

Lineare und kombinatorische Optimierung

Dieter Weninger
Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Mathematik
Lehrstuhl für Wirtschaftsmathematik

Wintersemester 2017/2018

Inhaltsverzeichnis

Inhaltsverzeichnis	2
I Kombinatorische Optimierung	6
1 Einleitung	7
2 Grundlagen der Graphentheorie	14
2.1 Grundlegende Begriffe	14
2.2 Exkurs: Algorithmen	22
2.3 Speicherung von Graphen	25
2.4 Suchalgorithmen auf Graphen	28
2.4.1 Breitensuche	28
2.4.2 Tiefensuche	30
3 Sortier-Algorithmen	33
3.1 Selectionsort	33
3.2 Bubblesort	34
3.3 Quicksort	35
3.4 Heapsort	37
3.5 Bucketsort	43
4 Bäume und Wälder	45
4.1 Grundlegende Definitionen	45
4.2 Maximale Wälder minimalen Gewichts und minimale Spann- bäume	46
5 Kürzeste Wege	54
5.1 Eigenschaften kürzester Wege	54
5.2 Digraphen mit nicht-negativen Gewichten	56
5.3 Digraphen mit beliebigen Gewichten	58
5.3.1 Behandlung negativer Kreise	63
5.4 Kürzeste Wege zwischen allen Knotenpaaren	64
5.5 Dynamische Programmierung	66

6	Maximale Flüsse	69
6.1	Das Max-Flow-Min-Cut-Theorem	70
6.2	Augmentierende-Wege-Algorithmus	73
6.3	Ein allgemeiner Push-Relabel-Algorithmus	77
7	Minimalkosten-Fluss-Probleme	86
7.1	Die Negative-Kreise-Optimalitätsbedingung	88
7.2	Der Kreis-Löschungs-Algorithmus	89
8	Unabhängigkeitssysteme und Matroide	92
8.1	Unabhängigkeitssysteme	92
8.2	Matroide	95
8.2.1	Kreisaxiome und Basisaxiome	97
8.2.2	Dualität und Isomorphie	99
8.2.3	Greedy-Algorithmus	102
8.2.4	Schnitt von Matroiden	105
II	Lineare Optimierung und der Simplex-Algorithmus	108
9	Das Simplex-Verfahren	109
9.1	Lineare Optimierungsprobleme mit Gleichungsrestriktionen .	114
9.2	Das Simplex-Verfahren für lineare Optimierungsprobleme in Standardform	116
9.3	Die Standardform ist keine Einschränkung	123
9.3.1	Minimieren vs. Maximieren	124
9.3.2	Lineare Optimierungsprobleme in allgemeiner Form . .	124
10	Geometrische Aspekte der linearen Optimierung	128
10.1	Polyeder und Polytope	128
10.2	Konvexität, Extremalmengen und Ecken von Polyedern	131
10.3	Basislösungen und Optimierung	132
11	Dualität	136
11.1	Untere Schranken für optimale Zielfunktionswerte eines linearen Problems	136
11.2	Das duale Problem	138
11.3	Das Farkas-Lemma	140
11.4	Der Dualitätssatz der linearen Optimierung	144
11.5	Komplementärer Schlupf	150
11.6	Dualität und das Simplex-Verfahren	153
11.7	Sensitivitätsanalyse I	155
11.8	Eine ökonomische Interpretation von Dualvariablen	156
12	Ein zweiter Blick auf das Simplex-Verfahren	158

12.1	Terminiertheit	159
12.1.1	Ein Negativ-Beispiel	159
12.1.2	Kreiseln	163
12.1.3	Die Regel von Bland	163
12.2	Die Effizienz der Simplex-Methode	165
12.3	Die Phase-1 des Simplex-Verfahrens	166
13	Der duale Simplex-Algorithmus	170
14	Sensitivitätsanalyse II	177
III	Exkurs: Gemischt-ganzzahlige lineare Optimierung	181
15	Branch-and-Bound	182
	Abbildungsverzeichnis	186
	Tabellenverzeichnis	188
	Algorithmenverzeichnis	189

Vorwort

Dieses Skript basiert auf den Skripten zur gleichnamigen Vorlesung von Prof. Dr. Alexander Martin und Prof. Dr. Martin Schmidt vergangener Jahre an der Technischen Universität Darmstadt und der Friedrich-Alexander-Universität Erlangen-Nürnberg.

Obgleich das Skript also schon eine gewisse Erprobung erfahren hat, sind Fehler nicht ausgeschlossen. Denken Sie daher beim Nacharbeiten der Veranstaltung mithilfe des Skripts kritisch nach.

Wenn Sie Fehler finden oder Verbesserungsvorschläge haben, teilen sie Sie mir bitte mit.

Dieter Weninger

Erlangen, Wintersemester 2017/2018

Teil I

Kombinatorische Optimierung

Kapitel 1

Einleitung

Die mathematische Optimierung beschäftigt sich damit, Minima und Maxima einer Funktion f über einer Menge Ω zu finden. Aus der Analysis ist mit dem Satz von Weierstraß bekannt, dass eine stetige Funktion über einer kompakten Teilmenge des \mathbb{R}^n ihr Minimum und ihr Maximum (in Punkten den x_{\min} und x_{\max}) annimmt. Dieser Satz ist ein reiner Existenzsatz. Er sagt nichts darüber aus, wie man die Punkte x_{\min} und x_{\max} berechnen kann. Die Optimierung im weitesten Sinn beschäftigt sich mit dem Problem der Bestimmung dieser Punkte. Die Funktion, deren Minimum oder Maximum gefunden werden soll, nennen wir *Zielfunktion* und die Menge Ω heißt *zulässige Menge*. Die Elemente $x \in \Omega$ heißen *zulässige Punkte* oder *zulässige Lösungen*. Die zulässige Menge kann der ganze Raum \mathbb{R}^n sein (dann spricht man von *unrestringierter Optimierung* bzw. von *Optimierung ohne Nebenbedingungen*) – sie kann aber auch eine Teilmenge des Raums sein, die durch sog. *Nebenbedingungen* beschrieben wird. In diesem Skript werden zwei verschiedene Schreibweisen für ein Minimierungsproblem verwendet:

$$\min \{f(x) : x \in \Omega\}$$

bzw.

$$\begin{array}{ll} \min_x & f(x) \\ \text{s.t.} & x \in \Omega. \end{array}$$

Beides bedeutet dasselbe: Wir suchen das Minimum der Funktion f über der zulässigen Menge Ω . Die Abkürzung “s.t.” steht für das englische “subject to”, was soviel bedeutet wie “unter den Nebenbedingungen”. Oft ist die zulässige Menge durch Gleichungen und Ungleichungen beschrieben, es gilt also

$$\Omega = \{x \in \mathbb{R}^n : g(x) = 0, h(x) \geq 0\}$$

mit Funktionen $g : \mathbb{R}^n \rightarrow \mathbb{R}^k$, $h : \mathbb{R}^n \rightarrow \mathbb{R}^m$. Die Menge der Punkte aus Ω , in denen das Minimum angenommen wird, bezeichnen wir mit

$$\arg \min(f, \Omega).$$

Formal bedeutet dies

$$\alpha = \min \{f(x) : x \in \Omega\} \iff \arg \min(f, \Omega) = \{x \in \Omega : f(x) = \alpha\}.$$

Beispiel 1. Wir suchen das Minimum der Funktion

$$f : \mathbb{R} \rightarrow \mathbb{R}, \quad f(x) = x^3 - 7.5x^2 + 18x - 10.5$$

über der Menge aller Punkte, die die beiden Nebenbedingungen

$$\begin{aligned} -x + 1 &\leq 0, \\ x^2 - 5x &\leq 0 \end{aligned}$$

erfüllen. Die Zielfunktion in diesem Beispiel ist also

$$f(x) = x^3 - 7.5x^2 + 18x - 10.5$$

und die zulässige Menge ist die Menge

$$\Omega = \{x \in \mathbb{R} : x - 1 \geq 0, x^2 - 5x \leq 0\} = [1, 5].$$

Formal wird das Optimierungsproblem demnach geschrieben als

$$\min \{x^3 - 7.5x^2 + 18x - 10.5 : x - 1 \geq 0, x^2 - 5x \leq 0\}$$

oder auch

$$\begin{aligned} \min_x \quad & x^3 - 7.5x^2 + 18x - 10.5 \\ \text{s.t.} \quad & x - 1 \geq 0, \\ & x^2 - 5x \leq 0. \end{aligned}$$

Siehe Abbildung 1.1 für eine Skizze. Das Minimum wird im Punkt $x = 1$ angenommen, d. h., $\arg \min(f, \Omega) = \{1\}$, der zugehörige Funktionswert ist $f(1) = 1$. \triangle

In der Grafik ist außerdem ein Phänomen zu sehen, das sehr oft beobachtet werden kann: Es gibt ein sog. *lokales Minimum* im Punkt $x = 3$. Diese Beobachtung führt uns zur folgenden Definition.

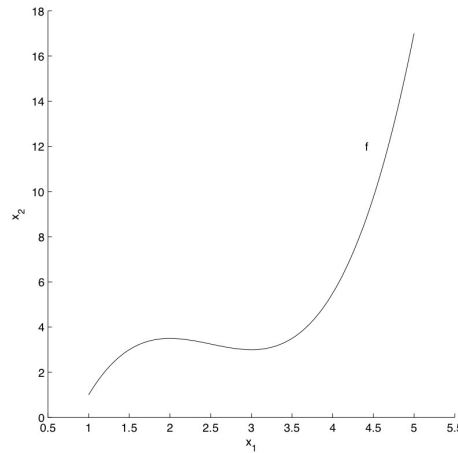


Abbildung 1.1: Die Funktion $f(x) = x^3 - 7.5x^2 + 18x - 10.5$ im Intervall $[1, 5]$

Definition 1.1 (Lokale und global Optima). Ein Punkt $\bar{x} \in \Omega$ heißt *lokaler Minimalpunkt* (oder *lokales Minimum*) der Funktion f über der Menge Ω , wenn eine offene Umgebung $\mathcal{U}(\bar{x})$ von \bar{x} existiert, so dass

$$f(\bar{x}) \leq f(x) \quad \forall x \in \mathcal{U}(\bar{x}) \cap \Omega$$

gilt. Der Punkt $\bar{x} \in \Omega$ heißt *globaler Minimalpunkt* (oder *globales Minimum*) der Funktion f über der Menge Ω , wenn

$$f(\bar{x}) \leq f(x) \quad \forall x \in \Omega$$

gilt. *Lokale und globale Maximalpunkte* (bzw. *lokale und globale Maxima*) sind analog definiert.

Eine weitere wichtige Beobachtung ist die, dass jedes Maximierungsproblem auch als Minimierungsproblem geschrieben werden kann. Dabei nutzt man aus, dass

$$\max \{f(x) : x \in \Omega\} = - \min \{-f(x) : x \in \Omega\}$$

gilt. Statt also ein Maximierungsproblem $\max\{f(x) : x \in \Omega\}$ zu lösen, kann man auch das Minimierungsproblem

$$\min \{-f(x) : x \in \Omega\}$$

lösen. Um schließlich den korrekten Wert des eigentlich gesuchten Maximums zu erhalten, muss die Lösung des Minimierungsproblems noch mit -1 multipliziert werden.

Beispiel 2 (Fortsetzung von Beispiel 1). Wir suchen nun das Maximum der Funktion

$$f(x) = x^3 - 7.5x^2 + 18x - 10.5$$

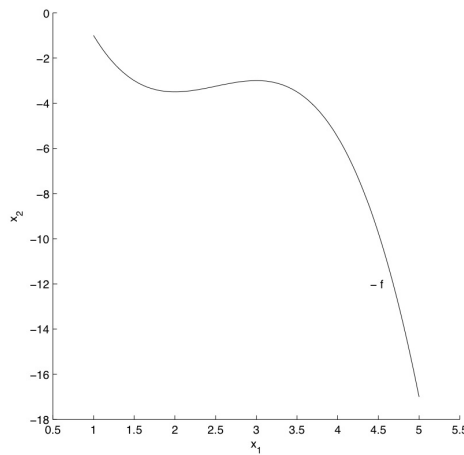


Abbildung 1.2: Die Funktion $-f(x) = -(x^3 - 7.5x^2 + 18x - 10.5)$ im Intervall $[1, 5]$

über der Menge

$$\Omega = \{x \in \mathbb{R}: x - 1 \geq 0, x^2 - 5x \leq 0\} = [1, 5].$$

Wie man in Abbildung 1.1 sieht, wird das Maximum im Punkt $x = 5$ angenommen und der zugehörige Zielfunktionswert ist $f(5) = 17$. Bestimmen wir dieses Maximum über den Umweg des Minimierungsproblem, so erhalten wir

$$\begin{aligned} & \max \{x^3 - 7.5x^2 + 18x - 10.5: x - 1 \geq 0, x^2 - 5x \leq 0\} \\ &= - \min \{-(x^3 - 7.5x^2 + 18x - 10.5): x - 1 \geq 0, x^2 - 5x \leq 0\}. \end{aligned}$$

Grafisch ist dies in Abbildung 1.2 dargestellt. Das Minimum von $-f$ über $[1, 5]$ wird im Punkt $x = 5$ angenommen, der zugehörige Funktionswert ist $f(5) = -17$. Um auf das Maximum der ursprünglichen Funktion zu kommen, multiplizieren wir jetzt -17 mit -1 und erhalten 17 . \triangle

Je nachdem, welche Form die Zielfunktion und die zulässige Menge haben, ist ein Optimierungsproblem verschieden schwer zu lösen. Der folgende Überblick soll eine grobe Einteilung von Optimierungsproblemen bieten.

Lineare Optimierungsprobleme

Von einem *linearen Optimierungsproblem*, auch *lineares Problem (LP)* genannt, spricht man, wenn sowohl die Zielfunktion f als auch die Nebenbedingungen $g_i, i = 1, \dots, k$, und $h_j, j = 1, \dots, m$, (affin-)lineare Funktionen vom

\mathbb{R}^n nach \mathbb{R} sind. Ein lineares Optimierungsproblem in *Standardform* ist von der Gestalt:

$$\min \quad c^\top x \quad \text{s.t.} \quad Ax = b, \quad x \geq 0 \quad (1.1)$$

mit $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ und $x \in \mathbb{R}^n$.¹ Wie man sich denken kann, ist dies die einfachste Klasse von Optimierungsproblemen. Es gibt eine Reihe von Algorithmen zur Lösung von LPs wie zum Beispiel den Simplex-Algorithmus oder Innere-Punkte-Verfahren. Verschiedene Varianten des Simplex-Algorithmus werden wir im Rahmen dieser Vorlesung kennenlernen.

Beispiel 3 (Ein Transportproblem). Ein Chemieunternehmen hat m Fabriken F_1, \dots, F_m und r Verkaufsstellen V_1, \dots, V_r . Jede Fabrik F_i kann pro Woche a_i Tonnen eines gewissen chemischen Produkts herstellen. Die Größe a_i heißt *Kapazität* der Fabrik F_i . Jede Verkaufsstelle V_j hat einen bekannten wöchentlichen *Bedarf* von b_j Tonnen des Produkts. Die Kosten, um eine Tonne des Produkts von Fabrik F_i an Verkaufsstelle V_j zu transportieren, bezeichnen wir mit $c_{ij} \geq 0$.

Informell lässt sich das zu lösende Problem wie folgt beschreiben: Welche Menge des Produkts muss man von jeder Fabrik zu jeder Verkaufsstelle transportieren, so dass die Kapazitäten der Fabriken eingehalten werden, der Bedarf aller Verkaufsstellen gedeckt ist und die resultierenden Kosten minimal sind?

Um diese Frage mithilfe der mathematischen Optimierung zu beantworten müssen wir das Problem zunächst modellieren. Es seien dazu $x_{ij} \geq 0$ für $1 \leq i \leq m$ und $1 \leq j \leq r$ die Zahl der Tonnen, die von F_i nach V_j transportiert werden. Dann kann man das Problem wie folgt formulieren:

$$\begin{aligned} \min \quad & \sum_{i=1}^m \sum_{j=1}^r c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^r x_{ij} \leq a_i, & i = 1, \dots, m, \\ & \sum_{i=1}^m x_{ij} \geq b_j, & j = 1, \dots, r, \\ & x_{ij} \geq 0, & i = 1, \dots, m, j = 1, \dots, r. \end{aligned}$$

Hierbei modelliert die Zielfunktion die Transportkosten, die erste Nebenbedingung beschreibt die Kapazitätsbedingungen und die zweite Nebenbedingung

¹Sind damit wirkliche alle möglichen linearen Formen abgedeckt? Was ist z. B. mit allgemeinen Ungleichungsbedingungen? Was ist mit konstanten Anteilen in der Zielfunktion?

sichert, dass der Bedarf erfüllt wird. Es handelt sich hierbei um ein lineares Optimierungsproblem.² \triangle

Diskrete Optimierungsprobleme

Bei den linearen Optimierungsproblemen der Form (1.1) sind alle Variablen x des Problems kontinuierliche Größen. In vielen Anwendungen ist diese Annahme aber nicht haltbar und es müssen sogenannte *Ganzzahligkeitsbedingungen* an die Variablen (oder an eine Teilmenge der Variablen) gestellt werden. Man hat also zusätzliche Bedingungen der Form

$$x_i \in \mathcal{Z} \subset \mathbb{Z} \quad \text{oder} \quad x_i \in \{0, 1\}.$$

Ist dies der Fall nennt man das resultierende Optimierungsproblem ein *diskretes* oder *ganzzahliges Optimierungsproblem*. Wird die Ganzzahligkeitsbedingung nur an eine Teilmenge der Variablen gestellt, so sprechen wir von einem *diskret-kontinuierlichen* oder *gemischt-ganzzahligen Optimierungsproblem*. Obwohl diese Ganzzahligkeitsbedingungen die Menge der zulässigen Lösungen weiter einschränkt, d. h. den Suchraum eigentlich verkleinert, sind diese Probleme typischerweise schwieriger zu lösen. Warum dies der Fall ist und mit welchem Methoden diese Probleme trotzdem gelöst werden können ist Gegenstand der Vorlesungen “Diskrete Optimierung 1 & 2”.

Im Gegensatz zu den diskreten Optimierungsproblemen bezeichnen wir Optimierungsprobleme, bei denen keine Ganzzahligkeitsbedingungen auftreten, als *kontinuierliche Optimierungsprobleme*. Ein LP der Form (1.1) ist also ein Spezialfall eines kontinuierlichen Optimierungsproblems.

Kombinatorische Optimierungsprobleme sind dadurch gekennzeichnet, dass aus einer Menge von diskreten Elementen (z.B. Gegenstände, Orte) eine Teilmenge zu konstruieren ist, welche gewisse Nebenbedingungen erfüllt und bezüglich einer Kostenfunktion optimal ist (z.B. kleinstes Gewicht, kürzeste Strecken). Derartige Fragestellungen spielen in der Praxis eine große Rolle und können häufig als Graphenproblem oder als (ganzzahliges) lineares Optimierungsproblem formuliert werden.

Konvexe Optimierungsprobleme

Ein sehr wichtiger Spezialfall der kontinuierlichen Optimierungsprobleme sind die *konvexen Optimierungsprobleme*.

²Warum? Wie sehen die LP-Daten c , A und b aus?

Um den Begriff eines konvexen Optimierungsproblems genau beschreiben zu können, benötigen wir zwei Definitionen.

Definition 1.2 (Konvexe Menge). Eine Menge $K \subseteq \mathbb{R}^n$ heißt konvex, wenn gilt:

$$\forall x, y \in K \quad \forall t \in [0, 1] \quad \Rightarrow \quad tx + (1 - t)y \in K.$$

Definition 1.3 (Konvexe Funktion). Sei $f : D \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ eine Funktion und $K \subseteq D$ eine konvexe Menge. Die Funktion f heißt konvex auf K , falls gilt:

$$x, y \in K, t \in [0, 1] \quad \Rightarrow \quad f(tx + (1 - t)y) \leq tf(x) + (1 - t)f(y).$$

Konvexe Optimierungsprobleme sind dadurch definiert, dass die zu minimierende Funktion und die zulässige Menge konvex sind.³

Zudem haben diese Probleme die nützliche Eigenschaft, dass jedes lokale Optimum auch ein globales Optimum ist. Wer mehr über konvexe Optimierung lernen möchte, dem sei das Buch *Convex Optimization* von Boyd und Vandenberghe [4] empfohlen.

Globale Optimierungsprobleme

Handelt es sich bei einem gegebenen Problem nicht um ein konvexes Problem, so können durchaus lokale Optima auftreten, die nicht gleichzeitig globale Optima sind. Diese Probleme nennt man daher auch *globale Optimierungsprobleme*. Diese Problemklasse beinhaltet sowohl Probleme mit als auch ohne Ganzzahligkeitsbedingungen.

Das Ziel dieser Vorlesung ist die Vermittlung von Grundlagen, die sowohl für diskrete als auch für die kontinuierliche Optimierung benötigt werden. Auf dieser Basis ist später eine Spezialisierung in die eine oder andere Richtung möglich.

