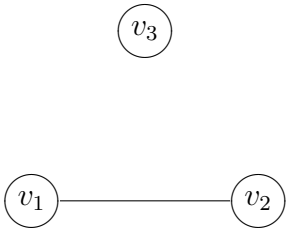
Ein solches maximales t existiert, da $|Y_2| > |Y_1|$. Es gibt also eine Aufgabe $A_i \in Y_2 \setminus Y_1$ mit $f_i = t + 1$. Weil wir t maximal gewählt haben, gilt zusätzlich, dass

$$\forall s \geq t + 1 : \quad |\{A_j \in Y_1 \mid f_j \leq s\}| < |\{A_j \in Y_2 \mid f_j \leq s\}| \leq s. \tag{2.1}$$

Die Aufgaben in $Y_1 \cup \{A_i\}$ sind fristgerecht ausführbar: Wir können die Aufgabe A_i in die fristgerechte Ausführung der Aufgaben in Y_1 zum Zeitpunkt $t + 1$ „einschieben“.
Der Greedy-Algorithmus findet, da $(\mathcal{P}, X)$ ein Matroid ist, eine Ausführung, die die Nicht-Strafe maximiert und daher die Strafe minimiert. Allerdings liefert der Greedy-Algorithmus nur eine Teilmenge $T \subseteq \{A_1, \dots A_n\}$, die ohne Fristüberschreitung ausführbar ist, nicht aber ein Scheduling dieser Teilmenge. Das Scheduling von T gelingt, indem wir die Aufgaben in T gemäß aufsteigender Frist ausführen.

Wir können auch CLIQUE als monotones Teilmengensystem formulieren: Für einen ungerichteten Graphen $G = (V, E)$ wählen wir $X := V$ und $\mathcal{P}$ als die Menge aller Cliquen, die

Teilgraphen von G sind. Für das zugehörigen Optimierungsproblem wählen wir 1 als Gewicht eines jeden Knotens. Mit Hilfe der Ergänzungseigenschaft (b) aus Satz 2.11 können wir beweisen, dass das monotone Teilmengensystem für CLIQUE kein Matroid ist. Dazu betrachten wir das folgende Beispiel:



Für $Y_1 := \{v_3\}$ und $Y_2 := \{v_1, v_2\}$ existiert kein Knoten v, so dass $\{v_3, v\}$ eine Clique ist.

Aufgabe 21
Wir betrachten das gewichtete MATCHING Problem für ungerichtete Graphen $G = (V, E)$ mit Kantengewicht $w(e) \geq 0$ für $e \in E$. Eine Teilmenge $M \subseteq E$ heißt ein Matching, falls es keine zwei Kanten in M mit einem gemeinsamen Endpunkt gibt. In MATCHING ist ein Matching M zu bestimmen, so dass $w_M := \sum_{e \in M} w(e)$ maximal unter allen Matchings ist.

(a) Betrachte das MATCHING Problem für *bipartite* Graphen $G = (V_1, V_2, E)$ als Mengensystem. Die Grundmenge X ist die Menge aller Kanten, das Teilmengensystem $\mathcal{M}$ besteht aus allen Matchings, also aus allen Mengen von Kanten ohne gemeinsamen Endpunkt.

Zeige, dass dieses Mengensystem $(\mathcal{M}, X)$ *kein* Matroid ist, aber als Durchschnitt zweier Matroide darstellbar ist.

(b) Sei $G = (V, E)$ ein ungerichteter Graph. Betrachte das Teilmengensystem $(\mathcal{P}, V)$ mit $Y \in \mathcal{P}$ genau dann, wenn es ein Matching M_Y in G gibt, so dass

$$Y \subseteq \{v \mid v \text{ ist ein Endpunkt einer Kante aus } M_Y\}.$$

Zeige, dass $(\mathcal{P}, V)$ ein Matroid ist. *Hinweis*: Ergänzungseigenschaft.

Also können wir das gewichtete Matching Problem, wobei diesmal aber Knoten und nicht Kanten gewichtet sind, mit dem Greedy-Algorithmus "lösen". Warum ist diese Lösung nicht befriedigend?

2.6 k-Matroide

Wir haben also alle monotonen Mengensysteme exakt charakterisieren können, für die der Greedy-Algorithmus 2.9 stets eine optimale Lösung findet. Können wir aber auch in dem Fall eine exakte Charakterisierung finden, wenn der Greedy-Algorithmus 2.9 nur k-approximativ für eine reelle Zahl $k \in \mathbb{R}$ ist?
Wir gehen genauso wie für Matroide vor und formulieren zuerst unsere Wunschvorstellung.

Definition 2.12 Ein monotones Teilmengensystem $(\mathcal{P}, X)$ heißt ein k-Matroid, wenn Algorithmus 2.9 eine k-approximative Lösung des zugehörige Optimierungsproblem für jede Gewichtung bestimmt.

Beachte, dass jedes 1-Matroid ein Matroid ist und umgekehrt. Glücklicherweise können wir k-Matroide, ähnlich wie in Satz 2.11, äquivalent charakterisieren.

Satz 2.13 *Für ein monotones Teilmengensystem $(\mathcal{P}, X)$ sind äquivalent:*

(a) $(\mathcal{P}, X)$ ist ein k-Matroid.

(b) Die k-Ergänzungseigenschaft gilt: Für je zwei unabhängige Mengen Y_1, Y_2 mit $|Y_2| > k \cdot |Y_1|$ gibt es ein $x \in Y_2 \setminus Y_1$, so dass $Y_1 \cup \{x\}$ unabhängig ist.

(c) Die k-Maximalitätseigenschaft gilt: Für jede beliebige Teilmenge $Z \subseteq X$ und je zwei nicht vergrößerbare unabhängige Teilmengen $Y_1, Y_2 \subseteq Z$ gilt $|Y_2| \leq k \cdot |Y_1|$.

Aufgabe 22
Beweise Satz 2.13.

Wir geben ein weiteres Kriterium an, das zwar k-Matroide nicht äquivalent beschreibt, aber impliziert, dass ein k-Matroid vorliegt. Der Nachweis dieses Kriteriums ist häufig recht einfach.

Definition 2.14 Ein monotones Teilmengensystem $(\mathcal{P}, X)$ heißt k-erweiterbar, wenn für je zwei unabhängige Mengen $Y_1 \subset Y_2$ und für jedes Element $x \notin Y_2$ gilt:

> Wenn $Y_1 \cup \{x\}$ unabhängig ist, dann gibt eine Teilmenge $T \subseteq Y_2 \setminus Y_1$ von höchstens k Elementen, so dass $(Y_2 \setminus T) \cup \{x\}$ unabhängig ist.

Lemma 2.15 *$(\mathcal{P}, X)$ sei ein monotones Teilmengensystem. Wenn $(\mathcal{P}, X)$ k-erweiterbar ist, dann ist $(\mathcal{P}, X)$ ein k-Matroid.*

Beweis: Wir können annehmen, dass $(\mathcal{P}, X)$ k-erweiterbar ist und müssen zeigen, dass $(\mathcal{P}, X)$ ein k-Matroid ist. Wir tun dies, indem wir die k-Ergänzungseigenschaft nachweisen. Wir müssen also beliebige unabhängige Mengen Y_1 und Y_2 mit $|Y_2| > k \cdot |Y_1|$ betrachten. Unser Ziel ist die Bestimmung eines Elements $x \in Y_2 \setminus Y_1$, so dass $Y_1 \cup \{x\}$ unabhängig ist. Wenn $Y_1 \subset Y_2$, dann sind wir offensichtlich bereits fertig, da $Y_1 \cup \{x\} \subseteq Y_2$ und Y_2 ist unabhängig. Also gilt $Y_1 \not\subset Y_2$.
Wir wählen ein beliebiges Element $x \in Y_1 \setminus Y_2$ und betrachten die unabhängigen Mengen $Z_1 := Y_1 \cap Y_2$ und $Z_2 := Y_2$. Wir wissen, dass $(\mathcal{P}, X)$ k-erweiterbar ist, und es gibt eine Teilmenge $T \subseteq Z_2 \setminus Z_1 = Y_2 \setminus Y_1$ mit höchstens k Elementen, so dass die Menge $Y_2' = (Y_2 \setminus T) \cup \{x\}$ unabhängig ist. Wenn Y_1 noch immer keine Teilmenge von Y_2 ist, dann wiederholen wir unser Vorgehen mit den Mengen Y_1 und Y_2'.
Beachte, dass wir stets nur Elemente aus Y_2 entfernen, die nicht zu Y_1 gehören. Nach höchstens $|Y_1 \setminus Y_2| \leq |Y_1|$ Schritten erhalten wir deshalb eine unabhängige Menge Y_2' mit $Y_1 \subseteq Y_2'$.
Es ist $|Y_2| > k \cdot |Y_1|$. In jedem Schritt entfernen wir höchstens k Elemente von Y_2 und fügen ein Element aus $Y_1 \setminus Y_2$ hinzu. Also gibt es ein Element $x \in Y_2 \setminus Y_1$, das auch in Y_2'

vorkommt und Y_1 ist eine echte Untermenge von Y_2'. Aber dann ist $Y_1 \cup \{x\}$ unabhängig, denn die Obermenge Y_2' ist unabhängig. □

Aufgabe 23
Wähle $X = Y \cup \{x\}$ als Grundmenge, wobei $|Y| = k + 2$ gelte und x kein Element von Y ist. Die einzigen nicht-erweiterbaren unabhängige Mengen sind die Menge Y sowie alle zwei-elementigen Mengen $\{x, y\}$ mit $y \in Y$.
Zeige, dass dieses Mengensystem ein k-Matroid ist, das nicht k-erweiterbar ist.

Wir haben bereits in einer Übungsaufgabe bemerkt, dass das Matching Problem für bipartite Graphen kein Matroid ist, wohl aber ein Durchschnitt von zwei Matroiden. Man kann allgemein zeigen, dass das Optimierungproblem für den Durchschnitt von k Matroiden effizient lösbar ist, wenn $k \leq 2$, aber $\mathcal{NP}$-vollständig für $k \geq 3$ sein kann. Wie gut können wir das Optimierungsproblem für den Durchschntt von k Matroiden approximieren?

Satz 2.16 *Ein Durchschnitt von k Matroiden ist k-erweiterbar und deshalb ein k-Matroid.*

Beweis: Wir überlegen uns zuerst, dass ein Matroid 1-erweiterbar ist. Betrachten wir also beliebige unabhängige Mengen $Y_1 \subset Y_2$ und ein Element $x \notin Y_2$, so dass $Y_1 \cup \{x\}$ unabhängig ist. Wir müssen ein Element $y \in Y_2 \setminus Y_1$ finden, so dass $Y_2 \setminus \{y\} \cup \{x\}$ unabhängig ist.
Wir wenden die Ergänzungseigenschaft des Matroids an, um die unabhängige Menge $Y_1 \cup \{x\}$ mit Elementen aus $Y_2 \setminus Y_1$ zu vergrößern und wiederholen dies, bis wir eine unabhängige Teilmenge Y_1' mit $Y_1 \cup \{x\} \subseteq Y_1' \subseteq Y_2 \cup \{x\}$ und $|Y_1'| = |Y_2|$ erhalten. Da $x \notin Y_2$, besteht $Y_2 \setminus Y_1'$ aus genau einem Element y. Aber dann ist $Y_2 \setminus \{y\} \cup \{x\} = Y_1'$ unabhängig und das war zu zeigen.
Ein Durchschnitt von k Matroiden ist also ein Durchschnitt von k vielen 1-erweiterbaren monotonen Mengensystemen. Wir müssen also nur zeigen, dass der Durchschnitt von k vielen 1-erweiterbaren monotonen Mengensystemen k-erweiterbar ist: Für Teilmengen $Y_1 \subset Y_2$ und $x \notin Y_2$ sei also die Menge $Y_1 \cup \{x\}$ unabhängig. Dann gibt es im i.ten 1-erweiterbaren monotonen Mengensystem ein Element $y_i \in Y_2 \setminus Y_1$, so dass $Y_2 \setminus \{y_i\} \cup \{x\}$ im i.ten System unabhängig ist. Dann ist aber auch $Y_2 \setminus \{y_1, \ldots, y_k\} \cup \{x\}$ eine unabhängige Menge des Durchschnitts. □

Der Greedy Algorithmus 2.9 ist also k-approximativ für den Durchschnitt von k Matroiden. Insbesondere erhalten wir also ein 2-approximatives Matching für bipartite Graphen. Tatsächlich können wir noch mehr:

Beispiel 2.5 f-MATCHING
$G = (V, E)$ sei ein ungerichteter Graph, dessen Kanten nicht-negative Gewichte erhalten. Für eine Funktion $f : V \to \mathbb{N}$ definieren wir f-Matchings: Ein f-Matching ist eine Kantenmenge M, so dass die Anzahl mit v inzidenter Kanten durch $f(v)$ für jeden Knoten v beschränkt ist. Im Optimierungsproblem f-MATCHING ist ein schwerstes f-Matching zu bestimmen.
Unser Mengensystem besteht also aus der Grundmenge E und allen Teilmengen von E, die f-Matchings entsprechen. Wir zeigen, dass das Mengensystem 2-erweiterbar ist: Der Greedy Algorithmus ist also 2-approximativ.

Für den Nachweis betrachten wir zwei f-Matchings $Y_1 \cup \{e\}$ und Y_2 mit $Y_1 \subset Y_2$, wobei die Kante $e = \{u, v\}$ nicht in Y_2 liege. Wenn $Y_2 \cup \{e\}$ ein f-Matching ist, dann sind wir fertig. Wenn nicht, dann besitzt Y_2 bereits $f(u)$ Kanten, die inzident mit u, oder $f(v)$ Kanten, die inzident mit v sind. Wir entfernen eine mit u und eine mit v inzidente Kante aus Y_2 und können jetzt e gefahrlos einsetzen!

Beispiel 2.6 Ein vollständiger gerichteter Graph mit nicht-negativen Kantengewichten ist gegeben. In MAX-TSP ist eine Rundreise mit maximaler Länge zu bestimmen. (Eine Rundreise besucht jeden Knoten genau einmal.)
Die Grundmenge besteht diesmal aus allen gerichteten Kanten. Eine Teilmenge ist unabhängig, wenn ihre Kanten Knoten-disjunkte Pfade oder eine Rundreise bilden. Wir zeigen, dass dieses Mengensystem 3-erweiterbar ist.
Seien also $Y_1 \cup \{e\}$ und Y_2 zwei unabhängige Mengen mit $Y_1 \subset Y_2$, wobei die Kante $e = (u, v)$ nicht in der Menge Y_2 liege. Wir entfernen zuerst, falls vorhanden, die u verlassende und die in in v eintreffende Kante aus Y_2.
Wenn wir jetzt e zu Y_2 hinzufügen, dann hat jeder Knoten höchstens eine eintreffende und eine ausgehende Kante. Wenn also die Hinzunahme von e zum jetzigen Zeitpunkt unzulässig ist, dann muss es genau einen Kreis geben, der keine Rundreise ist, aber e benutzt. Aber dann hat dieser Kreis mindestens eine Kante e', die nicht zur Menge Y_1 gehört. Wir entfernen die Kante e', der Kreis ist gebrochen, und wir haben eine unabhängige Menge erhalten.
Der Greedy-Algorithmus berechnet also eine 3-approximative Lösung für MAX-TSP. Der beste von effizienten Algorithmen erreichbare Approximationsfaktor ist 8/5.

Aufgabe 24
Wir betrachten das folgende Scheduling Problem: Aufgaben $A_1, \ldots, A_n$ können auf einer einzigen Maschine ausgeführt werden. Aufgabe A_i darf nach ihrer „Release Time“ r_i ausgeführt werden und muss vor ihrer Frist f_i erledigt werden. Wird Aufgabe A_i wie verlangt ausgeführt, dann wirft sie den Profit p_i ab. Jede Aufgabe benötigt dieselbe Anzahl von Zeitschritten für ihre Erledigung.
Zeige, dass das zugrunde liegende Mengensystem 2-erweiterbar ist. Ist genau ein Zeitschritt für die Erledigung notwendig, dann ist das Mengensystem sogar 1-erweiterbar.

2.7 Sequenzierung

Im Problem der Sequenzierung ist die Reihenfolge der Basenpaare für einen Strang eines DNA-Moleküls zu bestimmen. In der **direkten Sequenzierung** wird die entsprechende DNA-Sequenz sukzessive in fast positions-disjunkte kleine Fragmente zerlegt und die kleinen Fragmente werden sequenziert. Da dieser Prozess iterativ kurze Präfixe von ungefähr 500 Basenpaaren aus der Sequenz herausschneidet, ist der Prozess sehr zeitaufwändig und nicht für die Sequenzierung langer Sequenzen geeignet.
Andererseits lässt sich die Sequenz parallel, an fast zufällig verteilten Positionen zerschneiden, aber leider geht bei diesem Prozess die Reihenfolge der Fragmente verloren. In der **Shotgun-Sequenzierung** werden deshalb zuerst genügend viele Kopien einer zu sequenzierenden DNA-Sequenz erstellt. Diese Kopien werden dann parallel, an nicht korrelierten

Postionen in kleine Fragmente zerschnitten. Die Ordnung der Fragmente ist dann algorithmisch zu bestimmen.
Das Zusammensetzen des „Puzzles“ aus den Fragmenten bezeichnet man als das Problem der **Fragment-Assembly**.

2.7.1 Shotgun-Sequenzierung und Fragment Assembly

Die Sequenzierung einer unbekannten, nicht zu langen DNA-Sequenz besteht aus den folgenden Schritten:

(1) Führe das Shotgun-Verfahren durch.

(2) Bestimme die **Reads** für jedes Fragment, also die Folge der ungefähr 500 Basenpaare an einem Ende des Fragments.

(3) Führe Fragment-Assembly durch, also rekonstruiere die DNA-Sequenz aus den Reads.

Wir konzentrieren uns auf Schritt (3). Natürlich gibt es viele „Superstrings“, also Strings, die jeden Read als Teilstring besitzen. Es liegt aber nahe, nach einem kürzesten Superstring zu suchen, der alle Reads als Teilstrings besitzt.

Aufgabe 25
Die Strings $u_1, \ldots, u_k$ seien vorgegeben, wobei kein String Präfix eines anderen Strings sei. Zeige oder widerlege:

Jeder String u_i kommt genau einmal als Teilstring eines kürzesten Superstrings vor.

Aufgabe 26
In der Sequenzierung durch Hybridisierung ist ein unbekanntes DNA-Fragment S zu sequenzieren. Sei $V \subseteq \Sigma^k$ die Menge *aller* Strings der Länge k, die als Teilstring in S vorkommen. Zusätzlich müssen wir annehmen, dass jeder String in V *genau einmal* in S vorkommt. Rekonstruiere S in Zeit $O(|S|)$.
Hinweis: Formuliere das Rekonstruktionsproblem als ein graph-theoretisches Problem. Hilft das Problem des Auffindens eines Euler-Weges? (Ein Euler-Weg durchläuft jede Kante genau einmal. Wenn ein Euler-Weg existiert, dann kann ein Euler-Weg in Linearzeit bestimmt werden.)

Aufgabe 27
In einer anderen Variante der Hybridisierungsmethode versucht man ein DNA-Fragment S zu rekonstruieren, indem man das Fragment auf Vorkommen von (sorgfältig ausgewählten) Strings (Proben) in S untersucht. Das Ziel dabei ist, so wenig Proben wie möglich zu benutzen. Man kann das Problem wie folgt formulieren.
Sei S ein String (das “Geheimnis des Lebens”, eine DNA-Sequenz, etc.) der Länge $n = |S|$ über einem Alphabet Σ mit $|\Sigma| = \alpha$. Wir möchten den String rekonstruieren, wobei die Länge n des Strings uns bekannt sei. Es gebe ein Orakel, das für jede den String S betreffende Frage eine Antwort JA oder NEIN gibt. Sei $T(S)$ die minimale ausreichende Anzahl der Fragen an das Orakel, um den String S zu rekonstruieren.

(a) Zeige, dass unabhängig davon, welche Fragen wir stellen werden, es einen String S mit

$$T(S) \geq n \cdot \log_2 \alpha$$

gibt.

(b) Nehmen wir jetzt an, dass wir nur fragen können, ob ein (beliebiger) String u als *Teilstring* in S vorkommt. Das Orakel gibt uns die Antwort, sagt aber nicht wo und wie oft der String u in S vorkommt. Es ist bekannt, dass es Strings S mit $T(S) \geq \alpha n/4 - O(n)$ gibt. Zeige, dass

$$T(S) \leq \alpha \cdot (n+1).$$

Bemerkung: Mann kann auch $T(S) \leq (\alpha - 1) \cdot n + O(\log n + \alpha)$ zeigen.

Leider liegen die Daten nicht in idealer Form vor. Zu den problematischen Fällen gehören unter anderem:

(1) Fehler beim Sequenzieren der Fragmente, so dass einige angegebene Basen falsch sind.

(2) Fehler beim Kopieren: Zum Beispiel können sich Fragmente einer Wirt-DNA mit den Fragmenten der DNA-Sequenz vermischen.

(3) Fragmente von beiden Strängen des Moleküls tauchen auf.

(4) Überdeckungslücken: Einige Regionen der DNA-Sequenz werden möglicherweise nicht von Fragmenten überdeckt.

Neben diesen Problemquellen muss man sich auch um das Problem langer *Repeats*, also um wiederholt in der DNA-Sequenz auftretende Teilstrings, kümmern. Beispielsweise gibt es ein 300 Basenpaare langes Segment, das mit 5-15 prozentiger Variation mehr als 1 Million Mal im menschlichen Genom vorkommt. Algorithmen für das Zusammensetzen der Fragmente versuchen natürlich, möglichst kurze Superstrings vorzuschlagen und tendieren deshalb dazu, lange Repeats zu unterdrücken. Repeats sind ein sehr kritisches Problem und sind zum Beispiel dafür verantwortlich, dass oft Physical Mapping zur Vorbereitung der Shotgun-Sequenzierung benötigt wird.

2.7.2 Sequenzierung der Drosophila-DNA

Die Drosophila-DNA wurde von Celera Genomics in Zusammenarbeit mit dem öffentlichen Berkeley Drosophila Genom Project sequenziert.
Die Drosophila-DNA besteht aus ungefähr 137 Millionen Basenpaaren, die durch drei Millionen Fragmente fast 14-fach überdeckt wurden. Die Fragmente sind entweder kurz (2,000 Basenpaare) oder lang (10,000 Basenpaare) und Reads aus 500 Basenpaaren werden für die beiden Enden eines Fragments erstellt. Da beide Enden eines Fragments sequenziert werden, erhält man Paare von Reads, von denen man hochwahrscheinlich weiß, dass sie um ungefähr 1,000 oder um ungefähr 9,000 Basen auseinanderliegen. Wir nennen solche Paare *Mates-Paare*. Aufgrund der Distanz zwischen Mates werden Repeats im Allgemeinen nur ein Element des Paares besitzen und Mates-Paare helfen bei der Entdeckung von Repeats. Neben den Mates-Paaren werden auch STS-Karten eingesetzt: Da die ungefähre Distanz zwischen aufeinanderfolgenden STS-Markern bekannt ist, dienen die Karten der Orientierung und der Überprüfung während des Prozesses des Fragment-Assembly.
Das Zusammensetzen des „Puzzle“ beginnt mit „Zügen“ von geringstem Risiko und nimmt dann langsam an Risiko zu. Zuerst werden in einem äußerst rechenaufwändigen Verfahren

je zwei Reads verglichen. Ein Paar von Reads heiße signifikant, wenn die Reads einen Teilstring t der Länge mindestens 40 besitzen, so dass t ein Suffix des ersten und ein Präfix des zweiten Reads ist. Unter den signifikanten Paaren befinden sich Paare mit einer echten Überlappung und Paare, die sich aufgrund eines Repeats überlappen; eine „unechte“ Überlappung kann nicht ausgeschlossen werden, ist jedoch aufgrund der Länge des gemeinsamen Teilstrings sehr unwahrscheinlich.
Im ersten Zug werden mit Hilfe der signifikanten Paare Gruppen von Reads gebildet, so dass die Reads einer Gruppe

- eine gemeinsame Sequenz besitzen,
- die nicht von anderen (ebenfalls überlappenden) Reads in Frage gestellt wird.

Es stellt sich heraus, dass nur wenige Gruppen untereinander überlappen. Auch wird behauptet, dass 99% der Gruppen bereits richtig zusammengesetzt sind. Nach der Gruppenbildung beginnt die Fehlersuche mit Gruppen, die an einigen Positionen zu tief sind: Hier besteht Anlass, einen Repeat zu vermuten und die entsprechenden Gruppen werden disqualifiziert.
Die verbleibenden Gruppen sind jetzt mit Hilfe der Mates-Paare anzuordnen. Da die meisten Repeats nicht mehr als 7,000 Basenpaare umfassen, helfen Mates-Paare Repeats zu überbrücken. Wenn zwei Gruppen mindestens zwei Mates-Paare besitzen, dann werden die beteiligten Gruppen räumlich nahe beieinander liegen. Nachdem dieser Zug abgeschlossen ist, geht man zum riskanteren Zug über, diejenigen Gruppen nebeneinander zu legen, die nur ein Mates-Paar besitzen. Nach diesen beiden Zügen ist (hoffentlich) die genaue Permutation der Gruppen bekannt und die Sequenz ist bis auf Lücken bestimmt, wobei eine Lücke aufgrund fehlender Daten oder aufgrund eines Repeats entsteht.
Für den Fall eines Repeats suchen wir die disqualifizierte Gruppen auf. Zuerst werden disqualifizierte Gruppen in die Lücken geworfen, wobei mindestens zwei Mates-Paare auf jeder Seite die disqualifizierte Gruppe mit der jeweiligen Randgruppe verbinden. Auf diesen konservativen Zug folgt der riskantere Zug, eine disqualifizierte Gruppe auch dann in eine Lücke zu werfen, wenn an einer Seite nur ein Mates-Paar absichert. Schließlich ordnet man Repeats ohne Absicherung ein, indem man die größte Überlappung zu einer Gruppe wählt. (Warum greift man disqualifizierte Gruppen trotz der Gefahr der Repeats wieder auf? Repeats gefährden die genaue Lokalisierung, aber die Lokalisierung ist durch die Absicherung mit Mates-Paaren zumindest für die beiden ersten Zugtypen bereits gegeben. Natürlich besteht weiterhin die Gefahr, dass Repeats die Sequenz verfälschen und die Sequenzierungsmethode muss, nach Produktion der Sequenz, auf diese Schwachstelle der Sequenz hinweisen.)

2.7.3 SHORTEST COMMON SUPERSTRING

In SHORTEST COMMON SUPERSTRING ist eine Menge U von Strings gegeben. Wir suchen einen String S minimaler Länge, der alle Strings in U als Teilstrings enthält.
Wir führen zuerst das zentrale Konzepts des Overlaps zwischen zwei Strings ein.

Definition 2.17 Für Strings u und v sei $\text{overlap}(u, v)$ die Länge des längsten Suffix von u, der auch Präfix von v ist. Weiterhin ist $\text{länge}(u, v) = |u| - \text{overlap}(u, v)$ die Länge des nicht überdeckten Präfixes von u, den wir auch mit $\text{präfix}(u, v)$ bezeichnen.

Offensichtlich können wir annehmen, dass kein String in U ein Teilstring eines anderen Strings ist. Wenn nämlich zum Beispiel $u \in U$ ein Teilstring von $v \in U$ ist, dann kann u ausgelassen werden, da u von jedem Superstring der restlichen Strings überdeckt wird.

Definition 2.18 Eine Menge U von k Strings und eine Bijektion $\pi : \{1, \ldots, k\} \to U$ seien gegeben. Die Bijektion π definiert den Superstring

$$S(\pi) = \text{präfix}(\pi(1), \pi(2)) \cdot \text{präfix}(\pi(2), \pi(3)) \cdots \text{präfix}(\pi(k-1), \pi(k)) \cdot \pi(k).$$

Wir nennen $S(\pi)$ den durch π induzierten Superstring von U.

Lemma 2.19 *U sei eine Menge von Strings und $\pi : \{1, \ldots, k\} \to U$ sei eine Bijektion.*

(a) *Der String $S(\pi)$ ist ein Superstring der Reads in U.*

(b) *Sei $L = \sum_{u \in U} |u|$. Dann ist*

$$|S(\pi)| = L - \sum_{i=1}^{k-1} \mathit{overlap}(\pi(i), \pi(i+1)).$$

(c) *Wenn S ein kürzester Superstring ist, dann gibt es eine Permutation π mit $S = S(\pi)$.*

Beweis (a) sei dem Leser überlassen. **(b)** folgt aus der Beziehung $\text{länge}(\pi(i), \pi(i+1)) = |\pi(i)| - \text{overlap}(\pi(i), \pi(i+1))$, denn

$$\begin{aligned} |S(\pi)| &= |\pi(k)| + \sum_{i=1}^{k-1} \text{länge}(\pi(i), \pi(i+1)) \\ &= |\pi(k)| + \sum_{i=1}^{k-1} [\ |\pi(i)| - \text{overlap}(\pi(i), \pi(i+1))\ ] \\ &= L - \sum_{i=1}^{k-1} \text{overlap}(\pi(i), \pi(i+1)). \end{aligned}$$

Für **(c)** betrachte einen kürzesten Superstring S. Bestimme für jeden String $u \in U$ ein, vom Anfang von S aus gesehen, erstes Vorkommen in S. Da kein String ein Teilstring eines anderen Strings ist, beginnen je zwei Strings in verschiedenen Positionen. Also definieren die Anfangspositionen der einzelnen Strings eine Ordnung auf den Strings in U und damit eine Bijektion $\pi : \{1, \ldots, k\} \to U$.
Wir behaupten, dass $S = S(\pi)$. Hierzu brauchen wir nur zu beobachten, dass der kürzeste Superstring S die Strings $\pi(i+1)$ weitest möglich über ihre Vorgänger $\pi(i)$ „schieben“

wird, da sonst S verkürzt werden könnte. Die Behauptung folgt, da $S(\pi)$ diese maximale Verschiebung stets durchführt. □

Also genügt es, eine Permutation π mit möglichst großem Overlap

$$\sum_{i=1}^{k-1} \text{overlap}(\pi(i), \pi(i+1))$$

zu bestimmen.

Definition 2.20 Wir fassen einen String v als zyklischen String auf, indem wir die ersten und letzten Buchstaben von v aneinander kleben. Wir sagen, dass u die Periode v hat, falls wir u vollständig um v wickeln können. Trifft dies zu, dann sagen wir auch, dass u die Periodenlänge $|v|$ hat. v ist die minimale Periode von u, falls u die Periode v hat und falls $|v|$ die minimale Periodenlänge von u ist.

Die folgende zentrale Eigenschaft besagt, dass der Overlap zweier Strings durch die Summe ihrer Periodenlängen beschränkt ist.

Lemma 2.21 Overlap-Lemma
Seien u_1 und u_2 zwei beliebige Strings mit minimalen Periodenlängen p_1 und p_2. Wenn u_1 und u_2 verschiedene minimale Perioden besitzen, dann ist

$$\textit{overlap}(u_1, u_2) < p_1 + p_2.$$

Für den Beweis benötigen wir eine Beobachtung über periodische Strings.

Lemma 2.22 Das ggT-Lemma
Der String v habe Perioden der Längen p und q, wobei $p + q \leq |v|$ gelte. Dann besitzt v auch eine Periode der Länge $ggT(p, q)$.

Beweis: O.B.d.A. gelte $q \leq p$. Wir werden zeigen, dass v auch eine Periode der Länge $p - q$ besitzt. Beachte, dass Euklids Algorithmus den größten gemeinsamen Teiler von p und q rekursiv berechnet unter Ausnutzung von $\text{ggT}(p, q) = \text{ggT}(q, p - q)$. Wenn wir also die Periodenlänge $p-q$ nachweisen können und unsere Argumentation iterieren, dann simulieren wir Euklids Algorithmus und erhalten die Behauptung.
Beachte zuerst, dass für $i \leq |v| - p$ stets $v_i = v_{i+p}$ gilt, denn v hat Periode p. Andererseits hat v auch Periode q und deshalb ist $v_{i+p} = v_{i+p-q}$ für $i \leq |v|-p$ und insgesamt $v_i = v_{i+p-q}$ für $i \leq |v| - p$.
Mit analogem Argument zeigt man $v_i = v_{i-q} = v_{i-q+p}$ für $q < i \leq |v| - p + q$. Einer der Fälle $q < i \leq |v| - p + q$ und $i \leq |v| - p$ trifft aber für jedes i mit $i \leq |v| - p + q$ zu und v hat die Periode $p - q$. □

Beweis von Lemma 2.21: Wir führen den Beweis durch Widerspruch und nehmen an, dass $\text{overlap}(u_1, u_2) \geq p_1 + p_2$, wobei v_1 und v_2 die entsprechenden Perioden von u_1 und

u_2 seien. Sei u der Suffix-Präfix Match von u_1 und u_2. Dann ist $\text{overlap}(u_1, u_2) = |u| \geq p_1 + p_2 = |v_1| + |v_2|$.
Wir fassen v_1 und v_2 als zyklische Strings auf, und beachten, dass wir u vollständig um v_1 wie auch vollständig um v_2 wickeln können.

Fall 1: $|v_1| = |v_2|$.
Nach geeigneten zyklischen Shifts von v_1 und v_2 mit Resultaten v_1^* und v_2^* stimmen v_1^* und v_2^* mit den ersten $|v_1|$ Buchstaben von u überein. Also ist v_1 ein zyklischer Shift von v_2. Damit besitzen aber u_1 und u_2 im Gegensatz zur Annahme identische minimale Perioden.

Fall 2: $|v_1| \neq |v_2|$.
O.B.d.A. sei $|v_1| > |v_2|$. Nach Annahme ist $u \geq |v_1| + |v_2|$, und wir können das ggT-Lemma anwenden. Der String u hat also eine Periode v der Länge $\text{ggT}(|v_1|, |v_2|) < \max\{|v_1|, |v_2|\} = |v_1|$. Wir können u vollständig um v_1 wickeln und deshalb können wir auch v_1 um v wickeln. Aber $|v|$ ist ein Teiler von $|v_1|$: Der String u_1 hat die Periode $v < |v_1|$, und das ist unmöglich. □

Unser Approximationsalgorithmus benutzt den Begriff der Zyklus-Überdeckung.

Definition 2.23

(a) Für einen String v sei v^∞ der unendlich oft mit sich selbst konkatenierte String v.

(b) Ein String v ist eine zyklische Überdeckung eines Strings u, wenn u ein Teilstring des Strings v^∞ ist.

(c) Seien U und V Mengen von Strings. Wir sagen, dass V eine zyklische Überdeckung von U ist, wenn jedes $u \in U$ von mindestens einem String $v \in V$ zyklisch überdeckt wird. Weiterhin ist $\sum_{v \in V} |v|$ die Länge der Zyklus-Überdeckung V

Im Problem ZYKLUS-ÜBERDECKUNG ist ebenfalls eine Menge U von Strings gegeben, und eine zyklische Überdeckung der Menge U minimaler Länge ist zu bestimmen. Erstaunlicherweise werden wir später mit Algorithmus 2.26 eine minimale Zyklus-Überdeckung effizient berechnen.
Offensichtlich ist ein kürzester Superstring S auch eine zyklische Überdeckung von U und die Länge von S ist deshalb mindestens so groß wie die Länge einer minimalen zyklischen Überdeckung. Wir erhalten also eine gute *untere Schranke* durch die Länge einer minimalen zyklischen Überdeckung. Wir zeigen jetzt umgekehrt, wie man aus einen nicht zu langen Superstring aus einer minimalen zyklischen Überdeckung erhält.

Algorithmus 2.24 **Berechnung kurzer Superstrings**

(1) Die Eingabe besteht aus einer Menge U von Strings, wobei kein String aus U ein Teilstring eines anderen Strings aus U ist.

(2) Bestimme eine minimale Zyklus-Überdeckung V von U.

(3) Betrachte alle Zyklen $v \in V$ nacheinander.

Sei $U_v \subseteq U$ die Menge der von v zyklisch überdeckten Strings aus U. Das erste Vorkommen der Strings aus U_v in v^∞ definiert eine Ordnung auf den Strings aus U_v und damit eine Bijektion π_v. Berechne den durch π_v induzierten Superstring $S_v(\pi_v)$ von U_v.

Kommentar: Wir erhalten $S_v(\pi_v)$, wenn wir den Zyklus v „aufbrechen", wenn wir also die Strings in U_v gemäß ihrer Ordnung maximal übereinander schieben, aber den letzten String nicht mehr über den ersten schieben.

(4) Gib die Konkatenation $\Pi_{v \in V} S_v(\pi_v)$ der Superstrings als Resultat aus.

Kommentar: Angenommen, es gibt für jeden Zyklus $v \in V$ mindestens einen String in U_v, der höchstens so lang wie v ist. Da kein String in U Teilstring eines anderen Strings ist, sind somit alle Strings höchstens so lang wie ihre Zyklen. Damit führt das Aufbrechen der Zyklen höchstens zu einer Längenverdopplung und wir erhalten einen 2-Approximationsalgorithmus, da die Länge der minimalen Zyklus-Überdeckung die Länge eines kürzesten Superstrings nicht übersteigt.

Fazit: Das Problem sind hochgradig zyklische Strings in U, eine im Anwendungsfall durch Repeats tatsächlich gegebene Gefahr.

Lemma 2.25 *Algorithmus 2.24 berechnet einen 4-approximativen Superstring.*

Beweis: Für $v \in V$ sei $u_v \in U$ der gemäß π_v letzte von v zyklisch überdeckte String aus U. Wir erhalten, dass $S_v(\pi_v)$, der Superstring des Zyklus v, die Länge höchstens $|v| + |u_v|$ hat und Algorithmus 2.24 berechnet damit einen Superstring der Länge L mit

$$L \leq \sum_{v \in V} (|v| + |u_v|).$$

Wir betrachten einen kürzesten Superstring S^* für die jeweils letzten Strings und nehmen O.B.d.A. an, dass $u_1, \ldots, u_r$ diese letzten Strings sind und in dieser Reihenfolge in S^* vorkommen. Sei L_{opt} die Länge des kürzesten Superstrings für *alle* Strings in U, und wir erhalten wegen $L_{\text{opt}} \geq |S^*|$, dass

$$\begin{aligned} L_{\text{opt}} &\geq \sum_{i=1}^{r} |u_i| - \sum_{i=1}^{r-1} \text{overlap}(u_i, u_{i+1}) \\ &> \sum_{i=1}^{r} |u_i| - \sum_{i=1}^{r-1} (|v_i| + |v_{i+1}|) \end{aligned}$$

In der ersten Ungleichung haben wir Lemma 2.19(b) angewandt, während die zweite Ungleichung aus dem Overlap Lemma folgt. Also ist

$$\sum_{i=1}^{r} |u_i| \leq L_{\text{opt}} + 2 \cdot \sum_{v \in V} |v|,$$

und Algorithmus 2.24 berechnet einen Superstring der Länge

$$L \leq \sum_{v \in V} |v| + \sum_{i=1}^{r} |u_i| \leq L_{\text{opt}} + 3 \cdot \sum_{v \in V} |v| \leq 4 \cdot L_{\text{opt}}.$$

Die Behauptung folgt. □

2.7.4 ZYKLUS-ÜBERDECKUNG

Um die Beschreibung von Algorithmus 2.24 abzuschließen, müssen wir eine minimale Zyklus-Überdeckung für eine Menge U von Strings konstruieren. Dazu betrachten wir einen gerichteten Graphen $G = (U, E)$ mit $E = U \times U$ und weisen einer Kante (u_1, u_2) das Gewicht overlap(u_1, u_2) zu.
Wir betrachten jetzt einen Kreis $(u_1, \ldots, u_k, u_1)$ in G. Dieser Kreis erzeugt einen zyklischen String der Länge

$$\sum_{i=1}^{k} |u_i| - \text{overlap}(u_k, u_1) - \sum_{i=1}^{k-1} \text{overlap}(u_i, u_{i+1})$$

> **Fazit:** Eine disjunkte Zerlegung der Knotenmenge von G in Kreise von maximalem Kantengewicht liefert eine minimale Zyklus-Überdeckung, da dann die Summe der Zyklenlängen kleinstmöglich ist.

Wir haben ZYKLUS-ÜBERDECKUNG also auf das folgende Graphenproblem reduziert: Für einen gerichteten Graph $G = (U, E)$ mit $E = U \times U$ und Gewichten w_e für die Kanten $e \in E$ bestimme eine Zerlegung von U in disjunkte Kreise mit maximalem Gewicht.
Beachte, dass ein perfektes Matching von B einer Kreis-Zerlegung von G entspricht! Wir werden später sehen, dass maximale Matchings für bipartite Graphen mit Hilfe der linearen Programmierung effizient gefunden werden können, und damit haben wir die gewünschte effiziente Lösung für die maximale Kreis-Zerlegung erhalten. In unserem speziellen Fall gelingt aber eine noch einfachere Lösung.

Algorithmus 2.26 **Berechnung einer minimalen Zyklus-Überdeckung**

(1) Die Eingabe besteht aus einer Menge U von Strings.

(2) Bestimme den gerichteten Graph $G = (U, E)$ mit $E = U \times U$ und die Kantengewichtung gewicht : $E \to \mathbb{R}$ mit gewicht$(u_1, u_2) =$ overlap(u_1, u_2).

(3) Sortiere die Kanten in E nach ihrem Gewicht. Setze $E^* = \emptyset$.

(4) Wiederhole, solange $E \neq \emptyset$:

- **(4a)** Bestimme die Kante $e = (u_1, u_2) \in E$ mit größtem Gewicht. Füge e zu E^* hinzu.
- **(4b)** Entferne aus E alle den Knoten u_1 verlassenden und alle den Knoten u_2 erreichenden Kanten.

(5) Berechne eine Zyklus-Überdeckung aus den durch E^* definierten Kreisen.

Bemerkung 2.7 Wir wählen die Tripel (erster String, zweiter String, Overlap) als Datenelemente. Algorithmus 2.26 ist dann ein nicht-adaptiver Priority-Algorithmus, der Datenelemente nach absteigendem Overlap anordnet.

Dieser Greedy-Algorithmus funktioniert nur in unserer speziellen Situation, da die Monge-Ungleichung für Strings gilt:

Aufgabe 28
Seien u_1, u_2, v_1, v_2 vier beliebige Strings, wobei

$$\text{overlap}(u_1, v_1) = \max\{\,\text{overlap}(u_i, v_j) \mid 1 \leq i, j \leq 2\}$$

gelte. Zeige die *Monge-Ungleichung* (Gaspard Monge 1746-1818):

$$\text{overlap}(u_1, v_1) + \text{overlap}(u_2, v_2) \geq \text{overlap}(u_1, v_2) + \text{overlap}(u_2, v_1).$$

Satz 2.27 *Algorithmus 2.26 berechnet eine minimale Zyklus-Überdeckung. Insbesondere ist Algorithmus 2.24 4-approximativ.*

Aufgabe 29
(a) Beweise Satz 2.27 mit Hilfe der Monge-Ungleichung.
(b) Angenommen die Kantengewichte $w(u, v)$ für die Kante (u, v) besitzen nur die Eigenschaft $w(u, v) \geq 0$. Zeige, dass Algorithmus 2.26 in diesem Fall eine Zyklus-Überdeckung berechnet, die bis auf den Faktor 2 optimal ist.

Noch naheliegender als Algorithmus 2.24 ist der folgende Greedy-Algorithmus.

Algorithmus 2.28 **Der Greedy-Superstring Algorithmus**

(1) Gegeben ist eine Menge U von Strings, wobei keine zwei Strings in U Teilstrings voneinander sind.

(2) Wiederhole, solange bis $|U| = 1$.

(2a) Bestimme zwei Strings $u \neq v \in U$ mit maximalem $\text{overlap}(u, v)$.

(2b) Ersetze U durch $U \setminus \{u, v\} \cup \{w\}$, wobei $w = \text{präfix}(u, v) \cdot v$.

(3) Gib den verbleibenden String in U als Superstring aus.

Offenes Problem 1
Man kann zeigen, dass Algorithmus 2.28 4-approximative Superstrings berechnet, vermutet aber, dass der Algorithmus sogar 2-approximativ ist.

Aufgabe 30
Sei $S_{\text{opt}}(U)$ die Länge eines kürzesten Superstrings und $S_{\text{greedy}}(U)$ die Länge des vom Greedy-Algorithmus berechneten Superstrings für U.

(a) Konstruiere eine Menge U von drei Strings, so dass keine zwei Strings in U Teilstrings voneinander sind und der Approximationsfaktor

$$\alpha(U) = \frac{S_{\text{greedy}}(U)}{S_{\text{opt}}(U)}$$

größtmöglich ist.

(b) Zeige, dass $\alpha(U) \leq 2$, wann immer $|U| \leq 3$.

Bemerkung: Man vermutet sogar, dass $\alpha(U) \leq 2$ für beliebige Mengen U.

2.8 Heuristiken für das Metrische TSP

Wir besprechen das Traveling-Salesman Problem (TSP). Zuerst werden wir feststellen, dass TSP, wenn nicht eingeschränkt, keine vernünftigen Approximationen zulässt. Wir werden dann das metrische TSP betrachten. Hier werden wir exemplarisch Greedy-Algorithmen in ihrem Einsatzgebiet als Heuristiken untersuchen.
Im allgemeinen Traveling-Salesman Problem (TSP) sind n Orte $v_1, \ldots, v_n$ sowie Distanzen $d(v_i, v_j) \geq 0$ zwischen je zwei verschiedenen Orten gegeben. Eine Rundreise entspricht einer Permutation $v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)}$ der Orte: Wenn $v_{\pi(i)}$ besucht wird, dann wird $v_{\pi(i+1)}$ für $i < n$, bzw v_1 für $i = n$ als nächster Ort besucht. Die Länge der Rundreise ist dann

$$d\left(v_{\pi(n)}, v_{\pi(i)}\right) + \sum_{i=1}^{n-1} d\left(v_{\pi(i)}, v_{\pi(i+1)}\right).$$

Gesucht ist eine Rundreise minimaler Länge. Die Sprachenversion zum Traveling-Salesman Problem („Existiert eine Rundreise der Länge höchstens k?“) ist $\mathcal{NP}$-vollständig, und sogar das Approximationsproblem ist sehr schwierig.

Satz 2.29 *(a) Die Sprache*

$$TSP = \{(n, d, k) \mid \textit{Es gibt eine Rundreise der Länge höchstens } k\ \}$$

ist $\mathcal{NP}$-vollständig.
(b) Angenommen, es gilt $\mathcal{P} \neq \mathcal{NP}$. Dann gibt es keinen effizienten 2^n-approximativen Algorithmus für TSP!

Beweis (a): Wir zeigen, dass HAMILTONSCHER KREIS

> Gibt es einen Hamiltonschen Kreis, also einen Kreis, der alle Knoten eines ungerichteten Graphen $G = (V, E)$ genau einmal besucht?

auf TSP reduziert werden kann. Wenn der Graph $G = (V, E)$ mit n Knoten eine Eingabe für HAMILTONSCHER KREIS ist, dann erfinden wir die folgende Instanz $T(G)$ für TSP:

- Die Orte entsprechen den Knoten von V.

- Für alle Knoten $u, v \in V$: Wenn $\{u, v\}$ eine Kante in G ist, dann setzen wir $d(u, v) = 1$ und ansonsten $d(u, v) = D$.

Wir beachten zuerst, dass G genau dann einen Hamiltonschen Kreis besitzt, wenn es in der transformierten Instanz eine Rundreise der Länge höchstens n gibt. Besitzt G hingegen keinen Hamiltonschen Kreis, dann hat jede Rundreise mindestens die Länge $n-1+D$. Für $D > 1$ gilt also

$$G \in \text{HAMILTONSCHER KREIS} \Leftrightarrow T(G) \text{ hat eine Rundreise der Länge } n,$$

und die Reduktion ist nachgewiesen. Beachte, dass HAMILTONSCHER KREIS $\mathcal{NP}$ vollständig ist.
(b) Wir haben auch erhalten, dass

$$\begin{aligned} & G \notin \text{HAMILTONSCHER KREIS} \\ \Leftrightarrow \quad & T(G) \text{ hat eine Rundreise der Länge mindestens } n - 1 + D \end{aligned}$$

gilt. Wenn $\mathcal{P} \neq \mathcal{NP}$, dann können wir HAMILTONSCHER KREIS nicht mit einem effizienten Algorithmus lösen. Also hat jeder effiziente Approximationsalgorithmus für TSP mindestens den Approximationsfaktor $\frac{n-1+D}{n}$. Die Behauptung folgt für $D = n \cdot 2^n$. □

Wir haben in Satz 2.29 gesehen, dass effiziente Algorithmen keine vernünftigen Approximationsfaktoren für das unbeschränkte TSP Problem erreichen. Die Situation verbessert sich, wenn wir verlangen, dass die Distanzen zwischen den Orten (oder Punkten) durch eine Metrik gegeben sind: d ist genau dann eine Metrik, wenn für alle Punkte u, v, w gilt:

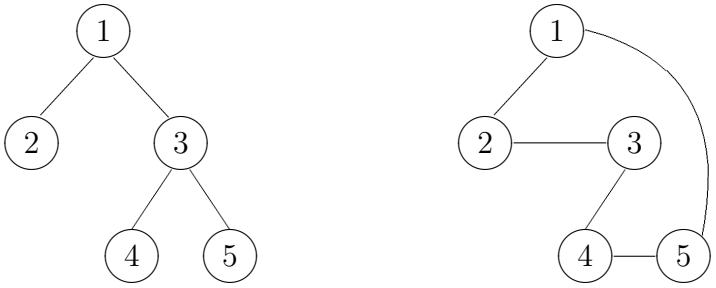
- $d(u, u) = 0$,
- $d(u, v) = d(v, u)$ und
- $d(u, v) + d(v, w) \geq d(u, w)$.

In diesem Fall spricht man vom metrischen TSP[2]. Wir beschreiben zuerst die Spannbaum-Heuristik, die den Approximationsfaktor 2 für das metrische TSP erreicht. Sei G der Graph mit allen Orten als Knoten und Verbindungen zwischen je zwei Knoten; die Kante $\{u, v\}$ erhält dann die Distanz zwischen u und v als Gewicht. Wir konstruieren eine Rundreise aus einem minimalen Spannbaum B für G, indem wir die zu besuchenden Orte in der Reihenfolge besuchen, in der Präorder die Knoten von B besucht.
Wie lang ist die Rundreise? Da ein Rückspringen während der Präorder-Traversierung einem nochmaligen Durchlauf der Kanten in der Gegenrichtung entspricht, ist die aus B erhaltene Rundreise höchstens doppelt so lang im Vergleich zum Gewicht von B. (Das nochmalige Durchlaufen der Kanten ist ein Umweg gegenüber einer Direktverbindung, und der Approximationsfaktor zwei folgt aus der Transitivität der Metrik.) Schließlich beachten wir, dass eine Rundreise mindestens so lang ist wie das Gewicht des minimalen Spannbaums B: Entfernen wir nämlich irgendeine Kante einer Rundreise, dann wird die Rundreise zu einem Spannbaum.

[2]Die Webseite `www.tsp.gatech.edu` erhält weitere Informationen

Satz 2.30 *Die Spannbaum Heuristik berechnet eine 2-approximative Rundreise für das metrische TSP.*

Beispiel 2.7 Wir durchlaufen einen Spannbaum in der Präorder-Reihenfolge und wählen Direktverbindungen während des Rückspringens.



Die Länge der Rundreise lässt sich abschätzen, wenn wir die Länge der Direktverbindungen durch die Länge des Umwegs abschätzen. Wir erhalten eine Länge von höchstens

$$d_{1,2} + (d_{2,1} + d_{1,3}) + d_{3,4} + (d_{4,3} + d_{3,5}) + (d_{5,3} + d_{3,1}).$$

Beachte, dass alle Kanten des Spannbaums genau zweimal in der Summe auftreten.

Im obigen Beispiel werden die Abkürzungen $\{2,3\}$, $\{4,5\}$ und $\{5,1\}$ gewählt. Wir versuchen jetzt bessere Abkürzungen zu finden.

Behauptung 2.1 *Sei G ein ungerichteter, zusammenhängender Graph. Dann besitzt G genau dann eine Euler-Tour (d.h. einen Kreis, der jede Kante genau einmal durchläuft), wenn jeder Knoten eine gerade Anzahl von Nachbarn besitzt.*

Beweis: Wenn es eine Euler-Tour gibt, betritt man auf einer solchen Tour jeden Knoten so oft, wie man ihn verlässt. Da jede Kante auf der Tour genau einmal besucht wird, besitzt jeder Knoten eine gerade Anzahl von Nachbarn.
Um die Umkehrung zu beweisen, zerlege den Graphen in kantendisjunkte Kreise. Zwei Kreise mit einem gemeinsamen Knoten lassen sich zu einem Kreis verschmelzen. Da G zusammenhängend ist, lassen sich somit alle Kreise zu einer Euler-Tour verschmelzen. □

Algorithmus 2.31 Der Algorithmus von Christofides für Δ-TSP

(1) Die Eingabe besteht aus einer Menge $V = \{1, \ldots, n\}$ von n Punkten und den Distanzen $d_{v,w}$. Die Distanzfunktion d erfüllt die Dreiecksungleichung.

(2) Setze $G := (V, \{\{v, w\} \mid v, w \in V, v \neq w\})$ und weise der Kante $e = \{v, w\}$ die Distanz $d_e = d_{v,w}$ zu.

(3) Berechne einen minimalen Spannbaum T für G.

(4) Sei U die Menge der Knoten von T mit einer ungeraden Anzahl von Nachbarn in T. Berechne ein minimales Matching M der Größe $\frac{1}{2} \cdot |U|$ auf den Knoten in U.

(5) Füge die Kanten in M zu den Kanten in T hinzu und konstruiere eine Euler-Tour E.

(6) Berechne aus E eine Rundreise durch Abkürzungen.

Bemerkung 2.8 Der Algorithmus von Christofides ist kein Priority-Algorithmus, da die Eingabe mehrfach betrachtet wird ohne Entscheidungen über die Rundreise zu treffen: Zuerst wird ein minimaler Spannbaum bestimmt und dann ein minimales Matching.
Deshalb sollte man den Algorithmus von Christofides auch nicht als Greedy-Algorithmus auffassen.

Beachte, dass jeder ungerichtete Graph $G = (V, E)$ eine gerade Anzahl von Knoten mit einer ungeraden Anzahl Nachbarn besitzt, denn in der Summe $\sum_{v \in V} |\{w \mid \{v, w\} \in E\}|$ zählen wir jede Kante zweimal und die Summe hat damit den Wert $2|E|$. Der Wert einer Summe kann aber nur dann gerade sein, wenn die Anzahl der ungeraden Summanden gerade ist. Also hat die Menge U aus Schritt (4) eine gerade Anzahl von Elementen.
Wir analysieren den Approximationsfaktor. Sei L_{opt} die Länge einer kürzesten Rundreise, MS_{opt} die Länge eines minimalen Spannbaumes und Match das Gewicht eines minimalen Matchings. Dann ist

$$\text{MS}_{\text{opt}} \leq L_{\text{opt}},$$

denn eine Rundreise mit einer entfernten Kante ist ein Spannbaum, dessen Gewicht mindestens MS_{opt} ist. Die Länge einer kürzesten Rundreise auf den Punkten in U ist natürlich durch L_{opt} nach oben beschränkt. Eine jede Rundreise auf den gerade vielen Punkten aus U zerlegt sich in zwei perfekte Matchings und wir erhalten

$$\text{Match} \leq \frac{1}{2} \cdot L_{\text{opt}}.$$

Schließlich ist die Länge einer Euler-Tour durch

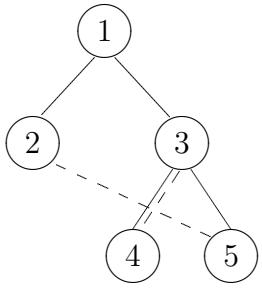
$$\text{MS}_{\text{opt}} + \text{Match} \leq \frac{3}{2} \cdot L_{\text{opt}}$$

beschränkt. Aber die Rundreise resultiert durch Abkürzungen aus der Euler-Tour und wir erhalten:

Satz 2.32 *Der Algorithmus von Christofides ist ein $\frac{3}{2}$-Approximationsalgorithmus. Seine Laufzeit für n Punkte ist durch $O(n^3)$ beschränkt.*

Beweis: Zur Zeitanalyse: Wir haben bisher keine Algorithmen zur Konstruktion minimaler perfekter Matchings kennengelernt. Es existieren aber Algorithmen mit Laufzeit $O(n^3)$ (siehe [PaSt]). Damit ist Schritt (4) der aufwändigste Schritt und wir erhalten die Laufzeit $O(n^3)$. □

Im obigen Beispiel haben die Knoten $2, 3, 4$ und 5 ungeraden Grad. Wir konstruieren das Matching $M = \{ \{2, 5\}, \{3, 4\} \}$ und eine Euler-Tour $(1, 2, 5, 3, 4, 3, 1)$. Dementsprechend erhalten wir mit Abkürzungen die Rundreise $(1, 2, 5, 3, 4, 1)$.



Wir besprechen zwei weitere Heuristiken. **Nearest-Insertion** beginnt mit der kürzesten Kante und baut iterativ eine Rundreise. In einem Iterationsschritt wird der Punkt gewählt, der unter allen noch nicht erfassten Punkten einen kleinsten Abstand zur gegenwärtigen partiellen Rundreise besitzt. Dieser Punkt wird dann zwischen ein Paar benachbarter Punkte der partiellen Rundreise eingefügt, wobei das Paar gewählt wird, das zu einem möglichst kleinen Anstieg der Länge der Rundreise führt.
Farthest-Insertion verfährt wie Nearest-Insertion, nur wird diesmal der am weitesten von der gegenwärtigen partiellen Rundreise entfernte Punkt gewählt. Dieser Punkt ist dann wie in Nearest-Insertion einzufügen. Farthest-Insertion beginnt mit einer längsten Kante.
Wie soll man Heuristiken miteinander vergleichen? Man kann zeigen, dass Nearest-Insertion den Approximationsfaktor zwei besitzt und damit vom theoretischen Standpunkt aus vergleichbar mit der Spannbaum-Heuristik und dem Algorithmus von Christofides ist. Betrachtet man die Größe der Approximationsfaktoren, ist der Christofides Algorithmus vorteilhafter.

Offenes Problem 2
Bestimme den Approximationsfaktor von Farthest-Insertion. Es ist bekannt, dass der Faktor höchstens $\lceil \log_2 n \rceil + 1$, aber mindestens $2,43$ in der Ebene und $6,5$ in metrischen Räumen ist.

Aufgabe 31
Nearest-Neighbor wächst eine partielle Rundreise Knoten um Knoten. Wenn gegenwärtig der Knoten v erreicht ist, dann wird nach dem Knoten w gesucht, dessen Distanz zu v minimal unter allen noch nicht traversierten Knoten ist. Die Kante $\{v, w\}$ wird hinzugefügt und das Verfahren wird mit w wiederholt. Sind alle Knoten traversiert, dann wird die partielle Rundreise zu einer vollständigen Rundreise geschlossen.
Zeige, dass der Approximationsfaktor von Nearest-Neighbor unbeschränkt ist. (Es ist bekannt, dass der Approximationsfaktor durch $\frac{1}{2} \cdot (\log_2 n + 1)$ nach oben beschränkt ist. Eine weitaus bessere Approximationsleistung ist aber durchaus möglich.)

Der Approximationsfaktor ist als Worst-Case-Fall definiert. Wie sieht der „erwartete“ Approximationsfaktor aus? Dazu hat D.S. Johnson [Joh] eine experimentelle Analyse vorgestellt, bei der bis zu 100.000 Punkte zufällig aus dem Quadrat $[0, 1]^2$ ausgewählt wurden. Auch hier ist der Algorithmus von Christofides der Gewinner, allerdings mit geringem Abstand zu Farthest-Insertion. Nearest-Insertion wird deutlich geschlagen und endet auf Platz drei. Der klare Verlierer ist die Spannbaum-Heuristik, die bis zu 40% über der optimalen Länge einer Rundreise liegt. Der Christofides Algorithmus und Farthest-Insertion sind ungefähr 10% über dem Wert einer optimalen Rundreise, während Nearest-Insertion um bis zu 25% falsch liegt.

Kapitel 3

Dynamische Programmierung

Im dynamischen Programmieren wird das Ausgangsproblem P_0 in Teilprobleme $P_1, \ldots, P_t$ aufgebrochen. Die Teilprobleme werden dann, einer Schwierigkeitshierarchie entsprechend, gelöst:

- Die Schwierigkeitshierarchie ist ein gerichteter azyklischer Graph mit Knotenmenge $P_0, P_1, \ldots, P_t$.
- Eine Kante (P_i, P_j) wird eingesetzt, wenn die Lösung zu Teilproblem P_i in der Berechnung der Lösung von P_j verwendet wird.

Von dieser Perspektive aus gesehen ist die dynamische Programmierung eine weitreichende Verallgemeinerung des Divide & Conquer Verfahrens: Die Schwierigkeitshierarchie eines dynamischen Prgrammierverfahrens wird durch einen gerichteten azyklischen Graphen beschrieben, die Schwierigkeitshierarchie eines Divide & Conquer Verfahrens hingegen entspricht einem Baum. Während die Lösung eines Teilproblems im Divide & Conquer Verfahren nur von dem einen Eltern-Problem benötigt wird, sind Lösungen von Teilproblemen im dynamischen Programmieren für alle Elternknoten von Relevanz und werden deshalb abgespeichert.

Aufgabe 32
Ein Automat soll beliebige eingetippte Beträge in Münzen ausgeben. Dabei soll eine minimale Anzahl von Münzen herausgegeben werden. Jeder Betrag sei durch Münzen darstellbar.

(a) Die k Münzen seien die k Potenzen $1, c, c^2, \ldots, c^{k-1}$ einer natürlichen Zahl $c > 1$. Löse das Problem durch einen Greedy-Algorithmus.

(b) Die Münzen seien beliebig. Löse das Problem durch dynamisches Programmieren und zeige, dass Greedy-Algorithmen mit den bekannten Strategien versagen.

Weise die Korrektheit der Algorithmen nach und analysiere jeweils die Laufzeit in Abhängigkeit vom Betrag B und der Anzahl verschiedener Münztypen.

Aufgabe 33
Verwende dynamisches Programmieren, um in Polynomialzeit zu einer gegebenen Folge $a_1, \ldots, a_n$ von n Zahlen eine längste monoton aufsteigende Teilfolge zu finden. Dabei ist eine Teilfolge eine Folge $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ mit $1 \leq i_1 < i_2 < \ldots < i_k \leq n$.

Aufgabe 34
Es ist ein Text mit n Worten $w_1, \ldots, w_n$ gegeben, wobei das Wort w_i die Länge l_i habe. Der Text soll ausgedruckt werden, wobei in eine Zeile m Zeichen passen. Dabei sollen „Lücken“ am rechten Rand möglichst klein gehalten werden. Wenn zwischen zwei Worten immer ein Leerzeichen steht, und in einer Zeile Worte w_i bis w_j stehen, bleiben am rechten Rand $m - j + i - \sum_{k=i}^{j} l_k$ Leerzeichen. Verwende dynamisches Programmieren, um in Polynomialzeit den Text so auszudrucken, dass die Summe der Quadrate aller Lückengrößen am rechten Rand (außer in der letzten Zeile) minimiert wird.

Aufgabe 35
Es sei ein gerichteter Graph $G = (V, E)$ mit Kantenmarkierungen $w(u, v) \in \{1, \ldots, k\}$ gegeben. Zusätzlich ist für jede Kante eine Wahrscheinlichkeit $p(u, v)$ gegeben, so dass die Summe aller Wahrscheinlichkeiten der Kanten, welche einen Knoten verlassen, 1 ergibt. Die Wahrscheinlichkeit eines Pfades ist das Produkt der Kantenwahrscheinlichkeiten.
Beschreibe einen effizienten Algorithmus, der, gegeben einen solchen Graphen, einen Startknoten, sowie eine Sequenz von Zahlen aus $\{1, \ldots, k\}$, entscheidet, ob im Graphen ein mit dem Startknoten beginnender Pfad diese Labelsequenz besitzt. Wenn dies der Fall ist, soll auch ein solcher Pfad mit maximaler Wahrscheinlichkeit ausgegeben werden.

Aufgabe 36
Die Editier Distanz zwischen zwei Zeichenketten sei die minimale Länge einer Folge elementarer Operationen, welche die eine Zeichenkette in die andere transformiert. Als elementare Operationen betrachten wir das Einfügen eines Buchstaben an beliebiger Stelle, das Löschen eines Buchstaben an beliebiger Stelle, sowie das Ersetzen eines Buchstaben an einer Stelle durch einen anderen.
Gib einen möglichst effizienten Algorithmus an, der zu zwei gegebenen Zeichenketten die Editier Distanz bestimmt.

Aufgabe 37
Zwei Strings $x, y \in (\Sigma \cup \{-\})^*$ bilden ein Alignment der Strings $u, v \in \Sigma^*$ genau dann, wenn

(1) u durch Entfernen der "$-$"-Symbole aus x entsteht und v durch Entfernen der "$-$"-Symbole aus y entsteht

(2) $|x| = |y|$ (d.h. x und y besitzen dieselbe Länge)

(3) $x_i \neq -$ oder $y_i \neq -$ für jedes i.

Gegeben ist ein Ähnlichkeitsmaß d zwischen Symbolen, wobei die Qualität eines Alignments x, y von u, v durch $d(x, y) = \sum_i d(x_i, y_i)$ definiert ist.
Im *lokalen Alignment Problem* sind zwei Strings $u = u_1 \ldots u_n \in \Sigma^*, v = v_1 \ldots v_m \in \Sigma^*$ gegeben. Gesucht sind Teilstrings $u' = u_i u_{i+1} \ldots u_k$ von u und $v' = v_j v_{j+1} \ldots v_l$ von v, so dass u', v' ein Alignment x, y mit möglichst großer Ähnlichkeit $d(x, y)$ besitzen. Zum Beispiel sei für alle $a \neq b \in \Sigma \cup \{-\}$, $d(a, a) = 2$, $d(a, b) = -2$ und $d(a, -) = d(-, b) = -1$. Dann ist

$$\begin{array}{cccccccc} x & = & G & C & T & C & A & A & G \\ y & = & G & - & T & - & A & T & G \end{array}$$

ein lokales Alignment von $u = GCTCAAGA$ und $v = CTGTATG$ mit Qualität $d(x, y) = 4$.
Zeige, dass das lokale Alignment Problem für beliebige Strings u und v in Zeit $O(|u| \cdot |v|)$ gelöst werden kann.

3.1 Das RUCKSACK-Problem

Wir erinnern an die Definition des RUCKSACK-Problems: Es sind eine Gewichtsschranke G und n Objekte $D_1, \ldots D_n$ gegeben. Das Objekt D_i hat das Gewicht $g_i > 0$ und den Wert $w_i > 0$; während die Gewichte reelle Zahlen sind, fordern wir, dass die Werte natürliche

Zahlen sind. Wir möchten einen imaginären Rucksack so bepacken, dass das Gesamtgewicht der eingepackten Objekte höchstens G beträgt und der Gesamtwert maximiert wird. Wir suchen also einen Vektor $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ mit

$$\sum_{i=1}^{n} x_i \cdot g_i \leq G,$$

so dass $\sum_{i=1}^{n} x_i \cdot w_i$ größtmöglich ist. Das RUCKSACK-Problem in der Sprachform (gibt es eine Auswahl mit Wert mindestens W und Gewicht höchstens G?) ist $\mathcal{NP}$-vollständig.

Aufgabe 38
Beim fraktionalen RUCKSACK-Problem sind n Objekte $D_1, \ldots, D_n$ mit Gewichten $g_i \in \mathbb{N}$ und Werten $w_i \in \mathbb{N}$ gegeben. Es soll eine Beladung des Rucksacks gefunden werden, so dass eine gegebebene Gewichtsschranke G nicht überschritten wird, aber der Gesamtwert der eingepackten Objekte maximal ist. Eine Beladung darf dabei von jedem Objekt D_i einen Bruchteil $x_i \in [0,1]$ wählen, und dieser Bruchteil hat Gewicht $x_i \cdot g_i$ und Wert $x_i \cdot w_i$.
(a) Löse das fraktionale RUCKSACK-Problem durch einen effizienten Algorithmus und weise die Optimalität der gefundenen Lösung nach.
(b) Zeige, dass der Approximationsfaktor unbeschränkt ist.
(c) Sei opt der Wert einer besten Bepackung, wpk der Wert der von unserer Heuristik erreicht wird und sei $w_{\max}$ der maximale Wert eines Objekts. Zeige

$$\text{opt} \leq 2 \cdot \max\{\text{wpk}, w_{\max}\}.$$

Satz 3.1 *Das RUCKSACK-Problem für n Objekte und Wertesumme $W = \sum_{i=1}^{n} w_i$ kann in Zeit $O(n \cdot W)$ gelöst werden.*

Beweis: Wir beschreiben eine Anwendung des dynamischen Programmierens.

(1) Wahl der Teilprobleme: Für jedes $W^* \leq W$ und jedes $i = 1, \ldots, n$ bestimme das minimale Gesamtgewicht $\text{Gewicht}_i(W^*)$ einer Auswahl aus den ersten i Objekten mit Wertesumme genau W^*.

(2) Lösung der Teilprobleme:

$$\text{Gewicht}_i(W^*) = \min\left\{\text{Gewicht}_{i-1}\left(W^* - w_i\right) + g_i,\ \text{Gewicht}_{i-1}\left(W^*\right)\right\}.$$

Analysieren wir die Laufzeit. Wir haben höchstens

$$nW = n \cdot \sum_{i=1}^{n} w_i$$

viele Teilprobleme, die jeweils in Zeit $O(1)$ gelöst werden können und die Gesamtlaufzeit ist deshalb $O(nW)$. □

Aufgabe 39
Der Beweis zu Satz 3.1 enthält einen dynamischen Programmieransatz, der nicht vollständig beschrieben ist.

Ergänze den Ansatz um die Initialisierung der Variablen Gewicht$_i(W^*)$ und beschreibe, wie der Wert der optimalen Bepackung ausgegeben wird.

Wir haben einen sehr guten Algorithmus erhalten, falls W klein ist. Wir zeigen jetzt, wie man aus der exakten Lösung für kleine Werte ein volles polynomielles Approximationsschema für beliebig große Werte erhält.

Algorithmus 3.2 **Ein volles polynomielles Approximationsschema für RUCKSACK**

(1) Sei $(w_1, \ldots, w_n, g_1, \ldots, g_n, G)$ eine beliebige Instanz des RUCKSACK-Problems. Der Approximationsfaktor $1+\varepsilon$ sei vorgegeben. Entferne alle Objekte, deren Gewicht die Gewichtsschranke G übersteigt.

(2) Packe nur das Objekt mit größtem Wert in den Rucksack. Der erhaltene Wert sei W_1.

(3) Die Werte werden nach unten skaliert, nämlich setze

$$s = \frac{\varepsilon \cdot \max_j w_j}{n} \quad \text{und} \quad w_i^* = \lfloor \frac{w_i}{s} \rfloor.$$

(4) Berechne eine exakte Lösung x für die Instanz $(w_1^*, \ldots, w_n^*, g_1, \ldots, g_n, G)$. Wenn x die Objektmenge $I \subseteq \{1, \ldots, n\}$ in den Rucksack packt, dann setze $W_2 = \sum_{i \in I} w_i$ und übernimm die Bepackung für die alten Werte.

(5) Gib die beste der beiden Bepackungen aus.

Satz 3.3 *Algorithmus 3.2 ist ein volles polynomielles Approximationsschema mit Laufzeit* $O(\frac{1}{\varepsilon} \cdot n^3)$.

Beweis: Wir können annehmen, dass alle Objekte eingepackt werden können, da der Algorithmus anfänglich zu schwere Objekte entfernt. Die neuen Werte sind durch $\frac{n}{\varepsilon}$ beschränkte natürliche Zahlen. Wir können somit das Optimierungsproblem in den neuen Werten (und alten Gewichten) in Zeit $O(n \cdot (n \cdot \frac{n}{\varepsilon})) = O(\frac{1}{\varepsilon} \cdot n^3)$ exakt lösen.
Sei B die gefundene Bepackung und B_{opt} die optimale Bepackung für die alten Werte. Dann gilt für den Skalierungsfaktor

$$s = \frac{\varepsilon \cdot \max_j w_j}{n},$$

dass

$$\begin{aligned}
\sum_{i \in B_{\text{opt}}} w_i &\leq \sum_{i \in B_{\text{opt}}} s \cdot (\lfloor \frac{w_i}{s} \rfloor + 1) \leq \sum_{i \in B_{\text{opt}}} s \cdot \lfloor \frac{w_i}{s} \rfloor + sn \\
&\leq \sum_{i \in B} s \cdot \lfloor \frac{w_i}{s} \rfloor + sn \text{ denn } B \text{ ist die beste Bepackung für die Werte } w_i^* = \lfloor \tfrac{w_i}{s} \rfloor \\
&\leq \sum_{i \in B} w_i + sn.
\end{aligned}$$

Fall 1: $\sum_{i\in B} w_i \geq \max_j w_j$. Es ist

$$\frac{\sum_{i\in B_{\text{opt}}} w_i}{\sum_{i\in B} w_i} \leq \frac{\sum_{i\in B} w_i \ + sn}{\sum_{i\in B} w_i} \leq 1 + \frac{sn}{\sum_{i\in B} w_i} \leq 1 + \frac{sn}{\max_j w_j} \leq 1 + \varepsilon. \tag{3.1}$$

Fall 2: $\sum_{i\in B} w_i < \max_j w_j$.
Algorithmus 3.2 findet eine Bepackung mit Wert $\max_j w_j$. Es ist

$$\frac{\sum_{i\in B_{\text{opt}}} w_i}{\max_j w_j} \leq \frac{\sum_{i\in B} w_i + sn}{\max_j w_j} \leq \frac{\max_j w_j + sn}{\max_j w_j},$$

denn wir haben die Fallannahme benutzt. Die Behauptung folgt jetzt analog zu (3.1). □

3.2 BIN PACKING

In BIN PACKING sind n Objekte mit Gewichten $g_1, \ldots, g_n$ in möglichst wenige Behälter zu verteilen, so dass jeder Behälter mit einem Gesamtgewicht von höchstens 1 belastet wird. Eine Anwendung von BIN PACKING ist die Verteilung von Werbespots auf möglichst wenige Programm-Unterbrechungen fester Länge.
Wir zeigen zuerst, dass BIN PACKING keine effizienten $(\frac{3}{2} - \varepsilon)$-approximativen Algorithmen besitzt! Dazu betrachten wir das $\mathcal{NP}$-vollständige PARTITION Problem:

> Für n natürliche Zahlen $x_1, \ldots, x_n$ ist festzustellen, ob es eine Teilmenge $I \subseteq \{1, \ldots, n\}$ mit
>
> $$\sum_{i\in I} x_i = \frac{1}{2} \cdot \sum_{i=1}^{n} x_i$$
>
> gibt.