Eine detailliertere Klassifizierung von Optimierungsproblemen als die hier vorgestellte findet man z. B. in dem Buch *Numerical Optimization* [14] von Nocedal und Wright.

³Als Übung kann man sich dann an die Beantwortung der Frage machen, warum diskrete oder diskret-kontinuierliche Optimierungsprobleme nicht konvex sind.

Kapitel 2

Grundlagen der Graphentheorie

In diesem Kapitel werden wir grundlegende Begriffe der Graphentheorie einführen, uns darum kümmern, wie man Graphen abspeichern kann und außerdem die ersten Graphenalgorithmen diskutieren.

Die allgemeinen Begriffe der Graphentheorie (und vieles mehr) findet man z. B. in Diestel [6]. Diejenigen Aspekte, die schon stärker in Richtung kombinatorischer Optimierung auf Graphen gehen, können gut in Korte und Vygen [12] nachgelesen werden.

2.1 Grundlegende Begriffe

- Beispiel 4.** 1. Bei dem sogenannten Königsberger Brückenproblem¹ lautet die Problemstellung wie folgt: Gibt es eine Rundreise durch die Stadt Königsberg (vgl. Abbildung 2.1), die jede Pregelbrücke genau einmal benutzt? Eine etwas abstraktere Darstellung des Problems ist in Abbildung 2.2 gegeben. Die Antwort ist “Nein”. Warum dem so ist, werden wir im Verlauf der Vorlesung lernen.
2. Das Haus vom Nikolaus (Abbildung 2.3). Hier lautet die Problemstellung: Kann man das Haus vom Nikolaus zeichnen ohne den Stift abzusetzen?
3. Ein Grundversorgungsproblem: Die Frage ist, ob man alle drei Häuser durch entsprechende Leitungen bzw. Rohre mit Gas, Wasser und Strom versorgen kann ohne dass sich zwei Verbindungen kreuzen; vgl. Abbildung 2.4. △

¹Leonhard Euler, 1736

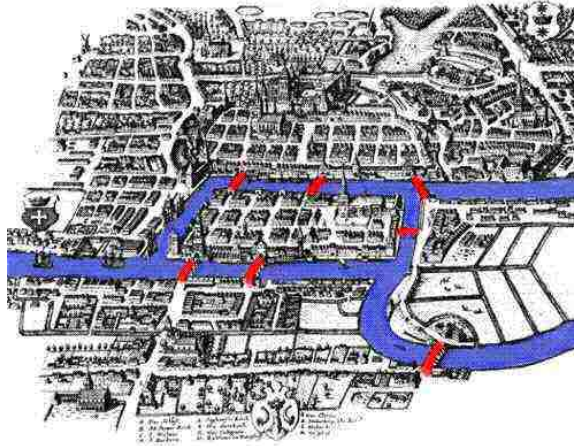


Abbildung 2.1: Die Königsberger Brücken (anno 1736) (siehe Diestel [6])

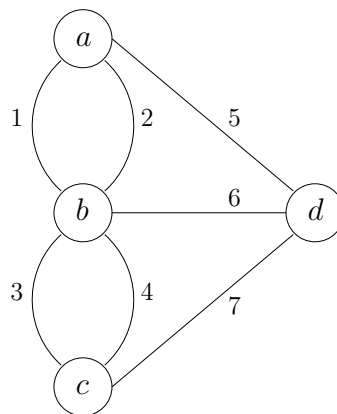


Abbildung 2.2: Abstrakte Darstellung des Königsberger Brückenproblems

Definition 2.1 (Ungerichteter Graph, Knoten, Kanten, Inzidenzfunktion). Ein *ungerichteter Graph* G ist ein Tripel $G = (V, E, \Psi)$ mit einer nicht-leeren Menge V , den *Knoten*, einer Menge E , den *Kanten*, und einer *Inzidenzfunktion* $\Psi : E \rightarrow V \times V$. Dabei bezeichnet $V \times V$ die Menge der ungeordneten Paare von Elementen aus V . Die Funktion Ψ weist also jeder Kante $e \in E$ ein Paar von Knoten $u, v \in V$ zu durch $\Psi(e) = uv = vu$.

Beispiel 5. Wir betrachten wieder den Graph des Königsberger Brücken-

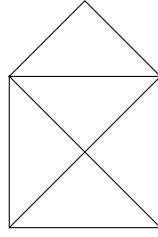


Abbildung 2.3: Das Haus vom Nikolaus

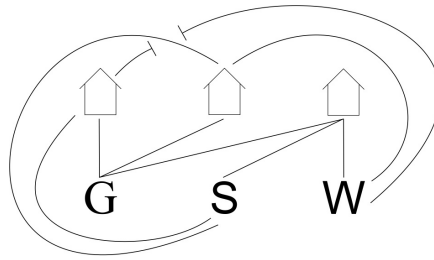


Abbildung 2.4: Ein Grundversorgungsproblem

problems aus Abbildung 2.2. Hier gilt

$$\begin{aligned}
 V &= \{a, b, c, d\}, \\
 E &= \{1, 2, \dots, 7\}, \\
 \Psi(1) &= \{a, b\} = ab = [a, b], \\
 \Psi(7) &= \{c, d\} = cd = [c, d], \\
 G &= (V, E).
 \end{aligned}
 \qquad \triangle$$

Definition 2.2 (Endlicher, unendlicher Graph). Ein Graph G heißt *endlich*, falls $|V| < \infty$ und $|E| < \infty$ gilt. Andernfalls heißt G *unendlich*.

Falls ein Graph nicht explizit als unendlich gekennzeichnet ist, gehen wir sowohl in den Übungen als auch im Skript von einem endlichen Graphen aus.

Am Beispielgraphen aus Abbildung 2.5 führen wir jetzt weitere Begriffe ein. Die Knoten 1 und 2 *liegen auf* bzw. sind *inzident* zu e . Die Kante e *verbindet* die Knoten 1 und 2. Die Knoten 1 und 2 sind *Nachbarn* bzw. *adjazent*. Zwei Kanten, die den selben Endknoten haben, werden sowohl *inzident* als auch *adjazent* genannt. Die Kante e_2 mit $\Psi(e_2) = 44$ heißt *Schlinge* und die Kanten $e_3, e_4 \in E$ mit $\Psi(e_3) = \Psi(e_4)$ heißen *parallel*. Ein Graph ohne Schlingen und parallele Kanten heißt *einfach*.

Definition 2.3 (Gerichteter Graph). Ein *gerichteter Graph* oder *Digraph* $D = (V, A, \Psi)$ ist ein Tripel bestehend aus einer Menge $V \neq \emptyset$, einer Menge A

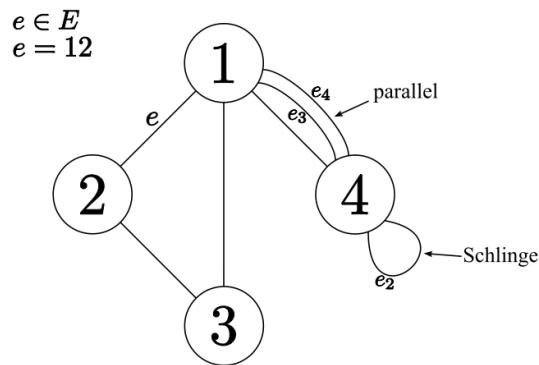


Abbildung 2.5: Die Graphenbegriffe inzident, adjazent, parallele Kanten und Schlinge

von Bögen und einer Inzidenzfunktion $\Psi : A \rightarrow V \times V$, wobei hier $(i, j) \neq (j, i)$ für $i, j \in V$ gilt.

Beispiel 6 (Gerichteter Graph). Wir betrachten den gerichteten Graphen aus Abbildung 2.6. Es ist $\Psi(a) = (2, 1)$, $\Psi(e) = (1, 2)$ und $\Psi(a) \neq \Psi(e)$. $\Psi(a)$ und $\Psi(e)$ sind gerichtet. $\{2, 1\}$ und $\{1, 2\}$ sind ungerichtet. \triangle

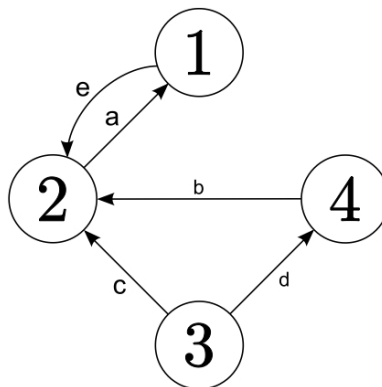


Abbildung 2.6: Gerichteter Graph

Die bisher gewählten Notationen orientieren sich an den englischen Namen:

- Bogenmenge A : engl. *arc*
- Kantenmenge E : engl. *edge*
- Knotenmenge V : engl. *vertex*
- Gerichteter Graph D : engl. *di-graph* oder *directed graph*

Für eine Kantenmenge $F \subseteq E$ bezeichnen wir $V(F)$ als die Menge aller Knoten, die zu einer Kante $f \in F$ inzident sind. Für eine Knotenmenge $W \subseteq V$ bezeichnen wir mit $E(W) \subseteq E$ die Menge aller Kanten mit beiden Endknoten in W . Für ein Beispiel siehe Abbildung 2.7.

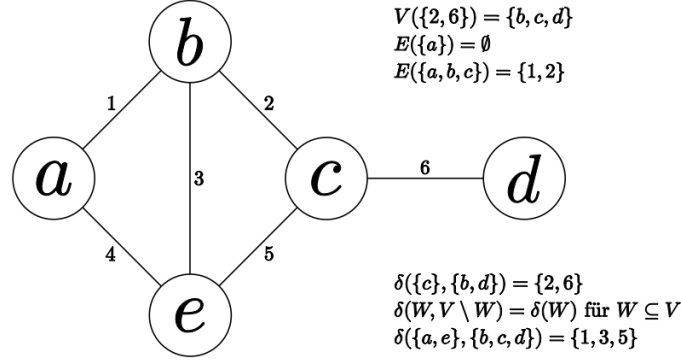


Abbildung 2.7: Kanten- und Knotenmenge

Für zwei Knotenmengen $U, W \subseteq V$ bezeichnen wir mit $[U : W]$ bzw. $\delta(U, W)$ die Menge der Kanten mit einem Endknoten in U und einem Endknoten in W . Anstelle von $\delta(W, V \setminus W)$ schreibt man auch kurz $\delta(W)$. Eine Kantenmenge $F \subseteq E$ wird als *Schnitt* bezeichnet, wenn es eine Knotenmenge $W \subseteq V$ mit $\delta(W) = F$ gibt. Die Menge $\delta(W)$ wird auch als *von W induzierter Schnitt* bezeichnet. Im Fall eines gerichteten Graphen $D = (V, A)$ bezeichnet

$$\delta^{\text{out}}(W) := \{(i, j) \in A : i \in W, j \in V \setminus W\}$$

die Menge der ausgehenden Bögen und

$$\delta^{\text{in}}(W) := \{(i, j) \in A : i \in V \setminus W, j \in W\}$$

die Menge der eingehenden Bögen²; vgl. Abbildung 2.8.

Die Anzahl der inzidenten Kanten eines Knoten v wird der Grad (engl. degree) von v genannt und mit $\deg(v)$ bezeichnet. Dabei erhöhen Schlingen den Grad eines Knoten um 2. Ist G ein einfacher Graph, dann entspricht $\deg(v)$ der Anzahl der Nachbarn von v . Ist $\deg(v) = 0$, so nennen wir v einen *isolierten* Knoten; siehe z. B. Knoten e in Abbildung 2.9. Ein Knoten v heißt *gerade* bzw. *ungerade*, falls $\deg(v)$ gerade bzw. ungerade ist.

Satz 2.4. Die Anzahl der Knoten ungeraden Grades in G ist stets gerade.

Beweis. Zählen wir die Grade aller Knoten in G zusammen, so zählen wir jede Kante vw genau zweimal: einmal von v und einmal von w aus. Es gilt also $|E| = \frac{1}{2} \sum_{v \in V} \deg(v)$ und damit ist $\sum_{v \in V} \deg(v)$ eine gerade Zahl. \square

²In der Literatur sind auch die Bezeichnungen $\delta^+(W) := \delta^{\text{out}}(W)$ und $\delta^-(W) := \delta^{\text{in}}(W)$ weit verbreitet.

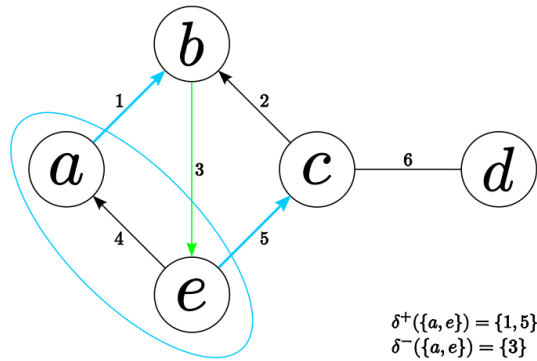


Abbildung 2.8: Ein- und ausgehende Bögen δ^{out} und δ^{in}

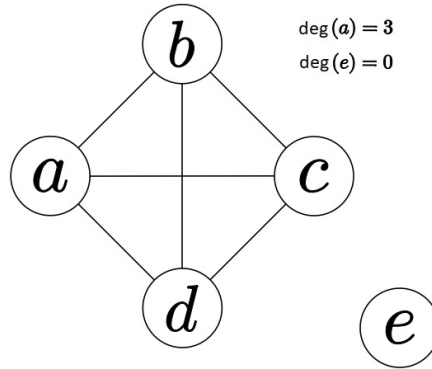


Abbildung 2.9: Ein Graph mit isoliertem Knoten e

Ein Graph $G = (V, E)$ heißt *vollständig*, falls jeder Knoten mit jedem anderen verbunden ist. Manchmal bezeichnen wir den vollständigen Graphen mit K_n , wobei n die Kardinalität der Knotenmenge ist. Der K_5 ist beispielsweise in Abbildung 2.10 gegeben.

Ein einfacher Graph, dessen Knotenmenge V in zwei disjunkte, nicht-leere Teilmengen V_1, V_2 mit $V_1 \cup V_2 = V$ eingeteilt werden kann, so dass keine zwei Knoten in V_1 und keine zwei Knoten in V_2 benachbart sind, heißt *bipartit*; vgl. Abbildung 2.11. Falls $uv \in E$ für alle $u \in V_1, v \in V_2$, so spricht man von einem *vollständigen bipartiten* Graphen. Diese eindeutig bestimmten Graphen werden auch als $K_{m,n}$ bezeichnet, wobei $m = |V_1|$ und $n = |V_2|$ gilt.

Eine endliche Folge $K = (v_0, e_1, v_1, e_2, v_2, \dots, e_k, v_k)$ mit $k \geq 0$ heißt *Kette*, falls die Folge mit einem Knoten beginnt und endet und jede Kante $e_i, i = 1, \dots, k$, mit den Knoten v_{i-1} und v_i inzidiert. Vereinfacht schreibt man auch $K = (v_0, v_1, \dots, v_k)$. Der Knoten v_0 heißt *Anfangsknoten*, der Knoten v_k heißt *Endknoten* und die Knoten v_1, \dots, v_{k-1} heißen *innere* Knoten der Kette. Eine Kette, in der alle Knoten voneinander verschieden sind, heißt *Weg*. Beachte, dass in einem Weg auch alle Kanten voneinander verschieden sind. Ein Weg,

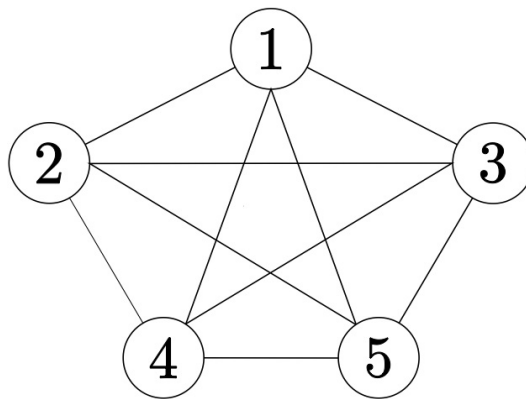


Abbildung 2.10: Der vollständige Graph K_5

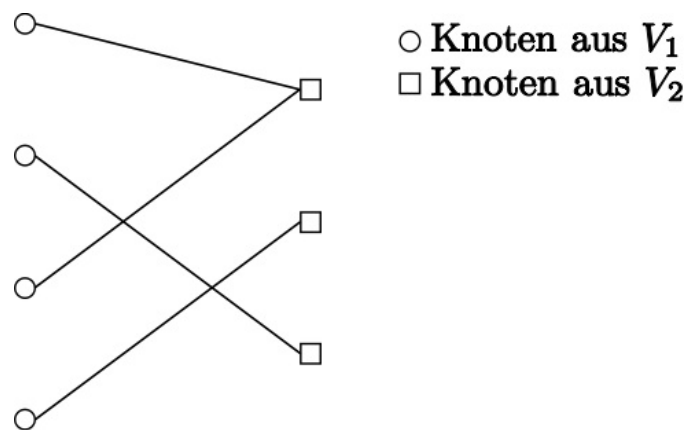


Abbildung 2.11: Ein bipartiter Graph

bei dem zwischen Anfangs- und Endknoten eine Kante existiert, heißt *Kreis*. Sind in einer Kette alle Kanten verschieden, so sprechen wir von einem *Pfad*. Beispiele für eine Kette, einen Pfad und einen Weg sind in Abbildung 2.12 zu sehen.

Ein Pfad, der jede Kante des Graphen genau einmal enthält, heißt *Eulerpfad*. Ist der Pfad geschlossen, so heißt er *Eulertour*. Ein Graph, der eine Eulertour enthält, heißt *eulersch*. Es ist nicht schwer zu sehen, dass ein Graph genau dann eulersch ist, wenn jeder Knoten geraden Grad hat.³

Ein Pfad, der jeden Knoten genau einmal enthält, heißt *Hamiltonweg*. Ist der Pfad geschlossen, dann heißt er *Hamiltontour* oder auch *Hamiltonkreis*. Ein Graph, der einen Hamiltonkreis enthält, heißt *hamiltonsch*.

Ein Graph, der derart in der Ebene zeichenbar ist, dass sich keine zwei Kanten

³Genauer werden wir uns dieses Thema in der Übung ansehen.

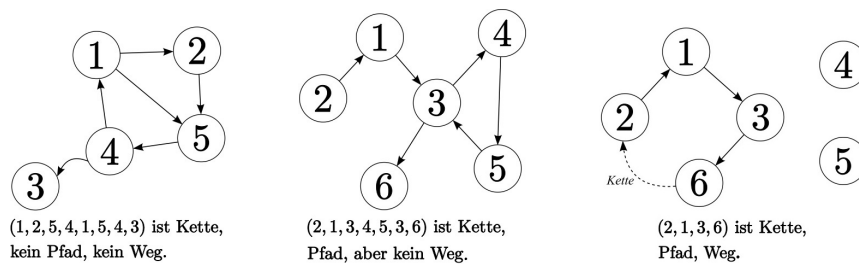


Abbildung 2.12: Kette, Pfad und Weg

kreuzen, nennt man *planar*. Häufig betrachtet man in der Graphentheorie die Frage, ob ein Graph oder eine gegebene Klasse von Graphen planar ist. Zum Beispiel könnte man sich fragen, ob der vollständige Graph $K_{3,3}$ planar ist. Auch diese Fragestellung werden wir in den Übungen betrachten. Nicht alle Graphen sind planar, wie man in Abbildung 2.4 zum Grundversorgungsproblem erkennen kann.

Ein planarer Graph teilt die Ebene in eine Menge von Flächen \mathcal{F}_G ein. Zu jeder Fläche $F \in \mathcal{F}_G$ gibt es genau einen Kreis $C_F \subseteq E$, der diese Fläche umschließt. Ein solcher Kreis heißt *Flächenkreis*. Führen wir einen Knoten für jede Fläche aus \mathcal{F}_G ein und Kanten zwischen den Knoten, deren zugehörige Flächen benachbart sind (genauer: deren zugehörige Flächenkreise eine gemeinsame Kante haben), so erhalten wir einen neuen Graphen, den *dualen Graphen* von G (in Zeichen $G^* = (V^*, E)$). Beachte, dass jede Kante im dualen Graphen genau einer Kante in G entspricht, so dass wir für beide Graphen die gleiche Kantenmenge E verwenden. In Abbildung 2.13 ist ein Beispiel für einen dualen Graphen zu sehen.