Wir können aber PARTITION offensichtlich als ein BIN PACKING Problem mit Kapazität $\frac{1}{2} \cdot \sum_{i=1}^{n} a_i$ statt Kapazität 1 formulieren. Als Konsequenz wird ein $(\frac{3}{2} - \varepsilon)$-approximativer Algorithmus für BIN PACKING das PARTITION Problem lösen!
Aber diese Beobachtung schließt nicht aus, dass es scharfe Approximationen mit einem kleinen additiven Zusatzterm gibt. Genau das nutzen wir in Abschnitt 3.2.2 aus, um ein polynomielles Approximationsschema für „die kleinste Anzahl von Behältern +1“ zu erhalten.

3.2.1 Greedy Algorithmen

Wir besprechen zuerst zwei On-line Algorithmen, die das $i$te Objekt einem Behälter zuweisen müssen, ohne die zukünftigen Objekte, also die Objekte $i + 1, \ldots$ zu kennen.

Algorithmus 3.4 **Next-Fit**

(1) n Objekte mit Gewichten $g_1, \ldots, g_n \in [0, 1]$ sind in Behälter zu verteilen.

(2) Bearbeite die Objekte der Reihe nach:

> Füge das $i$te Objekt in den zuletzt geöffneten Behälter ein, wenn es „passt“. Ansonsten öffne einen neuen Behälter.
>
> *Kommentar:* Zwei aufeinanderfolgende Behälter tragen eine Gesamtbelastung größer als 1. Da mindestens $\lceil \sum_i g_i \rceil$ Behälter benötigt werden, aber höchstens $2 \cdot \lceil \sum_i g_i \rceil$ Behälter geöffnet werden, besitzt Next-Fit den Approximationsfaktor 2.

Die sehr einfache Strategie Next-Fit erreicht also bereits den Approximationsfaktor 2 und es besteht somit berechtigter Anlass zur Hoffnung auf substantielle Verbesserungen. Wir erreichen zum Beispiel eine deutliche Verbesserung, wenn wir versuchen, das Öffnen neuer Behälter zu erschweren.

Algorithmus 3.5 **First-Fit**

(1) n Objekte mit Gewichten $g_1, \ldots, g_n$ sind in Behälter zu verteilen.

(2) Bearbeite die Objekte der Reihe nach:

> Füge das $i$te Objekt in den ersten Behälter ein, in den es passt. Passt das Objekt in keinen Behälter, dann öffne einen neuen Behälter.

Aufgabe 40
Sei opt die minimale Behälterzahl. Zeige, dass First-Fit höchstens $1.7 \cdot \text{opt} + 2$ Behälter öffnet.

Aufgabe 41
Zeige, dass kein On-line Algorithmus für BIN PACKING stets höchstens

$$\left(\frac{4}{3} - \delta\right) \cdot opt + C$$

Behälter öffnet. Hier ist $\delta > 0$ eine beliebig kleine und $C > 0$ eine beliebig große Konstante. Beachte, dass der On-line Algorithmus *nicht* effizient sein muss. *Hinweis*: Betrachte eine Instanz I, die aus n Objekten mit Gewicht $\frac{1}{2} - \epsilon$, gefolgt von n Objekten mit Gewicht $\frac{1}{2} + \epsilon$ besteht.

Wir wenden uns jetzt Off-line Strategien zu.

Algorithmus 3.6 **First-Fit-Decreasing**

(1) n Objekte mit Gewichten $g_1, \ldots, g_n$ sind in Behälter zu verteilen.

(2) Sortiere die Objekte absteigend nach ihrem Gewicht.

(3) Bearbeite die Objekte der Reihe nach:

> Füge das $i$te Objekt in den ersten Behälter ein, in den es passt. Passt das Objekt in keinen Behälter, dann öffne einen neuen Behälter.
>
> *Kommentar:* First-Fit-Decreasing verhält sich also wie First Fit *nachdem* die Ordnung der Objekte verändert wurde.

Zur Analyse von First-Fit-Decreasing unterteilen wir die Objekte in die folgenden vier Gewichtsklassen:

$$\begin{aligned} A &= \{i \mid \frac{2}{3} < g_i\}, \qquad B = \{i \mid \frac{1}{2} < g_i \leq \frac{2}{3}\}, \\ C &= \{i \mid \frac{1}{3} < g_i \leq \frac{1}{2}\}, \quad D = \{i \mid g_i \leq \frac{1}{3}\}. \end{aligned}$$

First-Fit-Decreasing arbeitet die Objekte nach absteigendem Gewicht ab und wir beobachten seine Vorgehensweise auf den Objekten in $A \cup B \cup C$. Zuerst werden die Objekte aus A verteilt, wobei jedes dieser schweren Objekte seinen eigenen Behälter benötigt. Danach werden die Objekte aus B bearbeitet, die ebenfalls jeweils einen eigenen Behälter benötigen. Bisher ist die Verteilung der Objekte natürlich optimal. Schließlich werden die Objekte aus C verteilt und zwar wird jedes C-Objekt in den Behälter gepackt, dessen verbleibende Restkapazität minimal ist.
Die Verteilung der C-Objekte ist aber auch bestmöglich, denn es geht nur darum, Behälter mit B-Objekten wenn möglich auch mit einem C-Objekt zu nutzen. (Die Nutzung eines Behälters durch zwei C-Objekte ist stets möglich.) Also ist die Vorgehensweise von First-Fit-Decreasing optimal und wir erhalten:

Satz 3.7 *Wenn First-Fit-Decreasing keinen Behälter erzeugt, in dem nur Objekte aus D liegen, dann berechnet First-Fit-Decreasing eine optimale Lösung. Ansonsten ist die Anzahl geöffneter Behälter durch $\frac{3}{2} \cdot$ opt $+1$ nach oben beschränkt.* [1]

Beweis: Wenn kein Behälter nur Objekte aus D enthält, dann musste First-Fit-Decreasing nach Verarbeitung der Objekte aus $A \cup B \cup C$ keinen weiteren Behälter öffnen und damit wird eine optimale Lösung berechnet.
Werden hingegen neue Behälter geöffnet, dann besitzen alle Behälter, bis auf möglicherweise den letzten Behälter, ein Mindestgewicht von mehr als $\frac{2}{3}$ und die Behauptung folgt. □

Bemerkung 3.1 Wir geben eine worst-case Eingabe für First-Fit-Decreasing an. Insgesamt liegen $5n$ Objekte vor, von denen n Objekte das Gewicht $\frac{1}{2} + \varepsilon$, n Objekte das Gewicht $\frac{1}{4} + 2\varepsilon$, n Objekte das Gewicht $\frac{1}{4} + \varepsilon$ und $2n$ Objekte das Gewicht $\frac{1}{4} - 2\varepsilon$ besitzen. Eine optimale Verteilung füllt genau $\frac{3}{2} \cdot n$ Behälter durch n-maliges Zusammenstellen der Gewichte $\frac{1}{2} + \varepsilon$, $\frac{1}{4} + \varepsilon$ und $\frac{1}{4} - 2\varepsilon$ sowie durch $\frac{n}{2}$-maliges Zusammenstellen der Gewichte $\frac{1}{4} + 2\varepsilon$, $\frac{1}{4} + 2\varepsilon$, $\frac{1}{4} - 2\varepsilon$ und $\frac{1}{4} - 2\varepsilon$.
First-Fit-Decreasing füllt zuerst n Behälter mit Gewicht $\frac{1}{2} + \varepsilon$ und schließt diese Behälter sodann durch die zusätzliche Bepackung mit Gewicht $\frac{1}{4} + 2\varepsilon$. Danach werden jeweils drei Objekte mit Gewicht $\frac{1}{4} + \varepsilon$ zusammengepackt und schließlich jeweils vier Objekte mit Gewicht $\frac{1}{4} - 2\varepsilon$. Insgesamt werden also $n + \frac{n}{3} + \frac{n}{2} = \frac{11 \cdot n}{6}$ Behälter benötigt.
Beachte, dass $\frac{11 \cdot n}{6} / \frac{3 \cdot n}{2} = \frac{11}{9}$.

Algorithmus 3.8 **Best-Fit-Decreasing**

[1] Es ist sogar bekannt, dass First-Fit-Decreasing höchstens $\frac{11}{9} \cdot \mathtt{opt} + 2$ Behälter öffnet.

(1) n Objekte mit Gewichten $g_1, \ldots, g_n$ sind in Behälter zu verteilen.

(2) Sortiere die Objekte absteigend nach ihrem Gewicht.

(3) Bearbeite die Objekte der Reihe nach:

> Füge das $i$te Objekt in den Behälter ein, dessen verbleibende Restkapazität minimal ist. Passt das Objekt in keinen Behälter, dann öffne einen neuen Behälter.

Best-Fit-Decreasing ist mindestens so gut wie First-Fit-Decreasing, denn:

Aufgabe 42
Zeige: Für jede Instanz von BIN PACKING wird Best-Fit-Decreasing höchstens so viele Behälter wie First-Fit-Decreasing öffnen.

Aufgabe 43
Im *d-dimensionalen* BIN PACKING sind n Objekte gegeben, deren Gewichte durch Vektoren $\vec{g}_i = (g_{i1}, \ldots, g_{id})$ der Länge d mit $g_{ij} \in [0, 1]$ beschrieben werden. Jeder Behälter besitzt die Kapazität $(1, 1, \ldots, 1)$. Die n Objekte sind in möglichst wenige Behälter zu verteilen, so dass jeder Behälter B mit einem Gesamtgewicht $\sum_{i \in B} \vec{g}_i$ von höchstens $(1, 1, \ldots, 1)$ belastet wird, d.h. es muss

$$\sum_{i \in B} g_{ij} \leq 1$$

für alle Koordinaten $j = 1, \ldots, d$ gelten.
Beschreibe einen effizienten Approximationsalgorithmus, der höchstens $2d \cdot opt$ Behälter öffnet. Der Algorithmus sollte möglichst schnell sein.

Aufgabe 44
Wir betrachten eine Variante von BIN PACKING. Für n Objekte mit Gewichten $g_1, \ldots, g_n \in [0, 1]$ und eine Zahl m wird nach der maximalen Anzahl von Objekten gefragt, die in m Behälter (jeweils mit Kapazität 1) gepackt werden können.

(a) **Zeige**, dass für $m = \lceil \sum_{i=1}^{n} g_i \rceil$ eine entsprechende Version vom First-Fit Algorithmus mindestens $n/2$ der n Objekte einpacken wird.

(b) **Zeige**, dass diese Variante kein volles polynomielles Approximationsschema besitzt, wenn $\mathcal{P} \neq \mathcal{NP}$. *Hinweis*: Erstes Übungsblatt.

3.2.2 Ein polynomielles Approximationsschema

Wir beobachten zuerst, dass BIN PACKING exakt durch „effiziente“ Algorithmen lösbar ist, wenn die Objekte nur zu wenigen verschiedenen Gewichtsklassen gehören. Als Vorbereitung benötigen wir die folgende Beobachtung.

Beobachtung 3.1 *Die Anzahl aller Vektoren* $(a_1, \ldots, a_g)$ *mit*

- $a_1, \ldots, a_g \in \mathbb{N}$ *und*
- $\sum_{i=1}^{g} a_i \leq r$

ist höchstens $\binom{g+r}{g}$.

Beweis: Die Anzahl dieser Vektoren ist beschränkt durch die Anzahl binärer Worte der Länge höchstens $g+r-1$ mit genau $g-1$ Nullen, denn wir können die Nullen als Trennzeichen zwischen den unären Darstellungen der a_i auffassen. Es gibt genau $\binom{R}{g-1}$ binäre Worte der Länge R mit $g-1$ Nullen. Also folgt

$$\sum_{R=g-1}^{r+g-1} \binom{R}{G-1} = \binom{r+g}{g}$$

und das war zu zeigen. □

Lemma 3.9 *Es gelte $\varepsilon > 0$ und $G \in \mathbb{N}$. n Objekte mit Mindestgewicht ε und höchstens G verschiedenen Gewichten seien in Behälter zu verteilen. Dann kann eine optimale Verteilung in Zeit proportional zu*

$$\binom{n+B}{B} = \text{poly}(n)$$

mit $B = \binom{\lfloor \frac{1}{\varepsilon} \rfloor + G}{G}$ bestimmt werden.

Beweis: Ein Behälter-Typ wird durch den Vektor $(a_i : 1 \leq i \leq G)$ beschrieben, wobei a_i die Anzahl der Objekte vom $i$ten Gewicht ist. Ein Behälter kann höchstens $\lfloor \frac{1}{\varepsilon} \rfloor$ Objekte aufnehmen. Folglich gibt es nach der obigen Beobachtung, wenn wir $g = G$ und $r = \lfloor \frac{1}{\varepsilon} \rfloor$ setzen, höchstens $B = \binom{\lfloor \frac{1}{\varepsilon} \rfloor + G}{G}$ verschiedene Behälter-Typen.
Wir stellen eine vollständige Bepackung durch den Vektor $(b_1, \ldots, b_B)$ dar, wobei b_i die Anzahl der Bins vom Typ i ist. Die Anzahl der verschiedenen Bepackungen ist also durch die Anzahl der Vektoren $(b_1, \ldots, b_B)$ mit $b_1, \ldots, b_B \in \mathbb{N}$ und $\sum_{i=1} b_i \leq n$ beschränkt. Deren Anzahl ist aber höchstens $\binom{n+B}{B}$.
Wir können also eine beste Bepackung bestimmen, indem wir alle verschiedenen Bepackungstypen aufzählen. Die Behauptung folgt, da $\binom{n+B}{B}$ ein Polynom in n ist. □

Wir nehmen weiterhin an, dass alle Objekte das Mindestgewicht ε besitzen. Als ersten Schritt unseres Algorithmus sortieren wir die n Objekte nach aufsteigendem Gewicht und zerlegen die sortierte Reihenfolge in $G = \lceil \frac{1}{\varepsilon^2} \rceil$ Intervalle mit jeweils höchstens $E = \lfloor n \cdot \varepsilon^2 \rfloor$ Elementen. Um Lemma 3.9 anzuwenden, runden wir die Gewichte eines jeden Intervalls auf und setzen sie auf das größte Gewicht ihres Intervalls. Wir bezeichnen die neue Instanz mit „`auf`“ und nennen die durch Abrundung auf das kleinste Gewicht eines jeden Intervalls erhaltene Instanz „`ab`“. Wenn $\text{bp}(x)$ die minimale Anzahl benötigter Behälter für Instanz x bezeichnet und wenn y die vorliegende Instanz ist, dann erhalten wir offensichtlich

$$\text{bp}(\texttt{ab}) \leq \text{bp}(y) \leq \text{bp}(\texttt{auf}).$$

Andererseits wird Instanz `auf` höchstens E mehr Behälter als Instanz `ab` besitzen, denn bis auf die E Gewichte des letzten Intervalls werden die Gewichte des $i$ten Intervalls von `auf` durch die Gewichte des $(i+1)$sten Intervalls von `ab` dominiert. Also ist

$$\text{bp}(\texttt{auf}) \leq \text{bp}(\texttt{ab}) + E \leq \text{bp}(y) + E.$$

Es ist aber $\mathrm{bp}(y) \geq n \cdot \varepsilon$, denn jedes Objekt hat Mindestgewicht ε. Deshalb folgt $E = \lfloor n \cdot \varepsilon^2 \rfloor \leq \varepsilon \cdot \mathrm{bp}(y)$ sowie

$$\mathrm{bp}(\texttt{auf}) \ \leq \ (1+\varepsilon) \cdot \mathrm{bp}(y).$$

Wir können aber eine optimale Verteilung für die Instanz `auf` mit Lemma 3.9 exakt berechnen, da nur G verschiedene Gewichtsklassen vorliegen. Also bleibt die Frage, wie die leichten Objekte (mit Gewicht $< \varepsilon$) zu verteilen sind. Wir nehmen an, dass alle nicht zu leichten Objekte bereits verteilt wurden und verteilen dann die leichten Objekte nach dem First-Fit Prinzip.
Wenn First-Fit keine neuen Behälter öffnet, dann haben wir die Approximationskonstante $1+\varepsilon$ erreicht. Ansonsten sei B die Gesamtanzahl geöffneter Behälter, wobei bis auf den letzten Behälter alle Behälter mit Gewicht mindestens $1-\varepsilon$ beladen sind. Das bedeutet, dass das Gesamtgewicht aller Objekte mindestens $(B-1) \cdot (1-\varepsilon)$ beträgt und folglich ist $\mathrm{bp}(y) \geq (B-1) \cdot (1-\varepsilon)$, bzw. es ist

$$B \leq \frac{\mathrm{bp}(y)}{1-\varepsilon} + 1 \leq (1+2\varepsilon) \cdot \mathrm{bp}(y) + 1,$$

wobei wir $\varepsilon \leq \frac{1}{2}$ benutzt haben.

Satz 3.10 *Sei $\frac{1}{2} \geq \varepsilon > 0$ vorgegeben. Dann gibt es einen Algorithmus A_ε, der höchstens $(1+2\varepsilon) \cdot \texttt{opt} + 1$ Behälter benötigt. Die Laufzeit von A_ε ist polynomiell in der Anzahl der Objekte; allerdings taucht eine Funktion von $1/\varepsilon$ in den Potenzen des Polynoms auf.*

Bemerkung 3.2 Leider ist dieser Algorithmus nicht praktikabel, da die vollständige Aufzählung in Lemma 3.9 zu Polynomen von utopisch hohem Grad führt. Damit besagt Satz 3.10 nicht mehr, als dass die Methode der Gewichtsaufrundung sowie die Sonderbehandlung leichter Objekte vielversprechend ist.
Beachte weiterhin, dass wir kein volles Approximationsschema erhalten haben. Dies ist auch nicht möglich, da BIN PACKING in $\mathcal{NPO} - \mathcal{PB}$ liegt und da seine Sprachenversion

> Gibt es eine Bepackung mit höchstens k Behältern?

$\mathcal{NP}$-vollständig ist.

Tatsächlich wird in [KK82] ein effizienter Algorithmus entwickelt, der höchstens $\texttt{opt} + O(\log_2^2(N))$ Behälter öffnet: N ist die Summe aller Gewichte.

Offenes Problem 3
Bis heute konnte die Existenz von effizienten Algorithmen nicht ausgeschlossen werden, die höchstens $\texttt{opt} + 1$ Behälter öffnen.

3.3 MINIMUM MAKESPAN SCHEDULING

In MINIMUM MAKESPAN SCHEDULING sind n Aufgaben $A_1, \ldots, A_n$ gegeben, wobei Aufgabe A_i die Laufzeit t_i besitzt. Die Aufgaben sind so auf m Maschinen auszuführen, dass der „Makespan“, also die für die Abarbeitung aller Aufgaben anfallende Bearbeitungszeit, minimal ist. Mit anderen Worten, wenn

$$I_j = \{i \mid A_i \text{ wird auf Maschine } j \text{ ausgeführt}\}$$

die Menge der auf Maschine j auszuführenden Aufgaben ist, dann ist

$$\max_{1 \leq j \leq m} \left\{ \sum\nolimits_{i \in I_j} t_i \right\}$$

der Makespan.
Wir zeigen, dass die Sprachenversion

> Gibt es eine Zuweisung von Aufgaben an Maschinen, so dass der Makespan höchstens T beträgt?

bereits für $m = 2$ Maschinen $\mathcal{NP}$-vollständig ist. Wir zeigen, dass PARTITION auf die Sprachenversion von Minimum Makespan reduziert werden kann: Wenn die Zahlen $x_1 \ldots, x_n$ Eingabe für PARTITION sind, dann erzeugen wir in MINIMUM MAKESPAN SCHEDULING n Aufgaben mit den Laufzeiten $x_1 \ldots, x_n$. Die Reduktion ist gelungen, denn es gilt

$$(x_1, \ldots, x_n) \in \text{PARTITION} \quad \Leftrightarrow \quad \text{Der Makespan beträgt } \frac{1}{2} \cdot \sum_{i=1}^{n} x_i.$$

3.3.1 Greedy Algorithmen

Zuerst werden wir den Approximationsfaktor 2 für Minimum Makespan mit List Scheduling, einem On-line Algorithmus erreichen. Die Strategie ist denkbar einfach: Führe die Aufgaben in irgendeiner Reihenfolge aus, wobei die aktuelle Aufgabe auf der Maschine mit der bisher geringsten Last ausgeführt wird. (Wir sprechen von einem On-line Algorithmus, da der Algorithmus die aktuellen Aufgabe ausführt, ohne zukünftig auszuführenden Aufgaben zu betrachten.)

Lemma 3.11 *List Scheduling besitzt den Approximationsfaktor* 2.

Beweis: Für eine gegebene Instanz sei Maschine i die am schwersten „beladene“ Maschine, die also als letzte Maschine noch rechnet. Falls i nur eine Aufgabe ausführt, ist der On-line Algorithmus offensichtlich optimal. Angenommen, i führt mindestens zwei Aufgaben aus. Sei A_j die als letzte von i ausgeführte Aufgabe. Wenn T die Gesamtlaufzeit von i ist, dann waren zum Zeitpunkt $T - t_j$ alle anderen Maschinen beschäftigt. Folglich ist

$$\sum_{k=1}^{n} t_k \geq (m-1) \cdot (T - t_j) + T = mT - (m-1) \cdot t_j \geq mT - m \cdot t_j$$

und damit ist

$$T \leq \frac{1}{m} \cdot \sum_{k=1}^{n} t_k \quad + t_j \leq 2 \cdot \max\left\{\frac{1}{m} \cdot \sum_{k=1}^{n} t_k,\ t_j\right\}.$$

Die Behauptung folgt mit der nächsten Aufgabe. □

Aufgabe 45
Für jede Aufgabe A_j ist der Makespan mindestens $\max\left\{\frac{1}{m} \cdot \sum_{k=1}^{n} t_k,\ t_j\right\}$.

Unsere Analyse kann nicht verbessert werden wie das folgende Beispiel zeigt. $m \cdot (m-1)$ Aufgaben der Länge 1 sowie eine Aufgabe der Länge m sind gegeben. Offensichtlich lässt sich der Makespan m erreichen, wenn eine Maschine für die lange Aufgabe reserviert wird. Werden andererseits zuerst die kurzen Aufgaben gleichmäßig über die m Maschinen verteilt und folgt dann die lange Aufgabe, so erhalten wir den Makespan $2 \cdot m - 1$.
Wir haben unsere Strategie durch die zuletzt präsentierte lange Aufgabe genarrt. Wir ändern deshalb unsere Strategie und fordern, dass die Aufgaben gemäß absteigender Bearbeitungszeit präsentiert werden, also zuerst die langen und dann die kürzer werdenden Aufgaben. Wir erhalten das LPT-Verfahren (largest processing time first) mit einem wesentlich besseren Approximationsfaktor.

Satz 3.12 *Der Approximationsfaktor sinkt auf höchstens $\frac{4}{3}$, wenn Aufgaben gemäß fallender Bearbeitungszeit präsentiert werden.*

Beweis: Wir zeigen ein schwächeres Ergebnis, nämlich dass der Approximationsfaktor auf 3/2 sinkt. Wir wissen, dass die Aufgaben nach fallenden Bearbeitungszeiten angeordnet sind, es gilt also $t_1 \geq t_2 \geq \cdots t_m \geq t_{m+1} \geq \cdots \geq t_n$. Sei $\texttt{opt}$ der optimale Makespan. Die folgende Beobachtung ist zentral.

Lemma 3.13 $\texttt{opt} \geq 2t_{m+1}$.

Beweis: Wir betrachten nur die ersten $m+1$ Aufgaben, die jeweils mindestens die Bearbeitungszeit t_{m+1} besitzen. Mindestens eine Maschine wird zwei Aufgaben erhalten und der optimale Makespan ist somit mindestens $2t_{m+1}$. □
Der Rest des Arguments verläuft parallel zum Beweis von Satz 3.11. Wir betrachten die Maschine i mit größter Last, die zuletzt die Aufgabe j ausführen möge. Es ist $t_j \leq t_{m+1}$, denn unsere Maschine arbeitet die Aufgaben nach fallender Bearbeitungszeit ab. Aber $2t_{m+1} \leq \texttt{opt}$ folgt mit Lemma 3.13 und deshalb ist $t_j \leq t_{m+1} \leq \frac{\texttt{opt}}{2}$.
Sei T der Makespan unserer neuen Strategie. Mit dem Argument von Satz 3.11 erhalten wir wiederum

$$T - t_j \leq \frac{1}{m} \cdot \sum_{k=1}^{n} t_k \leq \texttt{opt} \text{ und } t_j \leq \frac{\texttt{opt}}{2}.$$

Folglich ist $T = T - t_j + t_j \leq \texttt{opt} + \frac{\texttt{opt}}{2} = \frac{3\texttt{opt}}{2}$ und das war zu zeigen. □

Aufgabe 46
Beweise Satz 3.12 für $m = 3$.
Hinweis: Beim Beweis des Faktors 2 für den Greedy Algorithmus wurde die Länge des Schedules mit der durchschnittlichen Arbeit einer Maschine verglichen. Dies wird für $\frac{4}{3}$ nicht gelingen können, wie der

einfache Fall von 4 Jobs gleicher Länge belegt. Der Algorithmus ist sogar optimal, aber nicht innerhalb von $\frac{4}{3}$ bezüglich der durchschnittlichen Arbeitslast.

Aufgabe 47
Wir betrachten den Fall von $m = 2$ Maschinen.

(a) Zeige, dass jede on-line Strategie mindestens den Wettbewerbsfaktor 3/2 besitzt.

(b) Entwickle eine randomisierte on-line Strategie mit einem Wettbewerbsfaktor von höchstens 4/3.

Offenes Problem 4
Bestimme den Wettbewerbsfaktor einer besten on-line Strategie. Die gegenwärtig beste On-line strategie besitzt den Wettbewerbsfaktor 1.923. Der Wettbewerbsfaktor einer besten randomisierten on-line Strategie ist ebenfalls nicht bekannt.

3.3.2 Ein polynomielles Approximationsschema

Wir geben eine verbesserte Lösung mit Hilfe der dynamischen Programmierung an. Wir beginnen mit einer verwandten Fragestellung: Angenommen, es gibt nur wenige, nämlich r verschiedene Laufzeiten, und wir möchten wissen, ob Makespan K mit m Maschinen erreichbar ist. Dazu betrachten wir ein BIN PACKING Probleme mit Bins (oder Behältern) der Kapazität K. n Objekte, mit Gewichten aus der Menge $\{t_1, \ldots, t_r\}$ sind auf möglichst wenige Bins zu verteilen, so dass kein Bin ein Gesamtgewicht größer als K tragen muss. Insbesondere möge es n_i Objekte mit Gewicht t_i geben, es gelte also $n = \sum_{i=1}^{r} n_i$. Schließlich sei

$$\text{Bins}_K(n_1, \ldots, n_r)$$

die minimale Anzahl von Bins mit Kapazität K, so dass alle Objekte verstaut werden können. Wir berechnen $\text{Bins}_K(n_1, \ldots, n_r)$ mit dynamischer Programmierung.

Algorithmus 3.14 **Eine exakte Lösung für BIN PACKING mit wenigen verschiedenen Gewichten.**

(1) Die Eingabe besteht aus dem Vektor $(n_1, \ldots, n_r)$. Die $n = \sum_{i=1}^{r} n_i$ Objekte sind in möglichst wenige Bins zu verteilen, wobei jeweils n_i Objekte das gleiche Gewicht t_i besitzen.

(2) Bestimme die Menge

$$\text{passt} = \{(m_1, \ldots, m_r) \mid 0 \leq m_i \leq n_i \text{ für alle } i \text{ und } \text{Bins}_K(m_1, \ldots, m_r) = 1\}.$$

Kommentar: Wir betrachten alle möglichen Kombinationen $(m_1, \ldots, m_r)$ und überprüfen, ob ein einziges Bin ausreicht. Laufzeit proportional zu $n_1 \cdots n_r \leq n^r$ ist ausreichend.

(3) Wir berechnen $\text{Bins}_K(m_1, \ldots, m_r)$ für alle Kombinationen $(m_1, \ldots, m_r)$ mit $0 \leq m_i \leq n_i$ für $i = 1, \ldots, r$. Dies geschieht mit dynamischer Programmierung, indem wir

$$\text{Bins}_K(m_1, \ldots, m_r) = 1 + \min_{i \in \text{passt}} \text{Bins}_K(m_1 - i_1, \ldots, m_r - i_r)$$

ausnutzen.

Kommentar: Die Bestimmung von $\text{Bins}_K(m_1, \ldots, m_r)$ gelingt in Zeit $O(n^r)$. Da die Berechnung für jeden der höchstens n^r Vektoren m auszuführen ist, wird also Zeit $O(n^{2 \cdot r})$ benötigt.

Lemma 3.15 *$Bins_K(n_1, \ldots, n_r)$ kann in Zeit $O(n^{2 \cdot r})$ bestimmt werden, wobei $n = \sum_{i=1}^{r} n_i$.*

Wir können also überprüfen, ob Makespan K für m Maschinen erreichbar ist, solange die Anzahl r der verschiedenen Laufzeiten nicht zu groß ist. Deshalb werden wir lange Laufzeiten in wenige verschiedene Laufzeiten runden und einen minimalen, mit m Maschinen erreichbaren Makespan K mit Binärsuche bestimmen. „Kurze“ Aufgaben fügen wir dann ganz zuletzt mit List Scheduling ein.

Algorithmus 3.16 **Ein polynomielles Approximationsschema**

(1) Gegeben sind n Aufgaben $A_1, \ldots, A_n$ mit den Laufzeiten $t_1, \ldots, t_n$. Der Approximationsparameter ε sei ebenfalls gegeben.

(2) Bestimme

$$M = \max \left\{ \frac{1}{m} \cdot \sum_{k=1}^{n} t_k,\ \max_k \{t_k\} \right\}.$$

Kommentar: Die optimale Bearbeitungszeit liegt im Intervall $[M, 2 \cdot M]$.

(3) Führe eine binäre Suche im Interval $[M, 2 \cdot M]$ mit den anfänglichen Intervallgrenzen $L = M$ und $R = 3 \cdot M/2$ durch. Das Intervall mit den Grenzen $R = 3 \cdot M/2$ und $R' = 2 \cdot M$ ist das Zwillingsintervall.

(3a) Eine Aufgabe A_i heißt kurz, falls $t_i \leq R \cdot \varepsilon$. Für eine lange Aufgabe A_j definieren wir die gerundete Laufzeit als

$$t_j^* = R \cdot \varepsilon \cdot (1 + \varepsilon)^k,$$

falls $t_j \in [R \cdot \varepsilon \cdot (1 + \varepsilon)^k, R \cdot \varepsilon \cdot (1 + \varepsilon)^{k+1}[$.

(3b) Führe die langen Aufgaben mit Algorithmus 3.14 aus, wobei wir die Bin-Kapazität R annehmen.

Kommentar: Es gibt $r = \log_{1+\varepsilon} \frac{1}{\varepsilon}$ verschiedene lange Laufzeiten, denn $R \cdot \varepsilon \cdot (1 + \varepsilon)^r = R$. Die Laufzeit von Algorithmus 3.14 ist damit $O(n^{2 \cdot r})$.

Angenommen, die optimale Ausführung der langen Aufgaben erfordert b Bins der Kapazität R. Wenn $b \leq m$, dann wird die Ausführung der ungerundeten Aufgaben höchstens Kapazität $R \cdot (1 + \varepsilon)$ für m Maschinen erfordern.

(3c) Erhöhe die Bin-Kapazität von R auf $R \cdot (1+\varepsilon)$ und nimm die Rundungen zurück. Die kurzen Aufgaben werden jetzt abgearbeitet, wobei Aufgabe A_i in ein beliebiges Bin mit Restkapazität mindestens t_i eingefügt wird; existiert kein solches Bin, wird ein neues Bin aufgemacht.

Kommentar: Schaffen wir nach Aufnahme der kurzen Aufgaben keine Bepackung mit $\leq m$ Bins der Kapazität $R \cdot (1+\varepsilon)$, dann sind m Bins mit Last größer als R gefüllt.: Wir haben im falschen Intervall gesucht und sollten stattdessen das „Zwillingsintervall“ versuchen.

(3d) Wenn die Anzahl b der benötigten Bins höchstens m ist, dann setze die Binärsuche mit der linken Intervallhälfte von $[L, R]$ fort –wir versuchen, den Makespan zu verkleinern– und ansonsten mit der linken Intervallhälfte des Zwillingsintervalls $[R, R']$. Halte, wenn die Länge des neuen Intervalls höchstens $M \cdot \varepsilon$ ist und gehe ansonsten zu Schritt (3a).

Kommentar: Die Binärsuche wird höchstens $\lceil \log_2 \frac{1}{\varepsilon} \rceil$-mal iteriert. Das Stoppen der Binärsuche verursacht einen weiteren additiven Anstieg des Makespans um den Beitrag $M \cdot \varepsilon \leq \texttt{opt} \cdot \varepsilon$.

(4) Bestimme die Aufgabenverteilung der gefundenen Lösung.

Satz 3.17 *Das Minimum Makespan Problem für n Aufgaben kann in Zeit $O(n^{2r} \cdot \lceil \log_2 \frac{1}{\varepsilon} \rceil)$ gelöst werden, wobei $r = \log_{1+\varepsilon} \frac{1}{\varepsilon}$. Die gefundene Lösung besitzt einen Makespan von höchstens $(1+2\varepsilon) \cdot \texttt{opt}$.*

Wir haben „nur“ ein polynomielles Approximationsschema erhalten. Wir zeigen jetzt, dass das Minimum Makespan Problem kein vollständiges polynomielles Approximationsschema besitzt, sofern $\mathcal{P} \neq \mathcal{NP}$ gilt. Dazu betrachten wir das $\mathcal{NP}$-vollständige Problem 3-PARTITION:

> Für Zahlen $a_1, \ldots, a_{3n}$ (mit $\frac{B}{4} < a_i < \frac{B}{2}$) und eine Zahl $B \leq n$ stelle fest, ob die $3n$ Zahlen so in disjunkte Gruppen aufgeteilt werden können, dass jede Gruppe die Summe B besitzt.

(Ist eine solche Aufteilung möglich, dann besteht jede Gruppe aus genau drei Zahlen.) Wir reduzieren 3-PARTITION auf MINIMUM MAKESPAN SCHEDULING, in dem wir $3n$ Aufgaben mit Laufzeiten $t_i = a_i$ festlegen. Diese Aufgaben sind auf n Maschinen auszuführen. Offensichtlich gilt

$$(a_1, \ldots, a_n, B) \in \text{3-PARTITION} \quad \Leftrightarrow \quad \text{der Minimum Makespan ist höchstens B.}$$

Wenn aber MINIMUM MAKESPAN SCHEDULING ein volles polynomielles Approximationsschema $\mathcal{A}$ besitzt, dann kann 3-PARTITION mit Hilfe der Reduktion und $\mathcal{A}$ effizient gelöst werden.

Aufgabe 48

Wir betrachten dass Minimum Makespan Problem für n Jobs mit Laufzeiten $t_1, \ldots, t_n$, die auf m Maschinen zu verteilen sind.

1. Sei k eine Konstante. **Zeige**, dass das Problem für $m = 2$ und Joblaufzeiten, die natürliche Zahlen aus $\{1, \ldots, n^k\}$ sind, in polynomieller Zeit exakt lösbar ist.
2. **Zeige**, dass für $m = 2$ und beliebige Joblängen ein volles, polynomielles Approximationsschema existiert.

Aufgabe 49
Zeige, dass das Problem der letzten Aufgabe für allgemeines m kein volles, polynomielles Approximationsschema besitzt, falls $\mathcal{P} \neq \mathcal{NP}$. Es darf benutzt werden, dass das folgende Problem $\mathcal{NP}$-vollständig ist:

> Gegeben eine Menge S von n natürlichen Zahlen von 1 bis n und ein Parameter k. Frage: Gibt es k disjunkte Teilmengen $I_1, \ldots, I_k$, so dass gilt:
>
> $$\sum_{x \in I_i} x = \frac{1}{k} \cdot \sum_{x \in S} x$$
>
> für jedes i gilt.

3.3.3 MINIMUM MAKESPAN mit Prioritäten

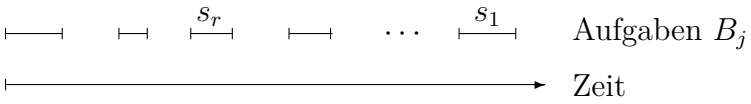
Wir betrachten jetzt das Minimum Makespan Problem mit Prioritäten. Zusätzlich ist jetzt ein gerichteter azyklischer Graph $P = (\{A_1, \ldots, A_n\}, E)$ gegeben, der die Ausführung der Aufgabe A_i vor Aufgabe A_j vorschreibt, falls es einen gerichteten Weg in P von A_i nach A_j gibt. Die Aufgaben sind wiederum so auf die m Prozessoren zu verteilen, dass alle Aufgaben zum frühest möglichen Zeitpunkt ausgeführt werden.
Wir modifizieren unsere Lösung für das Minimum Makespan Problem wie folgt: Führe eine beliebige ausführbare Aufgabe auf einer frei gewordenen Maschine aus.

Satz 3.18 *Das Minimum Makespan Problem mit Prioritäten kann effizient mit Faktor* 2 *approximiert werden.*

Beweis: Die Aufgaben $A_1, \ldots, A_n$ mit Laufzeiten $t_1, \ldots, t_n$ seien gemäß dem Prioritätsgraph G auf m Maschinen auszuführen. Sei B_1 die Aufgabe, die bei Ausführung unseres Algorithmus zuletzt terminiert und sei B_2 die Aufgabe unter den Vorgängern von B_1, die zuletzt terminiert usw. Wenn s_i die Laufzeit von B_i bezeichnet, dann gilt

$$\sum_i s_i \leq \texttt{opt},$$

denn eine überlappende Ausführung zweier Aufgaben B_i, B_j mit $i \neq j$ ist unmöglich, da eine Aufgabe die andere verhindert.



Die obige Abbildung stellt das Zeitverhalten der Aufgaben $B_1, B_2, \ldots$ dar. Für die „Lücke“ nach s_r gilt, dass die Abarbeitung von B_{r-1} verzögert wird, da alle Maschinen arbeiten. In den Zwischenräumen sind somit stets alle Maschinen aktiv, so dass die optimale Zeit `opt` mindestens so groß wie die Gesamtzeit aller Zwischenräume ist. Folglich ist der vom Algorithmus berechnete Makespan durch $2 \cdot \texttt{opt}$ beschränkt. □

3.4 TSP und Dynamische Programmierung

Wir entwerfen zuerst einen exakten Algorithmus, der eine kürzeste Rundreise bestimmt und untersuchen dann das euklidische TSP, dass sich für die Approximation als wesentlich einfacher herausstellen wird.

3.4.1 Ein exakter Algorithmus

Ein trivialer Algorithmus zählt alle $n!$ möglichen Rundreisen auf und gibt eine Rundreise minimaler Länge aus. Dieses naive Verfahren hat die Laufzeit $O(n \cdot n!) = O(2^{n \log_2 n})$. Mit dynamischem Programmieren gelingt eine Lösung in Zeit $O(n^2 \cdot 2^n)$. Wir wählen im Folgenden stets v_1 als Startort.

Algorithmus 3.19 **Dynamisches Programmieren für das allgemeine TSP**

(1) Definiere für jede Teilmenge $S \subseteq \{v_2, \ldots, v_n\}$ und jeden Ort $v_m \notin S$ das Teilproblem Rundreise(S, v_m). In diesem Teilproblem ist die minimale Länge einer Reise von v_1 nach v_m zu bestimmen, wobei die Reise die Orte in S besuchen muss.

(2) Setze Rundreise$(\emptyset, v_i) = d(v_1, v_i)$ für $i = 2, \ldots, n$.

(3) Die Teilprobleme können gemäß der „Rekursionsgleichung“

$$\text{Rundreise}(S, v_m) := \min_{w \in S}\{\text{Rundreise}\,(S \setminus \{w\}, w) + d(w, v_m)\}$$

mit dynamischem Programmieren gelöst werden.

(4) Die kürzeste Rundreise hat dann die Länge Rundreise$(\{v_2, \ldots, v_n\}, v_1)$.

Wir haben höchstens $n \cdot 2^n$ Teilprobleme –ein Teilproblem für jeden der n Orte und jede der 2^n Teilmengen der n Orte–. Da die Lösung eines Teilproblems jeweils in Zeit $O(n)$ gelingt, ist die Gesamtlaufzeit wie versprochen durch $O(n^2 \cdot 2^n)$ beschränkt. Für $n = 18$ stehen den 6 Billiarden(!) Schritten der trivialen Lösung 21 Millionen Schritte der dynamischen Programmierlösung gegenüber. Während das definitive Aus der trivialen Lösung erfolgt ist, befinden wir uns mit unserem neuen Verfahren weiter im grünen Bereich. Wir werden aber später mit Branch & Cut Verfahren bessere Methoden kennenlernen.

Satz 3.20 *Das allgemeine Traveling Salesman Problem für n Orte kann in Zeit $O(n^2 \cdot 2^n)$ gelöst werden. Allerdings wird Speicherplatz $\Omega(n \cdot 2^n)$ benötigt.*

3.4.2 Das Euklidsche TSP

Im euklidschen TSP sind n Punkte im $\mathbb{R}^d$ gegeben, die wir mit einer möglichst kurzen Rundreise alle besuchen müssen; die Distanz zwischen zwei Punkten ist ihre euklidsche Distanz. Während das unbeschränkte TSP überhaupt keine sinnvollen Approximationen

und das metrische TSP nur effiziente Approximationen mit konstantem Faktor zulässt, zeigen wir jetzt, dass das euklidsche TSP sogar ein polynomielles Approximationsschema besitzt.
Wir beschränken uns im Folgenden auf den 2-dimensionalen Fall und beschreiben den Algorithmus von Arora [Aro2][2]. Die Verallgemeinerung auf den d-dimensionalen Fall ist dann nicht schwer, allerdings wächst die Laufzeit doppelt exponentiell in d. Dieses Wachstum ist nicht überraschend, da auch gezeigt werden kann, dass das euklidsche Traveling Salesman Problem im $\mathbb{R}^{O(\log_2 n)}$ $\mathcal{APX}$-vollständig ist [Tre].

Schritt 1: Wir „verwackeln“ die Eingabe, um eine normalisierte Eingabe zu erhalten.
Das kleinste Quadrat Q, das alle Punkte einschließt, habe die Seitenlänge L_0. Als kleinstes Quadrat besitzt Q zwei Punkte auf gegenüberliegenden Seiten: Eine kürzeste Rundreise wird also mindestens die Distanz vom ersten zum zweiten Punkt und zurück durchlaufen, da ja die beiden Punkte besucht werden müssen. Wenn `opt` die Länge einer kürzesten Rundreise ist, folgt also

$$2L_0 \leq \texttt{opt}. \tag{3.2}$$

Wir legen ein Gitter mit Seitenlänge $\frac{\varepsilon \cdot L_0}{8n}$ über Q und verschieben jeden der n Punkte zu einem nächstliegenden Gitterpunkt: Eine solche Verschiebung wird die Länge einer beliebigen Rundreise um höchstens den Betrag

$$2n \cdot \frac{\varepsilon L_0}{8n} \leq \varepsilon \cdot \texttt{opt}/8$$

ansteigen lassen.
Zuletzt teilen wir jede Komponente durch $\varepsilon L_0/(64n)$. Als Konsequenz erhalten alle Punkte, die sich ja jetzt auf den Gitterpunkten befinden, ganzzahlige Koordinaten und alle auf verschiedenen Gitterpunkten sitzende Punkte haben den Mindestabstand 8. Außerdem hat das alle Punkte einschließende Quadrat Q die Seitenlänge

$$L = L_0/(\varepsilon L_0/(64n)) = (64n)/\varepsilon = O(n).$$

Schritt 2: Die Konstruktion eines Quadtrees.
Nach diesen Vorbereitungen können wir jetzt unseren Ansatz besprechen. Wir zerlegen Q in seine vier Teilquadrate der jeweiligen Länge $L/2$. Dann liegt es nahe, kurze Rundreisen für jedes Teilquadrat rekursiv zu bestimmen und die Rundreisen dann zusammenzubauen. Betrachten wir aber eine optimale Rundreise R: Dann zerfällt R in Rundreisen $R_1, R_2, \ldots$, wobei wir eine neue Teilrundreise R_{i+1} genau dann beginnen, wenn die Teilrundreise R_i gerade ihr Teilquadrat verlassen hat. Insbesondere stößt R möglicherweise mehrmals in dasselbe Teilquadrat und besucht natürlich jeweils verschiedene Punktmengen. Eine erfolgreiche Implementierung dieses rekursiver Ansatzes ist also alles Andere als klar.

[2] Arora hat im Jahr 2010 den Gödel Preis für seinen Algorithmus erhalten.

Zuerst setzen wir die Aufteilung des Quadrats Q in vier Teilquadrate rekursiv fort. Diese rekursive Aufteilung können wir durch einen Quadtree T_Q, einem 4-ären Baum, modellieren: Wir markieren die Wurzel von T_Q mit Q, die vier Kinder der Wurzel durch die vier Teilquadrate von Q und setzen diese Markierung solange rekursiv fort, bis jeder Knoten mit einem Teilquadrat markiert ist, dass nur einen Punkt enthält. (Blätter mit leerem Teilquadrat treten natürlich auf, aber nur dann, wenn sie einen Geschwisterknoten mit einem nichtleeren Geschwisterquadrat besitzen.)
Angenommen, v ist ein Knoten der Tiefe i und v ist mit dem Teilquadrat Q' markiert. In der Aufteilung von Q' in vier Teilquadrate werden zwei Trennlinien benutzt, die wir als Trennlinien der Schicht i bezeichnen. Auf jeder Trennlinie verteilen wir m „Türen“ im gleichen Abstand, wobei m eine Zweierpotenz mit

$$m \in [\frac{\log_2 n}{\varepsilon}, 2\frac{\log_2 n}{\varepsilon}]$$

ist. Zusätzlich setzen wir eine Tür auf jede Ecke von Q'. Was ist die Rolle der Türen? Wir zwingen Rundreisen Trennlinien nur durch Türen zu durchqueren und beschneiden damit die Anzahl der möglichen „legalen“ Rundreisen drastisch.

Schritt 3: Türen und legale Rundreisen.
Wir fordern, dass unser Algorithmus eine **legale** Rundreise konstruiert: Eine Rundreise ist genau dann legal, wenn sie ein Quadrat des Baums T_Q nur über eine Tür betritt und verläßt. Einerseits werden wir versuchen, mit möglichst wenigen Türen auszukommen, da die Anzahl der zu evaluierenden Rundreisen explosionsartig mit der Anzahl der Türen wächst. Andererseits werden wir in eine Mindestzahl von Türen gezwungen, da ansonsten nur Rundreisen zu großer Länge berechnet werden können. Auf den ersten Blick erstaunt es, dass wir genau so viele Türen auf den langen Trennlinien der niedrigen Schichten einsetzen wie auf den kurzen Trennlinien der hohen Schichten. Man beachte aber, dass es sehr viele kurze Trennlinien der hohen Schichten gibt und man wird erwarten, dass eine Rundreise diese vielen kurzen Trennlinien sehr viel häufiger trifft als die wenigen langen.
Wir verschieben die Beschreibung der algorithmischen Idee und fragen uns stattdessen ob unser Ansatz überhaupt erfolgreich sein kann. Insbesondere, wie lang ist denn eine kürzeste legale Rundreise im Vergleich zur Länge `opt` einer kürzesten Rundreise? Gibt es auch kurze legale Rundreisen, die Trennlinien, also die Kanten der Teilquadrate von T_Q nicht zu häufig durchstoßen? Solche Rundreisen lassen sich hoffentlich viel einfacher finden!

Das **Struktur-Theorem**, der *erste Versuch*: Es gibt stets eine legale Rundreise der Länge höchstens $(1+\varepsilon) \cdot \texttt{opt}$, so dass jede Kante eines Teilquadrats des Quadtrees T_Q höchstens $1/\varepsilon$ mal durchquert wird.

Stimmt die Aussage des Struktur-Theorems? Wir versuchen, eine hoffentlich kurze legale Rundreise $R_{\texttt{legal}}$ aus einer optimalen Rundreise $R_{\texttt{opt}}$ zu konstruieren. Dazu betrachten wir nacheinander alle Kanten e von $R_{\texttt{opt}}$ und ersetzen e durch einen nur durch Türen verlaufenden Weg W_e. Da wir nur durch Türen laufen dürfen, verschieben wir jede Kreuzung von e mit einer Trennlinie zur nächstliegenden Tür der Trennlinie; diese Verschiebung bedeutet

bei einer Trennlinie der Tiefe i einen Längenzuwachs von höchstens

$$\text{umweg}_i := \frac{1}{m}\frac{L}{2^i}.$$

An dieser Stelle haben wir mit unserem Ansatz aber leider schon verloren: Es ist durchaus möglich, dass $R_{\texttt{opt}}$ die vertikale Trennlinie der Schicht 0 in jeder Tür trifft. Der Längenzuwachs ist in diesem Fall mindestens L. Wir wissen aber nur, dass $R_{\texttt{opt}}$ mindestens die Länge $2L$ hat und können deshalb nicht ausschließen, dass wir nur eine Approximation innerhalb konstanter Faktoren erhalten und die Behauptung des Struktur-Theorems ist falsch, bzw. lässt sich so nicht zeigen.

Schritt 4: Wir führen eine zufällige Verschiebung des Gitters durch.
Die betrachtete worst-case Situation lässt sich mit Leichtigkeit vermeiden, wenn wir die Platzierung des Gitters der Trennlinien zufällig auswürfeln! Dazu wählen wir ganze Zahlen $a, b \in [0, L)$ zufällig, verschieben die x- und y-Koordinaten aller Trennlinien um a bzw. b und rechnen modulo L. Die von einer Rundreise zu durchlaufenden Punkte bleiben natürlich unverändert.
Warum funktioniert das zufällge Verschieben des Gitters? Dazu bestimmen wir die erwartete Länge $\mathbb{E}[\text{Umweg}]$ aller Umwege, die uns die Türen abverlangen. Sei Treffer_i die Häufigkeit mit der R_{opt} eine Trennlinie der Schicht i kreuzt. Dann ist

$$\mathbb{E}\left[\text{Umweg}\right] \leq \sum_i \text{umweg}_i \cdot \mathbb{E}\left[\text{Treffer}_i\right].$$

Wir müssen also die erwartete Anzahl der Kreuzungen von R_{opt} mit einer Trennlinie der Schicht i bestimmen. Dazu betrachten wir zuerst die Wahrscheinlichkeit p_i, dass eine beliebige fixierte Zeile (bzw. Spalte) des kleinsten, alle Punkte umschließenden Quadrats nach zufälliger Verschiebung des Gitters zu einer Zeile (bzw. Spalte) der Tiefe i wird. Wenn man sich alle Trennlinien der Schicht der gleichen Zeile oder Spalte vereinigt denkt, dann gehören höchstens 2^i Zeilen und 2^i Spalten von Q zur Schicht i. Aber Q hat L Zeilen und L Spalten. Also ist

$$p_i \leq \frac{2^i}{L}.$$

Eine Kante e von $R_{\texttt{opt}}$, die die Punkte x und y verbindet, kreuzt höchstens $O(d(x, y))$ Gitterlinien. Die erwartete Anzahl der Kreuzungen von e mit einer Gitterlinie der Schicht i ist also höchstens proportional zu $d(x, y) \cdot p_i$. Aber dann ist $\mathbb{E}\left[\text{Treffer}_i\right] = O(|R_{\texttt{opt}}| \cdot p_i)$, und wir erhalten

$$\begin{aligned}
\mathbb{E}\left[\text{Umweg}\right] &= O(\sum_i \text{umweg}_i \cdot |R_{\texttt{opt}}| \cdot p_i) \\
&= O(|R_{\texttt{opt}}| \cdot \sum_i \frac{1}{m}\frac{L}{2^i} \cdot \frac{2^i}{L}) \\
&= O(|R_{\texttt{opt}}| \cdot \varepsilon),
\end{aligned}$$

denn $m = \Theta(\frac{\log_2 n}{\varepsilon})$.

Das **Struktur-Theorem**, die *endgültige Version*: Nach einer zufälligen Verschiebung des Gitters gibt eine legale Rundreise der Länge höchstens $(1+\varepsilon) \cdot \texttt{opt}$, so dass jede Kante eines Teilquadrats des Quadtrees T_Q höchstens $1/\varepsilon$ mal durchquert wird. Diese Aussage gilt mit Wahrscheinlichkeit mindestens 1/2.

Wir haben zwar gezeigt, dass es kurze legale Rundreisen gibt, nicht aber, dass eine solche Rundreise jede Kante eines Teilquadrats von T_Q höchstens $1/\varepsilon$ mal durchquert. Für einen Beweis verweisen wir auf die Arbeit „Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems“ von Sanjeev Arora.

Schritt 5: Die schnelle Bestimmung einer kurzen legalen Rundreise.
Mit dem Struktur-Theorem genügt es, eine kürzeste legale Rundreise $R_{\texttt{legal}}$ zu bestimmen, die jede Kante eines Teilquadrats höchstens

$$r = 1/\varepsilon$$

mal durchquert. Natürlich nutzen wir aus, dass die relativ wenigen Kreuzungen der Trennlinien durch Türen verlaufen müssen. Die dynamische Programmierung wird uns helfen.
Wir betrachten ein beliebiges nicht-leeres Teilquadrat Q'. Für Q' lösen wir alle möglichen *Verbindungsprobleme*. In einer Instanz eines solchen Verbindungsproblems ist eine Folge $a_1, a_2, \ldots, a_{2i-1}, a_{2i}, \ldots, a_{2p-1}, a_{2p}$ von $2p$ Türen mit $2p \leq 4r$ gegeben, die alle auf den Kanten von Q' oder den Ecken von Q' platziert sind. Gesucht sind p in Q' verlaufende Wege $P_1, \ldots, P_p$ minimaler Länge, so dass

1. Weg P_i in Tür a_{2i-1} startet und in Tür a_{2i} endet,
2. alle in Q' enthaltenen Punkte durchlaufen werden und
3. all diese Wege jede Kante eines Teilquadrats von Q' insgesamt höchstens r mal und zwar nur in Türen durchqueren.

Es gibt höchstens $(4m+4)^{4r}$ Instanzen $\vec{a}$ für das Verbindungsproblem von Q'. Unser dynamisches Programm wird das Verbindungsproblem für alle Instanzen optimal lösen und zwar beginnend in den Blättern des Quadtrees.
Das Blatt b sei mit dem Teilquadrat Q' markiert. Wir lösen jedes Verbindungsproblem für Q' extrem schnell, nämlich in Zeit $O(r)$. Als einzige nichttriviale Operation müssen wir herausfinden, welcher Weg den in Q' enthaltenen Punkt traversieren sollte, wenn denn Q' nichtleer ist.
Im allgemeinen Fall sei v ein innerer Knoten des Quadtrees und v sei mit dem Teilquadrat Q' markiert. Sämtliche Verbingsprobleme für die Kinder von v seien bereits gelöst und gespeichert. Sei $\vec{a}$ eine Instanz des Verbindungsproblems von Q'. Für alle $((4m+4)^{4r})^4$ Kombinationen der Verbindungsprobleme der vier Kinder untersucht der Algorithmus zuerst, ob das Verbindungsproblem für $\vec{a}$ gelöst wird und bestimmt dann die beste Lösung. Aber v hat „nur“ $(4m+4)^{4r}$ Verbindungsprobleme, und wir können alle in Zeit

$$(4m+4)^{4r} \cdot ((4m+4)^{4r})^4 = m^{O(r)} = (\frac{\log_2 n}{\varepsilon})^{O(1/\varepsilon)}$$

lösen. Hier ist Arora's Algorithmus nochmal im Überblick.

Algorithmus 3.21 **Arora's Algorithmus**

(1) n Punkte im $\mathbb{R}^2$ sind vorgegeben. Das Traveling Salesman Problem für diese Punkte ist mit Approximationsfaktor höchstens $1+\varepsilon$ zu lösen.

(2)
- Bestimme das kleinste umschließende Quadrat Q und Quadtree der Teilprobleme.
- Der Tür-Parameter wird auf $m = \frac{2 \cdot \log_2 n}{\varepsilon}$ gesetzt.

(3) Löse jedes Verbindungsproblem für jeden Knoten von T_Q mit Hilfe der dynamischen Programmierung.

Kommentar: Jedes Teilproblem wird in Zeit $m^{O(r)}$ für $r = 1/\varepsilon$ gelöst.

Satz 3.22 *Arora's Algorithmus ist ein Approximationsschema für das Traveling Salesman Problem in der Ebene. Eine* $(1+\varepsilon)$*-approximative Lösung wird in Zeit*

$$n(\frac{\log_2 n}{\varepsilon})^{O(1/\varepsilon)}$$

gefunden.

Bemerkung 3.3
Die Laufzeit von Arora's Algorithmus kann durch die Anwendung mehrerer Tricks auf $O(n \cdot \log_2 n + n \cdot (\frac{1}{\varepsilon})^{O(1/\varepsilon)})$ verbessert werden. Aber auch in den schnellsten Varianten ist er den etablierten Algorithmen in praktischen Anwendungen unterlegen: Diese können $(1+\varepsilon)$-approximative Lösungen für allerdings nicht zu kleine Werte von ε relativ schnell bestimmen, während Arora's Algorithmus mit großer Laufzeit bezahlen muss. Die wesentliche Konsequenz von Arora's Algorithmus ist der Nachweis eines Approximationsschemas und damit die berechtigte Hoffnung, dass heuristische oder enumerative Verfahren in der Praxis gut abschneiden.
Arora's Algorithmus kann auch auf weitere geometrische Optimierungsprobleme in der Ebene angewandt werden. Zu diesen Problemen zählen das Facility Location Problem und das k-Median Problem.

Kapitel 4

Lokale Suche

Wir untersuchen jetzt das vielleicht einfachste, aber in vielen Anwendungen auch erfolgreiche Prinzip der lokalen Suche. Dazu nehmen wir an, dass das Minimierungsproblem $P = (\min, f, L)$ zu lösen ist, wobei zusätzlich zu jeder Lösung y auch eine Umgebung $\mathcal{N}(y)$ benachbarter Lösungen gegeben ist.

Algorithmus 4.1 **Strikte lokale Suche**

(1) Sei $y^{(0)}$ eine Lösung für Instanz x. Setze $i = 0$.

(2) Wiederhole solange, bis eine lokal optimale Lösung gefunden ist:

(2a) Bestimme einen Nachbarn $y \in \mathcal{N}(y^{(i)})$, so dass $f(x, y) < f(x, y^{(i)})$ und y eine Lösung ist.

(2b) Setze $y^{(i+1)} = y$ und $i = i + 1$.

Im Entwurf eines lokalen Suchverfahrens müssen die folgenden Fragen beantwortet werden.

(1) Wie sind die Nachbarschaften $\mathcal{N}(y)$ zu definieren?

Häufig werden k-Flip Nachbarschaften gewählt: Wenn nur Elemente aus $\{0,1\}^n$ als Lösungen in Frage kommen und wenn $y \in \{0,1\}^n$ die gegenwärtige Lösung ist, dann ist

$$\mathcal{N}_k(y) = \{y' \in \{0,1\}^n \mid y \text{ und } y' \text{ unterscheiden sich in höchstens } k \text{ Positionen}\}$$

die k-Flip Nachbarschaft von y. Man beachte aber, dass die k-Flip Nachbarschaft $\binom{n}{k}$ Elemente besitzt, und die Bestimmung eines Nachbarn mit kleinstem Funktionswert ist schon für kleine Werte von k eine Herausforderung.

(2) Mit welcher Anfangslösung $y^{(0)}$ soll begonnen werden?

Häufig werden lokale Suchverfahren benutzt, um eine durch eine Heuristik gefundene Lösung zu verbessern. In einem solchen Fall muss der Nachbarschaftsbegriff sorgfältig gewählt werden, damit die ursprüngliche Lösung nicht schon ein lokales Optimum ist. Eine zufällig ausgewürfelte Lösung ist eine zweite Option.

(3) Mit welcher Nachbarlösung soll die Suche fortgesetzt werden?

Wenn die Nachbarschaft genügend klein ist, dann liegt die Wahl eines Nachbarn mit kleinstem Zielfunktionswert nahe. Bei zu großen Nachbarschaften wählt man häufig benachbarte Lösungen mit Hilfe einer Heuristik, bzw. man wählt einen zufälligen Nachbarn.

Aufgabe 50

Im MAX-CUT Problem ist ein ungerichteter Graph $G = (V, E)$ gegeben. Es ist eine Knotenmenge $W \subseteq V$ zu bestimmen, so dass die Anzahl kreuzender Kanten (also die Anzahl der Kanten mit genau einem Endpunkt in W) größtmöglich ist.

Zeige, dass lokale Suche für *jede* Anfangslösung den Approximationsfaktor 2 besitzt. Wir nehmen dazu an, dass die Umgebung einer Knotenmenge W aus allen Knotenmengen U mit $|W \oplus U| = 1$ besteht. $W \oplus U = (W \cup V) \setminus (W \cap V)$ bezeichnet dabei die symmetrische Differenz der Mengen W und U.

Aufgabe 51

Im Max-2-SAT Problem sind ein Menge von Klauseln mit jeweils genau 2 Literalen gegeben. Literale sind positive oder negierte Ausprägungen der Variablen. Wir setzen voraus, dass jede Variable in mindestens einer Klausel auftritt, und dass die trivialen Klauseln $(x_i \vee \overline{x_i})$, $(x_i \vee x_i)$ sowie $(\overline{x_i} \vee \overline{x_i})$ nicht vorkommen. Gesucht ist eine Belegung der Variablen, so dass die Anzahl der erfüllten Klauseln maximiert wird.

Für die Variablen $x_1, \ldots, x_k$ fassen wir ihre Belegung als einen Vektor $\vec{x} \in \{0,1\}^k$ auf. Wir definieren $U(\vec{x}) = \{\vec{y} | \sum_{i=1}^{k} |x_i - y_i| = 1\}$ als Umgebung von $\vec{x}$.

Die lokale Suche erlaubt also genau dann die Belegung *einer* Variablen zu negieren, wenn sich dadurch die Anzahl der erfüllten Klauseln erhöht.

(a) **Zeige**, dass lokale Suche für Max-2-SAT für *jede* Anfangslösung höchstens den Approximationsfaktor 2 erreicht.

(b) **Konstruiere** für unendlich viele natürliche Zahlen k Max-2-Sat Instanzen über den Variablen $x_1, \ldots, x_k$, so dass die Instanzen ein möglichst kleines lokales Maximum im Vergleich zum globalen Maximum besitzen. Wie groß kann der Bruch $\frac{\text{Optimum}}{\text{Wert des lokalen Maximums}}$ gemacht werden?

Aufgabe 52

In dieser Aufgabe untersuchen wir den Einfluss des Abstandsbegriffs auf die Güte der Approximation bei der lokalen Suche. Wir betrachten das Problem des gewichteten bipartiten Matchings. Gegeben ist ein bipartiter Graph $G = (V_1 \cup V_2, E)$ mit $V_1 \cap V_2 = \emptyset$ und $E \subseteq V_1 \times V_2$ und eine Gewichtsfunktion $w : E \to \mathbb{R}$. Gesucht ist ein Matching maximalen Gewichts. Das Gewicht eines Matchings M ist die Summe der Gewichte der enthaltenen Kanten. Zu gegebenem Matching M definieren wir die $k - Umgebung$ als

$$U_k(M) := \{M' | |(M' \setminus M) \cup (M \setminus M')| \leq k\}.$$

Die lokale Suche gemäß der $k - Umgebung$ erlaubt den Übergang von einem Matching M zu einem Matching aus $U_k(M)$.

(a) **Zeige,** dass die lokale Suche gemäß der 2-Umgebung für das gewichtete bipartite Matching keinen konstant beschränkten Approximationsfaktor hat.

(b) **Zeige,** dass die lokale Suche gemäß der 3-Umgebung einen Approximationsfaktor von höchstens 2 hat.

(c) **Zeige,** dass die lokale Suche gemäß der 3-Umgebung einen Approximationsfaktor von mindestens 2 hat.

Beispiel 4.1 **Matroide**
Mit dem Greedy-Algorithmus für Matroide haben wir bereits einen lokalen Suchalgorithmus kennengelernt: Lösungen entsprechen Mengen eines monotonen Teilmengensystems $(\mathcal{P}, X)$. Für eine Menge $x \in \mathcal{P}$ wählt der Greedy-Algorithmus die Umgebung

$$U(x) = \{y \in \mathcal{P} \mid x \subseteq y \text{ und } |x| + 1 = |y| \}.$$

Man beachte allerdings, dass die strikte lokale Suche nicht notwendigerweise mit dem besten Nachbarn fortgesetzt wird: Die Fortsetzung der Suche mit irgendeinem besseren Nachbarn ist erlaubt. Nur die Variante, die tatsächlich den besten Nachbarn wählt, stimmt mit dem Greedy Algorithmus für Matroide überein.

Beispiel 4.2 **Der Simplex-Algorithmus für das lineare Programmieren**
Der Lösungsraum des linearen Programmierproblems

$$\min c^T \cdot y, \text{ so dass } A \cdot y \geq b \text{ und } y \geq 0$$

ist ein Durchschnitt von Halbräumen der Form $\{y \mid \alpha^T \cdot y \geq \beta \}$. Man kann sich deshalb leicht davon überzeugen, dass das Optimum an einer „Ecke“ des Lösungsraums angenommen wird.
Der Simplex-Algorithmus für die lineare Programmierung wandert solange von einer Ecke zu einer besseren, aber im Lösungsraum benachbarten Ecke, bis keine Verbesserung erreichbar ist. Da der Lösungsraum konvex[1] ist, kann man zeigen, dass Simplex stets eine optimale Ecke findet. (Siehe Kapitel 11.6.) Wenn wir also nur Ecken als Lösungen zulassen, dann ist Algorithmus 4.1 die Grobstruktur des sehr erfolgreichen Simplex-Algorithmus.

Beispiel 4.3 **Minimierung durch Gradientenabstieg**
Wir nehmen an, dass das Minimierungsproblem $P = (\min, f, L)$ eine differenzierbare Zielfunktion $f : \mathbb{R}^n \to \mathbb{R}$ besitzt und dass der Lösungsraum eine kompakte Teilmenge des $\mathbb{R}^n$ ist. Wenn wir uns gegenwärtig in der Lösung a befinden, in welcher Richtung sollten wir nach kleineren Funktionswerten suchen? In der Richtung des negativen Gradienten!

Lemma 4.2 (Gradientenabstieg) *Sei $f : \mathbb{R}^n \to \mathbb{R}$ eine zweimal im Punkt $a \in \mathbb{R}^n$ stetig differenzierbare Funktion mit $\nabla f(a) \neq 0$. Dann gibt es ein $\eta > 0$ mit*

$$f\left(a - \eta \cdot \nabla f(a)\right) < f(a),$$

wobei $\nabla f(a)$ der Gradient von f an der Stelle a ist.

Beweis: Wir können die Funktion f durch ihre linearen Taylor-Polynome approximieren und erhalten für hinreichend kleines z, dass

$$f(a + z) = f(a) + \nabla f(a)^T \cdot z + O(z^T \cdot z).$$

[1]Eine Teilmenge $X \subseteq \mathbb{R}^n$ heißt konvex, wenn X mit je zwei Punkten auch die verbindende Gerade enthält.

Für $z = -\eta \cdot \nabla f(a)$ bei hinreichend kleinem $\eta > 0$ erhalten wir

$$f(a - \eta \cdot \nabla f(a)) = f(a) - \eta \cdot \ ||\nabla f(a)||^2 + \eta^2 \cdot O(||\nabla f(a)||^2).$$

Für hinreichend kleines $\eta < 1$ wird somit $f(a - \eta \cdot \nabla f(a))$ kleiner als $f(a)$ sein. □

Wir verringern also den Funktionswert durch eine hinreichend kleine Bewegung in Richtung des negativen Gradienten und die Strategie

$$x^{(i+1)} = x^{(i+1)} - \eta \cdot \nabla f(x^{(i)})$$

für ein genügend kleines η liegt nahe. Wenn wir die Nachbarschaft einer Lösung x als einen kleinen Ball um x definieren, dann beschreibt Algorithmus 4.1 die Methode des iterierten Gradientenabstiegs. Diese Methode wird zum Beispiel in interior-point Verfahren (Kapitel 8) und in dem neuronalen Lernverfahren „Backpropagation“ angewandt.

Die große Gefahr in Anwendungen der lokalen Suche sind lokale Optima, also Lösungen x für die keine benachbarte Lösung besser als x ist.

Beispiel 4.4 Wir führen eine lokale Suche für VERTEX COVER durch. Für einen gegebenen Graphen $G = (V, E)$ und eine Kontenüberdeckung y definieren wir die Nachbarschaft $\mathcal{N}(y)$ als die Klasse aller Teilmengen $z \subseteq V$, die durch das Hinzufügen oder das Entfernen eines Elementes aus y entstehen. Wir beginnen die lokale Suche mit der Lösung $U = V$. Da der Suchalgorithmus 4.1 fordert, dass nur Nachbarn mit kleinerem Zielfunktionswert gewählt werden dürfen, werden nacheinander Elemente entfernt bis ein lokales Optimum erreicht ist.
Für den leeren Graphen $G = (V, \emptyset)$ funktioniert die lokale Suche komplikationslos und $U = \emptyset$ wird nach $|V|$ Suchschritten als lokales Optimum ausgegeben.
Betrachten wir als nächstes den Sterngraphen mit dem Sternzentrum 0 und den Satelliten $1, \ldots, n$. Die einzigen Kanten des Sterngraphen verbinden das Zentrum 0 mit allen Satelliten. Wird im ersten Schritt das Zentrum entfernt, dann haben wir ein sehr schlechtes lokales Minimum erreicht. Wird hingegen im ersten Schritt ein Satellit entfernt, dann wird zwangsläufig das nur aus dem Zentrum bestehende globale Minimum gefunden. (Beachte, dass nur Nachbarn betrachtet werden, die auch Knotenüberdeckungen sind.)
Komplizierter ist die Situation, wenn wir den aus den n Knoten $0 \ldots, n-1$ bestehenden Weg $0 \to 1 \to 2 \to \cdots \to n-2 \to n-1$ als Graphen wählen. Für gerades n ist $U_{\text{opt}} = \{0, 2, 4, \ldots, n-2\}$ eine optimale Knotenüberdeckung.

Aufgabe 53
Bestimme alle lokalen Optima für den Weg mit n Knoten. Bestimme den Wert des schlechtesten lokalen Optimums.

4.1 PLS: Polynomielle Suchprobleme

Wir haben bisher die Probleme, auf die lokale Suche anwendbar ist, nur informell eingeführt. Wir geben jetzt eine formale Definition und untersuchen dann die Frage, ob sich lokale Minima stets effizient bestimmen lassen.

Definition 4.3 Die Klasse $\mathcal{PLS}$ aller polynomiellen Suchprobleme besteht aus allen $\mathcal{NP}$-Optimierungsproblemen (opt, f, L) mit den folgenden zusätzlichen Eigenschaften:

- Die Funktion f nimmt nur ganzzahlige Werte an.
- Es gibt einen effizienten Algorithmus A, der für jede Instanz x_0 eine Anfangslösung y_0 berechnet.
- Es gibt einen effizienten Algorithmus B, der für jede Instanz x_0 und jede Lösung y mit $L(x_0, y)$ entscheidet, ob y ein lokales Minimum ist; ist dies nicht der Fall, dann bestimmt B einen besseren Nachbarn, also eine Lösung $y' \in N_{x_0}(y)$ mit $f(y') < f(y)$ für opt = min, bzw. $f(y') > f(y)$ für opt = max.

Wir beschreiben ein polynomielles Suchproblem durch das Tupel (opt, f, L, A, B).

Aufgabe 54
Die Komplexitätsklasse $\mathcal{FNP}$ besteht aus allen binären Relationen $R(x, y)$, so dass

- y höchstens polynomiell länger als x ist und
- $R(x, y)$ in polynomieller Zeit ausgewertet werden kann, wenn x und y bekannt sind.

Die Klasse $\mathcal{TFNP}$ besteht aus allen Relationen in $\mathcal{FNP}$, so dass es für jedes x ein y gibt, so dass $R(x, y)$ gilt.
$\mathcal{FNP}$-Algorithmen müssen für jedes x, falls möglich ein y bestimmen, so dass $R(x, y)$ gilt.
Zeige, dass $\mathcal{PLS}$ als eine Teilklasse von $\mathcal{TFNP}$ aufgefast werden kann.

Aufgabe 55
Wir definieren ein „Congestion Spiel“ wie folgt: Gegeben sind n Spieler und m Ressourcen. Spieler i ist daran interessiert, Ressourcen aus einer Menge $R_i \subseteq \{1, \ldots, m\}$ zu erwerben, allerdings hängt der Preis der Ressource j von der Nachfrage ob: Die Funktion $\text{preis}_j : \mathbb{N} \to \mathbb{Z}$ beschreibt die Preisgestaltung für Ressource j. Wenn Spieler i tatsächlich die Teilmenge $r_i \subseteq R_i$ erwirbt, dann muss er die Kosten

$$\sum_{j \in a_i} \text{preis}_j(|\{k \mid j \in r_k\}|)$$

tragen. Ein jeder Spieler ist egoistisch und versucht, seine Kosten zu minimieren. Wir sagen, dass $(r_1, \ldots, r_n)$ ein Nash-Gleichgewicht ist, wenn die Kosten für jeden Spieler höchstens ansteigen, wenn er seine Wahl r_i ändert.
Warum lässt die Bestimmung eines Nash-Gleichgewichts für Congestion-Spiele als ein polynomielles Suchproblem auffasssen?

Für jedes polynomielle Suchproblem (opt, f, L, A, B) erhalten wir den *Standardalgorithmus*, also das folgende lokale Suchverfahren:

Algorithmus 4.4

(1) Berechne $y = A(x_0)$.

(2) Solange y kein lokales Optimum ist, setze $y = B(x_0, y)$.

Wir fragen uns zuerst, ob das von Algorithmus 4.4 bestimmte lokale Optimum effizient berechnet werden kann. Die Antwort ist offensichtlich positiv, wenn f nur polynomiell (in der Eingabelänge) viele Werte annimmt. Aber was passiert, wenn f exponentiell viele Werte annehmen kann?

Aufgabe 56
Wir definieren das Standardproblem für ein PLS-Problem $P = (\text{opt}, F, L, A, B)$ wie folgt:

Für eine Instanz x bestimme das vom Standardalgorithmus berechnete lokale Optimum s.