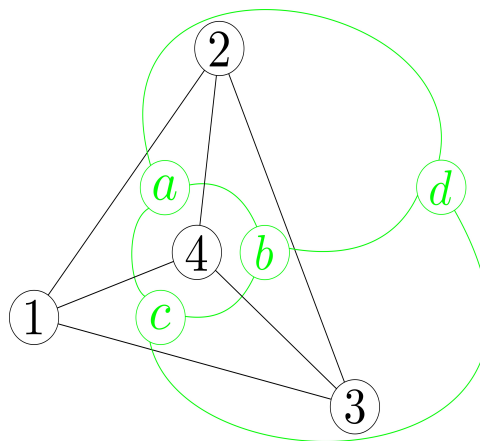


Abbildung 2.13: Dualer Graph (in grün)

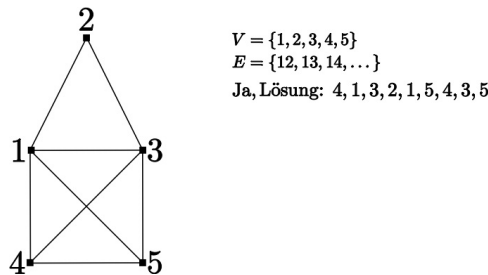


Abbildung 2.14: Eulerpfad in G

2.2 Exkurs: Algorithmen

Im Lauf der Vorlesung wollen wir unterscheiden zwischen “einfachen” und “schweren” Problemen sowie zwischen “guten” und “schlechten” Algorithmen. Um mit der Komplexität der Lösung von Problemen umgehen zu können, müssen wir formalisieren, was wir unter einem Problem und unter einem Algorithmus verstehen.

Ein *Problem* ist eine Fragestellung mit noch nicht festgelegten Parametern und einer Spezifikation, wie eine Lösung aussieht. Wenn alle Parameter spezifiziert sind, so spricht man von einer *Instanz* des Problems.

Beispiel 7. Ein Beispiel für ein Problem ist:

Sei $G = (V, E)$ gegeben. Enthält G einen Eulerpfad?

Die Fragestellung ist klar. Die noch nicht festgelegten Parameter sind der Graph G mit seinen (noch nicht spezifizierten) Kanten und Knoten. In Abbildung 2.14 ist eine Instanz des Problems gegeben. Dazu wurden die Knotenmenge als auch die Kantenmenge spezifiziert. Eine Spezifikation der Lösung kann die Antwort “Ja” oder “Nein” sein. Eine andere Spezifikation könnte aber zusätzlich auch die Kodierung eines Eulerpfads im “Ja”-Fall sein. \triangle

Die “Größe” einer Instanz wird angegeben durch die Verwendung eines Kodierungsschemas. Wir verwenden die Binärkodierung. Mit $\langle n \rangle$ bezeichnen wir die Kodierungslänge (Größe) einer Zahl $n \in \mathbb{Z}$:

$$\langle n \rangle = \lceil \log_2(|n| + 1) \rceil + 1.$$

Für eine rationale Zahl $r = p/q \in \mathbb{Q}$ mit teilerfremden p und q ist die Kodierungslänge gegeben durch

$$\langle r \rangle = \langle p \rangle + \langle q \rangle.$$

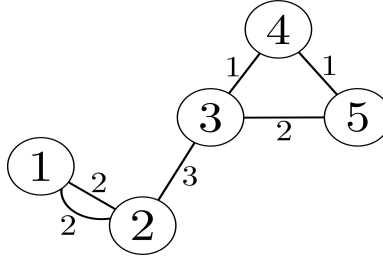


Abbildung 2.15: Der Graph G aus Beispiel 8

Die Kodierungslänge eines Graphen $G = (V, E)$ mit rationalen Kantengewichten oder Kapazitäten c_e für $e \in E$ ist definiert durch

$$\langle G \rangle = |V| + |E| + \sum_{e \in E} \langle c_e \rangle.$$

Beispiel 8. Wir betrachten den Graph aus Abbildung 2.15. Es gilt

$$\begin{aligned} \langle G \rangle &= |V| + |E| + \sum_{e \in E} \langle c_e \rangle \\ &= 5 + 6 + 2 \cdot (\lceil \log_2(|1| + 1) \rceil + 1) + 3 \cdot (\lceil \log_2(|2| + 1) \rceil + 1) \\ &\quad + 1 \cdot (\lceil \log_2(|3| + 1) \rceil + 1) \\ &= 5 + 6 + 2 \cdot 2 + 3 \cdot 3 + 1 \cdot 3 = 27. \end{aligned}$$

Die Kodierungslänge des Graphen G ist also $\langle G \rangle = 27$. \triangle

Ein *Algorithmus* ist eine endliche Menge von Operationen auf einer gegebenen Menge von Daten. Der initiale Zustand dieser Daten beschreibt die Eingabe des Algorithmus und beim Halten des Algorithmus steht die Ausgabe in einer festgelegten Position in den Daten.

Man beachte, dass die Menge der Operationen endlich und fest gegeben ist, wohingegen sowohl die Eingabe als auch die Länge der Eingabe variiert und typischerweise von der Instanz abhängig ist. In der Regel sind die Eingaben als Felder (engl. *arrays*), also als endliche Folgen gegeben. Die Anzahl dieser Felder ist konstant und nur abhängig vom Algorithmus, wohingegen die Länge der Felder von der Instanz abhängt.

Wir sagen, ein Algorithmus A *löst* ein Problem Π , falls A für alle Instanzen I von Π , eine Lösung in einer *endlichen* Anzahl an Operationen findet. Ein Algorithmus darf auf Variablen z_1, \dots, z_k und ein Feld a zugreifen, dessen Länge von der Instanz abhängen darf. Als *Operation* erlauben wir die folgenden

Typen, wobei $i, j, h \in \{1, \dots, k\}$ ist,

$$\begin{array}{ll} z_i \leftarrow a[z_j], & a[z_j] \leftarrow z_i, \\ z_i \leftarrow z_j + z_h, & z_i \leftarrow z_j - z_h, \\ z_i \leftarrow z_j z_h, & z_i \leftarrow z_j / z_h, \\ z_i \leftarrow z_i + 1, & z_i \leftarrow 1 \text{ falls } z_j > 0 \text{ und } z_i \leftarrow 0 \text{ sonst,} \end{array}$$

sowie Kontrollstrukturen wie z. B. while-Schleifen. Die Operationen beinhalten also insbesondere die elementaren arithmetischen Operationen Addition, Subtraktion, Multiplikation, Division und Vergleich.

Das hier beschriebene algorithmische Modell ist das einer *Random Access Machine* (oder kurz: *RAM*). Die RAM ist ein Rechnermodell der theoretischen Informatik, das einem realen Rechner sehr ähnlich ist; vgl. z. B. Kapitel 4 in Schrijver [19].

Die *Laufzeit* eines Algorithmus ist die Anzahl der Operationen, die zur Lösung der Instanz notwendig sind. Sei $T_A(l)$ die Laufzeit eines Algorithmus A zur Lösung einer Instanz mit Kodierungslänge höchstens $l \in \mathbb{N}$. Wir sagen, dass der Algorithmus A ein *Polynomialzeit-Algorithmus* ist, falls es ein Polynom p gibt mit

$$T_A(l) \leq p(l) \quad \text{für alle } l \in \mathbb{N}.$$

Die Menge der Probleme, die in Polynomialzeit lösbar sind, d. h., für die es einen Polynomialzeit-Algorithmus gibt, bezeichnen wir mit P .

Die *Turingmaschine*⁴ ist ein weiteres theoretisches Rechnermodell, welche die Arbeitsweise eines Computers auf besonders einfache und mathematisch gut zu analysierende Weise modelliert. Während eine RAM auf eine beliebige Feldposition $a[i]$ mit $i \in \mathbb{N}$ in konstanter Zeit zugreifen kann, kann eine Turingmaschine nur auf benachbarten Feldpositionen (z.B. $\dots, a[5], a[6], a[7], \dots$) arbeiten. Die Turingmaschine und die RAM können sich gegenseitig in Polynomialzeit simulieren.

Definition 2.5 (Landau-Symbole). Sei $M = \{f: \mathbb{N} \rightarrow \mathbb{R}\}$ die Menge der reellwertigen Funktionen. Für $g \in M$ definieren wir

$$\begin{aligned} \mathcal{O}(g) &:= \{f \in M: \exists c, n_0 \in \mathbb{N}: f(n) \leq cg(n) \text{ für alle } n \geq n_0\}, \\ \Omega(g) &:= \{f \in M: \exists c, n_0 \in \mathbb{N}: f(n) \geq cg(n) \text{ für alle } n \geq n_0\}, \\ \Theta(g) &:= \mathcal{O}(g) \cap \Omega(g). \end{aligned}$$

Informell lassen sich die Landau-Symbole wie in Tabelle 2.1 beschreiben. Das für uns wichtigste Landau-Symbol ist das \mathcal{O} , da es es uns erlaubt eine obere Schranke für die Laufzeit von Algorithmen zu beschreiben.

⁴Die Turingmaschine ist benannt nach Alan Turing, der sie 1936 einführte.

Notation	Anschauliche Bedeutung
$f \in \mathcal{O}(g)$	f wächst höchstens so schnell wie g
$f \in \Omega(g)$	f wächst mindestens so schnell wie g
$f \in \Theta(g)$	f wächst genau so schnell wie g

Tabelle 2.1: Landau-Symbole

Polynomialzeit-Algorithmen vom Grad k haben stets einen Laufzeitaufwand in $\mathcal{O}(n^k)$, wohingegen *exponentielle* Algorithmen den Laufzeitaufwand $\mathcal{O}(2^n)$, $\mathcal{O}(n!)$ oder $\mathcal{O}(n^n)$ haben.

Beispiel 9. Gegeben sei die Funktion $f(n) = 3n^2 + n + 1$. Da es sich hierbei um eine quadratische Funktion handelt, der Grad ist also $k = 2$, gilt $f(n) \in \mathcal{O}(n^2)$, was wir aber noch zeigen müssen.

Um dies zu zeigen, nutzen wir die entsprechende Definition:

$$\mathcal{O}(g) = \{f \in M : \exists c, n_0 \in \mathbb{N} : f(n) \leq cg(n) \text{ für alle } n \geq n_0\}.$$

Im Prinzip bedeutet dies, dass es eine Funktion g gibt, die schneller oder zumindest gleich schnell wächst wie f . Dazu multiplizieren wir auch noch eine Konstante c , die wir für den Beweis frei wählen können:

$$f(n) \leq cg(n) \iff 3n^2 + n + 1 \leq cn^2.$$

Jetzt müssen wir ein n_0 und c finden, für die diese Gleichung gilt. Zunächst dividieren wir mit n^2 , das ergibt dann

$$3 + \frac{1}{n} + \frac{1}{n^2} \leq c.$$

Nun können wir uns ein n_0 wählen, z. B. $n_0 = 1$. In die Gleichung für n eingesetzt ergibt dann, dass $c \geq 5$ gewählt werden muss; wir wählen also $c = 5$. Offensichtlich gilt für $n_0 = 1$ und $c = 5$, dass die Gleichung für alle $n \geq n_0$ erfüllt ist, was man noch, z. B. mittels vollständiger Induktion, zeigen muss.⁵ Somit ist also bewiesen, dass $f(n) \in \mathcal{O}(n^2)$ gilt und wir sind fertig. \triangle

2.3 Speicherung von Graphen

Im Folgenden diskutieren wir drei verschiedene Arten zur Speicherung von Graphen und illustrieren sie alle am Graphen in Abbildung 2.16. Es sei dabei $D = (V, A)$ mit $|V| = n$ und $|A| = m$.

⁵Bitte üben!

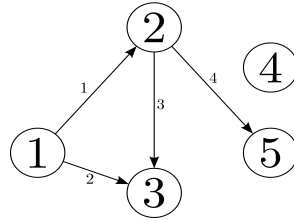


Abbildung 2.16: Ein Beispielgraph zur Diskussion unterschiedlicher Speicherungen von Graphen

Kanten- bzw. Bogen-Liste: Die Speicherung in einer Bogenliste lautet dann:

$$n, m, a_1, e_1, a_2, e_2, \dots, a_m, e_m,$$

wobei a_i der Anfangsknoten von Bogen $i \in A$ und e_i der Endknoten von Bogen i ist. Für den ungerichteten Fall ergibt sich eine äquivalente Darstellung.

Eigenschaften:

- Speicher $\mathcal{O}(m)$
- Zugriff $\mathcal{O}(m)$

In unserem Beispielgraphen in Abbildung 2.16 lautet die Bogenliste

$$5, 4, 1, 2, 1, 3, 2, 3, 2, 5.$$

Adjazenzmatrix: Ein Digraph wird spezifiziert durch die Angabe einer $n \times n$ Adjazenzmatrix $M = (a_{ij})_{1 \leq i, j \leq n}$, wobei $a_{ij} = 1$, wenn $(i, j) \in A$, und $a_{ij} = 0$ sonst.

Eigenschaften:

- Zugriff Adjazenzbeziehung $\mathcal{O}(1)$
- Speicher $\mathcal{O}(n^2)$

Für unseren Beispielgraphen lautet die Adjazenzmatrix

$$M = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{bzw. ungerichtet} \quad M = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Inzidenzmatrix: Ein Digraph wird spezifiziert durch die Angabe einer $n \times m$ Inzidenzmatrix $B = (a_{ij})_{1 \leq i \leq n, 1 \leq j \leq m}$, wobei $a_{ij} = 1$ wenn

Knoten $i \in V$ der Anfangsknoten von Bogen $j \in A$ ist, $a_{ij} = -1$ wenn Knoten i Endknoten von Bogen j ist und $a_{ij} = 0$ sonst. Im Fall von ungerichteten Graphen speichert man stets eine 1.

Eigenschaften:

- Zugriff Inzidenzbeziehung $\mathcal{O}(1)$
- Speicher $\mathcal{O}(nm)$

Für unseren Beispielgraphen lautet die Inzidenzmatrix

$$B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 1 \\ 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \quad \text{bzw. ungerichtet} \quad B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Die Inzidenzmatrix eines ungerichteten bipartiten Graphen ist eine *total unimodulare Matrix*, genauso wie die eines gerichteten Graphen. Dabei heißt eine Matrix total unimodular, falls jede quadratische Untermatrix Determinante $+1$, -1 oder 0 hat.

Adjazenzliste: Eine Adjazenzliste besteht aus der Anzahl der Knoten, der Anzahl der Kanten sowie für jeden Knoten seinen Grad und die Liste seiner inzidenten Kanten bzw. Bögen oder seiner benachbarten Knoten (eventuell mit Kosten oder weiteren Informationen).

Eigenschaften

- Speicher $\mathcal{O}(n + m)$
- Zugriff $\mathcal{O}(n)$

Für den obigen Beispielgraphen wäre die Adjazenzliste die folgende:

```

5
4
1, 2 : 1, 2, 1, 3
2, 3 : 1, 2, 2, 3, 2, 5
3, 2 : 1, 3, 2, 3
4, 0 :
5, 1 : 2, 5

```

2.4 Suchalgorithmen auf Graphen

Eine Frage, die in vielen Anwendungen und in vielen Algorithmen als Subproblem auftaucht, ist, ob es einen Weg zwischen zwei Knoten in einem Graphen gibt. Wir diskutieren in diesem Abschnitt zwei Algorithmen zur Beantwortung dieser Frage: die Breiten- und die Tiefensuche. Vorab benötigen wir aber noch eine Definition.

Definition 2.6 (Zusammenhang). Ein ungerichteter Graph $G = (V, E)$ heißt *zusammenhängend*, falls es zu je zwei beliebigen Knoten $v, w \in V$ einen Weg in G mit v als Anfangsknoten und w als Endknoten gibt. Einen maximalen zusammenhängenden Teilgraphen von G nennt man eine *Zusammenhangskomponente*.

2.4.1 Breitensuche

Algorithmus 1 gibt eine formale Beschreibung der Breitensuche⁶. Die Variable `next` gibt dabei am Ende des Algorithmus die Anzahl der Zusammenhangskomponenten an. Alle Knoten einer Zusammenhangskomponente sind dadurch gekennzeichnet, dass sie am Ende alle mit demselben Wert `mark` markiert sind. Die Breitensuche arbeitet nach dem FIFO-Prinzip (First-In-First-Out), d. h., derjenige Knoten, der bereits entdeckt aber noch nicht abgearbeitet wurde, den der Algorithmus als erstes “entdeckt” hat, wird als nächstes bearbeitet.

Beispiel 10 (Breitensuche). Wir illustrieren die Breitensuche jetzt an einem Beispiel; vgl. Abbildung 2.17.

Zuerst wählen wir einen Startknoten. Wir wählen Knoten 1, in Grafik (1) blau markiert. In Grafik (2) wurde dieser nun markiert, daher grün. Die Nachbarknoten von Knoten 1 sind blau markiert. Von diesen beiden Knoten wählen wir einen aus; wir wählen nach dem FIFO-Prinzip Knoten 2. Die Nachbarknoten von Knoten 2 sind ebenfalls blau markiert. Da aber Knoten 1 noch einen uns unbekannten Nachbar hat, nämlich Knoten 3, nehmen wir erstmal diesen in die Liste auf und merken uns schon einmal seine Nachbarn vor, indem wir sie blau markieren.

Nun gehen wir zurück zu unserem ersten gefundenen Nachbarn von Knoten 1, dem Knoten 2 und schauen, welche uns unbekannten Nachbarn dieser hat. Der erste Unbekannte ist Knoten 4, welcher dann in die Liste aufgenommen wird. Anschließend folgt Knoten 5. Da Knoten 5 auch noch einen unbekannten Nachbarn hat, wird dieser vorgemerkt und blau markiert. Damit hat Knoten 2 keine uns unbekannten Nachbarn mehr und wir kehren zurück zu Knoten 3.

⁶Engl.: *breadth-first-search* oder *BFS*

Algorithmus 1 Breitensuche

Eingabe: Graph $G = (V, E)$ **Ausgabe:** Anzahl der Zusammenhangskomponenten und die Menge der Knoten jeder Zusammenhangskomponente.

```
1: Setze  $\text{mark}(v) = -1$  für alle  $v \in V$ ,  $\text{next} = 0$  und  $L = \emptyset$ .
2: for  $v \in V$  do
3:   if  $\text{mark}(v) < 0$  then
4:      $\text{next} \leftarrow \text{next} + 1$ 
5:     Füge  $v$  ans Ende der Liste  $L$  an.
6:     while  $L \neq \emptyset$  do
7:       Wähle  $v$  vom Anfang der Liste und entferne  $v$  aus  $L$ .
8:       if  $\text{mark}(v) < 0$  then
9:          $\text{mark}(v) \leftarrow \text{next}$ 
10:      for alle zu  $v$  adjazenten  $w$  do
11:        if  $\text{mark}(w) < 0$  then
12:           $\text{mark}(w) \leftarrow \text{next}$ 
13:          Füge  $w$  am Ende der Liste an.
14: return  $\text{next}$ ,  $\text{mark}(\cdot)$ 
```

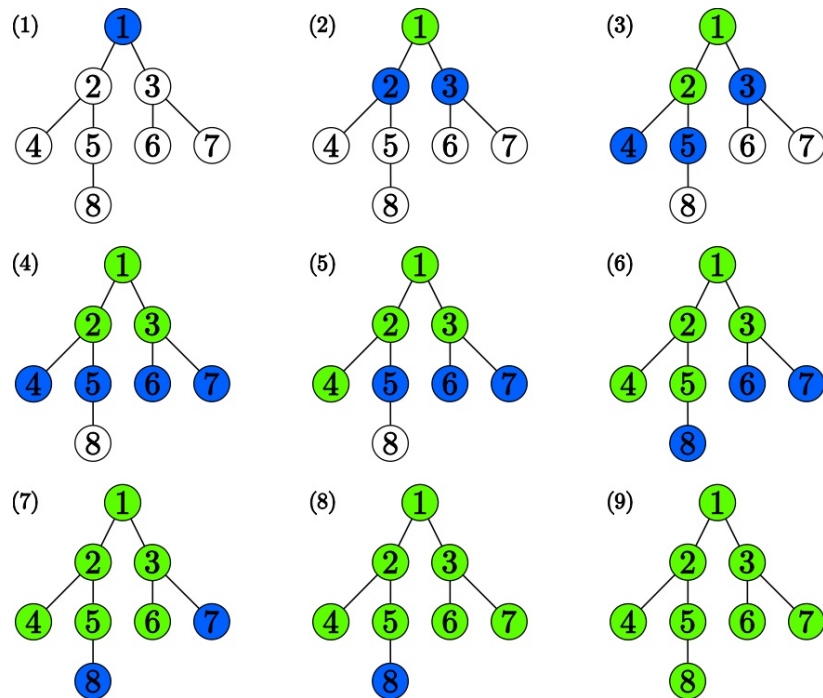


Abbildung 2.17: Grafische Darstellung der Breitensuche

Knoten 3 hat als erster Nachbar den Knoten 6, dieser ist uns nicht bekannt und wird daher in die Liste aufgenommen. Der nächste ist Knoten 7, auch dieser ist unbekannt und wird in die Liste aufgenommen.

Nun sind alle Nachbarn von Knoten 3 bekannt und wir kehren zum letzten Knoten zurück, der noch unbekannte Nachbarn hatte; das war Knoten 5. Der unbekannte Nachbar von Knoten 5 ist Knoten 8, welcher dann auch direkt in die Liste mit aufgenommen wird. Wie man in Grafik (9) erkennen kann, sind nun alle Knoten grün markiert und uns somit bekannt.