Konstruiere ein PLS-Problem P, so dass das Erfüllbarkeitsproblems effizient lösbar ist, wenn wir eine Subroutine kostenfrei nutzen kĺonnen, die das Standardproblem lĺost.
Als Konsequenz können wir sagen, dass die Lösung des Standardproblems „$\mathcal{NP}$-hart“ ist.

Was aber passiert, wenn wir nur irgendein lokales Optimum bestimmen wollen?

Aufgabe 57
Wir sagen, dass wir eine Sprache K mit Hilfe eines PLS-Problems P effizient lösen können, wenn es einen effizienten deterministischen Algorithmus für die Sprache K gibt, der eine Subroutine zur Lösung von P kostenfrei nutzen darf.
Zeige: Wenn wir eine $\mathcal{NP}$-vollständige Sprache K mit Hilfe irgendeines PLS -Problems effizient lösen können, dann ist $\mathcal{NP} = \text{co}\mathcal{NP}$. Als Konsequenz ist es unwahrscheinlich, dass wir die "$\mathcal{NP}$-Härte“ für irgendein PLS -Problem nachweisen können.
Hinweis: Es genügt zu zeigen, dass das Komplement von K in $\mathcal{NP}$ liegt.

Wenn wir Aussagen über die Schwierigkeit der Bestimmung eines lokalen Optimums machen wollen, dann wird uns also die Theorie der $\mathcal{NP}$-Vollständigkeit nicht weiterhelfen. Stattdessen betrachten wir zuerst schwierigste Probleme in $\mathcal{PLS}$ und definieren passende Reduktionen zwischen polynomiellen Suchproblemen.

Definition 4.5 Für polynomielle Suchprobleme $P_1 = (\text{opt}_1, f_1, L_1, A_1, B_1)$ und $P_2 = (\text{opt}_2, f_2, L_2, A_2, B_2)$ sagen wir, dass P_1 auf P_2 PLS-reduzierbar ist ($P_1 \leq_{PLS} P_2$), wenn es effiziente Transformationen Φ und Ψ mit den folgenden Eigenschaften gibt:

(a) Wenn x eine Instanz von P_1 ist, dann ist $\Phi(x)$ eine Instanz von P_2.

(b) Wenn y ein lokales Optimum für P_2 und Instanz $\Phi(x)$ ist, dann ist $\Psi(x, y)$ ein lokales Optimum für P_1 und Instanz x.

Wir sagen, dass P PLS-vollständig ist, wenn P ein polynomielles Suchproblem ist und wenn $Q \leq_{PLS} P$ für jedes polynomielle Suchproblem Q gilt.

Angenommen, P ist ein PLS-vollständiges Suchproblem. Wenn ein lokales Optimum für jede Instanz von P effizient berechnet werden kann, dann lassen sich offensichtlich lokale Optima für alle polynomiellen Suchprobleme bestimmen. Diese Beobachtung ist Anlass für die Vermutung, dass lokale Optima für PLS-vollständige Probleme wahrscheinlich nicht effizient berechenbar sind. Welche Probleme sind denn PLS-vollständig?

Beispiel 4.5 Im MINIMUM BALANCED CUT Problem wird zu einem gegebenen, ungerichteten Graphen $G = (V, E)$ mit gewichteten Kanten eine Knotenmenge $W \subseteq V$ mit

$|W| = \frac{1}{2} \cdot |V|$ gesucht, so dass das Gewicht

$$f(W) = \sum_{e \in E, |\{e\} \cap W| = 1} \text{gewicht}(e)$$

kreuzender Kanten minimal ist. Eine Anwendung von MINIMUM BALANCED CUT ist der Entwurf von VLSI-Layouts, indem zuerst Layouts rekursiv für

$$(W, \{e \in E \mid e \subseteq W\}) \quad \text{und} \quad (V \setminus W, \{e \in E \mid e \subseteq V \setminus W\})$$

gesucht werden. Wenige kreuzende Kanten sind für die nachfolgende, platz-sparende Kombination der beiden Layouts wichtig. Für die lokale Suche bietet sich zum Beispiel die $2k$-Flip Nachbarschaft an, also der Versuch, ein Element der gegenwärtigen Lösung W durch ein Elemente des Komplements zu ersetzen. Wir definieren deshalb

$$N(W) = \{ U \subseteq V \mid |U| = \frac{|V|}{2} \text{ und } |U \oplus W| = 2\}$$

als die Umgebung von W.

Ohne Beweis geben wir die folgenden $\mathcal{PLS}$-vollständigen Probleme an.

Satz 4.6 *Die folgenden polynomiellen Suchprobleme sind $\mathcal{PLS}$-vollständig:*

(a) Die Bestimmung eines lokalen Minimums für TSP, wenn eine Rundreise mit allen Rundreisen benachbart ist, die sich durch einen Austausch von höchstens k Kanten ergeben. (Die Konstante k muss hinreichend groß gewählt werden.)

(b) Die Bestimmung eines lokalen Minimums für MINIMUM BALANCED CUT, wenn zwei Knotenmengen hälftiger Größe genau dann benachbart sind, wenn sie durch die Ersetzung eines Knotens auseinander hervorgehen.

(c) Die Bestimmung eines Nash-Gleichgewichts für Congestion Spiele.

Aufgabe 58
Zeige: Wenn die von Algorithmus 4.4 berechneten lokale Optima für irgendein $\mathcal{PLS}$-vollständiges Problem effizient bestimmt werden können, dann gilt

$$\mathcal{NP} = co\mathcal{NP}.$$

(co$\mathcal{NP}$ besteht aus allen Sprachen, deren Komplement in $\mathcal{NP}$ liegt.)

Was bedeutet $\mathcal{PLS}$-Vollständigkeit für uns? Nichts, wenn wir wissen, dass das vorliegende Problem nur polynomiell viele verschiedene Werte annimmt. Bei sehr vielen verschiedenen Werten müssen wir aber vorsichtig sein: Die effiziente Berechnung eines lokalen Optimums kann für ein $\mathcal{PLS}$-vollständiges Probleme zu schwierig sein, und wir müssen uns mit approximativen lokalen Optima zufrieden geben!

Definition 4.7 $(\min, f, L, A, B)$ sei ein polynomielles Suchproblem und y sei eine Lösung für die Instanz x. Dann ist y ein ε-approximatives lokales Optimum, wenn

$$f(x, y) \leq (1 + \varepsilon) \cdot f(x, y')$$

für alle benachbarten Lösungen $y' \in N_x(y)$ gilt.

Gibt es stets ein volles polynomielles Approximationsschema für die Berechnung eines ε-approximativen lokalen Optimums? Ja, in Zeit polynomiell in der Länge der Eingabe und $1/\varepsilon$ gelingt eine solche Bestimmung, wenn wir *lineare kombinatorische Optimierungsprobleme* betrachten. In einem linearen kombinatorischen Optimierungsproblem ist ein Universum U und eine Gewichtung $w_u \in \mathbb{N}$ für alle Elemente $u \in U$ des Universums gegeben; desweiteren ist eine Klasse $\mathcal{Q}$ von Teilmengen von U gegeben, die den Lösungen entsprechen. Das Ziel ist die Bestimmung einer Teilemnge $y \in \mathcal{Q}$, so dass

$$\text{gewicht}(y) = \sum_{u \in y} w_u$$

eine leichteste Lösung ist[2]. Die Idee für die Bestimmung eines ε-approximativen lokalen Optimums ist simpel.

(1) Wir skalieren die Gewichte „herunter“, indem wir w_u durch $\lceil \frac{w_u}{q} \rceil$ ersetzen. Die Zahl q ist so gewählt, dass die Anzahl möglicher Werte hinreichend klein ist, so dass wir ein lokales Optimum effizient bestimmen können. Insbesondere setzen wir

$$q = \frac{\varepsilon}{1 + \varepsilon} \cdot \frac{\text{gewicht}(y)}{2|U|},$$

wenn wir gerade die Lösung y erreicht haben.

(2) Bestimme ein lokales Optimum y' für die neuen Gewichte.

Dieser Ansatz funktioniert noch nicht ganz: Wenn die Zielfunktion in der Berechnung in Schritt (2) zu stark absinkt, dann sollten wir q entsprechend verringern. Deshalb der neue Ansatz

(2) Berechne ein lokales Optimum für die neuen Gewichte schrittweise. Setze $y' = y$.

   (a) Wenn y' bereits ein lokales Optimum für die neuen Gewichte ist, dann halte.

   (b) Sonst, wenn $\text{gewicht}(y') < \text{gewicht}(y)/2$, dann setze $y = y'$ und gehe zu Schritt (1).

   (c) Sonst führe einen weiteren Schritt in der lokalen Suche mit den neuen Gewichten aus und nenne die aktualisierte Lösung wieder y'. Mach mit Schritt (2a) weiter.

[2]Man überzeuge sich, dass sich alle bisher betrachteten Optimierungsprobleme als lineare, kombinatorische Optimierungsprobleme auffassen lassen.

Wir stoppen mit einer Lösung y', so dass y' ein lokales Optimum für die neuen Gewichte ist und so dass gewicht$(y') \geq$ gewicht$(y)/2$ gilt. Ist y' ein ε-approximatives lokales Optimum? Für einen beliebigen Nachbarn y'' von y' folgt

$$\begin{aligned} \text{gewicht}(y') &= \sum_{u \in y'} w_u \leq \sum_{u \in y'} \lceil \frac{w_u}{q} \rceil \cdot q \leq \sum_{u \in y''} \lceil \frac{w_u}{q} \rceil \cdot q \leq \sum_{u \in y''} (\frac{w_u}{q} + 1) \cdot q \\ &\leq \text{gewicht}(y'') + |U| \cdot q. \end{aligned}$$

Aber gewicht$(y') >$ gewicht$(y)/2$ und deshalb erhalten wir

$$\begin{aligned} \frac{\text{gewicht}(y') - \text{gewicht}(y'')}{\text{gewicht}(y'')} &\leq \frac{|U| \cdot q}{\text{gewicht}(y'')} \leq \frac{|U| \cdot q}{\text{gewicht}(y') - |U| \cdot q} \\ &\leq \frac{|U| \cdot q}{\text{gewicht}(y)/2 - |U| \cdot q} = \varepsilon \end{aligned}$$

und y' ist tatsächlich ein ε-approximatives lokales Optimum.

Satz 4.8 *Für jedes lineare kombinatorische Optimierungsproblem kann ein ε-approximatives lokales Optimum in*

$$\frac{|U|}{\varepsilon} \cdot \log_2(\text{gewicht}(y_0))$$

Schritten einer lokalen Suche bestimmt werden, wenn y_0 die Anfangslösung ist.

Warum ist die Schranke für die Schrittzahl richtig? Höchstens $\log_2(\text{gewicht}(y))$-mal kann das Lösungsgewicht halbiert werden. Zwischen zwei Halbierungen liegen aber höchstens $O(\frac{|U|}{\varepsilon})$ Schritte der lokalen Suche.

4.2 FACILITY LOCATION und Cluster Probleme

Im FACILITY LOCATION Problem ist eine Menge K von Kunden, eine Menge S möglicher Service Stationen sowie eine Metrik d auf $K \cup S$ gegeben, wobei $d(k, s)$ die Distanz des Kunden $k \in K$ von der Service Station $s \in S$ angibt. Für jede Service Station $s \in S$ werden die Betriebskosten durch f_s angegeben. Schließlich definieren wir die Anschlusskosten von Kunde k an X durch

$$\text{distanz}_X(k) = \min_{s \in X} d(k, s).$$

Wir suchen eine Teilmenge $X \subseteq S$, so dass

$$C(X) = \sum_{k \in K} \text{distanz}_X(k) + \sum_{s \in X} f_s,$$

die Summe aller Anschluss- und Betriebskosten minimal ist.
Wenn $\text{distanz}_X(k) = d(k, s)$, dann definieren wir $\text{naechste}_X(k) = s$ als die günstigste Service Station für den Kunden k.

Bemerkung 4.1 Im SET COVER Problem sind ein Universum U und Teilmengen $T_1, \cdots, T_k$ mit den Gewichten $w_1, \ldots, w_k \geq 0$ gegeben. Die Teilmengen überdecken das Universum, d.h. es gilt $\bigcup_{i=1}^{k} T_i = U$. Unser Ziel ist eine Überdeckung mit möglichst geringem Gesamtgewicht.
SET COVER lässt keine effizienten Approximationsalgorithmen mit konstanten Faktoren zu. Wir zeigen jetzt, dass sich SET COVER sogar als ein FACILITY LOCATION Problem auffassen lässt, *wenn* wir nicht fordern, dass d eine Metrik ist: Setze

- $K := U$,
- $S := \{T_1, \ldots, T_k\}$ und $f_{T_i} = w_i$ sowie
- $d(u, T_i) = 0$, falls $u \in T_i$, und $d(u, T_i) = \infty$ sonst.

Da wir einen effizienten Approximationsalgorithmus mit konstantem Approximationsfaktor für FACILITY LOCATION erhalten werden, ist die Forderung der Metrik-Eigenschaft also wesentlich.

Unsere Lösung für FACILITY LOCATION ist extrem einfach: Wir führen lokale Suche auf der 2-Flip Nachbarschaft durch.

Algorithmus 4.9 Lokale Suche für FACILITY LOCATION

(1) Wir beginnen mit einer beliebigen Menge $X \subseteq S$ von Service Stationen.

(2) Wenn das Entfernen, Hinzufügen oder die Ersetzung von Service Stationen zu einer Verbesserung führt, dann wird eine beliebige solche Operation ausgeführt.

(3) Wiederhole Schritt (2) solange das möglich ist.

4.2.1 Lokale und Globale Minima

Um wieviel schlechter kann ein lokales im Vergleich zu einem globalen Minimum sein?

Satz 4.10 *Sei X^* ein globales Minimum und X ein lokales Minimum. Dann gilt*

$$C(X) \leq 3 \cdot C(X^*).$$

Beispiel 4.6 Das Ergebnis von Satz 4.10 kann nicht verbessert werden. Wir betrachten dazu die Menge $K = \{k_1, \ldots, k_n\}$ von n Kunden sowie die Menge $S = \{s_0, \ldots, s_n\}$ von $n+1$ Service Stationen. Service Station s_0 hat die Betriebskosten $2n-2$, aber alle anderen Service Stationen haben keine Betriebskosten.
Wir definieren die Metrik d durch die Länge kürzester Wege in dem folgenden ungerichteten Graphen G mit Knotenmenge $K \cup S$. G besitzt nur Kanten der Länge 1, nämlich zuerst die Kanten $\{k_i, s_i\}$ und schließlich die Kanten $\{k_i, s_0\}$ für $1 \leq i \leq n$.

Aufgabe 59
Sei $G = (V, E)$ ein ungerichteter Graph, dessen Kanten durch die Funktion länge: $E \to \mathbb{R}_{\geq 0}$ gewichtet sind. Dann ist

$$d(u, v) = \text{Länge eines kürzesten Weges von } u \text{ nach } v$$

eine Metrik auf V.

Insbesondere hat Kunde k_i die Distanz 1 zu den Service Stationen s_0 und s_i sowie die Distanz 3 zu jeder anderen Service Station.
Man überzeuge sich zuerst, dass $X^* = \{s_1, \ldots, s_n\}$ ein globales Minimum mit $C(X^*) = n$ ist. Wir behaupten, dass $X = \{s_0\}$ ein lokales Minimum mit $C(X) = 2n - 2 + n = 3n - 2$ ist. Offensichtlich dürfen wir s_0 aus X nicht entfernen. Desweiteren senkt die Hinzunahme einer weiteren Service Station die Gesamtkosten nicht. Also müssen wir nur eine Ersetzung von s_0 durch s_i für $1 \leq i \leq n$ betrachten: Die Betriebskosten sinken von $2n - 2$ auf Null, die Anschlusskosten steigen von n auf $1 + 3(n - 1) = 3n - 2$. Also gibt es auch hier keine Verbesserung.

Wir werden das etwas schwächere Ergebnis $C(X) \leq 5 \cdot C(X^*)$ zeigen. Als ersten Schritt im Nachweis von Satz 4.10 vergleichen wir die Anschlusskosten im lokalen Minimum X mit $C(X^*)$.

Lemma 4.11 *Sei X ein lokales Minimum und X^* ein globales Minimum. Dann gilt*

$$\sum_{k \in K} \text{distanz}_X(k) \leq C(X^*).$$

Beweis: Sei $s \in X^* \setminus X$. Wenn wir s zu X hinzufügen, dann ist $C(X) \leq C(X \cup \{s\})$, denn X ist ein lokales Minimum. Also folgt

$$\sum_{k, \text{naechste}_{X^*}(k) = s} \text{distanz}_{X^*}(k) + f_s \geq \sum_{k, \text{naechste}_{X^*}(k) = s} \text{distanz}_X(k).$$

Wenn wir diese Ungleichung über alle Service Stationen $s \in X^* \setminus X$ aufsummieren und um gemeinsame Kosten ergänzen, erhalten wir

$$C(X^*) \geq \sum_{k \in K} \text{distanz}_{X^*}(k) + \sum_{s \in X^* \setminus X} f_s \geq \sum_{k \in K} \text{distanz}_X(k),$$

und das war zu zeigen. □

Wir haben die Anschlusskosten eines lokalen Minimums mit den Kosten $C(X^*)$ eines globalen Minimums verglichen. Wir führen jetzt auch einen Vergleich für die Betriebskosten eines lokalen Minimums durch.

Lemma 4.12 *Sei X ein lokales Minimum und X^* ein globales Minimum. Dann gilt*

$$\sum_{s \in X} f_x \leq 2 \cdot \left[C(X^*) + \sum_{k \in K} \text{distanz}_X(k) \right] \leq 4 \cdot C(X^*).$$

Offensichtlich ist Satz 4.10 eine Konsequenz von Lemma 4.11 und Lemma 4.12, wenn wir das Vielfache drei durch fünf ersetzen.
Wir haben die Anschlusskosten von X mit $C(X^*)$ verglichen, indem wir versucht haben, Service Stationen aus X^* zu X hinzuzufügen. Diesmal versuchen wir, eine jede Service Station $s \in X$ durch eine nächstliegende Service Station $s^* \in X^*$ zu ersetzen. Da X ein lokales Minimum ist, gilt

$$f_s \leq f_{s*} + \sum_{k,\text{naechste}_X(k)=s} [\, d(k,s^*) - d(k,s) \,]\,. \tag{4.1}$$

Wie groß ist der Unterschied $d(k,s^*) - d(k,s)$ der Anschlusskosten in (4.1)? Mit der Dreiecksungleichung folgt $d(k,s^*) - d(k,s) \leq d(s,s^*)$. Aber Service Station s^* ist die zu s nächstliegende Service Station in X^*, und deshalb ist $d(s,s^*) \leq d(s, \text{naechste}_{X^*}(k))$. Wir wenden die Dreiecksungleichung wieder an, nämlich wir schieben k zwischen s und $\text{naechste}_{X^*}(k)$ und erhalten insgesamt

$$\begin{aligned} d(k,s^*) - d(k,s) &\leq d(s, \text{naechste}_{X^*}(k)) \leq d(s,k) + d(k, \text{naechste}_{X^*}(k)) \\ &= \text{distanz}_X(k) + \text{distanz}_{X^*}(k). \end{aligned}$$

Also folgt

$$f_s \leq f_{s*} + \sum_{k,\text{naechste}_X(k)=s} [\, \text{distanz}_X(k) + \text{distanz}_{X^*}(k) \,] \tag{4.2}$$

aus (4.1). Angenommen, jede Service Station in X^* tritt höchstens einmal als eine nächstliegende Service Station auf. Wir erhalten dann mit Lemma 4.11

$$\begin{aligned} \sum_{s \in X} f_s &\leq \sum_{k \in K} [\, \text{distanz}_X(k) + \text{distanz}_{X^*}(k)] + \sum_{s^* \in X^*} f_{s^*} \\ &= \sum_{k \in K} \text{distanz}_X(k) + C(X^*) \leq 2C(X^*), \end{aligned}$$

und wir haben sogar eine schärfere Aussage als Lemma 4.12 gezeigt. (Die verschärfte Form, zusammen mit Lemma 4.11 zeigt, dass ein lokales Minimum höchstens um den Faktor drei größer als ein globales Minimum ist.)
Leider werden die Betriebskosten von s^* dann möglicherweise mehrmals in der Abrechnung benötigt, nämlich wenn es zwei oder mehr zu s^* nächstliegende Service Stationen in X gibt. Wir zeichnen deshalb genau eine *primäre* Service Station s in X mit geringstem Abstand zu s^* aus.

Definition 4.13 Für s in X sei $s^* \in X^*$ die nächstliegende Service Station in X^*. Wir sagen, dass s eine primäre Service Station ist, wenn s die nächstliegende Service Station von s^* in X ist; ansonsten nennen wir s sekundär.

Betrachten wir also eine sekundäre Service Station $s \in X$. Sei s^* wieder die nächstliegende Service Station in X^*. Da s sekundär ist, ist eine Service Station $s' \neq s$ die zu s^* nächstliegende Service Station in X. Statt s zu ersetzen, entfernen wir s und weisen alle, bisher

s zugewiesenen Kunden der Service Station s' zu. Wir sparen die Betriebskosten, aber um wieviel steigen die Anschlusskosten für einen bisher s zugewiesenen Kunden k an?

$$\begin{aligned} d(k,s') - d(k,s) &\leq d(s,s^*) + d(s^*,s') \\ &\leq 2 \cdot d(s,s^*) \\ &\leq 2 \cdot d(s, \text{naechste}_{X^*}(k)) \\ &\leq 2 \cdot [d(s,k) + d(k, \text{naechste}_{X^*}(k))] = 2 \cdot [\text{distanz}_X(k) + \text{distanz}_{X^*}(k)] \,. \end{aligned}$$

Wir benutzen wieder, dass X ein lokales Minimum ist und erhalten

$$f_s \leq \sum_{k, \text{naechste}_X(k)=s} 2 \cdot [\text{distanz}_X(k) + \text{distanz}_{X^*}(k)] \,. \tag{4.3}$$

Wir wenden (4.2) und (4.3) an und erhalten

$$\begin{aligned} \sum_{s \in X} f_s &\leq \sum_{s^* \in X^*} f_{s^*} + 2 \sum_{k \in K} [\text{distanz}_X(k) + \text{distanz}_{X^*}(k)] \\ &\leq 2 \left[C(X^*) + \sum_{k \in K} \text{distanz}_X(k) \right] \\ &\leq 4C(X^*) \end{aligned}$$

mit Lemma 4.11. Damit ist Lemma 4.12 nachgewiesen. □

4.2.2 Wieviele Iterationen werden benötigt?

Bisher haben wir nicht sichergestellt, dass unser Algorithmus effizient ist: Viele Schritte mit nur sehr kleinen Verbesserungen können die Laufzeit drastisch erhöhen. Wir benutzen deshalb eine Variante von Algorithmus 4.9, wir führen nämlich nur dann einen Verbesserungsschritt aus, wenn wir eine Verbesserung um mindestens den Faktor $\Delta < 1$ erhalten. Wenn der Quotient des Resultats einer schlechtesten Lösung und einer optimalen Lösung durch Q beschränkt ist, dann werden jetzt höchstens $\log_{1/\Delta} Q$ Verbesserungsschritte durchgeführt. Aber wie gut ist die gefundene Lösung?
Wir führen eine sehr ähnliche Analyse aus, können aber nicht annehmen, dass unsere Lösung X ein lokales Minimum ist. Insbesondere könnte die Hinzunahme, die Entfernung oder die Ersetzung einer Service Station zu einer Verbesserung führen. Es genügt aber zu beobachten, dass keine Verbesserung den Betrag $(1-\Delta) \cdot C(X)$ übertrifft. Die zentralen Ungleichungen (4.2) und (4.3) gelten deshalb weiterhin, wenn wir die rechten Seiten um $(1-\Delta) \cdot C(X)$ erhöhen. In der Anwendung von (4.2) und (4.3) summieren wir die $|S|$ Ungleichungen und die entstehende rechte Seite steigt auf $|S| \cdot (1-\Delta) \cdot C(X)$. Wir setzen deshalb $\Delta = 1 - \frac{\varepsilon}{|S|}$ und die rechte Seite wächst um $\varepsilon \cdot C(X)$.

Satz 4.14 *Sei Q der Quotient zwischen dem Wert einer schlechtesten und besten Lösung. Wenn wir nur Verbesserungen akzeptieren, die den Wert der aktuellen Lösung um mindestens $1 - \frac{\varepsilon}{|S|}$ reduzieren, dann erhalten wir eine $3+\varepsilon$-approximative Lösung. Zudem werden höchstens $\log_{1/\Delta} Q$ Verbesserungsschritte durchgeführt.*

4.2.3 k-MEDIAN und k-CENTER

In k-MEDIAN ist eine Menge K von n Punkten gegeben sowie eine Metrik. Wir suchen eine Menge M von höchstens k Punkten aus K, so dass

$$\sum_{x \in K} \min_{y \in M} d(x, y)$$

minimal ist. k-MEDIAN ist ein Spezialfall von FACILITY LOCATION, wenn wir nur höchstens k Service Stationen erlauben und annehmen, dass keine Betriebskosten entstehen. Auch diesmal können wir mit lokaler Suche gute Lösungen finden.

Algorithmus 4.15 Lokale Suche für k-Median

(1) Beginne mit einer beliebigen Auswahl von k Punkten.

(2) Wiederhole solange möglich:

> Ersetze einen Punkt der gegenwärtigen Auswahl durch einen unbenutzten Punkt, wenn dies zu einer Verbesserung führt.

Wir erwähnen ohne Beweis:

Satz 4.16 *Der Wert eines lokalen Optimums übertrifft den Wert eines globalen Optimums um höchstens den Faktor fünf.*

Im k-CENTER Problem ist eine Menge V von Punkten gegeben. Auf der Punktmenge V ist eine Metrik $d : V \times V \to \mathbb{R}$ gegeben. Gesucht ist eine Teilmenge $C \subseteq V$ von k „Zentren“, so dass

$$D = \max_{v \in V} \min_{c \in C} d(c, v)$$

kleinstmöglich ist. In der folgenden Aufgabe wird ein 2-approximativer Algorithmus entwickelt.

Aufgabe 60
Für eine Punktmenge V und die Metrik d sei opt die kleinstmögliche Distanz. Sei $D \geq 0$ beliebig.

(1) Zu Anfang sind alle Punkte in V unmarkiert.

(2) Wiederhole, solange es unmarkierte Punkte in C gibt:

> Füge einen beliebigen unmarkierten Punkt $v \in V$ zu C hinzu und markiere alle Punkte $w \in V$ mit Abstand $d(v, w) \leq 2D$.

(a) Zeige: Wenn $D \geq$ opt, dann ist $|C| \leq k$.
(b) Gib einen 2-approximativen Algorithmus für k-CENTER an.

Das Ergebnis der obigen Übungsaufgabe ist optimal:

Satz 4.17 *Wenn $P \neq NP$, dann gibt es keinen δ-Approximationsalgorithmus für $\delta < 2$.*

Beweis: Wir führen das Problem DOMINATING SET ein: Für einen ungerichteten Graphen $G = (V, E)$ und eine Gewichtung $w_u \geq 0$ der Knoten $u \in V$ wird eine leichteste („dominierende“) Teilmenge $V' \subseteq V$ gesucht, so dass jeder Knoten $v \notin V'$ mit mindestens einem Knoten in V' verbunden ist. Wir werden später sehen, dass DOMINATING SET und SET COVER die gleiche Approximationskomplexität besitzen, insbesondere ist DOMINATING SET also $\mathcal{NP}$-vollständig.
Wir reduzieren DOMINATING SET auf k-CENTER. Sei (G, k) eine Instanz von DOMINATING SET und sei H der vollständige Graph mit Knotenmenge V und der Kantengewichtung

$$d(u, v) = \begin{cases} 1 & \{u, v\} \in E, \\ 2 & \text{sonst.} \end{cases}$$

Beachte, dass d eine Metrik ist. Wenn G eine dominierende Menge V' der Grösse k besitzt, dann können wir V' als Menge der Zentren wählen und erhalten eine Maximaldistanz von eins. Besitzt G hingegen nur dominierende Mengen, die grösser als k sind, dann ist die Maximaldistanz in H genau zwei.
Angenommen, es gibt einen effizienten δ-Approximationsalgorithmus A für das k-Zentren Problem mit $\delta < 2$. Für eine Instanz (G, k) füttern wir A mit der transformierten Instanz (H, d, k). Wir erhalten

$$(G, k) \in \text{DOMINATING SET} \Leftrightarrow A \text{ gibt einen Wert kleiner als zwei aus}$$

und DOMINATING SET ist effizient berechenbar: Ein Widerspruch zur NP-Vollständigkeit. □

4.3 Lokale Suche für Spannbaumprobleme

Unser Ziel ist die Bestimmung von Spannbäumen mit möglichst vielen Blättern oder möglichst kleinem maximalen Grad.

> Wenn B ein Spannbaum ist und wenn wir eine neue Kante e zu B hinzufügen, dann wird genau ein Kreis geschlossen. Entfernen wir eine Kante e' des Kreises, dann ist auch $B' = B \setminus \{e'\} \cup \{e\}$ ein Spannbaum.

Wir definieren die Nachbarschaft von B für beide Probleme als die Menge aller Bäume, die sich durch die Hinzunahme und das Entfernen genau einer Kante aus B ergeben.

4.3.1 MAX LEAF SPANNING TREE

In MAX LEAF SPANNING TREE ist ein ungerichteter, zusammenhängender Graph G gegeben. Gesucht ist ein Spannbaum mit der größtmöglichen Blattzahl. Unser Algorithmus ist denkbar einfach:

Algorithmus 4.18 Lokale Suche für MAX LEAF SPANNING TREE

(1) Sei B ein beliebiger Spannbaum für den ungerichteten, zusammenhängenden Graphen $G = (V, E)$.

(2) Solange B einen Nachbarn B' mit mehr Blättern besitzt, ersetze B durch B'.

(3) Gib B aus.

Beachte, dass die Solange-Schleife in Schritt (2) höchstens $|V|$ mal durchlaufen wird, da die Anzahl der Blätter in jeder Iteration um mindestens eins ansteigt. Algorithmus 4.18 ist also effizient.

Satz 4.19 *Algorithmus 4.18 ist 5-approximativ für MAX LEAF SPANNING TREE.*

Beweis: Wir zeigen nur, dass Algorithmus 4.18 10-approximativ ist. Für einen Baum B sei $k_d(B)$ die Anzahl der Knoten in B vom Grad d. Beachte, dass die Anzahl der Blätter von B mit $k_1(B)$ übereinstimmt.

Behauptung: $\sum_{d\geq 3} k_d(B) < k_1(B)$.
Um die Behauptung nachzuweisen, beachte zuerst, dass ein Baum mit n Knoten genau $n-1$ Kanten besitzt. Also ist

$$\begin{aligned} 2\cdot(n-1) &= \sum_d d\cdot k_d(B) = k_1(B) + 2\cdot k_2(B) + \sum_{d\geq 3} d\cdot k_d(B) \\ &\geq k_1(B) + 2\cdot k_2(B) + 3\cdot\sum_{d\geq 3} k_d(B). \end{aligned}$$

Als Konsequenz, da $n = k_1(B) + k_2(B) + \sum_{d\geq 3} k_d(B)$

$$\begin{aligned} k_1(B) + 3\cdot\sum_{d\geq 3} k_d(B) &\leq 2\cdot(n - k_2(B) - 1) \\ &= 2\cdot(k_1(B) + k_2(B) + \sum_{d\geq 3} k_d(B) - k_2(B) - 1) \\ &= 2\cdot(k_1(B) + \sum_{d\geq 3} k_d(B) - 1). \end{aligned}$$

Die Behauptung ist eine unmittelbare Konsequenz der Ungleichung

$$k_1(B) + 3\cdot\sum_{d\geq 3} k_d(B) < 2\cdot(k_1(B) + \sum_{d\geq 3} k_d(B)).$$

□

Die Anzahl der Blätter eines jeden Spannbaum ist also größer als die Anzahl der Knoten mit mindestens drei Nachbarn. Jeder Spannbaum ohne Knoten vom Grad zwei ist somit 2-approximativ.
Sei B_{opt} ein Baum mit größtmöglicher Blattzahl und sei B der von Algorithmus 4.18 bestimmte Baum. Dann genügt es zu zeigen, dass höchstens $8\cdot k_1(B)$ Blätter von B_{opt}

Knoten vom Grad 2 im Baum B sind, denn dann hat B_{opt} höchstens $k_1(B) + 8 \cdot k_1(B) + k_1(B) = 10 \cdot k_1(B)$ Blätter.
Die $k_2(B)$ vielen Knoten vom Grad zwei in B können wir durch maximale 2-Wege in disjunkte Klassen zerlegen, wobei ein maximaler 2-Weg nur Knoten vom Grad zwei als innere Knoten durchläuft und in Knoten vom Grad eins oder mindestens drei endet. Wieviele 2-Wege (und damit wieviele Klassen) gibt es? Da maximale 2-Wege in Knoten vom Grad eins oder mindestens drei enden, ist ihre Anzahl durch $k_1(B) + \sum_{d \geq 3} k_d(B) \leq 2 \cdot k_1(B)$ beschränkt. Es genügt also zu zeigen, dass jeder maximale 2-Weg höchstens vier Blätter von B_{opt} als innere Knoten durchläuft.
Wir nutzen jetzt zum ersten Mal aus, dass B von Algorithmus 4.18 berechnet wurde. Wir nehmen an, dass ein maximaler 2-Weg W die Blätter v_1, v_2, v, v_2', v_1' von B_{opt} in dieser Reihenfolge durchläuft. Sei v' ein Knoten, der nicht von W durchlaufen wird. In B_{opt} gibt es genau einen Weg W' von v nach v'. Der Weg W' beginnt in v, durchläuft Knoten von W und springt dann von einem Knoten w zum ersten Knoten u, der nicht von W durchlaufen wird. Beachte, dass W' weder den Knoten v_2' noch den Knoten v_2 durchläuft, da beide Knoten Blätter in B_{opt} sind.
Wir fügen die Kante $\{u, w\}$ zu B hinzu. Die Hinzunahme erzeugt einen Kreis in B, der entweder die Knoten v_1', v_2' oder die Knoten v_2, v_1 durchläuft. Mit der Hinzunahme von $\{u, w\}$ verlieren wir höchstens das Blatt u, gewinnen aber durch das Entfernen einer mit v_2', bzw. v_2 inzidenten Kante zwei neue Blätter: Im Widerspruch zur Annahme ist B kein lokales Optimum. □

4.3.2 MIN DEGREE SPANNING TREE

In MIN DEGREE SPANNING TREE ist ein Spannbaum zu bestimmen, dessen maximaler Grad kleinstmöglich ist. Das Sprachenproblem

$$\{(G, d) \mid G \text{ hat einen Spannbaum mit maximalem Grad höchstens } d\ \}$$

ist $\mathcal{NP}$-vollständig: G hat genau dann einen Spannbaum mit maximalem Grad zwei, wenn G einen Hamiltonpfad besitzt, also einen Weg, der alle Knoten genau einmal besucht. (Die Frage, ob ein ungericheter Graph einen Hamiltonpfad besitzt, ist $\mathcal{NP}$-vollständig.)
Wie für MAX LEAF SPANNING TREE könnten wir wieder eine lokale Suche mit besseren, benachbarten Spannbäumen durchführen. Leider erhalten wir unbeschränkte Approximationsfaktoren.

Aufgabe 61
Bestimme einen zusammenhängenden Graphen G mit einem Spannbaum von maximalem Grad drei, und konstruiere ein lokales Optimum mit möglichst großem Grad.

Trotzdem ist lokale Suche erfolgreich, aber wir sind gezwungen, die schlechten lokalen Optima auszuschalten.

Definition 4.20 Sei B' ein Nachbar von B, es gelte also $B' = B \setminus \{u_1, u_2\} \cup \{u_3, u_4\}$. Dann ist B' etwas-besser als B, wenn der maximale Grad der vier beteiligten Knoten in B größer

als der maximale Grad in B' ist; mit anderen Worten, es gilt

$$\max\{\text{grad}_B(u_1), \text{grad}_B(u_2)\} > \max\{\text{grad}_B(u_3), \text{grad}_B(u_4)\} + 1.$$

Mit dem jetzt weitergefassten Begriff eines „etwas-besseren“ Nachbarn verschwinden einige der alten lokalen Optima, aber wir verringern den maximalen Grad möglicherweise nicht. Ist es deshalb nicht möglich, dass die lokale Suche ohne Fortschritte zu machen „im Kreis läuft“? Nein, das passiert nicht: Denke dir die Grad aller Knoten in B als absteigende Folge f_B aufgeschrieben und tu dasselbe für den etwas-besseren Nachbarn B' mit der absteigenden Folge $f_{B'}$. Dann ist f_B lexikographisch größer als $f_{B'}$ und ein „im Kreis laufen“ ist unmöglich.
Trotzdem ist die Anzahl aller möglichen Gradfolgen exponentiell groß und damit droht auch die Laufzeit unserer lokalen Suche zu explodieren. Deshalb verlangen wir, dass der maximale Grad der vier beteiligten Knoten in B' groß sein muss.
Im Folgenden bezeichnet $\text{grad}_H(v)$ den Grad des Knotens v im Graphen H.

Algorithmus 4.21 Lokale Suche für MIN DEGREE SPANNING TREE

(1) Sei B ein beliebiger Spannbaum für den ungerichteten, zusammenhängenden Graphen $G = (V, E)$.

(2) Solange B einen etwas-besseren Nachbarn $B' = B \setminus \{u_1, u_2\} \cup \{u_3, u_4\}$ besitzt und

$$\max\{\text{grad}_B(u_i) \mid 1 \leq i \leq 4\} \geq \max_v \text{grad}_B(v) - \log_2 |V|$$

gilt, ersetze B durch B'.

(3) Gib B aus.

Aufgabe 62
Zeige, dass die Laufzeit von Algorithmus 4.21 polynomiell in $|V|$ ist.

Unser Algorithmus ist recht gut.

Satz 4.22 *Der Graph $G = (V, E)$ habe n Knoten. Wenn Δ_{opt} der kleinstmögliche maximale Grad eines Spannbaums für G ist und Δ der maximale Grad des von Algorithmus 4.21 berechneten Spannbaums ist, dann gilt*

$$\Delta \leq 2\Delta_{\text{opt}} + \log_2 n + 1.$$

Beweis: Wir beginnen mit einer guten Abschätzung für den optimalen Grad. Wir sagen, dass (X, Π) ein Zeuge ist, wenn $X \subseteq V$ eine Knotenmenge und Π eine Zerlegung von V ist. Darüberhinaus müssen alle Kanten, die verschiedene Klassen von Π verbinden mit mindestens einem Knoten aus X inzident sein. Die Anzahl der Klassen der Zerlegung Π bezeichnen wir mit $\text{rank}(\Pi)$.

Behauptung 1: Für jeden Zeugen (X, Π) gilt

$$\Delta_{\text{opt}} \geq \frac{\text{rank}(\Pi) - 1}{|X|}. \tag{4.4}$$

Warum ist die Behauptung richtig? Sei B ein beliebiger Spannbaum. Wieviele Kanten sind in B mit einem Knoten aus X inzident? Um diese Frage zu beantworten, entferne alle mit Knoten aus X inzidenten Kanten aus B. Wir erhalten mindestens rank(Π) Zusammenhangskomponenten und B benötigt deshalb mindestens rank(Π) $-$ 1 mit Knoten aus X inzidente Kanten. Die Behauptung folgt, da der maximale Grad in B eines Knotens in X mindestens so groß wie der Durchschnittsgrad der Knoten in X ist. □

Wir suchen nach einem guten „Zeugen“ (X, Π), so dass die Abschätzung (4.4) den Grad Δ_{opt} möglichst gut approximiert. Offensichtlich sollten wir eine Menge X suchen, die aus möglichst wenigen Knoten besteht. Desweiteren, um möglichst viele Zusammenhangskomponenten zu erzeugen, sollte der Grad der Knoten in X möglichst groß sein.
Sei B der von Algorithmus 4.21 berechnete Spannbaum. Wir betrachten die Mengen

$$X_d = \{v \in V \mid \text{grad}_B(v) \geq d\}.$$

Wenn d der maximale Grad einen Knotens in B ist, dann folgt offensichtlich $X_d \subseteq X_{d-1} \subseteq \cdots \subseteq X_{d-\log_2 n}$. Wie stark können diese Mengen in Größe anwachsen? Wenn sich die Mengen in Größe stets mehr als verdoppeln, dann gilt $|X_{d-\log_2 n}| > 2^{\log_2 n} \cdot |X_d| \geq n$ und dies ist unmöglich. Also gibt es $d^* > d - \log_2 n$ mit $|X_{d^*-1}| \leq 2|X_{d^*}|$.
Sei Π die Zerlegung, die aus den Einermengen für alle Knoten in X_{d^*} und den Zusammenhangskomponenten von B nach Herausnahme der Menge X_{d^*} besteht.

Behauptung 2: (X_{d^*-1}, Π) ist ein Zeuge.

Wir müssen zeigen, dass alle Kanten, die verschiedene Klassen in Π verbinden, mit einem Knoten in X_{d^*-1} inzident sind. Dies ist offensichtlich, falls eine der Klassen eine Einermenge mit einem Knoten $v \in X_{d^*}$ ist, denn $X_{d^*} \subseteq X_{d^*-1}$.
Angenommen, die Kante $\{u_3, u_4\}$ verbindet zwei Zusammenhangskomponenten, die durch die Herausnahme der Knoten in X_{d^*} entstanden sind. Wenn weder u_3 noch u_4 in X_{d^*-1} liegt, dann ist $\max\{\text{grad}_B(u_3), \text{grad}_B(u_4)\} < d^* - 1$.
Betrachte alle Klassen-internen Kanten. Wenn wir die Klassen-verbindende Kante $\{u_3, u_4\}$ hinzufügen, dann wird kein Kreis geschlossen. Fügen wir jetzt die Knoten aus X_{d^*} und ihre inzidenten Kanten hinzu, dann erhalten wir B und zuzüglich die Kante $\{u_3, u_4\}$. Es muss ein Kreis geschlossen werden, der zwangsläufig eine Kante $\{u_1, u_2\}$ mit $u_1 \in X_{d^*}$ besitzen muss. Dann aber ist $B \setminus \{u_1, u_2\} \cup \{u_3, u_4\}$ ein etwas-besserer Nachbar und Algorithmus 4.21 stoppt nicht mit B. Wir haben einen Widerspruch zur Annahme erhalten, dass weder u_3 noch u_4 zu X_{d^*-1} gehören. □

Wie groß ist rank(Π)? Sei A_{d^*} die Anzahl der Kanten, die in B mit einem Knoten aus X_{d^*} inzident sind. Aber B ist ein Baum, und deshalb ist rank(Π) $= A_{d^*} + 1$. Es ist

$$A_{d^*} \geq d^* \cdot |X_{d^*}| - (|X_{d^*}| - 1),$$

denn alle Knoten in X_{d^*} haben mindestens d^* inzidente Kanten. Desweiteren besitzt der Baum B höchstens $|X_{d^*}| - 1$ Kanten, die zwei Knoten aus X_{d^*} verbinden und deshalb werden höchstens $|X_{d^*}| - 1$ Kanten zweifach gezählt.
Wir wenden Behauptung 2 an und erhalten

$$\Delta_{\text{opt}} \geq \frac{\text{rank}(\Pi) - 1}{|X_{d^*-1}|} \geq \frac{d^* \cdot |X_{d^*}| - |X_{d^*}| + 1}{|X_{d^*-1}|} \geq \frac{d^* \cdot |X_{d^*}| - |X_{d^*}| + 1}{2|X_{d^*}|} > \frac{d^* - 1}{2}.$$

Also folgt $d^* < 2 \cdot \Delta_{\text{opt}} + 1$. Wir wissen, dass $d^* \geq d - \log_2 n$ gilt und erhalten

$$d - \log_2 n \leq d^* < 2 \cdot \Delta_{\text{opt}} + 1.$$

Die Behauptung folgt. □

4.4 Lokale Suche in variabler Tiefe

Die lokale Suche erlaubt nur Abwärtsbewegungen und als Konsequenz wird eine Lösung kurz über lang in ein lokales Minimum mit großer Sogwirkung fallen. Wir betrachten deshalb die Variante der lokalen Suche in variabler Tiefe, die auch Aufwärtsbewegungen erlaubt.
Das allgemeine Prinzip der lokalen Suche in variabler Tiefe lässt sich für Optimierungsprobleme, deren Lösungsraum eine Teilmenge von $\{0,1\}^n$ ist, in der folgenden vereinfachten Version beschreiben. Wir nehmen dazu an, dass die k-Flip Nachbarschaft für eine kleine Konstante k gewählt wird.

Algorithmus 4.23 **Lokale Suche in variabler Tiefe**
y sei die gegenwärtige Lösung eines Optimierungsproblems mit der k-Flip Nachbarschaft. Der Lösungsraum sei eine Teilmenge von $\{0,1\}^n$. Eine bessere, aber nicht notwendigerweise benachbarte Lösung y' ist zu bestimmen.

(1) Initialisierungen: Setze EINGEFROREN $= \emptyset$, LÖSUNGEN $= \{y\}$ und $z = y$.

/* EINGEFROREN wird stets eine Menge von Positionen sein. LÖSUNGEN ist eine Menge von Lösungen, die während der Berechnung inspiziert werden. */

(2) Wiederhole, solange EINGEFROREN $\neq \{1, \ldots, n\}$

(2a) Bestimme eine beste Lösung $z' \neq z$ in der k-Flip Nachbarschaft von z. z und z' dürfen sich allerdings nicht in Positionen $i \in$ EINGEFROREN unterscheiden.

(2b) Friere alle im Wechsel von z nach z' geänderten Positionen ein, füge z' zu LÖSUNGEN hinzu und setze $z = z'$.

/* Beachte, dass z' durchaus eine schlechtere Lösung als z sein darf. Wir erlauben also Aufwärtsbewegungen. Durch das Einfrieren geänderter Positionen wird die Schleife höchstens n Mal durchlaufen. */

(3) Gib die beste Lösung in LÖSUNGEN als y' aus.

Beispiel 4.7 MINIMUM BALANCED CUT
Zur Erinnerung: In MINIMUM BALANCED CUT müssen wir die Knotenmenge eines ungerichteten Graphen so in zwei gleich große Teile zerlegen, dass möglichst wenige Kanten kreuzen. Wir beschreiben den Kernighan-Lin Algorithmus.

Algorithmus 4.24 **Der Kernighan-Lin Algorithmus**

(1) Die Eingabe besteht aus dem ungerichteten Graphen $G = (V, E)$. Beginne mit einer Menge $W \subseteq V$ mit $|W| = \frac{|V|}{2}$. Anfangs sind keine Knoten eingefroren und wir setzen $i = 0$, $W_0 = W$ und $g_0 = 0$.

(2) Wiederhole, solange nicht alle Knoten eingefroren sind:

(2a) Bestimme für alle nicht eingefrorenen Knoten $w \in W$ und $w^* \in V \setminus W$,

$$\text{gewinn}(w, w^*) = f(W_i) - f(W_i \setminus \{w\} \cup \{w^*\}),$$

den „Gewinn“ nach Ersetzung von w durch w^*.

(2b) Bestimme den 1-Flip (w, w^*) mit größtem Gewinn und setze $W_{i+1} = W_i \setminus \{w\} \cup \{w^*\}$. Friere w und w^* ein.

Kommentar: Das Einfrieren garantiert, dass der Algorithmus terminiert.

(2c) Setze $g_{i+1} = \text{gewinn}(w, w^*)$ und $i = i + 1$.

(3) Bestimme i, so dass $\sum_{k=1}^{i} g_k$ maximal ist und gib W_i als „verbesserten Nachbarn“ von W aus.

Kommentar: Kernighan-Lin erlaubt somit Uphill-Bewegungen bei negativem g_i. Wesentlich ist, dass der kumulative Gewinn $\sum_{k=1}^{i} g_k$ positiv ist. Das in der Praxis beobachtete Approximationsverhalten der Heuristik ist gut.

Das Traveling Salesman Problem ist eine weitere Anwendung der Strategie der lokalen Suche in variabler Tiefe.

4.5 Der Metropolis Algorithmus und Simulated Annealing

Wir betrachten wieder das allgemeine Minimierungsproblem $P = (\min, f, L)$ und nehmen an, dass für jede Lösung eine Nachbarschaft $\mathcal{N}(y)$ von Lösungen definiert ist. Wir möchten auch diesmal Aufwärtsbewegeungen zulassen, aber werden dies nur widerstrebend tun. Die Bereitschaft schlechtere Nachbarn zuzulassen wird durch den Temperatur-Parameter T gesteuert: Je höher die Temperatur, umso höher die Wahrscheinlichkeit, dass ein schlechterer Nachbar akzeptiert wird.

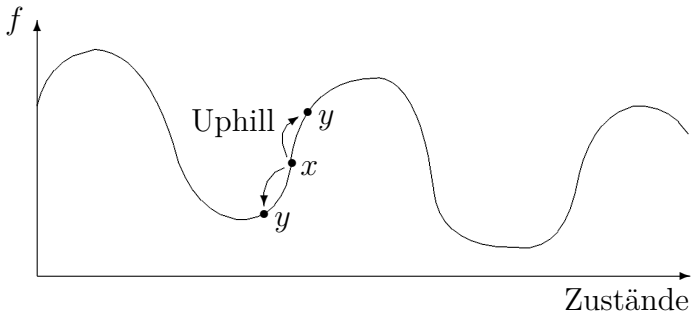
Algorithmus 4.25 **Der Metropolis Algorithmus**

(1) Sei y eine Anfangslösung und sei T die Temperatur.

(2) Wiederhole hinreichend oft:

(2a) Wähle zufällig einen Nachbarn $y' \in \mathcal{N}(y)$.

(2b) Wenn $f(y') \leq f(y)$, dann akzeptiere y' und setze $y = y'$. Ansonsten setze $y = y'$ mit Wahrscheinlichkeit $e^{-\frac{f(y')-f(y)}{T}}$.
/* Je schlechter der Wert des neuen Nachbarn y', umso geringer die Wahrscheinlichkeit, dass y' akzeptiert wird. Schlechte Nachbarn haben nur eine Chance bei entsprechend hoher Temperatur. */



Beachte, dass bei hoher Temperatur T die Ersetzung von x durch einen Nachbarn auch dann wahrscheinlich ist, wenn dieser Nachbar y einen großen Wert $f(y)$ hat, denn für $f(y) - f(x) \ll T$ ist:

$$e^{-\frac{f(y)-f(x)}{T}} \approx 1 - \frac{f(y) - f(x)}{T}$$

Bei hoher Temperatur durchläuft der Metropolis Algorithmus den Lösungsraum mit jeweils zufälliger Nachbarwahl. Bei geringer Temperatur wird eine Verschlechterung hingegen nur sehr widerstrebend in Kauf genommen.
Mit welcher Wahrscheinlichkeit wird die Lösung y besucht, wenn der Metroplis-Algorithmus genügend häufig wiederholt wird? Das folgende Ergebnis kann gezeigt werden:

Satz 4.26 *$f_y(t)$ bezeichne die relative Häufigkeit, mit der der Metropolis-Algorithmus die Lösung y in den ersten t Schritten besucht. Dann gilt*

$$\lim_{t\to\infty} f_y(t) = \frac{e^{-f(y)/T}}{Z}, \ \textit{wobei} \quad Z = \sum_{y, L(y)} e^{-f(y)/T}.$$

Auf den ersten Blick ist dies eine hervorragende Eigenschaft des Metroplis-Algorithmus, da die Wahrscheinlichkeit eines Besuchs für gute Lösungen y am größten ist. Allerdings ist Konvergenzgeschwindigkeit in vielen Anwendungen leider sehr langsam.

Beispiel 4.8 Wir greifen Beispiel 9.1 wieder auf, wenden diesmal aber nicht die lokale Suche, sondern den Metropolis Algorithmus auf VERTEX COVER an. Wir betrachten wieder den leeren Graphen $G = (V, \emptyset)$ und beginnen den Metropolis Algorithmus mit V als Knotenmenge. Anfänglich wird die Knotenmenge nur reduziert und Metropolis verhält sich wie die lokale Suche. Das Problem beginnt, wenn die gegenwärtige Lösung y nur noch wenige Knoten besitzt: y hat sehr viel mehr schlechtere als bessere Nachbarn und dementsprechend werden schlechtere Nachbarn mit sehr viel höherer Wahrscheinlichkeit gewählt. Die Akzeptanzwahrscheinlichkeit einer schlechteren Lösung ist nur unwesentlich kleiner und schlechtere Lösungen, wenn nicht im ersten oder zweiten Anlauf, werden kurz über lang gewählt: Der Metropolis-Algorithmus bekommt Angst vor der eigenen Courage, wenn gute, aber längst nicht optimale Lösungen erreicht werden.
Betrachten wir den Sterngraphen als zweites Beispiel. Hier zeigt der Metropolis-Algorithmus seine Stärke. Selbst wenn das Zentrum irgendwann entfernt wird, so ist die Wahrscheinlichkeit hoch, dass das Zentrum nach nicht zu langer Zeit wieder betrachtet wird. Mit beträchtlicher Wahrscheinlichkeit wird die neue Lösung akzeptiert und die schlechte Entscheidung, das Zentrum zu entfernen, wird revidiert. Danach, wenn mindestens ein Satellit nicht in der jeweiligen Lösung liegt, bleibt das Zentrum erhalten, da sonst keine Knotenüberdeckung vorliegt. Allerdings hat Metropolis auch für den Sterngraphen Angst vor der eigenen Courage, wenn die Überdeckung genügend klein ist.

Das obige Beispiel legt nahe, dass wir versuchen sollten, die Temperatur vorsichtig zu senken, um Aufwärtsbewegungen nach entsprechend langer Suchzeit signifikant zu erschweren. Genau dieses Vorgehen wird in dem Simulated Annealing Verfahren durchgeführt. Die folgende Analogie aus der Physik erklärt das Verfahren: Erhitzt man einen festen Stoff so stark, dass er flüssig wird und läßt man ihn dann wieder langsam abkühlen, so ist der Stoff bestrebt, möglichst wenig der zugeführten Energie zu behalten. Der Stoff bildet eine Kristallgitterstruktur (Eiskristalle sind ein Beispiel.) Je behutsamer nun das Ausglühen (*engl.: Annealing*) durchgeführt wird, umso reiner ist die Gitterstruktur; Unregelmäßigkeiten in der Gitterstruktur, die durch zu rasches Abkühlen entstehen, stellen lokale Energieminima dar.

Algorithmus 4.27 **Simulated-Annealing**

(1) Sei y die Anfangslösung und sei T die Anfangstemperatur.

(2) Wiederhole hinreichend oft:

(2a) Wähle zufällig einen Nachbarn $y' \in \mathcal{N}(y)$.

(2b) Wenn $f(y') \leq f(y)$, dann akzeptiere y' und setze $y = y'$. Ansonsten setze $y = y'$ mit Wahrscheinlichkeit $e^{-\frac{f(y)-f(x)}{T}}$.

(3) Wenn die Temperatur noch nicht genügend tief ist, dann wähle eine neue, niedrigere Temperatur T und führe Schritt (2) aus. Ansonsten gib die erhaltene Lösung y aus.

Die Wahl des Abkühlprozesses ist problemabhängig und erfordert experimentelle Arbeit. Aber selbst dann kann keine Garantie gegeben werden, dass eine gute Lösung auch gefunden wird: Wird zum falschen Zeitpunkt abgekühlt, bleibt man in einem lokalen Optimum gefangen und verliert die Möglichkeit zu entkommen.

Beispiel 4.9 **MINIMUM BALANCED CUT**
Um Simulated-Annealing auf MINIMUM BALANCED CUT anzuwenden, lässt man beliebige Zerlegungen $(W, V \setminus W)$ zu und wählt

$$f(W) := |\{e \in E \mid |\{e\} \cap W| = 1\}| + \underbrace{\alpha \cdot (|W| - |V \setminus W|)^2}_{\text{Strafterm}}$$

als zu minimierende Zielfunktion. Wir versuchen durch den Strafterm, eine perfekte Aufteilung, also $|W| = \frac{1}{2} \cdot |V|$, zu erzwingen.
Die Lösungsmenge ist die Potenzmenge von V. Als Startlösung für Simulated-Annealing wählen wir ein perfekte, zufällig gewählte Aufteilung. Für eine Knotenteilmenge $W \subseteq V$ definieren wir die Umgebung von W als

$$\mathcal{N}(W) := \{W' \subseteq V \mid |W \oplus W'| \leq 1\},$$

also als die 1-Flip Nachbarschaft von W. In [D.S. Johnson. C.R. Aragon. L.A.McGeoch und C.Schevon (1989): **Simulated Annealing: An experimental Evaluation, Part I: Graph Partitioning**, Operation Research, Band 37, Nr.6, Seiten 865-892.] wird die Anfangstemperatur so gewählt, dass 40% aller Nachbarn akzeptiert werden. Die Temperatur wird über den Zeitraum von $16 \cdot |V|$ konstant gehalten und dann bei jeder Wiederholung von Schritt (3) um den Faktor $0,95$ gesenkt („geometrische Abkühlung“). Bei *zufällig gewählten* Graphen schneidet Simulated-Annealing (erstaunlicherweise?) erfolgreicher ab als maßgeschneiderte Algorithmen wie Algorithmus 4.23. Dieses Phänomen tritt auch auf, wenn wir den maßgeschneiderten Algorithmen die gleiche (große) Laufzeit wie der Simulated-Annealing-Methode erlauben.
Weitere Experimente wurden für *strukturierte* Graphen durchgeführt. 500 (bzw. 1000) Punkte werden zufällig aus dem Quadrat $[0, 1]^2$ gewählt und zwei Punkte werden verbunden, wenn sie nahe beieinander liegen. Für diese Graphenklasse „bricht“ die Simulated-Annealing-Methode ein und Algorithmus 4.23 ist überlegen. Eine mögliche Erklärung für das unterschiedliche Abschneiden wird in der unterschiedlichen Struktur der lokalen Minima liegen. Die geometrisch generierten Graphen werden tiefe Minima mit weitaus größeren Anziehungsbereichen als die zufällig generierten Graphen haben. Diese „Sogwirkung“ lokaler Minima ist bei den geometrisch generierten Graphen schwieriger als für Zufallsgraphen zu überwinden.

Bemerkung 4.2 Simulated Annealing wie auch der Metropolis-Algorithmus sind lokale Suchverfahren, die Uphillbewegungen erlauben. Ihr großer Vorteil, nämlich die Anwendbarkeit auf eine große Klasse von Problemen, ist allerdings auch ihre große Schwäche: Eigenschaften des vorliegenden Optimierungsproblems können nur durch die Wahl des Umgebungsbegriffs und durch die Temperaturregelung ausgenutzt werden.

Wenn genügend Laufzeit investiert werden kann, dann ist eine Anwendung dieser randomisierten Suchverfahren als ein *erster* Schritt in der Untersuchung eines Optimierungsproblems sicherlich zu empfehlen.

4.6 Evolutionäre Algorithmen

Für Simulated Annealing haben wir das sorgfältige Abkühlen eines verflüssigten Stoffes in Verbindung mit der Lösung eines Minimierungsproblems gebracht: Das Erreichen einer reinen Kristallgitterstruktur entspricht einem globalen Minimum, während Unreinheiten lokalen Minima entsprechen. Diesmal untersuchen wir Maximierungsprobleme[3] $(\max, f, L)$ und wählen die Evolution als Vorbild. Wie können wir das Erfolgsprinzip der Evolution, „Survival of the Fittest“, in einen Approximationsalgorithmus umsetzen?
Es liegt nahe, die Fitness eines Individuums (oder einer Lösung) y mit dem Funktionswert $f(y)$ gleichzusetzen. Weiterhin sollten wir versuchen, eine gegebene Population von Lösungen zu verbessern, indem wir Lösungen „mutieren“ oder zwei oder mehrere Lösungen miteinander „kreuzen“. Der Vorteil evolutionärer Algorithmen liegt in der systematischen Suche des Lösungsraums mit Hilfe sorgfältig ausgewählter Populationen. Allerdings lassen sich Mutations- und Crossover-Operatoren selten problem-spezifisch bestimmen, mit der Konsequenz, dass die Effektivität von auf das jeweilige Problem maßgeschneiderten Strategien nicht erreicht wird.

Algorithmus 4.28 **Die Grobstruktur eines evolutionären Algorithmus**

(1) **Initialisierung:** Eine Anfangspopulation P von μ Lösungen ist zu bestimmen.

Verschiedene Methoden kommen zum Ansatz wie etwa die zufällige Wahl von Lösungen oder die sorgfältige Wahl von Anfangslösungen über problem-spezifische Heuristiken. Im letzten Fall muss aber *Diversität* gewährleistet sein: Die Anfangspopulation sollte den gesamten Lösungsraum „repräsentieren“.

(2) Wiederhole, bis eine genügend gute Lösung gefunden wurde:

(2a) **Selektion zur Reproduktion:** Jede Lösung $y \in P$ wird mit ihrer Fitness $f(y)$ bewertet. Um Nachkommen der Population P zu erzeugen, wird zuerst eine Teilmenge $P' \subseteq P$ von Elternlösungen ausgewählt. Die Auswahlverfahren sind häufig randomisiert, um Diversität sicher zu stellen. Zu den häufig benutzten Auswahlverfahren gehören

- die zufällige Auswahl nach der Gleichverteilung: Jede Lösung in P wird mit gleicher Wahrscheinlichkeit gewählt,

[3] Natürlich können wir auch Maximierungsprobleme mit Simulated Annealing und Minimierungsproblem mit evolutionären Algorithmen lösen: Die Maximierung von f ist äquivalent zur Minimierung von $-f$.

- die zufällige Auswahl nach der Fitness: Falls f positiv ist, wird $y \in P$ mit Wahrscheinlichkeit $\frac{f(y)}{N}$ für $N = \sum_{z \in P} f(z)$ oder mit Wahrscheinlichkeit $\frac{e^{f(x)/T}}{M}$ für $M = \sum_{z \in P} e^{f(z)/T}$ und einen Temperaturparameter T gewählt oder die
- Turnier-Selektion: Wähle zuerst k Lösungen aus P nach der Gleichverteilung und lasse die k' fittesten der k ausgewählten Lösungen zu.

(2b) **Variation:** μ Nachkommen werden aus P' mit Hilfe von Mutations- und Crossover-Operatoren erzeugt.

(2c) **Selektion zur Ersetzung:** Die neue Generation ist festzulegen.

Wende Verfahren wie in Schritt (2a) an, um die neue Generation aus der alten Generation und ihren Nachkommen zu bestimmen. Zum Beispiel werden in der Plus-Auswahl die μ Fittesten aus den μ Lösungen der alten Generation und ihren λ Nachkommen bestimmt; man spricht von der $(\mu + \lambda)$-Selektion. In der Komma-Auswahl wird $\lambda \geq \mu$ angenommen, und die μ fittesten Nachkommen bilden die neue Generation; man spricht von der (μ, λ)-Selektion.

Bevor wir die Mutations- und Crossover-Operatoren beschreiben, schränken wir die Suchräume ein. Im Wesentlichen werden drei verschiedene Typen von Suchräumen betrachtet. Neben dem n-dimensionalen Würfel $\mathbb{B}_n = \{0,1\}^n$ werden vor Allem der $\mathbb{R}^n$ sowie $\mathbb{S}^n$, der Raum aller Permutationen von n Objekten, betrachtet.

- Mutationsoperatoren:
  - Wir beginnen mit **Bitmutationen** für den n-dimensionalen Würfel $\mathbb{B}^n$ Entweder flippt man jedes Bit einer Elternlösung $y \in \mathbb{B}^n$ mit einer kleinen Wahrscheinlichkeit p (mit $p \cdot n = \Theta(1)$) oder man ersetzt y durch einen Nachbarn y' in der k-Flip Nachbarschaft von y für kleine Werte von k.
  - Im $\mathbb{R}^n$ ersetzt man eine Elternlösung y durch $\mathbf{y}' = \mathbf{y} + \mathbf{m}$, wobei die Komponenten von m zufällig und unabhängig voneinander mit Erwartungswert 0 gewählt werden. Entweder man wählt die Komponenten m_i von m zufällig aus einem fixierten Intervall $[-a, a]$ oder man erlaubt unbeschränkte Komponenten, die zum Beispiel gemäß der Normalverteilung gezogen werden. Im letzten Fall erlaubt man eine hohe Standardabweichung bei schlechten Lösungen und reduziert die Standardabweichung je besser die Lösung ist.
  - Für den Permutationsraum $\mathbb{S}^n$ betrachtet man zum Beispiel **Austausch-** und **Sprungoperatoren**. In einem Austauschoperator wird ein Paar (i, j) mit $1 \leq i \neq j \leq n$ zufällig und gleichverteilt aus allen möglichen Paaren gezogen und die $i$te und $j$te Komponente der Elternlösung werden vertauscht. Auch für einen Sprungoperator wird ein Paar (i, j) mit $1 \leq i \neq j \leq n$ zufällig und gleichverteilt aus allen möglichen Paaren gezogen. Die $i$te Komponente y_i einer Elternlösung y wird an die Position j gesetzt und die restlichen Komponenten werden passend verschoben.

- Crossover-Operatoren: $y_1, \ldots, y_k$ seien k Eltern, wobei $k \geq 2$ gelte.

  - Für den Würfel betrachtet man nur den Fall $k = 2$. Im **r-Punkt Crossover** wählt man r Positionen $i_1, \ldots, i_r$ mit $1 \leq i_1 < \cdots < i_r \leq n$. Der Sprößling z von y_1 und y_2 erbt die ersten i_1 Bits von y_1, die $i_2 - i_1$ Bits in den Positionen $[i_1 + 1, i_2]$ von y_2, die $i_3 - i_2$ Bits in den Positionen $[i_2 + 1, i_3]$ von y_1 und so weiter. Typischerweise wird $r = 1$ oder $r = 2$ gewählt.
  - Auch im $\mathbb{R}^n$ verwendet man den r-Punkt Crossover. Sehr verbreitet ist das **arithmetische Crossover**, in dem $\sum_{i=1}^{k} \alpha_i y_i$ zum Nachkommen von $y_1, \ldots, y_k$ wird. Der wichtige Spezialfall $\alpha_1 = \cdots = \alpha_k = \frac{1}{k}$ wird intermediäre Rekombination genannt.
  - Im Permutationsraum $\mathbb{S}^n$ wählt man meistens 2 Eltern y_1 und y_2. Zum Beispiel werden Positionen $1 \leq i_1 < i_2 \leq n$ zufällig ausgewürfelt. Die in den Positionen i_1 bis i_2 liegenden Komponenten von y_1 werden dann gemäß y_2 umgeordnet.

Beispiel 4.10 **Einfache Genetische Algorithmen**
Der evolutionäre Algorithmus mit

- Suchraum $\mathbb{B}^n$,
- zufälliger Auswahl nach Fitness für die Selektion zur Reproduktion,
- Bitmutation mit Wahrscheinlichkeit $p \leq \frac{1}{n}$ und 1-Punkt-Crossover sowie
- (μ, μ)-Selektion zur Ersetzung

wird ein einfacher genetischer Algorithmus genannt. Hier erzeugt man häufig zwei Nachkommen, wobei der zweite Nachkomme genau die Elternanteile erbt, die der erste Nachkomme nicht erhalten hat.
Die Wahrscheinlichkeit, dass ein Nachfahre durch Crossover erzeugt wird, liegt zwischen $0,5$ und $0,9$; die Mutation gilt hier als weniger wichtiger Hintergrundoperator. Andere evolutionäre Algorithmen betonen hingegen die Mutation.

Beispiel 4.11 Für das Traveling Salesman Problem betrachtet man auch die folgende Crossover-Operation:

> Wähle zwei Rundreisen T_1 und T_2 der gegenwärtigen Population sowie eine Teiltour T_1' von T_1. Setze T_1' in T_2 ein, so dass die Reihenfolge der nicht in T_1' aufgeführten Punkte in T_2 unverändert bleibt.

Zum Beispiel, wenn $T_1' = (1, 9, 3, 7)$ und $T_2 = (9, 8, 1, 2, 7, 6, 3, 4, 5)$, dann ist

$$\not{9}, 8, \underbrace{1, 9, 3, 7}_{T_1'}, \not{1}, 2, \not{7}, 6, \not{3}, 4, 5$$

das Crossover von T_1' und T_2.

Bemerkung 4.3 Wir können auch Entscheidungsprobleme mit evolutionären Algorithmen lösen. Dazu betrachten wir exemplarisch das Entscheidungsproblem $KNF - SAT$. Hier bietet sich der Suchraum $\mathbb{B}^n$ aller Belegungen an. Wir transformieren $KNF - SAT$ in ein Optimierungsproblem, indem wir die Zahl erfüllter Klauseln maximieren.

Kapitel 5

Local Ratio

Für ein Minimierungsproblem mit Instanz x sei $A(x)$ der Wert für einen Approximationsalgorithmus A und $\text{opt}(x)$ sei der optimale Wert. Wenn A r-approximativ ist, dann gilt

$$\frac{A(x)}{\text{opt}(x)} \leq r$$

für jede Instanz x. Für die Analyse von A verwendet man häufig eine untere Schranke $u(x) \leq \text{opt}(x)$ und zeigt

$$\frac{A(x)}{\text{opt}(x)} \leq \frac{A(x)}{u(x)} \stackrel{!}{\leq} r.$$

Der globale Quotient $\frac{A(x)}{u(x)}$ ist durch r beschränkt. Die „Local Ratio"-Methode versucht, die „Kosten" $A(x)$ als Summen $A(x) = c_1 + \cdots + c_k$ partieller Kosten aufzufassen. Wenn auch die untere Schranke als Summe $u(x) = u_1 + \cdots + u_k$ mit der Eigenschaft $c_i \leq r \cdot u_i$ aufgefasst werden kann, dann folgt

$$\begin{aligned} A(x) - r \cdot u(x) &= c_1 + \cdots c_k - r \cdot [u_1 + \cdots + u_k] \\ &= \sum_{i=1}^{k} (c_i - r \cdot u_i) \leq 0, \end{aligned}$$

und der globale Quotient ist durch r beschränkt, weil die lokalen Quotienten durch r beschränkt sind.

5.1 VERTEX COVER

Der ungerichtete Graph $G = (V, E)$ sei gegeben ebenso wie eine Gewichtung $w_u \geq 0$ der Knoten $u \in V$. Wir suchen eine leichteste Knotenüberdeckung $V' \subseteq V$: Jede Kante muss mindestens einen Endpunkt in der Menge V' besitzen und das Gewicht $\sum_{u \in V'} w_u$ muss kleinstmöglich sein.

Algorithmus 5.1 Local Ratio für VERTEX COVER

(1) Solange es eine Kante $\{u, v\} \in E$ mit $\min\{w_u, w_v\} > 0$ gibt:

   (a) Setze $\varepsilon = \min\{w_u, w_v\}$,

   (b) und $w_u = w_u - \varepsilon$, $w_v = w_v - \varepsilon$.

(2) Gib $V' = \{u \mid w_u = 0\}$ aus.

Wir arbeiten also alle Kanten in beliebiger Reihenfolge ab. Wenn wir irgendwann die Kante $\{u, v\}$ verarbeiten, dann haben wir bereits die Kosten $c_1 + \cdots + c_i$ angehäuft, denen die unteren Schranken $u_1 + \cdots + u_i$ gegenüberstehen. Im $i+1$ten Schritt bezahlen wir die Kosten $c_{i+1} = 2 \cdot \min\{w_u, w_v\}$, wobei die Knotengewichte gemäß den bisherigen Anzahlungen reduziert wurden. Andererseits verursacht jede Knotenüberdeckung mindestens die Kosten $u_{i+1} = \min\{w_u, w_v\}$. Wir haben einen 2-approximativen Algorithmus erhalten.

Satz 5.2 *Algorithmus 5.1 ist 2-approximativ und läuft in Zeit $O(|V| + |E|)$.*

Hätten wir denn nicht noch einfacher vorgehen können und stets den billigsten Endpunkt wählen können ohne vorher Anzahlungen zu machen? Nein! Betrachte den Sterngraphen mit Sternzentrum s_0 und Satelliten $s_1, \ldots, s_n$. Wir nehmen an, dass s_0 das Gewicht $1 < \Delta < 2$ und die Satelliten das jeweilige Gewicht 1 besitzen. Unser neuer Ansatz wählt alle Satelliten (mit einem Gesamtgewicht von n), während das Sternzentrum (mit einem Gewicht von Δ) eine optimale Knotenüberdeckung ist. Beachte, dass Algorithmus 5.1 das Sternzentrum im zweiten Schritt wählt, da die Anzahlung im ersten Schritt das Gewicht von s_0 auf kleiner als Eins gesenkt hat. Algorithmus 5.1 berechnet also eine Knotenüberdeckung mit Gewicht $1 + \Delta < 2 \cdot \Delta$. Für $\Delta = 1 + \varepsilon$ und $\varepsilon > 0$ beliebig klein ist der Approximationsfaktor beliebig nahe bei zwei und die Analyse von Satz 5.2 lässt sich somit nicht verbessern.
Warum ist Algorithmus 5.1 erfolgreich? Für jede zu überdeckende Kante wird sichergestellt, dass die tatsächlichen Kosten plus Anzahlung die optimalen Kosten um höchstens den Faktor zwei übertreffen. Dazu haben wir die Kosten wie auch die unteren Schranke als Summe über alle zu überdeckenden Kanten aufgefasst.

Aufgabe 63
Im Problem HITTING SET sind Teilmengen $T_1, \ldots, T_k \subseteq U$ eines Universums U gegeben wie auch eine Gewichtung $w_u \geq 0$ für alle Elemente $u \in U$. Gesucht ist eine leichteste Teilmenge $U' \subseteq U$, die jede Teilmenge T_i in mindestens einem Element trifft.
Es sei $g = \max\{|T_i| \mid 1 \leq i \leq k\}$. Beschreibe einen g-approximativen Algorithmus für HITTING SET.

Aufgabe 64
Im SET COVER Problem sind ein Universum U und Teilmengen $T_1, \cdots, T_k$ mit den Gewichten $w_1, \ldots, w_k \geq 0$ gegeben. Die Teilmengen überdecken das Universum, d.h. es gilt $\bigcup_{i=1}^{k} T_i = U$. Unser Ziel ist eine Überdeckung mit möglichst geringem Gesamtgewicht.
Zeige, dass sich SET COVER als ein HITTING SET Problem auffassen lässt.

Im PRICE COLLECTING VERTEX COVER ist ein ungerichteter Graph $G = (V, E)$ gegeben sowie Gewichtungen $w_u \geq 0$ der Knoten $u \in V$ und w_e der Kanten $e \in E$.

Lösungen sind Teilmengen $V' \subseteq V$, wobei die Kosten von V' durch

$$\sum_{u \in V'} w_u + \sum_{e=\{u,v\} \in E;\ u \notin V', v \notin V'} w_e$$

beschrieben werden: Eine optimale Lösung minimiert die Summe ihres Gesamtgewichts und des Gesamtgewichts aller nicht überdeckten Kanten. Beachte, dass VERTEX COVER als ein Spezialfall von PRICE COLLECTING VERTEX COVER aufgefasst werden kann, wenn wir die Kantenkosten w_e hinreichend hoch setzen.

Algorithmus 5.3 Local Ratio für PRICE COLLECTING VERTEX COVER

(1) Solange es eine Kante $e = \{u, v\} \in E$ mit $\min\{w_u, w_v, w_e\} > 0$ gibt:

   (a) Für $\varepsilon = \min\{w_u, w_v, w_e\}$ setze

   (b) $w_u = w_u - \varepsilon$, $w_v = w_v - \varepsilon$ und $w_e = w_e - \varepsilon$.

(2) Gib $V' = \{u \mid w_u = 0\}$ aus.

Was passiert, wenn die Kante $e = \{u, v\}$ zu überdecken ist? Wir bezahlen 3ε für die Reduktion der Gewichte w_u, w_v und w_e. Eine optimale Entscheidung verursacht aber mindestens die Kosten ϵ, denn entweder wird mindestens einer der Knoten u oder v aufgenommen oder e wird nicht überdeckt. Wir haben also bereits nachgewiesen, dass Algorithmus 5.3 mindestens 3-approximativ ist. Tatsächlich gilt aber:

Satz 5.4 *Algorithmus 5.3 ist 2-approximativ.*

Aufgabe 65
Zeige Satz 5.4.

5.2 Das Local Ratio Theorem

Wir haben in VERTEX COVER und seinen Varianten leicht argumentieren können, dass der Algorithmus nur dann Kosten von 2ε einfährt, wenn das Optimum um mindestens ε abnimmt. Wir geben später eine 2-approximative Lösung für das FEEDBACK VERTEX SET Problem für ungerichtete Graphen:

> Für einen ungerichteten Graphen $G = (V, E)$ mit einer Gewichtung $w_u \geq 0$ der Knoten $u \in V$ bestimme eine leichteste Knotenmenge $V' \subseteq V$, deren Herausnahme alle Kreise zerstört.

In diesem Problem können wir sicherlich nicht erwarten, dass wir immer zwei Knoten finden, so dass einer mit Sicherheit zu einer optimalen Lösung gehört. Das Local Ratio Theorem wird uns mehr Flexibilität geben.
Wir müssen uns auf die Optimierungsprobleme der Form (w, L) einschränken:

Für $w \in \mathbb{R}^n$ und eine Lösungsmenge $L \subseteq \mathbb{R}^n$ bestimme $x \in L$, so dass das innere Produkt $w^T \cdot x = \sum_{i=1}^{n} w_i \cdot x_i$ möglichst klein ist.

Satz 5.5 *Das Local Ratio Theorem*
Es gelte $w = w_1 + w_2$. Wenn x eine r-approximative Lösung für (w_1, L) und (w_2, L) ist, dann ist x auch r-approximativ für (w, L).

Beweis: Seien x^*, x_1^*, x_2^* optimale Lösungen von $(w, L), (w_1, L)$ und (w_2, L). Dann gilt offensichtlich $w_1^T \cdot x_1^* \leq w_1^T \cdot x^*$ wie auch $w_2^T \cdot x_2^* \leq w_2^T \cdot x^*$. Also folgt

$$w^T \cdot x = w_1^T \cdot x + w_2^T \cdot x \leq r \cdot (w_1^T \cdot x_1^* + w_2^T \cdot x_2^*) \leq r \cdot (w_1^T \cdot x^* + w_2^T \cdot x^*) = r w^T \cdot x^*$$

aus $w_i^T \cdot x \leq r w_i^T \cdot x_i^*$ für $i = 1, 2$. □

Statt iterativer Algorithmen wie die Algorithmen 5.1 und 5.3 drängen sich jetzt rekursive Algorithmen auf:

(0) Es gelte $w \geq 0$.

(1) Wenn es eine Lösung $x \in L$ mit $x_i = 0$ für alle Komponenten i mit $w_i > 0$ gibt, dann gib x aus.

(2) Sonst bestimme eine Darstellung $w = w' + (w - w')$ mit $w' \leq w$, so dass *jede* Lösung r-approximativ für w' ist.

Kommentar: Später werden wir nur noch fordern, dass *jede nicht-verkleinerbare* Lösung r-approximativ für w' ist.

(3) Bestimme rekursiv eine Lösung $x \in L$ für $(w - w', L)$.

Wir können Algorithmus 5.1 in dieses Format bringen, wenn wir in Schritt (2) eine Kante $\{u, v\}$ mit $\varepsilon = \min\{w_u, w_v\} > 0$ wählen und $w'_x = \varepsilon$ für $x \in \{u, v\}$ und sonst $w'_x = 0$ setzen. Beachte, dass jede Knotenüberdeckung für die Gewichtung w' mindestens 2-approximativ ist. Die rekursive Version von Algorithmus 5.1 ist damit 2-approximativ als Konsequenz des Local Ratio Theorems, wenn wir $w_1 = w'$ und $w_2 = w - w'$ setzen.

Beispiel 5.1 Im FEEDBACK VERTEX SET Problem für Turniere ist ein Turniergraph $G = (V, E)$ gegeben, wobei G für je zwei Knoten u, v genau eine der beiden Kanten (u, v), bzw. (v, u) besitzt. (Ein Turniergraph gibt also alle Spielergebnisse wieder, wenn jeder Spieler gegen jeden anderen Spieler antreten muss.) Für eine Gewichtung $w_u \geq 0$ der Knoten $u \in V$ ist eine leichteste Knotenmenge $V' \subseteq V$ gesucht, so dass G nach Herausnahme aller Knoten in V' kreisfrei ist.

Lemma 5.6 *Ein Turniergraph G besitzt genau dann einen Kreis, wenn G ein Dreieck, also einen Kreis der Länge drei besitzt.*

Beweis: Natürlich besitzt G einen Kreis, wenn G ein Dreieck besitzt. Wir müssen zeigen, dass G ein Dreieck besitzt, wenn in G ein Kreis vorkommt. Sei $(v_1, \ldots, v_k, v_1)$ ein Kreis minimaler Länge. Wenn G die Kante (v_3, v_1) besitzt, dann hat G den Kreis (v_1, v_2, v_3, v_1) der Länge drei und wir sind fertig. Ansonsten besitzt G die Kante (v_1, v_3): Wir können den Knoten 2 überspringen und erhalten den Kreis $(v_3, \ldots, v_k, v_1, v_3)$ der kürzeren Länge $k-1$, im Wiederspruch zur Annahme. □

Es genügt also, wenn wir alle Dreiecke zerstören.

Algorithmus 5.7 FEEDBACK VERTEX SET für TURNIERE

(1) Wenn es eine Knotenmenge $V' \subseteq V$ gibt, deren Herausnahme alle Kreise zerstört und wenn $\sum_{u \in V'} w_u = 0$, dann gibt V' aus.

(2) Sonst wähle ein Dreieck $\{a, b, c\}$ mit $\varepsilon = \min\{w_a, w_b, w_c\} > 0$ aus und setze

$$w'_x = \begin{cases} \varepsilon & x \in \{a, b, c\}, \\ 0 & \text{sonst.} \end{cases}$$

Kommentar: Beachte, dass jede Lösung mindestens 3-approximativ ist, denn mindestens einer der Knoten a, b oder c muss gewählt werden.

(3) Bestimme rekursiv eine Lösung für die Gewichtung $w - w'$.

Satz 5.8 *Algorithmus 5.7 ist 3-approximativ.*

Der beste bekannte Algorithmus ist 2.5-approximativ.

5.3 FEEDBACK VERTEX SET für ungerichtete Graphen

In FEEDBACK VERTEX SET suchen wir, für einen ungerichteten Graphen $G = (V, E)$ und eine Gewichtung $w_u \geq 0$ der Knoten $u \in V$, eine leichteste Knotenmenge $V' \subseteq V$, so dass die Herausnahme von V' alle Kreise von G zerstört. Wir haben FEEDBACK VERTEX SET bereits für Turniergraphen, also für eine Klasse gerichteter Graphen kennengelernt und einen 3-approximativen Algorithmus entworfen. Auch jetzt beginnen wir mit einem 3-approximativen Algorithmus, den wir dann später zu einem 2-approximativen Algorithmus verbessern.
Zuerst stellen wir einen Zusammenhang zum Problem VERTEX COVER her. Sei $G = (V, E)$ eine Instanz für VERTEX COVER mit den Gewichten w_u für $u \in V$. Wir transformieren G in eine Instanz $H = (V \cup E, E^*)$ für FEEDBACK VERTEX SET. Die Knoten von H entsprechen also den Knoten und Kanten von G. Das Gewicht eines Knotens $u \in V$ definieren wir ebenfalls durch w_u, das Gewicht eines „Kantenknotens“ $e = \{u, v\} \in E$ ist das Maximum von w_u und w_v.

Wir verbinden zwei Knoten $u, v \in V$ in H genau dann, wenn u und v in G miteinander verbunden sind. Zusätzlich verbinden wir $u \in V$ in H mit jedem Kantenknoten $e = \{u, v\} \in E$: Wir verbinden also jeden Knoten mit jeder inzidenten Kante. Insbesondere erhalten wir für jede Kante $e = \{u, v\}$ einen Kreis $u - e - v - u$, der von einem FEEDBACK VERTEX SET zerstört werden muss.

Beobachtung 5.1 *Jede Knotenüberdeckung in G ist auch ein Feedback Vertex Set in H.*

Beweis: Angenommen, $V' \subseteq V$ ist eine Knotenüberdeckung von G. Die Herausnahme von V' in G zerstört alle Kanten in G. Was bewirkt die Herausnahme von V' im Graphen H? Alle Kantenknoten erhalten den Grad eins, gehören also nicht mehr einem Kreis an. Damit können Kreise in H nur durch Knoten $u \in V$ gebildet werden. Aber zwei Knoten $u, v \in V$ werden auch in H nur durch Kanten aus E verbunden, und diese Kanten sind nach Herausnahme der Knotenüberdeckung V' zerstört: Die Herausnahme von V' zerstört auch alle Kreise in H. □

Beobachtung 5.2 *Zu jedem Feedback Vertex Set W in H gibt es eine Knotenüberdeckung W' von gleichem Gewicht in G. W' kann effizient bestimmt werden.*

Beweis: Nehmen wir umgekehrt an, dass die Knotenmenge $W \subseteq V \cup E$ ein Feedback Vertex Set ist, dass also alle Kreise in H durch die Herausnahme von W zerstört werden. Die Herausnahme eines Kantenknotens $e = \{u, v\} \in E$ zerstört nur den einen Kreis $u-e-v-u$, den wir auch durch die Herausnahme von u oder v zerstören können. Wir können also davon ausgehen, dass W nur aus Knoten in V besteht. Da W alle Dreier-Kreise $u - e - v - u$ für $e = \{u, v\}$ zerstören muss, ist W auch eine Knotenüberdeckung für den Graphen G. □

Als eine erste Konsequenz der Beobachtungen 5.1 und 5.2 können wir folgern, dass die Gewichte einer leichtesten Knotenüberdeckung für G und eines leichtesten Feedback Vertex Set für H übereinstimmen. Insgesamt haben wir eine „approximationserhaltende“ Reduktion von VERTEX COVER auf FEEDBACK VERTEX SET erhalten: Um einen Approximationsalgorithmus für VERTEX COVER zu erhalten,

1. transformieren wir zuerst eine Instanz G für VERTEX COVER auf die Instanz H für FEEDBACK VERTEX SET,
2. benutzen dann einen Approximationsalgorithmus für FEEDBACK VERTEX SET, um eine Knotenmenge $W \subseteq V \cup E$ zu berechnen
3. und erhalten mit $W' \subseteq V$ auch eine Knotenüberdeckung gleicher Größe für G.

Satz 5.9 *Wenn FEEDBACK VERTEX SET einen r-approximative Algorithmus besitzt, dann besitzt auch VERTEX COVER einen r-approximativen Algorithmus.*

Da nur 2-approximative Algorithmen für VERTEX COVER bekannt sind, können wir bestenfalls 2-approximative Algorithmen für FEEDBACK VERTEX SET erwarten und genau das versuchen wir zu erreichen. Wir beginnen mit einer oberen Schranke für die Größe von Feedback Vertex Sets.

Lemma 5.10 *Der ungerichtete, zusammenhängende Graph G besitze n Knoten und m Kanten. Dann besitzt jeder nicht-verkleinerbare Feedback Vertex Set höchstens $m - n + 1$ Knoten.*

Beweis: Angenommen, $G = (V, E)$ besitzt einen kleinsten Feedback Vertex Set W. Wenn wir die Knoten aus W herausnehmen, erhalten wir einen Wald F. Da W minimal ist, gibt es zu jedem Knoten $u \in W$ einen Baum T des Walds F, so dass u mit zwei Knoten von T verbunden ist.
G ist zusammenhängend. Wenn wir also die Knoten aus W wieder einfügen, werden die Bäume des Walds verschmolzen und *zusätzlich* wird für jeden Knoten aus W ein Baum mindestens zweimal getroffen. Jeder zusammenhängende Graph besitzt aber mindestens $n-1$ Kanten und das zweimalige Treffen von Bäumen des Walds für jeden Knoten aus W fügt weitere $|W|$ Kanten hinzu. Also folgt $m \geq |W|+n-1$ und deshalb gilt $|W| \leq m-n+1$. □

Knoten vom Grad eins sind uninteressant, denn kein Kreis kann einen solchen Knoten benutzen. Wir können also iterativ alle Knoten vom Grad eins entfernen, bis wir einen **sauberen Graphen**, also einen Graphen ohne Knoten vom Grad eins erhalten.
Wir betrachten jetzt sehr spezielle Knotengewichtungen, wir nehmen nämlich an, dass $w_u = \alpha \cdot (\text{grad}(u) - 1)$ für eine beliebige Konstante $\alpha > 0$ gilt. In diesem anscheinend sehr speziellen Fall stellt sich sogar heraus, dass jeder Feedback Vertex Set W 3-approximativ ist, solange W *nicht-verkleinerbar*[1] ist. Für diese speziellen Knotengewichtungen können wir also keinen Fehler machen, solange wir nur alle überflüssigen Knoten entfernen.

Lemma 5.11 *Es gelte $w_u = \alpha \cdot (\text{grad}(u) - 1)$ für alle Knoten u eines ungerichteten, zusammenhängenden und sauberen Graphen G. Wenn* `fvs` *das Gewicht eines nicht-verkleinerbaren Feedback Vertex Sets ist, dann folgt*

$$m - n + 1 \leq \frac{\texttt{fvs}}{\alpha} \leq 3 \cdot (m - n) + 1.$$

Beweis: Sei W ein nicht-verkleinerbares Feedback Vertex Set. Wir zeigen zuerst die untere Schranke für `gewicht`(W), dem Gewicht der Knotenmenge W. Es ist

$$\frac{\texttt{gewicht(W)}}{\alpha} = \sum_{u \in W} w_u = \sum_{u \in W} (\text{grad}(u) - 1) = \sum_{u \in W} \text{grad}(u) - |W|. \tag{5.1}$$

$\sum_{u \in W} \text{grad}(u)$ ist mindestens so groß wie die Anzahl der mit Knoten aus W inzidenten Kanten (und möglicherweise größer, da Kanten zwischen Knoten aus W zweifach gezählt werden). Diese Anzahl beträgt aber mindestens $m - (n - 1 - |W|)$, denn der nach Herausnahme von W entstandene Wald hat höchstens $n - 1 - |W|$ Kanten. Also erhalten wir

$$\frac{\texttt{gewicht}(W)}{\alpha} = \sum_{u \in W} \text{grad}(u) - |W| \geq m - (n - 1 - |W|) - |W| = m - n + 1$$

[1] W ist nicht-verkleinerbar, wenn jede echte Teilmenge von W kein Feedback Vertex Set ist.

als Konsequenz von (5.1). Jetzt schätzen wir $\texttt{gewicht}(W)$ nach oben ab. Da die Summe aller Knotengrade mit der doppelten Knotenzahl übereinstimmt, erhalten wir

$$\frac{\texttt{gewicht}(W)}{\alpha} = \sum_{u \in W} \text{grad}(u) - |W| = 2m - \sum_{u \notin W} \text{grad}(u) - |W|. \tag{5.2}$$

Angenommen, der nach Herausnahme von W entstandene Wald besteht aus k Bäumen. In $\sum_{u \notin W} \text{grad}(u)$ werden zuerst alle Kanten zwischen Knoten aus $V \setminus W$ zweifach gezählt (und dies sind $2 \cdot (n - |W| - k)$ Kanten) und sodann alle Kanten zwischen Knoten aus $V \setminus W$ und W. Der Graph ist sauber: Es muss also mindestens zwei Kanten zwischen einem jeden der k Bäume und der Menge W geben. Es gibt also mindestens $2k$ Kanten zwischen $V \setminus W$ und W und deshalb ist

$$\sum_{u \notin W} \text{grad}(u) \geq 2 \cdot (n - |W| - k) + 2k = 2 \cdot (n - |W|).$$

Also folgt mit (5.2) und Lemma 5.10

$$\begin{aligned} \frac{\texttt{gewicht}(W)}{\alpha} &= 2m - \sum_{u \notin W} \text{grad}(u) - |W| \\ &\leq 2m - (2 \cdot (n - |W|)) - |W| = 2(m - n) + |W| \\ &\leq 2(m - n) + m - n + 1 = 3(m - n) + 1 \end{aligned}$$

und das war zu zeigen. □

5.3.1 Eine 3-approximative Lösung

Wir beschreiben jetzt einen Local Ratio Algorithmus für den Graphen $G = (V, E)$ und beliebige Gewichte $w_u \geq 0$ für die Knoten $u \in V$.

Algorithmus 5.12 FEEDBACK VERTEX SET für ungerichtete Graphen $G = (V, E)$ mit den Knotengewichten $w_u \geq 0$.

(1) Wenn $V = \emptyset$, dann gib die leere Menge als Lösung aus.

Enferne iterativ alle Knoten vom Grad 1 aus V, bis der Graph sauber ist.

(2) Bestimme $\alpha = \min_{u \in V} \frac{w_u}{d_u - 1}$ und die neue Gewichtung

$$w'_u = \alpha \cdot (d_u - 1).$$

Setze $W' = \{u \in V \mid w'_u = w_u\}$.

(3) Nimm die Knotenmenge W' aus V heraus und nenne den neuen Graphen G''. Wende das Verfahren rekursiv auf den kleineren Graphen G'' mit den Knotengewichten $w_u - w'_u$ an, um einen Feedback Vertex Set W'' zu erhalten.

(4) Bestimme eine Teilmenge $W \subseteq W' \cup W''$, so dass W ein Feedback Vertex Set für G ist, aber Kreise entstehen, wenn irgendein Knoten aus W entfernt wird.

Kommentar: Mit einem Induktionsargument folgt, dass W'' ein Feedback Vertex Set für G'' ist. G'' entsteht aus G durch Herausnahme von W' und deshalb ist $W' \cup W''$ ein Feedback Vertex Set für G. Aber W ist dann irgendein nicht-verkleinerbarer Feedback Vertex Set in $W' \cup W''$.

Satz 5.13 *Algorithmus 5.12 bestimmt eine 3-approximative Lösung für FEEDBACK VERTEX SET für ungerichtete Graphen.*

Beweis: In der Beschreibung von Algorithmus 5.12 haben wir bereits herausgestellt, dass die Menge W ein Feedback Vertex Set ist. Wie schwer ist W? Es ist

$$\begin{aligned} \texttt{gewicht}(W) &= \sum_{u \in W} w_u = \sum_{u \in W} \alpha \cdot (d_u - 1) + \sum_{u \in W} w_u - \alpha \cdot (d_u - 1) \\ &= \sum_{u \in W} w'_u + \sum_{u \in W} (w_u - w'_u). \end{aligned}$$

W ist nicht verkleinerbar. Wir wenden Lemma 5.11 an und erhalten deshalb

$$\sum_{u \in W} w'_u \leq 3 \cdot \texttt{opt}' \tag{5.3}$$

für das Gewicht $\texttt{opt}'$ eines optimalen Feedback Vertex Sets für die Knotengewichte w'. Schließlich beachte

$$\begin{aligned} \sum_{u \in W} (w_u - w'_u) &\leq \sum_{u \in W'} (w_u - w'_u) + \sum_{u \in W''} (w_u - w'_u) = \sum_{u \in W''} (w_u - w'_u) \\ &\leq 3 \cdot \texttt{opt}'', \end{aligned} \tag{5.4}$$

wobei $\texttt{opt}''$ das Gewicht eines optimalen Feedback Vertex Sets für G'' ist. In der letzten Ungleichung haben wir induktiv angenommen, dass Algorithmus 5.12 den Approximationsfaktor drei auf dem kleineren Graphen G'' erreicht.
Die Behauptung folgt jetzt aus (5.3) und (5.4), da $\texttt{opt}' + \texttt{opt}'' \leq \texttt{opt}$ für das Gewicht $\texttt{opt}$ eines optimalen Feedback Vertex Sets für G gilt. □

Aufgabe 66
Weise nach, dass $\texttt{opt}' + \texttt{opt}'' \leq \texttt{opt}$ gilt.

5.3.2 Eine 2-approximative Lösung

Nenne einen Kreis fast-disjunkt, wenn der Kreis höchstens einen Knoten vom Grad mindestens drei durchläuft. Wir erhalten eine 2-approximative Lösung wie folgt.

Algorithmus 5.14 Eine Verbesserung von Algorithmus 5.12

(1) Wenn G keine fast-disjunkte Kreise besitzt, dann verfahre im nicht-rekursiven Schritt wie Algorithmus 5.12.

(2) Besitzt G hingegen einen fast-disjunkten Kreis K, dann:

(a) Bestimme den Knoten $u_{\min}$ des Kreises K mit kleinstem Gewicht.

(b) Bestimme den Knoten u_K in K mit Grad mindestens drei. Entferne, bis auf u_K, alle Knoten des Kreises und nenne den neuen Graphen G''. Reduziere das Gewicht von u_K um das Gewicht von $u_{\min}$.

(c) Bestimme einen Feedback Vertex Set W'' für G'' rekursiv.

Kommentar: Wenn wir induktiv annehmen, dass W'' eine 2-approximative Lösung für G'' ist, dann ist $W'' \cup \{u_{\min}\}$ eine 2-approximative Lösung für G, denn $\{u_{\min}\}$ ist ein leichtester Feedback Vertex Set für K.

Warum zahlt sich die Sonderbehandlung von fast-disjunkten Kreisen aus?

Lemma 5.15 *G sei ein ungerichteter Graph, der keinen fast-disjunkten Kreis enthält. Zusätzlich gelte $w_u = \alpha \cdot (\text{grad}(u) - 1)$ für alle Knoten u. Wenn* `fvs` *das Gewicht eines nicht-verkleinerbaren Feedback Vertex Sets ist, dann folgt*

$$m - n + 1 \leq \frac{\texttt{fvs}}{\alpha} \leq 2 \cdot (m - n) + 1.$$

Aufgabe 67
Zeige Lemma 5.15.

Wir können jetzt die Analyse von Algorithmus 5.12 übernehmen und erhalten mit Hilfe von Lemma 5.15:

Satz 5.16 *Algorithmus 5.14 bestimmt eine 2-approximative Lösung für FEEDBACK VERTEX SET für ungerichtete Graphen.*

Aufgabe 68
Bestimme einen ungerichteten Graphen $G = (V, E)$ und eine Gewichtung w_u der Knoten $u \in V$, so dass Algorithmus 5.14 eine Lösung mit möglichst schlechten Approximationsfaktoren berechnet.

Kapitel 6

Branch & Bound

Das Branch & Bound Verfahren versucht, eine optimale Lösung eines Optimierungsproblems durch eine intelligente Suche unter den möglichen Lösungen zu finden. Wir beschreiben die Grobstruktur dieses Ansatzes für das allgemeine Minimierungsproblem

$$\text{minimiere } f(x), \text{ so dass Lösung}(x).$$

Für den Branching-Schritt des Branch & Bound Verfahrens muss zuerst ein Branching Operator B definiert werden: B zerlegt eine Menge von Lösungen in disjunkte Teilmengen. Die wiederholte Anwendung des Branching Operators erzeugt dann einen Branch & Bound Baum $\mathcal{B}$:

> Die Wurzel von $\mathcal{B}$ entspricht der Menge aller Lösungen. Ist v ein Knoten von $\mathcal{B}$ mit Lösungsmenge $\mathcal{L}(v)$ und hat v die Kinder $v_1, \ldots, v_k$, dann zerlegen die Lösungsmengen $\mathcal{L}(v_i)$ der Kinderknoten die Lösungsmengen $\mathcal{L}(v)$ des Elternknotens.

Für den Bounding-Schritt wird eine untere Schranke benötigt: Für jeden Knoten v von $\mathcal{B}$ nehmen wir nämlich an, dass eine untere Schranke `unten(v)` gegeben ist, so dass

$$\texttt{unten}(v) \leq f(y)$$

für jede Lösung $y \in \mathcal{L}(v)$ gilt.

> v kann verworfen werden, wenn $\texttt{unten}(v) \geq f(y_0)$ für eine aktuelle beste Lösung y_0 gilt, denn keine Lösung in $\mathcal{L}(v)$ ist besser als die Lösung y_0.

Algorithmus 6.1 **Branch & Bound**
Das Minimierungsproblem $(\min, f, L)$ sei zu lösen. Der Branching Operator B sei gegeben. Anfänglich besteht der Branch & Bound Baum $\mathcal{B}$ nur aus der (aktivierten) Wurzel.

(1) Eine Lösung y_0 wird mit Hilfe einer **Heuristik** berechnet.

(2) Wiederhole, solange es aktivierte Blätter gibt:

(2a) Wähle das **erfolgversprechendste** aktivierte Blatt v von $\mathcal{B}$.

(2b) **Branching:** Wende den Branching Operator B auf v an, um die Kinder $v_1, \ldots, v_k$ zu erhalten. Inspiziere die k Teilmengen nacheinander:

- Wenn es offensichtlich ist, dass v_i eine Lösung y_i enthält, die besser als y_0 ist, dann setze $y_0 = y_i$ und deaktiviere gegebenenfalls Blätter.
- Ansonsten führe den **Bounding-Schritt** durch: Nur wenn $\texttt{unten}(v_i) < f(y_0)$, wird v_i aktiviert.

(3) Gib die Lösung y_0 als optimale Lösung aus.

Eine erfolgreiche Implementierung von Branch & Bound muss die folgenden Probleme lösen:

- Die Anfangslösung y_0 muss möglichst nahe am Optimum liegen, damit der Bounding-Schritt schlechte Knoten frühzeitig disqualifiziert. Aus genau demselben Grund muss die untere Schranke für v die Qualität der besten Lösung in v möglichst gut voraussagen.

- Die Wahl eines erfolgversprechendsten Blatts bestimmt die Art und Weise, in der $\mathcal{B}$ durchsucht wird und bestimmt damit den Speicherplatzverbrauch.

  - Tiefensuche schont den Speicherplatzverbrauch. Allerdings wird die schnelle Entdeckung guter Lösungen leiden, da stets mit dem letzten, und nicht mit dem besten Knoten gearbeitet wird.
  - Der große Speicherverbrauch schließt Breitensuche als ein praktikables Suchverfahren aus.
  - In der „best first search“ wird der Knoten v mit der niedrigsten unteren Schranke gewählt. Man versucht also, schnell gute Lösungen zu erhalten. Häufig werden Varianten der Tiefensuche und der best-first search kombiniert, um einerseits mit dem vorhandenen Speicherplatz auszukommen und um andererseits gute Lösungen möglichst schnell zu entdecken.

6.1 Das RUCKSACK-Problem

n Objekte mit Gewichten $g_1, \ldots, g_n \in \mathbb{R}$ und Werten $w_1, \ldots, w_n \in \mathbb{N}$ sind vorgegeben ebenso wie eine Gewichtsschranke $G \in \mathbb{R}$. Der Rucksack ist mit einer Auswahl von Objekten zu bepacken, so dass einerseits die Gewichtsschranke G nicht überschritten wird und andererseits der Gesamtwert maximal ist.
Wir betrachten zuerst das fraktionale RUCKSACK-Problem. Hier dürfen wir Anteile x_i ($0 \leq x_i \leq 1$) des $i$ten Objekts in den Rucksack packen. Das fraktionale RUCKSACK-Problem kann sehr einfach mit einem Greedy-Algorithmus gelöst werden. Wir nehmen

dazu ohne Beschränkung der Allgemeinheit an, dass die Objekte bereits absteigend nach ihrem „Wert pro Kilo“ sortiert sind, dass also

$$\frac{w_1}{g_1} \geq \frac{w_2}{g_2} \geq \cdots \geq \frac{w_n}{g_n}$$

gilt. Wenn $\sum_{i=1}^{k} g_i \leq G < \sum_{i=1}^{k+1} g_i$, dann erhalten wir eine optimale Lösung, wenn wir die Objekte $1, \ldots k$ einpacken und die Restkapazität des Rucksacks mit dem entsprechenden Anteil an Objekt $k+1$ füllen.
Angeregt durch den Greedy-Algorithmus sortieren wir die Objekte absteigend nach ihrem Wert pro Kilo. Jeder Knoten v des Branch & Bound Baums wird dann durch ein Paar (J, i) beschrieben: Die Objekte aus $J \subseteq \{1, \ldots, i\}$ wurden bereits, ohne die Kapazität G zu überschreiten, in den Rucksack gepackt. Der Branching Operator erzeugt die beiden Kinder mit den Beschreibungen $(J, i+1)$ und $(J \cup \{i+1\}, i+1)$ entsprechend dem Auslassen oder der Hinzunahme des $i+1$.sten Objekts.
Beachte, dass wir diesmal ein Maximierungsproblem lösen möchten und Branch & Bound benötigt eine obere statt einer unteren Schranke. Wir besorgen uns sehr gute obere Schranken mit Hilfe des Greedy-Algorithmus für das fraktionalen RUCKSACK-Problem: Für den Knoten (J, i) berechnen wir die Restkapazität $G' = G - \sum_{j \in J} g_j$ und wenden den Greedy-Algorithmus auf die Objekte $i+1, \ldots, n$ mit der neuen Gewichtsschranke G' an. Da der Greedy-Algorithmus eine optimale Lösung des fraktionalen RUCKSACK-Problems berechnet, und da der optimale Wert des fraktionalen Problems mindestens so groß wie der optimale Wert des ganzzahligen Problems ist, haben wir die gewünschte obere Schranke gefunden.
Eine Anfangslösung y_0 können wir mit dem dynamischen Programmier-Algorithmus aus Satz 3.1 bestimmen. Schließlich ist eine erfolgversprechendste Lösung in Schritt (2a) zu bestimmen. Die Wahl einer wertvollsten Bepackung ist sicherlich eine sinnvolle Option.

Bemerkung 6.1 Wir haben die obere Schranke im RUCKSACK-Problem durch die Methode der Relaxierung gefunden: Wir haben die Forderung der Ganzzahligkeit abgeschwächt und reelle Lösungen zugelassen. In vielen Fällen werden dadurch Problemstellungen stark vereinfacht; wir kommen auf diesen Aspekt später zurück, wenn wir nach einer Diskussion der linearen Programmierung Branch & Cut Verfahren beschreiben.

6.2 TSP: Branch & Bound mit 1-Bäumen

Wir betrachten das metrische Traveling Salesman Problem M-TSP: n Städte $1, \ldots, n$ sowie Distanzen $d(r, s)$ zwischen je zwei Städten r und s sind gegeben. Die Distanz d möge einer Metrik entsprechen. Unser Ziel ist die Bestimmung einer kürzesten Rundreise, die jede Stadt genau einmal besucht.
Wir haben bereits viele Heuristiken für das Traveling Salesman Problem in Abschnitt 2.8 kennengelernt und können uns somit gute Anfangslösungen beschaffen.
Unsere Branch & Bound Implementierung erzeugt einen Branch & Bound Baum $\mathcal{B}$, dessen Knoten durch Paare (S, T) beschrieben werden: $S \subseteq \{\ \{r, s\} \mid r \neq s\}$ ist eine Menge

erzwungener Kanten, die eine Rundreise durchlaufen muss, und $T \subseteq \{ \{r, s\} \mid r \neq s\}$ eine Menge verbotener Kanten. Das Ausgangsproblem wird durch $(\emptyset, \emptyset)$ beschrieben, nachfolgende Anwendungen des Branching Operators schreiben aber Kanten vor bzw. verbieten Kanten.
Das schwierige Problem im Entwurf von Branch & Bound Algorithmen ist im Allgemeinen die Ableitung guter **unterer Schranken**. Um die Diskussion zu erleichtern, führen wir untere Schranken nur für das Ausgangsproblem $(\emptyset, \emptyset)$ ein. Wir wissen, dass die Länge eines minimalen Spannbaums eine untere Schranke für die Länge von Rundreisen ist. Wir zeigen jetzt, wie man diese untere Schranke deutlich verbessert. Zuerst beobachten wir, dass ein Spannbaum nur $n-1$ Kanten besitzt, während eine Rundreise n Kanten durchläuft. Wir ersetzen deshalb die Spannbaum Methode durch die Methode der 1-Bäume[1]:

> Wir berechnen einen minimalen Spannbaum B für alle Städte bis auf Stadt 1. Dann fügen wir 1 zu B hinzu, indem wir 1 mit den beiden nächstliegenden Städten verbinden.
>
> Durch die Hinzunahme von 1 und seiner beiden billigsten Kanten haben wir einen 1-Baum B^* erhalten, nämlich einen Graphen mit einem einzigen Kreis. Das Gewicht des 1-Baums ist unsere neue untere Schranke.

Haben wir tatsächlich eine untere Schranke erhalten? Wenn wir die Stadt 1 aus einer Rundreise entfernen, dann erhalten wir einen Spannbaum für die verbleibenden $n-1$ Städte, und das Gewicht dieses Spannbaums ist natürlich mindestens so groß wie das Gewicht eines minimalen Spannbaums. Schließlich haben wir die Stadt 1 über seine beiden kürzesten Kanten mit dem minimalen Spannbaum verschweißt und diese beiden Kanten sind nicht schwerer als die beiden mit Stadt 1 inzidenten Kanten der Rundreise.
Aber selbst die neue untere Schranke ist nicht gut genug. Wir ändern die Distanzen $d(r, s)$ zwischen Stadt r und Stadt s wie folgt: Wir wählen Parameter λ_r für jede Stadt r und legen $d^*(r, s) = d(r, s) + \lambda_r + \lambda_s$ als neue Distanz fest.

- Wenn L die Länge einer Rundreise für die alten Distanzen ist, dann ist $L + 2 \cdot \sum_{r=1}^{n} \lambda_r$ die Länge für die neuen Distanzen. Insbesondere wird eine für die neuen Distanzen optimale Rundreise auch für die alten Distanzen optimal sein.

- Sei $\text{grad}_{B*}(r)$ die Anzahl der Nachbarn von Stadt r in einem beliebigen 1-Baum B^*. Wenn $l(B^*)$ das Gewicht von B^* für die alten Distanzen ist, dann ist $l(B^*) + \sum_{i=1}^{n} \lambda_i \cdot \text{grad}_{B^*}(i)$ das Gewicht von B^* für die neuen Distanzen.

  > Es ist also nicht zu erwarten, dass minimale 1-Bäume für die Distanz d auch für die neue Distanz optimal bleiben.

Wie sollte der Vektor $\lambda = (\lambda_1, \ldots, \lambda_n)$ gewählt werden? Sei $\texttt{opt}_1(\lambda)$ das Gewicht eines minimalen 1-Baums für die durch λ definierten neuen Distanzen: Wenn B^* ein optimaler

[1] Ein 1-Baum ist ein Baum auf den Knoten $2, \ldots, n$, der Knoten 1 wird durch Kanten zu den beiden nächstliegenden Städten angeschlossen.

1-Baum für λ ist, dann folgt also $\texttt{opt}_1(\lambda) = l(B^*) + \sum_{i=1}^{n} \lambda_i \cdot \text{grad}_{B^*}(i)$. Für eine optimale Rundreise mit Länge opt für die alten Distanzen und für jeden Vektor λ gilt

$$\begin{aligned} \texttt{opt} + 2 \cdot \sum_{i=1}^{n} \lambda_i &\geq \texttt{opt}_1(\lambda) \\ &= l(B^*) + \sum_{i=1}^{n} \lambda_i \cdot \text{grad}_{B^*}(i), \end{aligned} \tag{6.1}$$

da wir eine Rundreise mit einem minimalen 1-Baum vergleichen. Wir betrachten deshalb die Funktion

$$f(\lambda) = \min_{B^*} \{\ l(B^*) + \sum_{i=1}^{n} \lambda_i \cdot (\text{grad}_{B^*}(i) - 2)\ \}$$

und

$$\texttt{opt} \geq f(\lambda)$$

gilt für jeden Vektor λ. Wir sollten deshalb versuchen, f zu maximieren, um eine möglichst gute untere Schranke $f(\lambda)$ zu erhalten. Man nennt $\max_{\lambda \in \mathbb{R}^n} f(\lambda)$ die Held-Karp Schranke, die in praktischen Anwendungen hervorragende Resultate liefert. In Abschnitt 9.3 werden wir die mathematische Begründung der Held-Karp Schranke als Lagrange Relaxierung kennenlernen.
Beachte, dass $f(\lambda)$ für jeden fixierten Vektor λ einfach zu berechnen ist: Bestimme einen minimalen Spannbaum für die Kantengewichte $d(r,s) + \lambda_r + \lambda_s$ auf den Knoten $2, \ldots, n$ und füge den Knoten 1 über zwei kürzeste Kante ein. Die Schwierigkeit in der Berechnung der Held-Karp Schranke besteht deshalb in der Berechnung der Maximierung.
Das **erfolgversprechendste Blatt** $v = (S, T)$ wählen wir stets gemäß Tiefensuche. Um den **Branching Operator** für das Blatt v zu beschreiben, nehmen wir zuerst wieder $S = T = \emptyset$ an. Angenommen, wir haben den Vektor λ^* für die Bestimmung der unteren Schranke berechnet und $B^*_{\min}$ ist ein minimaler 1-Baum für die neuen Distanzen. Wir versuchen mit dem Branching Operator entweder $B^*_{\min}$ als optimalen 1-Baum auszuschalten oder aber ein signifikant einfacheres TSP-Problem zu erhalten.
Zuerst beachten wir, dass $B^*_{\min}$ eine Stadt s vom Grad mindestens drei besitzt, denn sonst ist $B^*_{\min}$ bereits eine Rundreise und wegen (6.1) sogar eine optimale Rundreise. Die Kanten $\{r,s\}$ und $\{t,s\}$ seien die längsten, mit s inzidenten Kanten von $B^*_{\min}$. Der Branching Operator erzeugt drei Kinder, wobei alle Kinder $B^*_{\min}$ als optimalen 1-Baum ausschalten.

(1) Für das erste Kind verbieten wir die Benutzung der Kante $\{r,s\}$.

(2) Für das zweite Kind wird die Benutzung von $\{r,s\}$ erzwungen, aber die Benutzung von $\{t,s\}$ verboten.

(3) Für das dritte Kind wird sowohl die Benutzung von $\{r,s\}$ wie auch die Benutzung von $\{t,s\}$ erzwungen. Alle anderen mit s inzidenten Kanten werden verboten.

Wird im folgenden ein Branching-Schritt für einen Knoten s durchgeführt, der bereits eine erzwungene Kante besitzt, dann erzeugt man das dritte Kind nicht und verbietet für das zweite Kind alle sonstigen mit s inzidenten Kanten, bis auf die erzwungene Kante.

Teil II

Lineare Programmierung

Mit dem Simplex-Algorithmus und den Interior-Point-Verfahren für die lineare Programmierung lernen wir mächtige Techniken zum Lösen von linearen Optimierungsproblemen kennen. Insbesondere werden wir effiziente Algorithmen für das allgemeine Matching Problem mit gewichteten Kanten wie auch für das Netzwerkfluss Problem kennenlernen. Weitere wichtige Anwendungen der linearen Programmierung wie zum Beispiel das Branch & Cut Verfahren werden wir in den nachfolgenden Kapiteln kennenlernen.
In der linearen Programmierung sind für n reellwertige Variablen $x_1, \ldots, x_n$ die m linearen Gleichungen

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad \text{für } i = 1, \ldots, m$$

unter den Nichtnegativitätsbedingungen $x_1, \ldots, x_n \geq 0$ zu erfüllen. Unter allen Lösungsvektoren $x := (x_1, \ldots, x_n)^T$ suchen wir einen Vektor, der die lineare Zielfunktion

$$c^T \cdot x = \sum_{j=1}^{n} c_j x_j$$

minimiert. Sei

$$A = [a_{ij}]_{1 \leq i \leq m, 1 \leq j \leq n}$$

die $m \times n$-Matrix des Gleichungssystems und b und c seien die entsprechenden Spaltenvektoren. Wir nennen

$$\text{minimiere } c^T \cdot x, \text{ so dass } Ax = b, x \geq 0$$

die *Standardform* der linearen Programmierung. Die Standardform deckt auch die folgenden Fälle ab:

(1) Minimierungsprobleme können auf Maximierungsprobleme zurückgeführt werden und umgekehrt, denn eine Maximierung von $c^T \cdot x$ ist äquivalent zu einer Minimierung von $-c^T \cdot x$.

(2) Falls wir eine Variable x nicht der Nichtnegativitätsbedingung unterwerfen wollen, ersetzen wir x durch die Differenz $(x_+ - x_-)$ für zwei neue Variablen x_- und x_+. Wir fordern $x_+ \geq 0$ und $x_- \geq 0$.

In der Literatur findet man auch die folgende, *kanonische Form* der linearen Programmierung:

$$\begin{aligned} \text{minimiere } c^T \cdot x, \text{ so dass} \quad Ax &\geq b \\ x &\geq 0 \end{aligned}$$

Beide Formulierungen beschreiben dieselbe Klasse von Optimierungsproblemen, da einerseits

$$a^T \cdot x = b \quad \Leftrightarrow \quad a^T \cdot x \geq b \text{ und } -a^T \cdot x \geq -b$$

und andererseits für eine neu eingeführte „Slackvariable“ s

$$a^T \cdot x \geq b \;\; \Leftrightarrow \;\; a^T \cdot x - s = b \;\text{ und }\; s \geq 0.$$

Beispiel 6.1 **Ein lineares Programm für das gewichtete Matching Problem**
Wir betrachten das gewichtete Matching Problem für allgemeine Graphen $G = (V, E)$ mit Kantengewicht w_e für $e \in E$. Eine Teilmenge $M \subseteq E$ heißt ein Matching, falls es keine zwei Kanten in M mit einem gemeinsamen Endpunkt gibt. Das Ziel des Matching Problems ist die Bestimmung eines Matchings M, so dass

$$\sum_{e \in M} w_e$$

maximal unter allen Matchings ist. Unsere Formalisierung als lineares Programm verwendet für jede Kante e die Variable x_e mit

$$x_e = \begin{cases} 1 & e \text{ gehört zum Matching} \\ 0 & \text{sonst} \end{cases}$$

als *beabsichtigte* Interpretation. Wir wählen die kanonische Form des linearen Programms und erhalten die folgende Formulierung.

(1) Fordere $x_e \geq 0$ und $-x_e \geq -1$.

(2) Damit jeder Knoten $v \in V$ nur zu einer gewählten Kante gehört, fügen wir für v die Bedingung

$$-\sum_{w, \{v,w\} \in E} x_{\{v,w\}} \geq -1$$

hinzu.

(3) Wir wählen

$$\sum_{e \in E} w_e \cdot x_e$$

als zu maximierende Zielfunktion in den Variablen x_e.

Wir zeigen später, dass dieses Programm für bipartite Graphen ganzzahlige (und damit binäre) Lösungen besitzt und dass die beabsichtigte Interpretation für bestimmte Lösungen, nämlich für die Ecken des Lösungsraums erzwungen ist.

Beispiel 6.2 **Das Flussproblem als ein Problem der linearen Programmierung**
Ein Flussproblem wird durch das Quintupel $\mathcal{N} = (V, E, t, s, c)$ beschrieben. V ist eine Menge von Knoten und E eine Menge von gerichteten Kanten mit Endpunkten aus V. s, t sind zwei ausgezeichnete Knoten, die wir als Quelle, bzw. als Senke bezeichnen. Schließlich ist $c : E \to \mathbb{R}_{\geq 0}$ eine Funktion, die jeder Kante $e \in E$ ihre maximale Kapazität $c(e) \geq 0$ zuweist.
Ein Fluss im Netzwerk $\mathcal{N}$ ist ebenfalls eine Funktion $f : E \to \mathbb{R}_{\geq 0}$ mit den folgenden Eigenschaften:

(1) $0 \leq f(e) \leq c(e)$ für jede Kante $e \in E$: Der Fluss entlang einer Kante darf die Kapazität der Kante nicht übersteigen.

(2) Für jeden Knoten $v \in V \setminus \{s, t\}$ gilt

$$\sum_{(u,v)\in E} f(u, v) = \sum_{(v,u)\in E} f(v, u)$$

und damit fordern wir Flusserhaltung: Der in v eingehende Fluss muss v auch wieder verlassen.

Das Ziel des Flussproblems ist die Konstruktion eines maximalen Flusses von der Quelle s zur Senke t. Mit anderen Worten, wir müssen die lineare Zielfunktion

$$\sum_{(s,v)\in E} f_{(s,v)} - \sum_{(v,s)\in E} f_{(v,s)}$$

maximieren: Der aus s hinausfließende Fluss ist zu maximieren, wobei aber der zurückfließende Fluss abzuziehen ist. Wir formulieren das Flussproblem als ein lineares Programm und führen dazu für jede Kante $e \in E$ die Flussvariable f_e ein:

(1) Zur Einhaltung der Kapazitätsschranken fordern wir $-f_e \geq -c(e)$ für alle Kanten $e \in E$. Beachte, dass $f_e \geq 0$ zu fordern ist und dies der Form des linearen Programms entspricht.

(2) Um Flusserhaltung auszudrücken, wählen wir für jeden Knoten $v \in V \setminus \{s, t\}$ die Bedingungen:

$$\sum_{(v,w)\in E} f_{(v,w)} - \sum_{(w,v)\in E} f_{(w,v)} = 0$$

(3) Die Zielfunktion ist natürlich

$$\sum_{(s,v)\in E} f_{(s,v)} - \sum_{(v,s)\in E} f_{(v,s)}$$

Durch Maximieren der Zielfunktion erhalten wir einen maximalen Fluss.

Aufgabe 69

Aus einem physikalischen Experiment seien n Messpunkte $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ gegeben. Wenn man Gründe hat, anzunehmen, eine Gerade erkläre die Messdaten, versucht man eine Gerade zu finden, so dass die Abhängigkeit der y_i von den x_i möglichst gut dargestellt wird. Zum einen verwendet man hierzu die „lineare Regression“: Man versucht eine Gerade $ax + b$ zu finden, so dass $\sum_{i=1}^{n}(ax_i + b - y_i)^2$ minimiert wird. In dieser Aufgabe wollen wir im Unterschied dazu eine Gerade $ax + b$ bestimmen, welche die Summe der vertikalen Abstände zur Geraden, also $\sum_{i=1}^{n} |ax_i + b - y_i|$, minimiert. Löse das Problem durch lineares Programmieren.

Aufgabe 70

Ein Party-Service benötigt an N aufeinanderfolgenden Tagen $i = 1, \ldots, n$ jeweils r_i Teller. Dabei kann der Betreiber Teller entweder

- kaufen (für p Cent),
- schnell waschen lassen (in m Tagen für f Cent) oder
- langsam waschen lassen (in $n > m$ Tagen für $s < f$ Cent).

Jeden Tag muss der Betreiber entscheiden, wieviele schmutzige Teller schnell bzw. langsam zu waschen sind und wieviele Teller zu kaufen sind, wobei am Tag i auf jeden Fall r_i saubere Teller vorhanden sein müssen. Finde eine Darstellung des Problems als lineares Programm.

Wir führen als Nächstes die Grundbegriffe der linearen Programmierung für die Standardform

$$\text{minimiere } c^T \cdot x, \text{ so dass } \quad Ax = b, \text{ und } x \geq 0.$$

ein.

Definition 6.2

(a) Ein Vektor x mit $Ax = b$ und $x \geq 0$ heißt eine Lösung.

(b) Ein lineares Programm heißt lösbar, wenn Lösungen existieren, ansonsten ist das Programm unlösbar.

(c) Wir bezeichnen die Lösungsmenge mit $L(A, b)$, wobei

$$L(A, b) := \{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}.$$

(d) Eine Lösung $x \in L(A, b)$ ist optimal, falls

$$c^T \cdot x = \inf\{c^T \cdot x \mid x \in L(A, b)\}.$$

(e) Eine Lösung $x^* \in L(A, b)$ ist genau dann eine Ecke von $L(A, b)$, wenn kein Vektor $y \neq 0$ mit $x^* + y \in L(A, b)$ und $x^* - y \in L(A, b)$ existiert.

Aufgabe 71
Wir nehmen an, dass $0 \in L(A, b)$. Zeige, dass der Nullpunkt eine Ecke ist.

Die geometrische Sichtweise: Wir führen zuerst die zentralen Konzepte einer Hyperebene und eines Halbraums ein.

Definition 6.3 Eine Hyperebene $H \subseteq \mathbb{R}^n$ ist die Menge aller Lösungen einer linearen Gleichung $a^T x = b$, also $H = \{x \mid a^T x = b\}$. Die Lösungsmenge $\{x \mid a^T x \geq b\}$ wird ein Halbraum genannt.
Im allgemeinen Fall mehrerer Ungleichungen definieren wir die Lösungsmenge $L^*(A, b) = \{x \in \mathbb{R}^n \mid Ax \geq b,\ x \geq 0\}$. Wir nennen $L^*(A, b)$ ein *Polyeder*, bzw. ein *Polytop* wenn $L^*(A, b)$ beschränkt ist.

Ein Lösungsraum $L^*(A,b)$ ist also der Durchschnitt von Halbräumen und zwar der Durchschnitt der folgenden $m+n$ Halbräume (e_i sei der i-te Einheitsvektor)

$$\begin{aligned} H_i &:= \{x \in \mathbb{R}^n \mid a_i^T \cdot x \geq b_i\} \text{ für } i = 1, \ldots, m \\ N_i &:= \{x \in \mathbb{R}^n \mid e_i^T \cdot x \geq 0\} \text{ für } i = 1, \ldots, n. \end{aligned}$$

Das nächste wichtige Konzept ist der Begriff der Seitenfläche eines Polyeders P. Zuerst führen wir aber die Dimension von P als die Dimension des kleinsten affinen Raumes ein, der P enthält: Ein einzelner Punkt $x \in \mathbb{R}^n$ ist also stets 0-dimensional und ein Geradenstück 1-dimensional.
Liegt ein Polyeder P *vollständig auf einer Seite einer Hyperebene* H, dann wird der Durchschnitt $P \cap H$ eine Seitenfläche genannt.

- Eine Ecke x ist eine 0-dimensionale Seitenfläche: Der Durchschnitt $P \cap H$ besteht also nur aus x.

- Eine Facette F ist eine Seitenfläche im „eigentlichen Sinn“: Der Durchschnitt $F = P \cap H$ ist ein $(d-1)$-dimensionales Polyeder, falls P ein d-dimensionales Polyeder ist.

- Wir sagen, dass eine Ecke x entartet ist, wenn die Anzahl der Facetten, die x enthalten, größer ist als die Dimension von P. Die Spitze einer 3-dimensionalen Pyramide mit quadratischer Grundfläche ist ein Beispiel einer entarteten Ecke.

Man kann zeigen, dass ein Polyeder P stets konvex ist: Mit je zwei Punkten $u, v \in P$ gehört auch stets die u und v verbindende Gerade zu P. Wir beobachten als Nächstes ohne Beweis, dass ein Polytop stets die konvexe Hülle seiner Ecken ist.

Definition 6.4 Für die Punktmenge $X = \{x_1, \ldots, x_N\} \subseteq \mathbb{R}^n$ ist

$$\text{conv}(X) = \{ \sum_{i=1}^{N} \lambda_i \cdot x_i \mid \lambda_1, \ldots, \lambda_N \geq 0, \sum_{i=1}^{N} \lambda_i = 1. \}$$

die konvexe Hülle von X.

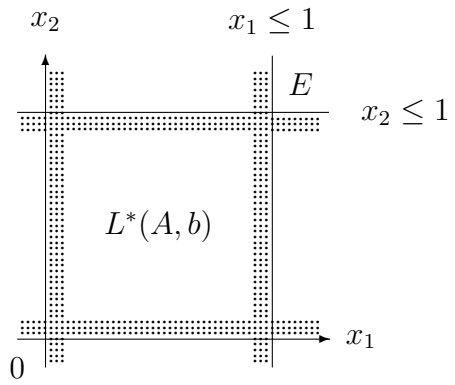
Wenn $E \subseteq P$ die Menge der Ecken des Polytops P ist, dann ist also $P = \text{conv}(E)$.

Beispiel 6.3 Wir betrachten den Lösungsraum $L^*(A,b) := \{x \in \mathbb{R}^2 \mid Ax \geq b,\ x \geq 0\}$ mit

$$A := \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \text{ und } b^T := (-1, -1).$$

Wir erhalten den Lösungsraum als Duchschnitt der Halbräume

$$\begin{aligned} L^*(A,b) &= \{(x_1,x_2) \in \mathbb{R}^2 \mid x_1 \geq 0\} \cap \{(x_1,x_2) \in \mathbb{R}^2 \mid x_2 \geq 0\} \\ &\cap \{(x_1,x_2) \in \mathbb{R}^2 \mid x_1 \leq 1\} \cap \{(x_1,x_2) \in \mathbb{R}^2 \mid x_2 \leq 1\}. \end{aligned}$$



Der Lösungsraum $L^*(A,b)$.

Offenbar ist $E = (1,1)$ eine Ecke: Für $y = (y_1, y_2) \neq 0$ mit $E + y \in L^*(A,b)$ gilt $y_1 < 0$ oder $y_2 < 0$. Also ist

$$E - y = (1 - y_1, 1 - y_2) \notin L^*(A,b),$$

da mindestens eine Komponente echt größer als 1 ist.

Das Konzept einer Ecke ist grundlegend: Es existiert immer eine optimale Ecke, sofern die Lösungsmenge $L(A,b)$ nichtleer und der minimale Zielwert endlich ist.

Lemma 6.5 *Die Lösungsmenge* $L(A,b) := \{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}$ *sei nichtleer und zusätzlich sei das Infimum* $\inf\{c^T \cdot x \mid x \in L(A,b)\}$ *endlich. Dann gibt es eine Ecke, die optimal ist.*

Beweis: Wir nehmen zuerst $c \neq 0$ an. Ist dieser Fall gezeigt, folgt offensichtlich der Fall $c = 0$, da hier nur zu zeigen ist, dass eine Ecke existiert. Die Existenz einer Ecke wird aber schon für $c \neq 0$ implizit nachgewiesen.
Sei $x \in L(A,b)$ eine optimale Lösung des linearen Programms in Standardform. Es genügt, eine Ecke x^* mit $c^T \cdot x \geq c^T \cdot x^*$ zu finden. Wir versuchen in mehreren Schritten aus x eine mindestens so gute Ecke zu erhalten.
Falls x bereits eine Ecke ist, sind wir fertig. Sei $x \geq 0$ mit $x \neq 0$ keine Ecke, d.h. es existiert ein $y \neq 0$ mit $x \pm y \in L(A,b)$. Insbesondere ist $x \pm y \geq 0$ und aus $Ax + Ay = b$ und $Ax = b$ folgt $Ay = 0$. Durch einen möglichen Übergang von y auf $-y$ können wir erreichen, dass

$$c^T \cdot y \leq 0 \tag{6.2}$$

gilt. Falls $c^T \cdot y = 0$, wählen wir zwischen y und $-y$, so dass der gewählte Vektor mindestens eine negative Komponente besitzt. Ist $c^T \cdot y < 0$ arbeiten wir mit dem ursprünglichen y weiter. Schließlich führen wir eine Fallunterscheidung durch.

Fall 1: Es gibt eine Komponente j mit $y_j < 0$. Wähle $\lambda > 0$ maximal, so dass $x + \lambda y \geq 0$. Wegen $\lambda > 0$ hat der Vektor $x + \lambda y$ im Vergleich zu x mindestens eine Null-Komponente mehr: Wenn $x_i = 0$, dann muss $y_i = 0$ gelten, da $x \pm y \in L(A,b)$. Wir setzen

$$x^{\text{neu}} := x + \lambda y \geq 0.$$

Da $x^{\text{neu}} \geq 0$, genügt für $x^{\text{neu}} \in L(A,b)$ der Nachweis von $A \cdot x^{\text{neu}} = b$. Dies folgt, da

$$A \cdot x^{\text{neu}} = A(x + \lambda y) = Ax + \lambda \cdot \underbrace{(Ay)}_{=0} = Ax = b.$$

Fall 2: Für alle Komponenten j ist $y_j \geq 0$. Für jedes $\lambda \geq 0$ gilt $x + \lambda y \in L(A,b)$, da

$$A(x + \lambda y) = Ax + \lambda \cdot \underbrace{(Ay)}_{=0} = Ax = b$$

und da $x + \lambda y \geq x \geq 0$. Da nach Fallannahme y nur nichtnegative Komponenten besitzt, muss $c^T y < 0$ gelten: Es gibt also eine positive Komponente von y, so dass die entsprechende Komponente von c negativ ist. Aber dann verstoßen wir wegen (6.2) gegen die Endlichkeitsbedingung von inf $\{c^T \cdot x \mid x \in L(A,b)\}$ und haben einen Widerspruch zur Fallannahme erhalten.
Wir wiederholen Fall 1, bis wir eine Ecke erhalten. Da der neue Vektor im Vergleich zum Vorgängervektor eine Null-Komponente mehr hat, erhalten wir spätestens nach n Iterationen eine Ecke. □

Optimale Lösungen sind aber im Allgemeinen nicht notwendigerweise Ecken: Wenn wir x_1 unter den Nichtnegativitätsbedingungen $x_1, x_2 \geq 0$ minimieren, sind alle Werte $(0, x_2)$ mit $x_2 \geq 0$ minimal, obwohl nur $(0,0)$ eine Ecke ist. In diesem Fall ist die Lösung des linearen Programms mehrdeutig.

Korollar 6.6 *Die nichtleere Lösungsmenge $L(A,b) = \{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}$ hat eine Ecke.*

Beweis Wende Lemma 6.5 mit $c = (0, \ldots, 0)$ an. □

Eine algebraische Charakterisierung ist für den Simplex-Algorithmus von zentralem Interesse. Diese Charakterisierung ist Inhalt des folgenden Satzes.

Satz 6.7 *$x \in L(A,b)$ sei eine Lösung mit $B = \{j \mid x_j > 0\}$. Die Matrix A besitze die Spalten $A_1, \ldots, A_n$ (also $A = (A_1, \ldots, A_n)$) und die Matrix A_B bestehe aus den Spalten A_i für $i \in B$. Wenn B nicht-leer ist, dann gilt*

$$x \text{ ist eine Ecke} \iff \text{die Spalten von } A_B \text{ sind linear unabhängig.}$$

Beweis: Wir nehmen $B \neq \emptyset$ an und erhalten deshalb auch $x \neq 0$.
$\Rightarrow$ Angenommen, A_B hat linear abhängige Spalten. Dann gibt es einen Vektor $y_B \neq 0$ mit $A_B \cdot y_B = 0$. Wir ergänzen y_B durch Null-Komponenten zu einem Vektor $y \in \mathbb{R}^n$ und es gilt $Ay = 0$. Wähle einen Skalar $\lambda > 0$, so dass $x \pm \lambda y \geq 0$. Dies ist wegen $y_i = x_i = 0$ für $i \notin B$ und $x_i > 0$ für $i \in B$ möglich. Aus

$$A(x \pm \lambda y) = Ax \pm \lambda \cdot \underbrace{(Ay)}_{=0} = Ax = b$$

erhalten wir $x \pm \lambda y \in L(A, b)$ und x ist keine Ecke.
$\Leftarrow$ Angenommen, der Vektor x ist keine Ecke. Dann existiert nach Definition ein $y \neq 0$ mit $x \pm y \in L(A, b)$. Wegen

$$Ax + Ay = b, \quad Ax - Ay = b$$

gilt $Ay = 0$. Aus

$$x + y \geq 0, \quad x - y \geq 0$$

erhalten wir: Wenn $x_i = 0$ (oder äquivalent, wenn $i \notin B$), dann ist $y_i = 0$. Sei y_B der Vektor, der aus den Werten von y zu Komponenten in B besteht. Es ist also insbesondere $y_B \neq 0$, da $y \neq 0$. Andererseits ist

$$0 = Ay = A_B \cdot y_B$$

und A_B hat wegen $y_B \neq 0$ linear abhängige Spalten. □

Aufgabe 72
Sei $P = \{x \in \mathbb{R}^n | Ax \leq b\}$ ein Polyeder.
Zeige: v ist eine Ecke von P dann und nur dann, wenn es n linear unabhängige Zeilen von A gibt, so dass die Ungleichungen dieser Zeilen im Punkt v als Gleichungen erfüllt sind.

Kapitel 7

Der Simplex-Algorithmus

George Dantzig hat Ende der vierziger Jahre den Simplex-Algorithmus entwickelt. Der Name leitet sich ab aus der anfänglichen Vermutung, dass das Verfahren nur für Simplices[1] geeignet sei. Der Simplex Algorithmus versucht, eine optimale Ecke zu bestimmen, indem ein Weg von benachbarten Ecken gelaufen wird. Simplex ist ein lokaler Suchalgorithmus der folgenden Form:

(1) Wähle eine beliebige Ecke x des Lösungspolyeders.

(2) Bestimme eine benachbarte Ecke x^{neu} mit niedrigerem Zielwert.

(3) Falls keine existiert, stoppe mit der Ausgabe x. Ansonsten setze $x := x^{\text{neu}}$ und gehe zu Schritt (2).

Nach Lemma 6.5 gibt es eine optimale Ecke, sofern die Lösungsmenge nicht-leer ist und wir den Zielwert nicht beliebig verkleinern können. Wir werden sehen, dass der Simplex-Algorithmus entweder eine optimale Ecke findet oder die beiden Sonderfälle erkennt.

Es sei m die Anzahl der Zeilen von A. Wir wollen triviale Fälle ausschließen. Wir fordern deshalb:

- Es gilt $\text{Rang}(A) = m$. Wenn $\text{Rang}(A) < m$ ist, sind Restriktionen überflüssig oder es gibt überhaupt keine Lösung. Wir entfernen gegebenenfalls überflüssige Zeilen aus der Matrix A und die entsprechenden Einträge im Vektor b.

- Der Lösungsraum ist nichtleer.

Um benachbarte Ecken aufzuspüren, führen wir den Begriff der „Basis“ und der zugehörigen Lösung ein.

Definition 7.1 Eine Menge $B \subseteq \{1, \ldots, n\}$ mit $|B| = m$ heißt Basis, falls A_B regulär und $A_B^{-1} b \geq 0$ ist. Der Vektor $x = (x_B, x_N)$ mit $x_B = A_B^{-1} b$ und $x_N = 0$ ist die Basislösung zu B.

[1] Ein Simplex ist ein n-dimensionales Polytop mit $n+1$ Ecken. Ein Beispiel ist das Polytop $\{\vec{x} \mid \sum_i x_i \leq 1, x \geq 0\}$. Ein Simplex ist also ein besonders einfaches Polytop.

Sei x^* eine Ecke, also $Ax^* = b$ und $x^* \geq 0$ für eine $m \times n$-Matrix A. Wir setzen $B := \{j \mid x_j^* > 0\}$ und unterscheiden zwei Fälle

Fall 1: Es gilt $|B| = m$. Wir nennen die Ecke x^* *nicht-entartet*. Wir wissen, dass m Spalten von A_B linear unabhängig sind und x^* eindeutig durch

- $x_j^* = 0$ für $j \notin B$ und
- $Ax^* = A_B x_B^* = b$

bestimmt ist, wobei A_B bzw. x_B^* aus A (bzw. x^*) und B durch Streichen aller Spalten (Komponenten) entstehen, die nicht zu B gehören. Die Matrix A_B ist regulär und wir erhalten

- $x_B^* = A_B^{-1} \cdot b$ und
- $x_N^* = 0$

für $N := \{1, \ldots, n\} \setminus B$. Die Komponenten $i \in B$ nennen wir *Basis-Komponenten* und die Komponenten $i \notin B$ nennen wir *Nichtbasis-Komponenten*. Bezogen auf die x_i^* sprechen wir von *Basis-Variablen* und *Nicht-Basis-Variablen*.

Fall 2: Es gilt $|B| < m$. In diesem Fall heißt die Ecke *entartet*. Ergänze die Matrix A_B durch $m - |B|$ linear unabhängige Spaltenvektoren der Matrix A zu einer regulären Matrix $A_{B'}$, wobei B' die Menge der m gewählten Spaltenindizes der Matrix A sei. Auch in diesem Fall ist

- $x_{B'}^* = A_{B'}^{-1} \cdot b$ und
- $x_{N'}^* = 0$

für $N' := \{1, \ldots, n\} \setminus B'$. Beachte, dass x^* in diesem Fall mehreren Basen zugeordnet ist.

Wir fassen zusammen: Eine Lösung $x \in L(A, b)$ ist genau dann eine Ecke, wenn eine m-elementige Teilmenge $B \subseteq \{1, \ldots, n\}$, die Basis, existiert mit

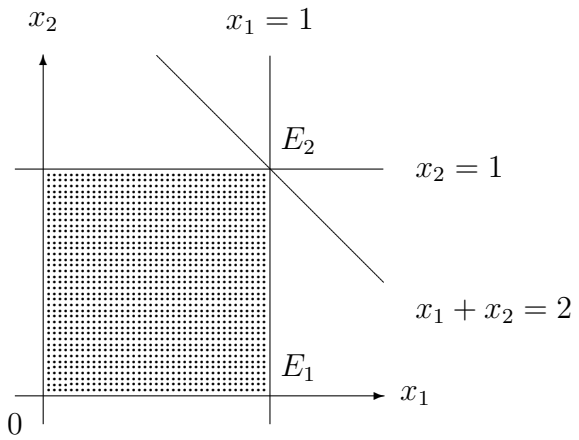
- $x_N = 0$ für $N = \{1, \ldots, n\} \setminus B$,
- A_B ist eine reguläre $m \times m$-Matrix,
- $x_B = A_B^{-1} \cdot b \geq 0$.

Wir wollen die Begriffe der Ecke, Basis, Basislösung und Degeneration anhand eines Beispiels erläutern.

Beispiel 7.1 Wir betrachten das folgende lineare Programm in kanonischer Form:

$$\text{minimiere } -x_1 - x_2, \text{ so dass} \qquad \begin{aligned} -x_1 &\geq -1 \\ -x_2 &\geq -1 \\ -x_1 - x_2 &\geq -2 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Der Lösungsraum hat die Form



wobei die Ecke E_2 offenbar die optimale Lösung ist. Wir führen Slackvariablen $s_3, s_4, s_5 \geq 0$ ein und erhalten

$$\begin{aligned} x_1 + s_3 &= 1 \\ x_2 + s_4 &= 1 \\ x_1 + x_2 + s_5 &= 2. \end{aligned}$$

In der Standardform ist das Problem also die Minimierung von $-x_1 - x_2$, so dass $x_1, x_2, s_3, s_4, s_5 \geq 0$ und

$$\underbrace{\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}}_{:=A} \cdot \begin{pmatrix} x_1 \\ x_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}}_{=:b}$$

Die Matrix A hat den Rang 3. Da $x = (1, 0, 0, 1, 1)$ eine Lösung ist und da die Matrix

$$A_B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

regulär ist, ist x nach Satz 6.7 eine Ecke, sie entspricht der Ecke E_1 in der obigen Abbildung. Auch $x = (1, 1, 0, 0, 0)$ ist eine Lösung und da die ersten beiden Spalten der Matrix A linear unabhängig sind, ist x eine Ecke (sie entspricht der Ecke E_2). Diese Ecke ist entartet, weil die Anzahl der Enträge ungleich 0 echt kleiner ist als der Rang der Matrix A. Zugeordnete Basen sind

$$\{1, 2, 3\},\ \{1, 2, 4\},\ \{1, 2, 5\}.$$

Die Ecke $x = (1, 1, 0, 0, 0)$ im $\mathbb{R}^5$ ist der Schnittpunkt der 6 Hyperebenen, die durch folgende Gleichungen definiert werden

$$\begin{aligned} &x_1 = 1, \quad x_1 + x_2 = 2, \quad s_4 = 0, \\ &x_2 = 1, \quad s_3 = 0, \qquad\quad s_5 = 0. \end{aligned}$$

Wir können eine Ecke im $\mathbb{R}^5$ als Schnittpunkt von 5 Hyperebenen beschreiben. Durch die zusätzliche Hyperebene ist die Ecke entartet.
Zuletzt betrachten wir beispielshaft die Indexmenge $B = \{1, 3, 4\}$. Die Matrix A_B ist regulär, denn

$$A_B = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Aber B ist keine Basis, denn die Eigenschaft $A_B^{-1} \cdot b \geq 0$ ist verletzt, da

$$A_B^{-1} \cdot b = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}.$$

B entspricht in der obigen Abbildung dem Schnittpunkt $(2, 0)$ der Geraden $x_2 = 0$ und $x_1 + x_2 = 2$ und liegt außerhalb des Lösungsbereichs.

Eine entartete Ecke im $\mathbb{R}^n$ entspricht geometrisch dem Schnittpunkt von mehr als n Hyperebenen. In Beispiel 7.1 haben wir gesehen, dass entartete Ecken mehrere Basen haben können und dass die Zielfunktion für diese verschiedenen Basen jeweils den gleichen Wert hat. Die Gefahr ist, dass wir im Simplex-Algorithmus die Basen der gleichen Ecke zyklisch durchlaufen, ohne die Ecke zu verlassen und dass damit unser Algorithmus nicht terminiert. Diese Situtation nennt man *Cycling*.
Für eine Basislösung x zur Basis B bezeichne $x_B, x_N, c_B, c_N, A_B, A_N$ die Einschränkungen auf Basis- bzw. Nicht-Basis-Komponenten. Für jede Lösung x gilt:

- $c^T \cdot x = c_B^T \cdot x_B + c_N^T \cdot x_N$,
- $Ax = A_B \cdot x_B + A_N \cdot x_N$,
- $x_B = A_B^{-1} \cdot b - A_B^{-1} \cdot A_N \cdot x_N$,
- $x_B, x_N \geq 0$.

Eine Iteration des Simplex-Algorithmus versucht, die gegenwärtige Basis B zur Basislösung x^* durch eine neue Basis B^{neu} zu ersetzen, so dass die Basislösung zu B^{neu} mindestens so gut wie die Basislösungen zu B ist. Zu einer festgewählten Basis B gilt für jede Lösung $x \in L(A, b)$

$$\begin{aligned} c^T \cdot x &= c_B^T \cdot x_B + c_N^T \cdot x_N \\ &= c_B^T \cdot \left(A_B^{-1} \cdot b - A_B^{-1} \cdot A_N \cdot x_N \right) + c_N^T \cdot x_N \\ &= \underbrace{c_B^T \cdot A_B^{-1} \cdot b}_{\text{aktueller Zielwert}} + \underbrace{\left(c_N^T - c_B^T \cdot A_B^{-1} \cdot A_N \right)}_{=\bar{c}_N} \cdot x_N. \end{aligned}$$

Um den aktuellen Zielwert zu erniedrigen, betrachten wir den zweiten Summanden. Wir unterscheiden zwei Fälle:

Fall 1: Es ist $\tilde{c}_j \geq 0$ für alle $j \in N$. Dann ist die Basislösung x^* zu B bereits optimal, denn für alle $x \in L(A,b)$ gilt wegen $x_N \geq 0$

$$c^T \cdot x^* = c_B^T \cdot A_B^{-1} \cdot b \leq \underbrace{c_B^T \cdot A_B^{-1} \cdot b}_{\text{aktueller Zielwert}} + \underbrace{\tilde{c}_N \cdot x_N}_{\geq 0}$$

Wir können den aktuellen Zielwert nicht senken!

Fall 2: Es gibt ein $j \in N$ mit $\tilde{c}_j < 0$. Sei $x^*(\lambda)$ das Resultat, wenn wir die Nichtbasisvariable x_j^* der Basislösung x^* zu B von 0 auf $\lambda \in \mathbb{R}_0^+$ erhöhen und die Basisvariablen wie folgt ändern:

$$\begin{aligned} x_N^*(\lambda) &:= (0, \cdots 0, \lambda, 0, \cdots, 0)^T \\ x_B^*(\lambda) &:= A_B^{-1} \cdot b - A_B^{-1} \cdot A_N \cdot x_N^*(\lambda). \end{aligned} \tag{7.1}$$

Die Einteilung in Basis- und Nichtbasisvariablen bezieht sich auf die Basis B zur Basislösung x^*. Für jedes $\lambda \geq 0$ mit $x_B^*(\lambda) \geq 0$ ist $x^*(\lambda)$ eine Lösung, denn

$$\begin{aligned} A \cdot x^*(\lambda) &= A_B \cdot x_B^*(\lambda) + A_N \cdot x_N^*(\lambda) \\ &= b - A_N \cdot x_N^*(\lambda) + A_N \cdot x_N^*(\lambda) = b. \end{aligned}$$

Wähle λ maximal mit $x_B^*(\lambda) \geq 0$. Wegen $x^*(0) = x^* \geq 0$ ist insbesondere $\lambda \geq 0$, aber der Fall $\lambda = 0$ kann für eine entartete Ecke nicht ausgeschlossen werden. Falls wir λ beliebig groß wählen können, ist das Minimum der Zielfunktion nicht nach unten beschränkt und der minimale Wert ist $-\infty$.

Algorithmus 7.2 **Der Simplex-Algorithmus**

(1) Gegeben ist eine $m \times n$-Matrix A mit $m \leq n$ und $\text{Rang}(A) = m$. Weiterhin gegeben ist ein Vektor b und die Zielfunktion $c^T \cdot x$. Gesucht ist eine Lösung $x^* \in L(A,b)$, so dass $c^T \cdot x^* = \min\{c^T \cdot x | x \in L(A,b)\}$. Wir setzen voraus, dass $L(A,b) \neq \emptyset$ ist.

(2) Beginne an einer Ecke x^* mit Basis B.

(3) Sei j die kleinste Komponente in $N = \{1, \ldots, n\} \setminus B$ mit der Eigenschaft $\tilde{c}_j < 0$. Wenn es kein solches j gibt, stoppe mit Ausgabe x^*.