Die BFS-Reihenfolge der Knoten, die wir um beschriebenen Beispiel gefunden haben, lautet also

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8.$$

Eine andere Möglichkeit wäre

$$1 \rightarrow 3 \rightarrow 2 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 8. \quad \triangle$$

2.4.2 Tiefensuche

Algorithmus 2 beschreibt das Verfahren der Tiefensuche⁷. Diese Methode arbeitet nach dem LIFO-Prinzip (Last-In-First-Out). Bei der formalen Beschreibung machen wir das erste Mal davon Gebrauch, dass ein Algorithmus auch einen anderen Algorithmus als Subprozedur aufrufen kann. Dies erlaubt es häufig, ein Verfahren übersichtlicher darzustellen und einfacher zu analysieren.

Algorithmus 2 Tiefensuche

Eingabe: Graph $G = (V, E)$

Ausgabe: Anzahl der Zusammenhangskomponenten und die Menge der Knoten jeder Zusammenhangskomponente

Setze $\text{mark}(v) = -1$ für alle $v \in V$ und $\text{next} = 0$.

```

1: for  $v \in V$  do
2:   if  $\text{mark}(v) < 0$  then
3:      $\text{next} \leftarrow \text{next} + 1$ 
4:      $\text{mark}(v) \leftarrow \text{next}$ 
5:     for  $w$  adjazent zu  $v$  do
6:        $\text{checkNeighbor}(w, \text{next})$ 
7: return  $\text{next}, \text{mark}(\cdot)$ 
```

Wie bei der Breitensuche steht am Ende des Verfahrens in der Variable next die Anzahl der Zusammenhangskomponenten des Graphen G und mark hat ebenfalls die gleiche Bedeutung wie bei der Breitensuche.

⁷Engl.: *depth-first-search* oder *DFS*

Algorithmus 3 checkNeighbor(w, next)

Eingabe: $G = (V, E), w \in V, \text{next}, \text{mark}(\cdot)$

- 1: **if** $\text{mark}(w) < 0$ **then**
 - 2: $\text{mark}(w) = \text{next}$
 - 3: **for** alle zu w adjazenten u **do**
 - 4: checkNeighbor(u, next)
-

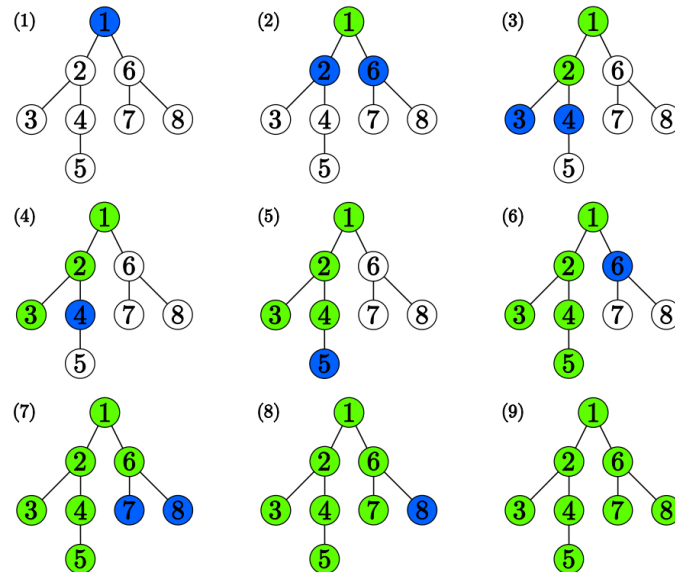


Abbildung 2.18: Grafische Darstellung der Tiefensuche

Beispiel 11 (Tiefensuche). Wir illustrieren die Tiefensuche wieder an einem Beispiel; vgl. Abbildung 2.18.

Auch hier wählen wir zuerst einen Startknoten, dieser ist Knoten 1. Nun suchen wir uns einen Nachbarn von Knoten 1 aus, zur Auswahl stehen Knoten 2 und 6, wir wählen Knoten 2.

Nun schauen wir, welche Nachbarn Knoten 2 hat, wovon wir einen wählen. Wir wählen Knoten 3. Knoten 3 hat keine uns unbekannten Nachbarn, daher gehen wir wieder einen Knoten zurück, also zu Knoten 2. Der letzte unbekannte Nachbar von Knoten 2 ist Knoten 4, weshalb wir zu Knoten 4 gehen. Die möglichen Nachbarn von Knoten 4 ist nur ein Knoten, Knoten 5, welchen wir dann auch wählen. Knoten 5 hat keinen Nachbarn, also geht es zurück zu Knoten 4, auch dieser hat keine weiteren unbekannten Nachbarn, also weiter zurück zu Knoten 2. Auch dieser hat keine weiteren Nachbarn, daher gehen wir noch einen Schritt weiter zurück, also zu Knoten 1. Knoten 1 hat wieder einen uns unbekannten Nachbarn, Knoten 6, welchen wir wählen. Knoten 6 hat die beiden Nachbarn 7 und 8, wovon wir Knoten 7 wählen.

Nun wird wieder geschaut, welche Nachbarn Knoten 7 hat. Da Knoten 7 keine Nachbarn hat, geht es wieder einen Schritt zurück zu Knoten 6. Knoten 6 hat noch einen weiteren uns unbekannten Nachbarn, Knoten 8, welchen wir wählen. Da Knoten 8 keinen weiteren Nachbarn mehr hat und wir bereits alle Knoten erfasst haben, sind wir fertig.

Die DFS-Reihenfolge der Knoten, die in diesem Beispiel beschrieben wurden, lautet demnach

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8.$$

Eine andere Möglichkeit wäre

$$1 \rightarrow 6 \rightarrow 8 \rightarrow 7 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 3. \quad \triangle$$

Satz 2.7. Die Laufzeit von Tiefensuche (2) beträgt $\mathcal{O}(|V| + |E|)$.

Beweis. Der Algorithmus beginnt damit, die Markierung aller Knoten zu initialisieren. Dies benötigt offenbar eine Laufzeit von $\mathcal{O}(|V|)$.

Nun betrachten wir den restlichen Teil des Algorithmus. Die Zeilen 2, 3 und 4 der Tiefensuche sowie die Zeilen 1 und 2 von `checkNeighbor` werden in $\mathcal{O}(1)$ behandelt.

Jeder Knoten $v \in V$ wird genau einmal markiert. Dies sieht man dadurch, dass `mark(v)` nur dann verändert wird, wenn vorher die Bedingung `mark(v) < 0` überprüft wurde und `next` nie negative Werte annimmt. Wenn eine Markierung also einmal auf `next` gesetzt wurde, ist die Bedingung `mark(v) < 0` nicht mehr erfüllt.

Da die Schleifen aus den Zeilen 5 (Tiefensuche) und 3 (`checkNeighbor`) immer auf eine Markierungsanweisung folgen, führt der Algorithmus für jeden Knoten genau eine solche Schleife aus, welche alle von $v \in V$ ausgehenden Kanten durchläuft. Jede Kante ist zu maximal zwei Knoten inzident. Also besucht der Algorithmus jede Kante höchstens zwei Mal. Damit ergibt sich eine Laufzeit von $\mathcal{O}(2 \cdot |E|) = \mathcal{O}(|E|)$.

Unter Verwendung einer Adjazenzliste, benötigt der Algorithmus somit eine Laufzeit von $\mathcal{O}(|V| + |E|)$. \square

Satz 2.8. Die Laufzeit von Breitensuche (1) beträgt $\mathcal{O}(|V| + |E|)$.

Beweis. Der Beweis erfolgt analog zu Satz 2.7. \square

Kapitel 3

Sortier-Algorithmen

In diesem Kapitel beschäftigen wir uns mit dem Sortierproblem, das heißt mit dem Problem:

Gegeben seien n Zahlen a_1, \dots, a_n . Finde eine Permutation

$$\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\},$$

so dass

$$a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}.$$

In der Regel sind die zu sortierenden Zahlen nicht isolierte Elemente, sondern Teil komplexer Strukturen oder Objekte. Jedes dieser Objekte enthält einen Schlüssel¹, nach dem sortiert werden soll. Diese Schlüssel sind häufig Zahlen, können aber auch andere Datentypen wie Strings oder Characters sein. Entsprechend muss man im Allgemeinen eine Ordnungsrelation auf diesen Elementen definieren. Wir beschränken uns in diesem Kapitel jedoch auf die “einfachste” Form, nämlich auf ganze Zahlen als Schlüssel, sowie auf die herkömmliche Ordnung auf Zahlen.

Sortieren an sich ist ein sehr großes Gebiet: Es gibt ganze Bücher, die sich ausschließlich mit Sortieralgorithmen beschäftigen; vgl. Knuth [11], aber beispielsweise auch Sedgewick [20]. Wir wollen in diesem Rahmen nur auf einige wichtige Algorithmen eingehen.

3.1 Selectionsort

Das wohl einfachste Sortierverfahren ist, dass man sich zunächst das kleinste Element sucht, dieses an die erste Stelle tauscht, danach das zweit-kleinste

¹Engl.: *key*

unter den verbleibenden Zahlen sucht, dieses an die zweite Stelle tauscht, usw. Dieses Vorgehen wiederholt man genau n -mal, bis alle Elemente sortiert sind.

Formal ist das Verfahren in Algorithmus 4 gegeben. Man beachte, dass

Algorithmus 4 Selectionsort

Eingabe: Ein Array a der Länge n mit $a[i] \in \mathbb{Z}$.

Ausgabe: Das sortierte Array a mit $a[1] \leq a[2] \leq \dots \leq a[n]$.

```

1: for  $i = 1 : n$  do
2:    $\text{min} \leftarrow i$ 
3:   for  $j = i + 1 : n$  do
4:     if  $a[j] < a[\text{min}]$  then
5:        $\text{min} \leftarrow j$ 
6:   Tausche  $a[\text{min}]$  und  $a[i]$ .
7: return  $a$ 

```

wir die Elemente innerhalb des Arrays getauscht haben. Möchte man die Elemente innerhalb des Arrays nicht tauschen, zum Beispiel weil sie Teil größerer Objekte oder Strukturen sind, so führt man wie bei der Definition des Sortierproblems einen zusätzlichen Permutationsvektor π mit, initialisiert diesen mit $\pi(i) = i$ und führt die Tauschoperationen auf diesem durch. Die Laufzeit von Algorithmus 4 ist $\mathcal{O}(n^2)$.² Wir nennen ihn *stabil*, da Elemente mit gleichem Wert ihre relative Ordnung behalten, und wir klassifizieren ihn als *in-place* Verfahren, da der zusätzliche Speicherbedarf konstant, d. h., $\mathcal{O}(1)$, ist.

3.2 Bubblesort

Seinen Namen hat der folgende Algorithmus von der Eigenschaft, dass er im ersten Durchlauf das größte Element sucht (die größte Blase) und dieses nach oben “blubbern” lässt. Im zweiten Durchlauf sucht man sich unter den verbleibenden $n - 1$ Elementen das zweit-größte Element (die zweit-größte Blase), lässt diese auch nach oben “blubbern”, usw. Dieses Verfahren wiederholt man dann so lange, bis alle Elemente an die richtige Stelle “geblubbert” sind. Das Verfahren ist in Algorithmus 5 dargestellt. Bubblesort hat die gleichen Charakteristika wie Selectionsort (Algorithmus 4), d. h., seine Laufzeit ist $\mathcal{O}(n^2)$, er ist stabil und läuft in-place.

Da beide bisher betrachteten Algorithmen eine Laufzeit von $\mathcal{O}(n^2)$ haben, stellt sich die Frage, ob es nicht auch besser (schneller) geht. Die Antwort ist “Ja”; die beiden folgenden Algorithmen sind Beispiele dafür.

²Warum?

Algorithmus 5 Bubblesort

Eingabe: Ein Array a der Länge n mit $a[i] \in \mathbb{Z}$.

Ausgabe: Das sortierte Array a mit $a[1] \leq a[2] \leq \dots \leq a[n]$.

```
1: for  $i = 1 : n - 1$  do
2:   for  $j = 1 : n - i$  do
3:     if  $a[j] > a[j + 1]$  then
4:       Tausche  $a[j]$  und  $a[j + 1]$ .
5: return  $a$ 
```

3.3 Quicksort

Die Idee des Quicksort-Algorithmus ist es, sich ein (beliebiges) Element, das sogenannte Pivotelement, herauszugreifen und die anderen Elemente in zwei Untermengen aufzuteilen. Eine Untermenge enthält die Elemente, die kleiner oder gleich als das gewählte Element sind, und eine Untermenge der Elemente, die größer sind. Danach werden beide Hälften nach dem gleichen Prinzip (also rekursiv) behandelt. Hat eine Untermenge weniger als zwei Elemente, muss diese nicht mehr sortiert werden, wodurch die Rekursion endet. Wir benutzen hier zur Aufteilung in jedem Teilvektor des Arrays immer das Element mit dem größten Index. Im Detail ist das Verfahren in Algorithmus 6 gegeben.

Algorithmus 6 Quicksort

Eingabe: Ein Array a der Länge n mit $a[i] \in \mathbb{Z}$, eine untere und obere Grenze l, r mit $1 \leq l \leq r \leq n$.

Ausgabe: Das sortierte Array a mit $a[l] \leq a[l + 1] \leq \dots \leq a[r]$.

```
1: Setze  $i \leftarrow l$  und  $j \leftarrow r$ .
2: while  $i < j$  do
3:   while  $a[i] \leq a[r]$  und  $i < j$  do
4:     Setze  $i \leftarrow i + 1$ 
5:   while  $a[j] \geq a[r]$  und  $i < j$  do
6:     Setze  $j \leftarrow j - 1$ 
7:   Tausche  $a[i]$  und  $a[j]$ .
8: Tausche  $a[i]$  und  $a[r]$ .
9: if  $l < i - 1$  then
10:   Quicksort( $a, l, i - 1$ )
11: if  $i + 1 < r$  then
12:   Quicksort( $a, i + 1, r$ )
13: return  $a$ 
```

Der erste Aufruf erfolgt mit den Parametern $(a, 1, n)$.

Beispiel 12. Wir wollen das Array $(2, 5, 3, 6, 7, 1, 2)$ mit dem Quicksort-

Algorithmus sortieren. Das rote Element ist jeweils das Pivotelement, während die blauen Elemente einen anstehenden Tausch anzeigen. Q1 steht für den ersten rekursiven Quicksort-Aufruf in Zeile 10 und Q2 entsprechend für den Aufruf in Zeile 12. Weitere rekursive Aufrufe sind durch einen Punkt getrennt dargestellt.

Quicksort	2_i	5	3	6	7	1	2_j
i++	2	5_i	3	6	7	1	2_j
j--	2	5_i	3	6	7	1_j	2
\leftrightarrow	2	1_i	3	6	7	5_j	2
i++	2	1	3_i	6	7	5_j	2
j--	2	1	3_i	6	7_j	5	2
j--	2	1	3_i	6_j	7	5	2
j--	2	1	3_{ij}	6	7	5	2
\leftrightarrow	2	1	3_{ij}	6	7	5	2
\leftrightarrow	2	1	2_{ij}	6	7	5	3
Q1	2_i	1_j	2	6	7	5	3
j--	2_{ij}	1	2	6	7	5	3
\leftrightarrow	2_{ij}	1	2	6	7	5	3
\leftrightarrow	1_{ij}	2	2	6	7	5	3
Q2	1	2	2	6_i	7	5	3_j
j--	1	2	2	6_i	7	5_j	3
j--	1	2	2	6_i	7_j	5	3
j--	1	2	2	6_{ij}	7	5	3
\leftrightarrow	1	2	2	6_{ij}	7	5	3
\leftrightarrow	1	2	2	3_{ij}	7	5	6
Q2.2	1	2	2	3	7_i	5	6_j
j--	1	2	2	3	7_i	5_j	6
\leftrightarrow	1	2	2	3	5_i	7_j	6
i++	1	2	2	3	5	7_{ij}	6
\leftrightarrow	1	2	2	3	5	7_{ij}	6
\leftrightarrow	1	2	2	3	5	6_{ij}	7
Fertig	1	2	2	3	5	6	7

△

Satz 3.1. Die durchschnittliche Laufzeit von Quicksort ist $\mathcal{O}(n \log n)$.

Beweis. Übungsaufgabe. □

Algorithmus 6 kann im Worst-Case-Fall auch langsamer laufen als $\mathcal{O}(n \log n)$.

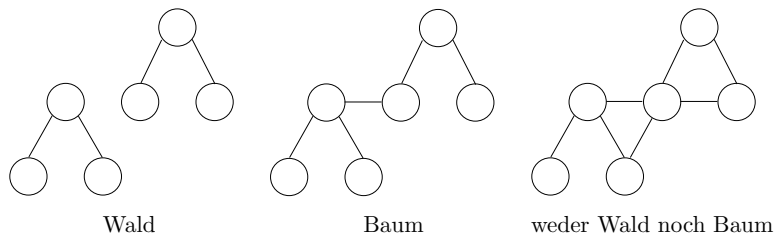


Abbildung 3.1: Beispiele für einen Wald, einen Baum und für einen Graphen, der weder Wald noch Baum ist

Dann ergibt sich eine Laufzeit von $\mathcal{O}(n^2)$. Dies ist auch tatsächlich die Worst-Case-Laufzeit von Algorithmus 6. Dazu ebenfalls mehr in den Übungen.

Damit haben wir schon einmal einen Sortieralgorithmus kennengelernt, der zumindest im Durchschnitt schnell, d. h. besser als $\mathcal{O}(n^2)$, ist. Die Frage ist nun, ob es nicht auch ein Sortierverfahren gibt, das auch im Worst-Case besser als $\mathcal{O}(n^2)$ ist.

3.4 Heapsort

Um das Heapsort-Verfahren betrachten zu können, müssen wir zunächst definieren, was ein binärer Baum ist.

Definition 3.2 (Wald, Baum). Ein Graph G ist ein *Wald*, falls er keinen Kreis enthält. Ist G außerdem zusammenhängend, so heißt G *Baum*.

In Abbildung 3.1 sind Beispiele für einen Wald, einen Baum und für einen Graphen, der weder Wald noch Baum ist, gegeben.

Definition 3.3 (Binärer Baum). Ein *binärer Baum* ist ein Baum $B = (V, E)$, bei dem ein Knoten $r \in V$, die *Wurzel*, ausgezeichnet ist. Die restlichen Knoten sind in zwei disjunkte Teilmengen partitioniert, die jeweils wieder binäre Bäume oder leer sind. Diese beiden Teilmengen werden als *linkes* bzw. *rechtes Kind* bezeichnet. Sind beide Teilmengen eines Knoten $v \in V$ leer, wird v als *Blatt* bezeichnet.

Definition 3.4 (Heap). Ein *(Max-)Heap* ist ein binärer Baum, an dessen Knoten i Schlüssel $a[i]$ angeheftet sind mit

$$a[\text{linkes Kind von } i] \leq a[i] \quad \text{und} \quad a[\text{rechtes Kind von } i] \leq a[i].$$

Das bedeutet insbesondere, dass die Wurzel immer den maximalen Wert hat

und dass jeder Weg von der Wurzel zu einem Blatt monoton fallend bzgl. der Schlüsselwerte ist.

Definition 3.5 (Heap-Eigenschaft). Wir sagen ein Array a hat die (*Max-Heap-Eigenschaft*) im Bereich $[l, r]$ falls

$$\begin{aligned} l \leq i \leq r, l \leq 2i \leq r &\implies a[i] \geq a[2i], \\ l \leq i \leq r, l \leq 2i + 1 \leq r &\implies a[i] \geq a[2i + 1] \end{aligned}$$

gilt.

Die Idee des Heapsort-Algorithmus ist es, zunächst die Heap-Eigenschaft für das gegebene Array a herzustellen. Danach haben wir das größte Element in $a[1]$, d. h. an der Wurzel, stehen. Nun tauschen wir das erste und das letzte Element und stellen die Heap-Eigenschaft wieder für die Elemente von 1 bis $n - 1$ her. Dieses Verfahren wiederholen wir insgesamt n -mal, bis das gesamte Array sortiert ist; vgl. Algorithmus 7.

Algorithmus 7 Heapsort

Eingabe: Ein Array a der Länge n mit $a[i] \in \mathbb{Z}$.

Ausgabe: Das sortierte Array a mit $a[1] \leq a[2] \leq \dots \leq a[n]$.

```

1: for  $i = \lfloor n/2 \rfloor : 1$  do
2:   HEAPIFY( $a, i, n$ )
3: for  $i = n : 2$  do
4:   Tausche  $a[1]$  und  $a[i]$ .
5:   HEAPIFY( $a, 1, i - 1$ )
6: return  $a$ 

1: procedure HEAPIFY( $a, l, r$ )
2:   Setze  $f \leftarrow l$  und  $s \leftarrow 2f$ .
3:   while  $s \leq r$  do
4:     if  $s + 1 \leq r$  und  $a[s] \leq a[s + 1]$  then
5:       Setze  $s \leftarrow s + 1$ .
6:     if  $a[f] \leq a[s]$  then
7:       Tausche  $a[f]$  und  $a[s]$ .
8:       Setze  $f \leftarrow s$  und  $s \leftarrow 2f$ .
9:     else
10:      break
```

Algorithmus 7 läuft tatsächlich auch im Worst-Case in $\mathcal{O}(n \log n)$.