Kommentar: Unsere Wahl von j wird das Cycling ausschließen. Existiert kein solches j, haben wir nach unserer Analyse in Fall 1 eine optimale Lösung gefunden.

(4) Wähle λ maximal mit $x^*(\lambda) \geq 0$.

(4a) Wenn $\lambda = \infty$, halte.

Kommentar: Gemäß der Analyse von Fall 2 ist $-\infty$ das Infimum.

(4b) Wenn $\lambda > 0$, dann gibt es $k \in B$ mit $x_k^* > x^*(\lambda)_k = 0$. Setze $B = (B \setminus \{k\}) \cup \{j\}$ und $x^* = x^*(\lambda)$.

(4c) Wenn $\lambda = 0$, wähle $k \in B$ minimal mit $x_k^* = 0$. Setze $B = (B \setminus \{k\}) \cup \{j\}$.

(5) Gehe zu Schritt (3).

Wir haben die Schritte (3) und (4a) bereits verifiziert und analysieren jetzt Schritt (4b). Wir wollen zu einer Ecke mit kleinerem Zielwert übergehen und erinnern uns an $\tilde{c}_j < 0$. Im Fall (4b) ist $\lambda > 0$, und wir haben die Zielfunktion um $\lambda \cdot \tilde{c}_j$ senken können. Mindestens eine Komponente k von $x_B^*(\lambda)$ wird auf 0 herabgesetzt, und wir ersetzen k durch j in der Basis. Warum ist B^{neu} mit Basislösung x^{neu} wieder eine Basis?
Es gilt $|B^{\text{neu}}| = m$ und $x^{\text{neu}} \geq 0$. Zu zeigen ist noch, dass $A_{B^{\text{neu}}}$ regulär ist. Seien $A_1, \ldots, A_n$ die Spaltenvektoren der Koefffizientenmatrix A. Aus (7.1) erhalten wir mit $A_B \cdot x_B^* = b$:

$$\begin{aligned} A_B \cdot x_B^*(\lambda) &= b - \lambda \cdot A_j \\ &= A_B \cdot x_B^* - \lambda \cdot A_j \end{aligned}$$

Wir können die beiden Matrix-Vektor-Produkte als Linearkombination der Spaltenvektoren der Matrix A_B schreiben und erhalten

$$\sum_{i \in B} x_i^*(\lambda) \cdot A_i = A_B \cdot x_B^*(\lambda) = A_B \cdot x_B^* - \lambda \cdot A_j = \sum_{i \in B} x_i^* \cdot A_i - \lambda \cdot A_j.$$

Aus $x_k^*(\lambda) = 0$ und $\lambda > 0$ folgt:

$$A_j = \frac{x_k^*}{\lambda} \cdot A_k + \sum_{i \in B \setminus \{k\}} \frac{x_i^*}{\lambda} \cdot A_i - \sum_{i \in B \setminus \{k\}} \frac{x_i^*(\lambda)}{\lambda} \cdot A_i.$$

Da x_k^* herabgesetzt wurde, ist $x_k^* \neq 0$. Da B eine Basis ist und da wir A_k als Linearkombination der Vektoren in $B \cup \{j\} \setminus \{k\}$ darstellen können, ist auch $B^{\text{neu}} = B \cup \{j\} \setminus \{k\}$ eine Basis mit einer besseren Basislösung.
Schließlich ist noch die Analyse von Schritt (4c) zu erbringen und wir müssen $\lambda = 0$ annehmen. (Dieser Fall tritt nur für entartete Ecken auf.) Der Simplexalgorithmus nimmt die Komponente j im Austausch mit einer Komponente $k \in B$ zu B hinzu.
Es besteht die bereits oben angesprochene Gefahr des Cycling: Eine Basis wird gewählt, die wiederum zur ursprünglichen Ecke gehört und wir verlassen die Ecke nicht mehr. In [Kar, Kapitel 2.3, Theorem 19], wird gezeigt, dass die Wahl eines kleinsten $j \in N$ in Schritt (3c) garantiert, dass wir irgendwann dennoch die Ecke x^* verlassen.
In Schritt (2) des Simplex-Algorithmus' setzen wir voraus, dass wir eine Ecke x^* des Lösungspolyeders kennen. In der Praxis, speziell bei linearen Programmen zu ökonomischen Problemen, ist häufig $x^* = 0$ eine Ecke. Aber der Nullvektor muss keine zulässige Lösung sein. Um eine Ecke zu einem linearen Programm in Standardform (also $Ax = b$) zu finden, führen wir Hilfsvariablen r ein und lösen das folgende lineare Programm mit dem Simplex-Algorithmus.

$$\text{minimiere } \textstyle\sum_i r_i \text{ so dass } \; Ax + r = b, \;\; x, r \geq 0.$$

Wir erhalten eine Basislösung dieses Programmes, wenn wir $(x, r) := (0, b)$ wählen; gegebenenfalls multipliziere Zeilen von A und Komponenten von b mit -1, um $b \geq 0$ zu garantieren. Dieses Programm hat genau dann das Optimum 0, also $r = 0$, wenn das Ausgangsproblem lösbar ist. Aus der optimalen Ecke $(x, r) = (x_0, 0)$ des Programms erhalten wir die Ecke x_0 zum Ausgangsproblem. Das entsprechende Simplex-Verfahren heißt *Zwei-Phasen Simplex*.

Aufgabe 73

Wir möchten entscheiden, ob ein gegebenes lineares Ungleichungssystem lösbar ist. Sei

$$\mathrm{LU} = \{(A, b) \in \mathbb{Z}^{m \times n} \times \mathbb{Z}^m \mid \begin{array}{l}\text{Es existiert ein } x \in \mathbb{R}^n \\ \text{mit } Ax \geq b\end{array}\}.$$

Gegeben sei ein Algorithmus für LU mit polynomieller Laufzeit. Gib einen Algorithmus mit polynomieller Laufzeit an, der das lineare Programmierungsproblem nur mit Hilfe des Algorithmus' für LU löst.
Fazit: Das Lösen von linearen Ungleichungen, bzw die Bestimmung irgendeiner Lösung und die lineare Programmierung sind äquivalente Probleme.

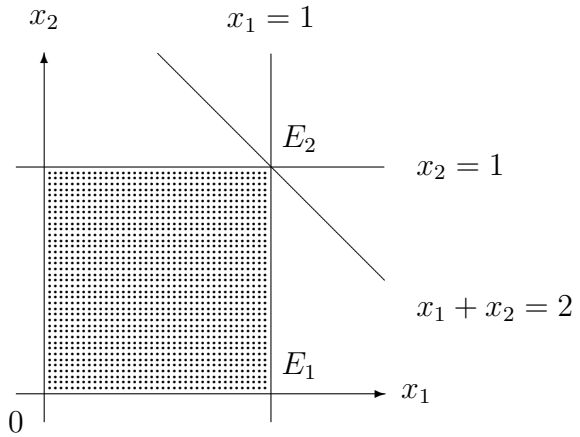
Beispiel 7.2 Wir lösen das lineare Programm aus Beispiel 7.1 mit Hilfe des Simplex-Algorithmus'. Die Zielfunktion $z(x_1, x_2, s_3, s_4, s_5) = -x_1 - x_2$ besitzt den Koeffizientenvektor

$$c = (-1, -1, 0, 0, 0)$$

und ist unter den Nebenbedingungen $x_1, x_2, s_3, s_4, s_5 \geq 0$ sowie unter den Bedingungen

$$\underbrace{\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}}_{:=A} \cdot \begin{pmatrix} x_1 \\ x_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}}_{=:b}$$

zu minimieren, wobei wir die Lösungsmenge



erhalten. Wir beginnnen mit der Ecke $x^* = (0, 0, 1, 1, 2)$ und der Basis $B = \{3, 4, 5\}$. Diese Ecke entspricht dem Nullpunkt in der Abbildung. Es ist $c^t \cdot x^* = 0$ und

$$A_B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad A_B^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ und } A_N = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

Wir berechnen $\tilde{c}_N$:

$$\begin{aligned} \tilde{c}_N &= c_N^T - c_B^T \cdot A_B^{-1} \cdot A_N = \begin{pmatrix} -1, & -1 \end{pmatrix} - \begin{pmatrix} 0, & 0, & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \\ &= \begin{pmatrix} -1, & -1 \end{pmatrix}. \end{aligned}$$

In Schritt (3) wählen wir $j = 1$ und dementsprechend ist

$$x_B^*(\lambda) = x_B^* \cdot b - A_B^{-1} \cdot A_N \cdot \begin{pmatrix} \lambda \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} \lambda \\ 0 \end{pmatrix} = \begin{pmatrix} 1 - \lambda \\ 1 \\ 2 - \lambda \end{pmatrix}.$$

Wir wählen stets $\lambda \geq 0$ maximal mit $x_B^*(\lambda) \geq 0$ und erhalten hier $\lambda = 1$. Die dritte Komponente in $x^*(1)$ ist gleich 0 und $B = \{1, 4, 5\}$ ist die neue Basis. Weiterhin ist

$$A_B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \quad A_B^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \text{ und } A_N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}.$$

und als Konsequenz

$$x_B^* = A_B^{-1} \cdot b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

Die Ecke $x^* = (1, 0, 0, 1, 1)$ entspricht dem Punkt E_1 in der Abbildung. Es gilt $c^T \cdot x^* = -1$ und

$$\begin{aligned} \tilde{c}_N &= c_N^T - c_B^T \cdot A_B^{-1} \cdot A_N \\ &= \begin{pmatrix} -1, & 0 \end{pmatrix} - \begin{pmatrix} -1, & 0, & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} -1, & 1 \end{pmatrix}. \end{aligned}$$

In Schritt (3) wählen wir $j = 2$ und erhalten

$$\begin{aligned} x_B^*(\lambda) &= x_B^* - A_B^{-1} \cdot A_N \cdot \begin{pmatrix} \lambda \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \lambda \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 - \lambda \\ 1 - \lambda \end{pmatrix}. \end{aligned}$$

Wir wählen $\lambda = 1$. Die Komponenten 4 und 5 aus $B = \{1, 4, 5\}$ sind in $x^*(1)$ auf 0 gesetzt und wir wählen $k = 4$. $B = \{1, 2, 5\}$ ist die neue Basis mit

$$A_B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \quad A_B^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix} \text{ und } A_N = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$$

und es ist

$$A_B^{-1} \cdot b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}.$$

Die Ecke $x^* = (1, 1, 0, 0, 0)$ entspricht dem Punkt E_2 in der Abbildung. Es gilt $c^T \cdot x^* = -2$. Wir berechnen $\tilde{c}_N$:

$$\begin{aligned} \tilde{c}_N &= c_N^T - c_B^T \cdot A_B^{-1} \cdot A_N \\ &= \begin{pmatrix} 0 & 0 \end{pmatrix} - \begin{pmatrix} -1 & -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix}. \end{aligned}$$

Da $\tilde{c}_N \geq 0$, ist die Ecke $x^* = (1, 1, 0, 0, 0)$ optimal.

Wir führen eine stark vereinfachte Laufzeitanalyse des Simplex-Algorithmus' im worst case Fall durch. In jeder Iteration müssen wir Matrizen invertieren und multiplizieren, was offensichtlich effizient gelingt. Daher ist die Laufzeit bis auf einen polynomiellen Faktor durch die Anzahl der Ecken bestimmt. Da jede Ecke einer Basis, also einer m-elementigen Teilmenge von $\{1, \ldots, n\}$ entspricht, gibt es höchstens

$$\binom{n}{m} \leq 2^n$$

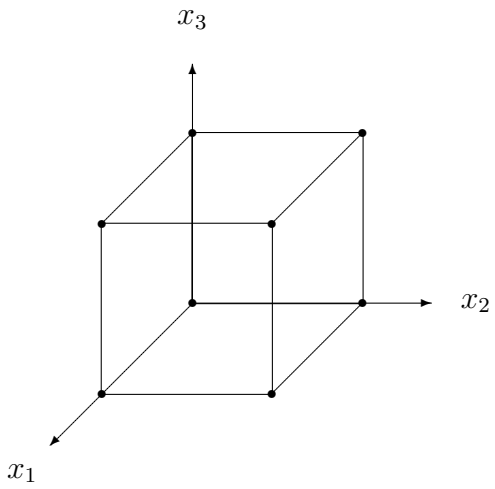
Ecken.

Satz 7.3 *Der Simplex-Algorithmus löst ein lineares Programm mit n Variablen und $m \leq n$ Restriktionen in Zeit $O(poly(n) \cdot 2^n)$.*

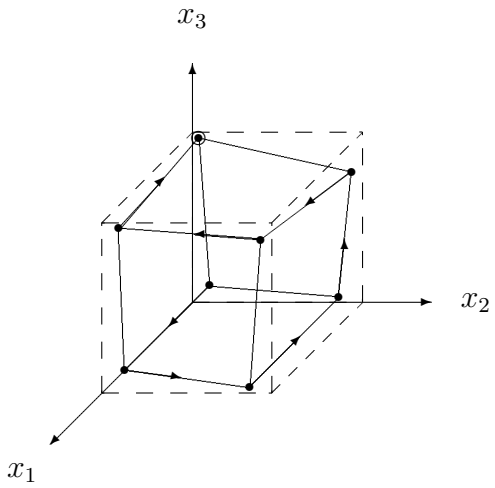
Der exponentielle Faktor in der Laufzeitschranke des Simplex-Algorithmus' folgt aus der Abschätzung der Eckenanzahl des Polyeders. Es kann tatsächlich 2^n Ecken geben, wie das Beispiel des n-dimensionalen Würfels W_n zeigt. Es ist

$$W_n := \{x \in \mathbb{R}^n \mid 0 \leq x_i \leq 1 \text{ für } i = 1, \ldots, n\}$$

W_n hat bei n Variablen die 2^n Ecken $\{0, 1\}^n$.



Für Analysen zur Laufzeit im Average-Case verweisen wir auf [Schr].
V. Klee und G.J. Minty [Klee] zeigen, dass der Simplex-Algorithmus, abhängig von der Pivotstrategie, in exponentiell viele Ecken gezwungen werden kann. (Avis und Chvátal [ACh] haben das negative Beispiel auf Blands Pivotwahl übertragen). Die folgende Abbildung zeigt die Idee des worst-case Polyeders von Klee und Minty (vergleiche [PaSt, Kapitel 8.7]). Wir „verwackeln“ (perturbieren) den Würfel geringfügig, so dass der Simplex-Algorithmus anstatt direkt vom Startpunkt in die optimale Ecke zu gehen, zunächst eine andere, benachbarte Ecke bevorzugt und anschließend über die übrigen Ecken zum Optimum gelangt.



In der Praxis haben der Simplex-Algorithmus und seine Varianten aber akzeptable Laufzeit. So bleibt der Simplex-Algorithmus zum Beispiel der bevorzugte Algorithmus in inkrementellen linearen Programmierproblemen, also Problemen, in denen sukzessive Restriktionen eingefügt werden. Eine weitere Stärke ist die exakte Berechnung des Optimums.
Es ist bisher nicht bekannt, ob zufällige Pivotstrategien zu polynomieller Laufzeit führen. Es ist auch nicht ausgeschlossen, dass deterministische Pivotstrategien mit polynomieller Laufzeit existieren.

Die Frage, ob Polynomialzeit-Algorithmen für die lineare Programmierung existieren, war bis 1979 offen. Der Durchbruch gelang L.G. Khachiyan [Kar] mit der Ellipsoid-Methode, dem ersten Polynomialzeit-Verfahren. Der Algorithmus konnte sich jedoch in der Praxis nicht bewähren.

Aufgabe 74
Betrachte das Problem des Lösens linearer Ungleichungssysteme. Es ist ein Punkt im Polyeder

$$P = \{x \in \mathbb{R}^n \mid Ax \geq b\}$$

zu bestimmen(falls ein solcher existiert). Sei L die Gesamtlänge der Binärdarstellungen der Einträge in A und b.
Zeige, dass das Problem in polynomieller (in L) Zeit lösbar ist, wenn folgendes gegeben ist.

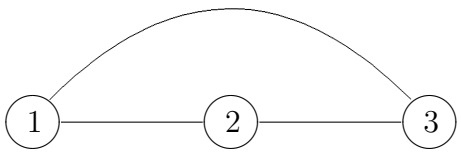
- Zu Beginn ist eine konvexe Menge M bekannt, in der die Lösungsmenge liegt. Die Menge habe ein Volumen von höchstens $2^{O(L)}$.
- P habe ein Volumen von mindestens $1/2^{O(n \cdot L)}$.
- Sei x_M der Schwerpunkt der Menge M und sei H eine Hyperebene, die x_M enthält. Desweiteren sei H^* einer der beiden Halbräume zu H. Dann gibt es eine in polynomieller Zeit (sogar in n) laufende Prozedur, die eine weitere konvexe Menge M' mit den folgenden Eigenschaften bestimmt:
  - M' enthält den Schnitt von M mit dem Halbraum H^*,
  - der Quotient der Volumina von M' und M ist höchstens $2^{-1/(2 \cdot n+2)}$,
  - und die Prozedur kann wiederum auf die neue konvexe Menge M' angewandt werden.

Fazit: Da das Problem des Lösens linearer Ungleichungen äquivalent zum linearen Programmierungsproblem ist, erhält man mit einem solchen Schema einen Polynomialzeitalgorithmus für das lineare Programmierungsproblem.

Die Lücke zwischen Theorie und Praxis schloss 1984 M. Karmarkar [Karm], auf dessen Algorithmus die Interior-Point-Methode, die wir in Kapitel 8 kennenlernen werden, aufbaut. Interior-Point-Algorithmen sind dem Simplex-Verfahren vor Allem für große Programme überlegen, wenn nur eine approximative Lösung verlangt wird.

7.1 Vollständig unimodulare Programme

Lineare Programme haben im Allgemeinen reellwertige Lösungen. In wichtigen Fällen, wie etwa dem Matching Problem für bipartite Graphen und dem Flussproblem für ganzzahlige Kapazitäten sind alle Ecken überraschenderweise ganzzahlig. Andererseits ist das Matching Problem für allgemeine Graphen „bösartig“. Dazu betrachten wir den Graphen



Ein maximales Matching besteht aus genau einer Kante. Für das lineare Programm aus Beispiel 6.1 setzen wir

$$x_{\{1,2\}} = x_{\{1,3\}} = x_{\{2,3\}} = \frac{1}{2}$$

und der Vektor $x := \left(x_{\{1,2\}}, x_{\{1,3\}}, x_{\{2,3\}}\right)$ ist eine zulässige, aber für das Matching Problem nicht verwertbare Lösung mit Wert $\frac{3}{2}$.
Der Begriff einer vollständig unimodularen Matrix spielt für die Ganzzahligkeit von Ecken eine wesentliche Rolle.

Definition 7.4 Eine Matrix A heißt vollständig unimodular, falls $\det B \in \{-1, 0, 1\}$ für jede quadratische Teilmatrix B von A gilt.

Insbesondere hat eine vollständig unimodulare Matrix nur Einträge aus $\{-1, 0, 1\}$, da jeder Eintrag eine quadratische Teilmatrix ist.

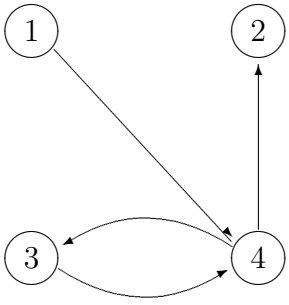
Definition 7.5 Zu einem gerichteten Graphen $G = (V, E)$ definieren wir die Knoten-Kanten-Inzidenzmatrix $A = [a_{v,e}]_{v \in V, e \in E}$ wie folgt

$$a_{v,e} = \begin{cases} +1 & \text{falls } e = (v, w) \text{ für ein } w \in V \\ -1 & \text{falls } e = (w, v) \text{ für ein } w \in V \\ 0 & \text{sonst.} \end{cases}$$

Für einen ungerichteten Graphen $G = (V, E)$ ist $a_{v,e} = 1$ genau dann, wenn $v \in e$.

In der Knoten-Kanten-Inzidenzmatrix zu einem gerichteten Graphen steht also in jeder Spalte genau eine $+1$ und genau eine -1, die übrigen Einträge sind Null. Für ungerichtete Graphen enthält jede Spalte genau zwei Einsen und die restlichen Einträge sind ebenfalls Null.

Beispiel 7.3 Wir betrachten den Graphen



und erhalten die Knoten-Kanten-Inzidenzmatrix

$a_{v,e}$	$(1,4)$	$(3,4)$	$(4,2)$	$(4,3)$
1	$+1$	0	0	0
2	0	0	-1	0
3	0	$+1$	0	-1
4	-1	-1	$+1$	$+1$

Für ungerichtete Kanten sind die negativen Einträge durch $+1$ zu ersetzen.

Aus Satz 7.6 wird folgen, dass das Polyeder zum Flussproblem wie auch das Polyeder zum gewichteten Matching Problem für bipartite Graphen nur ganzzahlige Ecken hat.

Satz 7.6

(a) Die Knoten-Kanten-Inzidenzmatrix für einen gerichteten Graphen oder einen ungerichteten, bipartiten Graphen ist vollständig unimodular.

(b) Falls die Matrix A vollständig unimodular und b ein ganzzahliger Vektor ist, haben die Ecken der Polyeder $\{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}$ und $\{x \in \mathbb{R}^n \mid Ax \geq b\}$ nur ganzzahlige Koordinaten.

Aufgabe 75

(a) **Zeige,** dass die Knoten-Kanten Inzidenzmatrix eines gerichteten Graphen vollständig unimodular ist.

(b) **Zeige,** dass die Knoten-Kanten Inzidenzmatrix eines bipartiten ungerichteten Graphen vollständig unimodular ist.

(c) Beweise Satz 7.6 (b).

Hinweis: Der Determinantenentwicklungssatz könnte sich als nützlich erweisen. Beachte, dass Matrizen mit linear abhängigen Zeilen oder Spalten die Determinante 0 besitzen.

Aufgabe 76

Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit $|V|$=n. Das INDEPENDENT SET Problem fragt nach einer maximalen Knotenmenge $W \subseteq V$, so dass für je zwei Knoten $v, w \in W$ gilt: $\{v, w\} \notin E$. Das Problem lässt sich als lineares Programm formulieren, wenn man nur ganzzahlige Lösungen zulässt: Maximiere $\sum_{v \in V} x_v$, so dass für alle $\{v, w\} \in E$ gilt: $x_v + x_w \leq 1$, sowie $x_v \geq 0$ für jeden Knoten v. Die unabhängige Knotenmenge besteht dann aus allen Knoten v mit $x_v = 1$.
Zeige, dass bei der kanonischen Formulierung von INDEPENDENT SET als lineares Programm nichtganzzahlige Lösungen existieren, welche um einen Faktor $\Omega(n)$ größer sind als das ganzzahlige Optimum.

Aufgabe 77

Formuliere die folgenden Probleme als lineare Programme.

(a) Das Problem des kürzesten Pfades zwischen zwei Knoten in einem gegebenen gerichteten und mit Kantenlängen gewichteten Graphen. Was passiert, wenn man negative Kantengewichte erlaubt?

(b) Gegeben sind m Erzeuger eines Produktes und n Verbraucher, wobei Erzeuger e_i von dem Produkt die Menge $a_i \in \mathbb{N}$ besitzt und Verbraucher v_j die Menge $b_j \in \mathbb{N}$ benötigt. Der Transport einer Einheit des Produktes von e_i zu v_j verursacht Kosten $c_{i,j}$. Die Wünsche aller Verbraucher sollen bei insgesamt niedrigsten Kosten befriedigt werden.

Warum sind alle Ecken der zu diesen linearen Programmen gehörenden Lösungsräume ganzzahlig?

Man kann zeigen, dass die Knoten-Kanten-Inzidenzmatrix eines ungerichteten Graphen genau dann vollständig unimodular ist, wenn der Graph bipartit ist. Aus Satz 7.6 (b) folgt ferner, dass der maximale Fluss in einem Netzwerk mit ganzzahligen Kapazitäten ganzzahlig ist. Diese Eigenschaft nennt man auch die Ganzzahligkeitseigenschaft für Flüsse.

Wir haben untersucht, wann die Ecken nur ganzzahlige Koordinaten haben. Warum betrachten wir nicht einfach nur ganzzahlige Lösungen? Die Bestimmung eines ganzzahligen optimalen Vektors ist ein äußerst schwieriges Problem, wie schon aus der $\mathcal{NP}$-Vollständigkeit der 0-1 Programmierung folgt!

Aufgabe 78

(a) Gegeben sei eine beliebige Menge von Vektoren $F \subseteq \{0,1\}^n$. Zeige, dass die konvexe Hülle von F durch ein System von linearen Ungleichungen charakterisiert werden kann. Hinweis: Beschreibe, wie ein beliebiger Vektor $v \in \{0,1\}^n$ aus der Menge $[0,1]^n \subseteq \mathbb{R}^n$ durch eine Hyperebene entfernt werden kann, ohne dass sich nichtganzzahlige Ecke ergeben.

(b) Gegeben sei ein ganzzahliges lineares Programm. Schreibe das Programm als allgemeinenes lineares Programm (gegebenenfalls mit exponentiell vielen Ungleichungen).

Aufgabe 79
Eine Firma stellt während der Semesterferien Studenten ein, um durch sie Computertageskurse betreuen zu lassen. Die optimale Planung für die kommenden Ferien soll mittels eines linearen Programms erstellt werden. Dabei sind die folgenden Gegebenheiten relevant:
Für jeden angeheuerten Studenten ist zunächst ein Fixbetrag F zu zahlen, der unabhängig davon anfällt, wieviele Kurse der betreffende Student nun wirklich anbietet. Von jedem der N Tage der Semesterferien ist bekannt, wieviele Kurse die Firma an diesem Tag anbieten muss, diese Nachfragezahlen seien $n_1, \ldots, n_N$. Für einen abgehaltenen Kurs erhält die Firma den Preis P und muss einen Studenten einsetzen, der dafür einen Lohn L erhält. Des weiteren ist vereinbart, dass ein Student, der an zwei aufeinanderfolgenden Tagen einen Kurs abhält einen Zuschlag in Höhe von Z erhält. (Arbeitet ein Student also t Tage in Folge, so erhält er dafür $tL + (t-1)Z$). Die Firma möchte natürlich ihren Gewinn maximieren.
Bestimme die Variablen und die linearen Ungleichungen sowie die Zielfunktion für diese Aufgabe. **Begründe** Deinen Ansatz. **Warum** liefert Deine Implementierung nur ganzzahlige Lösungen?

7.2 Dualität und komplementäre Slackness

Wir beginnen mit motivierenden Bemerkungen zum Lemma von Farkas. Wenn ein lineares Gleichungssystem $Ax = b$ keine Lösung hat, ist eine Zeile von den anderen linear abhängig und diese spezielle Abhängigkeit wird vom Vektor b nicht „eingehalten“: Es gibt ein y mit $y^T \cdot A = 0$ und $y^T \cdot b < 0$. Zum Beispiel hat das folgende, lineare Gleichungssystem

$$\begin{pmatrix} 3 & 4 & 2 \\ 0 & 4 & 0 \\ 6 & 12 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 5 \end{pmatrix}$$

keine Lösung und der gesuchte Vektor ist $y^T = (2, 1, -1)$. Wir möchten eine entsprechende Aussage für Polyeder erhalten. Das folgende Resultat geht auf G. Farkas (1894) zurück.

Satz 7.7 *Das Farkas Lemma*
Zu einer Matrix A und einem Vektor b ist genau eine der folgenden Aussagen wahr:

(a) Es gibt ein x mit $Ax = b$ und $x \geq 0$.

(b) Es gibt ein y mit $y^T \cdot A \geq 0$ und $y^T \cdot b < 0$.

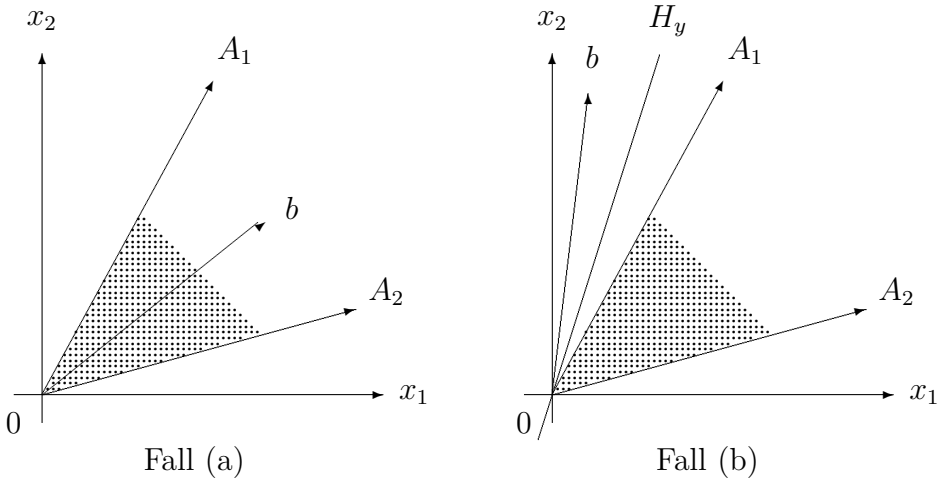
Bevor wir das Lemma von Farkas beweisen, wollen wir die Aussage geometrisch interpretieren und motivieren. Die Spaltenvektoren $A_1, \ldots, A_n$ der Matrix A erzeugen den Kegel

$$\text{cone}(A) = \{\sum_{i=1}^{n} x_i A_i \mid x_1, \ldots, x_n \geq 0\} = \{Ax \mid x \geq 0\}.$$

Das Farkas Lemma besagt, dass entweder der Vektor b in diesem Kegel liegt (Aussage (a)) oder dass eine Hyperebene

$$H_y := \{x \mid y^T \cdot x = 0\},$$

existiert, die b und cone(A) strikt trennt (Aussage (b)).



Beweis: Wir zeigen zunächst, dass sich die Aussagen (a) und (b) gegenseitig ausschließen. Angenommen, es gibt x und y mit $Ax = b$, $x \geq 0$ sowie $y^T \cdot A \geq 0$ und $y^T \cdot b < 0$. Wir erhalten

$$y^T \cdot A \cdot x = y^T \cdot b < 0.$$

Wegen $y^T \cdot A \geq 0$ und $x \geq 0$ folgt der Widerspruch, da

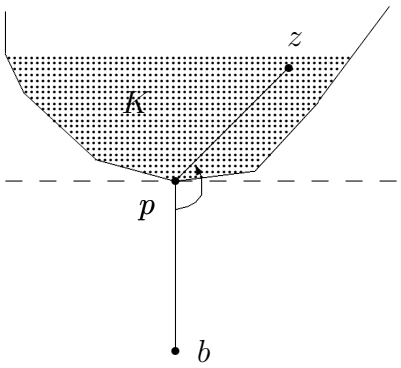
$$y^T \cdot A \cdot x \geq 0.$$

Es genügt, die Aussage (b) aus der Voraussetzung herzuleiten, dass es kein $x \geq 0$ mit $Ax = b$ gibt. Wir betrachten die abgeschlossene und konvexe[2] Menge

$$K := \{Ax \mid x \geq 0\}.$$

Nach Voraussetzung ist $b \notin K$. Sei p der Punkt in K mit kürzestem Abstand von b.

[2] Eine Menge K ist konvex, falls mit $x, y \in K$ und $\mu \in [0,1]$ auch $\mu \cdot x + (1-\mu) \cdot y$ in K liegt, falls also die Gerade $\overline{xy}$ in K liegt.



Wir behaupten, dass, für alle $z \in K$, $(z-p)^T \cdot (b-p) \leq 0$ gilt. Allgemein gilt für zwei Vektoren v, w

$$\cos \angle(v,w) = \frac{v^T \cdot w}{||v|| \cdot ||w||}$$

und die Behauptung ist, dass der Winkel zwischen $z-p$ und $b-p$ zwischen 90° und 270° ist. Die Behauptung folgt somit unmittelbar aus der obigen Abbildung.
Wir setzen $y := p - b$. Da alle Elemente in K die Form Ax mit $x \geq 0$ haben, gilt nach den vorherigen Überlegungen für jeden Vektor x, dass

$$(Ax-p)^T \cdot \underbrace{(b-p)}_{=-y} \leq 0.$$

Als erstes zeigen wir, dass $y^T \cdot A \geq 0$. Wegen $p \in K$ existiert ein $w \geq 0$ mit $p = A \cdot w$ und es ist

$$0 \leq (Ax-p)^T \cdot y = (Ax-Aw)^T \cdot y = (A(x-w))^T \cdot y \tag{7.2}$$

für jeden Vektor $x \geq 0$. Durch Transponieren erhalten wir

$$y^T \cdot A(x-w) \geq 0. \tag{7.3}$$

Wir wählen $x := w + e_i$ mit e_i als i-tem Einheitsvektor. Wegen $w \geq 0$ ist auch $x \geq 0$ und wir erhalten aus (7.3) mit $x - w = e_i$, dass die i-te Komponente von $y^T \cdot A$ größer oder gleich 0 ist und wir haben

$$y^T \cdot A \geq 0$$

gezeigt. Es ist noch $y^T \cdot b < 0$ zu zeigen. Es gilt

$$y^T \cdot b = y^T \cdot (p-y) = y^T \cdot p - y^T \cdot y, \tag{7.4}$$

da $y = p - b$. Aus (7.2) folgt mit $x = 0$, dass $0 \leq -p^T \cdot y$ oder äquivalent $y^T \cdot p \leq 0$. Wegen der Voraussetzung $b \notin K$ und $p \in K$ ist $y = p - b \neq 0$ und es gilt $y^T \cdot y = ||y||^2 > 0$. Wir erhalten den zweiten Teil der Behauptung aus 7.4, denn

$$y^T \cdot b = \underbrace{y^T \cdot p}_{\leq 0} - \underbrace{y^T \cdot y}_{>0} < 0.$$

□

Analog zeigt man die folgende Variante des Lemmas von Farkas:

Satz 7.8 *Zu einer Matrix A und einem Vektor b ist genau eine der folgenden Aussagen wahr:*

(a) Es gibt ein x mit $Ax \leq b$.

(b) Es gibt ein $y \geq 0$ mit $y^T \cdot A = 0$ und $y^T \cdot b < 0$.

In der linearen Programmierung gilt ein überraschendes Dualitätsprinzip. Die zur Erläuterung wesentlichen Begriffe stellen wir jetzt zusammen.

Definition 7.9 Für

$$\begin{array}{ll} (P) & \text{minimiere } c^T \cdot x, \text{ so dass } A \cdot x = b \text{ und } x \geq 0 \\ (D) & \text{maximiere } y^T \cdot b, \text{ so dass } y^T \cdot A \leq c \end{array}$$

ist (P) das primale und (D) das zugehörige duale Programm.

Beispiel 7.4 Wir betrachten das primale lineare Programm (P) aus Beispiel 7.2, d.h. minimiere $c^T \cdot x$ unter den Restriktionen $Ax = b$ und $x \geq 0$, wobei:

$$c = \begin{pmatrix} -1 \\ -1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad A^T = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{und} \quad b^T = \begin{pmatrix} 1 & 1 & 2 \end{pmatrix}$$

Das zugehörige duale lineare Programm lautet:

$$(D) \text{ maximiere } y_1 + y_2 + 2y_3, \text{ so dass } \begin{array}{ll} y_1 + y_3 & \leq -1 \\ y_2 + y_3 & \leq -1 \\ y_1, y_2, y_3 & \leq 0 \end{array}$$

Durch Fallunterscheidung $y_3 = 0$ und $y_3 < 0$ erhält man, dass $y = (-1, -1, 0)$ die optimale Ecke von (D) mit demselben Zielwert -2 wie im primalen Programm (P) ist.

Wir können sofort eine offensichtliche Beziehung zwischen den Zielfunktionen des primalen und des dualen Progamms herstellen.

- Wähle einen Vektor y und summiere die Gleichungen gemäß y, d.h. berechne

$$y^T \cdot (A \cdot x) = y^T \cdot b.$$

- Wenn wir $y^T \cdot A \leq c$ fordern, d.h. wenn wir fordern, dass y eine Lösung des dualen Programms ist, dann folgt wegen $x \geq 0$

$$y^T \cdot b = y^T \cdot A \cdot x \leq c^T \cdot x.$$

Der optimale Wert des dualen (Maximierungs-)Programms liegt also stets unterhalb des Wertes des primalen (Minimierungs-)Programms.
Im Beispiel 7.4 ist das endliche Optimum der Zielfunktionen für die primale und die duale Aufgabe identisch. Diese Eigenschaft gilt allgemein wie J. von Neumann (1947) formuliert hat und dann von D. Gale, H.W. Kuhn und A.W. Tucker (1951) gezeigt wurde.

Satz 7.10 (Dualitäts-Theorem) *Sei x eine Lösung des primalen Problems (P) und y eine Lösung des dualen Problems (D), sowie x_{opt} und y_{opt} endliche, optimale Lösungen von (P) bzw. (D). Dann gilt*

(a) $c^T \cdot x \geq b^T \cdot y$

(b) $c^T \cdot x_{\text{opt}} = b^T \cdot y_{\text{opt}}$

Eigenschaft (a) heißt schwache Dualität und Eigenschaft (b) starke Dualität.

Beweis: Die schwache Dualität haben wir bereits nachgewiesen.
Für **(b)** genügt wegen der schwachen Dualität der Nachweis von

$$c^T \cdot x_{\text{opt}} \leq b^T \cdot y_{\text{opt}} \tag{7.5}$$

Sei $c_{\text{opt}} := c^T \cdot x_{\text{opt}}$ der optimale Zielwert. Angenommen, es gibt kein y mit $y^T \cdot A \leq c$ und $c_{\text{opt}} \leq b^T \cdot y$. Um das Farkas Lemma anzuwenden, setzen wir:

$$\overline{A} := \begin{pmatrix} A^T \\ -b^T \end{pmatrix} \quad \text{und} \quad \overline{b} := \begin{pmatrix} c \\ -c_{\text{opt}} \end{pmatrix}$$

Es existiert nach Annahme kein Vektor $\overline{y}$ mit $\overline{A} \cdot \overline{y} \leq \overline{b}$. Wir wenden die Variante des Farkas Lemma 7.8 an und erhalten ein $\overline{z} \geq 0$ mit $\overline{z}^T \cdot \overline{A} = 0$ und $\overline{z}^T \cdot \overline{b} < 0$. Mit anderen Worten, für $\overline{z} = (z, \lambda)^T$ gilt

(1) $z \geq 0$ und $\lambda \geq 0$,

(2) $\begin{pmatrix} z & \lambda \end{pmatrix}^T \cdot \begin{pmatrix} A^T \\ -b^T \end{pmatrix} = 0$ und damit $Az = \lambda b$.

(3) $\overline{z}^T \cdot \overline{b} = z^T \cdot c - c_{\text{opt}} \cdot \lambda < 0$ und damit $z^T \cdot c < c_{\text{opt}} \cdot \lambda$.

Wir treffen eine Fallunterscheidung.
Fall 1: $\lambda = 0$. Dann ist $Az = 0$, $z \geq 0$ und

$$z^T \cdot c = c^T \cdot z < 0.$$

Also ist auch $x_{\text{opt}} + \mu \cdot z$ für jedes $\mu \geq 0$ eine Lösung und x_{opt} ist nicht optimal — Widerspruch.

Fall 2: $\lambda > 0$. Wir setzen $x = \frac{z}{\lambda}$ und erhalten

$$A \cdot x = \frac{1}{\lambda} \cdot (A \cdot z) = \frac{\lambda b}{\lambda} = b$$

sowie

$$c^T \cdot x = c^T \cdot \frac{z}{\lambda} = \frac{1}{\lambda} \cdot \left(c^T \cdot z\right) < \frac{c_{\text{opt}} \cdot \lambda}{\lambda} = c_{\text{opt}}.$$

Daher ist x eine Lösung des primalen Problems und x_{opt} ist nicht optimal — Widerspruch. □

Aufgabe 80

Zwei Personen spielen folgendes Spiel: Jeder wählt heimlich ein Element aus $\{1, \ldots, k\}$. Dann legen beide ihre Wahl offen. Eine Matrix enthält an der Stelle i, j den (positiven oder negativen) Gewinn von Spieler 1 (negative Gewinne werden von Spieler 1 an Spieler 2 gezahlt, positive Gewinne von Spieler 2 an Spieler 1 gezahlt).

Die Spieler dürfen randomisiert wählen. Spieler 1 sucht also eine Wahrscheinlichkeitsverteilung, welche seinen Gewinn maximiert, Spieler 2 sucht eine Wahrscheinlichkeitsverteilung, welche seinen Gewinn maximiert.

Zeige, daß der optimale erwartete Gewinn von Spieler 1 gleich dem optimalen erwarteten Verlust von Spieler 2 ist.

Bemerkung 7.1 Der Zusammenhang zwischen primalen und dualen Programmen kann verallgemeinert werden. Dazu betrachte eine verallgemeinerte Form

$$(P) \text{ minimiere } c_1^T \cdot x_1 + c_2^T \cdot x_2, \text{ so dass } \quad \begin{array}{l} A_1 \cdot x = b_1, \\ A_2 \cdot x \geq b_2 \text{ und } x_1 \geq 0 \end{array}$$

des primalen Programms. Beachte, dass gemischte Bedingungen vorliegen: Einige Bedingungen entsprechen Gleichungen, die anderen Bedingungen entsprechen Ungleichungen. Man kann zeigen, dass das duale Programm die Form

$$(D) \text{ maximiere } b_1^T \cdot y_1 + b_2^T \cdot y_2, \text{ so dass } \quad \begin{array}{l} y_1^T \cdot A_1 \leq c_1, \\ y_2^T \cdot A_2 \leq c_2 \text{ und } y_2 \geq 0 \end{array}$$

hat.

Aufgabe 81

Zeige, dass die Optima für (P) und (D) übereinstimmen.

Die allgemeinen Transformationsregeln sind wie folgt:

- Einer primalen Ungleichung ist eine nicht-negative duale Variable zugeordnet.
- Einer primalen Gleichung ist eine nicht-vorzeichenbehaftete duale Variable zugeordnet.

- Einer primalen nicht-negativen Variablen ist eine duale Ungleichung zugeordnet.
- Einer primalen nicht-vorzeichenbehafteten Variablen ist eine duale Gleichung zugeordnet.

Wir besprechen jetzt den für uns wichtigen algorithmischen Aspekt des Dualitätstheorems. Angenommen, wir haben eine Lösung x für das primale Ausgangsproblem

$$(P) \text{ minimiere } c^T \cdot x, \text{ so dass } A \cdot x = b \text{ und } x \geq 0$$

und eine Lösung y für das duale Problem. Dann ist insbesondere für Slackvariablen $s \geq 0$

$$y^T \cdot A + s = c,$$

und wir können y durch (y, s) ersetzen. Das duale Problem lautet dann

$$(D) \text{ maximiere } y^T \cdot b, \text{ so dass } y^T \cdot A + s = c \text{ und } s \geq 0.$$

Die *Dualitätslücke* gibt den Abstand zwischen der Lösung x des primalen und der Lösung (y, s) des dualen Problems an:

$$\begin{aligned} c^T \cdot x - b^T \cdot y &= c^T \cdot x - (Ax)^T \cdot y \\ &= x^T \cdot c - x^T \cdot A^T \cdot y \\ &= x^T \cdot \left(c - A^T \cdot y\right) \\ &= x^T \cdot s. \end{aligned}$$

Wir erhalten als Konsequenz des Dualitäts-Theorems

Satz 7.11 Komplementäre Slackness
Sei x eine Lösung des primalen Problems (P) und (y, s) eine Lösung des dualen Problems (D). Dann sind äquivalent

(a) Die Lösung x ist optimal für (P) und (y, s) ist optimal für (D).

(b) $x^T \cdot s = 0$.

(c) Für alle i gilt $x_i \cdot s_i = 0$.

(d) Aus $s_i > 0$ folgt $x_i = 0$.

Für optimale Lösungen x und y des primalen, bzw. dualen Programms gilt also: Wenn eine duale Ungleichung „Slack“ hat (also wenn $s_i > 0$), ist die entsprechende primale Variable des Ausgangsproblems gleich 0.

Aufgabe 82
Sei A eine $m \times n$ Matrix. Zum primalen Programm $P = \min\{c^T \cdot x \mid A \cdot x = b,\, x \geq 0\}$ ist $D = \max\{y^T \cdot b \mid A^T \cdot y \leq c\}$ das zugehörige duale Programm. Sei x eine Lösung des primalen Problems (P) und y eine

Lösung des dualen Problems (D). Wir wissen (siehe Satz 3.14 im Skript), dass immer $c^T \cdot x \geq b^T \cdot y$ gilt. Außerdem gilt $c^T \cdot x = b^T \cdot y$ genau dann, wenn $(c^T - y^T \cdot A) \cdot x = 0$, d.h.

$$\sum_{i=1}^{m} a_{ij} y_i = c_j \qquad \text{für alle } j \text{ mit } x_j \neq 0.$$

Sei $\alpha \geq 1$ eine reelle Zahl. Statt der obigen komplementären Slackness-Bedignung nehmen wir die folgenden, schwächeren Ungleichungen an:

$$\frac{1}{\alpha} \cdot c_j \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j \qquad \text{für alle } j \text{ mit } x_j \neq 0.$$

Zeige, dass dann $c^T \cdot x \leq \alpha \cdot (b^T \cdot y)$ gilt. **Zeige** desweiteren, dass x damit eine α-approximative Lösung des primalen Problems (P) ist.

Aufgabe 83
Die linearen Programme (P) und (D) seien in der Form (P) : $\min\{c^T \cdot x \mid A \cdot x \geq b,\, x \geq 0\}$ und in der Form (D) : $\max\{b^T \cdot y \mid A^T \cdot y \leq c,\, y \geq 0\}$ gegeben.
Zeige, dass für eine Lösung x von (P) und eine Lösung y von (D) gilt:

$$\begin{aligned} & x \text{ ist eine optimale Lösung von } (P) \text{ und } y \text{ ist eine optimale Lösung von } (D) \\ \Leftrightarrow \quad & y^T(A \cdot x - b) = 0 \;\wedge\; x^T(c - A^T y) = 0 \end{aligned}$$

Hinweis: Führe Slack-Variablen ein.

Aufgabe 84
In den nächsten beiden Aufgaben betrachten wir das Flussproblem. Gegeben ist ein gerichteter Graph $G = (V, E)$ mit $V = \{1, 2, \ldots, n\}$ und eine Funktion $k : E \to \mathbb{N}$, die jeder Kante $(i, j) \in E$ ihre maximale *ganzzahlige*(!) Kapazität $k_{ij} \geq 0$ zuweist. Um die Analyse zu vereinfachen, erweitern wir die Kapazitätsfunktion k auf alle Paare (i, j) von Knoten: Wir setzen $k_{n1} := +\infty$ (damit erhält $(n, 1)$ unbegrenzte Kapazität) und setzen $k_{ij} := 0$ für alle anderen Paare $(i, j) \notin E$ (insbesondere sind alle $k_{ii} = 0$.) Ein Fluss im G ist ebenfalls eine Funktion f, die jedem Paar (i, j) der Knoten den Wert $f_{ij} \geq 0$ zuweist und die folgenden beiden Eigenschaften erfüllt:

1. $0 \leq f_{ij} \leq k_{ij}$ für alle i, j (entlang der Paare (i, j), die keine Kanten sind, fließt kein Fluss und der Fluss entlang einer Kante darf die Kapazität der Kante nicht übersteigen).
2. Für jeden Knoten i gilt $\sum_{i,j} f_{ji} = \sum_{i,j} f_{ij}$ und damit fordern wir Flusserhaltung: Der in i eingehende Fluss muss i auch wieder verlassen. Im Gegensatz zur Vorlesung fordern wir also Flusserhaltung auch für die Quelle 1 und die Senke n.

Das Ziel des Flussproblems ist die Konstruktion eines maximalen Flusses von der Quelle 1 zur Senke n. Da der Fluss entlang $(n, 1)$ wieder zur Quelle 1 zurückfließen kann, ist der Betrag des Flusses von 1 zu n genau f_{n1}. Das entsprechende Programm (mit Slack-Variablen t_{ij}) ist:

$$\begin{array}{llll} (P) & \text{maximiere} & f_{n1} & \\ & \text{so dass} & f_{ij} + t_{ij} = k_{ij} & \text{für alle } (i,j) \neq (n,1), \\ & & \sum_j (f_{ij} - f_{ji}) = 0 & \text{für alle } i \text{ und} \\ & & f_{ij} \geq 0, t_{ij} \geq 0 & \text{für alle } i, j. \end{array}$$

(a) **Zeige**, dass der maximale Wert eines Flusses in der neuen Formulierung übereinstimmt mit dem maximalen Wert eines Flusses in der Formulierung der Vorlesung.

(b) Das Programm $P = \max\{c^T \cdot x \mid A \cdot x = b,\, x \geq 0\}$ ist in der Standardform beschrieben.
Bestimme die Matrix A sowie die Vektoren c, x, b.

Das duale Programm zu $\max\{c^T \cdot x \mid A \cdot x = b,\, x \geq 0\}$ ist $\min\{b^T \cdot y \mid A^T \cdot y \geq c\}$. **Zeige**, dass das duale Programm die folgende Form hat:

$$\begin{array}{lll} (D) & \text{minimiere} & b^T \cdot y = \sum_{(i,j)\in E} k_{ij} y_{ij} \\ & \text{so dass} & y_n - y_1 \geq 1 \\ & & y_{ij} + y_i - y_j \geq 0 \\ & & y_{ij} \geq 0 \end{array}$$

(c) **Zeige**, dass das primale Programm (P) wie auch das duale Programm (D) nur *ganzzahlige* Ecken besitzt. *Hinweis*: Satz 3.7(b) im Skript.

Ein *Schnitt* in $G = (V, E)$ ist eine Untermenge $W \subseteq V$ der Knoten mit $1 \in W$ und $n \notin W$. Sei $e(W, \bar{W})$ die Menge aller Kanten $(i, j) \in E$ mit $i \in W$ und $j \notin W$, also ist $e(W, \bar{W})$ die Menge der *kreuzenden* Kanten. Die *Kapazität* des Schnitts W ist die Gesamtkapazität

$$K(W) := \sum_{(i,j)\in e(W,\bar{W})} k_{ij}$$

der kreuzenden Kanten.

(d) Sei W ein Schnitt in G. **Zeige**, dass es einen 0-1 wertigen Vektor y_W gibt, so dass y_W eine Lösung für das duale Programm (D) mit $b^T \cdot y_W = K(W)$ ist.

(e) **Zeige**, dass es für jede Ecke y des dualen Programms (D) einen Schnitt W mit $b^T \cdot y \geq K(W)$ gibt. *Hinweis*: Mit Aufgabe 7.2(c) können wir annehmen, dass y nur ganzzahlige Komponenten besitzt.

(f) **Stelle** eine Beziehung **her** zwischen dem Wert eines maximalen Flusses und dem Wert eines minimalen Schnitts.

Aufgabe 85

$G = (V, E)$ sei ein ungerichteter Graph mit zwei ausgezeichneten Knoten $s, t \in V$. Wir bezeichnen einen Pfad von s nach t als (s, t)-Pfad.

(a) **Zeige:** Es gilt:

$$\begin{array}{ll} & \max\{|P| : P \text{ ist eine Menge von paarweise kantendisjunkten } (s,t)\text{-Pfaden.}\} \\ = & \min\{|E^*| : E^* \subseteq E \wedge \text{ im Graph } (V, E \setminus E^*) \text{ existiert kein } (s,t)\text{-Pfad.}\} \end{array}$$

(b) Gilt die folgende Gleichheit

$$\begin{array}{ll} & \max\{|P| : P \text{ ist eine Menge von paarweise intern knotendisjunkten } (s,t)\text{-Pfaden.}\} \\ = & \min\{|V^*| : V^* \subseteq V \ \wedge\ \{s,t\} \cap V^* = \emptyset \ \wedge\ \text{nach Herausnahme der Knoten} \\ & \qquad \text{in } V^* \text{ existiert im Graph kein } (s,t)\text{-Pfad.}\}, \end{array}$$

falls s und t nicht benachbart sind? (Zwei (s, t)-Pfade heißen intern knotendisjunkt, wenn die Pfade bis auf die beiden Endpunkte s und t knotendisjunkt sind.)

Hinweis: Die Teile (d)-(f) der vorigen Aufgabe.

Aufgabe 86

Zeige, dass VERTEX COVER für bipartite Graphen effizient lösbar ist. Gilt diese Aussage auch für das gewichtete VERTEX COVER Problem?

7.3 Primal-duale Algorithmen

Anwendungen der linearen Programmierung sind zeitaufwändig. Wir betrachten deshalb die schnelleren primal-dual Algorithmen für die Lösung linearer Programme. Wir benutzen die komplementäre Slackness (Satz 7.11), die wir diesmal für die folgende Form des primalen Programms (P) und des dualen Programms (D)

$$\begin{aligned} (P) \quad & \text{minimiere } c^T \cdot x, \text{ so dass } A \cdot x \geq b \text{ und } x \geq 0 \\ (D) \quad & \text{maximiere } y^T \cdot b, \text{ so dass } y^T \cdot A \leq c \text{ und } y \geq 0 \end{aligned}$$

formulieren.

Satz 7.12 *x und y seien Lösungen für das primale Problem (P) und das duale Problem (D). Dann sind die beiden folgenden Aussagen äquivalent:*

(a) x und y sind optimal.

(b) Es gelten primale und duale komplementäre Slackness:

- *Primale komplementäre Slackness: Für alle i ist $x_i = 0$ oder die x_i zugeordnete duale Ungleichung ist exakt erfüllt.*
- *Duale komplementäre Slackness: Für alle j ist $y_j = 0$ oder die y_j zugeordnete primale Ungleichung ist exakt erfüllt.*

Ein primal-dualer Approximationsalgorithmus (für $c \geq 0$) geht wie folgt vor:

(1) Beginne mit $x = 0$ und der dualen Lösung $y = 0$. (Der Vektor $x = 0$ wird im Allgemeinen keine Lösung sein.) Die folgende Invariante gilt somit anfänglich:

   (a) y ist eine Lösung von (D).

   (b) Die primale komplementäre Slackness Bedingung gilt.

(2) Ein Schritt:

   - Erhöhe eine oder mehrere Komponenten von y bis eine erste duale Ungleichung exakt erfüllt wird; y muss weiterhin eine Lösung bleiben.
   - Erhöhe die zu den exakt erfüllten dualen Ungleichungen gehörenden x-Komponenten, um zusätzliche primale Ungleichungen zu erfüllen.

(3) Wiederhole Schritt (2) solange, bis alle primalen Ungleichungen erfüllt sind. Wenn $c^T x \leq \rho \cdot b^T y$, dann ist x eine ρ-approximative Lösung, denn $b^T y \leq \text{opt}_P$.

7.3.1 VERTEX COVER

Primales und duales Programm für VERTEX COVER haben die Form

$$\begin{array}{lll} (P)\ \text{minimiere} \sum_{u\in V} w_u \cdot x_u & \text{so dass } x_u + x_v \geq 1 \text{ für alle Kanten } \{u,v\} \in E & \\ & \text{und } x_u \geq 0 \text{ für alle Knoten } u \in V. & \\ (D)\ \text{maximiere} \quad \sum_{e\in E} y_e, & \text{so dass } \sum_{x,\, e=\{x,u\}\in E} y_e \leq w_u \text{ für alle } u \in V & (7.6) \\ & \text{und } y_e \geq 0 \text{ für alle } e \in E. & \end{array}$$

Wir möchten das Rezept eines primal-dualen Algorithmus umsetzen, indem wir eine duale Variable y_e solange erhögen bis eine duale Ungleichung exakt erfüllt wird. Wenn e die Endpunkte $\{u,v\}$ besitzt, dann taucht y_e nur in den Ungleichungen $\sum_{x,\, e=\{x,u\}\in E} y_e \leq w_u$ und $\sum_{x,\, e=\{x,v\}\in E} y_e \leq w_v$ auf. Wird Gleichheit für die erste Ungleichung erreicht, erhöhen wir x_u auf eins (fügen also u zur Knotenüberdeckung hinzu) und setzen ansonsten $x_v = 1$, bzw fügen v zur Knotenüberdeckung hinzu.

Algorithmus 7.13

(1) Die Eingabe besteht aus dem ungerichteten Graphen $G = (V, E)$ und den Knotengewichten w_u für $u \in V$.

(2) Beginne mit $V' = \emptyset$ und dem Vektor $y = 0$.

(3) Wiederhole solange, bis V' eine Knotenüberdeckung ist:

(3a) Sei $e = \{u, v\}$ eine nicht von V' überdeckte Kante. Erhöhe y_e solange, bis eine der Ungleichungen $\sum_{x,\, e=\{x,u\}\in E} y_e \leq w_u$ oder $\sum_{x,\, e=\{x,v\}\in E} y_e \leq w_v$ mit Gleichheit erfüllt wird.

Kommentar: Die duale Lösung bestimmt, welcher der beiden Endpunkte der nicht überdeckten Kante in die Knotenüberdeckung gelangt.

(3b) Wenn Gleichheit für die duale Ungleichung zu u erreicht wurde, dann setze $V' = V' \cup \{u\}$ und ansonsten setze $V' = V' \cup \{v\}$.

Satz 7.14 *Algorithmus 7.13 ist ein 2-Approximationsalgorithmus.*

Beweis: Sei V' die von Algorithmus 7.13 berechnete Knotenüberdeckung. Für jeden Knoten $u \in V'$ gilt $\sum_{x,\, e=\{x,u\}\in E} y_e = w_u$ und deshalb ist

$$\sum_{u\in V'} w_u = \sum_{u\in V'} \sum_{x,\, e=\{x,u\}\in E} y_e \ \leq \ \sum_{v\in V} \sum_{x,\, e=\{x,v\}\in E} y_e \ = \ 2 \cdot \sum_{e\in E} y_e$$

denn y_e wird für jeden Endpunkt von e einmal gezählt.

Aber $\sum_{e \in E} y_e$ ist der Wert der dualen Lösung und mit dem schwachen Dualitätssatz folgt

$$\sum_{e \in E} y_e \leq \sum_{v \in V} w_v x_v^* \leq \mathrm{vc}^*(G)$$

für die optimale primale Lösung x^*. Also gilt

$$\sum_{u \in V'} w_u \leq 2 \cdot \sum_{e \in E} y_e \leq 2 \cdot \mathrm{vc}^*(G)$$

und dies war zu zeigen. □

Bemerkung 7.2 Im Local Ratio Ansatz (Algorithmus 5.1 in Abschnitt 5.1) haben wir mit einer beliebigen Kante $e = \{u, v\}$ begonnen und haben den Endpunkt mit kleinstem Gewicht in unsere Überdeckung aufgenommen, das Gewicht des anderen Endpunkts haben wir um $\min\{w_u, w_v\}$ erniedrigt.
Was macht Algorithmus 7.13 anfänglich, wenn wir ebenfalls mit der Kante $\{u, v\}$ beginnen? Die Variable y_e wird auf $\min\{w_u, w_v\}$ gesetzt und der Endpunkt mit kleinerem Gewicht wird ebenfalls in die Überdeckung aufgenommen. Die einzige Auswirkung der Hochsetzung von y_e befindet sich in der Ungleichung für den nicht aufgenommen Endpunkt: Die Hochsetzung von y_e ist äquivalent zur Erniedrigung der rechten Seite um $\min\{w_u, w_v\}$.
Wenn wir den zweiten Schritt für beide Algorithmen verfolgen und davon ausgehen, dass dieselbe noch nicht überdeckte Kante behandelt wird, dann ist die Situation im Vergleich zum ersten Schritt identisch: Die beiden Algorithmen sind äquivalent!
Tatsächlich kann man zeigen, dass die Paradigmen „primal-duale Algorithmen“ und „Local Ratio Algorithmen“, nach geeigneter Definition der Paradigmen, äquivalent sind [BR05].

Aufgabe 87

(a) **Entwerfe** einen primal-dualen Approximationsalgorithmus für SET COVER. *Hinweis*: Betrachte das duale Programm 7.6.

(b) **Zeige**, dass der Approximationsfaktor durch f beschränkt ist, wenn jedes Element in höchstens f Mengen vorkommt.

(c) **Konstruiere** n Mengen $T_1, \ldots, T_n$ mit Gewichten $g_1, \ldots, g_n$, so dass der Approximationsfaktor mindestens $\Omega(n)$ ist.

7.4 SET COVER

Wir betrachten SET COVER, eine Verallgemeinerung von VERTEX COVER. Gegeben sind Mengen $T_1, \ldots, T_m$, wobei die Menge T_i das Gewicht $g_i \geq 0$ besitzt. Wir suchen die leichteste Überdeckung des Universums $\bigcup_{i=1}^m T_i$, d.h. eine Indexmenge I mit

$$\bigcup_{i \in I} T_i = \bigcup_{i=1}^{m} T_i, \tag{7.7}$$

so dass $\sum_{i \in I} g_i$ minimal unter allen Indexmengen mit Eigenschaft (7.7) ist.

Beispiel 7.5 SET COVER als Ursachenforschung.
Wir beschreiben eine Anwendung von SET COVER im Entwurf von Anti-Viren Software. Für die Virenerkennung möge eine große Menge von 3-Byte langen Sequenzen vorliegen, die nicht in „gutem Code“ wohl aber in vielen Viren vorkommen. Wir fassen die (bekannten) Viren als Elemente und jede verdächtige Sequenz als die Menge der Viren auf, in denen die Sequenz vorkommt.
Das Ziel ist die Erstellung einer Software, die aufgrund des Vorkommens verdächtiger Sequenzen lernt, einen vorher nicht bekannten Virus hochwahrscheinlich korrekt vorauszusagen. Das wesentliche Problem in diesem Lernprozess ist die große Anzahl verdächtiger Viren: Die enorm große Anzahl an Freiheitsgraden, also die ernorm große Zahl möglicher Ursachen, macht ein erfolgreiches Lernen unmöglich.
Was wurde gemacht? Eine Überdeckung aller Viren mit möglichst wenigen verdächtigen Sequenzen wurde mit Hilfe von SET COVER bestimmt. (Tatsächlich wurde auch sichergestellt, dass jeder Virus genügend häufig überdeckt wurde.) Das abschließende Lernproblem war machbar, da nur wenige (einige Hundert) verdächtige Sequenzen für die Überdeckung ausreichten.

Im Folgenden nehmen wir an, dass das Universum $\bigcup_{i=1}^{m} T_i$ aus genau n Elementen besteht. Nach Umbennenung der Elemente können wir sogar fordern, dass das Universum mit $\{1, \ldots, n\}$ übereinstimmt. Den Wert der optimalen Lösung bezeichnen wir mit `opt`.

Algorithmus 7.15 **Approximationsalgorithmus für SET COVER**

(1) Gegeben sind die Mengen $T_1, \ldots, T_m \subseteq \{1, \ldots, n\}$ und ihre Gewichte $g_1, \ldots, g_m$.

(2) Setze $U_0 = \emptyset$, $I = \emptyset$ und $i = 0$.

(3) Wiederhole, solange wie $U_i \neq \{1, \ldots, n\}$

(3a) $\mid T_k \setminus U_i \mid$ ist die Anzahl noch nicht überdeckter Elemente von T_k. Bestimme eine Menge T_K mit bestem „Preis-Leistungsverhältnis“, d.h. mit

$$\frac{g_K}{|T_K \setminus U_i|} = \min_r \left\{ \frac{g_r}{|T_r \setminus U_i|} \right\}.$$

(3b) Setze $U_{i+1} = U_i \cup T_K$, $I = I \cup \{K\}$, $i = i + 1$.

SET COVER besitzt das primale Programm

$$\text{opt} = \text{minimiere} \sum_{k=1}^{m} g_k \cdot x_k, \text{ so dass} \quad \sum_{k,\, j \in T_k} x_k \geq 1 \ \text{ für alle } j \in \{1, \ldots, n\} \quad (7.8)$$
$$\text{und} \ \ x_k \geq 0 \ \text{ für } k = 1, \ldots m.$$

Das duale Programm hat die Form:

$$\text{opt} = \text{maximiere} \sum_{j=1}^{n} y_j, \quad \text{so dass} \quad \begin{aligned} \sum_{j \in T_k} y_j &\leq g_k \quad \text{für } k = 1, \ldots, m, \\ y_j &\geq 0 \quad \text{für } j \in \{1, \ldots, n\}. \end{aligned}$$

Das primale Programm heißt ein „Überdeckungsprogramm“, während das duale Programm ein „Packungsprogramm“ ist. Die duale Sichtweise zu einer billigsten Überdeckung mit gewichteten Mengen ist also eine schwerste Bepackung der Mengen mit Elementen, wobei jede Menge eine Kapazitätsschranke besitzt, die mit ihrem Gewicht übereinstimmt.
Offensichtlich berechnet Algorithmus 7.15 eine Lösung für das primale Programm. Tatsächlich wird sogar simultan eine „fast-legale“ Lösung y des dualen Programms berechnet. Wie? Angenommen, das Element j wird erstmalig in der Iteration $i(j)$ und dann durch die Menge $T_{k(j)}$ überdeckt. Wir weisen dann dem Element j die Überdeckungskosten

$$\text{preis}(j) = \frac{g_{k(j)}}{|T_{k(j)} \setminus U_{i(j)}|}$$

zu.

Zwischenbehauptung: Sei $H_k = \sum_{r=1}^{k} \frac{1}{r}$ die $k$te harmonische Zahl. Dann ist der Vektor y mit $y_j = \frac{\text{preis}(j)}{H_n}$ eine Lösung des dualen Problems.

(Beweis folgt.) Sei $I \subseteq \{1, \ldots, m\}$ die von Algorithmus 7.15 bestimmte Überdeckung. Dann ist

$$\sum_{j=1}^{n} \text{preis}(j) = \sum_{j=1}^{n} \frac{g_{k(j)}}{|T_{k(j)} \setminus U_{i(j)}|} = \sum_{i \in I} \sum_{j=1, k(j)=i}^{n} \frac{g_{k(j)}}{|T_{k(j)} \setminus U_{i(j)}|} = \sum_{i \in I} g_i.$$

Aus der Zwischenbehauptung folgt also mit dem schwachen Dualitätssatz

$$\sum_{j=1}^{n} y_j = \frac{1}{H_n} \cdot \sum_{j=1}^{n} \text{preis}(j) \leq \texttt{opt}$$

und wir erhalten

$$\sum_{i \in I}^{n} g_i = \sum_{j=1}^{n} \text{preis}(j) \leq H_n \cdot \texttt{opt}. \tag{7.9}$$

Beweis der Zwischenbehauptung: Wir konzentrieren uns auf die Menge T_i und numerieren die $r = |T_i|$ Elemente von T_i nach dem Zeitpunkt ihrer Überdeckung durch Algorithmus 7.15; demgemäß sei $T_i = \{e_1, \ldots, e_r\}$. Wenn Element e_j überdeckt wird, dann sind vor der Überdeckung von e_j mindestens $r - (j-1) = r - j + 1$ Elemente von T_i noch zu überdecken.
Algorithmus 7.15 wählt stets das beste Preis-Leistungsverhältnis. Wenn also Element j zum Zeitpunkt $k(j)$ überdeckt wird, gilt

$$\text{preis}(j) = \frac{g_{k(j)}}{|T_{k(j)} \setminus U_{i(j)}|} \leq \frac{g_i}{r - j + 1}.$$

Also folgt

$$\sum_{j \in T_i} \text{preis}(j) \leq g_i \cdot \sum_{j=1}^{r} \frac{1}{r - j + 1} \leq g_i \cdot H_n$$

und der Vektor y erfüllt die Nebenbedingung $\sum_{j \in T_i} y_j \leq g_i$ für ein beliebiges i. Also ist y eine Lösung des dualen Programms. □

Satz 7.16 *Algorithmus 7.15 ist ein effizienter Approximationsalgorithmus für SET COVER mit Approximationsfaktor* $\sum_{i=1}^{n} \frac{1}{i} \leq 1 + \ln(n)$.

Wir zeigen später, dass der Greedy-Algorithmus 7.15 sogar den asymptotisch bestmöglichen Approximationsfaktor erreicht! Selbst das allgemeine Überdeckungsproblem

$$\text{opt} = \text{minimiere } \sum_{i=1}^{m} g_i \cdot x_i, \text{ so dass } \; A \cdot x \geq b \; \text{ und } x \text{ ist ganzzahlig mit } x \geq 0$$

kann mit einer etwas komplizierteren Anwendung des Greedy-Algorithmus approximativ gelöst werden. (Es muss $A, g, b \geq 0$ gefordert werden und man erhält den Approximationsfaktor $O(\ln(n))$.)

Aufgabe 88
Zeige, dass der Approximationsfaktor des Algorithmus 7.15 sogar durch

$$\sum_{i=1}^{\mathbf{r}} \frac{1}{i} = H_{\mathbf{r}}$$

beschränkt ist, wobei r die Mächtigkeit der größten Menge T_i ist.

Aufgabe 89
Wir betrachten den Spezialfall VERTEX COVER von SET COVER. Sei also ein Graph $G = (V, E)$ vorgegeben. Wir wählen als Universum E und ordnen jedem Knoten die Menge der zu ihm inzidenten Kanten zu.

$$T_v = \{e \in E | v \in e\}$$

Zeige durch die Angabe einer entsprechenden Graphenfamilie, dass Algorithmus 4.7 sogar für den Spezialfall VERTEX COVER einen Verlustfaktor $\Omega(\log(n))$ besitzt, wobei n die Zahl der Knoten ist.

Aufgabe 90
Ein ungerichteter Graph $G = (V, E)$ heißt dreiecksfrei, wenn er keine 3-Clique enthält, also keine Menge von drei Knoten besitzt, die alle miteinander verbunden sind.
Wir wollen einen beliebigen ungerichteten Graphen durch das Entfernen einer möglichst kleinen Kantenmenge dreiecksfrei machen. **Gib** einen Algorithmus **an**, der für dieses Problem $\ln(d-1)+1$-approximativ ist, wenn $d \geq 3$ der maximale Grad eines Knoten in G ist.

Wir betrachten jetzt SET COVER unter der Annahme, dass jedes Element in höchstens f Mengen vorkommt. In diesem Fall erhalten wir eine besonders einfache Lösung, wenn wir zuerst eine optimale Lösung x des primalen Programms (7.8) berechnen und sodann alle Mengen T_k mit $x_k \geq \frac{1}{f}$ in unsere Überdeckung aufnehmen.
Wir erhalten tatsächlich eine Überdeckung, denn für jedes Element j muss es eine Menge T_k mit $j \in T_k$ und $x_k \geq \frac{1}{f}$ geben, da natürlich auch j in höchstens f Mengen vorkommt. Durch die Rundung wird der Wert der optimalen Lösung um den Faktor f vergrössert und wir erhalten:

Satz 7.17 *Wenn jedes Element in höchstens f Mengen vorkommt, dann führt der obige Rundungsprozess zu einem f-approximativen Algorithmus.*

Aufgabe 91
Wir betrachten die folgende Version von SET COVER. Gegeben sind: das Universum $X = \{1, \ldots, n\}$

und nicht-negative Gewichte $w_1, \ldots, w_n \geq 0$ der Elemente von X, ein Mengensystem $\mathcal{S}$ von Teilmengen von X sowie eine natürliche Zahl $k \geq 1$. Für $\mathcal{S}' \subseteq \mathcal{S}$ sei $w(\mathcal{S}')$ das Gesamtgewicht der von Mengen in $\mathcal{S}'$ überdeckten Elemente. (Beachte, dass diesmal nicht die Mengen, sondern die *Elemente* gewichtet sind.) Gesucht ist ein Mengensystem $\mathcal{S}' \subseteq \mathcal{S}$, so dass $\mathcal{S}'$ höchstens k Mengen enthält und $w(\mathcal{S}')$ *maximal* unter allen solchen Mengensystemen $\mathcal{S}' \subseteq \mathcal{S}$ der Größe $\leq k$ ist.
Betrachte den folgenden Greedy-Algorithmus, der in jedem Schritt die "beste" Teilmenge nimmt.

$\mathcal{S}' := \emptyset$
$U := \emptyset$
for $i = 1, \ldots, k$ **do**
  Bestimme eine Menge $S_i \in \mathcal{S}$, so dass das Gesamtgewicht $w(S_i \setminus U)$ der neu überdeckten Elemente maximal ist
  $\mathcal{S}' := \mathcal{S}' \cup \{S_i\}$
  $U := U \cup S_i$
end for

Seien $S_1, \ldots, S_k$ die vom Algorithmus gewählten Mengen und $S'_i = S_i \setminus (S_1 \cup \cdots \cup S_{i-1})$ die Menge der im i-ten Schritt neu überdeckten Elemente. Dann ist

$$W(i) = \sum_{j=1}^{i} w(S'_j)$$

das Gesamtgewicht der von den ersten i Mengen $S_1, \ldots, S_i$ überdeckten Elemente; wir setzen $W(0) = 0$. Sei weiterhin $w(OPT)$ das maximale Gesamtgewicht, das für k Mengen aus $\mathcal{S}$ erreichbar ist.

(a) **Zeige**, dass für jedes $i = 1, \ldots, k$

$$w(S'_i) \geq \frac{w(OPT) - W(i-1)}{k}.$$

(b) **Zeige**, dass für jedes $i = 1, \ldots, k$

$$W(i) \geq \alpha_i \cdot w(OPT)$$

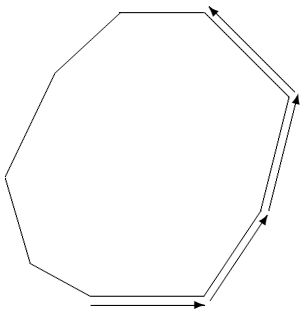
mit $\alpha_i := 1 - \left(1 - \frac{1}{k}\right)^i$ gilt. *Hinweis*: Teil (a).

Da $\alpha_k = 1 - \left(1 - \frac{1}{k}\right)^k > 1 - \frac{1}{e}$, ist der Greedy-Algorithmus damit $1/\alpha_k = e/(e-1) \approx 1.58$ approximativ.

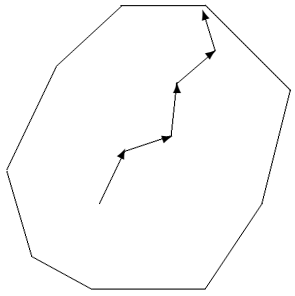
Kapitel 8

Die Interior-Point-Methode von Ye

Der Polynomialzeit-Algorithmus von M. Karmarkar [Karm] aus dem Jahr 1984 hat zur Entwicklung weiterer verbesserter Interior-Point-Methoden für die lineare Programmierung geführt. Interior Point Methoden sind Simplex und seinen Varianten vor Allem dann überlegen, wenn große Programme gelöst werden müssen und gute approximative Lösungen ausreichen. Während der Simplex-Algorithmus sich von Ecke zu Ecke des Lösungspolyeders bewegt, marschieren Interior-Point-Methoden im Inneren des Lösungspolyeders und zwar in Richtung des Optimums.



Simplex-Methode



Interior-Point-Methode

Wir besprechen in diesem Abschnitt einen Algorithmus von Y. Ye [Ye1] mit Modifikationen von R.M. Freund [Fre]. Das Verfahren verbessert den Polynomialzeit-Algorithmus von M. Karmarkar [Karm] und basiert auf einer *Potential-Reduktions Methode*. Für eine Übersicht über Interior-Point-Methoden in der linearen Programmierung verweisen wir auf [Van, Ye1].
Wir verzichten bei der Vorstellung von Yes Algorithmus auf vollständige Beweise und konzentrieren uns stattdessen auf die Motivation. Der Interior-Point-Algorithmus von Y. Ye versucht, das primale Ausgangsproblem (P) und das duale Problem (D) simultan zu lösen. Das primale Programm sei

$$(P) \quad \text{minimiere } c^T \cdot x, \quad \text{so dass} \quad \begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned}$$

mit einer $m \times n$-Matrix A, einem Spaltenvektor b mit m Einträgen und einem Spaltenvektor c mit n Einträgen. Das duale Programm mit den Slackvariablen s ist

$$(D) \quad \text{maximiere } y^T \cdot b, \quad \text{so dass} \quad \begin{aligned} y^T \cdot A + s &= c^T \\ s &\geq 0. \end{aligned}$$

Die Variablen x und s haben n Komponenten, während y m Komponenten hat. Wir wollen im Weiteren triviale Fälle ausschließen. Dazu setzen wir voraus:

- Es gilt $\text{Rang}(A) = m$. Wenn $\text{Rang}(A) < m$ ist, sind Restriktionen überflüssig oder es gibt überhaupt keine Lösung. Wir entfernen gegebenenfalls überflüssige Zeilen aus der Matrix A und die entsprechenden Einträge im Vektor b.

- Der Lösungsraum besitzt innere Punkte.

Der allgemeine Ansatz des Interior-Point-Verfahrens von Y. Ye benutzt die Potentialfunktion

$$G(x, s) = q \cdot \ln\left(x^T \cdot s\right) - \sum_{j=1}^{n} \ln\left(x_j \cdot s_j\right) \tag{8.1}$$

für eine später zu bestimmende reelle Zahl q. Wir versuchen in jeder Iteration, das alte Lösungspaar $(x_{\text{alt}}, s_{\text{alt}})$ durch ein neues Lösungspaar $(x_{\text{neu}}, s_{\text{neu}})$ mit

$$G\left(x_{\text{neu}}, s_{\text{neu}}\right) < G\left(x_{\text{alt}}, s_{\text{alt}}\right)$$

zu ersetzen. Die zugrundeliegende Motivation ist, einerseits die Dualitätslücke zu verringern, also

$$x_{\text{neu}}^T \cdot s_{\text{neu}} < x_{\text{alt}}^T \cdot s_{\text{alt}}$$

zu erreichen und andererseits einen relativ großen Abstand vom Rand des Polyeders, genauer von den Hyperebenen $x_i = 0$ und $s_i = 0$ zu wahren. Die letzte Bedingung wird erreicht, wenn die „Barriere-Funktion“

$$\sum_{j=1}^{n} \ln\left(x_j^{\text{neu}} \cdot s_j^{\text{neu}}\right) = \ln\left(\prod_{j=1}^{n} x_j^{\text{neu}} \cdot s_j^{\text{neu}}\right)$$

möglichst groß ist. Warum ist ein möglichst großer Abstand vom Rand zu bewahren? Wir möchten die Gradientenmethode anwenden, um die Dualitätslücke zu minimieren.

Algorithmus 8.1 **Der Interior-Point-Algorithmus mit Potential-Reduktion: Die Grobstruktur**

(1) Die Eingabe besteht aus der $m \times n$-Matrix A mit $m \leq n$ und $\text{Rang}(A) = m$, dem Vektor $b \in \mathbb{R}^m$ und der Zielfunktion $c^T \cdot x$.

(2) Wir beginnen mit einer Lösung x des primalen Problems und einer Lösung (y, s) des dualen Problems.

(3) Solange die Dualitätslücke zu groß ist, wiederhole

Berechne eine Lösung x_{neu} des primalen Problems und $(y_{\text{neu}}, s_{\text{neu}})$ des dualen Problems, so dass das Potential gesenkt wird. Mit anderen Worten, für die Potentialfunktion G aus 8.1 gelte

$$G(x_{\text{neu}}, s_{\text{neu}}) < G(x_{\text{alt}}, s_{\text{alt}}).$$

Aktualisiere x und s.

8.1 Bestimmung einer Anfangslösung

Bereits eine Implementierung von Schritt (2) ist nicht-trivial, da man zeigen kann, dass das Auffinden einer Lösung bereits so schwierig ist wie das Lösen des linearen Programms. Zuerst ändern wir deshalb das primale und das duale Problem so, dass

(a) Lösungen $(x, s) > 0$ einfach konstruierbar sind und

(b) optimale Lösungen für die neuen Probleme unmittelbar auf optimale Lösungen der Ausgangsprobleme führen.

Der erste Ansatz mit einer Konstanten L und einem beliebigen Vektor $x_0 > 0$ definiert das Minimierungsproblem

$$\begin{aligned} \text{minimiere } c^T \cdot x + L \cdot x_{n+1}, \text{ so dass} \quad Ax + x_{n+1} \cdot (b - Ax_0) &= b \\ x, x_{n+1} &\geq 0. \end{aligned}$$

Wir haben erreicht, dass die Lösung $(x, x_{n+1}) := (x_0, 1)$ strikt positiv ist. Da wir simultan Lösungen für das primale und duale Problem suchen, verfolgen wir nachstehenden Ansatz mit Konstanten L_1, L_2, L_3 und Vektoren $x_0, s_0 > 0$:

$$(P_{\text{neu}}) \qquad \begin{aligned} \text{minimiere } c^T \cdot x + L_1 \cdot x_{n+1}, \text{ so dass} \quad Ax + x_{n+1} \cdot (b - Ax_0) &= b \\ (c - s_0)^T \cdot x + L_3 \cdot x_{n+2} &= L_2 \\ x, x_{n+1}, x_{n+2} &\geq 0 \end{aligned}$$

Das duale Problem lautet:

$$(D_{\text{neu}}) \qquad \begin{aligned} \text{maximiere } b^T \cdot y + L_2 \cdot y_{m+1}, \quad \text{so dass} \quad A^T \cdot y + y_{m+1} \cdot (c - s_0)^T + s &= c \\ (b - Ax_0) \cdot y + s_{n+1} &= L_1 \\ L_3 \cdot y_{m+1} + s_{n+2} &= 0 \\ s, s_{n+1}, s_{n+2} &\geq 0 \end{aligned}$$

Der Vektor (x, x_{n+1}, x_{n+2}) mit $x = x_0$, $x_{n+1} = 1$ und $x_{n+2} > 0$ bildet eine strikt positive Lösung des neuen primalen Problems, während der Vektor $(y, y_{m+1}, s, s_{n+1}, s_{n+2})$ mit $y =$

$(0,\ldots,0)$, $y_{m+1}=1$, $s=s_0$ und $s_{n+1},s_{n+2}>0$ eine Lösung des dualen Problems mit strikt positiven Slackvariablen ist, solange s_0 strikt positiv gewählt wird.
Wir haben noch zu untersuchen, ob optimale Lösungen des transformierten Problems auf optimale Lösungen des Ausgangsproblems führen. Angenommen, wir haben optimale Lösungen

$$\left(x^{\text{opt}}, x_{n+1}^{\text{opt}}, x_{n+2}^{\text{opt}}\right) \text{ und } \left(y^{\text{opt}}, y_{m+1}^{\text{opt}}, s^{\text{opt}}, s_{n+1}^{\text{opt}}, s_{n+2}^{\text{opt}}\right)$$

erhalten. Dann können wir $s_{n+1}^{\text{opt}}>0$ durch geeignete Wahl der Konstanten L_1 erzwingen. Mit der komplementären Slackness folgt $x_{n+1}^{\text{opt}}=0$ und somit bedingt die erste Restriktion von (P_{neu}), dass

$$Ax^{\text{opt}}=b.$$

Wir können durch geeignete Wahl der Konstanten L_2 und L_3 erreichen, dass $x_{n+2}^{\text{opt}}>0$ ist. Aus der komplementären Slackness folgt $s_{n+2}^{\text{opt}}=0$ und damit $y_{m+1}^{\text{opt}}=0$. Aus der ersten Restriktion von (D_{neu}) folgt deshalb

$$A^T\cdot y^{\text{opt}}+s^{\text{opt}}=c$$

und wir haben eine optimale Lösung des (ursprünglichen) dualen Problems gefunden. Insgesamt erhalten wir aus optimalen Lösungen für die neuen Probleme unmittelbar optimale Lösungen für die alten Probleme. Wir können also Schritt (2) implementieren.

8.2 Die Iteration

Wir kommen zum wesentlichen Problem. Wie können wir die gegenwärtigen Lösungen $x_{\text{alt}}>0$ und $(y_{\text{alt}},s_{\text{alt}})$ (mit $s_{\text{alt}}>0$) verbessern? Wir gehen wie in Algorithmus 8.1 vor und erläutern im weiteren die einzelnen Phasen.

Algorithmus 8.2 **Grob-Implementierung von Schritt (3)**

(1) Skalierung: Wir verändern die Polyeder geringfügig, um stets denselben Abstand vom Rand zu gewährleisten.

(2) Minimierung: Wende Gradientenabstieg an.

(3) Reskalierung: Gehe zurück zum Ausgangspolyeder, also zur Situation vor Schritt (1).

Betrachten wir zunächst den Skalierungs- und Reskalierungsschritt. Zur gegenwärtigen Lösung x_{alt} wählen wir die folgende Skalierung:

$$\text{skalierung}(z_1,\ldots,z_n):=\left(\ \tfrac{z_1}{x_1^{\text{alt}}},\ \ \tfrac{z_2}{x_2^{\text{alt}}},\ \ \ldots,\ \ \tfrac{z_n}{x_n^{\text{alt}}}\ \right). \tag{8.2}$$

Insbesondere gilt skalierung$(x_{\text{alt}}) = (1, \ldots, 1)$ und dieser Punkt ist jetzt weit vom Rand $x = 0$ entfernt. Für die Matrix

$$D := \begin{pmatrix} \frac{1}{x_1^{\text{alt}}} & 0 & \cdots & 0 \\ 0 & \frac{1}{x_2^{\text{alt}}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \frac{1}{x_n^{\text{alt}}} \end{pmatrix}$$

gilt skalierung$(z) = D \cdot z$. Die inverse Matrix zu D ist:

$$D^{-1} = \begin{pmatrix} x_1^{\text{alt}} & 0 & \cdots & 0 \\ 0 & x_2^{\text{alt}} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & x_n^{\text{alt}}. \end{pmatrix}$$

Wenn wir mit den transformierten Variablen arbeiten, aber das ursprüngliche Problem lösen wollen, ändert sich die Aufgabe. Dazu beachte, dass

$$c^T \cdot z = c^T \cdot (D^{-1} D z) = (D^{-1} \cdot c)^T \cdot Dz \text{ und}$$

$$Az = A \cdot D^{-1} \cdot D \cdot z = (A \cdot D^{-1})\, Dz = b$$

Wir haben das primale Problem

$$(P_{\text{skal}}) \quad \text{minimiere } (D^{-1} \cdot c)^T \cdot x, \quad \text{so dass} \quad \begin{aligned} (A \cdot D^{-1}) \cdot x &= b \\ x &\geq 0 \end{aligned}$$

zu lösen. Wir multiplizieren die Gleichungen des dualen Problems mit D^{-1} und erhalten

$$\text{maximiere } b^T \cdot y, \quad \text{so dass} \quad \begin{aligned} (A \cdot D^{-1})^T \cdot y + D^{-1} \cdot s &= D^{-1} \cdot c \\ s &\geq 0 \end{aligned}$$

Dies ist wegen $s \geq 0 \Leftrightarrow D^{-1} \cdot s \geq 0$ äquivalent zum dualen Problem von (P_{skal}), nämlich

$$(D_{\text{skal}}) \quad \text{maximiere } b^T \cdot y, \quad \text{so dass} \quad \begin{aligned} (A \cdot D^{-1})^T \cdot y + s &= D^{-1} \cdot c \\ s &\geq 0 \end{aligned}$$

Die Skalierung hat also die folgenden Eigenschaften:

- Die primale Lösung x_{alt} wird auf $x_{\text{skal}} := (1, \ldots, 1)$ transformiert und liegt im Inneren des Lösungspolyeders.

- Die duale Lösung $(y_{\text{alt}}, s_{\text{alt}})$ wird auf $(y_{\text{skal}}, s_{\text{skal}}) := (y_{\text{alt}}, D^{-1} \cdot s_{\text{alt}})$ transformiert. Insbesondere ist $s_i^{\text{skal}} = s_i^{\text{alt}} \cdot x_i^{\text{alt}}$.

- Die Dualitätslücke ist unverändert, da

$$x_{\text{skal}}^T \cdot s_{\text{skal}} = \sum_{i=1}^{n} x_i^{\text{skal}} \cdot s_i^{\text{skal}} = \sum_{i=1}^{n} \frac{x_i^{\text{alt}}}{x_i^{\text{alt}}} \cdot \left(x_i^{\text{alt}} \cdot s_i^{\text{alt}} \right) = \sum_{i=1}^{n} x_i^{\text{alt}} \cdot s_i^{\text{alt}} = x_{\text{alt}}^T \cdot s_{\text{alt}}.$$

- Der Wert der Potentialfunktion ist unverändert, denn

$$\begin{aligned} G\left(x_{\text{skal}}, s_{\text{skal}}\right) &= q \cdot \ln\left(x_{\text{skal}}^T \cdot s_{\text{skal}}\right) - \sum_{i=1}^{n} \ln\left(x_i^{\text{skal}} \cdot s_i^{\text{skal}}\right) \\ &= q \cdot \ln\left(x_{\text{alt}}^T \cdot s_{\text{alt}}\right) - \sum_{i=1}^{n} \ln\left(\frac{x_i^{\text{alt}}}{x_i^{\text{alt}}} \cdot x_i^{\text{alt}} \cdot s_i^{\text{alt}}\right) \\ &= G\left(x_{\text{alt}}, s_{\text{alt}}\right). \end{aligned}$$

Wie sieht die Reskalierung in Schritt (3) aus? Wenn Schritt (2) x_{skal} durch x_{skalneu} und $(y_{\text{skal}}, s_{\text{skal}})$ durch $(y_{\text{skalneu}}, s_{\text{skalneu}})$ ersetzt, ist die Skalierung zurückzunehmen, indem wir

- $x_{\text{neu}} := D^{-1} \cdot x_{\text{skalneu}}$ und

- $(y_{\text{neu}}, s_{\text{neu}}) := (y_{\text{skalneu}}, D \cdot s_{\text{skalneu}})$

berechnen. Man überlege sich, dass durch die Reskalierung das Potential und die Dualitätslücke unverändert bleiben.
Wir müssen noch das zentrale Problem, nämlich die Implementierung von Schritt (2) in Algorithmus 8.2 lösen. Wir kürzen das primale und das duale Problem nach Skalierung durch

$$(P_{\text{skal}}) \quad \text{minimiere } c_{\text{skal}}^T \cdot x, \quad \text{so dass} \quad \begin{aligned} A_{\text{skal}} \cdot x &= b \\ x &\geq 0 \end{aligned}$$

und

$$(D_{\text{skal}}) \quad \text{maximiere } b^T \cdot y, \quad \text{so dass} \quad \begin{aligned} A_{\text{skal}}^T \cdot y + s &= c_{\text{skal}} \\ s &\geq 0 \end{aligned}$$

ab. Unser Ziel ist eine Neuberechnung entweder

- der gegenwärtigen Lösung x_{skal} von (P_{skal}) (primaler Schritt) oder

- der gegenwärtigen Lösung $(y_{\text{skal}}, s_{\text{skal}})$ von (D_{skal}) (dualer Schritt),

so dass der Wert der Potentialfunktion G verringert wird.

8.2.1 Gradientenabstieg

An dieser Stelle nutzen wir aus, dass die Lösung im Inneren des Lösungspolyeders liegt: Wir können die Gradientenmethode einsetzen. Wir berechnen deshalb als nächstes den Gradienten unserer Potentialfunktion G. Mit der Kettenregel und wegen $\ln'(x) = \frac{1}{x}$ erhalten wir für $i = (1, \ldots, n)$

$$\begin{aligned}\frac{\partial G(x,s)}{\partial x_i} &= q \cdot \frac{1}{x^T \cdot s} \cdot \sum_{j=1}^{n} \frac{\partial (x_j \cdot s_j)}{\partial x_i} - \sum_{j=1}^{n} \frac{1}{x_j \cdot s_j} \cdot \frac{\partial (x_j \cdot s_j)}{\partial x_i} \\ &= \frac{q}{x^T \cdot s} \cdot s_i - \frac{s_i}{x_i \cdot s_i} \\ &= \frac{q}{x^T \cdot s} \cdot s_i - \frac{1}{x_i}\end{aligned}$$

Wir setzen $e := (1, \ldots, 1)$ zur Vereinfachung der Notation. Der Gradient der Ableitungen nach $x = (x_1, \ldots, x_n)$ an der Stelle $(x, s) = (x_{\text{skal}}, s_{\text{skal}})$ ist wegen $x_{\text{skal}} = e$ daher

$$g := \nabla_x G\left(x_{\text{skal}}, s_{\text{skal}}\right) = \left(\frac{q}{e^T \cdot s_{\text{skal}}} \cdot s^i_{\text{skal}} - 1 \right)_{i=1,\ldots,N} = \frac{q}{e^T \cdot s_{\text{skal}}} \cdot s_{\text{skal}} - e. \tag{8.3}$$

Die Iteration

$$x_{\text{skal}} \mapsto x_{\text{skal}} - \eta \cdot g$$

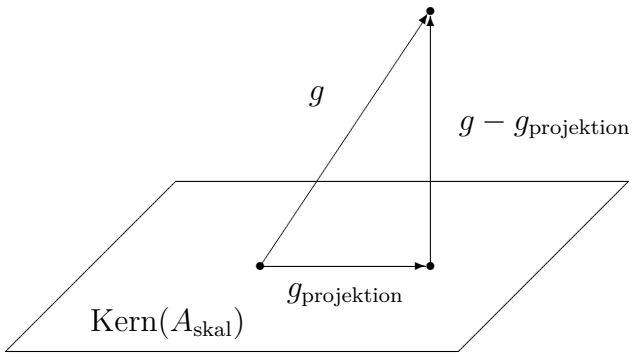
bietet sich zwar an, wird aber im Allgemeinen keine Lösung zu (P_{skal}) finden, da die Restriktion $A_{\text{skal}} \cdot x_{\text{skalneu}} = b_{\text{skal}}$ verletzt sein kann. Unsere Iteration muss daher von der Form

$$x_{\text{skal}} \mapsto x_{\text{skal}} - \eta \cdot u$$

für ein u mit $A_{\text{skal}} \cdot u = 0$ sein, und wir müssen den negativen Gradienten auf den Vektorraum

$$\text{Kern}(A_{\text{skal}}) = \{u \mid A_{\text{skal}} \cdot u = 0 \}$$

projizieren.



Wir beachten zur Berechnung der Projektion $g_{\text{projektion}}$ von g, dass der Orthogonalraum von Kern(A_{skal}) von den Zeilenvektoren $A_1^{\text{skal}}, A_2^{\text{skal}}, \ldots, A_m^{\text{skal}}$ aufgespannt wird, denn für $u \in \text{Kern}(A_{\text{skal}})$ ist $A_{\text{skal}} \cdot u = 0$ und u steht senkrecht auf den Zeilenvektoren der Matrix A_{skal}.
Kern$(A_{\text{skal}})^{\perp}$ wird also von den Zeilenvektoren der Matrix A_{skal} aufgespannt, und jeder Vektor $u \in \text{Kern}(A_{\text{skal}})^{\perp}$ ist eine Linearkombination der Spaltenvektoren von A_{skal}^T. Da $g - g_{\text{projektion}} \neq 0$ orthogonal zu Kern(A_{skal}) ist, gibt es einen Vektor w mit

$$A_{\text{skal}}^T \cdot w = g - g_{\text{projektion}}. \tag{8.4}$$

Andererseits ist $g_{\text{projektion}} \in \text{Kern}(A_{\text{skal}})$ und somit

$$A_{\text{skal}} \cdot g_{\text{projektion}} = 0.$$

Insbesondere gilt daher

$$A_{\text{skal}} \cdot A_{\text{skal}}^T \cdot w = A_{\text{skal}} \cdot (g - g_{\text{projektion}}) = A_{\text{skal}} \cdot g. \tag{8.5}$$

Um den projizierten Gradienten $g_{\text{projektion}}$ zu bestimmen, lösen wir die Gleichungssysteme (8.4) und (8.5). Aus (8.5) folgt

$$w = \left(A_{\text{skal}} \cdot A_{\text{skal}}^T\right)^{-1} \cdot A_{\text{skal}} \cdot g$$

und wir erhalten deshalb aus (8.4)

$$g_{\text{projektion}} = g - A_{\text{skal}}^T \cdot \left(A_{\text{skal}} \cdot A_{\text{skal}}^T\right)^{-1} \cdot A_{\text{skal}} \cdot g. \tag{8.6}$$

8.2.2 Der primale Schritt

Als neue Lösung für den primalen Schritt setzen wir:

$x_{\text{skalneu}} := x_{\text{skal}} - \frac{1}{4 \cdot ||g_{\text{projektion}}||} \cdot g_{\text{projektion}}$

$y_{\text{skalneu}} := y_{\text{skal}}$ und $s_{\text{skalneu}} := s_{\text{skal}}$.

Offenbar bleibt x_{skalneu} eine innere Lösung, denn

$$x_{\text{skalneu},i} = 1 - \frac{g_{\text{projektion},i}}{4 \cdot ||g_{\text{projektion}}||} \geq \frac{3}{4}.$$

Eine entscheidende Verringerung des Potentials ist nicht zu erwarten, wenn der negative Gradient fast orthogonal zu Kern(A_{skal}) ist. Im gegenteiligen Fall erreichen wir jedoch in einem primalen Schritt eine wesentliche Verringerung des Werts der Potentialfunktion.

Lemma 8.3 Primaler Schritt
Sei $||g_{\text{projektion}}|| > 0,4$. *Für den primalen Schritt gilt dann*

$$G(x_{\text{skalneu}}, s_{\text{skalneu}}) \leq G\left(x_{\text{skal}}, s_{\text{skal}}\right) - \frac{7}{120}.$$

Beweis: Wegen $\ln\left(\frac{\alpha}{\beta}\right) = \ln\alpha - \ln\beta$ und $x_{\text{skal}} = e$ gilt

$$\begin{aligned}
& G(x_{\text{skalneu}}, s_{\text{skalneu}}) - G(e, s_{\text{skal}}) \\
= \; & q \cdot \ln\left(e^T \cdot s_{\text{skal}} - \frac{g^T_{\text{projektion}} \cdot s_{\text{skal}}}{4 \cdot ||g_{\text{projektion}}||}\right) - \sum_{j=1}^{n} \ln\left[\left(1 - \frac{g_{\text{projektion},j}}{4 \cdot ||g_{\text{projektion}}||}\right) \cdot s_{\text{skal},j}\right] \\
& -q \cdot \ln\left(e^T \cdot s_{\text{skal}}\right) + \sum_{j=1}^{n} \ln s_{\text{skal},j} \\
= \; & q \cdot \ln\left(1 - \frac{g^T_{\text{projektion}} \cdot s_{\text{skal}}}{4 \cdot ||g_{\text{projektion}}|| \cdot (e^T \cdot s_{\text{skal}})}\right) - \sum_{j=1}^{n} \ln\left(1 - \frac{g_{\text{projektion},j}}{4 \cdot ||g_{\text{projektion}}||}\right).
\end{aligned}$$

Wir möchten die Logarithmus-Funktion eliminieren. Wegen

$$\ln(1-x) = \sum_{k=1}^{\infty} -\frac{x^k}{k} \geq -x - \frac{x^2}{2} \cdot \frac{1}{1-x}$$

ist für $x \leq \frac{1}{4}$

$$-x - \frac{2x^2}{3} = -x - \frac{x^2}{2} \cdot \frac{4}{3} \leq -x - \frac{x^2}{2} \cdot \frac{1}{1-x} \leq \ln(1-x) \leq -x$$

Also gilt

$$\begin{aligned}
& G(x_{\text{skalneu}}, s_{\text{skalneu}}) - G(e, s_{\text{skal}}) \\
\leq \; & (-q) \cdot \frac{g^T_{\text{projektion}} \cdot s_{\text{skal}}}{4 \cdot ||g_{\text{projektion}}|| \cdot (e^T \cdot s_{\text{skal}})} + \sum_{j=1}^{n} \frac{g_{\text{projektion},j}}{4 \cdot ||g_{\text{projektion}}||} + \sum_{j=1}^{n} \frac{2 \cdot (g_{\text{projektion},j})^2}{48 \cdot ||g_{\text{projektion}}||^2} \\
= \; & (-q) \cdot \frac{g^T_{\text{projektion}} \cdot s_{\text{skal}}}{4 \cdot ||g_{\text{projektion}}|| \cdot (e^T \cdot s_{\text{skal}})} + \frac{e^T \cdot g_{\text{projektion}}}{4 \cdot ||g_{\text{projektion}}||} + \frac{||g_{\text{projektion}}||^2}{24 \cdot ||g_{\text{projektion}}||^2} \\
= \; & \frac{1}{4 \cdot ||g_{\text{projektion}}||} \cdot \underbrace{\left(e^T - \frac{q \cdot s_{\text{skal}}}{e^T \cdot s^T_{\text{skal}}}\right)}_{= -g \text{ nach (8.3)}} \cdot g_{\text{projektion}} + \frac{1}{24}.
\end{aligned}$$

Aber es gilt

$$g^T \cdot g_{\text{projektion}} = g^T \cdot g_{\text{projektion}} - (g - g_{\text{projektion}})^T \cdot g_{\text{projektion}},$$

da $g - g_{\text{projektion}}$ orthogonal zu $\text{Kern}(A_{\text{skal}})$ und insbesondere orthogonal zu $g_{\text{projektion}} \in \text{Kern}(A_{\text{skal}})$ ist. Es folgt

$$g^T \cdot g_{\text{projektion}} = g^T_{\text{projektion}} \cdot g_{\text{projektion}} = ||g_{\text{projektion}}||^2$$

und wir erhalten wegen der Voraussetzung $||g_{\text{projektion}}|| > 0,4$:

$$G(x_{\text{skalneu}}, s_{\text{skalneu}}) - G(e, s_{\text{skal}}) \leq \frac{-||g_{\text{projektion}}||^2}{4 \cdot ||g_{\text{projektion}}||} + \frac{1}{24} \leq \frac{-0,4}{4} + \frac{1}{24} = -\frac{12}{120} + \frac{5}{120} = -\frac{7}{120}$$

und dies war zu zeigen. □

Um dieses Ergebnis einschätzen zu können, müssen wir den anfänglichen Wert und den zu erreichenden Wert der Funktion G kennen.

8.2.3 Der duale Schritt

Wir behandeln das letzte verbleibende algorithmische Problem: Was soll der Algorithmus machen, wenn der Gradient g fast orthogonal auf $\text{Kern}(A_{\text{skal}})$ steht, wenn also $||g_{\text{projektion}}|| \leq 0,4$ ist?
Statt einer Neuberechnung der Lösung x von (P_{skal}) werden wir in einem dualen Schritt die Lösung (y, s_{skal}) neuberechnen. Auch dabei wollen wir den Wert der Potentialfunktion G verringern, indem wir „in etwa“ in Richtung des negativen Gradienten

$$h := \nabla_s G\left(x_{\text{skal}}, s_{\text{skal}}\right) = \frac{q}{e^T \cdot s_{\text{skal}}} \cdot e - \begin{pmatrix} 1/s_{\text{skal},1} \\ 1/s_{\text{skal},2} \\ \vdots \\ 1/s_{\text{skal},n} \end{pmatrix}$$

laufen. Beachte, dass $h_j = \frac{g_j}{s_j}$ für $j = 1, \ldots, N$ ist, und g, h zeigen „in etwa“ in dieselbe Richtung. Wir müssen aus der alten Lösung $(y_{\text{skal}}, s_{\text{skal}})$ eine neue Lösung $(y_{\text{skalneu}}, s_{\text{skalneu}})$ für (D_{skal}) berechnen. Es muss also insbesondere

$$A_{\text{skal}}^T \cdot y_{\text{skalneu}} + s_{\text{skalneu}} = c_{\text{skal}}$$

gelten. Wegen $A_{\text{skal}}^T \cdot y_{\text{skal}} + s_{\text{skal}} = c_{\text{skal}}$ folgt

$$s_{\text{skalneu}} - s_{\text{skal}} = A_{\text{skal}}^T \cdot \left(y_{\text{skal}} - y_{\text{skalneu}}\right),$$

und $s_{\text{skalneu}} - s_{\text{skal}}$ ist stets eine Linearkombination der Spaltenvektoren von A_{skal}^T: Wähle $s_{\text{skalneu}} - s_{\text{skal}}$ als senkrecht auf $\text{Kern}(A_{\text{skal}})$ stehend. Wenn wir der Methode des Gradientenabstiegs folgen, ist der Gradient h auf den Orthogonalraum $\text{Kern}(A_{\text{skal}})^\perp$ zu projizieren. Stattdessen projizieren wir den Gradienten g auf den Orthogonalraum und weil

$g = g - g_{\text{projektion}} + g_{\text{projektion}}$ sowie

$g_{\text{projektion}}$ die Projektion von g auf $\text{Kern}(A_{\text{skal}})$ ist,

bildet $g - g_{\text{projektion}}$ die Projektion von g auf $\text{Kern}(A_{\text{skal}})^\perp$. Wir setzen

$$s_{\text{skalneu}} = s_{\text{skal}} - \frac{e^T \cdot s_{\text{skal}}}{q} \cdot \left(g - g_{\text{projektion}}\right)$$

und bestimmen y_{skalneu} als Lösung von

$$A_{\text{skal}}^T \cdot y_{\text{skalneu}} = c_{\text{skal}} - s_{\text{skalneu}}.$$

Wir wählen den Skalar $\frac{1}{q} \cdot \left(e^T \cdot s_{\text{skal}}\right)$, denn wegen $||g_{\text{projektion}}|| \leq 0,4$ gilt

$$s_{\text{skalneu}} = s_{\text{skal}} - \frac{e^T \cdot s_{\text{skal}}}{q} \cdot \left(\frac{q}{e^T \cdot s_{\text{skal}}} \cdot s_{\text{skal}} - e - g_{\text{projektion}}\right) = \frac{e^T \cdot s_{\text{skal}}}{q} \cdot \left(e + g_{\text{projektion}}\right)$$

und $s_{\text{skalneu}} > 0$. Auch im dualen Schritt können wir den Wert der Potentialfunktion G vermindern.

Lemma 8.4 Dualer Schritt
Sei $||g_{\text{projektion}}|| \leq 0,4$. *Für die Potentialfunktion* G *mit* $q = n + \sqrt{n}$ *gilt dann*

$$G\left(x_{\text{skal}}, s_{\text{skalneu}}\right) \leq G\left(x_{\text{skal}}, s_{\text{skal}}\right) - \frac{1}{6}.$$

Beweis: Für $e = (1, \ldots, 1)$ gilt

$$G\left(e, s_{\text{skalneu}}\right) - G\left(e, s_{\text{skal}}\right) = q \cdot \ln\left(\frac{e^T \cdot s_{\text{skalneu}}}{e^T \cdot s_{\text{skal}}}\right) - \sum_{j=1}^{n} \ln s_{\text{skalneu},j} + \sum_{j=1}^{n} \ln s_{\text{skal},j}. \quad (8.7)$$

Da die Logarithmus-Funktion konkav ist, folgt

$$\sum_{j=1}^{n} \frac{\ln s_{\text{skal},j}}{n} \leq \ln\left(\sum_{j=1}^{n} \frac{s_{\text{skal},j}}{n}\right) = \ln\left(\frac{e^T \cdot s_{\text{skal}}}{n}\right). \quad (8.8)$$

Andererseits gilt

$$\begin{aligned}
& \sum_{j=1}^{n} \ln s_{\text{skalneu},j} - n \cdot \ln\left(\frac{e^T \cdot s_{\text{skalneu}}}{n}\right) \\
= & \sum_{j=1}^{n} \ln\left(\frac{e^T \cdot s_{\text{skal}}}{q} \cdot \left[e + g_{\text{projektion},j}\right]\right) - n \cdot \ln\left(\frac{e^T \cdot s_{\text{skal}}}{q} \cdot \left[1 + \frac{e^T \cdot g_{\text{projektion},j}}{n}\right]\right) \\
= & \sum_{j=1}^{n} \ln\left(e + g_{\text{projektion},j}\right) - n \cdot \ln\left(1 + \frac{e^T \cdot g_{\text{projektion}}}{n}\right) \\
\geq & \sum_{j=1}^{n} \left(g_{\text{projektion},j} - \frac{\left(g_{\text{projektion},j}\right)^2}{2 \cdot 0,6}\right) - n \cdot \frac{e^T \cdot g_{\text{projektion}}}{n}. \quad (8.9)
\end{aligned}$$

Im letzten Schritt haben wir die Abschätzung

$$x - \frac{x^2}{2} \cdot \frac{1}{1 - 0,4} \leq \ln(1 + x) \leq x$$

benutzt. Die Abschätzung folgt aus der Entwicklung $\ln(1 + x) = \sum_{k=1}^{\infty} (-1)^{k+1} \cdot \frac{x^k}{k}$, wenn man $0 < x \leq 0,4$ fordert. Wegen $e^T \cdot g_{\text{projektion}} = \sum_{j=1}^{n} g_{\text{projektion},j}$ erhalten wir aus (8.9)

$$\sum_{j=1}^{n} \ln s_{\text{skalneu},j} - n \cdot \ln\left(\frac{e^T \cdot s_{\text{skalneu}}}{n}\right) \geq -\frac{10}{12} \cdot \sum_{j=1}^{n} g_{\text{projektion},j}^2 \quad = \quad -\frac{10}{12} \cdot ||g_{\text{projektion}}||^2.$$

Aus der Voraussetzung $||g_{\text{projektion}}|| \leq 0,4$ folgt

$$\sum_{j=1}^{n} \ln s_{\text{skalneu},j} - n \cdot \ln\left(\frac{e^T \cdot s_{\text{skalneu}}}{n}\right) \geq -\frac{10 \cdot 4^2}{12 \cdot 10^2} = -\frac{2}{15}. \quad (8.10)$$

Wir erhalten schliesßlich aus (8.7) in Verbindung mit (8.8) und 8.10

$$G\left(e, s_{\text{skalneu}}\right) - G\left(e, s_{\text{skal}}\right) \;\leq\; q \cdot \ln\left(\frac{e^T \cdot s_{\text{skalneu}}}{e^T \cdot s_{\text{skal}}}\right) + n \cdot \ln\left(\frac{e^T \cdot s_{\text{skal}}}{e^T \cdot s_{\text{skalneu}}}\right) + \frac{2}{15}.$$

Wir wenden die Voraussetzung $q = n + \sqrt{n}$ an und erhalten

$$\begin{aligned} G\left(e, s_{\text{skalneu}}\right) - G\left(e, s_{\text{skal}}\right) &\leq \left(n + \sqrt{n}\right) \cdot \ln\left(\frac{e^T \cdot s_{\text{skalneu}}}{e^T \cdot s_{\text{skal}}}\right) - n \cdot \ln\left(\frac{e^T \cdot s_{\text{skalneu}}}{e^T \cdot s_{\text{skal}}}\right) + \frac{2}{15} \\ &= \sqrt{n} \cdot \ln\left(\frac{e^T \cdot s_{\text{skalneu}}}{e^T \cdot s_{\text{skal}}}\right) + \frac{2}{15}. \end{aligned} \tag{8.11}$$

Es ist

$$s_{\text{skalneu}} = \frac{e^T \cdot s_{\text{skal}}}{q} \cdot \left(e + g_{\text{projektion}}\right)$$

und wegen $e^T \cdot e = n$, der Voraussetzung $||g_{\text{projektion}}|| \leq 0,4$ sowie der Cauchy-Schwarz-Ungleichung

$$|u^T \cdot v| \leq ||u|| \cdot ||u||$$

gilt

$$e^T \cdot s_{\text{skalneu}} = \frac{e^T \cdot s_{\text{skal}}}{q} \cdot \left(e^T \cdot e + e^T \cdot g_{\text{projektion}}\right) \leq \frac{e^T \cdot s_{\text{skal}}}{q} \cdot \left(n + 0,4 \cdot \sqrt{n}\right).$$

Wir erhalten aus (8.11) mit $q = n + \sqrt{n}$:

$$\begin{aligned} G\left(e, s_{\text{skalneu}}\right) - G\left(e, s_{\text{skal}}\right) &\leq \sqrt{n} \cdot \ln\left(\tfrac{e^T \cdot s_{\text{skalneu}}}{e^T \cdot s_{\text{skal}}}\right) + \tfrac{2}{15} \\ &\leq \tfrac{2}{15} + \sqrt{n} \cdot \ln\left(\tfrac{n+0,4\cdot\sqrt{n}}{n+\sqrt{n}}\right) \\ &\leq \tfrac{2}{15} + \sqrt{n} \cdot \ln\left(1 - \tfrac{0,6\cdot\sqrt{n}}{n+\sqrt{n}}\right) \\ &\leq \tfrac{2}{15} - \tfrac{0,6\cdot n}{n+\sqrt{n}} \end{aligned}$$

Wegen $\frac{0,6\cdot n}{n+\sqrt{n}} \geq \frac{0,6\cdot n}{2n} = \frac{9}{30}$ folgt die Behauptung, denn

$$G\left(e, s_{\text{skalneu}}\right) - G\left(e, s_{\text{skal}}\right) \leq \frac{4}{30} - \frac{9}{30} = -\frac{1}{6}.$$

und dies war zu zeigen. □

8.3 Zusammenfassung

Für die Laufzeitanalyse führen wir den Begriff der Größe eines linearen Programms ein.

Definition 8.5 Das lineare Programm $\min\{c^T \cdot x \mid Ax = b,\ x \geq 0\ \}$ habe nur rationale Koeffizienten, also $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$ und $c \in \mathbb{Q}^n$. Die Größe L des linearen Programms ist die Gesamtlänge der Binärdarstellung der Einträge von A, b und c (wobei jeweils die Binärdarstellungen von Zähler und Nenner zu zählen sind).

Algorithmus 8.6 **Der Interior-Point-Algorithmus von Ye.**

(1) Die Eingabe besteht aus einer rationalen $m \times n$-Matrix A mit $m \leq n$ und $\text{Rang}(A) = m$, einem rationalen Vektor b und der zu minimierende Zielfunktion $x \mapsto c^T \cdot x$ mit rationalen Koeffizienten. Schließlich ist noch die Größe L des linearen Programms gegeben.

(2) Transformiere das Problem auf (P_{neu}) und (D_{neu}). Bestimme eine innere Lösung $x^{(0)}$ zu (P_{neu}) und eine innere Lösung $\left(y^{(0)}, s^{(0)}\right)$ zu (D_{neu}) mit $G\left(x^{(0)}, s^{(0)}\right) = O(\sqrt{n} \cdot L)$. Schließlich setze $i = 0$.

(3) WHILE $(G\left(x^{(i)}, s^{(i)}\right) > -2\sqrt{n}L)$ DO

(3a) Setze $x_{\text{alt}} = x^{(i)}$ und $s_{\text{alt}} = s^{(i)}$

(3b) Skaliere

(3c) Berechne den Gradienten $g := \nabla_x G\,(x_{\text{skal}}, s_{\text{skal}})$.
Wenn $||g_{\text{projektion}}|| \geq 0,4$, dann führe den primalen Schritt durch: Setze

$$\begin{aligned} x_{\text{skalneu}} &:= x_{\text{skal}} - \frac{1}{4 \cdot ||g_{\text{projektion}}||} \cdot g_{\text{projektion}} \\ y_{\text{skalneu}} &:= y_{\text{skal}} \ \text{ und } s_{\text{skalneu}} := s_{\text{skal}}. \end{aligned}$$

Ansonsten befinden wir uns im dualen Schritt. Bestimme y_{skalneu} als Lösung von $A_{\text{skal}} \cdot y_{\text{skalneu}} = c_{\text{skal}} - s_{\text{skalneu}}$. Setze

$$\begin{aligned} x_{\text{skalneu}} &:= x_{\text{skal}} \\ s_{\text{skalneu}} &:= s_{\text{skal}} - \frac{e^T \cdot s_{\text{skal}}}{q} \cdot (g - g_{\text{projektion}})\,. \end{aligned}$$

(3d) Reskaliere und berechne $x^{(i+1)}$, $y^{(i+1)}$ und $s^{(i+1)}$. Setze $i = i + 1$.

Unsere Analyse war bisher exakt. Für die weiteren Betrachtungen geben wir meist nur die Resultate an und verweisen für die Beweise auf die entsprechenden Originalarbeiten. Sei L die Größe des linearen Programms. Es gilt:

(1) Man kann L_1, L_2, L_3, x_0, s_0 für (P_{neu}) und (D_{neu}) so wählen, dass für die inneren Lösungen $x^{(0)}, s^{(0)}$ stets $G\left(x^{(0)}, s^{(0)}\right) = O(\sqrt{n} \cdot L)$ gilt.

(2) Wenn $G\left(x^{(i)}, s^{(i)}\right) \leq -2\sqrt{n} \cdot L$, ist jede Ecke x^* mit $c^T \cdot x^* \leq c^T \cdot x^{(i)}$ optimal und eine solche Ecke kann effizient gefunden werden.

Wir nehmen $G\left(x^{(i)}, s^{(i)}\right) \leq -2\sqrt{n} \cdot L$ an. Beachte, dass wegen

$$\begin{aligned}
-2\sqrt{n} \cdot L &\geq G\left(x^{(i)}, s^{(i)}\right) \\
&= (n + \sqrt{n}\,) \cdot \ln\left({x^{(i)}}^T \cdot s^{(i)}\right) - \sum_{j=1}^{n} \ln\left(x_j^{(i)} \cdot s_j^{(i)}\right) \\
&\geq (n + \sqrt{n}\,) \cdot \ln\left({x^{(i)}}^T \cdot s^{(i)}\right) - n \cdot \ln\left(\frac{{x^{(i)}}^T \cdot s^{(i)}}{n}\right) \\
&= \sqrt{n} \cdot \ln\left({x^{(i)}}^T \cdot s^{(i)}\right) + n \cdot \ln n
\end{aligned}$$

dann

$$\ln\left({x^{(i)}}^T \cdot s^{(i)}\right) \leq -2L - \sqrt{n} \cdot \ln n \leq -2L$$

gilt und die Dualitätslücke ist höchstens e^{-2L}: Interior Point Verfahren finden sehr schnell sehr gute approximative Lösungen! Wir fassen unsere Resultate zusammen.

Satz 8.7 *Der Interior-Point-Algorithmus 8.6 löst ein lineares Programm der Größe L in n Variablen in Zeit $O(n^{3,5} \cdot L)$. Hier nehmen wir an, dass eine arithmetische Operation in einem Schritt ausführbar ist.*

Beweis: Wir benötigen $O(\sqrt{n} \cdot L)$ Iterationen. Jede Iteration gelingt in Zeit $O(n^3)$, da nur Matrixinversionen, Matrizenprodukte und Gleichungssysteme zu berechnen bzw. zu lösen sind. □

Für weitere Informationen über Interior-Point Verfahren verweisen wir auf [Ye2]. Es ist unbekannt, ob es Algorithmen für die lineare Programmierung gibt, deren Laufzeit polyomiell in der Anzahl der Ungleichungen und der Anzahl der Variablen ist, d.h. es ist unbekannt, ob es „starke polynomielle“ Algorithmen gibt. Dabei zählen wir Elementarschritte (Addition, Multiplikation etc. zweier Zahlen) unabhängig von der Operandengröße als eine Zeiteinheit.

Kapitel 9

Lineare Programmierung und Approximation

Mit der linearen Programmierung erhalten wir ein mächtiges Werkzeug zur Entwicklung von Approximationsalgorithmen. Insbesondere, wenn wir ein Minimierungsproblem (bzw. Maximierungsproblem) P_z für eine Instanz z approximativ lösen möchten, bietet sich das folgende Verfahren an:

(1) Formuliere P_z als ein lineares Programm P'_z mit Integralitätsbedingungen. Den optimalen Wert von P'_z Wert bezeichnen wir mit $\texttt{opt}(P'_z)$.

(2) Relaxiere P: Wir erhalten ein lineares Programm P''_z ohne Integralitätsbedingungen.

(3) Sei x eine optimale Lösung von P'' und sei $\texttt{opt}(P''_z)$ der optimale Wert für P''.

$$\texttt{opt}(P''_z) \leq \texttt{opt}(P'_z) \text{ und } \texttt{opt}(P''_z) \text{ ist eine untere (bzw. obere) Schranke für } P.$$

(4) „Runde“ x: Wir erhalten eine integrale Lösung x^* für P.

(5) Analysiere den Approximationsfaktor.

Im allgemeinen wird die Relaxierung den optimalen Wert senken. Der Erfolg dieses Ansatzes hängt deshalb entscheidend von der **Integralitätslücke**

$$\sup_z \frac{\texttt{opt}(P'_z)}{\texttt{opt}(P''_z)}$$

ab.

Beispiel 9.1 VERTEX COVER
Wir errinern an die Definition von VERTEX COVER. Für einen ungerichteten Graphen $G = (V, E)$ und Knotengewichte w_u suchen wir eine überdeckende Knotenmenge $V' \subseteq V$ mit kleinstem Gewicht. (V' ist überdeckend, wenn jede Kante in G mindestens einen Endpunkt in V' besitzt.)

Das lineare Programm

$$(P'_G) \quad \text{minimiere } \sum_{v\in V} w_v \cdot x_v, \text{ so dass } x_u + x_v \geq 1 \text{ für alle } \{u,v\} \in E \\ \text{und } x_u \in \{0,1\} \text{ für alle } u \in V$$

beschreibt VERTEX COVER. Wir ersetzen die Integralitätsbedingungen $x_u \in \{0,1\}$ durch ihre Relaxierung $0 \leq x_u \leq 1$ und erhalten das lineare Programm

$$(P''_G) \quad \text{minimiere } \sum_{v\in V} w_v \cdot x_v, \text{ so dass } x_u + x_v \geq 1 \text{ für alle } \{u,v\} \in E \\ \text{und } 0 \leq x_u \leq 1 \text{ für alle } u \in V.$$

Wie groß ist die Integralitätslücke mindestens? Betrachte den vollständigen Graphen V_n mit n Knoten, die alle das Gewicht eins erhalten. Eine minimale Überdeckung enthält $n-1$ Knoten, die Relaxierung (P''_{V_n}) hingegen besitzt den Vektor $x = (0.5, \ldots, 0.5)$ als Lösung. Also ist

$$\frac{\texttt{opt}(P'_{V_n})}{\texttt{opt}(P''_{V_n})} \geq \frac{n-1}{n/2} = 2 \cdot (1 - \frac{1}{n}),$$

und die Integralitätslücke kommt zwei beliebig nahe. Wir werden im übernächsten Abschnitt sehen, dass geschicktes Runden zum Approximationsfaktor zwei führt; insbesondere zeigt dies, dass die Integralitätslücke höchstens zwei beträgt.

Wir beginnen mit einer ersten Anwendung der linearen Programmierung, in der wir ohne Rundung auskommen: Wir bestimmen eine beliebige Ecke x und werden feststellen, dass „viele“ Komponenten von x ganzahlig sind.

9.1 Scheduling für nicht-identische Maschinen

Die Menge $A = \{1, \ldots, n\}$ von n Aufgaben und die Menge $M = \{1, \ldots, m\}$ der Maschinen ist gegeben. Wir nehmen an, dass die Maschinen nicht korreliert sind, und dementsprechend haben Aufgaben unterschiedliche Laufzeiten auf den jeweiligen Maschinen: Insbesondere sei t_i^j die Laufzeit von Aufgabe i auf Maschine j.
Mit dem folgenden 0-1 Programm können wir die Mininmierung des Makespans ausdrücken:

$$\text{minimiere } t, \text{ so dass} \quad \sum_{j\in M} x_{i,j} = 1 \text{ für alle Aufgaben } i \in A, \\ \sum_{i\in A} t_i^j \cdot x_{i,j} \leq t \text{ für alle Maschinen } j \in M \text{ und } x_{i,j} \in \{0,1\}.$$

Eine Lösung mit $x_{i,j} = 1$ impliziert, dass Aufgabe A_i auf Maschine j auszuführen ist. Die Relaxierung, also die Ersetzung von $x_{i,j} \in \{0,1\}$ durch $0 \leq x_{i,j} \leq 1$, erlaubt leider, dass eine Aufgabe über verschiedene Maschinen verteilt werden kann. Insbesondere wird die Relaxierung für eine Aufgabe mit den Laufzeiten $t_1^1 = t_1^2 = \cdots = t_1^m = t$ die Laufzeit

$\frac{t}{m}$ liefern, während t natürlich das tatsächlich erreichbare Optimum darstellt. Für eine große Zahl von Maschinen wird die Relaxierung also die „Realität“ zu stark verzerren. Stattdessen betrachten wir für einen Parameter T und die Menge $P_T = \{ (i,j) \mid t_i^j \leq T \}$ das Ungleichungssystem

$$\begin{aligned} \sum_{j \in M \text{ mit } (i,j) \in P_T} x_{i,j} &= 1 \text{ für alle Aufgaben } i \in A, \\ \sum_{i \in A \text{ mit } (i,j) \in P_T} t_i^j \cdot x_{i,j} &\leq T \text{ für alle Maschinen } j \in M \text{ und} \\ x_{i,j} &\geq 0 \text{ für alle } (i,j) \in P_T. \end{aligned}$$

In unseren neuen Restriktionen fordern wir also implizit für jede Maschine j, dass j in Zeit T nur Aufgaben erledigen kann, für die j individuell höchstens Zeit T benötigt. Wir bestimmen T durch eine binäre Suche auf dem Intervall $[0, \sum_{i,j} t_i^j]$, indem wir nach dem kleinsten T suchen für das das lineare Programm zu T lösbar ist. Beachte, dass dann $T \leq \texttt{opt}$ gilt, wobei `opt` der optimale Makespan ist.
Die Matrix des Ungleichungssystems besitzt maximalen Rang. Wir besitzen genau die $|P_T|$ vielen Variablen $x_{i,j}$ und erhalten Ecken, indem wir $|P_T|$ viele Ungleichungen exakt erfüllen. Genau $|P_T|$ viele Ungleichungen sind von der Form $x_{i,j} \geq 0$, n Ungleichungen fordern, dass alle Aufgaben ausgeführt werden und m Ungleichungen erzwingen, dass keine Maschine mehr als Zeit T benötigt. In einer Ecke müssen also mindestens $|P_T| - n - m$ Ungleichungen $x_{i,j} \geq 0$ exakt erfüllt werden. Als Konsequenz:

Eine Ecke des Polytops besitzt höchstens $n+m$ von Null verschiedene Komponenten.

Wir bestimmen zuerst eine Ecke x^T für das gerade bestimmte T und konstruieren eine Ausführungsfolge aus x. Insbesondere definieren wir den bipartiten Graphen $G = (A \cup M, E)$ mit Kantenmenge $E = \{ (i,j) \mid x_{i,j}^T \neq 0 \}$. Die Menge A aller Aufgaben zerfällt in die integralen Aufgaben, also Aufgaben i mit $x_{i,j} = 1$ für ein j, und in die Menge A_F der fraktionalen Aufgaben. Wir bezeichnen den durch $A_F \cup M$ induzierten Teilgraph von G mit H. Beachte, dass (i,j) genau dann eine Kante von H ist, wenn $0 < x_{i,j}^T < 1$. Weiterhin erhält man H aus G, indem man alle integralen Aufgaben i aus G entfernt.
Wir werden zeigen, dass H ein perfektes Matching besitzt. Dementsprechend können wir in einer ersten Phase die integralen Aufgaben gemäß den Vorgaben von x^T ausführen und sodann in einer zweiten Phase die verbleibenden Aufgaben, dem perfekten Matching folgend, ausführen. Das perfekte Matching belastet aber jede Maschine höchstens einmal und der Makespan der zweiten Phase ist sicherlich durch den optimalen Makespan `opt` beschränkt. Aber auch der Makespan der ersten Phase ist durch den optimalen Makespan beschränkt: Es ist $T \leq \texttt{opt}$ und damit können insbesondere alle integralen Aufgaben im Makespan höchstens `opt` ausgeführt werden. Wir erhalten also einen 2-approximativen Algorithmus. Warum aber besitzt H ein perfektes Matching?

Lemma 9.1

(a) Jede Zusammenhangskomponente von G ist ein Pseudobaum, besitzt also höchstens so viele Kanten wie Knoten.

(b) Der Graph H besitzt ein perfektes Matching.

Beweis (a): Sei x^T die gefundene Ecke. Wir betrachten eine Zusammenhangskomponente Z von G sowie das auf die Kanten von Z eingeschränkte Ungleichungssystem. Die Ecke x^T zerfällt dann in den Vektor x_Z^T der Kanten von Z und den Vektor y_Z^T der restlichen Kanten. Wir behaupten, dass x_Z^T eine Ecke des eingeschränkten Ungleichungssystems ist. Ist dies nicht der Fall, dann ist sowohl $x_Z^T - u$ wie auch $x_Z^T + u$ eine Lösung und damit sind aber auch $(x_Z^T - u, y_Z^T)$ und $(x_Z^T + u, y_Z^T)$ Lösungen des vollständigen Ungleichungssystems und $x^T = (x_Z^T, y_Z^T)$ ist keine Ecke.
Also ist x_Z^T eine Ecke des eingeschränkten Ungleichungssystems und wir können die zentrale Beobachtung auf das eingeschränkte Ungleichungssystem anwenden: Die Anzahl der Kanten von Z, also die Anzahl der von Null verschiedenen Komponenten von x_Z^T, ist durch die Anzahl der Knoten beschränkt!

(b) Beachte, dass eine integrale Aufgabe nur die Kante zu „ihrer“ Maschine in G besitzt. Also erhält man H aus G, indem man genau so viele Knoten wie Kanten aus G entfernt: Die Zusammenhangskomponenten von H bleiben Pseudobäume.
Wie sehen die Pseudobäume aus? Jede Aufgabe wird auf mindestens zwei Maschinen verteilt und besitzt deshalb mindestens zwei Kanten. Also entsprechen die Blätter bestimmten Maschinen. Wir können also in einer ersten Phase alle „Blatt-Maschinen“ mit ihren „Väter-Aufgaben“ matchen und die Kanten des partiellen Matchings mitsamt ihren Endpunkten entfernen. In der zweiten und folgenden Phasen wiederholen wir dieses Vorgehen. Wenn keine Blätter mehr vorhanden sind, dann ist der Pseudobaum entweder leer oder wir sind auf den einzigen Zyklus gestoßen.
G und damit auch H ist ein bipartiter Graph und damit ist auch jeder Pseudobaum bipartit und besitzt nur Kreise gerader Länge. Wir erhalten deshalb ein perfektes Matching, wenn wir jede zweite Kante des Kreises zu unserem Matching hinzufügen. □

Wir werden also auf den folgenden Algorithmus geführt.

Algorithmus 9.2 **Scheduling für unkorrelierte Maschinen.**

(1) Bestimme einen optimalen Parameter T durch binäre Suche. Sei x^T eine Ecke.

(2) Führe zuerst alle integralen Aufgaben gemäß x^T aus.

(3) Bestimme ein perfektes Matching mit dem Verfahren des Beweises von Lemma 9.1 (a) und führe die verbleibenden Aufgaben nach diesem Matching aus.

Satz 9.3 *Algorithmus 9.2 erreicht den Approximationsfaktor 2 für das Scheduling Problem für nicht-identische Maschinen.*
Wenn $\mathcal{P} \neq \mathcal{NP}$ kann gezeigt werden, dass effiziente Algorithmen einen Approximationsfaktor kleiner als $3/2$ nicht erreichen.

Unsere Analyse lässt sich nicht verbessern. Dazu wählen wir bei m Maschinen $m^2 - m + 1$ Aufgaben, die bis auf die erste Aufgabe auf allen Maschinen in einem Schritt ausführbar sind. Die erste Aufgabe benötige auf jeder Maschine Laufzeit m. Wir können Makespan m erreichen, wenn wir die erste Aufgabe der ersten Maschine geben und die restlichen $m^2 - m = (m-1) \cdot m$ Aufgaben auf die restlichen $m-1$ Maschinen gleichmäßig verteilen. Aber wir erhalten auch eine sehr ärgerliche Ecke, wenn wir den Bruchteil $\frac{1}{m}$ der ersten Aufgabe auf jede der Maschinen verteilen und ansonsten jeweils $m-1$ kurze Aufgaben pro Maschine ausführen lassen.
Die Ausführung der (kurzen) integralen Aufgaben hält jede Maschine bis zum Zeitpunkt $m-1$ beschäftigt, die Ausführung der fraktionalen Aufgabe führt auf den Makespan $2 \cdot m - 1$.

9.2 Approximation durch Rundung

Wir verfolgen den zu Beginn des Kapitels beschriebenen Ansatz weiter und approximieren ein Optimierungsproblem P_z durch ein relaxiertes lineares Programm P_z''. Wie erhalten wir eine Lösung x^* des ursprünglichen Problems P_z aus der optimalen, fraktionalen Lösung x für P_z''? Wir besprechen zwei Ansätze, nämlich deterministisches und randomisiertes Runden.

9.2.1 VERTEX COVER

Wir beginnen mit einem kombinatorischen Algorithmus für den ungewichteten Fall.

Algorithmus 9.4

(1) Die Eingabe besteht aus einem ungerichteten Graphen $G = (V, E)$.

(2) Setze $M := \emptyset$. WHILE $(E \neq \emptyset)$ DO

Wähle eine Kante $e \in E$ und füge e zu M hinzu.

Entferne alle Kanten aus E, die einen Endpunkt mit e gemeinsam haben.

(3) Gib die Knotenmenge $V' = \{v \in V \mid v \text{ ist Endpunkt einer Kante in } M \}$ aus.

Satz 9.5 *Algorithmus 9.4 ist ein 2-Approximationsalgorithmus für VERTEX COVER mit identischen Knotengewichten.*

Beweis: Sei $\text{vc}(G)$ die minimale Grösse einer überdeckenden Knotenmenge V'. Offensichtlich haben keine zwei Kanten aus M einen gemeinsamen Endpunkt und M is deshalb ein Matching. Jede überdeckende Knotenmenge muss aber dann mindestens einen Endpunkt für jede Kante aus M besitzen. Also ist

$$|M| \leq \text{vc}(G).$$

Andererseits ist das Matching M nicht erweiterbar, und damit hat jede Kante $e \in E$ einen Endpunkt mit einer Kante aus M gemeinsam, d.h. der Algorithmus findet eine überdeckende Knotenmenge V' der Größe

$$|V'| = 2 \cdot |M| \leq 2 \cdot \mathrm{vc}(G).$$

Wir haben damit eine 2-approximative Lösung erhalten. □

Bemerkung 9.1 Natürlich stellt sich die Frage, ob Algorithmus 9.4 eine bessere Approximationskonstante besitzt. Leider ist die Antwort negativ: Der vollständige bipartite Graph mit n Knoten „auf jeder Seite“ besitzt natürlich eine überdeckende Knotenmenge der Grösse n, aber Algorithmus 9.4 wird alle $2 \cdot n$ Knoten als Überdeckung ermitteln.

Für den allgemeinen Fall beliebiger nichtnegativer Gewichtungen greifen wir die lineare Relaxierung aus Beispiel 9.1 wieder auf, betrachten also das lineare Programm

$$\text{minimiere } \sum_{v \in V} w_v \cdot x_v, \qquad \text{so dass } x_u + x_v \geq 1 \text{ für alle } \{u,v\} \in E \qquad (9.1)$$
$$\text{und } x_u \geq 0 \text{ für alle } u \in V.$$

Sei x^* eine optimale Lösung und sei $\mathrm{vc}^*(G)$ das minimale Gewicht einer überdeckenden Knotenmenge V'. Dann ist

$$\sum_{v \in V} w_v \cdot x_v^* \leq \mathrm{vc}^*(G),$$

denn für die optimale überdeckenden Knotenmenge $V' \subseteq V$ ist der Inzidenzvektor $x^{V'}$ mit $x_v^{V'} = \begin{cases} 1 & v \in V', \\ 0 & \text{sonst} \end{cases}$ eine Lösung des linearen Programms und es ist $\sum_{v \in V} w_v \cdot x_v^{V'} = \mathrm{vc}^*(G)$. Da x^* im allgemeinen ein reellwertiger Vektor ist, runden wir x^*, um eine überdeckende Knotennmenge zu erhalten. Wir setzen also

$$V' = \{v \in V \mid x_v^* \geq \frac{1}{2}\}.$$

V' ist tatsächlich eine überdeckende Knotenmenge, denn für jede Kante $\{u,v\} \in E$ ist die Ungleichung $x_u^* + x_v^* \geq 1$ erfüllt und damit folgt $x_u^* \geq \frac{1}{2}$ oder $x_v^* \geq \frac{1}{2}$. Andererseits ist

$$\sum_{v \in V'} w_v \leq \sum_{v \in V} 2 \cdot w_v \cdot x_v^* \leq 2 \cdot \mathrm{vc}^*(G),$$

denn $x_v^* \geq \frac{1}{2}$ für $v \in V'$, und wir haben eine 2-approximative Lösung erhalten.

Satz 9.6 *Das gewichtete VERTEX COVER Problem kann mit Approximationsfaktor* 2 *effizient gelöst werden.*

Wir haben jetzt „zwei Waffen“ in unserem Arsenal, da der Local Ratio Algorithmus 5.1 ebenfalls den Approximationsfaktor zwei erreicht. Vom Standpunkt der Laufzeit ist Algorithmus 5.1 sogar vorzuziehen.
Wir können relativ genaue Aussagen über das Polytop des Minimierungsproblems (9.1) machen.

Lemma 9.7 *Wenn x eine Ecke des Polytops zu (9.1) ist, dann ist $x \in \{0, \frac{1}{2}, 1\}^*$.*
(Leider tritt also der gefährliche Fall $x_i = \frac{1}{2}$ „häufig“ auf.)

Beweis: Sei x eine Lösung von (9.1). Wir nehmen an, dass $x \notin \{0, \frac{1}{2}, 1\}^*$ und müssen zeigen, dass es einen Vektor y gibt, so dass sowohl $x + y$ wie auch $x - y$ Lösungen sind.
Für die Konstruktion von y betrachten wir die Positionen, die Werte ausserhalb der Menge $\{0, \frac{1}{2}, 1\}$ annehmen. Insbesondere setzen wir

$$I_< = \{i \mid 0 < x_i < \frac{1}{2}\} \text{ und } I_> = \{i \mid \frac{1}{2} < x_i < 1\}$$

und definieren den Vektor y komponentenweise durch

$$y_i = \begin{cases} \varepsilon & i \in I_< \\ -\varepsilon & i \in I_> \\ 0 & \text{sonst.} \end{cases}$$

Wie ist die positive reelle Zahl ε zu wählen? Zuerst wähle $\varepsilon > 0$ so klein, dass sowohl $x - y$ wie auch $x + y$ nur nicht-negative Komponenten besitzen. Weiterhin müssen wir für jede Kante $\{u, v\} \in E$ garantieren, dass $(x_u \pm y_u) + (x_v \pm y_v) \geq 1$.
Wenn $x_u + x_v > 1$, dann ist die Garantie durch eine möglicherweise zusätzliche Verkleinerung von ε problemlos möglich. Ansonsten ist $x_u + x_v = 1$ mit den vier Möglichkeiten $x_u = 1, x_v = 0$ oder $x_u = 0, x_v = 1$ oder $x_u = \frac{1}{2} = x_v$ oder o.B.d.A. $u \in I_<$ und $v \in I_>$. Wir müssen nur den letzten Fall behandeln, da in allen anderen Fällen $x_u, x_v \in \{0, \frac{1}{2}, 1\}$ gilt. Aber im letzten Fall ist $y_u + y_v = 0$ und $(x_u \pm y_u) + (x_v \pm y_v) = 1$ folgt. □

9.2.2 SET COVER

Die Teilmengen $T_1, \ldots, T_m \subseteq \{1, \ldots, n\}$ sind gegeben, wobei Teilmenge T_i das Gewicht g_i besitzt. Wir müssen eine leichteste Überdeckung des Universums $\{1, \ldots, n\}$ bestimmen.
Wir erinnern an die Relaxierung

$$\text{opt} = \text{minimiere } \sum_{k=1}^{m} g_k \cdot x_k, \text{ so dass } \quad \sum_{k,\, j \in T_k} x_k \geq 1 \text{ für alle } j \in \{1, \ldots, n\} \quad (9.2)$$
$$\text{und } x_k \geq 0 \text{ für } k = 1, \ldots m.$$

von SET COVER aus Abschnitt 7.4. Sei x eine optimale Lösung von (9.2)

Aufgabe 92
Zeige, dass es alles Andere als eine gute Idee ist, wenn alle positiven Komponenten der optimalen primalen Lösung x nach 1 aufgerundet werden.

Was ist von der Idee des **randomisierten Rundens** zu halten, also der Idee, die Menge T_i mit Wahrscheinlichkeit x_i in die Überdeckung zu übernehmen? Sehr viel, wie wir gleich feststellen werden!
Zuerst betrachten wir das erwartete Gewicht E_x der berechneten Überdeckung und erhalten

$$E_x = \sum_{i=1}^{m} g_i \cdot \text{prob}[\, T_i \text{ wird gewählt }] = \sum_{i=1}^{m} g_i \cdot x_i \leq \texttt{opt}.$$

Als nächstes muss uns interessieren mit welcher Mindestwahrscheinlichkeit ein fixiertes Element j überdeckt wird. Angenommen, j kommt in den ersten k Mengen, aber nicht in den verbleibenden $m-k$ Mengen vor. Dann ist

$$\sum_{i=1}^{k} x_i \geq 1,$$

denn die fraktionale Lösung muss j fraktional überdecken. Also folgt[1]

$$\text{prob}[\ j \text{ wird nicht überdeckt }\ ] \leq (1-\frac{1}{k})^k \leq e^{-k/k} \leq e^{-1}.$$

Aufgabe 93
Unabhängige Ereignisse $E_1, \ldots, E_k$ mögen mit den Wahrscheinlichkeiten $p_1, \ldots, p_k$ eintreten, wobei $\sum_{i=1}^{k} p_i \geq 1$ gelte. Zeige

$$\text{prob}[\text{ kein Ereignis tritt ein }] \leq (1-\frac{1}{k})^k.$$

Um eine vollständige Überdeckung zu erhalten, wiederholen wir die zufällige Wahl $K = 1+\ln n$ mal und erhalten

$$\text{prob}[\ j \text{ wird nach } K \text{ Versuchen nicht überdeckt }\ ] \leq e^{-K} \leq \frac{1}{2n}.$$

Andererseits ist

$$\begin{aligned} & \text{prob}[\ \text{ nach } K \text{ Versuchen wird mindestens ein Element nicht überdeckt }\ ] \\ \leq\ & n \cdot \text{prob}[\ \text{ Element 1 wird nach } K \text{ Versuchen nicht überdeckt }\ ] \leq \frac{1}{2}. \end{aligned}$$

Satz 9.8 *Mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ erhalten wir durch randomisiertes Runden eine vollständige Überdeckung mit Gewicht höchstens* $\texttt{opt} \cdot (1+\ln n)$.

9.2.3 Ein Routing Problem

Gegeben ist ein gerichteter Graph $G=(V,E)$ und ausgezeichnete Knoten $s_1, \ldots, s_r$ und $t_1, \ldots, t_r$. Das Ziel ist die Bestimmung von Wegen $P_1, \ldots, P_r$, so dass

- P_i im Knoten s_i beginnt und im Knoten t_i endet und
- so dass die maximale Belastung

$$\max_{e \in E} \{ \text{ Anzahl der Wege } P_i, \text{ die über die Kante } e \text{ laufen } \}$$

einer Kante möglichst klein ist.

[1] Es ist $e^{-x/(1-x)} \leq 1-x \leq e^{-x}$.

Dieses Routingproblem tritt in Rechnernetzen auf, wenn Nachrichten zwischen Rechnern des Netzes auszutauschen sind: Natürlich sind die Nachrichten so zu „routen“, dass die Belastung der direkten Rechnerverbindungen möglichst klein ist. Wir werden dieses Problem mit der Methode des randomisierten Rundens approximativ lösen.
Für eine Formulierung als ganzahliges lineares Programm verwenden wir die 0-1 wertigen Variablen $x_i(e)$ für $1 \leq i \leq r$ und $e \in E$. Insbesondere soll $x_i(e)$ genau dann den Wert 1 annehmen, wenn P_i über die Kante e läuft. Ein Pfad P_i wird dann für jedes $v \in V$ durch die Bedingungen

$$\sum_{w,(v,w)\in E} x_i(v,w) - \sum_{w,(w,v)\in E} x_i(w,v) = \begin{cases} 1 & v = s_i \\ -1 & v = t_i \\ 0 & \text{sonst} \end{cases}$$

„beschrieben“: Wenn ein Pfad über den Knoten v als inneren Knoten läuft, dann geschieht dies über die beiden Kanten (w, v) und (v, w'). Die obige Bedingung ist also stets erfüllt, solange $x_i(u, w) = 1$ genau dann gilt, wenn die Kante (u, w) von P_i durchlaufen wird.
Sei B_i die entsprechende Menge der Bedingungen. Das 0-1 Programm hat dann die Form

$$\begin{aligned} &\text{minimiere } W, \text{ so dass} \quad && \text{die Bedingungen } B_i \text{ für } i = 1, \ldots, r \text{ gelten,} \\ &&& \sum_{i=1}^{r} x_i(v,w) \leq W, \text{ und } x_i(v,w) \in \{0,1\} \text{ ist für} \\ &&& i = 1, \ldots, r \text{ und alle Kanten } (v,w) \in E \text{ erfüllt.} \end{aligned} \tag{9.3}$$

Unser Approximationsalgorithmus wird in seinem ersten Schritt die Relaxierung von (9.3) exakt lösen und *fraktionale* Werte $x_i^*(v, w)$ bestimmen. Wir fixieren i und betrachten den Graphen $G_i^* = (V, E_i^*)$, der aus allen Kanten $e \in E$ mit $x_i^*(v, w) > 0$ besteht.
Die Bedingungen B_i fordern einen „Fluss“ vom Wert 1 von s_i nach t_i im Graphen G_i^*. Dementsprechend besitzt G_i^* einen Weg von s_i nach t_i und wir können einen ersten Weg $P_{i,1}$ von s_i nach t_i bestimmen. Wenn $\alpha_{i,1}$ der minimale Wert $x_i^*(e)$ einer Kante e von $P_{i,1}$ ist, dann reduzieren wir den Wert $x_i^*(e)$ einer jeden Kante e von $P_{i,1}$ um $\alpha_{i,1}$. Wir führen sodann eine Neuberechnung von G_i^* durch und beachten, dass die Bedingungen B_i jetzt einen Fluss vom Wert $1 - \alpha_{i,1}$ von s_i nach t_i im Graphen G_i^* garantieren. Also können wir auch jetzt, $\alpha_{i,1} < 1$ vorausgesetzt, einen Weg $P_{i,2}$ und sein zugehöriges „Gewicht“ $\alpha_{i,2}$ bestimmen.
Dieses Verfahren wird solange wiederholt bis es keinen Weg von s_i nach t_i mehr gibt. Beachte, dass wir in jeder Iteration mindestens eine Kante entfernen: Wir führen höchstens $|E|$ viele Iterationen durch.

Algorithmus 9.9 **Routing durch randomisiertes Runden.**

(1) Löse die Relaxierung von (9.3) und bestimme die optimalen fraktionalen Werte $x_i^*(v, w)$.

(2) „Zerlege“ die fraktionale Lösung in die Wege $P_{i,1}, P_{i,2}, \ldots$ und bestimme die zugehörigen Gewichte $\alpha_{i,1}, \alpha_{i,2}, \ldots$.

Kommentar: Wir können die Gewichte $\alpha_{i,j}$ als Wahrscheinlichkeiten auffassen, da stets $\alpha_{i,j} > 0$ und $\sum_j \alpha_{i,j} = 1$ gilt. Desweiteren ist

$$\sum_{j,(v,w) \text{ liegt auf } P_{i,j}} \alpha_{i,j} \leq x_i^*(v,w). \tag{9.4}$$

(3) Bestimme die Wege P_i zufällig, wobei $P_i = P_{i,j}$ mit Wahrscheinlichkeit $\alpha_{i,j}$ gewählt wird.

Wir kommen zur Analyse von Algorithmus 9.9. Zuerst fixieren wir eine Kante $e = (v,w)$ von G. Sei X_i die binäre Zufallsvariable mit $X_i = 1$ genau dann, wenn e auf dem Weg P_i liegt. Dann ist $W(v,w) = \sum_{i=1}^r X_i$ die Belastung der Kante (v,w). Wie groß ist die erwartete Belastung $W(v,w)$ der Kante (v,w)? Es ist

$$p_i = \text{prob}[\, X_i = 1 \,] = \sum_{j,(v,w) \text{ liegt auf } P_{i,j}b} \alpha_{i,j}$$

die Wahrscheinlichkeit, dass (v,w) auf dem Weg P_i liegt. Wir erinnern uns daran, dass damit $p_i \leq x_i^*(v,w)$ folgt. Wenn also W^* der optimale Wert einer fraktionalen Lösung für (9.3) und W_{opt} der optimale Wert einer integralen Lösung ist, dann erhalten wir

$$\mathbb{E}[\, W(v,w) \,] = \sum_{i=1}^r p_i \leq \sum_{i=1}^r x_i^*(v,w) \leq W^* \leq W_{\text{opt}}.$$

An dieser Stelle können wir bereits Hoffnung schöpfen, denn zumindest für die Kante (v,w) ist das erwartete Verhalten unseres Vorgehens so gut wie der optimale fraktionale Wert! Allerdings besteht die Gefahr, dass der Erwartungswert der *maximalen* Belastung groß ist. Die Chernoff Schranke aus Lemma 1.15 weist nach, dass eine Summe unabhängiger, binärer Zufallsvariablen nur mit sehr geringer Wahrscheinlichkeit *wesentlich* größer als ihr Erwartungswert ist. Wir wenden die Chernoff Schranke an und erhalten die Abschätzung

$$\text{prob}[\, W(v,w) \geq (1+\beta) \cdot W_{\text{opt}} \,] \leq e^{-W_{\text{opt}} \cdot \beta^2/3}.$$

Wir setzen

$$\beta = \sqrt{\frac{3 \cdot \ln(n^2/\varepsilon)}{W_{\text{opt}}}}$$

und erhalten

$$\begin{aligned} \text{prob}[\, \max_{(v,w) \in E} W(v,w) \geq (1+\beta) \cdot W_{\text{opt}} \,] &\leq \sum_{(v,w) \in E} \text{prob}[\, W(v,w) \geq (1+\beta) \cdot W_{\text{opt}} \,] \\ &\leq |E| \cdot e^{-W_{\text{opt}} \cdot \beta^2/3} = |E| \cdot \frac{\varepsilon}{n^2} \leq \varepsilon. \end{aligned}$$

Wir bestimmen also Wege $P_1, \ldots, P_r$ mit Wahrscheinlichkeit mindestens $1 - \varepsilon$, so dass die maximale Belastung durch $(1+\beta) \cdot W_{\text{opt}} = W_{\text{opt}} + \beta \cdot W_{\text{opt}} = W_{\text{opt}} + \sqrt{3 \cdot W_{\text{opt}} \cdot \ln(n^2/\varepsilon)}$ beschränkt ist. Wir erhalten deshalb

Satz 9.10 *Mit Wahrscheinlichkeit mindestens* $1 - \varepsilon$ *bestimmt Algorithmus 9.9 Wege mit maximaler Belastung höchstens*

$$W_{\text{opt}} + \sqrt{3 \cdot W_{\text{opt}} \cdot \ln(n^2/\varepsilon)}.$$

Wir erhalten also eine sehr scharfe Approximation, falls W_{opt} genügend groß ist.