Beispiel 13. Wir wollen das Array $(4, 1, 6, 2, 2, 7)$ mit dem Heapsort-Algorithmus sortieren. Blaue Elemente zeigen wieder einen anstehenden Tausch an.

Heapsort	4	1	6	2	2	7
HEAPIFY(a,3,6)	4	1	6 _f	2	2	7 _s
↔	4	1	7 _f	2	2	6 _s
Update	4	1	7	2	2	6 _f
HEAPIFY(a,2,6)	4	1 _f	7	2 _s	2	6
s++	4	1 _f	7	2	2 _s	6
↔	4	2 _f	7	2	1 _s	6
Update	4	2	7	2	1 _f	6
HEAPIFY(a,1,6)	4 _f	2 _s	7	2	1	6
s++	4 _f	2	7 _s	2	1	6
↔	7 _f	2	4 _s	2	1	6
Update	7	2	4 _f	2	1	6 _s
↔	7	2	6 _f	2	1	4 _s
Update	7	2	6	2	1	4 _f
↔	4	2	6	2	1	7
HEAPIFY(a,1,5)	4 _f	2 _s	6	2	1	7
s++	4 _f	2	6 _s	2	1	7
↔	6 _f	2	4 _s	2	1	7
Update	6	2	4 _f	2	1	7
↔	1	2	4	2	6	7
HEAPIFY(a,1,4)	1 _f	2 _s	4	2	6	7
s++	1 _f	2	4 _s	2	6	7
↔	4 _f	2	1 _s	2	6	7
Update	4	2	1 _f	2	6	7
↔	2	2	1	4	6	7
HEAPIFY(a,1,3)	2 _f	2 _s	1	4	6	7
↔	2 _f	2 _s	1	4	6	7
Update	2	2 _f	1	4	6	7
↔	1	2	2	4	6	7
HEAPIFY(a,1,2)	1 _f	2 _s	2	4	6	7
↔	2 _f	1 _s	2	4	6	7
Update	2	1 _f	2	4	6	7
↔	1	2	2	4	6	7
HEAPIFY(a,1,1)	1 _f	2	2	4	6	7
Fertig	1	2	2	4	6	7

△

Satz 3.6. Die Laufzeit von Algorithmus 7 ist $\mathcal{O}(n \log n)$.

Beweis. Ein Durchlauf der Funktion HEAPIFY benötigt Laufzeit $\gamma(\log r - \log l)$ mit einer Konstanten $\gamma \in \mathbb{N}$. Insgesamt gilt damit für die Gesamtlaufzeit, wobei $\delta \in \mathbb{N}$ eine weitere Konstante ist:

$$\begin{aligned} T(n) &= \delta \left(\sum_{i=1}^n \gamma(\log n - \log i) + \sum_{i=1}^n \gamma(\log i - \log 1) \right) \\ &= \delta \gamma \sum_{i=1}^n \log n \\ &= \delta \gamma n \log n. \end{aligned} \quad \square$$

Damit ist Heapsort ein Algorithmus der auch im Worst-Case in $\mathcal{O}(n \log n)$ läuft.

Jetzt stellt sich natürlich die Frage, ob es nicht noch besser geht? In der Tat geht es nicht besser, sofern man sich auf Algorithmen beschränkt, die auf dem paarweisen Vergleichen der Schlüssel basieren. Diese Gegebenheit wollen wir nun beweisen.

Lemma 3.7 (Produktformel von Wallis).

$$\sqrt{\pi} = \lim_{n \rightarrow \infty} \frac{2^{2n} \cdot (n!)^2}{(2n)! \cdot \sqrt{n + \frac{1}{2}}}. \quad (3.1)$$

Beweis. Aus der Produktentwicklung der Sinus-Funktion,

$$\sin(\pi x) = \pi x \cdot \prod_{k=1}^{\infty} \left(1 - \frac{\pi x}{\pi k}\right) \left(1 + \frac{\pi x}{\pi k}\right) = \pi x \cdot \prod_{k=1}^{\infty} \left(1 - \frac{x^2}{k^2}\right)$$

folgt für $x = \frac{1}{2}$

$$1 = \frac{\pi}{2} \cdot \prod_{k=1}^{\infty} \frac{4k^2 - 1}{4k^2},$$

und durch Umstellung

$$\frac{\pi}{2} = \prod_{k=1}^{\infty} \frac{(2k)^4}{(2k-1)2k \cdot 2k(2k+1)} = \lim_{n \rightarrow \infty} \frac{2^{4n} \cdot (n!)^4}{((2n)!)^2 (2n+1)}.$$

Weiter erhalten wir

$$\pi = \lim_{n \rightarrow \infty} \frac{2^{4n} \cdot (n!)^4}{((2n)!)^2 (n + \frac{1}{2})}$$

und schließlich die Behauptung

$$\sqrt{\pi} = \lim_{n \rightarrow \infty} \frac{2^{2n} \cdot (n!)^2}{(2n)! \cdot \sqrt{n + \frac{1}{2}}}.$$

□

Als nächstes benötigen wir die Stirling-Formel, mit der man große Fakultäten näherungsweise berechnen kann.

Satz 3.8 (Stirling-Formel). Für alle natürlichen Zahlen $n \geq 1$ gilt

$$\sqrt{2\pi n} \left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \cdot e^{\frac{1}{12n}}.$$

Beweis. Zuerst zeigen wir, dass die Folge

$$a_n = \frac{n!}{\left(\frac{n}{e}\right)^n \cdot \sqrt{n}}$$

monoton fällt. Da alle Folgenglieder positiv sind, konvergiert diese Folge somit. Die Division aufeinanderfolgender Glieder ergibt

$$\frac{a_n}{a_{n+1}} = \frac{1}{e} \cdot \left(\frac{n+1}{n}\right)^{n+\frac{1}{2}}$$

und durch Anwendung des Logarithmus

$$\log \frac{a_n}{a_{n+1}} = -1 + \left(n + \frac{1}{2}\right) \cdot \log \frac{n+1}{n}. \quad (3.2)$$

Für $|x| < 1$ gilt die Potenzreihen-Entwicklung

$$\log \frac{1+x}{1-x} = 2 \cdot \left(x + \frac{x^3}{3} + \frac{x^5}{5} + \dots\right),$$

also für $0 < x < 1$ die Abschätzung

$$2x < \log \frac{1+x}{1-x} < 2 \cdot \left(x + \frac{x^3}{3} \cdot (1+x^2+x^4+\dots)\right) = 2x + \frac{2}{3} \cdot \frac{1}{\frac{1}{x}(\frac{1}{x^2}-1)}. \quad (3.3)$$

Setzt man $x = 1/(2n+1)$, so ist

$$\frac{1+x}{1-x} = \frac{n+1}{n}$$

und mit (3.3) erhalten wir

$$\frac{1}{n+\frac{1}{2}} < \log \frac{n+1}{n} < \frac{1}{n+\frac{1}{2}} + \frac{2}{3} \cdot \frac{1}{(2n+1)((2n+1)^2-1)}.$$

Multiplikation mit $n + \frac{1}{2}$ liefert

$$1 < \left(n + \frac{1}{2}\right) \cdot \log \frac{n+1}{n} < 1 + \frac{1}{3} \cdot \frac{1}{4n^2 + 4n}$$

und indem wir -1 addieren erhalten wir mit (3.2)

$$0 < \log \frac{a_n}{a_{n+1}} < \frac{1}{12} \cdot \left(\frac{1}{n} - \frac{1}{n+1} \right),$$

woraus $a_n > a_{n+1}$ folgt.

Sei nun $a = \lim_{n \rightarrow \infty} a_n$. Dann ist $a \geq 0$ und

$$0 < \log \frac{a_n}{a_{n+k}} < \frac{1}{12} \cdot \left(\frac{1}{n} - \frac{1}{n+k} \right).$$

Für $k \rightarrow \infty$ folgt nun

$$0 \leq \log \frac{a_n}{a} \leq \frac{1}{12n}$$

und durch Anwendung der Exponentialfunktion

$$1 \leq \frac{a_n}{a} \leq e^{\frac{1}{12n}}.$$

Zum Beweis der Behauptung ist also noch $a = \sqrt{2\pi}$ zu zeigen.

Aus Lemma 3.7 folgt nun zusammen mit $n! = (a_n n^{n+\frac{1}{2}})/e^n$ sofort

$$\sqrt{\pi} = \lim_{n \rightarrow \infty} \frac{a_n^2 \cdot n^{2n+1} \cdot 2^{2n} \cdot e^{2n}}{e^{2n} \cdot a_{2n} \cdot (2n)^{2n+\frac{1}{2}} \cdot \sqrt{n+\frac{1}{2}}} = \frac{a}{\sqrt{2}}.$$

Also ist $a = \sqrt{2\pi}$. □

Satz 3.9. Jeder (deterministische)³ Sortieralgorithmus, der auf paarweisen Vergleichen der Schlüssel basiert, braucht im Worst-Case $\Omega(n \log n)$ Vergleiche.

Beweis. Jeder Sortieralgorithmus, der auf paarweisen Vergleichen basiert, kann durch einen Entscheidungsbaum dargestellt werden. Dabei stellen die inneren Knoten des Baums die Vergleiche im Algorithmus dar, wobei man vereinbart, dass wenn der Vergleich **wahr** ist, links im Baum weitergelaufen wird, und wenn der Vergleich **falsch** ist, rechts im Baum weitergelaufen wird. Die Blätter des Baums entsprechen den möglichen Sortierungen des Arrays.

Bei jeder Eingabe eines Arrays wird ein Weg in diesem Entscheidungsbaum durchlaufen. Jeder Weg von der Wurzel des Baums bis zu einem Blatt entspricht dabei der Folge der angestellten Vergleiche.

Da es bei einer Eingabe von n Zahlen, insgesamt $n!$ verschiedene Sortierungen geben kann, hat dieser Baum $n!$ verschiedene Blätter. Daraus folgt, dass

³Ein deterministischer Algorithmus ist ein Algorithmus, bei dem nur definierte und reproduzierbare Zustände auftreten.

in diesem Entscheidungsbaum ein Weg der Länge mindestens $\log(n!)$ existiert. Mit anderen Worten, es gibt eine Eingabe, bei der mindestens $\log(n!)$ Vergleiche durchgeführt werden müssen.

Es bleibt also die Aufgabe, $\log(n!)$ abzuschätzen. Mit Satz 3.8 gilt

$$\sqrt{2\pi n} \left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \cdot e^{\frac{1}{12n}}.$$

Damit erhalten wir für die Laufzeit $T_{\text{wc}}(n)$ im Worst-Case:

$$\begin{aligned} T_{\text{wc}}(n) &\geq \log(n!) \\ &\geq \log\left(\sqrt{2\pi n} \left(\frac{n}{e}\right)^n\right) \\ &= n(\log n - \log e) + \frac{1}{2} \log(2\pi n) \\ &\in \Omega(n \log n). \end{aligned} \quad \square$$

Für vergleichsbasierte Sortierverfahren geht es also nicht besser als $\mathcal{O}(n \log n)$. Stellen wir die nächste Frage: Gibt es Verfahren, die nicht auf paarweisen Vergleichen von Schlüsseln basieren?

3.5 Bucketsort

Die Idee des Bucketsort-Verfahrens ist, auf die Werte der Einträge selbst zu schauen. Nehmen wir einmal an, wir wüssten, dass alle auftretenden Zahlen zwischen 1 und 100 liegen. Dann können wir 100 Eimer (engl.: *bucket*) bilden, wobei wir in Eimer i alle Array-Einträge j mit $a[j] = i$ aufnehmen. Dann laufen wir einmal durch das Array und weisen jedem Element seinen entsprechenden Eimer zu. Danach laufen wir noch einmal durch alle Eimer und ordnen die Elemente in der auftretenden (sortierten) Reihenfolge an.

Im Allgemeinen ist es natürlich nicht so einfach, da das größte Element sehr groß sein kann. Damit muss die Anzahl *Buckets* reduziert werden, was dazu führen kann, dass innerhalb der einzelnen Buckets nochmal sortiert werden muss. Ziel ist es also, eine Anzahl B an Buckets und eine Funktion

$$f : \{a[1], \dots, a[n]\} \rightarrow \{1, \dots, B\}$$

mit folgenden Eigenschaften zu finden:

1. $f(a[i]) \leq f(a[j]) \implies a[i] \leq a[j]$ für alle i, j
2. $|\{a[i] : i \in \{1, \dots, n\} \text{ und } f(a[i]) = b\}| \leq \Theta$ für alle $b = 1, \dots, B$ mit geeigneter Konstante Θ

Eigenschaft 1 ist Voraussetzung, um abschließend in einem Durchlauf durch alle Buckets die endgültige Sortierung zu erhalten. Eigenschaft 2 ist hingegen maßgeblich für die Laufzeit von Bucketsort, da diese die Verteilung der Funktionswerte von f bestimmt.

Wir wollen im Rahmen dieser Vorlesung nicht genauer auf solche Funktionen und deren Eigenschaften eingehen, sondern lediglich den allgemeinen Rahmen der Klasse der Bucketsort-Verfahren diskutieren; vgl. Algorithmus 8.

Algorithmus 8 Bucketsort

Eingabe: Ein Array a der Länge n mit $a[i] \in \mathbb{Z}$.

Ausgabe: Das sortierte Array a mit $a[1] \leq a[2] \leq \dots \leq a[n]$.

1: Bestimme die Anzahl der Buckets B und eine Funktion

$$f : \{a[1], \dots, a[n]\} \rightarrow \{1, \dots, B\}.$$

2: **for** $i = 1 : n$ **do**

3: Weise $a[i]$ Bucket $f(a[i])$ zu.

4: **for** $b = 1 : B$ **do**

5: Sortiere Bucket b .

6: Kopiere Elemente aus dem Bucket in a zurück.

7: **return** a

Für $\nu_b := |\{a[i] : i \in \{1, \dots, n\} \text{ und } f(a[i]) = b\}|$ ist die Laufzeit von Bucketsort $\mathcal{O}(n) + \sum_{b=1}^B \mathcal{O}(\nu_b \log \nu_b)$. Durch Angabe einer geeigneten Konstanten Θ erfolgt eine Gleichverteilung in den Buckets, wodurch die Summe über die Buckets linear ist und die jeweiligen Summanden somit als konstant angesehen werden können. In diesem Fall hat Bucketsort eine Laufzeit von $\mathcal{O}(n)$ Zeit und ist damit schneller als die Sortieralgorithmen, die auf paarweisen Vergleichen beruhen.

Kapitel 4

Bäume und Wälder

In diesem Kapitel beschäftigen wir uns mit dem Problem in einem Graphen mit Kantengewichten einen *aufspannenden Baum minimalen Gewichts* oder einen *Wald maximalen Gewichts* zu finden. Diese beiden Probleme sind in direkter Weise äquivalent; siehe Übung.

4.1 Grundlegende Definitionen

Die wichtigsten Definitionen für dieses Kapitel haben wir bereits in Abschnitt 3.4 eingeführt. Zusätzlich benötigen wir noch einige Begriffe, die wir nun definieren wollen.

Für den einfacheren Sprachgebrauch nennen wir im Folgenden auch eine Kantenmenge $B \subseteq E$ einen *Wald* in einem Graphen $G = (V, E)$, falls $(V(B), B)$ keinen Kreis enthält. Ist $(V(B), B)$ außerdem zusammenhängend, so nennen wir die Kantenmenge B einen *Baum*.

Definition 4.1 (Wald maximalen Gewichts). Gegeben sei ein Graph $G = (V, E)$ mit Kantengewichten $c_e \in \mathbb{R}$ für alle $e \in E$. Ein Wald $\bar{E} \subseteq E$ mit $c(\bar{E}) \geq c(E')$ für alle Wälder $E' \subseteq E$ wird *Wald maximalen Gewichts* genannt.

Definition 4.2 (Aufspannender Baum). Es sei $G = (V, E)$ ein Graph. Ein Graph $\bar{B} = (\bar{V}, \bar{E})$ heißt *aufspannender Baum* von G , falls \bar{B} ein Baum ist und $\bar{V} = V$ sowie $\bar{E} \subseteq E$ gilt.

Definition 4.3 (Aufspannender Baum minimalen Gewichts). Es sei $G = (V, E)$ ein Graph mit Kantengewichten $c_e \in \mathbb{R}$ für alle $e \in E$. Ein aufspannender Baum $\bar{E} \subseteq E$ mit $c(\bar{E}) \leq c(E')$ für alle aufspannenden Bäume $E' \subseteq E$ wird *aufspannender Baum minimalen Gewichts* (*minimaler Spannbaum*) genannt.

Definition 4.4 (Maximaler Wald minimalen Gewichts). Sei $\bar{E} \subseteq E$ ein Wald mit $|\bar{E}| \geq |E'|$ für alle Wälder $E' \subseteq E$. Gilt für alle Wälder $\tilde{E} \subseteq E$ mit $|\bar{E}| = |\tilde{E}|$ die Ungleichung $c(\bar{E}) \leq c(\tilde{E})$, wird \bar{E} *maximaler Wald minimalen Gewichts* genannt.

Bemerkung 4.5. Wenn G nicht zusammenhängend ist, dann existiert kein aufspannender Baum von G .

Bemerkung 4.6. Wenn G zusammenhängend ist, so ist ein maximaler Wald minimalen Gewichts identisch mit einem aufspannenden Baum minimalen Gewichts.

4.2 Maximale Wälder minimalen Gewichts und minimale Spannbäume

In diesem Abschnitt beschäftigen wir uns mit Algorithmen zur Bestimmung von maximalen (bzgl. Kanteninklusion) Wäldern minimalen Gewichts und mit Algorithmen zur Bestimmung minimaler Spannbäume in zusammenhängenden Graphen. Das erste Verfahren ist der Algorithmus 9 von Kruskal.

Algorithmus 9 Algorithmus von Kruskal, 1956

Eingabe: Graph $G = (V, E)$, Gewichte c_e für $e \in E$

Ausgabe: Maximaler Wald (bzgl. Kantenzahl) $W \subseteq E$ minimalen Gewichts.

1: Sortiere die Kantengewichte in nicht-absteigender Reihenfolge

$$c_{e_1} \leq c_{e_2} \leq \dots \leq c_{e_m}.$$

2: Setze $W \leftarrow \emptyset$.

3: **for** $i = 1 : m$ **do**

4: **if** $W \cup \{e_i\}$ enthält keinen Kreis **then**

5: Setze $W \leftarrow W \cup \{e_i\}$.

6: **return** W

Die Laufzeit des Algorithmus von Kruskal beträgt

$$\mathcal{O}(m \log m + mn) = \mathcal{O}(mn),$$

wobei $m = |E|$ und $n = |V|$ gilt. Der erste Term $m \log m$ resultiert aus der Sortierung im ersten Schritt des Algorithmus. Danach müssen wir m (for-Schleife) Kreisprüfungen durchführen. Letzteres ist in $\mathcal{O}(n)$ möglich, wodurch wir insgesamt $\mathcal{O}(mn)$ erhalten.

Beispiel 14. Wir wollen einen aufspannenden Baum minimalen Gewichts im Graphen aus Abbildung 4.1 finden. Zu Beginn werden die Kanten nach

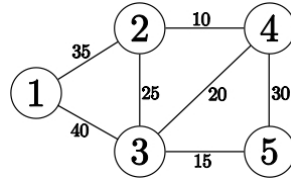


Abbildung 4.1: Ausgangsgraph im Beispiel 14 zum Kruskal-Algorithmus

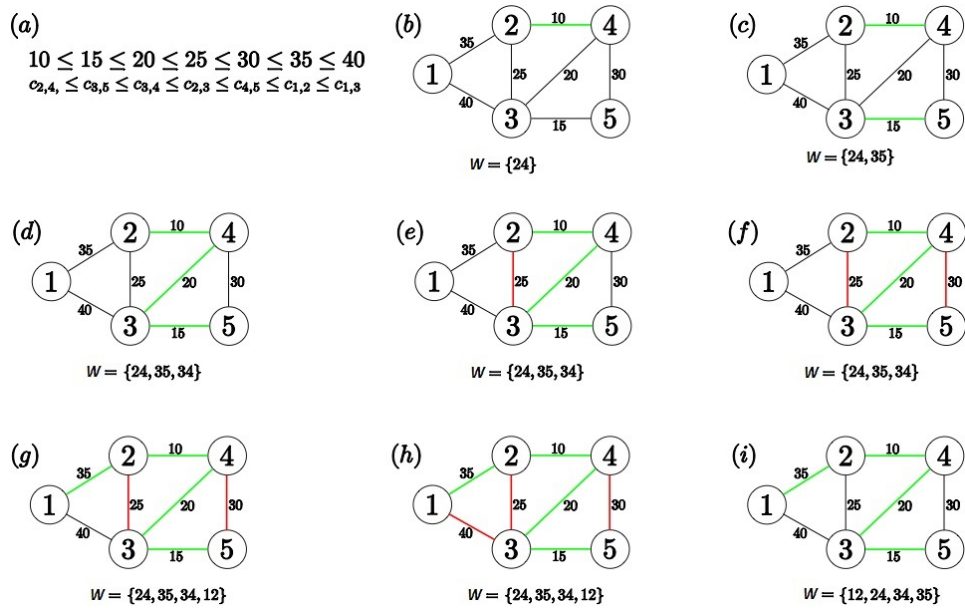


Abbildung 4.2: Bestimmung eines maximalen Waldes minimalen Gewichts nach Kruskal

ihrer Gewichtung sortiert, in unserem Beispiel haben wir also $10 \leq 15 \leq 20 \leq 25 \leq 30 \leq 35 \leq 40$, wie es in Grafik (a) in Abbildung 4.2 zu sehen ist.