9.2.4 MAX-SAT

In MAX-SAT ist eine Menge K von Klauseln gegeben, wobei Klausel $k \in K$ das Gewicht $w_k \geq 0$ besitzt. Es wird eine Wahrheitsbelegung gesucht, die eine Klauselmenge von größtmöglichem Gewicht erfüllt.

Algorithmus 9.11

(1) Die Eingabe besteht aus einer Menge K von gewichteten Klauseln mit Literalen aus der Menge $\{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$. Zu Anfang sind alle Klauseln unentschieden.

(2) FOR $i = 1$ TO n DO

> Stelle fest, welche Klauseln entschieden sind, d.h. den Wert „wahr“ oder „falsch“ ergeben.
>
> Setze $x_i = 1$, wenn das Gewicht aller Klauseln, in denen x_i positiv vorkommt mindestens so groß ist wie das Gewicht aller Klauseln, in denen x_i negativ vorkommt. Ansonsten setze $x_i = 0$.

(3) Gib die Belegung aus.

Satz 9.12 *Algorithmus 9.11 ist ein effizienter 2-Approximationsalgorithmus für MAX-SAT.*

Beweis: Die Behauptung folgt, da das Gewicht erfüllter Klauseln in jeder Iteration mindestens so groß wie das Gewicht nicht-erfüllter Klauseln ist. □

Aufgabe 94
Konstruiere Instanzen, so dass Algorithmus 9.11 dem Approximationsfaktor zwei beliebig nahekommt.

Aufgabe 95
Wir modifizieren Algorithmus 9.11: In der Berechnung des Gewichts erfüllter, bzw. nicht-erfüllter Klauseln wird das Gewicht w_{x_i} und $w_{\neg x_i}$ verdoppelt. Konstruiere Instanzen mit Approximationsfaktor 3/2.
Es kann gezeigt werden, dass das neue Verfahren den Approximationsfaktor 3/2 besitzt.

Wir stellen zuerst eine randomisierte Version von Algorithmus 9.11 vor, die für lange Klauseln eine bessere Approximationsleistung erzielt.

Algorithmus 9.13 **Randomisierung für lange Klauseln**

(1) Die Eingabe besteht aus einer Menge K von Klauseln mit Literalen aus der Menge $\{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$. Zusätzlich erhält jede Klausel $k \in K$ das nicht-negative Gewicht w_k.

(2) Bestimme den Wahrheitswert einer jeden Variable x_i zufällig und gib die Belegung aus.

Wir führen für jede Klausel k die Zufallsvariable

$$W_k = \begin{cases} w_k & \text{Klausel } k \text{ wird erfüllt,} \\ 0 & \text{sonst} \end{cases}$$

ein. Dann ist

$$\mathbb{E}[W_k] = (1 - 2^{-l}) \cdot w_k,$$

wenn die Klausel aus l Variablen besteht: Denn $\mathbb{E}[W_k] = w_k \cdot \text{prob}[\ k \text{ wird erfüllt }]$ und $\text{prob}[\ k \text{ wird erfüllt }] = 1 - 2^{-l}$. Wenn alle Klauseln mindestens l Literale besitzen, dann ist das erwartete Gesamtgewicht erfüllter Klauseln also mindestens $\sum_{k \in K} \mathbb{E}[W_k] \geq (1 - 2^{-l}) \cdot \sum_{k \in K} w_k$ und wir erhalten einen trivialen $\frac{1}{1-2^{-l}}$-approximativen Algorithmus.
Interessanter ist der Fall kurzer Klauseln. Hier versuchen wir einen Ansatz mit linearer Programmierung. Für eine Klausel k sei $V^+(k) = \{i \mid x_i \text{ kommt in } k \text{ vor }\}$ (bzw. $V^-(k) = \{i \mid \neg x_i \text{ kommt in } k \text{ vor }\}$). Das 0-1 Programm hat dann die folgende Form:

$$\begin{aligned} \text{maximiere} \quad & \sum_{k \in K} w_k \cdot z_k \text{ so dass} \\ & \text{für alle Klauseln } k \in K\text{: } \sum_{i \in V^+(k)} y_i + \sum_{i \in V^-(k)} (1 - y_i) \geq z_k \\ & \text{für alle Klauseln } k \in K \text{ ist } z_k \in \{0, 1\} \text{ und für jedes } i \text{ ist } y_i \in \{0, 1\}. \end{aligned}$$

In der Relaxierung wird dementsprechend $0 \leq z_k, y_i \leq 1$ gefordert.

Algorithmus 9.14 **Randomisiertes Runden für kurze Klauseln**

(1) Die Eingabe besteht aus einer Menge K von Klauseln mit Literalen aus der Menge $\{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$. Zusätzlich erhält jede Klausel $k \in K$ das nicht-negative Gewicht w_k.

(2) Bestimme eine optimale Lösung (z^*, y^*) der Relaxierung.

(3) Bestimme den Wahrheitswert einer jeden Variable x_i zufällig, wobei die Belegung $x_i = 1$ mit Wahrscheinlichkeit y_i^* gewählt wird. Gib die Belegung aus.

Kommentar: Wir verwenden also die Methode des randomisierten Rundens.

Beispiel 9.2 Wir betrachten Klauseln mit jeweils genau zwei Literalen. Man überzeuge sich, dass die Setzung $y_i = \frac{1}{2}$ und $z_k = 1$ stets zu einer optimalen Lösung der Relaxierung führen. In diesem Spezialfall erzeugen also die Algorithmen 9.13 und 9.14 Belegungen nach derselben Verteilung.

Für die Formel $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ erhält man bei Klauselgewichten 1 das 0-1-Optimum drei, während vier das fraktionale Optimum ist. Die „Integralitätslücke" zwischen dem 0-1 Programm und seiner Relaxierung ist also mindestens $\frac{4}{3}$.

Wir bestimmen wiederum die Wahrscheinlichkeit p_k, dass Klausel k nach randomisiertem Runden erfüllt wird, bzw. den Erwartungswert $\mathbb{E}[W_k] = w_k \cdot p_k$.

Lemma 9.15 *Sei k eine Klausel mit l Literalen. Dann ist $\mathbb{E}[W_k] = (1 - 2^{-l}) \cdot w_k$ für Algorithmus 9.13 und $\mathbb{E}[W_k] \geq (1 - (1 - \frac{1}{l})^l) \cdot w_k \cdot z_k^*$ für Algorithmus 9.14.*

Beweis: Die Analyse von Algorithmus 9.13 haben wir bereits oben ausgeführt. Für die Analyse von Algorithmus 9.14 nehmen wir o.B.d.A. an, dass $k \equiv x_1 \vee \cdots \vee x_l$. Wir verwenden die Beziehung

$$\frac{\sum_{i=1}^{l} a_i}{l} \geq (\Pi_{i=1}^{l} a_i)^{1/l}$$

zwischen dem arithmetischen und dem geometrischen Mittel. (Diese Beziehung folgt nach logarithmieren, da der Logarithmus konkav[2] ist.) Also ist

$$\begin{aligned} p_k &= 1 - \Pi_{i=1}^{l}(1 - y_i^*) \geq 1 - \left(\frac{\sum_{i=1}^{l}(1 - y_i^*)}{l}\right)^l \\ &= 1 - \left(1 - \frac{\sum_{i=1}^{l} y_i^*}{l}\right)^l \geq \underbrace{1 - \left(1 - \frac{z_k^*}{l}\right)^l}_{=g_l(z_k^*)}, \end{aligned}$$

denn $\sum_{i=1}^{l} y_i \geq z_k$ wird in der Relaxierung gefordert. Jetzt beachte, dass

$$g_l(z) = 1 - (1 - \frac{z}{l})^l$$

für $z \leq 1$ eine konkave Funktion ist, denn die zweite Ableitung $-\frac{l-1}{l} \cdot (1 - \frac{z}{l})^{l-2}$ ist negativ. Für $z = 0$ wird der Wert 0 und für $z = 1$ wird der Wert $g_l(1)$ angenommen: Die Funktion liegt also oberhalb der Geraden durch $(0, 0)$ und $(1, g_l(1))$ und es ist $g_l(z) \geq z \cdot g_l(1)$. Also ist $p_k \geq z_k^* \cdot g_l(1)$ und $\mathbb{E}[W_k] = w_k \cdot p_k \geq (1 - (1 - \frac{1}{l})^l) \cdot w_k \cdot z_k^*$ wie behauptet. □

Wir kombinieren jetzt die Ansätze für kurze und lange Klauseln.

Algorithmus 9.16 **Eine $\frac{4}{3}$-Approximation für MAX-SAT**

(1) Die Eingabe besteht aus einer Menge K von Klauseln mit Literalen aus der Menge $\{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$. Zusätzlich erhält jede Klausel $k \in K$ das nicht-negative Gewicht w_k.

(2) Bestimme eine zufällige Belegung B_1 mit Algorithmus 9.13.

(3) Bestimme eine Belegung B_2 durch zufälliges Runden mit Algorithmus 9.14.

(3) Gib die beste der beiden Belegungen aus.

[2]Eine Funktion f ist konkav, wenn stets $f(\frac{\sum_{i=1}^{l} a_i}{l}) \geq \frac{1}{l} \cdot \sum_{i=1}^{l} f(a_i)$ ist.

Satz 9.17 *Algorithmus 9.16 ist ein $\frac{4}{3}$-approximativer Algorithmus für MAX-SAT.*

Beweis: Wir analysieren einen schwächeren Algorithmus, der mit Wahrscheinlichkeit $\frac{1}{2}$ einen der Algorithmen 9.13 und 9.14 auswählt. Wir bestimmen $\mathbb{E}[W_k]$ für diese Kombination und erhalten für eine Klausel mit l Literalen

$$\begin{aligned}\mathbb{E}[W_k] &\geq \frac{1}{2}\cdot(1-2^{-l})\cdot w_k + \frac{1}{2}\cdot(1-(1-\frac{1}{l})^l)\cdot w_k\cdot z_k^* \\ &\geq w_k\cdot z_k^*\cdot\frac{1-2^{-l}+1-(1-\frac{1}{l})^l}{2} \\ &\geq w_k\cdot z_k^*\cdot\frac{3}{4},\end{aligned}$$

denn $1-2^{-l}+1-(1-\frac{1}{l})^l \geq \frac{3}{2}$. Dann ist aber

$$\sum_{k\in K}\mathbb{E}[W_k] \geq \frac{3}{4}\cdot\sum_{k\in K} w_k\cdot z_k^* = \frac{3}{4}\cdot \texttt{opt}$$

für das Maximum `opt` der Relaxierung, das natürlich mindestens so groß wie das Maximum des 0-1 Programms ist. □

9.3 Die Held-Karp Schranke für TSP

Eine Instanz des metrischen Traveling Salesman Problems mit Metrik $d(,)$ und n Städten $1,\ldots,n$ sei vorgegeben. Wir erinnern an die Definition der Held-Karp Schranke:

- Ein 1-Baum B ist aus einem minimalen Spannbaum für die Städte $2,\ldots,n$ aufgebaut. Die Stadt 1 wird mit den beiden nächstliegenden Städten verbunden.
- $l(B)$ ist das Gewicht von B.
- Für die Funktion

$$f(\lambda) = \min_{B^*}\{\ l(B^*) + \sum_{i=1}^{n}\lambda_i\cdot(\mathrm{grad}_{B^*}(i)-2)\ \} \tag{9.5}$$

  ist die Held-Karp Schranke

$$\max_{\lambda\in\mathbb{R}^n} f(\lambda)$$

  zu berechnen.

Wir erinnern uns, dass $f(\lambda)$ für jeden fixierten Vektor λ einfach zu berechnen ist: Bestimme einen minimalen Spannbaum für die Kantengewichte $d(r,s)+\lambda_r+\lambda_s$ auf den Knoten $2,\ldots,n$ und füge den Knoten 1 über die beiden kürzesten Kanten ein. Die Schwierigkeit in der Berechnung der Held-Karp Schranke besteht deshalb in der Berechnung der Maximierung.

Die Held-Karp Schranke resultiert aus einer Anwendung der Lagrange Relaxierung. Um diesen Zusammenhang allgemein zu erklären, betrachten wir das Minimierungsproblem

$$\min_{x \in Q}\{c^T \cdot x \ : \ A \cdot x \geq b\}$$

für eine endliche Menge Q. In der Lagrange Relaxierung wird jeder Ungleichung aus $A \cdot x \geq b$ ein Lagrange Multiplikator zugewiesen. Die Ungleichungen werden dann so in die Zielfunktion gehoben, dass eine Verletzung bestraft wird. Man betrachtet also das neue Optimierungsproblem

$$\min_{x \in Q}\{c^T \cdot x + \lambda^T \cdot (b - A \cdot x)\}$$

für den Vektor $\lambda \geq 0$ der Lagrange Multiplikatoren.
In der Anwendung auf die Held-Karp Schranke wählen wir $c_{i,j} = d(i,j)$, Q ist die Menge aller 1-Bäume und das Ungleichungssystem $A \cdot x \geq b$ beschreibt die Bedingungen

$$\sum_{j, j \neq i} x_{\{i,j\}} = 2 \text{ für jede Stadt } i. \tag{9.6}$$

(Man denke sich die geforderte Gleichung durch zwei Ungleichungen ersetzt.) Die beabsichtigte Interpretation von $x_{\{i,j\}} = 1$ ist, dass Stadt j auf Stadt i in einer Rundreise folgt. Die Gleichungen versuchen also zu erzwingen, dass jede Stadt i der Rundreise zwei Nachbarn hat.
Beachte, dass die Lagrange Multiplikatoren für die Held-Karp Schranke tatsächlich auch negativ sein dürfen, da die Held-Karp Bedingungen (9.6) für jeden Knoten i auf die Beiträge

$$\lambda_{i,1} \cdot (2 - \sum_{j, j \neq i} x_{\{i,j\}}) + \lambda_{i,2} \cdot (\sum_{j, j \neq i} x_{\{i,j\}} - 2) = (\lambda_{i,2} - \lambda_{i,1}) \cdot (\sum_{j, j \neq i} x_{\{i,j\}} - 2)$$

führen: Die Parameter $\lambda_i = \lambda_{i,2} - \lambda_{i,1}$ können somit auch negative Werte annehmen. Wenn x der Inzidenzvektor eines 1-Baums B^* ist, dann folgt

$$\sum_i \lambda_i \cdot (\sum_{j, j \neq i} x_{\{i,j\}} - 2) = \sum_i \lambda_i \cdot (\text{grad}_{B^*}(i) - 2).$$

Also erhalten wir die Held-Karp Schranke als Ergebnis der Lagrange Relaxierung:

Beobachtung 9.1 *Für $c_{i,j} = d(i,j)$ und die Gleichungen $\sum_{j, j \neq i} x_{\{i,j\}} = 2$ für jede Stadt i ist*

$$\min_{x \in Q}\{c^T \cdot x + \lambda^T \cdot (b - A \cdot x)\} = f(\lambda).$$

Wie können wir

$$\max_{\lambda \geq 0} \min_{x \in Q}\{c^T \cdot x + \lambda^T \cdot (b - A \cdot x)\} \tag{9.7}$$

effizient approximieren?

Algorithmus 9.18 **Das Subgradienten-Verfahren**

- Beginne mit einem Vektor $\lambda^{(0)} \geq 0$.
- In Iteration i
  - bestimme einen Vektor $x^{(i)} \in Q$, der $c^T \cdot x + \lambda^{(i)} \cdot (b - A \cdot x)$ über der Menge Q minimiert und
  - setze $\lambda^{(i+1)} = \max\{0, \lambda^{(i)} + \alpha_i(b - A \cdot x^{(i)})\}$ für eine positive reelle Zahl α_i.
    Kommentar: Die Lagrange Multiplikatoren wachsen also mit wachsendem Verstoß. Sie fallen, wenn kein Verstoß vorliegt.)

Algorithmus 9.18 bietet sich besonders dann an, wenn $\min_{x \in Q}\{c^T \cdot x + \lambda^T \cdot (b - A \cdot x)$ schnell berechnet werden kann. Dies trifft zum Beispiel für die Held-Karp Schranke zu, da die Minimierung zu einem minimalen Spannbaum Problem äquivalent ist.

Fakt 9.1 *[KV05] Wenn* $\lim_i \alpha_i = 0$ *und* $\sum_{i=0}^{\infty} \alpha_i = \infty$*, dann konvergiert das Subgradienten Verfahren gegen* $\max_{\lambda \geq 0} \min_{x \in Q}\{c^T \cdot x + \lambda^T \cdot (b - A \cdot x)\}$.

Die wichtige Frage der Konvergenzgeschwindigkeit ist damit allerdings nicht geklärt. Zumindest in der Held-Karp Anwendung scheint das Verfahren aber praktikabel zu sein. Wir können die Held-Karp Schranke also gut approxinmieren, wie gut wird aber die Länge einer kürzesten Rundreise approximiert? Auch hier können wir eine allgemeine Erklärung geben. (konvex(Q) bezeichne die konvexe Hülle von Q.)

Lemma 9.19 *Wenn die Menge Q endlich ist und wenn $A \cdot x \geq b$ in Q lösbar ist, dann gilt*

$$\max_{\lambda \geq 0} \min_{x \in Q}\{c^T \cdot x + \lambda^T \cdot (b - A \cdot x)\} = \min_{y \in \text{konvex(Q)}}\{c^T \cdot y \ : \ A \cdot y \geq b\}.$$

Beweis: Wir überführen zuerst das max-min Problem auf ein lineares Maximierungsproblem

$$\begin{aligned} & \max_{\lambda \geq 0} \min_{x \in Q}\{c^T \cdot x + \lambda^T \cdot (b - A \cdot x)\} \\ = \ & \max_{\lambda \geq 0}\{ \eta \ : \ \eta \leq c^T \cdot x + \lambda^T \cdot (b - A \cdot x) \text{ für alle } x \in Q\} \\ = \ & \max_{\lambda \geq 0}\{ \eta \ : \ \eta - \lambda^T \cdot (b - A \cdot x) \leq c^T \cdot x \text{ für alle } x \in Q\}. \end{aligned}$$

Beachte, dass das lineare Programm die Variablen η und λ besitzt. Weiterhin fügt jedes x in Q eine lineare Bedingung hinzu. Jetzt wenden wir das Dualitätstheorem:

- Wir erhalten eine duale Variable α_x für die primale Bedingung zu $x \in Q$.
- Die Zielfunktion $\sum_{x \in Q} \alpha_x \cdot (c^T \cdot x)$ ist zu minimieren.
- Die zu der nicht vorzeichen-behafteten primalen Variablen η gehörende duale Bedingung ist $\alpha_x = 1$

- und die zu den nicht-negativen λ-Variablen gehörenden dualen Bedingungen sind $-\sum_{x\in Q}\alpha_x\cdot(b-A\cdot x)\geq 0$.

Also ist

$$\begin{aligned}
&= \max_{\lambda\geq 0}\{\ \eta\ :\ \eta-\lambda^T\cdot(b-A\cdot x)\leq c^T\cdot x \text{ für alle } x\in Q\}\\
&= \min_{\alpha\geq 0}\{\ \sum_{x\in Q}\alpha_x\cdot(c^T\cdot x)\ :\ \sum_{x\in Q}\alpha_x=1, \sum_{x\in Q}\alpha_x\cdot(b-A\cdot x)\leq 0\}\\
&= \min_{\alpha\geq 0}\{\ c^T\cdot\sum_{x\in Q}\alpha_x\cdot x\ :\ \sum_{x\in Q}\alpha_x=1,\ b\leq A\cdot\sum_{x\in Q}\alpha_x\cdot x\}.
\end{aligned}$$

Das letzte Optimum stimmt aber mit

$$\min_{y\in \text{konvex(Q)}}\{\ c^T\cdot y\ :\ b\leq A\cdot y\}$$

überein. □

Was bedeutet Lemma 9.19 für die Held-Karp Schranke? Wir wählen Q als die Menge aller 1-Bäume über den Knoten $1,\ldots,n$ und benutzen den folgenden Fakt.

Fakt 9.2 *[KV05] Die Menge* konvex(Q) *besteht aus allen Vektoren x (mit einer Komponente für jede Kante e des ungerichteten vollständigen Graphen mit n Knoten), so dass:*

(1) $\sum_e x_e = n$ *und* $0\leq x_e\leq 1$ *für alle Kanten e,*

(2) $\sum_j x_{\{1,j\}} = 2$ *und*

(3) $\sum_{i<j,i,j\in X} x_{\{i,j\}}\leq |X|-1$ *für jede nicht-leere Teilmenge* $X\subseteq\{2,\ldots,n\}$.

Beachte für die Bedingung (2), dass wir nur 1-Bäume zulassen, die für Knoten 1 genau zwei Nachbarn haben. Weiterhin bilden die 1-Bäume auf den Knoten $2,\ldots n$ einen Baum, besitzen also für jede Teilmenge $X\subseteq\{2,\ldots,n\}$ höchstens $|X|-1$ Kanten. Die exponentiell vielen Bedingungen in (3) heißen auch Subtour-Eliminationsbedingungen, da diese Bedingungen „Fast-Rundreisen“ verbieten, die einer Vereinigung von Teilrundreisen entsprechen.

Aufgabe 96
Betrachte das Minimierungsproblem

$$\begin{aligned}
\min_y \quad & \sum_{i<j} d(i,j)\cdot y_{\{i,j\}},\ \text{so dass } y_{\{i,j\}}\in\{0,1\} \text{ für alle } i<j \text{ und}\\
& \sum_{j,i\neq j} y_{\{i,j\}} = 2 \text{ für alle } i.
\end{aligned}$$

Zeige, dass dieses 0-1 Programm nach einer disjunkten Vereinigung von Teilrundreisen geringster Länge sucht.

Wir wissen mit Lemma 9.19, dass die Held-Karp Schranke mit $\min\sum_{i<j} d(i,j)\cdot y_{\{i,j\}}$ übereinstimmt, wobei über alle Vektoren y der konvexen Hülle aller 1-Bäume zu minimieren ist, die zusätzlich die Eigenschaften $\sum_{j,i\neq j} y_{\{i,j\}}=2$ für alle Knoten i erfüllen. Wir kennen aber mit Fakt 9.2 eine Beschreibung der konvexen Hülle und erhalten deshalb

Lemma 9.20 *Die Held-Karp Schranke stimmt überein mit*

$$\begin{aligned}\min_y \quad & \sum_{i<j} d(i,j)\cdot y_{\{i,j\}}, \text{ so dass } 0 \le y_{\{i,j\}} \le 1 \text{ für alle } i<j,\\ & \sum_{j,i\neq j} y_{\{i,j\}} = 2 \text{ für alle } i \text{ und}\\ & \sum_{i<j,i,j\in X} y_{\{i,j\}} \le |X|-1 \text{ für jede nicht-leere Teilmenge } X\subseteq\{2,\ldots,n\}.\end{aligned}$$

Beweis: Die Bedingung $\sum_e x_e = n$ aus Fakt 9.2 ist eine Konsequenz der Bedingungen $\sum_{j,i\neq j} y_{\{i,j\}} = 2$ für alle Knoten i und taucht deshalb nicht mehr auf. □

Die Held-Karp Schranke ist also im Wesentlichen das Resultat der linearen Minimierung, wenn wir fordern, dass jeder Knoten zwei (fraktionale) Nachbarn hat und wenn wir Teilrundreisen verbieten.

Offenes Problem 5
Wenn wir fordern, dass die Variablen $y_{i,j}$ 0-1-wertig sind, dann beschreibt die Held-Karp Schranke die Länge einer kürzesten Rundreise. Es ist bekannt, dass die Integralitätslücke für die Held-Karp Schranke höchstens $\frac{3}{2}$, aber mindestens $\frac{4}{3}$ beträgt.
In vielen praktisch-relevanten Anwendungen liegt die Held-Karp Schranke aber wesentlich näher am Optimum.

9.4 Branch & Cut

Viele wichtige Optimierungsprobleme lassen sich als Probleme der ganzzahligen linearen Programmierung, bzw. der 0-1 Programmierung auffassen. Dazu erinnern wir an das gewichtete VERTEX COVER Problem mit dem linearen Programm

$$\begin{aligned}\text{minimiere } \sum_{v\in V} w_v\cdot x_v, \qquad & \text{so dass } x_u + x_v \ge 1 \text{ für alle } \{u,v\}\in E\\ & \text{und } x_u \in \{0,1\} \text{ für alle } u\in V\end{aligned}$$

oder an das gewichtete INDEPENDENT SET Problem mit der Formulierung

$$\begin{aligned}\text{maximiere } \sum_{v\in V} x_v, \qquad & \text{so dass } x_u + x_v \le 1 \text{ für alle } \{u,v\}\in E\\ & \text{und } x_u \in \{0,1\} \text{ für alle } u\in V.\end{aligned} \tag{9.8}$$

Ein komplizierteres Beispiel ist die Held-Karp Schranke für das Traveling Salesman Problem mit der Formulierung

$$\begin{aligned}\min_y \quad & \sum_{i<j} d(i,j)\cdot x_{\{i,j\}}, \text{ so dass } x_{\{i,j\}} \in \{0,1\} \text{ für alle } i<j,\\ & \sum_{j,i\neq j} x_{\{i,j\}} = 2 \text{ für alle } i \text{ und}\\ & \sum_{i<j,i,j\in X} x_{\{i,j\}} \le |X|-1 \text{ für jede nicht-leere Teilmenge } X\subseteq\{2,\ldots,n\}.\end{aligned} \tag{9.9}$$

Damit ist der Entwurf von Approximationsalgorithmen für die ganzzahlige lineare Programmierung oder die 0-1 Programmierung besonders wichtig. Aber gerade wegen der Ausdrucksstärke der ganzzahligen linearen Programmierung sind effiziente Algorithmen mit annehmbaren Approximationsfaktoren im Allgemeinen ausgeschlossen.[3] Nichtsdestotrotz sind einige wichtige Teilklassen der ganzzahligen linearen Programmierung effizient und scharf approximierbar. Wir betrachten zuerst den allgemeinen Fall

$$\min c^T \cdot x \text{ so dass } x \in \mathbb{Z}^n, Ax \geq b$$

der ganzzahligen Programmierung, nehmen aber an, dass alle Einträge der Matrix A ganzzahlig sind. Zuerst lösen wir das relaxierte Programm

$$\min c^T \cdot x \text{ so dass } x \in \mathbb{R}^n, Ax \geq b$$

und erhalten eine optimale Lösung x. Wenn alle Komponenten von x ganzzahlig sind, dann ist x auch eine optimale Lösung des ganzzahligen Programms. Was aber tun, wenn x fraktionale Komponenten hat?

Definition 9.21 Eine Schnittebene ist eine zusätzliche lineare Ungleichung, die von allen integralen Punkten des ganzzahligen linearen Programms erfüllt wird, aber nicht von x.
Ein Gomory-Chvatal Schnitt für das Ungleichungssystem $Ax \geq b$ ist eine lineare Ungleichung $y^T Ax \geq \lceil y^T b \rceil$. Der Vektor y muss aus nicht-negativen Komponenten bestehen und $y^T A$ darf nur ganzzahlige Komponenten besitzen.

Beachte, dass alle ganzzahligen Lösungen des Systems $Ax \geq b$ auch den Gomory-Chvatal Schnitt $y^T Ax \geq \lceil y^T b \rceil$ erfüllen: Ein Gomory-Chvatal Schnitt bewahrt also alle integralen Lösungen des Systems $Ax \geq b$, schneidet aber hoffentlich viele ungewünschte fraktionale Lösungen heraus. Schnittebenen werden solange eingeführt bis ein ganzzahliges Optimum berechnet wird.

Beispiel 9.3 Wir betrachten das INDEPENDENT SET Problem. Da es sich hier um ein Maximierungsproblem handelt, sind Gomory-Chvatal Schnitte mit unterer statt oberer Gauß-Klammer zu definieren.
Für den vollständigen Graphen V_n mit n Knoten ist das integrale Optimum von (9.8) offensichtlich Eins. Wenn wir aber relaxieren, dann springt das fraktionale Optimum auf $n/2$. Wir zeigen jetzt, dass man mit relativ wenigen Gomory-Chvatal Schnitten alle unerwünschten fraktionalen Lösungen herausschneiden kann.
Wir nehmen induktiv an, dass wir bereits Schnitte hinzugefügt haben, so dass wir die Ungleichung $\sum_{i=1}^{n-1} x_i \leq 1$ „hergeleitet“ haben: Die Ungleichung befindet sich also unter den bisherigen Schnitten. Unser Ziel ist die Herleitung der Ungleichung $\sum_{i=1}^{n} x_i \leq 1$.
Die Ungleichungen $x_i + x_n \leq 1$ sind für $1 \leq i \leq n-1$ Teil des ursprünglichen Ungleichungssystems $Ax \leq b$. Wir summieren alle Ungleichungen auf und erhalten die Ungleichung

[3] Das allgemeine Traveling Salesman Problem gehört nicht zur Klasse $poly(n) - \mathcal{APX}$, ist also mit annehmbaren Approximationsfaktoren nicht effizient approximierbar. Dann ist aber auch das 0-1 Programm (9.9) mit annehmbaren Approximationsfaktoren nicht effizient approximierbar.

$\sum_{i=1}^{n-1} x_i + (n-1) \cdot x_n \leq n-1$. Zu dieser Ungleichung addieren wir das $(n-2)$-fache der Ungleichung $\sum_{i=1}^{n-1} x_i \leq 1$ und erhalten deshalb

$$(n-2) \cdot \sum_{i=1}^{n-1} x_i + \sum_{i=1}^{n-1} x_i + (n-1) \cdot x_n = (n-1) \cdot \sum_{i=1}^{n} x_i \leq n-1+n-2.$$

Wir teilen beide Seiten durch $n-1$ und erhalten den Gomory-Chvatal Schnitt $\sum_{i=1}^{n} x_i \leq 1$.

Branch & Cut Verfahren arbeiten weiterhin nach dem Schema des Branch & Bound Algorithmus 6.1, versuchen aber verbesserte untere Schranken durch die Hinzunahme neuer Schnittebenen zu erreichen.

Algorithmus 9.22 **Branch & Cut**
Das ganzzahlige Programm

$$\min c^T \cdot x \text{ so dass } x \in \mathbb{Z}^n, A \cdot x \geq b$$

sei zu lösen. Der Branching Operator B sei gegeben. Anfänglich besteht der Branch & Cut Baum $\mathcal{B}$ nur aus der (aktivierten) Wurzel.

(1) Eine Anfangslösung wird durch eine Heuristik berechnet.

(2) Wiederhole, solange es aktivierte Blätter gibt:

  (2a) Wähle das **erfolgversprechendste** aktivierte Blatt v von $\mathcal{B}$.

  (2b) Führe Cut-Schritte durch: Füge sorgfältig ausgewählte Schnittebenen als neue Bedingungen hinzu.

  (2c) Rufe das Branch & Bound Verfahren auf.

  Kommentar: Beachte, dass wir das Optimum des relaxierten Programm als untere Schranke wählen können. Durch die Aufnahme neuer Schnittebenen in Schritt (2b) wird sich die untere Schranke stetig verbessern.

(3) Gib die beste gefundene integrale Lösung aus.

Die Auswahl guter Schnittebenen ist das zentrale Problem im Entwurf eines Branch & Cut Verfahrens. Wir betrachten dazu exemplarisch das Traveling Salesman Problem, das wir mit dem ganzzahligen Programm (9.9) formulieren. Wir relaxieren (9.9), nachdem wir die viel zu vielen Subtour-Eliminationsbedingungen entfernt haben. Dann berechnen wir eine optimale Lösung x.

1. Wenn x einer Rundreise entspricht, dann haben wir eine optimale Rundreise gefunden.

2. Wenn x eine Subtour-Eliminationsbedingung verletzt, dann fügen wir diese Bedingung hinzu, lösen das um die verletzte Bedingung erweiterte relaxierte Programm und wiederholen unser Vorgehen für die neue optimale Lösung x.

Was aber tun, wenn x keiner Rundreise entspricht, aber sehr wohl alle Subtour-Eliminationsbedingungen erfüllt? Wir sollten zwar sehr gute untere Schranken erhalten, denn die Held-Karp Schranke kann jetzt exakt berechnet werden. Aber wir benötigen neue Bedingungen:

Aufgabe 97
Sei $X \subseteq \{1, \ldots, n\}$ eine beliebige Teilmenge und F sei eine Menge von Kanten, die mit einem Knoten aus X inzident sind. F habe ungerade Mächtigkeit. Dann erfüllt jede Rundreise die Ungleichung

$$\sum_{i<j, i,j \in X} x_{\{i,j\}} + \sum_{e \in F} x_e \leq |X| - 1 + \frac{|F|+1}{2}.$$

Aufgabe 98
Um die Kamm-Ungleichungen einzuführen, benötigen wir paarweise disjunkte Mengen $T_1, \ldots, T_r \subseteq \{1, \ldots, n\}$ für eine ungerade Zahl $r \geq 3$. Desweiteren sei $H \subseteq \{1, \ldots, n\}$ eine Teilmenge, die mit jeder Menge T_i mindestens ein Element gemeinsam hat, aber keine dieser Mengen enthält. Dann erfüllt jede Rundreise die Ungleichung

$$\sum_{e \in \delta(H)} x_e + \sum_{i=1}^{r} \sum_{e \in \delta(T_i)} x_e \geq 3 \cdot r + 1.$$

$\delta(X)$ besteht aus allen Kanten mit mindestens einem Endpunkt in der Menge X.

Mit Hilfe dieser Ungleichungen und den Clique-Baum Ungleichungen [KV05] können metrische Traveling Salesman Probleme mit mehreren Tausend Städten exakt gelöst werden.

Teil III

Schwierige Optimierungsprobleme

Wie weit kann der Approximationsfaktor effizienter Algorithmen für ein vorgegebenes Optimierungsproblem verbessert werden? Wann besitzt das Optimierungsproblem keine effizienten Approximationsalgorithmen mit scharfen Approximationen?
Wir haben in Kapitel 1.1 einige wichtige Komplexitätsklassen eingeführt. Wir wissen jetzt zum Beispiel, dass

- das RUCKSACK-Problem zur Klasse $\mathcal{FPTAS}$ gehört
- und dass BIN PACKING, MINIMUM MAKESPAN SCHEDULING und das euklidsche TSP zur Klasse $\mathcal{PTAS}$, nicht aber zur Klasse $\mathcal{FPTAS}$ gehören.

Wir haben mit VERTEX COVER bereits ein erstes Optimierungsproblem in der für die Praxis sehr wichtigen Komplexitätsklasse $\mathcal{APX}$ kennengelernt. Weitere Beispiele für Optimierungsprobleme in $\mathcal{APX}$ sind

- FEEDBACK VERTEX SET für ungerichtete Graphen,
- MAX-SAT,
- Scheduling für nicht-identische Maschinen,
- FACILITY LOCATION,
- das k-CENTER Clustering Problem,
- das metrische TSP
- oder SHORTEST COMMON SUPERSTRING.

Wir haben mit SET COVER ein erstes Problem außerhalb von $\mathcal{APX}$ kennengelernt, da SET COVER wahrscheinlich keine effizienten $(1-\varepsilon)\cdot\log_2 n$-approximativen Algorithmen zulässt. Wir werden in diesem Kapitel sehen, dass CLIQUE und sein Zwilling, INDEPENDENT SET, in einer ganz anderen Schadensklasse liegen: Beide Probleme besitzen keine effizienten $n^{1-\varepsilon}$-approximativen Algorithmen!
Der Aspekt der Optimierung zerschlägt somit die schöne Struktur, die wir von der Klasse $\mathcal{NP}$ gewöhnt sind: Während die $\mathcal{NP}$-vollständigen Sprachen paarweise „äquivalent“ sind, zerfallen Optimierungsprobleme in $\mathcal{NPO}$ in verschiedenste Äquivalenzklassen. Wie können wir negative Resultate über die effiziente Approximierbarkeit eines Optimierungsproblem P erhalten? Wir weisen dazu P ein „Promise-Problem“ zu.

Definition 9.23 Sei P ein Optimierungsproblem und $\mathtt{opt}_P(x)$ sei der optimale Wert für Instanz x. Für Funktionen $f, g : \Sigma^* \to \mathbb{R}$ mit $f(x) < g(x)$ für alle $x \in \Sigma^*$ definieren wir

$$\begin{aligned} \mathrm{YES}_P &= \{x \mid \mathtt{opt}_P(x) \geq g(x)\} \\ \mathrm{NO}_P &= \{x \mid \mathtt{opt}_P(x) < f(x)\}. \end{aligned}$$

Wir sagen, dass ein Algorithmus A das Promise Problem $[f, g]$-gap-P löst, wenn A für jedes $x \in \mathrm{YES}_P$ die Ausgabe “ja” und für jede Instanz $x \in \mathrm{NO}_P$ die Ausgabe “nein” ausgibt.

Beachte, dass YES_P und NO_P disjunkte Mengen sind. Von keiner der beiden Mengen erfasst wird die „Lücke“, also alle Instanzen x mit

$$f(x) \leq \texttt{opt}_P(x) < g(x).$$

Wenn wir einem Algorithmus „versprechen“, dass eine Instanz nicht in diesen Graubereich fällt, dann löst dieser Algorithmus das Problem $[f, g]$-gap-P genau dann, wenn die richtige Zuordnung zu den Mengen YES_P und NO_P bestimmt wird.

Definition 9.24 Sei P ein Maximierungsproblem. Dann ist $[f, g]$-gap-P ein $\mathcal{NP}$-hartes Problem, wenn es für jede Sprache $L \in \mathcal{NP}$ eine effizient berechenbare Transformation T gibt, die für jede Eingabe w von L eine Instanz $T(w)$ des Optimierungsproblems P bestimmt. Es muss gelten

- Für $w \in L$ ist $T(w) \in \text{YES}_P$ (also $\texttt{opt}_P(T(w)) \geq g(T(w))$) und
- für $w \notin L$ ist $T(w) \in \text{NO}_P$ (also $\texttt{opt}_P(T(w)) < f(T(w))$).

Ist P ein Minimierungsproblem, dann müssen Eingaben $w \in L$ auf Instanzen aus NO_P und Eingaben $w \notin L$ auf Instanzen aus YES_P reduziert werden.

Was besagt die $\mathcal{NP}$-Härte von $[f, g]$-gap-P? Für jede Sprache $L \in \mathcal{NP}$ kann durch eine „**lückenschaffende Reduktion**“ auf das Optimierungsproblem P reduziert werden.

Aufgabe 99
Angenommen, es ist $\mathcal{P} \neq \mathcal{NP}$.
Zeige: Wenn $[f, g]$-gap-P ein $\mathcal{NP}$-hartes Problem ist, dann kann $[f, g]$-gap-P nicht von effizienten Algorithmen gelöst werden.

Satz 9.25 *Sei P ein Maximierungsproblem. Wir nehmen an, dass die Funktionen f und g in polynomieller Zeit berechnet werden können und dass*

$$\frac{g(x)}{f(x)} \geq \alpha(|x|)$$

für alle x gilt. Wenn $[f, g]$-gap-P ein $\mathcal{NP}$-hartes Problem ist, dann besitzt P keine effizienten Approximationsalgorithmen mit Faktor höchstens $\alpha(n)$, solange $\mathcal{P} \neq \mathcal{NP}$ gilt.

Beweis: Wir nehmen an, dass P ein Maxinmierungsproblem ist und dass der effiziente Approximationsalgorithmus A mit Faktor höchstens $g(x)/f(x) \geq \alpha(|x|)$ approximiert. Wir zeigen jetzt im Gegensatz zur Annahme, dass $[f, g]$-gap-P doch effizient gelöst werden kann. Hier ist unsere effiziente Lösung:

(1) Setze A auf Instanz x an.

(2) Wenn A die von A berechnete Lösung einen Funktionswert mindestens $f(x)$ berechnet, dann gib “ja” und ansonsten “nein” aus.

Angenommen, wir geben die Antwort "ja". Dann wissen wir, dass $\texttt{opt}_P(x) \geq f(x)$ und unsere Antwort ist richtig. Geben wir andererseits die Antwort "nein", dann ist

$$\texttt{opt}(x) < \alpha(|x|) \cdot f(x) \leq \frac{g(x)}{f(x)} \cdot f(x) = g(|x|),$$

denn A ist $\alpha(|x|)$-approximativ. Also ist auch diesmal die Antwort richtig. □

Satz 9.25 reduziert die Frage nach der effizienten Approximierbarkeit eines Optimierungsproblems P auf die Frage, für welche Funktionen das Gap-Problem $[f, g]$-gap-P ein $\mathcal{NP}$-hartes Problem ist. Angenommen wir wissen, dass das Gap-Problem für P hart ist, wie zeigt man die Härte des Gap-Problems für ein anderes Optimierungsproblem Q?

Definition 9.26 P_1 und P_2 seien zwei Optimierungsprobleme.
Wir sagen, dass $[f_1, g_1]$-gap-P_1 **lückenerhaltend** auf $[f_2, g_2]$-gap-P_2 reduziert werden kann, wenn es eine effizient berechenbare Transformation T gibt, so dass für alle Instanzen x von P_1 gilt:

- Wenn $\texttt{opt}_{P_1}(x) \geq g_1(x)$, dann $\texttt{opt}_{P_2}(x) \geq g_2(x)$.
- Wenn $\texttt{opt}_{P_1}(x) < f_1(x)$, dann $\texttt{opt}_{P_2}(x) < f_2(x)$.

Warum betrachten wir lückenerhaltende Reduktionen?

Lemma 9.27 *Es gelte $\mathcal{P} \neq \mathcal{NP}$.*
Wenn $[f_1, g_1]$-gap-P_1 lückenerhaltend auf $[f_2, g_2]$-gap-P_2 reduziert werden kann und wenn $[f_1, g_1]$-gap-P_1 ein $\mathcal{NP}$-hartes Problem ist, dann

(a) ist $[f_2, g_2]$-gap-P_1 ein $\mathcal{NP}$-hartes Problem und

(b) es gibt keine effizienten $\alpha(n)$-approximativen Algorithmen, wenn $\alpha(|x|) \leq \frac{g_2(x)}{f_2(x)}$ für alle x gilt.

Mit einem ersten $\mathcal{NP}$-harten Gap-Problem können wir also eine Lawine weiterer Nicht-Approximierbarkeitsergebnisse auslösen. Wie aber erhalten wir ein erstes $\mathcal{NP}$-hartes Gap-Problem? Die Komplexität eines Gap-Problems ist schwierig zu analysieren, da wir keine Klassifizierung „in der Lücke" verlangen, das Klassifizierungsproblem könnte durch die Herausnahme der Lücke zwischen $f(x)$ und $g(x)$ potentiell sehr viel einfacher geworden sein.
Wir werden deshalb mit der Methode „probabilistisch überprüfbarer Beweise" eine gänzlich neue Sichtweise der Klasse $\mathcal{NP}$ einführen: Statt $\mathcal{NP}$ als die Klasse aller Probleme mit kurzen Beweisen aufzufassen, werden wir zeigen, dass $\mathcal{NP}$ die Klasse aller Probleme ist, die nicht zu lange Beweise besitzen, die zudem probabilistisch, durch die Inspektion nur weniger Bits (!) überprüft werden können.
Bevor wir uns dieser zentralen Fragestellung zuwenden, führen wir das Konzept „approximationsbewahrender" Reduktionen. Während lückenerhaltende Reduktionen nur sicherstellen, dass Lücken nicht zu klein werden, zeigt eine approximationsbewahrende Reduktion, dass das Approximationsverhalten überall ähnlich ist.

Definition 9.28 $P_1 = (\text{opt}_1, f_1, L_1)$ und $P_2 = (\text{opt}_2, f_2, L_2)$ seien zwei $\mathcal{NPO}$ Optimierungsprobleme. $\texttt{opt}_{P_i}(x)$ ist der optimale Wert für Instanz x im Optimierungsproblem P_i. Wir sagen, dass P_1 auf P_2 AP-reduzierbar ist ($P_1 \leq_{AP} P_2$), falls es in polynomieller Zeit berechenbare Funktionen INSTANZ und LÖSUNG und ein Polynom p gibt, so dass die folgenden Eigenschaften für jede Instanz x_1 von P_1 erfüllt sind:

(1) $x_2 = \text{INSTANZ}(x_1)$ ist eine Instanz von P_2 und es ist $\texttt{opt}_{P_2}(x_2) \leq p(|x_1|) \cdot \texttt{opt}_{P_1}(x_1)$.

Kommentar: INSTANZ bildet also P_1-Instanzen auf P_2-Instanzen ab, wobei der optimale Wert für die P_2-Instanz höchstens polynomiell größer als der optimale Wert für die P_1-Instanz ist.

(2) Für jede Lösung y_2 mit $L_2(x_2, y_2)$ ist $y_1 = \text{LÖSUNG}(x_1, y_2)$ eine Lösung zur Instanz x_1, d.h. das Prädikat $L_1(x_1, y_1)$ ist wahr. Darüberhinaus fordern wir, dass LÖSUNG approximationsbewahrend ist, d.h. es gibt eine Konstante $\alpha > 0$ mit

$$\max\left\{\frac{\texttt{opt}_{P_1}(x_1)}{f_1(x_1,y_1)}, \frac{f_1(x_1,y_1)}{\texttt{opt}_{P_1}(x_1)}\right\} - 1 \leq \alpha \cdot \left(\max\left\{\frac{\texttt{opt}_{P_2}(x_2)}{f_2(x_2,y_2)}, \frac{f_2(x_2,y_2)}{\texttt{opt}_{P_2}(x_2)}\right\} - 1\right).$$

Kommentar: LÖSUNG bildet also Lösungen zur Instanz x_2 auf Lösungen zur Instanz x_1 approximationsbewahrend ab.

AP steht abkürzend für approximationsbewahrend (approximation preserving).

Aufgabe 100
Zeige: VERTEX COVER $\leq_{AP}$ SET COVER.

Aufgabe 101
Der ungerichtete Graph $G = (V, E)$ sei vorgegeben. Eine Knotenmenge $W \subseteq V$ heißt *dominierend*, wenn jeder Knoten in $V \setminus W$ mit einem Knoten in W benachbart ist. In DOMINATING SET ist eine domierende Knotenmenge kleinster Größe zu bestimmen. Wir betrachten die *AP*-Reduktionen zwischen DOMINATING SET und dem ungewichteten SET COVER Problem.

(a) **Zeige**, dass

$$\text{Dominating Set} \leq_{AP} \text{SET COVER}.$$

(b) Sei $|V| = n$. **Konstruiere** einen effizienten, $O(\log n)$-approximativen Algorithmus für Dominating Set.

(c) **Zeige**, dass

$$\text{Set Cover} \leq_{AP} \text{Dominating Set}.$$

Hinweis: Konstruiere eine Instanz von Dominating Set, so dass die Knotenmenge aus drei Klassen besteht. Die Knoten der ersten Klasse entsprechen den Elementen des Universums und die Knoten der zweiten Klasse entsprechen den Mengen von Set Cover. Die Knoten der dritten Klasse sind sorgfältig zu bestimmen.

Satz 9.29 *Es gelte $P_1 \leq_{AP} P_2$ mit Konstante α.*

(a) Wenn P_2 einen effizienten, $(1+\varepsilon)$-approximativen Algorithmus besitzt, dann besitzt P_1 einen effizienten $(1 + \alpha \cdot \varepsilon)$-approximativen Algorithmus.

(b) Wenn $P_2 \in \mathcal{APX}$, dann ist auch $P_1 \in \mathcal{APX}$.

(c) Wenn $P_2 \in \mathcal{PTAS}$, dann ist auch $P_1 \in \mathcal{PTAS}$.

Beweis (a): Sei A_2 ein effizienter $(1+\varepsilon)$-approximativer Algorithmus für P_2. Wir entwerfen einen effizienten $(1 + \beta \cdot \varepsilon)$-approximativen Algorithmus A_1 für P_1, der als Eingabe eine Instanz x_1 von P_1 erhält:

(1) Berechne die transformierte Instanz $x_2 = \text{INSTANZ}(x_1)$ von P_2 in polynomieller Zeit.

(2) Berechne eine $(1 + \varepsilon)$-approximative Lösung y_2 mit Hilfe des effizienten Approximationsalgorithmus A_2.

(3) Führe die effiziente Rücktransformation $y_1 = \text{LÖSUNG}(x_1, y_2)$ aus.

y_2 ist eine $(1 + \varepsilon)$-approximative Lösung für die Instanz x_2 von P_2, und deshalb ist y_1 eine $(1 + \beta \cdot \varepsilon)$-approximative Lösung für die Instanz x_1 von P_1.

(b) und **(c)** folgen direkt aus (a). □

Kapitel 10

Probabilistisch überprüfbare Beweise

Für „wirklich schwierige“ Optimierungsprobleme wie etwa dem Problem der 0-1 Programmierung oder dem allgemeinen Traveling Salesman Problem lässt sich mit einfachen Mitteln zeigen, dass eine effiziente und gute Approximation ausgeschlossen ist. In vielen anderen Fällen hingegen, wie etwa für MAX-SAT, gelingt die Bestimmung der Komplexität des Optimierungsproblems nur durch den Umweg über das $\mathcal{PCP}$-Theorem, das wir jetzt beschreiben und teilweise verifizieren. Probabilistisch überprüfbare Beweise (probabilistically checkable proofs) werden eine neue Sichtweise von $\mathcal{NP}$-Beweisen erlauben. Mit Hilfe dieser neuen Sichtweise bestimmen wir im nächsten Abschnitt die Komplexität von wichtigen Approximationsproblemen.
Ein $\mathcal{NP}$-Beweis besteht aus einem Beweis $b \in \{0,1\}^*$ für eine Eingabe $w \in \{0,1\}^*$. Dieser Beweis ist in polynomieller Zeit in der Länge der Eingabe (nichtdeterministisch) zu raten und anschließend (deterministisch) zu verifizieren. Wir können diesen Prozess durch ein Spiel modellieren, an dem ein Prover und ein Verifier teilnimmt: Der Prover präsentiert einen (möglicherweise falschen) Beweis, der vom Verifier in polynomieller Zeit deterministisch zu verifizieren ist. Wir betrachten jetzt ein ähnliches Spiel, aber erlauben, dass der Verifier Zugang zu Zufallsbits erhält und eine nur hochwahrscheinlich richtige Antwort geben muss.

Definition 10.1 $r, q : \mathbb{N} \to \mathbb{N}$ seien vorgegebene Funktionen.
(a) Ein (r,q)-Verifier V ist ein deterministischer Algorithmus, der auf eine Eingabe w und $r(|w|)$ Zufallsbits zugreifen kann. V erhält auch die Möglichkeit auf bis zu $q(|w|)$ Beweisbits zuzugreifen.

> Für eine Eingabe w und eine Zufallsfolge σ berechnet V eine Folge von höchstens $q(|w|)$ Bitpositionen und erfragt die Beweisbits an diesen Positionen vom Prover. Die nachfolgende Rechnung hängt nur von w, σ und den erhaltenen Beweisbits ab. Insgesamt muss V in polynomieller Zeit in n rechnen.

(b) Wir sagen, dass eine Sprache L zur Komplexitätsklasse $\mathcal{PCP}(r,q)$ gehört, wenn es einen (r,q)-Verifier V und eine Konstante α $(0 < \alpha < 1)$ mit den folgenden Eigenschaften gibt:

(a) *Vollständigkeit*: Wenn die Eingabe w zur Sprache L gehört, dann gibt es einen Beweis b, so daß V stets akzeptiert, wenn b auf dem Orakelband gespeichert ist.

Wir sagen auch, dass V die Vollständigkeit 1 besitzt.

(b) *Soundness*: Wenn die Eingabe w nicht zur Sprache L gehört, dann wird V für jeden Beweis b mit Wahrscheinlichkeit höchstens α akzeptieren.

Wir sagen auch, dass V die Soundness α besitzt.

Kommentar: Wir fordern also, dass der Beweis lokal überprüfbar ist.

In der mathematischen Logik bezeichnet man die Eigenschaft, dass alle wahren Aussagen ableitbar sind, als „Vollständigkeit“ und die Eigenschaft, dass keine falschen Aussagen ableitbar sind, als „Soundness“.

Aufgabe 102
Betrachte eine erweiterte Definition, bei der ein (adaptiver) Verifier seine Fragen nacheinander stellt, und so die i-te Frage von der Antwort auf die $(i-1)$.te Frage abhängen lassen darf. $\mathcal{PCP}^*(r,q)$ sei wie zuvor, jetzt aber mit adaptivem Verifier definiert.
Zeige: $\mathcal{PCP}(O(r),O(q)) = \mathcal{PCP}^*(O(r),O(q))$.

Aufgabe 103
$NTIME(t(n))$ sei die Klasse aller Sprachen, die von einer nichtdeterministischen Turingmaschine in Zeit $O(t)$ akzeptiert werden. Zeige, dass $\mathcal{PCP}(r,q) \subseteq NTIME(q \cdot 2^r \cdot poly(n))$ gilt.

Aufgabe 104
Angenommen, die Sprache $L \in \mathcal{NP}$ besitzt einen (r,q)-Verifier V mit Soundness $\alpha < 1$.
Zeige, dass es dann einen $(c \cdot r, c \cdot q)$-Verifier V_c mit Soundness α^c gibt.

Wie verhalten sich die $\mathcal{PCP}$-Klassen zum „Rest der Welt“? Wenn wir weder Zufallsbits noch Beweisbits zulassen, dann erhalten wir offensichtlich deterministische Berechnungen und somit ist $\mathcal{PCP}(0,0) = \mathcal{P}$. Lassen wir polynomiell viele Zufallsbits, aber keine Beweisbits, zu, dann erhalten wir effiziente probabilistische Berechnungen, die für Worte der Sprachen keinen Fehler machen. Erlauben wir jetzt neben polynomiell vielen Zufallsbits noch konstant viele Beweisbits, dann explodiert die Berechnungskraft, und wir erhalten die Klasse aller Sprachen, die mit nichtdeterministischen Algorithmen in exponentieller Zeit erkannt werden können.[1]

Beispiel 10.1 Wir zeigen, dass die Nicht-Isomorphie[2] von Graphen zur Komplexitätsklasse $\mathcal{PCP}(\text{poly}(n),1)$ gehört: Wir möchten also überprüfen, ob zwei Graphen G_1, G_2 nicht-isomorph sind und fordern dazu polynomiell viele Zufallsbits und ein Beweisbit an.
V wählt zufällig einen der beiden Graphen, also den Graphen G_b, aus und permutiert G_b mit einer zufälligen Permutation π um den Graphen H zu erhalten. Der Verifier erwartet einen Beweis, der für Anfrage H (bzw in der H entsprechenden Position des Beweises) das Beweisbit b besitzt. Erhält V die Antwort b, dann wird der Beweis akzeptiert und sonst verworfen.

[1] Den komplexen Beweis dieser Aussage lassen wir aus.

[2] Die Komplexität des Nicht-Isomorphie Problems für Graphen ist bis heute ungeklärt. Man weiss, dass Nicht-Isomorphie von Graphen mit beschränktem Grad effizient überprüft werden kann.

Wenn tatsächlich $G_1 \not\equiv G_2$ gilt, dann gibt es einen Beweis, den V mit Wahrscheinlichkeit 1 akzeptiert. Ist hingegen $G_1 \equiv G_2$ dann wird V nur mit Wahrscheinlichkeit $1/2$ akzeptieren.

Gehen wir zu dem anderen Extrem und lassen polynomiell viele Beweisbits, aber keine Zufallsbits zu, dann ergibt sich offensichtlich die Klasse $\mathcal{PCP}(0, \text{poly}(n)) = \mathcal{NP}$.
Beachte, dass wir einen $(O(\log_2 n), \text{poly}(n))$-Verifier V durch einen deterministischen Verifier V^* simulieren können, wobei V^* alle Zufallsfolgen σ systematisch aufzählt. Deshalb folgt insbesondere $\mathcal{PCP}(O(\log_2 n), \text{poly}(n)) = \mathcal{PCP}(0, \text{poly}(n)) = \mathcal{NP}$. Offensichtlich müssen für $\mathcal{NP}$-vollständige Sprachen mindestens konstant viele Beweisbits inspiziert werden. Ist dies auch hinreichend, wenn wir auf logarithmisch viele Zufallsbits zugreifen können?

Satz 10.2 Das $\mathcal{PCP}$-Theorem
Es gilt $\mathcal{PCP}(O(\log_2 n), O(1)) = \mathcal{NP}$.

Aufgabe 105
Folgere aus dem PCP Theorem: Es gibt eine Konstante K, so dass $\mathcal{NP} \subseteq \mathcal{PCP}(O(\log_2 n), K)$. Es gibt also eine Zahl K, so dass es für jede Sprache $L \in \mathcal{NP}$ einen Verifier gibt, der höchstens K Beweisbits nachfragt.

Einen ersten Schritt im Beweis des $\mathcal{PCP}$-Theorems führen wir in Kapitel 11. Dort zeigen wir die Inklusion $\mathcal{NP} \subseteq \mathcal{PCP}(\text{poly}(n), O(1))$.
Die Beziehung $\mathcal{PCP}(O(\log_2 n), O(1)) \subseteq \mathcal{NP}$ ist offensichtlich, denn es ist

$$\mathcal{PCP}(O(\log_2 n), O(1)) \subseteq \mathcal{PCP}(O(\log_2 n), \text{poly}(n)) = \mathcal{NP}.$$

Die ungemein überraschende Aussage von Satz 10.2 ist vielmehr die Inklusion

$$\mathcal{NP} \subseteq \mathcal{PCP}(O(\log_2 n), O(1)).$$

Zum Beispiel garantiert das $\mathcal{PCP}$-Theorem, dass es für eine erfüllbare 3KNF Formel ϕ einen polynomiell langen Beweis der Erfüllbarkeit gibt, so dass die Inspektion von *konstant vielen* Beweisbits bereits hinreichend überzeugend ist. Ein konventioneller Beweis, der zum Beispiel aus einer erfüllenden Belegung besteht, ist unzureichend: Bestenfalls kann die Richtigkeit von konstant vielen Klauseln überprüft werden, aber dies wird im Allgemeinen nicht zum Verwerfen nicht-erfüllbarer Formeln mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ führen. Wenn wir allerdings a priori wissen, dass entweder alle Klauseln simultan erfüllbar sind oder dass nur ein relativ kleiner Bruchteil erfüllbar ist, dann genügt es, einige wenige, zufällig gewählte Klauseln auf ihre Richtigkeit hin zu überprüfen. Natürlich ist diese a priori Annahme nicht zulässig, aber möglicherweise lassen sich Sprachen aus $\mathcal{NP}$ auf diese Art und Weise „robust“ kodieren....
In der *New York Times* wurde das $\mathcal{PCP}$-Theorem dahin gehend interpretiert, dass mathematische Beweise umgeschrieben werden können –ohne ihre Länge mehr als poynomiell zu vergrößern–, so dass ein probabilistisch arbeitender Leser nach der Abfrage nur weniger Beweisbits die Richtigkeit des Beweises überprüfen kann. Diese Interpretation ist für solche Axiomensysteme richtig, für die die Sprache

$$\{\ (\alpha, 1^n) \mid \text{Die Aussage } \alpha \text{ besitzt einen Beweis der Länge höchstens } n\ \}$$

zur Klasse $\mathcal{NP}$ gehört, denn das $\mathcal{PCP}$-Theorem garantiert dann die schnelle probabilistische Verifikation. Leider wird das $\mathcal{PCP}$-Theorem aber nicht den Vorlesungsbetrieb revolutionieren, da ein Beweis natürlich nicht nur die Funktion der Verifizierbarkeit, sondern vorrangig die Funktion der Erklärung haben muss.

10.1 $\mathcal{PCP}$ und Approximierbarkeit

Warum ist das $\mathcal{PCP}$-Theorem von zentraler Bedeutung für die Approximation? Weil der Nachweis der Nicht-Approximierbarkeit über die „lückenschaffenden Reduktionen“ von Definition 9.24 äquivalent zum $\mathcal{PCP}$-Theorem ist:

Satz 10.3 *Die beiden folgenden Aussagen sind äquivalent:*

(a) Für eine KNF-Formel x sei $g(x)$ die Anzahl der Klauseln von x. Dann gibt es eine Konstante $0 < \alpha < 1$, so dass $[\alpha \cdot g, g]$-gap-MAX-3SAT ein $\mathcal{NP}$-hartes Problem ist.

Kommentar*: Wir haben damit für jede Sprache $L \in \mathcal{NP}$ eine lückenschaffende Reduktion von L auf MAX-3SAT erhalten.*

(b) $\mathcal{PCP}(O(\log_2 n), O(1)) = \mathcal{NP}$.

Wir haben damit ein erstes Zwischenziel, nämlich die Bestimmung der Approximationskomplexität von MAX-3SAT und damit von MAX-SAT erreicht. Warum? Mit dem $\mathcal{PCP}$-Theorem wissen wir, dass $\mathcal{PCP}(O(\log_2 n), O(1)) = \mathcal{NP}$ gilt. Deshalb können wir schließen, dass $[\alpha \cdot g, g]$-gap-MAX-3SAT ein $\mathcal{NP}$-hartes Problem ist. Wir wenden Satz 9.25 an und erhalten, dass es keine effizienten $1/\alpha$-approximativen Algorithmen für MAX-3SAT gibt, solange $\mathcal{P} \neq \mathcal{NP}$ gilt.

Korollar 10.4 *Es gelte $\mathcal{P} \neq \mathcal{NP}$. Dann gibt es eine Konstante $\beta > 1$, so dass MAX-3SAT keine effizienten β-approximativen Algorithmen besitzt. Insbesondere besitzt MAX-3SAT kein polynomielles Approximationsschema.*

Beweis von Satz 10.3 (a) $\Rightarrow$ (b): Sei $L \in \mathcal{NP}$ eine beliebige Sprache. Wir können annehmen, dass $[\alpha \cdot g, g]$-gap-MAX-3SAT ein $\mathcal{NP}$-hartes Problem ist, wobei g die Anzahl der Klauseln angibt. Also gibt es eine effizient berechenbare Transformation T, so dass für Eingaben $w \in L$ alle Klauseln der Formel $T(w)$ erfüllbar sind, während weniger als der Prozentsatz α aller Klauseln für $w \notin L$ erfüllbar ist.
Wir konstruieren einen Verifier V für L, der logarithmisch viele Zufallsbits anfordert und drei Beweisbits nachfragt. Für eine Eingabe w für L nimmt V an, dass der Beweis aus einer erfüllenden Belegung für $T(w)$ besteht. V wählt eine Klausel k von $T(w)$ mit Hilfe der logarithmisch vielen Zufallsbits aus, fragt die Wahrheitswerte der drei Variablen von k ab und akzeptiert genau dann, wenn k erfüllt wird.
Wenn $w \in L$, dann ist $T(w)$ erfüllbar und V akzeptiert den aus der erfüllenden Belegung bestehenden Beweis mit Wahrscheinlichkeit 1. Ist $w \notin L$, dann ist weniger als der Bruchteil α aller Klauseln erfüllbar und V führt, für jeden „Beweis“, einen positiven Klauseltest

mit Wahrscheinlichkeit höchstens α durch. Also akzeptiert V fälschlicherweise mit Wahrscheinlichkeit höchstens α. Fazit: V ist ein ($O(\log_2 n), O(1)$)-Verifier, und dies war zu zeigen.

(b) $\Rightarrow$ (a): Wir können das $\mathcal{PCP}$-Theorem annehmen und müssen zeigen, dass wir jede Sprache $L \in \mathcal{NP}$ lückenschaffend auf $[\alpha \cdot g, g]$-gap-MAX-3SAT für ein passendes $\alpha < 1$ reduzieren können.
Da $L \in \mathcal{NP}$, folgt $L \in \mathcal{PCP}(O(\log n), O(1))$ aus dem $\mathcal{PCP}$-Theorem. Sei V ein Verifier, der L mit $O(\log_2 n)$ Zufallsbits und $k = O(1)$ inspizierten Beweisbits akzeptiert. Für eine Eingabe $w \in \{0,1\}^n$ inspiziert V also k viele Bits eines unbekannten Beweises $b = b_1 b_2 \cdots b_{n^d}$, wobei die Wahl der inspizierten Bitpositionen nur von

- der Folge $\sigma \in \{0,1\}^{O(\log n)}$ der Zufallsbits und
- der Eingabe w

abhängt. Wir fixieren σ. Dann können wir mit einer DNF-Formel $D_{w,\sigma}(x_1, \ldots, x_{n^d})$ mit höchstens k Literalen pro Monom festhalten, für welche Werte der Beweisbits der Verifier verwirft. Mit anderen Worten, die k-KNF Formel $\neg D_{w,\sigma}(x)$ beschreibt, wann V akzeptiert. Beachte, dass die Konstruktion von $\neg D_{w,\sigma}(x)$ in polynomieller Zeit gelingt, da V nur für die höchstens $2^k = O(1)$ vielen Kombinationen der Beweisbits zu simulieren ist. Wir definieren die k-KNF Formel

$$K_w^* = \bigwedge_\sigma \neg D_{w,\sigma}(x)$$

und beachten, dass auch K_w^* in polynomieller Zeit konstruierbar ist, da K_w^* eine Konjunktion von höchstens

$$2^{O(\log n)} = \text{poly}(n)$$

Einzelformeln $\neg D_{w,\sigma}$ ist. Damit sind wir fast fertig, denn:

- Für $w \in L$ erfüllt b als Wahrheitsbelegung interpretiert jede Klausel von K_w^*.
- Für $w \notin L$ wird jeder Beweis mit Wahrscheinlichkeit höchstens α akzeptiert und damit wird jede Wahrheitsbelegung höchstens den Bruchteil α aller Klauseln erfüllen.

Sei k_σ die Anzahl der Klauseln von $\neg D_{w,\sigma}(x)$ und r die Anzahl der verschiedenen Zufallsstrings σ. Dann wird für $w \notin L$ höchstens der Anteil

$$\frac{\sum_\sigma k_\sigma \ - (1-\alpha)\cdot r}{\sum_\sigma k_\sigma} = 1 - \frac{(1-\alpha)\cdot r}{\sum_\sigma k_\sigma} \leq 1 - \frac{(1-\alpha)\cdot r}{r \cdot 2^k} = 1 - \frac{1-\alpha}{2^{k+1}} =: \alpha'$$

aller Klauseln erfüllt. Allerdings müssen wir noch die k-Sat Formel in eine 3-Sat Formel übersetzen. Dazu führen wir neue Variablen y_i ein, und verwenden die Transformation

$$x_1 \vee x_2 \vee \cdots \vee x_m \ \Leftrightarrow \ (x_1 \vee x_2 \vee y_3) \wedge (\neg y_3 \vee x_3 \vee y_4) \wedge \cdots \wedge (\neg y_{m-1} \vee x_{m-1} \vee x_m)$$

Wie sieht der neue Wert von α' aus? $\square$

Aufgabe 106
Zeige, dass

$$\mathcal{PCP}(0, \log(n)) = \mathcal{P}$$

gilt.
Hinweis: Stellt euch vor, ihr sollt ein Vokabelabfrageprogramm schreiben, welches einen Schüler jeweils k von n Vokabeln abfragt. Beantwortet der Schüler alle k Fragen richtig, wird unterstellt, er habe alle Vokabeln gekonnt. Wie gut kann euer Programm sein, wenn euch kein Zufall zur Verfügung steht?

10.2 VERTEX COVER, INDEPENDENT SET und CLIQUE

Wir wissen mit Satz 10.3, dass $[\alpha g, g]$-gap-MAX-3SAT ein $\mathcal{NP}$-hartes Problem ist, wenn $g(x)$ die Anzahl der Klauseln von x und α eine Konstante mit $0 < \alpha < 1$ ist. Können wir jetzt die versprochene Lawine auslösen? In der folgenden Aufgabe wird gezeigt, dass auch MAX-2SAT ein $\mathcal{NP}$-hartes Gap-Problem besitzt.

Aufgabe 107

(a) Sei $K = (x_1 \vee x_2 \vee x_3)$ eine Klausel. **Konstruiere** eine 2-KNF Formel $\phi_K(x_1, x_2, x_3, y)$ mit einer neuen Variable y, so dass folgendes gilt. Für eine Belegung $\alpha \in \{0,1\}^3$ der Variablen x_1, x_2, x_3 und eine Belegung $b \in \{0,1\}$ von y sei

$$\#\phi_K(\alpha, b) = \text{die Anzahl der von der Belegung } (\alpha, b) \text{ erfüllten Klauseln in } \phi_K.$$

Dann gibt es eine Konstante $D \geq 1$, so dass

1. Für alle Belegungen (α, b) gilt $\#\phi_K(\alpha, b) \leq D$.
2. Für jede Belegung α gilt:
   * Falls $K(\alpha) = 1$, dann gibt es eine Belegung b, so dass $\#\phi_K(\alpha, b) = D$ ist.
   * Falls $K(\alpha) = 0$, dann $\#\phi_K(\alpha, b) \leq D - 1$ für beide Belegungen $b \in \{0,1\}$ von y und es gibt eine Belegung b mit $\#\phi_K(\alpha, b) = D - 1$.

Hinweis: Die kürzeste uns bekannte Lösung ϕ_K besteht aus 10 Klauseln (vier Klauseln der Länge 1 und sechs Klauseln der Länge 2) und die ensprechende Konstante ist $D = 7$.

(b) Für eine KNF-Formel x sei $g(x)$ die Anzahl der Klauseln von x. **Konstruiere** für jedes $\alpha < 1$ eine lückenerhaltende Reduktion von $[\alpha \cdot g, g]$-gap-MAX-3SAT auf $[\alpha' \cdot g, g]$-gap-MAX-2SAT für eine geeignet zu wählende Konstante $\alpha' < 1$.

Fazit: MAX-2SAT besitzt keine effizienten Algorithmen mit beliebig kleinen Approximationsfaktoren. Dieses Resultat ist überraschend, denn das Entscheidungsproblem 2-SAT liegt in $\mathcal{P}$.

Wir könnten schon jetzt zeigen, dass VERTEX COVER ein $\mathcal{NP}$-hartes Gap-Problem besitzt und ein analoges Resultat ist auch für INDEPENDENT SET leicht erreichbar. Unser Ziel ist aber ambitioniert: Wir möchten zeigen, dass INDEPENDENT SET bereits für Graphen von kleinem Grad nicht effizient mit beliebig kleinen Faktoren approximiert werden kann. Für allgemeine Graphen mit n Knoten möchten wir dieses Schwierigkeitsergebnis amplifizieren und zeigen, dass sogar effiziente n^ε-approximative Algorithmen ausgeschlossen sind!

Unser erstes Ziel ist der Nachweis, dass INDEPENDENT SET für Graphen von kleinem Grad schwierig zu approximieren ist. Als ein Zwischenziel betrachten wir deshalb MAX-3SAT$_B$ für eine Konstante B: Für eine gegebene 3KNF Formel, deren Variablen in höchstens B Klauseln vorkommen, sind möglichst viele Klauseln zu erfüllen. Wir zeigen, dass MAX-3SAT$_5$ ein $\mathcal{NP}$-hartes Gap-Problem besitzt.

Lemma 10.5 *Für eine KNF-Formel x sei $g(x)$ die Anzahl der Klauseln von x. Dann gibt es eine Konstante B, so dass $[\alpha g, g]$-gap-MAX-3SAT$_B$ ein $\mathcal{NP}$-hartes Problem für eine geeignete Konstante $\alpha < 1$ ist.*

Beweis: Wir nutzen aus, dass MAX-3SAT ein $\mathcal{NP}$-hartes Gap-Problem besitzt und konstruieren eine lückenerhaltende Reduktion von MAX-3SAT auf MAX-3SAT$_B$ für eine geeignete Konstante B.
Sei ϕ eine 3KNF-Formel mit den n Variablen $x_1, \ldots, x_n$ und m Klauseln. Die $i$te Variable komme m_i-mal in den Klauseln von ϕ vor. Wir setzen

$$N = \sum_i m_i$$

und erhalten $N \leq 3 \cdot m$, denn ϕ ist eine 3KNF-Formel.
Wir weisen ϕ eine 3KNF Formel ψ zu. Für eine geeignete natürliche Zahl B wird ψ die Eigenschaft besitzen, dass keine Variable mehr als B-mal auftritt. Hierzu ersetzen wir das $j$te Vorkommen der Variablen x_i durch die neue Variable $x_{i,j}$.
Jetzt müssen wir allerdings Sorge tragen, dass die Variablen $x_{i,1}, \ldots, x_{i,m_i}$ in optimalen Wahrheitsbelegungen auch den gleichen Wahrheitswert erhalten. Deshalb werden in ψ für jedes i neue Klauseln eingeführt.

Fakt 10.1 *Es gibt eine natürliche Zahl B, so dass für jede natürliche Zahl m ungerichtete Graphen $G_m = (V_m, E_m)$ in Zeit poly(m) konstruiert werden können. Die Graphen G_m besitzen die folgenden Eigenschaften:*

- $|V_m| = m$.
- *Jeder Knoten besitzt genau B Nachbarn.*
- *Die Expansionseigenschaft gilt: Für jede Knotenmenge $W \subseteq V_m$ gibt es mindestens $1+min\{|W|, |V_m - W|\}$ Kanten, die einen Knoten in W mit einem Knoten aus $V_m - W$ verbinden.*

Wir interpretieren die Knoten von G_{m_i} als Repräsentanten der neuen Variablen $x_{i,1}, \ldots, x_{i,m_i}$. Für jede Kante $\{r, s\}$ in G_{m_i} fügen wir jetzt die beiden Klauseln $x_{i,r} \vee \neg x_{i,s}$ und $\neg x_{i,r} \vee x_{i,s}$ als neue Klauseln zu ψ hinzu. Die Formel ψ besteht also insgesamt aus $B \cdot \sum_i m_i = B \cdot N$ neuen Klauseln und m modifizierten alten Klauseln. Die Expansions-Eigenschaft der Graphen G_m zahlt sich jetzt aus:

Behauptung 10.1 *Jede Wahrheitsbelegung, die eine größtmögliche Anzahl von Klauseln von ψ erfüllt, erfüllt* **alle** *neuen Klauseln.*

Beweis der Behauptung: Angenommen, eine Wahrheitsbelegung weist den Variablen $x_{i,1},.., x_{i,m_i}$ unterschiedliche Wahrheitswerte zu. Wir definieren $W = \{j \mid x_{i,j} = 0\}$ und dementsprechend ist $V_{m_i} \setminus W = \{j \mid x_{i,j} = 1\}$. Die Menge W habe, ohne Beschränkung der Allgemeinheit, höchstens $\frac{m_i}{2}$ Elemente. Dann garantiert der Fakt, dass G_{m_i} mindestens $1 + |W|$ Kanten mit einem Endpunkt in W und einem Endpunkt in $V_{m_i} \setminus W$ besitzt.
Wir flippen den Wahrheitswert der Variablen in W und erfüllen deshalb möglicherweise $|W|$ alte Klauseln nicht mehr. Im Gegenzug haben wir aber mindestens $1 + |W|$ neue Klauseln erfüllen können. □

Offensichtlich ist ϕ genau dann erfüllbar, wenn ψ erfüllbar ist. Wenn andererseits weniger als $\alpha \cdot m$ Klauseln von ϕ simultan erfüllbar sind, dann sind höchstens $\alpha \cdot m + B \cdot N$ Klauseln von ψ simultan erfüllbar. Es ist aber $N \leq 3m$ und deshalb sind in diesem Fall mehr als

$$(1-\alpha)\cdot m = \frac{(1-\alpha)\cdot m}{m + B \cdot N} \cdot (m + B \cdot N) \geq \frac{(1-\alpha)\cdot m}{m + 3 \cdot B \cdot m} \cdot (m + B \cdot N) = \frac{1-\alpha}{1 + 3 \cdot B} \cdot (m + B \cdot N)$$

Klauseln nicht erfüllt worden. □

Aufgabe 108
Zeige, dass Lemma 10.5 bereits für $B = 5$ gilt.
Hinweis: Benutze das Ergebnis von Lemma 10.5. Dann bleibt die Aufgabe, eine Formel mit bis zu B-maligem Vorkommen ihrer Variablen in eine Formel mit höchstens 5-maligem Vorkommen ihrer Variablen zu transformieren.
Führe wiederum neue Variablen für solche Variablen ein, die zu häufig vorkommen. Da wir jetzt davon ausgehen, dass Variablen maximal B mal vorkommen, können wir mit einfacheren Graphen –im Vergleich zur Konstruktion mit Hilfe von Fakt 10.1– "erzwingen", dass Kopien den gleichen Wahrheitswert annehmen.

Wir zeigen als nächstes, dass INDEPENDENT SET selbst für Graphen von beschränktem Grad schwierig zu approximieren ist. Dazu reduzieren wir MAX-3SAT$_B$ lückenerhaltend auf INDEPENDENT SET$_{B+1}$. (INDEPENDENT SET$_B$ ist die Einschränkung von INDEPENDENT SET auf Graphen mit höchstens B Nachbarn pro Knoten.)

Lemma 10.6 *Sei $B \in \mathbb{N}$ vorgegeben. Dann gilt*

$$\text{MAX} - \text{3SAT}_B \leq_{AP} \text{INDEPENDENT SET}_{B+1}.$$

Beweis: Sei ϕ eine 3KNF-Formel, so dass jede Variable in höchstens B Klauseln auftritt. Wir konstruieren einen ungerichteten Graphen G_ϕ aus ϕ, indem wir für jede Klausel eine Gruppe von 3 Knoten (mit einem Knoten pro Literal) anlegen. Wir setzen Kanten wie folgt ein:

- Je zwei Knoten einer Gruppe werden verbunden.
- Knoten verschiedener Gruppen werden genau dann verbunden, wenn die beiden Knoten einem Literal und seiner Negation entsprechen.

Offenbar ist jeder Knoten von G_ϕ mit höchstens $B - 1$ „negierten Knoten" aus anderen Klauseln und mit maximal zwei Knoten aus der Klausel verbunden. Folglich besitzt jeder Knoten in G_ϕ höchstens $B + 1$ Nachbarn.

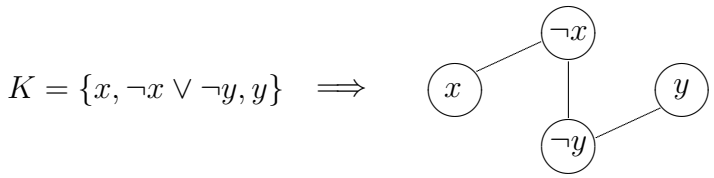


Abbildung 10.1: Die Konstruktion des Graphen G_ϕ

Sei U eine unabhängige Knotenmenge in G_ϕ. Dann besitzt U offensichtlich höchstens einen Knoten pro Gruppe. Da die Literale, die Knoten aus U entsprechen, simultan auf wahr gesetzt werden können, gibt es mindestens $|U|$ simultan erfüllbare Klauseln. Da es aber auch andererseits für jede Menge von u erfüllbaren Klauseln eine entsprechende unabhängige Menge der Größe u gibt (warum?), stimmt die maximale Anzahl simultan erfüllbarer Klauseln überein mit der Größe der größten unabhängigen Menge. □

Korollar 10.7 *Es gelte $\mathcal{P} \neq \mathcal{NP}$.*
Die Probleme MAX-3SAT$_5$, VERTEX COVER und INDEPENDENT SET$_6$ besitzen keine polynomiellen Approximationsschemata.

Beweis: Wir erhalten aus Lemma 10.5 und der darauf folgenden Übungsaufgabe, dass MAX-3SAT$_5$ kein polynomielles Approximationsschema besitzt. Auch INDEPENDENT SET$_6$ besitzt deshalb gemäß Lemma 10.6 kein polynomielles Approximationsschema. Die Behauptung für VERTEX COVER ist als Übungsaufgabe gestellt. □

Aufgabe 109
Zeige die folgende Aussage: Sei $L \in \mathcal{NP}$. Dann können wir für jede Eingabe w deterministisch in polynomieller Zeit einen Graphen G_w und eine Zahl k_w bestimmen, so dass es eine Konstante $\delta > 1$ mit den folgenden Eigenschaften gibt:

- Für $w \in L$ besitzt G_w eine Knotenüberdeckung der Größe höchstens k_w.
- Für $w \notin L$ besitzt jede Knotenüberdeckung mindestens $\delta \cdot k_w$ Knoten.

Hinweis: Betrachte den Beweis von Lemma 10.6.

Aufgabe 110
Gehört INDEPENDENT SET$_6$ zur Klasse $\mathcal{APX}$?

Wir kommen jetzt zu dem zentralen Ergebnis dieses Abschnitts.

Satz 10.8 *Es gibt $\varepsilon > 0$, so dass sowohl CLIQUE wie auch INDEPENDENT SET keine effizienten Approximationsalgorithmen mit Verlustfaktor n^ε besitzen.*

Beweis: CLIQUE und INDEPENDENT SET sind äquivalente Probleme, da die Größe einer größtmöglichen Clique in einem Graphen übereinstimmt mit der Größe einer größtmöglichen unabhängigen Menge im Komplementgraphen. Wir konzentrieren uns deshalb im Folgenden auf CLIQUE.

Wir skizzieren zuerst die Idee und führen das Graph-Produkt $G = G_1 \times G_2$ für ungerichtete Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ ein. Insbesondere ist $G = (V, E)$ mit $V = V_1 \times V_2$ und

$$E = \{ \{(u_1, v_1), (u_2, v_2)\} \mid \{u_1, u_2\} \in E_1, \{v_1, v_2\} \in E_2, \}.$$

Wir nehmen an, dass sowohl G_1 wie auch G_2 alle Eigenschleifen „$\{u, u\}$“ enthalten. Man überzeuge sich, dass

$$\text{clique}(G_1 \times G_2) = \text{clique}(G_1) \cdot \text{clique}(G_2)$$

gilt, wobei clique(H) die Größe der größten Clique von H bezeichne. Insbesondere ist

$$\text{clique}(H)^r = \text{clique}(H^r)$$

und es scheint, dass sich ein Approximationsfaktor v für den Graphen H in einen Approximationsfaktor v^r für den Graphen H^r übersetzt.

Aufgabe 111
Zeige: Wenn CLIQUE nicht mit dem Faktor α effizient approximierbar ist, dann ist CLIQUE auch nicht mit dem Faktor α^2 effizient approximierbar.

Leider aber ist dem nicht so, denn der Graph H^r ist wesentlich größer als der Graph H und Approximationsalgorithmen dürfen auf der sehr viel größeren Eingabe H^r dementsprechend mehr Laufzeit investieren. Wir benötigen vielmehr das folgende Konzept, um einerseits noch die Eigenschaften des Graphprodukts zu erhalten und um andererseits nicht die Größe des Produkt-Graphen hochzutreiben.

Definition 10.9 Seien n und k natürliche Zahlen und sei δ eine reelle Zahl. Ein (n, k, δ)-Booster ist eine Menge B von k-elementigen Teilmengen von $\{1, \ldots, n\}$. Für jede Teilmenge $A \subseteq \{1, \ldots, n\}$ gilt

$$(\frac{|A|}{n} - \delta)^k \cdot |B| \leq |\{T \in B \mid T \subseteq A\}| \leq (\frac{|A|}{n} + \delta)^k \cdot |B|.$$

Wenn B die Menge aller k-elementigen Teilmengen von $\{1, \ldots, n\}$ ist, dann ist $(\frac{|A|}{n})^k$ die (ungefähre) Wahrscheinlichkeit, dass eine fixierte Menge $T \in B$ in der Menge A enthalten ist.

> Ein (n, k, δ)-Booster ermöglicht also eine gute Approximation der Größe der Menge A, solange wir die Anzahl der Boostermengen kennen, die in A enthalten sind. Um die Anzahl der enthaltenen Teilmengen effizient bestimmen zu können, werden wir zusätzlich verlangen, dass der Booster aus nicht nicht zu vielen, effizient konstruierbaren Teilmengen besteht.

Insbesondere benötigen wir (n, k, δ)-Booster für $k = O(\log_2 n)$ und für kleines $\delta > 0$.

Fakt 10.2 *Für jedes $k = O(\log_2 n)$ und für jedes $\delta > 0$ gibt es (n, k, δ)-Booster, so dass alle Teilmengen des Boosters in polynomieller Zeit in n konstruiert werden können. (Insbesondere haben diese Booster also nur polynomiell viele Teilmengen.)*

Sei G ein ungerichteter Graph mit Knotenmenge $\{1,\ldots,n\}$. Anstelle des Graphprodukts G^r betrachten wir jetzt das Booster-Produkt $B(G)$, dessen Knoten den Teilmengen des Boosters B entsprechen. Zwei Boostermengen $S_1, S_2 \in B$ werden in $B(G)$ durch eine Kante verbunden, falls $S_1 \cup S_2$ eine Clique in G ist. Das Booster-Produkt erfüllt seinen Zweck, denn:

Behauptung 10.2 *Für jeden Graphen G und für jeden (n,k,δ)-Booster B gilt*

$$(\frac{\textit{clique}\ (G)}{n} - \delta)^k \cdot |B| \leq \textit{clique}\ (\ B(G)\ ) \leq (\frac{\textit{clique}\ (G)}{n} + \delta)^k \cdot |B|.$$

Beweis der Behauptung: Die Menge $A \subseteq \{1,\ldots,n\}$ sei eine größte Clique in G. Dann ist $|A| = \text{clique}\ (G)$, und der Booster besitzt mindestens $(\frac{\text{clique}\ (G)}{n} - \delta)^k \cdot |B|$ Teilmengen von A. Sämtliche in A enthaltenen Boostermengen bilden aber eine Clique in $B(G)$ und es ist

$$(\frac{\text{clique}\ (G)}{n} - \delta)^k \cdot |B| \leq \text{clique}\ (\ B(G)\ ).$$

Andererseits sei A' die größte Clique in $B(G)$. Wir definieren A als die Vereinigung aller Mengen des Boosters, die den Elementen von A' entsprechen. Dann ist A offensichtlich eine Clique in G und es ist $|A| \leq \text{clique}(G)$. Der Booster besitzt also höchstens $(\frac{|A|}{n} + \delta)^k \cdot |B| \leq (\frac{\text{clique}\ (G)}{n} + \delta)^k \cdot |B|$ viele Teilmengen von A. Wir haben also

$$\text{clique}\ (\ B(G)\ ) = |A'| \leq |\{T \in B \mid T \subseteq A\}| \leq (\frac{\text{clique}\ (G)}{n} + \delta)^k \cdot |B|$$

erhalten. □

Wir wissen mit Lemma 10.6, dass $[\alpha n, \beta n]$-gap-INDEPENDENT SET_6 ein $\mathcal{NP}$-hartes Problem ist. Also ist auch $[\alpha n, \beta n]$-gap-CLIQUE_{n-6} ein $\mathcal{NP}$-hartes Problem. Im Gap-Problem für CLIQUE sind aber nur solche Graphen G interessant, für die

$$\text{clique}\ (G) < \alpha \cdot n \ \text{ oder } \ \beta \cdot n \leq \text{clique}\ (G).$$

gilt. Wir konstruieren einen $(n, \log_2 n, \delta)$-Booster und das entsprechende Booster-Produkt $B(G)$. Als Konsequenz der Behauptung 10.2 erhalten wir also

$$\text{clique}\ (\ B(G)\ ) \leq (\alpha + \delta)^{\log_2 n} \cdot |B| \ \text{ oder } \ (\beta - \delta)^{\log_2 n} \cdot |B| \leq \text{clique}\ (B(G))\ .$$

Für ein genügend kleines δ können wir also die Lücke von vorher $\frac{\beta}{\alpha}$ auf jetzt $(\frac{\beta-\delta}{\alpha+\delta})^{\log_2 n}$ polynomiell amplifizieren. □

In [Ha1] wird sogar gezeigt, dass CLIQUE, für jedes $\varepsilon < 0$, keine effizienten $n^{1-\varepsilon}$-approximativen Algorithmen besitzt.

In COLORING ist ein ungerichteter Graph G gegeben und es wird nach einer Knotenfärbung mit einer kleinstmöglichen Farbenzahl gefragt, so dass keine zwei benachbarten Knoten in G die gleiche Farbe besitzen. Beachte, dass eine alternative Formulierung die

Frage nach einer Zerlegung von G in eine kleinstmögliche Zahl unabhängiger Mengen ist. In [FK] wird gezeigt, dass auch COLORING keine effizienten $n^{1-\varepsilon}$-approximativen Algorithmen besitzt.
Diese Ergebnisse sind ein enormer Fortschritt, wenn man beachtet, dass für lange Jahre sogar polynomiellen Approximationsschemata für CLIQUE nicht ausgeschlossen waren. Allerdings ist unser Wissen immer noch sehr eingeschränkt:

- Angenommen, wir erhalten das Versprechen, daß ein gegebener Graph 3-färbbar ist. Dann sind die besten effizienten Algorithmen bisher nur im Stande, eine Färbung mit $O(n^{0,25} \cdot \log n)$ Farben zu finden.

- Die besten effizienten Approximationsalgorithmen für INDEPENDENT SET erreichen nur den Approximationsfaktor $O(\frac{n}{\log^2 n})$.

In beiden Fällen fehlen Resultate, die die Existenz effizienter Algorithmen ausschließen, bzw. die Nicht-Existenz plausibel machen.

Aufgabe 112
Es gelte $\mathcal{P} \neq \mathcal{NP}$. **Zeige**, dass COLORING keine effizienten $\frac{4}{3}$-approximativen Algorithmen besitzt.
Hinweis: Das 3-Färbbarkeitsproblem ist $\mathcal{NP}$-vollständig.

Aufgabe 113
Wir stellen die Probleme FEEDBACK VERTEX SET und FEEDBACK ARC SET vor, die in der Lösung von Deadlock Problemen eine zentrale Rolle spielen.
In MINIMUM FEEDBACK VERTEX SET ist ein gerichteter Graph $G = (V, E)$ gegeben. Gesucht ist eine minimale Knotenmenge $W \subseteq V$, so dass W mindestens einen Knoten eines jeden gerichteten Zyklus enthält. Die Herausnahme von W bricht somit alle Zyklen.
In MINIMUM FEEDBACK ARC SET ist ein gerichteter Graph $G = (V, E)$ gegeben. Gesucht ist eine minimale Kantenmenge $A \subseteq E$, so dass A mindestens eine Kante eines jeden gerichteten Zyklus enthält.

(a) **Zeige:** VERTEX COVER $\leq_{AP}$ MINIMUM FEEDBACK VERTEX SET.

(b) **Zeige:** MINIMUM FEEDBACK VERTEX SET $\leq_{AP}$ MINIMUM FEEDBACK ARC SET.

Fazit: Auch für MINIMUM FEEDBACK VERTEX SET und MINIMUM FEEDBACK ARC SET sind keine polynomiellen Approximationsschemata zu erwarten.

10.3 Hastad's 3-Bit $\mathcal{PCP}$

Im $\mathcal{PCP}$-Theorem wird gezeigt, dass jede Sprache verifizierbare Beweise besitzt, für die die Nachfrage konstant vieler Beweisbits genügt, wenn logarithmisch viele Zufallsbits zur Verfügung stehen. Genügt die Nachfrage nach drei oder sogar nach nur zwei Beweisbits? Zwei Bits sind unzureichend, denn:

Aufgabe 114
Zeige, dass

$$\mathcal{PCP}(O(\log n), 2) = \mathcal{P}$$

gilt. *Hinweis*: Eine Sprache $L \in \mathcal{PCP}(O(\log n), 2)$ werde von dem Verifier V erkannt. Versuche für eine Eingabe x eine 2-KNF Formel K_x zu konstruieren, so dass K_x genau dann erfüllbar ist, wenn es einen Beweis gibt, der von V stets akzeptiert wird. Wir wissen, dass 2-SAT in polynomieller Zeit lösbar ist.

Andereseits genügen tatsächlich drei Bits fast, denn:

Satz 10.10 *Für alle Konstanten $\varepsilon, \eta > 0$ und jede Sprache $L \in \mathcal{NP}$ gibt es einen Verifier V, so dass*

(a) V nur drei Beweisbits x, y, z anfordert, mit $O(\log_2 n)$ Zufallsbits auskommt

(b) und Akzeptanz über den Wert einer Summe $\alpha \cdot x + \beta \cdot y + \gamma \cdot z \bmod 2$ entscheidet.

Weiterhin gibt es für jede Eingabe $w \in L$ stets einen Beweis, der mit Wahrscheinlichkeit $1 - \varepsilon$ akzeptiert wird. Ist hingegen $w \notin L$, dann wird jeder Beweis mit Wahrscheinlichkeit höchstens $1/2 + \eta$ akzeptiert.

Aufgabe 115
Zeige: Wenn der Verifier in Satz 10.10 für jede Eingabe $w \in L$ einen Beweis mit Wahrscheinlichkeit 1 akzeptieren würde, dann gilt $\mathcal{P} = \mathcal{NP}$.

Wir geben keinen Beweis an, sondern untersuchen die Konsequenzen der Aussage. Zuerst betrachten wir das Optimierungsproblem MAX-3LIN: Wir erhalten ein lineares Gleichungssystem

$$A \cdot x = b \bmod 2.$$

über den ganzen Zahlen modulo zwei. Unsere Aufgabe ist die Bestimmung eines Vektors x, so dass möglichst viele Gleichungen des Systems erfüllt werden. Was passiert, wenn wir einen Vektor x zufällig auswürfeln? Jede einzelne Gleichung wird mit Wahrscheinlichkeit $1/2$ erfüllt, und wir werden im Erwartungsfall die Hälfte aller Gleichungen erfüllen.
Betrachten wir als Nächstes MAX-3SAT. Wenn wir eine Belegung zufällig auswürfeln, dann werden wir eine bestimmte Klausel mit Wahrscheinlichkeit $7/8$ erfüllen, da die Klausel von sieben ihrer acht möglichen Belegungen erfüllt wird. Geht es besser? Überraschenderweise ist in beiden Fällen „nicht mehr drin".

Satz 10.11 *Es gelte $\mathcal{P} \neq \mathcal{NP}$.*

(a) Dann besitzt MAX-3LIN für jedes $\varepsilon > 0$ keine effizienten, $2 - \varepsilon$-approximativen Algorithmen.

(b) Auch MAX-3SAT besitzt für jedes $\varepsilon > 0$ keine effizienten, $8/7 - \varepsilon$-approximativen Algorithmen.

Beweis (a): Wir versuchen das $\mathcal{NP}$-vollständige Erfüllbarkeitsproblem zu lösen. Wir wenden Satz 10.10 an und erhalten einen Verifier, der für jede Folge σ von logarithmisch vielen Zufallsbits akzeptiert, wenn die Gleichung

$$\alpha_\sigma \cdot x_\sigma + \beta_\sigma \cdot y_\sigma + \gamma_\sigma \cdot z_\sigma = b_\sigma$$

für die nachgefragten Beweisbits $x_\sigma, y_\sigma, z_\sigma$ erfüllt ist. Weiterhin wissen wir, dass die Eingabe zu akzeptieren ist, wenn fast alle Gleichungen erfüllt werden, und zu verwerfen ist,

wenn nur wenig mehr als die Hälfte aller Gleichungen erfüllt werden. Wenn wir MAX-3LIN mit einem $(2-\varepsilon)$-approximativen Algorithmus lösen könnten, dann hätten wir das Erfüllbarkeitsproblem geknackt und das geht nun mal nicht.
(b) Wir reduzieren MAX-3LIN auf MAX-3SAT. Dazu schreiben wir jede Gleichung des Systems $A \cdot x = b \bmod 2$ als die Konjunktionen von vier Klauseln und erhalten statt des linearen Systems eine 3KNF Formel ϕ.
Wenn $A \cdot x = b \bmod 2$ „fast“ erfüllbar ist, dann sind auch fast alle Klauseln erfüllbar. Sind hingegen nur wenig mehr als die Hälfte aller Gleichungen erfüllbar, dann sind bei m Gleichungen ungefähr

$$4 \cdot \frac{m}{2} + 3 \cdot \frac{m}{2} = \frac{7}{2} \cdot m = \frac{7}{8} \cdot 4m$$

der $4m$ Klauseln erfüllbar. Und MAX-3SAT kann keine effizienten $(8/7-\varepsilon)$-approximativen Algorithmen besitzen. □

Bemerkung 10.1 Beachte, dass wir den Beweis von Teil (b) durch eine lückenbewahrende Reduktion zwischen den Gap-Versionen von MAX-3LIN und MAX-3SAT durchgeführt haben: Die ursprüngliche Lücke von fast einer Hälfte wurde auf eine Lücke von fast einem Achtel transformiert.

10.4 Parallel Repetition

Das Ziel dieses Abschnitts ist die Untersuchung der Approximationskomplexität von SET COVER. Wir müssen einen ziemlichen Anlauf nehmen: Das Parallel Repetition Theorem wird die zentrale Methode sein, die uns helfen wird.
Anwendungen des PCP-Theorems für die Nichtapproximierbarkeit sind umso stärker je kleiner der Soundness-Parameter α ist. Ein (r, q)-Verifier kann seine Soundness von α auf α^k senken, indem q Anfragen k-mal **sequentiell** wiederholt werden. Der große Nachteil dieses Vorgehens ist die um den Faktor k angestiegene Anzahl der Anfragen: In Hastad's 3-Bit $\mathcal{PCP}$ war die Konstruktion eines Verifiers mit nur drei Anfragen und Soundness fast $1/2$ der wesentliche Beitrag.
Wenn wir die Anzahl q der Nachfragen unverändert lassen wollen, bietet es sich an, alle k „ersten“ Anfragen in einer „Kombi-Anfrage“ zu bündeln, die dann vom Prover mit einem k-Bit String **parallel** beantwortet werden. Wir verfahren mit den k „zweiten“ „dritten“, ... „$q$ten“ Anfragen analog, bleiben also bei k Anfragen, haben aber die 1-Bit Antworten des Provers auf k-Bit Antworten erhöht, was wir uns als Vergrößerung des Alphabets des Provers vorstellen können. Überraschenderweise ist die Alphabetvergrößerung in vielen Anwendungen unproblematisch, aber funktioniert dieses Vorgehen, d.h. wird der Soundness Parameter tatsächlich von α auf α^k erniedrigt?
Die deprimierende Antwort ist `nein`, die Vergrößerung des Alphabets von einem auf k Bits ist das Problem. Für $k = 2$ können wir uns vorstellen, dass sich das 1-dimensionale Format des alten Beweises in ein 2-dimensionales Format ändert: Für Anfragen (i, j) und (i, l) sind jetzt Antworten (a, b) und (c, d) mit $a \neq c$ möglich, und der Verifier muss sich jetzt mit nach seiner Ansicht inkonsisten Beweisen auseinandersetzen!

Um dieses Problem in den Griff zu bekommen, betrachten wir 2-Prover Spiele.

10.4.1 2-Prover Spiele

Statt einem Verifier und einem Prover betrachten wir jetzt einen Verifier und zwei Prover. Der Verifier formuliert genau eine Anfrage an den ersten und ebenfalls genau eine Anfrage an den zweiten Prover. Der erste Prover antwortet mit einem Buchstaben aus dem Alphabet Σ_1, der zweite Prover antwortet mit einem Buchstaben aus Alphabet Σ_2. Die beiden Prover dürfen sich vor Beginn der Rechnung absprechen, dürfen aber während der Rechnung nicht mehr miteinander kommunizieren.
Wir sagen, dass eine Sprache L zur Klasse $2\mathrm{P}_{\beta,\alpha}(r)$ gehört, wenn es einen effizienten Verifier V gibt der für Eingaben der Länge n höchstens $r(n)$ Zufallsbits anfordert. Es muss gelten:

(a) *Vollständigkeit*: Wenn die Eingabe w zur Sprache L gehört, dann gibt es Beweise der beiden Prover, so dass V mit Wahrscheinlichkeit mindestens β akzeptiert.

(b) *Soundness*: Wenn die Eingabe w nicht zur Sprache L gehört, dann wird V alle Beweise mit Wahrscheinlichkeit höchstens α akzeptieren.

Obwohl wir den Verifier sehr stark eingeschränkt haben –es darf nur jeweils ein Beweisbit nachgefragt werden– erhalten wir ein neues $\mathcal{PCP}$-Theorem:

Satz 10.12 *Es gibt $\alpha < 1$ mit $\mathcal{NP} = 2P_{1,\alpha}(O(\log_2 n))$.*

Beweis: Wenn $L \in 2P_{1,\alpha}(O(\log_2 n))$, dann rate zwei Beweise nichtdeterministisch. Falls $w \in L$ werden wir zwei Beweise finden, so dass V für alle polynomiell vielen Zufallstrings akzeptiert. Falls $w \notin L$ werden wir mindestens einen Zufallsstring finden, für den V nicht akzeptiert. Da V ein effizienter deterministischer Algorithmus ist, haben wir einen effizienten nichtdeterministischen Algorithmus für L gefunden und $L \in \mathcal{NP}$.
Für die Umkehrung $\mathcal{NP} \subseteq 2P_{1,\alpha}(O(\log_2 n))$ genügt der Nachweis, dass das $\mathcal{NP}$-vollständige 3SAT Problem in $2P_{1,\alpha}(O(\log_2 n))$ liegt. Sei ϕ eine 3KNF-Formel. Wir wenden Satz 10.3 an und erhalten eine effiziente Transformation T mit den Eigenschaften:

- Wenn ϕ erfüllbar ist, dann ist auch $T(\phi)$ erfüllbar.

- Ist ϕ nicht erfüllbar, dann ist höchstens ein Bruchteil $\alpha < 1$ aller Klauseln von $T(\phi)$ erfüllbar.

Unser Verifier V erwartet, dass der Beweis des ersten Provers eine erfüllende Belegung von $T(\phi)$ ist, der erste Prover verwendet demgemäß das binäre Alphabet für die Beantwortung. V erwartet vom zweiten Prover eine erfüllende Belegung für jede Klausel von $T(\phi)$; das Alphabet des zweiten Provers hat also die Größe acht.
V wählt eine Klausel und eine Variable der Klausel zufällig. Der erste Prover antwortet mit dem Wert der Variablen, der zweite Prover mit der Belegung der Klausel. V akzeptiert, wenn beide Antworten konsistent sind und die Klausel erfüllt wird.

Fall 1: ϕ hat eine erfüllende Belegung x. Die beiden Beweise, nämlich die erfüllende Belegung x und die Belegung x auf alle Klauseln eingeschränkt, werden mit Wahrscheinlichkeit 1 akzeptiert.
Fall 2: ϕ ist nicht erfüllbar. Dann wird höchstens der Bruchteil $\alpha < 1$ aller Klauseln erfüllt. Mit Wahrscheinlichkeit mindestens $1 - \alpha$ wird V eine falsifizierte Klausel finden. Aber die beiden Prover haben noch die Chance des Betrugs: Der zweite Prover könnte nämlich eine erfüllende Belegung der Klausel präsentieren! Mit Wahrscheinlichkeit 1/3 fliegt der Betrug aber auf, da V die Inkonsistenz bemerkt. Wir erreichen somit den Soundness-Parameter $1 - \frac{1-\alpha}{3} < 1$. □

Wie sieht es jetzt mit der parallelen Wiederholung von Anfragen aus?

Satz 10.13 *Das Parallel Repetition Theorem von Raz*

Der Verifier V möge Vollständigkeit 1 und Soundness α erreichen. Die k-malige parallele Wiederholung für Verifier V führt auf Vollständigkeit 1 und Soundness höchstens $(\alpha')^k$. Die Konstante α' hängt nur von α und den Alphabetgrößen der beiden Prover ab; die Anzahl der Anfragen bleibt bei Eins, die Anzahl nachgefragter Zufallsbits steigt auf $k \cdot r$ an.

Den komplizierten Beweis zeigen wir nicht.
Wir haben keine „perfekte“ Reduktion des Soundness-Parameters erreicht, da der Soundness Parameter auf $(\alpha')^k$ und nicht auf α^k reduziert wurde. Dieses Ergebnis ist aber bestmöglich, insbesondere kann eine perfekte Reduktion nicht erreicht werden.

Bemerkung 10.2 Warum ist eine Reduktion des Soundness-Parameters von α auf α^k im Allgemeinen nicht möglich? Der Verifier V wählt ein Paar $(r_1, r_2) \in \{0,1\}^2$ zufällig aus und sendet r_i an Prover i. Der Verifier erwartet von Prover i eine Antwort (j, r_j) für $j \in \{1,2\}$ und akzeptiert, falls beide Antworten richtig und identisch sind.
Mit welcher Wahrscheinlichkeit p akzeptiert V? Wenn beide Prover mit $(1, r_1)$ antworten, dann ist $p \leq \frac{1}{2}$, da Prover 2 das Bit r_1 nicht kennt. Offensichtlich gilt $p \leq \frac{1}{2}$ auch unter der Bedingung, dass beide Prover mit $(2, r_2)$ antworten: Der Verifier akzeptiert also mit Wahrscheinlichkeit höchstens 1/2.
Wir führen $k = 2$ parallele Wiederholungen durch. Der Verifier wählt also eine Folge (r_1, r_2, s_1, s_2) von vier Bits zufällig aus und sendet (r_1, r_2) an Prover 1 und (s_1, s_2) an Prover 2. Diesmal erwartet V Antworten der Form $(i_1, r_{i_1}, i_2, s_{i_2})$ und akzeptiert, wenn beide Antworten richtig und identisch sind.
Wir erwarten, dass der Soundness-Parameter auf höchstens 1/4 sinkt, tatsächlich bleibt der Soundness-Parameter unverändert auf 1/2. Dazu antwortet Prover 1 mit $(1, r_1, 2, r_1)$ und Prover 2 mit $(1, s_2, 2, s_2)$. Der Verifier akzeptiert genau dann, wenn $r_1 = s_2$ und dies passiert mit Wahrscheinlichkeit 1/2.

Aufgabe 116
Welchen Wert des Soundness-Parameters kann der Verifier by k-maliger paralleler Wiederholung nicht unterschreiten?

10.4.2 LABEL COVER

Im Problem LABEL COVER modellieren wir unser Vorgehen in Satz 10.12. Eine Instanz von LABEL COVER besteht aus einem bipartiten Graphen $G = (A \cup B, E)$, wobei alle Kanten einen Knoten in A mit einem Knoten in B verbinden. Für jeden Knoten $v \in A \cup B$ ist eine endliche Menge L_v von Markierungen und für jede Kante $e = \{a, b\}$ mit $a \in A$ und $b \in B$ ist eine Funktion $f_{a,b} : L_a \to L_b$ gegeben. Unser Ziel ist eine „Färbung“ f_A der Knoten $a \in A$ (mit $f_A(a) \in L_a$) sowie eine „Färbung“ f_B der Knoten in $b \in B$ (mit $f_B(b) \in L_b$), so dass

$$|\{e = \{a, b\} \in E \mid f_{a,b}(f_A(a)) = f_B(b)\}|$$

möglichst groß ist: Die Knoten sind also so zu färben, dass die Endpunkte möglichst vieler Kanten $\{a, b\}$ konsistent, also gemäß der „Bedingung“ $f_{a,b}$ gefärbt sind.

Bemerkung 10.3 Sei p eine Primzahl. Wir betrachten lineare Gleichungssysteme, die nur aus Gleichungen der Form

$$a_{i,j} \cdot x_i + b_{i,j} \cdot y_j = c_{i,j}$$

bestehen, wobei die Koeffizienten $a_{i,j}$ und $b_{i,j}$ nicht durch p teilbar seien. Unser Ziel ist die simultane Erfüllung möglichst vieler Gleichungen.
Dieses Problem können wir als ein LABEL COVER Problem auffassen, wenn wir mit $L = \{0, \ldots, p-1\}$ als Menge der Markierungen und mit den Kantenbedingungen $f_{i,j}$ durch $f_{i,j}(x) = (c_{i,j} - a_{i,j} \cdot x) \cdot b_{i,j}^{-1}$ arbeiten: Die Knotenfärbungen f_A und f_B entsprechen Wertezuweisungen an die Variablen x_i und x_j, die nur dann eine Kantenbedingung erfüllen, wenn die zugehörige Gleichung erfüllt ist.
Man beachte, dass die Kantenbedingungen sogar Permutationen sind. Man spricht deshalb auch von eindeutigen Spielen.

Aufgabe 117
Zeige, dass die Frage, ob alle Kantenbedingungen eingehalten werden können, effizient beantwortet werden kann.

In Satz 10.12 haben wir einen effizienten Verifier V für 3SAT im 2-Prover Spiel angegeben. Wir haben eine 3KNF Formel ϕ in eine andere 3KNF Formel $\psi = T(\phi)$ transformiert, wobei entweder alle oder nur der Bruchteil $\alpha < 1$ aller Klauseln von ψ erfüllbar ist. Desweiteren besitze ψ genau n Variablen und m Klauseln. Nach k-maliger paralleler Wiederholung von V stellt V genau m^k parallele Fragen nach k Klauseln und n^k Fragen nach k in den jeweiligen Klauseln auftauchenden Variablen; die Soundness sinkt auf höchstens $(\alpha')^k$.
Wir modellieren die k-malige Wiederholung von V durch die folgende LABEL COVER Instanz $G = (A \cup B, E)$.

- Die Knoten in A entsprechen den (parallelen) Klausel-Anfragen von V an Prover 2, die Knoten in B den (parallelen) Variablen-Anfragen an Prover 1.

- Wir verbinden $a \in A$ und $b \in B$ mit einer Kante genau dann, wenn die Anfragen zu a und b für mindestens einen Zufallsstring simultan gestellt werden.

- Wenn die Fragen nach erfüllenden Belegungen der ψ-Klauseln $K_1, \ldots, K_k$ dem Knoten a entspricht, dann definieren wir L_a als die Menge aller 7^k, die Klauseln K_i erfüllenden Belegungen. Für alle „rechten“ Knoten $b \in B$ ist $L_b = \{0,1\}^k$.

- Für jede Kante $\{a,b\} \in E$ setzen wir $f_{a,b}(x) = y$ genau dann, wenn der Verifier nach Erhalt der Antworten x (von Prover 2) und y (von Prover 1) akzeptiert. (Wir benutzen hier, dass V genau dann akzeptiert, wenn x die angefragte Klauseln erfüllt und die Bits y konsistent mit x sind. Die Funktion $f_{a,b}$ ist also eine Projektion.)

Jedes Paar von Beweisen definiert eine Knotenfärbung und umgekehrt. Weiterhin stimmt die relative Anzahl der Kanten, die kompatibel gefärbt sind, überein mit der Akzeptanzwahrscheinlichkeit. Also folgt:

Lemma 10.14 *Sei $k \in \mathbb{N}$.*
Wenn für Graphen mit $O(n^k)$ Knoten und Labelmengen L_v der Größe höchstens $2^{O(k)}$ eine Klassifizierung in

- *YES-Instanzen (alle Kantenbedingungen werden eingehalten) und*

- *NO-Instanzen (nur der Bruchteil α^k aller Kantenbedingungen wird eingehalten)*

in Zeit $\mathrm{poly}(n)$ gelingt, dann kann jede Sprache $L \in \mathcal{NP}$ in Zeit $\mathrm{poly}(n^k)$ akzeptiert werden.

Zusätzlich kann sogar noch gefordert werden, dass die LABEL COVER Instanzen G genauso viele linke wie rechte Knoten besitzen und dass alle Knoten in G den gleichen Grad besitzen. Dazu benutzt man, dass MAX-3SAT$^*_5$ eine lückenschaffende Reduktion besitzt. (3KNF$^*_5$ besteht aus allen 3KNF-Formeln, so dass jede Klausel in *genau* fünf Klauseln vorkommt.) Die Klausel ψ besteht dann aus $m = 5n/3$ Klauseln (mit insgesamt n Knoten) und der von uns aufgebaute Graph hat deshalb $m^k = (5n/3)^k$ linke und n^k rechte Knoten, jeder linke Knoten hat 3^k rechte Nachbarn und jeder rechte Knoten hat 5^k linke Nachbarn. Jetzt erzeuge 3^k Kopien für jeden linken Knoten, 5^k Kopien für jeden rechten Nachbarn und verbinde die Kopien entsprechend. Wir haben jetzt insgesamt jeweils $(5n)^k$ linke, bzw. rechte Knoten und jeder Knoten hat $(15)^k$ Nachbarn.

10.4.3 SET COVER

Wir beginnen mit einer Vorüberlegung. Wie können wir eine SET COVER Instanz konstruieren, so dass es nur sehr wenige optimale Überdeckungen gibt und alle anderen Überdeckungen sehr viel mehr Mengen benötigen? Für ein Universum U würfeln wir t Teilmengen $S_1, \ldots, S_t \subseteq U$ aus, indem jedes Element $u \in U$ mit Wahrscheinlichkeit genau $1/2$ aufgenommen wird. Wir vervollständigen das Mengensystem durch die Aufnahme der Komplementmengen $\overline{S_1}, \ldots, \overline{S_t}$ und erhalten das System

$$S(U,t) = \{S_1, \overline{S_1}, \ldots, S_t, \overline{S_t}\}.$$

Wir haben genau t-viele 2er Überdeckungen $S_i, \overline{S_i}$, während mindestens $l = \Theta(\ln(|U|))$ Mengen notwendig sind, wenn kein komplementäres Paar in der Überdeckung auftritt:

Die erste Menge der Überdeckung wird im Erwartungsfall genau die Hälfte des Universums überdecken, die zweite Menge überdeckt ein fehlendes Viertel, die dritte Menge ein fehlendes Achtel und so weiter.

Wir müssen LABEL COVER lückenerhaltend auf SET COVER reduzieren. Sei also $G = (A \cup B, E)$ eine LABEL COVER Instanz mit $|A| = |B| = O(n^k)$ und Labelmengen L_v der Größe höchstens $2^{O(k)}$.

(*) Wir weisen den Kanten $e = \{a, b\} \in E$ disjunkte Universen U_e mit $|U_e| = n^k$ zu. Desweiteren weisen wir e das Mengensystem $S(U_e, t)$ mit $t = |L_b|$ zu. Es ist also

$$S(U_e, t) = \{S_{e,y} \mid y \in L_b\} \cup \{\overline{S_{e,y}} \mid y \in L_b\}.$$

(*) Das Universum für unsere SET COVER Instanz ist $U = \bigcup_{e \in E} U_e$.

(*) Jedes Paar (a, x) (mit $a \in A$ und $x \in L_a$) und (b, y) (mit $b \in B$ und $y \in L_b$) erzeugt genau eine Teilmenge $S_{a,x}$, bzw. $S_{b,y}$ in unserer SET COVER Instanz, nämlich

$$S_{a,x} = \bigcup_{a \text{ ist Endpunkt von } e;\ f_e(x)=y} \overline{S_{e,y}} \quad \text{und} \quad S_{b,y} = \bigcup_{b \text{ ist Endpunkt von } e} S_{e,y}$$

Haben wir eine lückenerhaltende Reduktion erreicht, wenn unser System von Teilmengen genau aus den Mengen $S_{a,x}$ (für $a \in A$ und $x \in L_a$) und $S_{b,y}$ (für $b \in B$ und $y \in L_b$) besteht?

Fall 1: Es gibt Knotenfärbungen f_A, f_B, die alle Kantenbedingungen einhalten. Wir wählen genau die Mengen $S_{a,f_A(a)}$ und $S_{b,f_B(b)}$ aus: Wenn $e = \{a, b\}$ eine Kante ist, dann ist die Kantenbedingung $f_{a,b}(f_A(a)) = f_B(b)$ erfüllt. Also gilt $\overline{S_{e,f_B(b)}} \subseteq S_{a,f_A(a)}$ wie auch $S_{e,f_B(b)} \subseteq S_{b,f_B(b)}$.
Wir haben U_e für jede Kante e und damit auch U überdeckt, wobei wir höchstens $O(n^k)$ Mengen benutzen.

Fall 2: Alle Knotenfärbungen f_A, f_B halten höchstens den Bruchteil α^k aller Kantenbedingungen ein.

Behauptung 10.3 *Sei $l = \Theta(\log_2 |U|)$ die Mindestanzahl von Mengen in einer Überdeckung einer Mengen U_e, wenn kein komplementäres Paar von Mengen in der Überdeckung benutzt wird.*
Dann besteht jede Überdeckung aus mindestens $\Omega(l \cdot n^k)$ Mengen für $k = \Theta(\log \log n)$.

Beweis: Für eine optimale Überdeckung $\mathcal{U}$ von U und einen Knoten v sei

$$A_v = \{S_{v,z} \mid S_{v,z} \text{ wird in } \mathcal{U} \text{ benutzt}\}.$$

Betrachte die Menge `gering` aller Knoten v mit $|A_v| < l/2$. Wenn $a \in \texttt{gering} \cap A$ und $b \in \texttt{gering} \cap B$ durch eine Kante in G miteinander verbunden sind, dann besitzen A_a und

A_b ein komplementäres Paar: Es ist $|A_a| + |A_b| < l$ und deshalb wird ein komplementäres Paar zur Überdeckung von U_e (für $e = \{a, b\}$) benötigt.
Jetzt wähle zufällig genau ein Element aus jeder Menge A_v für $v \in$ `gering` und färbe v mit diesem Element. Für jede Kante $e = \{a, b\}$ mit $a, b \in$ `gering` ist die Wahrscheinlichkeit, dass die Kantenbedingung zu e eingehalten wird, mindestens $\frac{1}{l/2} \cdot \frac{1}{l/2} = \frac{4}{l^2}$. Wenn E' die Menge aller Kanten zwischen Knoten zur Menge `gering` gehört, dann werden im Erwartungsfall also mindestens $\frac{4 \cdot |E'|}{l^2}$ Kantenbedingungen eingehalten. Maximal können aber nur $(\alpha')^k \cdot |E|$ Kantenbedingungen eingehalten werden und

$$\frac{4 \cdot |E'|}{l^2} \leq (\alpha')^k \cdot |E|$$

folgt. Jetzt wähle k so, dass $(\alpha')^k \cdot l^2/4 \leq 1/2$ ist, und wir erhalten $|E'| \leq |E|/2$. Wenn aber die Kantenmenge E' klein ist, dann sollte doch die Menge `gering` nicht zu groß sein! Warum? Der Graph G ist regulär. Wenn mehr als drei Viertel aller Knoten in A und mehr als drei Viertel aller Knoten in B zu `gering`, dann ist $|E'| > |E|/2$.
Also gehört mindestens ein Achtel aller Knoten nicht zur Menge `gering`. Die Anzahl der Mengen in der Überdeckung $\mathcal{U}$ ist also mindestens $\Omega(l \cdot n^k)$. □

Wir haben die Lücke für $k = \Theta(\log_2 \log_2 n)$ erhalten können, denn entweder reicht eine Überdeckung mit $O(n^k)$ Mengen oder mindestens $\Omega(n^k \cdot \log_2(n^k))$ Mengen sind notwendig. Wir wenden Lemma 10.14 an und erhalten als Konsequenz:

Satz 10.15 *Wenn SET COVER $o(\log_2 N)$-approximative Algorithmen für Universen der Größe N besitzt, dann kann jede Sprache $L \in \mathcal{NP}$ in Zeit $n^{O(\log_2 \log_2 n)}$ erkannt werden.*

In [F] wird sogar gezeigt, dass $(1 - o(1)) \cdot \ln N$-approximative Algorithmen unter der Annahme aus Satz 10.15 ausgeschlossen sind: Der Greedy Algorithmus 7.15 ist optimal!

10.4.4 Die Unique Games Vermutung

Wir betrachten eine eingeschränkte Version von LABEL COVER. Wiederum ist ein bipartiter Graph $G = (A \cup B, E)$ gegeben. Diesmal erlauben wir nur eine endliche Menge L von Markierungen für jeden Knoten, eine allerdings unerhebliche Einschränkung. Wie bisher sind „Bedingungen“ $f_{a,b} : L \to L$ für jede Kante $\{a, b\} \in E$ gegeben, wobei wir aber diesmal fordern, dass $f_{a,b}$ stets eine **Permutation** ist. Wir sagen, dass Eingaben von diesem Typ ein **eindeutiges Spiel** beschreiben.
Wir suchen Färbungen $f_A : A \to L$ und $f_B : B \to L$, so dass die Anzahl

$$\text{opt} = |\{e = \{a, b\} \in E \mid f_{a,b}(f_A(a)) = f_B(b)\}|$$

der Kanten, deren Endpunkte „konsistent“ gefärbt sind, möglichst groß ist. Wenn also eine Kante konsistent gefärbt ist, dann legt die Farbe des einen Endpunkts stets die Farbe des anderen Endpunkts eindeutig fest. Die Unique Games Vermutung besagt:

Offenes Problem 6
Für jedes $\delta > 0$ ist es $\mathcal{NP}$-hart zu entscheiden, ob opt $\geq (1-\delta) \cdot |E|$ oder opt $\leq \delta \cdot |E|$ für ein eindeutiges Spiel gilt.

Wenn die Unique Games Vermutung richtig ist, dann kann gezeigt werden, dass

- VERTEX COVER für kein $\varepsilon > 0$ effiziente $(2-\varepsilon)$-approximative Algorithmen besitzt.
- Weiterhin kann der beste, von effizienten Algorithmen für MAX-CUT wie auch für MAX-2SAT erreichbare Approximationsfaktor exakt bestimmt werden.

(In MAX-CUT ist ein ungerichteter Graph $G = (V, E)$ gegeben. Die Knotenmenge V ist so in zwei disjunkte Klassen aufzuteilen, dass die Anzahl „kreuzender“ Kanten maximal ist.)
Ist die Unique Games Vermutung richtig? Die Antwort ist natürlich nicht bekannt. Allerdings wurde gezeigt, dass das Entscheidungsproblem in Zeit $2^{n^{\mathrm{poly}(\delta)}}$ gelöst werden kann.

Kapitel 11

$\mathcal{NP} \subseteq \mathcal{PCP}(\text{poly}(n), O(1))$

Der Beweis von Satz 10.2 ist sehr umfangreich. Wir beschränken uns auf den Beweis des folgenden, deutlich schwächeren Resultats.

Lemma 11.1 $\mathcal{NP} \subseteq \mathcal{PCP}(\text{poly}(n), O(1))$.

Zwar ist diese Aussage für Anwendungen in der Approximationskomplexität zu schwach[1], aber trotzdem ist das Ergebnis überraschend: Ein Verifizierer mit unbeschränktem Zugang zu Zufallsbits kann sich von der Richtigkeit eines Beweises durch die Inspektion konstant vieler Beweisbits überzeugen!

Beweis von Lemma 11.1: Es genügt zu zeigen, dass die $\mathcal{NP}$-vollständige Sprache 3-SAT in $\mathcal{PCP}(\text{poly}(n), O(1))$ liegt. (Warum?)
Sei $\phi = \bigwedge_{j=1}^{n} k_j$ eine 3KNF Formel mit den Klauseln $k_1, \ldots, k_n$. Wir *arithmetisieren* die Klauseln k_j von ϕ, d.h. wir übersetzen Klauseln in Polynome. Dazu überführe das Literal x_i (bzw. $\neg x_i$) in den Ausdruck $(1 - x_i)$ (bzw. x_i) und ersetze die Boolesche Operation ODER durch die Multiplikation. Wir erhalten also für jede Klausel k_j ein Polynom p_j vom Grad drei, so dass für jede Belegung (a_1, a_2, a_3) der Variablen von k_j gilt:

$$(a_1, a_2, a_3) \text{ erfüllt } k_j \;\Leftrightarrow p_j(a_1, a_2, a_3) = 0.$$

ϕ ist offenbar genau dann erfüllbar, wenn es eine Belegung im Körper $\mathbb{Z}_2$ gibt, so dass die Folge $(p_j \mid 1 \leq j \leq n)$ *nach* Auswertung der einzelnen Polynome die Null-Folge ergibt. Zudem können wir im Körper $\mathbb{Z}_2$ effizient Null-Folgen entdecken, denn:

Behauptung 11.1 *Sei $v \in \mathbb{Z}_2^n$ vom Null-Vektor verschieden. Dann gilt*

$$prob_w[\sum_{i=1}^{n} v_i \cdot w_i \not\equiv 0 \ mod\ 2] = \frac{1}{2}.$$

[1]Beachte, dass exponentiell lange Beweise sich jetzt auszahlen, da der Verifier auf polynomiell viele Zufallsbits zugreifen kann und mit ihrer Hilfe in verschiedenen Berechnungen exponentiell viele Positionen nachfragen kann. Für logarithmisch viele Zufallsbits können hingegen nur polynomiell viele Positionen nachgefragt werden.

Beweis: Warum ist das innere Produkt (v, w) mit Wahrscheinlichkeit $\frac{1}{2}$ von Null verschieden? Sei O.B.d.A. $v_1 \neq 0$. Wir denken uns die Komponenten $w_2, \ldots, w_n$ bereits gewählt und beachten, dass dann $v_1 \cdot w_1 = \sum_{i=2}^{n} v_i \cdot w_i$ mit Wahrscheinlichkeit $\frac{1}{2}$ gilt. □

Sei $a = (a_1, \ldots, a_n)$ eine (nicht notwendigerweise erfüllende) Belegung von ϕ. Wir setzen $v = (p_j(a) \mid 1 \leq j \leq n)$. Unser Ziel ist die Überprüfung, ob a eine erfüllende Belegung ist, d.h. ob v der Nullvektor ist. Wir betrachten deshalb das innere Produkt $\langle v, w \rangle$ von v mit einem zufällig ausgewürfelten Vektor w. Die Komponenten von v sind an der Stelle a ausgewertete Polynome vom Grad drei und $\langle v, w \rangle$ lässt sich somit als eine Summe über $\mathbb{Z}_2$ von konstanten, linearen, quadratischen und kubischen Termen auffassen:

$$\langle v, w \rangle = c_w \oplus \sum_{i \in S_1(w)} a_i \oplus \sum_{(i,j) \in S_2(w)} a_i \cdot a_j \oplus \sum_{(i,j,k) \in S_3(w)} a_i \cdot a_j \cdot a_k. \tag{11.1}$$

Beachte, dass die Mengen $S_1(w), S_2(w)$ und $S_3(w)$ nur von dem zufälligen Vektor w und von ϕ abhängen, nicht aber von der gewählten Belegung a; gleiches gilt auch für das Bit c_w. Insbesondere kann ein Verifier die drei Mengen effizient berechnen, da er Zugang zu w und ϕ hat.

Definition 11.2 Für Vektoren $x \in \mathbb{Z}_2^a$ und $y \in \mathbb{Z}_2^b$ definieren wir das Tensorprodukt von x und y als den Vektor $x \otimes y \in \mathbb{Z}_2^{a \cdot b}$ mit

$$(x \otimes y)_{i,j} = x_i \cdot y_j.$$

Wenn wir statt der Mengen $S_1(w)$, $S_2(w)$ und $S_3(w)$ ihre Inzidenzvektoren $\alpha_1(w), \alpha_2(w)$ und $\alpha_3(w)$ wählen, dann erhalten wir die zu (11.1) äquivalente Darstellung

$$\langle v, w \rangle = c_w \ \oplus \ \langle\, a, \alpha_1(w) \,\rangle \ \oplus \ \langle\, a \otimes a, \alpha_2(w) \,\rangle \ \oplus \ \langle\, a \otimes (a \otimes a), \alpha_3(w) \,\rangle.$$

Es liegt also nahe, einen Beweis zu wählen, aus dem die Vektoren

$$\begin{aligned} A &= (\, (a, x) \mid x \in \mathbb{Z}_2^n \,), \quad B = (\, (a \otimes a, y) \mid y \in \mathbb{Z}_2^{n^2} \,) \quad \text{und} \\ C &= (\, (a \otimes (a \otimes a), z) \mid z \in \mathbb{Z}_2^{n^3} \,) \end{aligned}$$

abgelesen werden können.

Aufgabe 118
Zeige, dass $A(x) \cdot A(y) = B(x \otimes y)$ wie auch $A(u) \cdot B(v) = C(u \otimes v)$ für alle Vektoren $x, y, u \in \mathbb{Z}_2^n$ und $v \in \mathbb{Z}_2^{n^2}$ gilt.
Diese Eigenschaft suggeriert bereits eine Überprüfung der Tensoreigenschaften von B und C durch die zufällige Wahl der Vektoren $x, y, u \in \mathbb{Z}_2^n$ und $v \in \mathbb{Z}_2^{n^2}$.

Wenn A, B bzw. C jeweils diesen linearen Funktionen entsprechen, dann programmieren wir den Verifier wie folgt.

Algorithmus 11.3 Erfüllbarkeitstest für eine 3KNF Formel ϕ.
Sei a eine nicht notwendigerweise erfüllende Belegung von ϕ. Der Beweis bestehe aus Funktionstabellen für die linearen Funktionen

$$A(x) = \langle a, x \rangle, \quad B(y) = \langle a \otimes a, y \rangle \ \text{ und } \ C(z) = \langle a \otimes a \otimes a, z \rangle.$$

(1) Der Verifier wählt einen zufälligen Vektor $w \in \mathbb{Z}_2^n$.

(2) Der Verifier berechnet c_w, $S_1(w), S_2(w)$ und $S_3(w)$ aus w und ϕ.

(3) Der Verifier fordert die Bits $\sum_{i \in S_1(w)} a_i, \sum_{(i,j) \in S_2(w)} a_i \cdot a_j$ und $\sum_{(i,j,k) \in S_3(w)} a_i \cdot a_j \cdot a_k$ an und akzeptiert genau dann, wenn

$$c_w \oplus \sum_{i \in S_1(w)} a_i \oplus \sum_{(i,j) \in S_2(w)} a_i \cdot a_j \oplus \sum_{(i,j,k) \in S_3(w)} a_i \cdot a_j \cdot a_k = 0.$$

Wenn $a = (a_1, \ldots, a_n)$ eine erfüllende Belegung von ϕ ist, dann wird der Verifier stets akzeptieren und anderenfalls mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ verwerfen. Allerdings haben wir bisher vorausgesetzt, dass

(a) A die Wertetabelle einer linearen Funktionen ist und dass

(b) $B = A \otimes A$ sowie $C = A \otimes B$ gilt.

Wir müssen jetzt garantieren, dass der Verifier einen Beweis mit hoher Wahrscheinlichkeit ablehnt, falls der Beweis gegen eine der beiden Anforderungen (a) oder (b) verstößt. Wir betrachten zuerst einen Verstoß gegen die Anforderung (b).

Algorithmus 11.4 Test auf Tensor-Eigenschaft

(1) Der Verifier wählt zufällige Vektoren $x, y \in \mathbb{Z}_2^n$ und fragt die Werte $A(x)$ und $A(y)$ von A an den Stellen x und y nach. Ebenso wird der Wert $B(x \otimes y)$ von B an der Stelle $x \otimes y$ nachgefragt. Der Verifier verwirft, falls

$$A(x) \cdot A(y) \neq B(x \otimes y).$$

(2) Der Verifier wählt zufällige Vektoren $u \in \mathbb{Z}_2^n, v \in \mathbb{Z}_2^{n^2}$ und fragt die Werte $A(u)$ und $B(v)$ wie auch den Wert $C(u \otimes v)$ nach. Der Verifier verwirft, falls

$$A(u) \cdot B(v) \neq C(u \otimes v)$$

und akzeptiert ansonsten.

Wir nehmen zuerst an, dass $B = A \otimes A$ und dass $C = A \otimes B$. Beachte, dass in diesem Fall $A(x) \cdot A(y) = B(x \otimes y)$ wie auch $A(u) \cdot B(v) = C(u \otimes v)$ und der Verifier akzeptiert richtigerweise. Im verbleibenden Fall ist die Tensoreigenschaft verletzt und die folgende Behauptung weist eine nur kleine Fehlerwahrscheinlichkeit nach.

Behauptung 11.2 *A, B und C seien die Wertetabellen von lineare Funktionen $\langle a, s_1 \rangle : \mathbb{Z}_2^n \to \mathbb{Z}_2$, $\langle b, s_2 \rangle : \mathbb{Z}_2^{n^2} \to \mathbb{Z}_2$ und $\langle c, s_3 \rangle : \mathbb{Z}_2^{n^3} \to \mathbb{Z}_2$. Es gelte $b \neq a \otimes a$ oder $c \neq a \otimes b$. Dann akzeptiert der Tensor Test mit einer Wahrscheinlichkeit von höchstens $\frac{3}{4}$. Eine k-malige Wiederholung führt auf eine Akzeptanzwahrscheinlichkeit von höchstens $(\frac{3}{4})^k$.*

Beweis von Behauptung 11.2: Wir weisen nach, dass der Tensor Test mit Wahrscheinlichkeit mindestens $\frac{1}{4}$ verwirft. Dazu nehmen wir zuerst an, dass $B \neq A \otimes A$. Nach Annahme ist A die Wertetabelle der linearen Funktion $\langle a, s_1 \rangle : \mathbb{Z}_2^n \to \mathbb{Z}_2$ und B ist die Wertetabelle der linearen Funktion $\langle b, s_2 \rangle : \mathbb{Z}_2^{n^2} \to \mathbb{Z}_2$. Wir beachten

$$A(x) \cdot A(y) = \langle a, x \rangle \cdot \langle a, y \rangle = \sum_{i,j}^{n} a_i \cdot x_i \cdot a_j \cdot y_j = x^T \cdot (a_i \cdot a_j)_{i,j} \cdot y$$

und

$$B(x \otimes y) = \langle b, x \otimes y \rangle = \sum_{i,j} x_i \cdot b_{i,j} \cdot y_j = x^T \cdot (b_{i,j})_{i,j} \cdot y.$$

Da wir $B \neq A \otimes A$ angenommen haben, ist die Matrix $(a_i \cdot a_j)_{i,j} - (b_{i,j})_{i,j}$ nicht die Nullmatrix. Ihr Kern hat somit die Dimension höchstens $n-1$ und ein zufällig gewählter Vektor gehört mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ nicht zum Kern. Es ist also

$$\text{prob}[(a_i \cdot a_j)_{i,j} \cdot y = (b_{i,j})_{i,j} \cdot y] \leq \frac{1}{2}$$

Andererseits folgt

$$\text{prob}[\ x^T \cdot (a_i \cdot a_j)_{i,j} \cdot y = x^T \cdot (b_{i,j})_{i,j} \cdot y \mid (a_i \cdot a_j)_{i,j} \cdot y \neq (b_{i,j})_{i,j} \cdot y\ ] = \frac{1}{2}$$

bei fest gewähltem Vektor y mit Behauptung 11.1. Damit wird also der Tensor Test in diesem Fall mit einer Wahrscheinlichkeit von mindestens $\frac{1}{4}$ verwerfen. Ein analoges Argument weist dieselbe Fehlerwahrscheinlichkeit auch im Falle $c \neq a \otimes b$ nach. □

Wir behandeln die letzte vorhandene Verstoß-Möglichkeit, nämlich den Fall, dass A nicht die Wertetabelle einer linearen Funktionen ist. Tatsächlich können wir mit konstant vielen Anfragen Verstöße gegen die Linearität im Allgemeinen nicht feststellen. Können wir aber zumindest schwere Verstöße gegen die Linearität feststellen?

Definition 11.5 Seien $f_1, f_2 : \mathbb{Z}_2^m \to \mathbb{Z}_2$ vorgegeben und sei $\delta \in [0,1]$. Dann heißen f_1 und f_2 δ-nahe, falls

$$\mid \{x \in \mathbb{Z}_2^m \mid f_1(x) \neq f_2(x)\} \mid \leq \delta \cdot 2^m.$$

f_1 heißt δ-linear, falls es eine lineare Funktion f_2 gibt, so dass f_1 und f_2 δ-nahe sind.

Die Algorithmen 11.3 und 11.4 fragen konstant viele Funktionswerte der Funktion A nach. Wenn aber die im Beweis repräsentierte Funktion δ-linear ist und wenn $\frac{1}{\delta}$ wesentlich größer als die Anzahl der Anfragen ist, dann nehmen unsere Protokolle die Funktionen „als linear wahr“. Das folgende Protokoll wird die δ-Linearität bei genügend häufiger Wiederholung mit hoher Wahrscheinlichkeit richtig feststellen.

Algorithmus 11.6 Test für δ-Linearität.

Die Eingabe bestehe aus einer Funktion $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$.

(1) Der Verifier wählt zufällige Vektoren $x, y \in \mathbb{Z}_2^n$.

(2) Der Verifier fragt nach den Funktionswerten $f(x), f(y)$ und $f(x \oplus y)$ und akzeptiert genau dann, wenn $f(x) \oplus f(y) = f(x \oplus y)$.

Behauptung 11.3 *Sei $\delta < \frac{1}{3}$ und für die Funktion $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ gelte*

$$\text{prob}_{x,y}[\ f(x) \neq f(x \oplus y) \oplus f(y)\ ] \leq \frac{\delta}{2}. \tag{11.2}$$

Dann ist f δ-linear.

Beweisskizze von Behauptung 11.3: Wir müssen zeigen, dass f δ-linear ist und konstruieren deshalb eine lineare Funktion g, die δ-nahe zu f ist. Wir setzen

$$g(x) = b \quad \Leftrightarrow \quad \text{für mindestens die Hälfte aller } y \in \mathbb{Z}_2^n \text{ ist } f(x \oplus y) \oplus f(y) = b.$$

Wir zeigen zuerst, dass f und g δ-nahe sind. Nach Definition von g gilt

$$\text{prob}_y[\ g(x) = f(x \oplus y) \oplus f(y)\ ] \geq \frac{1}{2} \tag{11.3}$$

für jedes $x \in \mathbb{Z}_2^n$. Für jedes x mit $g(x) \neq f(x)$ folgt deshalb

$$\text{prob}_y[\ f(x) \neq f(x \oplus y) \oplus f(y)\ ] \geq \frac{1}{2}$$

und wir erhalten

$$\text{prob}_{x,y}[\ f(x) \neq f(x \oplus y) \oplus f(y)\ ] > \frac{\delta}{2},$$

wenn f und g sich auf mehr als $\delta \cdot 2^n$ Vektoren unterscheiden. Da dies der Annahme der Behauptung widerspricht, sind f und g also δ-nahe. Um die Linearität von g nachzuweisen, zeigen wir zuerst, dass die Abschätzung (11.3) verbessert werden kann. Wir behaupten nämlich, dass sogar

$$p_x = \text{prob}_y[\ g(x) = f(x + y) \oplus f(y)\ ] \geq 1 - \delta \tag{11.4}$$

für jedes $x \in \mathbb{Z}_2^n$ gilt. Zum Nachweis fixieren wir x und erhalten zuerst

$$\text{prob}_{y,z}[\ f(x \oplus y \oplus z) \neq f(x \oplus y) \oplus f(z)\ ], \text{prob}_{y,z}[\ f(x \oplus y \oplus z) \neq f(x \oplus z) \oplus f(y)\ ] \leq \frac{\delta}{2}$$

aus der Annahme der Behauptung. Mit Wahrscheinlichkeit mindestens $1-\delta$ gilt also $f(x \oplus y) \oplus f(z) = f(x \oplus y \oplus z) = f(x \oplus z) \oplus f(y)$ und als Konsequenz ist

$$\begin{aligned}
1 - \delta &\leq \text{prob}_{y,z}[\ f(x \oplus y) \oplus f(z) = f(x \oplus z) \oplus f(y)\ ] \\
&= \sum_{b \in \{0,1\}} \text{prob}_y[\ b = f(x \oplus y) \oplus f(y)\ ] \cdot \text{prob}_z[\ b = f(x \oplus z) \oplus f(z)\ ] \\
&= \sum_{b \in \{0,1\}} \text{prob}_y[\ b = f(x \oplus y) \oplus f(y)\ ]^2 \\
&= p_x^2 + (1 - p_x)^2 \\
&\leq p_x^2 + p_x \cdot (1 - p_x) \qquad\qquad \text{denn } p_x \geq \tfrac{1}{2} \\
&= p_x
\end{aligned}$$

und (11.4) folgt. Die Behauptung des Lemmas ergibt sich jetzt aus der folgenden Übungsaufgabe. □

Aufgabe 119
Zeige mit Hilfe von (11.4), dass g linear ist.
Hinweis: In der Behauptung wird $\delta < \frac{1}{3}$ gefordert.

Durch genügend häufige Wiederholung von Algorithmus 11.6 können wir also den Grad der Linearität entsprechend steigern. Die Strategie des Verifiers lässt sich jetzt vollständig beschreiben.

Algorithmus 11.7 Überprüfung der Erfüllbarkeit einer Formel ϕ.

Sei a eine nicht notwendigerweise erfüllende Belegung von ϕ. Der Verifier nimmt an, dass der Beweis aus Funktionstabellen für die linearen Funktionen

$$A(x) = \langle a, x\rangle, \quad B(y) = \langle a \otimes a, y\rangle \text{ und } C(z) = \langle a \otimes a \otimes a, z\rangle$$

besteht. Der Verifier muss die Richtigkeit dieser Annahme überprüfen.

(1) Die Linearität der Funktionen A, B und C wird mit Algorithmus 11.6 überprüft.

(2) Die Tensor-Struktur der Funktionen B und C wird mit Algorithmus 11.4 überprüft.

(3) Der Verifier akzeptiert genau dann, wenn Algorithmus 11.3 akzeptiert. Mit anderen Worten:

- Der Verifier wählt einen zufälligen Vektor $w \in \mathbb{Z}_2^n$ und berechnet c_w, $S_1(w), S_2(w)$ und $S_3(w)$ aus w und ϕ.
- Er fordert die Bits

$$\sum_{i \in S_1(w)} a_i, \quad \sum_{(i,j) \in S_2(w)} a_i \cdot a_j, \quad \sum_{(i,j,k) \in S_3(w)} a_i \cdot a_j \cdot a_k$$

  an und
- akzeptiert genau dann, wenn

$$c_w \oplus \sum_{i \in S_1(w)} a_i \oplus \sum_{(i,j) \in S_2(w)} a_i \cdot a_j \oplus \sum_{(i,j,k) \in S_3(w)} a_i \cdot a_j \cdot a_k = 0$$

  gilt.

Wir können jetzt dn Beweis von Lemma 11.1 abschließen. Algorithmus 11.7 arbeitet in polynomieller Zeit. Seine Korrektheit folgt aus der Korrektheit der aufgerufenen Algorithmen. Da jeder dieser aufgerufenen Algorithmen nur konstant viele Beweisbits nachfragt, werden somit insgesamt nur konstant viele Beweisbits nachgefragt: 3-SAT liegt in $\mathcal{PCP}(\text{poly}(n), O(1))$. □

Bemerkung 11.1 Beachte, dass der „erfolgreiche“ Beweis aus den drei Funktionstabellen besteht. Dieser Beweis hat exponentielle Länge!

Literaturverzeichnis

[Aro1] S. Arora, Approximation schemes for $\mathcal{NP}$-hard geometric optimization problems: a survey, *unveröffentlichtes Manuskript*, auf der Webseite von Sanjeev Arora publiziert, 2002.

[Aro2] S. Arora, Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems, *Journal of the ACM* 45 (5), pp. 753Ű782, 1998.

[AB] Sanjeev Arora, Boaz Barak, Computational Complexity: A Modern Approach, Cambridge University Press, 2009.

[ARR] S. Arora, P. Raghavan und S. Rao, Approximation schemes for euclidean k-medians and related problems, *Proceedings of the 30th ACM Symposium on Theory of Computing*, pp. 106-113, 1998.

[ACh] D. Avis und V. Chvátal, Notes on Bland's Pivoting Rule, *Mathematical Programming Study* 8, pp. 23–34, 1978.

[ACGK] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela und M. Protasi, Complexity and Approximation, Problems and Their Approximability Properties, *Springer-Verlag*, 1999.

[BR05] R. Bar-Yehuda and D. Rawitz, On the equivalence between the primal-dual schema and the local ratio technique, *SIAM J. Discrete Math*, vol 19 (3), pp. 762-797, 2005.

[BSS92] M. Bazaara, D. Sherali und C. Shetty, Nonlinear Programming: Theory and Algorithms, *Wiley*, 1992.

[BS90] R.B. Boppana und M. Sipser, The Complexity of finite Functions, in: J. van Leeuven (Ed.), Handbook of Theoretical Computer Science A, chapter 14, pp. 757-804, *Elsevier Science Publishers B.V.*, 1990.

[CS00] N. Christianini und J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, *Cambridge University Press*, 2000.

[F] U. Feige, A threshold of $\ln n$ for approximating set cover, *Journal of the ACM* 45 (4), pp. 634Ű652, 1998.

[FK] U. Feige und J. Kilian, Zero knowledge and the chromatic number, *Proceedings of the 11th Annual Conference on Complexity Theory*, 1996.

[Fre] R.M. Freund, Polynomial-Time Algorithms for Linear Programming based only primal Scaling and Project Gradients of a Potential Function, *Mathematical Programming*, Band 51, pp. 203-222, 1991.

[Gus] D. Gusfield, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, *Cambridge University Press*, 1999.

[Ha1] J. Hastad, Clique is hard to approximate within $n^{1-\varepsilon}$, *Proceedings of the 37th Annual IEEE Symposium on the Foundations of Computer Science*, pp. 627-636, 1996.

[Ha2] J. Hastad, Some optimal inapproximability results, *Proceedings of the 28th ACM Symposium on Theory of Computing*, pp. 1-10, 1997.

[Hoch] D. Hochbaum, Approximation Algorithms for $\mathcal{NP}$-Hard Problems, *PWS Publishing Company*, Boston, 1997.

[Joh] D.S. Johnson, Local Optimization and the Traveling Salesman Problem, *18.th International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, Band 443, Springer-Verlag, Berlin/Heidelberg, pp. 446-461, 1991.

[JAGS] D.S. Johnson, C.R. Aragon, L.A. McGeoch und C. Schevon, Simulated Annealing: An experimental Evaluation, Part I: Graph Partioning, *Operation Research* vol (37) nr. 6, pp. 865-892, 1989.

[Kar] H. Karloff, Linear Programming, *Progress in Theoretical Computer Science*, Birkhäuser, Boston, 1991.

[KZw] H. Karloff und U. Zwick, A $\frac{7}{8}$-approximation for Max 3Sat, *Proceedings of the 38th Annual IEEE Symposium on the Foundations of Computer Science*, 1997.

[KMS] S. Khanna, R. Motwani, M. Sudan und U. Vazirani, On syntactic versus computational views of approximability, *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science*, pp. 819-836, 1994.

[Karm] N. Karmarkar A new Polynomial-Time Algorithm for linear programming, *Combinatorica* 4, pp. 373–395, 1984.

[KK82] N. Karmarkar und R. Karp, An efficient approximation scheme for the one-dimensional bin packing problem, *Proc. 23rd Annual Symposium Foundation Computer Science*, pp. 312-320, 1982.

[Klee] V. Klee und G.J. Minty, How good is the Simplex Algorithms?, in O. Sisha (Hrsg.): *„Inequalities“, Band III*, Academic Press, New York, pp. 159–175, 1972.

[KV05] B. Korte und J. Vygen, Combinatorial Optimization: Theory and Algorithms, 3rd edition, Springer Verlag, 2005.

[M97] T. M. Mitchell, Machine Learning, *McGraw-Hill*, 1997.

[PaSt] C.H. Papadimitriou und K. Steiglitz, Combinatorical Optimization: Algorithms and Complexity, Prentice-Hall, Eaglewood Cliffs, New Jersey, 1982.

[PaYa] C.H. Papadimitriou und M. Yannakakis, The traveling salesman problem with distances one and two, *Math. Oper. Res.* 18, pp. 1-11, 1993.

[Schr] A. Schrijver, Theory of Linear and Integer Programming, *Wiley-Interscience Series in discrete Mathematics and Optimization*, John Wiley & Son Ltd, 1986.

[Tre] L. Trevisan, When Hamming meets Euclid: the approximability of geometric TSP and MST, *Proceedings of the 29th ACM Symposium on Theory of Computing*, pp. 21-39, 1997.

[Van] R.J. Vanderbei, Linear Programming: Foundations and Extensions, *Kluwer Academic Publishers*, 2001.

[Vaz] V.V. Vazirani, Approximation Algorithms, *Springer Verlag* 2001.

[Wols] L.A. Wolsey, Integer Programming, *Wiley*, 1998.

[Ye1] Y. Ye, Potential Reduction Algorithm for Linear Programming, *Mathematical Programming*, Band 51, pp. 239-258, 1991.

[Ye2] Y. Ye, Interior Point Algorithms, *Wiley*, 1997.