Anschließend werden die Kanten, die keinen Kreis bilden, von der kleinsten bis zur größten Kantengewichtung, vereinigt und bilden die Menge W .

Die Kante mit der kleinsten Kantengewichtung ist 24, sie ist also die erste Kante (siehe (b)), die mit der Menge W vereinigt wird. Die zweite ist 35, da 24 und 35 keinen Kreis bilden, wie es in (c) zu sehen ist. Die nächste Kante ist 34; da auch 24, 35 und 34 keinen Kreis bilden, werden diese zur Menge W vereinigt, wie es in (d) zu sehen ist. Die nächste Kante ist 23. Die Kanten 24, 35, 34 und 23 bilden aber einen Kreis und verstoßen somit gegen Schritt 4 im Pseudocode. Also wird die Kante 23 nicht genommen, daher ist sie in (e) rot markiert. Die nächste Kante ist 45; auch $W \cup \{45\}$ (zur Erinnerung: $W = \{24, 35, 34\}$) bilden einen Kreis, womit Kante 45 auch entfällt; also bekommt 45 eine rote Färbung. Nun ist Kante 12 an der Reihe, da die Kanten in $W \cup \{12\}$ keinen Kreis bilden, wird 12 mit W vereinigt. Nun ist Kante 13 an der Reihe. Aber auch hier enthält $W \cup \{13\}$ einen Kreis, und Kante 13 wird daher nicht mit aufgenommen, wie in (h) zu sehen.

Die Schleife endet, wenn der Laufindex i bei m angekommen ist, also die Schleife m -mal durchlaufen wurde. Da unser Graph aus genau sieben Kanten besteht, wird die Schleife 7-mal durchlaufen und endet dann. Der Code kommt also nun bei Schritt 6 im Pseudocode an, wo W ausgegeben wird. W , also der maximale Wald minimalen Gewichts, ist gegeben durch $W = \{24, 35, 34, 12\}$, wie in (i) zu sehen ist. \triangle

Satz 4.7. Der Algorithmus von Kruskal ist korrekt.

Beweis. Wir zeigen, dass es nach dem i -ten Schritt einen (bzgl. Kanteninklusion) maximalen Wald \bar{W} minimalen Gewichts mit $\bar{W} \cap \{e_1, \dots, e_i\} = W \cap \{e_1, \dots, e_i\}$ gibt. Das erledigen wir mittels Induktion über i .

Für $i = 0$ ist die Behauptung offensichtlich erfüllt.

Jetzt zeigen wir noch den Induktionsschritt $(i - 1 \rightarrow i)$. Hier wird zwischen zwei Fällen unterschieden:

1. $W \cup \{e_i\}$ enthält einen Kreis, d. h., $e_i \notin W$.

Da $\bar{W} \cap \{e_1, \dots, e_{i-1}\} = W \cap \{e_1, \dots, e_{i-1}\}$ gilt, enthält auch $\bar{W} \cup \{e_i\}$ einen Kreis und somit ist $e_i \notin \bar{W}$.

2. $W \cup \{e_i\}$ enthält keinen Kreis, d. h., $e_i \in W$.

(a) Wenn $e_i \in \bar{W}$ gilt, ist nichts zu zeigen.

(b) $e_i \notin \bar{W}$. Damit enthält $\bar{W} \cup \{e_i\}$ einen Kreis C . Somit muss es ein $j > i$ geben mit $e_j \in \bar{W}$, $e_j \notin W$ und $c_{e_j} \geq c_{e_i}$, da $\bar{W} \cap$

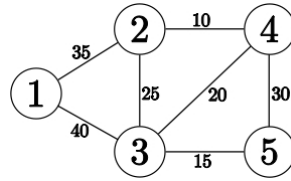


Abbildung 4.3: Ausgangsgraph im Beispiel 15

$\{e_1, \dots, e_{i-1}\} = W \cap \{e_1, \dots, e_{i-1}\}$ und $W \cup \{e_i\}$ keinen Kreis enthält. Das heißt, $\bar{W}' = (\bar{W} \setminus \{e_j\}) \cup \{e_i\}$ ist auch ein (bzgl. Kanteninklusion) maximaler Wald minimalen Gewichts, der die Behauptung erfüllt.

Für $i = m$ folgt schließlich die Behauptung. \square

Bemerkung 4.8. Der Kruskal-Algorithmus gehört zu der Klasse der *Greedy-Algorithmen*. “Greedy” bedeutet soviel wie gefräßig oder gierig. Der Kruskal-Algorithmus ist *greedy*, weil er in jedem Schritt diejenige Kante auswählt, die hinsichtlich der Kostenfunktion bestmöglich ist. Wir werden in Kapitel 8 näher auf Greedy-Algorithmen eingehen.

Ein gemeinsames Gerüst für viele Verfahren zur Bestimmung minimaler Spannbäume in zusammenhängenden Graphen ist in Algorithmus 10 gegeben.

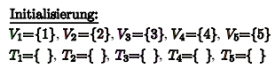
Algorithmus 10 Bestimmung minimaler Spannbäume

Eingabe: Ein zusammenhängender Graph $G = (V, E)$ mit Gewichten c_e für alle Kanten $e \in E$.

Ausgabe: Aufspannender Baum $T \subseteq E$ minimalen Gewichts.

- 1: Für alle $i \in V$ setze $V_i \leftarrow \{i\}$ und $T_i \leftarrow \emptyset$.
 - 2: **for** $|V| - 1$ mal **do**
 - 3: Wähle nichtleeres V_i .
 - 4: Wähle $uv \in \delta(V_i)$ mit $u \in V_i$ und $c_{uv} \leq c_{pq}$ für alle $pq \in \delta(V_i)$.
 - 5: Bestimme j , so dass $v \in V_j$.
 - 6: Setze $V_i \leftarrow V_i \cup V_j$, $V_j \leftarrow \emptyset$ und $T_i \leftarrow T_i \cup T_j \cup \{uv\}$, $T_j \leftarrow \emptyset$.
 - 7: **return** T_i mit $T_i \neq \emptyset$.
-

Beispiel 15. Gesucht ist wieder ein minimal aufspannender Baum im Graphen aus Abbildung 4.3. Zunächst müssen wir laut Schritt 1 des Pseudocodes die Initialisierung vornehmen. Da die Menge V genau 5 Elemente enthält, erstellen wir 5 Teilmengen $V_i \subseteq V$ mit $V_i = \{i\}$ für $i \in \{1, 2, 3, 4, 5\}$ und 5 Mengen T_i mit $i \in \{1, 2, 3, 4, 5\}$, wobei vorerst $T_i = \emptyset$ gilt.



dick hervorgehoben ist. Die zulässigen Kanten sind wieder blau markiert und die zu wählende Kante ist die Kante 34. Wir suchen also ein j für das $4 \in V_j$, also ist $j = 4$. Nun wird in Schritt 6 erneut V_5 und V_4 redefiniert, indem wir V_5 als Vereinigung der beiden Mengen V_5 und V_4 definieren und anschließend $V_4 = \emptyset$ setzen. Auch muss wieder T_5 und T_4 wie gehabt umdefiniert werden. T_5 wird als Vereinigung von T_5, T_4 und $\{34\}$ definiert und anschließend $T_4 = \emptyset$ gesetzt. Die neu entstandenen Mengen kann man nun in Grafik (d) sehen. Da dies erst der 3. Durchlauf war, benötigen wir noch einen 4-ten und kehren daher erneut zu Schritt 3.) im Pseudocode zurück.

Die einzigen verbleibenden nicht-leeren Mengen V_i sind V_1 und V_5 . Dieses mal wählen wir die Menge V_1 , welche in Grafik (d) fett hervorgehoben ist. Die gültigen Kanten sind wie gehabt blau markiert, also 12, 23 und 24. Die Kante mit dem geringsten Kantengewicht ist 24, welche wir daher auch wählen. Da der Knoten 4 ein Element der Mengen V_5 ist, wählen wir $j = 5$ und kommen zu Schritt 6. Hier wird wie gehabt die Menge $V_1 = V_1 \cup V_5$ redefiniert und anschließend $V_5 = \emptyset$ gesetzt. Die Kantenmenge T_1 wird nun noch mit der Menge T_5 und der Kante $\{24\}$ vereinigt und als T_1 definiert. Die nun entstandenen Mengen sind in Grafik (e) zu sehen.

Da Schritt 2 nun 4-mal ausgeführt wurde, gehen wir nun weiter zu Schritt 7, in dem wir $T_i = T_1$ ausgegeben. Die Ausgabe $T_1 = \{12, 24, 34, 35\}$ ist also der aufspannende Baum minimalen Gewichts. Wir können sehen, dass dies die gleichen Kanten sind, die auch mit dem Algorithmus von Kruskal gefunden wurden. \triangle

Satz 4.9. Algorithmus 10 ist korrekt.

Beweis. Wir zeigen durch Induktion über $k = \sum_{i \in V} |T_i|$, dass G einen minimalen aufspannenden Baum T mit $T_i \subseteq T$ für alle i enthält.

Die Induktionsannahme ($k = 0$) ist trivial. Wir zeigen jetzt den Induktionsschritt ($k - 1 \rightarrow k$). Sei uv die hinzugefügte Kante. Das heißt, in der aktuellen Iteration des Verfahrens ist $u \in V_i$ und $v \in V_j$ für passende i und j . Außerdem gibt es (nach Induktionsannahme) einen minimal aufspannenden Baum T mit $T_i \subseteq T$ für alle bisher bestimmten Mengen T_i .

Der Fall $uv \in T$ erfüllt die Behauptung. Sei also $uv \notin T$. Dann muss $T \cup \{uv\}$ einen Kreis enthalten. Also gibt es eine Kante $rs \in T$ mit $r \in V_i$ und $s \in V \setminus V_i$. Nach Wahl der Kante uv in Schritt 4 gilt $c_{rs} \geq c_{uv}$. Also ist $(T \setminus \{rs\}) \cup \{uv\}$ ebenfalls ein minimal aufspannender Baum, der die Bedingung der Induktionsannahme erfüllt. \square

Ein Spezialfall von Algorithmus 10 ist der Algorithmus 11 von Prim, der auf R. C. Prim zurückgeht.

Algorithmus 11 Algorithmus von Prim, 1957

Eingabe: Ein zusammenhängender Graph $G = (V, E)$ mit Gewichten c_e für alle Kanten $e \in E$.

Ausgabe: Ein aufspannender Baum $T \subseteq E$ minimalen Gewichts.

- 1: Wähle $w \in V$ beliebig und setze $T \leftarrow \emptyset, W \leftarrow \{w\}$ und $V' \leftarrow V \setminus \{w\}$.
 - 2: **while** $V' \neq \emptyset$ **do**
 - 3: Wähle $c_{uv} = \min\{c_e : e \in \delta(W)\}$ mit $u \in W$ und $v \in V'$.
 - 4: Setze $T \leftarrow T \cup \{uv\}, W \leftarrow W \cup \{v\}$ und $V' = V' \setminus \{v\}$.
 - 5: **return** T .
-

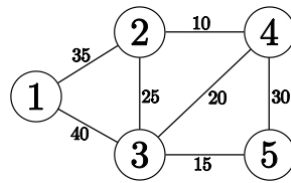


Abbildung 4.5: Ausgangsgraph im Beispiel 16

Die Laufzeit des Algorithmus von Prim ist $\mathcal{O}(n^2)$.¹ Für vollständige Graphen ist dies bestmöglich.

Beispiel 16. Gesucht ist wieder ein minimal aufspannender Baum im Graphen aus Abbildung 4.5. Wie aus dem Pseudocode erkennbar, dürfen wir uns wieder einen beliebigen Knoten aus V wählen, der Einfachheit halber wählen wir wieder Knoten 1.

Nun werden wieder alle Kanten betrachtet, die vom Knoten 1 ausgehen, in der Grafik (a) blau gekennzeichnet; diese wären 12 und 13. Da 12 die kleinere Kantengewichtung hat, wählen wir diese und fügen sie in T ein, fügen Knoten 2 in die Menge W ein und nehmen Knoten 2 aus der Menge V' . Der so entstandene Graph ist in Grafik (b) in Abbildung 4.6 grün gekennzeichnet.

Die nun zur Auswahl stehenden Kanten sind wieder blau gekennzeichnet, dies sind 13, 23 und 24. Aus diesen 3 Kanten wird wieder die mit der geringsten Kantengewichtung gewählt, das wäre Kante 24. Kante 24 wird also wieder in die Menge T einsortiert, der Knoten 4 der Menge W hinzugefügt und aus der Menge V' genommen. Den neu entstandenen Graphen kann man in Grafik (c) wieder in grün sehen, die nun zur Auswahl stehenden Kanten sind wieder blau; dies sind 13, 23, 34 und 45.

Die Kante mit der geringsten Gewichtung ist die Kante 34. Wir fügen also Kante 34 wieder in T ein, fügen Knoten 3 in die Menge W ein und entfernen ihn aus V' . Den nun entstandenen Graphen kann man in Grafik (d) in grün

¹Warum?

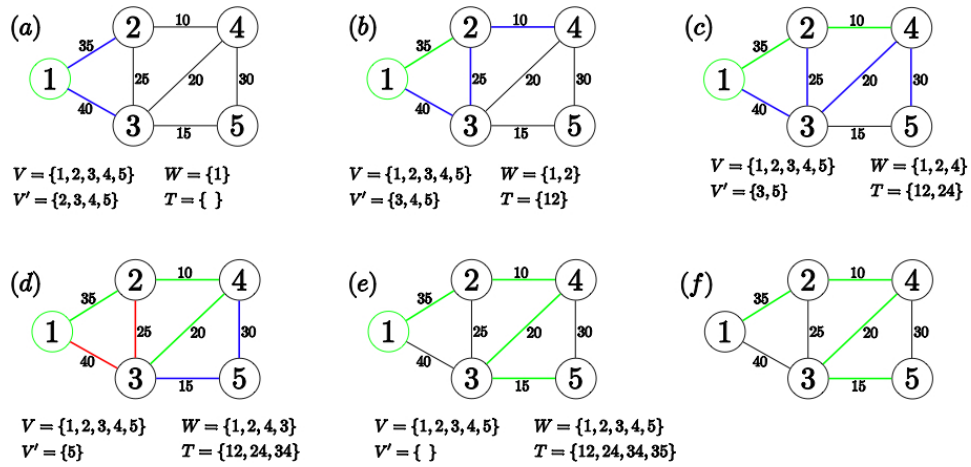


Abbildung 4.6: Bestimmung eines minimal aufspannenden Baums mit Algorithmus 11

sehen und die nun zur Auswahl stehenden Kanten sind wieder blau, also 35 und 45. Die beiden Kanten 13 und 23 stehen nicht zur Verfügung, daher sind sie rot gekennzeichnet, weil $v \in V'$ gilt, aber Knoten 3, 1 und 2 nicht mehr Elemente von V' sind.

Wir wählen also die Kante 35 aus, fügen diese zu T , vereinigen Knoten 5 mit der Menge W und nehmen Knoten 5 aus V' . Die Menge V' ist nun leer und laut dem Rekursionsanker in Punkt 2 soll T ausgegeben werden, wenn $V' = \emptyset$. Also wird T ausgegeben mit $T = \{1, 2, 3, 4, 5\}$. Der aufspannende Baum $T \subseteq E$ minimalen Gewichts ist nun in Grafik (f) in grün zu sehen. \triangle

Kapitel 5

Kürzeste Wege

Gegeben sei ein gerichteter Graph $D = (V, A)$ mit Bogengewichten $c_a, a \in A$, und zwei ausgezeichneten Knoten $s, t \in V$. Das Problem, das wir in diesem Kapitel behandeln werden ist die Bestimmung eines kürzesten Weges von s nach t .

Diese Problemstellung tritt in vielen Anwendungen auf. Die offensichtlichste Anwendung findet dabei wohl in Navigationssystemen auf. Hierbei modellieren, in einfachster Betrachtungsweise, die Kanten des Graphen die Straßen und die Knoten die Städte. Der Startort ist dann s und das Ziel ist der Knoten t .

In weiten Teilen ist dieses Kapitel dem Buch [1] von Ahuja u. a. und dort den Kapiteln 4 und 5 entnommen.

5.1 Eigenschaften kürzester Wege

Lemma 5.1. Ist $(s = i_0, i_1, \dots, i_k = t)$ ein kürzester Weg von s nach t mit $i_\ell \in V, \ell = 0, \dots, k$, so ist auch jeder Teilweg (i_0, \dots, i_ℓ) mit $\ell = 1, \dots, k - 1$ ein kürzester Weg von s nach i_ℓ falls $c_a \geq 0$ für alle $a \in A$ gilt.

Beweis. Siehe Übung. □

Lemma 5.2. Es seien $d : V \rightarrow \mathbb{R}$ die Werte der kürzesten Wege von s nach v für alle $v \in V$. Dann gilt: Ein gerichteter Weg P von s nach v ist genau dann ein kürzester Weg, falls $d(j) = d(i) + c_{ij}$ für alle $(i, j) \in P$.

Beweis. Die Hinrichtung folgt direkt aus Lemma 5.1. Für die Rückrichtung sei $P = (s = i_0, i_1, \dots, i_k = v)$ ein Weg von s nach v . Mit $d(i_0) = d(s) = 0$

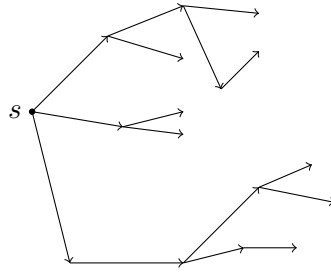


Abbildung 5.1: Kürzeste-Wege-Baum

gilt dann

$$\begin{aligned}
 d(v) &= d(i_k) \\
 &= [d(i_k) - d(i_{k-1})] + [d(i_{k-1}) - d(i_{k-2})] + \dots + [d(i_1) - d(i_0)] \\
 &= \sum_{(i,j) \in P} d(j) - d(i) = \sum_{(i,j) \in P} c_{ij},
 \end{aligned}$$

d. h., P hat die Länge $d(v)$ und somit ist P ein kürzester Weg. \square

Definition 5.3 (Branching, Arboreszenz). Ein Digraph $D = (V, A)$ ist ein *Branching*, falls der zugrunde liegende ungerichtete Graph ein Wald ist und $|\delta^-(v)| \leq 1$ für alle $v \in V$ gilt. Ein zusammenhängendes Branching heißt *Arboreszenz*.

Definition 5.4 (Kürzeste-Wege-Baum). Gegeben sei ein Digraph $D = (V, A)$ und $B \subseteq A$. Eine Arboreszenz (V, B) mit Wurzel s heißt *Kürzester-Wege-Baum*, falls der eindeutige Weg von s nach v in B ein kürzester Weg von s nach v ist für alle $v \in V$.

Lemma 5.5. Ein Kürzester-Wege-Baum existiert, sofern es einen gerichteten Weg von s nach v für alle $v \in V$ gibt.

Beweis. Da es nur endlich viele Wege von s nach v gibt, existiert ein kürzester Weg P_v von s nach v für alle $v \in V$. Lemma 5.2 impliziert, dass $d(j) = d(i) + c_{ij}$ für alle $(i, j) \in P_v$ gilt. Starten wir in s und laufen durch den Graphen per Tiefensuche entlang der Bögen (i, j) , die $d(j) = d(i) + c_{ij}$ erfüllen, so müssen wir jeden Knoten erreichen. Die daraus resultierende Arboreszenz ist ein Kürzester-Wege-Baum. \square

Satz 5.6 (Optimalitätsbedingung). Es seien $d(v)$ für $v \in V$ die Werte beliebiger (d. h., nicht notwendigerweise kürzester) Wege von s nach v mit $d(s) = 0$. Dann gilt: $d(v)$ ist der Wert eines kürzesten Weges von s nach v

genau dann, wenn

$$d(v) \leq d(u) + c_{uv} \quad \text{für alle } (u, v) \in A.$$

Beweis. Die Hinrichtung ist klar. Für die Rückrichtung sei

$$P = (s = i_0, i_1, \dots, i_k = v)$$

ein Weg von s nach v . Dann gilt

$$d(v) \leq d(i_{k-1}) + c_{i_{k-1}i_k} \leq \dots \leq d(i_0) + \sum_{(i,j) \in P} c_{ij} = \sum_{(i,j) \in P} c_{ij}.$$

Das heißt, $d(v)$ ist eine auch untere Schranke für den Wert eines kürzesten Weges von s nach v . Da $d(v)$ nach Voraussetzung auch obere Schranke ist, folgt die Behauptung. \square

Definition 5.7 (Reduzierte Bogengewichte). Für eine Distanzfunktion $d : V \rightarrow \mathbb{R}$ bezeichnen wir mit

$$c_{ij}^d = c_{ij} + d(i) - d(j)$$

die *reduzierten Bogengewichte* bezüglich d .

Lemma 5.8. 1. Für einen gerichteten Kreis C gilt

$$\sum_{(i,j) \in C} c_{ij}^d = \sum_{(i,j) \in C} c_{ij}.$$

2. Für einen gerichteten Weg P von u nach v gilt

$$\sum_{(i,j) \in P} c_{ij}^d = \sum_{(i,j) \in P} c_{ij} + d(u) - d(v).$$

3. Falls $d(\cdot)$ die Werte kürzester Wege sind, gilt

$$c_{ij}^d = c_{ij} + d(i) - d(j) \geq 0 \quad \text{für alle } (i, j) \in A.$$

Beweis. Teil 1. und 2. sind klar. Teil 3. folgt aus Satz 5.6. \square

5.2 Digraphen mit nicht-negativen Gewichten

Satz 5.9. Algorithmus 12 (Dijkstra) arbeitet korrekt.

Beweis. Ein Knoten wird markiert, wenn er in S aufgenommen wird. Per Induktion über die Anzahl der markierten Knoten zeigen wir zwei Invarianten:

Algorithmus 12 Dijkstra, 1959

Eingabe: Ein Digraph $D = (V, A)$, $c_a \geq 0, a \in A$, ein Startknoten $s \in V$.

Ausgabe: Kürzeste gerichtete Wege von s nach v für alle $v \in V$ in folgender Form:

$$d(v) = \text{Distanz von } s \text{ nach } v,$$

$$\text{pred}(v) = \text{Vorgänger von } v \text{ auf dem Weg von } s \text{ nach } v.$$

- 1: Setze $S \leftarrow \emptyset, \bar{S} \leftarrow V, d(j) \leftarrow +\infty$ für alle Knoten $j \in V$.
 - 2: Setze $d(s) \leftarrow 0$ und $\text{pred}(s) \leftarrow s$.
 - 3: **while** $|S| < |V|$ **do**
 - 4: Wähle einen Knoten i mit $d(i) = \min\{d(j) : j \in \bar{S}\}$.
 - 5: Setze $S \leftarrow S \cup \{i\}$ und $\bar{S} \leftarrow \bar{S} \setminus \{i\}$.
 - 6: **for all** $(i, j) \in \delta^{\text{out}}(i)$ **do**
 - 7: **if** $d(j) > d(i) + c_{ij}$ **then**
 - 8: Setze $d(j) \leftarrow d(i) + c_{ij}$.
 - 9: Setze $\text{pred}(j) \leftarrow i$.
 - 10: **return** $d(\cdot)$ und $\text{pred}(\cdot)$.
-

1. Ist v markiert, so enthält $d(v)$ die Länge eines kürzesten Weges.
2. Ist v unmarkiert, so enthält $d(v)$ die Länge eines kürzesten Weges, der als innere Knoten nur markierte Knoten besitzt.

Wir fangen mit der ersten Invariante an. Ist nur ein Knoten markiert (nämlich s), so ist die Behauptung erfüllt (Schritt 2). Wir nehmen an, die Behauptung ist korrekt für k Knoten und wir markieren (Schritte 4-5) den $(k+1)$ -sten Knoten, sagen wir u . Nach Induktionsvoraussetzung wissen wir, $d(u)$ ist der Wert eines kürzesten Weges, der als innere Knoten nur markierte Knoten hat. Angenommen, es gäbe einen kürzeren Weg P . Sei $(v, w) \in P$ mit v markiert und w unmarkiert. Dann gilt

$$c(P) \underbrace{\geq}_{c \geq 0} d(w) \underbrace{\geq}_{\text{Schritt 4}} d(u),$$

ein Widerspruch.

Nun zeigen wir die zweite Invariante. Es bleibt noch zu zeigen, dass für die derzeit unmarkierten Knoten v , $d(v)$ der Wert eines kürzesten Weges ist, dessen innere Knoten alle markiert sind. Sei u der neu markierte Knoten. Angenommen, es gibt einen Weg P von s nach v , dessen innere Knoten alle markierte Knoten sind (inkl. u) und dessen vorletzter Knoten w ungleich u ist und dessen Länge kürzer als die Länge der bislang betrachteten Wege ist. Da $d(w)$ die Länge eines kürzesten (s, w) -Weges enthält, gibt es einen Weg

P' , der nur markierte Knoten enthält, die verschieden von u sind (w wurde vor u markiert wegen Schritt 4). Es folgt ein Widerspruch. \square

Satz 5.10. Die Laufzeit von Algorithmus 12 (Dijkstra) ist $\mathcal{O}(n^2)$.

Beweis. Die Knotenauswahl in Schritt 4 führen wir genau n mal durch und dabei müssen jedes Mal $|\bar{S}|$ viele Distanzlabels testen. Insgesamt haben wir also

$$n + (n - 1) + (n - 2) + \cdots + 1 \in \mathcal{O}(n^2)$$

Operationen. Die Distanzlabel-Aktualisierungen in den Schritten 7–9 können in $\mathcal{O}(1)$ durchgeführt werden. Die Frage ist also, wie oft wir sie durchführen. Da jeder Knoten genau einmal der Knoten i in Schritt 6 ist, haben wir insgesamt

$$\sum_{i \in V} \delta^{\text{out}}(i) = |A| = m$$

Durchläufe. Somit erhalten wir eine Laufzeit von $n^2 + m \in \mathcal{O}(n^2)$. \square

Die Laufzeit von $\mathcal{O}(n^2)$ des Dijkstra-Algorithmus ist bestmöglich für vollständige Graphen, kann aber für dünnbesetzte Graphen verbessert werden. Varianten des Dijkstra-Algorithmus die entweder die theoretische Worst-Case-Komplexität oder die praktische Effizienz des Verfahrens verbessern sind z. B. in den Abschnitten 4.6–4.8 von Ahuja u. a. [1] beschrieben.

5.3 Digraphen mit beliebigen Gewichten

In der Übung werden wir uns ansehen, warum der Dijkstra-Algorithmus nicht für negative Gewichte funktioniert. Bei den bislang behandelten Algorithmen wurde in jeder Iteration das Distanzlabel $d(v)$ für ein $v \in V$ als permanent erklärt, d. h., es wurde gezeigt, dass $d(v)$ tatsächlich der Wert eines kürzesten Weges ist. Solche Verfahren nennt man auch *Label-Setting Algorithmen* oder *Markenfixierungsalgorithmen*. Eine andere Idee, die auf Moore (1957) und Bellmann (1958) zurückgeht ist, sich auf die reduzierten Bogen Gewichte zu konzentrieren. Satz 5.6 und Lemma 5.8 (Teil 3) aus Abschnitt 5.1 sagen aus, dass $d(\cdot)$ genau dann optimal ist, falls

$$c_{ij}^d = c_{ij} + d(i) - d(j) \geq 0 \quad \text{für alle } (i, j) \in A$$

gilt. Die Idee ist nun, Bögen, die das reduzierte Bogenkriterium verletzen, auszuwählen, und die Labels zu reduzieren. Das resultierende Verfahren nennt man daher auch *Label-Correcting Algorithmus* (oder *Markenverbesserungsalgorithmus*). Wir wollen zunächst annehmen, dass D keine (gerichteten) negativen Kreise, d. h., Kreise $C \subseteq A$ mit $c(C) < 0$, enthält.

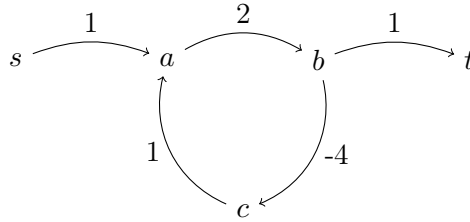


Abbildung 5.2: Beispiel für einen negativen Kreis

Ein Algorithmus zur Bestimmung kürzester Wege für Digraphen mit beliebigen Gewichten, die aber keine negativen Kreise enthalten, ist der Algorithmus 13 von Moore–Bellmann.

Algorithmus 13 Grundversion des Moore–Bellmann-Algorithmus

Eingabe: Ein Digraph $D = (V, A)$ mit Gewichten $c_a, a \in A$, und ohne negative Kreise, ein Startknoten $s \in V$.

Ausgabe: Kürzeste Wege von s nach v , für alle $v \in V$ und deren Längen.

1: Setze

$$d(v) = \begin{cases} 0, & \text{falls } v = s, \\ +\infty, & \text{sonst,} \end{cases}$$

$$\text{pred}(v) = \begin{cases} s, & \text{falls } v = s, \\ \text{ungesetzt,} & \text{sonst.} \end{cases}$$

2: **while** $\exists(i, j) \in A$ mit $d(j) > d(i) + c_{ij}$ **do**

3: Setze $d(j) \leftarrow d(i) + c_{ij}$.

4: Setze $\text{pred}(j) \leftarrow i$.

5: **return** $d(\cdot)$ und $\text{pred}(\cdot)$.

Beispiel 17. Um einzusehen, warum wir negative Kreise ausgeschlossen haben, kann man das Verfahren von Dijkstra oder Algorithmus 13 auf die Instanz aus Abbildung 5.2 anwenden. Wir werden uns später überlegen, wie wir negative Kreise detektieren können. \triangle

Wir zeigen also die Korrektheit von Algorithmus 13 für den Fall, dass D keine negativen Kreise enthält. Der Beweis erfolgt in mehreren Schritten.

Zunächst benötigen wir die folgenden Notationen: Es seien

$$A_k := \{(\text{pred}(i), i) : i \in V \setminus \{s\} \text{ mit } \text{pred}(i) \text{ gesetzt}\},$$

$$H_k := (V(A_k), A_k),$$

der Vorgängergraph von Algorithmus 13 sowie $d_k(\cdot)$ die entsprechenden Distanzlabel nach der k -ten Iteration. Mit dieser Notation zeigen wir die folgende Invariante:

Lemma 5.11. Für $k = 0, 1, 2, \dots$ gilt

$$c_{ij}^{d_k} = c_{ij} + d_k(i) - d_k(j) \leq 0 \quad \text{für alle } (i, j) \in A_k.$$

Beweis. Wir beweisen die Aussage mittels vollständiger Induktion über die Iterationen k von Algorithmus 13.

Für $k = 0$ haben wir $A_0 = \emptyset$ und $H_0 = (\emptyset, \emptyset)$, womit die Behauptung trivialerweise gilt.

Wir zeigen jetzt den Induktionsschritt $(k \rightarrow k + 1)$. Sei (i, j) der Bogen, der in der $(k + 1)$ -ten Iteration hinzugefügt wird. Es gilt also $\text{pred}(j) = i$. Offensichtlich bleiben alle Werte c_{rs}^d unberührt, die nicht Knoten j enthalten (also die, für die $r \neq j$ und $s \neq j$ gilt). Für Bogen $(i, j) \in A_{k+1}$ gilt

$$c_{ij}^{d_{k+1}} = c_{ij} + d_{k+1}(i) - d_{k+1}(j) = 0 \leq 0.$$

Für alle anderen Bögen $(r, l) \in A_{k+1}$ mit $r = j$ erhalten wir

$$0 \geq c_{jl}^{d_k} = c_{jl} + d_k(j) - d_k(l) > c_{jl} + d_{k+1}(j) - d_{k+1}(l) = c_{jl}^{d_{k+1}},$$

da $d_{k+1}(j) < d_k(j)$ gilt. □

Lemma 5.12. Gegeben sei ein Digraph $D = (V, A)$ mit Wurzel $s \in V$ und jeder Knoten $v \in V$ sei von s aus erreichbar. Der Graph $H_k = (V(A_k), A_k)$ ist eine Arboreszenz mit Wurzel s für $k = 0, 1, 2, \dots$, falls D keinen negativen Kreis enthält.

Beweis. Wir zeigen die Aussage durch Kontraposition. Sei k der kleinste Index, so dass H_k einen Kreis C enthält und damit keine Arboreszenz ist. Sei (i, j) der Bogen, der in der k -ten Iteration hinzugefügt wird. Da C ein Kreis ist, enthält H_k einen gerichteten Weg P von j nach i . Weiter gilt

$$\begin{aligned} c(C) &= c^{d_k}(C), & (\text{Lemma 5.8, Teil 1}) \\ &= c^{d_k}(P) + c_{ij}^{d_k} \\ &= c(P) + d_k(j) - d_k(i) + c_{ij}^{d_k} & (\text{Lemma 5.8, Teil 2}) \\ &= c(P) + d_k(j) - d_k(i) + c_{ij} + d_k(i) - d_k(j) \\ &= c(P) + d_k(j) - d_k(i) \\ &< c(P) + d_{k-1}(j) - d_{k-1}(i) \\ &= c^{d_{k-1}}(P) & (\text{Lemma 5.8, Teil 2}) \\ &\leq 0. & (\text{Lemma 5.11}) \end{aligned}$$

D enthält also einen negativen Kreis.

Da $\text{pred}(\cdot)$ eine Funktion von V nach V ist, ist H_k ein Branching. Das heißt, jeder Knoten ist Endknoten maximal eines Bogens aus A_k . Aufgrund der Voraussetzung, dass jeder Knoten von s erreichbar ist, folgt, dass H_k eine Arboreszenz ist. \square

Satz 5.13. Algorithm 13 (Grundversion Moore–Bellmann) arbeitet korrekt. Seine Laufzeit beträgt $\mathcal{O}(n^2\theta)$ für $c_a \in \mathbb{Z}$ und $\theta = \max_{a \in A} |c_a|$.¹

Beweis. Da alle Gewichte ganzzahlig sind, wird in jeder Iteration das Distanzlabel eines Knotens immer um mindestens 1 reduziert. Da der Wert eines kürzesten Weges im Intervall $[-n\theta, n\theta]$ liegt, wird für jeden Knoten i der Wert $d(i)$ maximal $2n\theta$ mal geändert und damit endet Algorithmus 13 nach maximal $2n^2\theta = \mathcal{O}(n^2\theta)$ Iterationen.

Sei k die letzte Iteration, die der Algorithmus durchführt. Dann gilt

$$c_{ij}^{d_k} \geq 0 \quad \text{für alle } (i, j) \in A \quad (5.1)$$

und

$$c_{ij}^{d_k} \leq 0 \quad \text{für alle } (i, j) \in A_k, \quad (5.2)$$

wobei die letzte Ungleichung aus Lemma 5.11 folgt.

Sei P der Weg von s nach i . Nach Lemma 5.8, Teil 2 gilt die Ungleichung

$$0 \geq c^{d_k}(P) = c(P) + d_k(s) - d_k(i) = c(P) - d_k(i).$$

Damit ist $d_k(i)$ eine obere Schranke für den kürzesten Weg von s nach i . Also folgt aus Satz 5.6 und Algorithmus 13, dass $d(\cdot)$ die Werte kürzester Wege von s nach v für alle $v \in V(A_k)$ sind.

Aus den Ungleichungen 5.1 und 5.2 folgt schließlich

$$c_{ij}^{d_k} = 0 \quad \text{für alle } (i, j) \in A_k.$$

Damit ist H_k ein Kürzester-Wege-Baum und $v \in V(A_k)$ genau dann, wenn es einen Weg von s nach v gibt. \square

Bemerkung 5.14. 1. Man kann zeigen, dass Algorithmus 13 (Grundversion Moore–Bellmann) auch für irrationale Gewichte terminiert.

2. Die Laufzeit von Algorithmus 13 ist nicht polynomiell.²

3. Hohe Flexibilität besteht in der Auswahl des Bogens in Schritt 2. Es gibt Auswahlregeln, die zu Polynomialzeitverfahren führen. Einen solchen schauen wir uns jetzt an: Algorithmus 14.

Algorithmus 14 Bellmann, 1958

Eingabe: Ein Digraph $D = (V, A)$ mit Gewichten $c_a \in \mathbb{Z}$ (oder \mathbb{Q}, \mathbb{R}) für $a \in A$ und ohne negative Kreise, ein Startknoten $s \in V$.

Ausgabe: Die kürzesten s - v -Wege mit Längen $d(\cdot)$.

1: Setze

$$d(v) = \begin{cases} 0, & \text{falls } v = s, \\ +\infty, & \text{sonst,} \end{cases}$$
$$\text{pred}(v) = \begin{cases} s, & \text{falls } v = s, \\ \text{ungesetzt,} & \text{sonst.} \end{cases}$$

2: Nummeriere die Bögen in $A = \{a_1, \dots, a_m\}$ in beliebiger Reihenfolge.

3: **for** $k = 1 : n$ **do**

4: **for** $l = 1 : m$ **do**

5: Wähle $a_l = (i, j)$.

6: **if** $d(j) > d(i) + c_{ij}$ **then**

7: Setze $d(j) \leftarrow d(i) + c_{ij}$.

8: Setze $\text{pred}(j) \leftarrow i$.

9: **return** $d(\cdot)$ und $\text{pred}(\cdot)$.

Satz 5.15. Algorithmus 14 (Bellmann) ist korrekt. Seine Laufzeit ist $\mathcal{O}(nm)$.

Beweis. Wir zeigen zunächst per Induktion über die Anzahl der äußeren Schleifendurchläufe (Schritte 3–8), dass nach der k -ten Iteration $d(j)$ (falls $d(j) < \infty$) die Länge eines kürzesten s - j -Weges ist, wobei nur Wege der Länge $\leq k$ zugelassen sind.

Für $k = 1$ gilt die Behauptung offensichtlich. Wir zeigen also den Induktionsschritt ($k \rightarrow k + 1$). Das heißt, die Behauptung gelte nach Iteration k . Für einen beliebigen Knoten $j \in V$ mit $d(j) < \infty$ ist das Label $d(j)$ der Wert eines kürzesten Weges zum Knoten j falls es einen kürzesten Weg der Länge $\leq k$ gibt oder, anderenfalls, $d(j)$ ist eine obere Schranke an den kürzesten s - j -Weg. Im ersten Fall gilt die Behauptung nach Induktionsannahme. Wir betrachten jetzt einen beliebigen Knoten j , der mit dem Knoten s über einen kürzesten Weg

$$P = (s = i_0, i_1, \dots, i_k, i_{k+1} = j)$$

mit der Länge $k + 1$ und mit keinem kürzesten Weg der Länge $\leq k$ verbunden ist. Nach Lemma 5.1 ist dann $(s = i_0, \dots, i_{k-1}, i_k)$ ein kürzester Weg der Länge k von s nach i_k und nach Induktionsannahme enthält $d(i_k)$ nach Iteration k gerade den Wert dieses kürzesten Weges.

¹Zur Erinnerung: $n = |V|$.

²Warum nicht?

Beim $(k + 1)$ -sten Durchlauf wird dann also bei Betrachtung des Bogens (i_k, i_{k+1}) der Wert $d(i_{k+1}) = d(i_k) + c_{i_k i_{k+1}}$ gesetzt und damit auf die Länge des kürzesten Weges P gesetzt. Das war zu zeigen.

Nach $k = |V|$ Iterationen erhalten wir somit, dass $d(j)$ der Wert eines kürzesten s - j -Weges ist für alle $j \in V$.

Die Laufzeit ist offensichtlich $\mathcal{O}(nm)$, da wir wieder annehmen, dass die Schritte 5–8 in $\mathcal{O}(1)$ durchgeführt werden können. \square

Bemerkung 5.16. Man beachte, dass man nicht zwangsläufig alle Schleifendurchläufe $k = 1, 2, \dots, n$ durchführen muss: Man kann bereits abbrechen, wenn sich in einem Durchlauf der inneren l -Schleife keine Distanzlabels mehr geändert haben.

5.3.1 Behandlung negativer Kreise

Die Korrektheit von Algorithmus 13 und 14 benötigt die Eigenschaft, dass D keine negativen Kreise enthält; vgl. Lemma 5.12. Enthält D einen negativen Kreis, so liefern beide Algorithmen nichts sinnvolles bzw. terminieren nicht, da es keine Distanzlabels gibt, die das Optimalitätskriterium erfüllen; vgl. Beispiel 17. Wir wollen nun unsere Algorithmen so modifizieren, dass sie als Eingabe beliebige Digraphen erlauben, und mit der Meldung “ D hat einen negativen Kreis” abbrechen, falls D einen solchen besitzt. Die Frage lautet also, wie man negative Kreise in Digraphen entdecken kann.

Wenn es einen Kreis negativer Länge gibt, so werden Algorithmus 13 und 14 unendlich oft Distanzlabel verringern und nicht stoppen. Wir wissen aber, dass eine $d(j) \geq -n\theta$ gelten muss, falls der Graph keine negativen Kreise enthält. Wir können also detektieren, ob es einen Knoten j mit $d(j) < -n\theta$ gibt und dann den Algorithmus stoppen. Den negativen Kreis kann man schließlich rekonstruieren, indem man die Vorgängerliste **pred** ausgehend vom Knoten j durchläuft.

Diese Detektion hat aber eine Worst-Case Laufzeit von $\mathcal{O}(n\theta)$, wodurch wir die Polynomialität von Algorithmus 14 verlieren.

Eine bessere Variante liefert folgende Beobachtung: Algorithmus 13 und 14 laufen korrekt, solange der aufgebaute Vorgängergraph H_k keinen gerichteten Kreis enthält. Wir können nun nach jeder Iteration überprüfen, ob H_k einen gerichteten Kreis enthält. Dies geht mit der Tiefensuche (vgl. Algorithmus 2 aus Kapitel 2.4) in Linearzeit. Haben wir einen solchen Kreis gefunden, so entspricht er einem negativen Kreis in D und wir können abbrechen (siehe Beweis von Lemma 5.12).

5.4 Kürzeste Wege zwischen allen Knotenpaaren

Als nächstes betrachten wir einen Algorithmus von Floyd und Warshall aus dem Jahr 1962, welcher die kürzesten Wege zwischen allen Knotenpaaren bestimmt.

Algorithmus 15 Floyd–Warshall, 1962

Eingabe: Ein Digraph $D = (V, A)$ mit Knoten $V = \{1, \dots, n\}$ und Gewichten c_a für alle $a \in A$.

Ausgabe: Die $n \times n$ Kürzeste-Weglängen-Matrix W und eine $n \times n$ -Matrix P mit folgenden Eigenschaften:

w_{ij} = Länge eines kürzesten (i, j) -Weges,
 w_{ii} = Länge eines kürzesten (i, i) -Kreises,
 p_{ij} = vorletzter Knoten auf einem kürzesten (i, j) -Weg,
 p_{ii} = vorletzter Knoten auf einem kürzesten (i, i) -Kreis.

1: **for** $i = 1 : n$ **do**
 2: **for** $j = 1 : n$ **do**
 3: Setze

$$w_{ij} = \begin{cases} c_{ij}, & \text{falls } (i, j) \in A, \\ +\infty, & \text{sonst,} \end{cases} \quad \text{und} \quad p_{ij} = \begin{cases} i, & \text{falls } (i, j) \in A, \\ 0, & \text{sonst.} \end{cases}$$

4: **for** $l = 1 : n$ **do**
 5: **for** $i = 1 : n$ **do**
 6: **for** $j = 1 : n$ **do**
 7: **if** $w_{ij} > w_{il} + w_{lj}$ **then**
 8: Setze $w_{ij} \leftarrow w_{il} + w_{lj}$ und $p_{ij} = p_{lj}$.
 9: **if** $i = j$ und $w_{ii} < 0$ **then**
 10: **go to** Schritt 11.
 11: **return** W und P .

Beispiel 18. Wir betrachten den Beispielgraph aus Abbildung 5.3. In diesem Fall ist das Ergebnis des Initialisierungsschrittes

$$(w_{ij})_{1 \leq i, j \leq n} = W = \begin{bmatrix} \infty & 6 & 4 & \infty \\ \infty & \infty & 1 & \infty \\ -4 & \infty & \infty & 3 \\ -2 & 0 & \infty & \infty \end{bmatrix}, \quad (p_{ij})_{1 \leq i, j \leq n} = P = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 3 & 0 & 0 & 3 \\ 4 & 4 & 0 & 0 \end{bmatrix}.$$

△

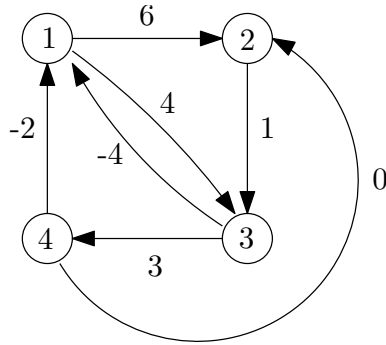


Abbildung 5.3: Graph aus dem Beispiel 18 zum Floyd–Warshall-Algorithmus 15

Bemerkung 5.17. Für (i, j) berechnet sich der kürzeste i - j -Weg aus P wie folgt. Es ist $(i, v_1, v_2, \dots, v_q, j)$ ein kürzester i - j -Weg, wenn

$$\begin{aligned} v_q &= p_{ij}, \\ v_{q-1} &= p_{iv_q}, \\ v_{q-2} &= p_{iv_{q-1}}, \\ &\vdots \\ i &= p_{iv_1} \end{aligned}$$

gilt.

Satz 5.18. Sei $D = (V, A)$ ein Digraph mit beliebigen Bogengewichten c_a für alle $a \in A$. Sei W eine $n \times n$ -Matrix, die vom Floyd–Warshall-Algorithmus berechnet wird. Dann gilt:

- W ist eine Kürzeste-Weglängen-Matrix genau dann, wenn D keine negativen gerichtete Kreise enthält.
- D enthält negative gerichtete Kreise genau dann, wenn ein i existiert für das gilt $w_{ii} < 0$.

Beweis. Zur Vereinfachung der Notation bezeichnen wir die Anfangsmatrix W nach Abarbeitung der Schritte 1–3 mit W^0 und die Matrix W nach Beendigung des l -ten Durchlaufs der äußeren Schleife in Schritt 4 mit W^l .

Durch Induktion über $l = 0, 1, \dots, n$ zeigen wir zunächst, dass W^l genau dann die Matrix der kürzesten Längen von (i, j) -Wegen (bzw. von (i, i) -Kreisen) ist, bei denen die Knoten $1, \dots, l$ als innere Knoten auftreten können, wenn D keinen negativen Kreis in der Knotenmenge $1, \dots, l$ besitzt. Ist letzteres der Fall, so gilt $w_{ii}^l < 0$ für ein $i \in \{1, \dots, l\}$.

Für $l = 0$ ist die Behauptung offenbar richtig. Angenommen, sie ist für $l \geq 0$ richtig, und wir haben die äußere Schleife von Schritt 4 zum $(l + 1)$ -sten Male durchlaufen. Bei diesem Durchlauf haben wir folgende Operation ausgeführt:

$$\text{Falls } w_{ij}^l > w_{i,l+1}^l + w_{l+1,j}^l, \text{ dann setze } w_{ij}^{l+1} = w_{i,l+1}^l + w_{l+1,j}^l.$$

Das heißt, wir haben die (nach Induktionsvoraussetzung) kürzeste Länge eines (i, j) -Weges über die Knoten $1, \dots, l$ verglichen mit der Summe der kürzesten Länge eines $(i, l + 1)$ -Weges und eines $(l + 1, j)$ -Weges; jeweils über die Knoten $1, \dots, l$. Die Summe enthält also die Länge eines kürzesten (i, j) -Weges über $1, \dots, l + 1$, der den Knoten $l + 1$ enthält. Falls diese Summe kleiner als w_{ij}^l ist, setzen wir $w_{ij}^{l+1} = w_{i,l+1}^l + w_{l+1,j}^l$, andernfalls gilt $w_{ij}^{l+1} = w_{ij}^l$. Daraus folgt die Behauptung, es sei denn, $w_{ij}^l > w_{i,l+1}^l + w_{l+1,j}^l$ und die Verkettung, sagen wir K , des $(i, l + 1)$ -Weges mit dem $(l + 1, j)$ -Weges ist gar kein Weg, d. h., K ist eine gerichtete (i, j) -Kette, die einen Knoten mindestens zweimal enthält. Die Kette K enthält natürlich einen (i, j) -Weg, den wir \bar{K} nennen, und \bar{K} geht aus K dadurch hervor, dass wir die in K vorkommenden gerichteten Kreise entfernen. Der Knoten $l + 1$ ist in einem der Kreise nach der Konstruktion enthalten, also ist \bar{K} ein (i, j) -Weg, der nur Knoten aus $1, \dots, l$ enthält, d. h., $w_{ij}^l \leq c(\bar{K})$. Aus

$$c(K) = w_{i,l+1}^l + w_{l+1,j}^l < w_{ij}^l \leq c(\bar{K})$$

folgt, dass mindestens einer der aus K entfernten gerichteten Kreise eine negative Länge hat. Für jeden Knoten i dieses negativen Kreises muss folglich $w_{ii}^{l+1} < 0$ gelten. Daraus folgt die Behauptung. \square

Wer sich für die Geschichte der Methoden zur Bestimmung kürzester Wege interessiert, dem sei der Artikel [17] von Schrijver empfohlen.

5.5 Dynamische Programmierung

Bei vielen Optimierungsproblemen sind Teillösungen von optimalen Lösungen selbst optimal für das entsprechende Teilproblem. Dieses Prinzip heißt *Optimalitätsprinzip von Bellman* und wird von der dynamischen Programmierung genutzt, indem eine optimale Lösung des ursprünglichen Problems rekursiv aus optimalen Teillösungen konstruiert wird. Die dynamische Programmierung ist ein allgemeines Verfahren um Optimierungsprobleme zu lösen, für die das Optimalitätsprinzip gilt (z.B. Kürzeste Wege, Maximale Flüsse oder das Rucksack-Problem). Teillösungen des Ausgangsproblems werden dabei meist nur einmal bestimmt und anschließend gespeichert, damit sie, auf Kosten von zusätzlichem Speicherplatz, für weitere Berechnungen erneut verwendet

werden können. Das abstrakte Verfahren wird in [2] von Bellman beschrieben. Wir betrachten nun die dynamische Programmierung am Beispiel von Kürzesten Wegen.

Sei $D = (V, A)$ wieder ein gerichteter Graph mit nichtnegativen Bogengewichten c_a für $a \in A$ und $s \in V$ der Startknoten. Gesucht seien kürzeste Wege von s zu allen anderen Knoten $v \in V$. $d(v)$ bezeichne die Länge eines kürzesten (s, v) -Wegs. Da die Bogengewichte nichtnegativ sind, sind Teilwege kürzester Wege wieder kürzeste Wege, weswegen das Optimalitätsprinzip von Bellman gilt (vgl. Lemma 5.1).

Sei $d_k(v)$ die Länge eines kürzesten (s, v) -Wegs, der höchstens k Bögen benutzt. Dann gilt:

$$d_k(v) = \min \left\{ d_{k-1}(v), \min_{(u,v) \in A} \{d_{k-1}(u) + c_{uv}\} \right\}. \quad (5.3)$$

Denn für $s \neq v$ setzt sich ein kürzester (s, v) -Weg, der maximal k Bögen benutzt, aus einem Bogen $(u, v) \in A$ und einem kürzesten (s, u) -Weg, der höchstens $k - 1$ Bögen benutzt, zusammen. Mit 5.3 erhalten wir somit ein Verfahren um die Längen kürzester (s, v) -Wege zu berechnen. Man startet mit der Bestimmung von $d_1(v)$ für alle $v \in V$ und berechnet anschließend d_k rekursiv für $k \in \{2, \dots, |V| - 1\}$ mithilfe der bereits bekannten Werte d_{k-1} .

Beispiel 19. Wir betrachten den Graphen $G = (V, E)$ aus Abbildung 19 und wollen die Längen der kürzesten Wege vom Startknoten $s = 1$ zu allen anderen Knoten bestimmen.

Wir beginnen mit der Berechnung von $d_1(v)$ für alle $v \in V$. Offenbar gilt in einfachen Graphen $d_1(v) = c_{sv}$, falls eine Kante $(s, v) \in E$ existiert, ansonsten $d_1(v) = \infty$ bzw. $d_1(s) = 0$. Anschließend nutzen wir 5.3 um d_k für $k = 2, \dots, 4$ zu bestimmen:

$$\begin{aligned} d_2(2) &= \min \left\{ 7, \min \left\{ 0 + 7, \infty + 1 \right\} \right\} = 7, \\ d_2(3) &= \min \left\{ 2, \min \left\{ 0 + 2, 3 + 5, \infty + 4 \right\} \right\} = 2, \\ d_2(4) &= \min \left\{ 3, \min \left\{ 0 + 3, 2 + 5, \infty + 2 \right\} \right\} = 3, \\ d_2(5) &= \min \left\{ \infty, \min \left\{ 7 + 1, 2 + 4, 3 + 2 \right\} \right\} = 5, \dots \end{aligned}$$

Da jeder kürzeste Weg über höchstens vier Kanten verläuft, gilt $d(v) = d_4(v)$ für alle $v \in V$.

△

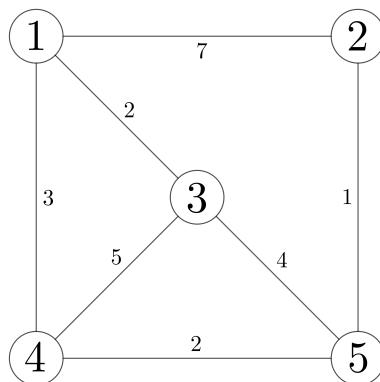


Abbildung 5.4: Ausgangsgraph in Beispiel 19

Knoten v	1	2	3	4	5
$d_1(v)$	0	7	2	3	∞
$d_2(v)$	0	7	2	3	5
$d_3(v)$	0	6	2	3	5
$d_4(v)$	0	6	2	3	5

Kapitel 6

Maximale Flüsse

In diesem Kapitel sei $D = (V, A)$ ein Digraph mit Bogenkapazitäten $c_a \geq 0$ für alle Bögen $a \in A$. Ferner seien $s, t \in V$ mit $s \neq t$ zwei verschiedene ausgezeichnete Knoten. Der Knoten s heißt *Quelle* (engl. source) und t heißt *Senke* (engl. target).

Definition 6.1. Eine Funktion $x : A \rightarrow \mathbb{R}$ heißt *zulässiger (s, t) -Fluss* (oder oft auch einfach *(s, t) -Fluss*), wenn

$$0 \leq x_a \leq c_a \quad \text{für alle } a \in A, \quad (6.1)$$

$$\sum_{a \in \delta^{\text{in}}(v)} x_a = \sum_{a \in \delta^{\text{out}}(v)} x_a \quad \text{für alle } v \in V \setminus \{s, t\} \quad (6.2)$$

gilt. Wenn x ein zulässiger (s, t) -Fluss ist, so ist

$$\text{val}(x) = \sum_{a \in \delta^{\text{out}}(s)} x_a - \sum_{a \in \delta^{\text{in}}(s)} x_a$$

der Wert des (s, t) -Flusses x .

Definition 6.2. Das Problem, einen zulässigen Fluss maximalen Wertes zu finden, heißt *Maximal-Fluss-Problem* (engl. *Maximum Flow Problem*).

Es handelt sich hierbei um das folgende lineare Optimierungsproblem:¹

$$\begin{aligned} \max_{x \in \mathbb{R}^{|A|}} \quad & \sum_{a \in \delta^{\text{out}}(s)} x_a - \sum_{a \in \delta^{\text{in}}(t)} x_a \\ \text{s.t.} \quad & 0 \leq x_a \leq c_a \quad \text{für alle } a \in A, \\ & \sum_{a \in \delta^{\text{in}}(v)} x_a = \sum_{a \in \delta^{\text{out}}(v)} x_a \quad \text{für alle } v \in V \setminus \{s, t\}. \end{aligned}$$

Anwendungen

- Rohrleitungssysteme wie z. B. Gasnetze, Stromnetze, Wassernetze, ...
- Kommunikationsnetze wie z. B. Telefonnetze, ...
- Verkehrsnetzwerke, ...

Es gibt eine Reihe von Artikeln zur Geschichte des Maximal-Fluss-Problems. Ein guter Ausgangspunkt ist der Artikel [18] von Schrijver oder der Zeit-Artikel [8]. Der darin behandelte militärische Originalartikel ist [9] von Harris und Ross.

6.1 Das Max-Flow-Min-Cut-Theorem

Lemma 6.3. Sei $D = (V, A)$ ein Digraph mit Bogenkapazitäten $c_a \geq 0$ für alle $a \in A$ und $s, t \in V$ mit $s \neq t$. Sei \mathcal{P} die Menge aller gerichteten (s, t) -Wege in D und \mathcal{C} die Menge aller gerichteten Kreise. Dann gilt, dass x genau dann ein zulässiger (s, t) -Fluss ist, wenn es Wege $P_1, \dots, P_p \in \mathcal{P}$ und Kreise $C_1, \dots, C_q \in \mathcal{C}$ sowie $\lambda_1, \dots, \lambda_p > 0, \mu_1, \dots, \mu_q > 0$ gibt mit

$$x_{uv} = \sum_{\{i: uv \in P_i\}} \lambda_i + \sum_{\{j: uv \in C_j\}} \mu_j \leq c_{uv} \quad \text{für alle } uv \in A.$$

Ferner gilt $p + q \leq |A|$.

Beweis. Übungsaufgabe. □

Lemma 6.4. Es sei $W \subseteq V$ ein (s, t) -Schnitt, d. h. es gilt $s \in W$ und $t \in V \setminus W$. Dann gilt:

1. Für jeden zulässigen (s, t) -Fluss x gilt

$$\text{val}(x) = x(\delta^{\text{out}}(W)) - x(\delta^{\text{in}}(W)) := \sum_{a \in \delta^{\text{out}}(W)} x_a - \sum_{a \in \delta^{\text{in}}(W)} x_a.$$

¹Überlegen Sie sich, wann dieses LP unbeschränkt ist und welcher Situation im Digraphen D dieses Phänomen entspricht.

2. Für jeden zulässigen (s, t) -Fluss x gilt

$$\text{val}(x) \leq c(\delta^{\text{out}}(W)) := \sum_{a \in \delta^{\text{out}}(W)} c_a.$$

Beweis. 1. Nach Definition gilt

$$\begin{aligned} \text{val}(x) &= x(\delta^{\text{out}}(s)) - x(\delta^{\text{in}}(s)) \\ &= x(\delta^{\text{out}}(s)) - x(\delta^{\text{in}}(s)) + \sum_{v \in W \setminus \{s\}} (x(\delta^{\text{out}}(v)) - x(\delta^{\text{in}}(v))) \\ &= \sum_{v \in W} (x(\delta^{\text{out}}(v)) - x(\delta^{\text{in}}(v))) \\ &= x(\delta^{\text{out}}(W)) - x(\delta^{\text{in}}(W)). \end{aligned}$$

2. Nach 1. gilt

$$\begin{aligned} \text{val}(x) &= x(\delta^{\text{out}}(W)) - x(\delta^{\text{in}}(W)) \\ &\leq x(\delta^{\text{out}}(W)) \\ &\leq c(\delta^{\text{out}}(W)). \end{aligned} \quad \square$$

Korollar 6.5. Der Wert eines minimalen (s, t) -Schnittes ist eine obere Schranke für den Wert eines maximalen Flusses.

Beweis. Folgt direkt aus Lemma 6.4. \square

Dieser Wert kann tatsächlich erreicht werden, wie wir gleich sehen werden. Eine Idee zur Vorgehensweise liefert uns Lemma 6.3:

Gegeben ein zulässiger (s, t) -Fluss x , z. B. $x = 0$. Erhöhe x stückweise auf gerichteten (s, t) -Wegen P mit $x_a < c_a$ für alle $a \in P$.

Diese Idee reicht aber nicht aus, wie folgendes Beispiel zeigt:

Beispiel 20. Wir betrachten das Flussproblem aus Abbildung 6.1. Der Wert des minimalen Schnittes ist 4 und der Wert des Flusses, den wir durch obige Idee erreichen, ist nur 3. \triangle

Es reicht also nicht, nur die Bögen in Richtung eines gerichteten (s, t) -Weges zu betrachten. Dies führt uns zu folgender Definition.

Definition 6.6 (Vorwärts- und Rückwärtsbogen, augmentierende Wege). Sei x ein zulässiger (s, t) -Fluss in dem Digraphen $D = (V, A)$. In einem (ungerichteten) $[s, v]$ -Weg P nennen wir einen Bogen (i, j) , der in P in Richtung von s nach v läuft einen *Vorwärtsbogen*, andernfalls einen *Rückwärtsbogen*. P heißt *augmentierender* $[s, v]$ -Weg (bzgl. x), falls $x_{ij} < c_{ij}$ für jeden Vorwärtsbogen und $x_{ij} > 0$ für jeden Rückwärtsbogen gilt.

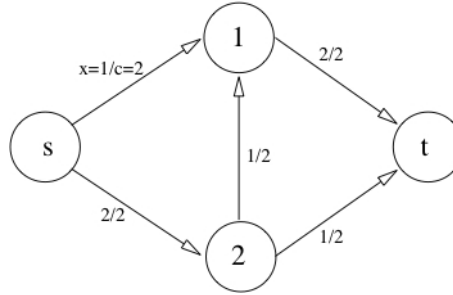


Abbildung 6.1: Ein Fluss mit Wert 3 in einem Graphen mit minimalem Schnitt 4

Satz 6.7 (Optimalitätskriterium; Augmentierende-Wege-Theorem). Ein (s, t) -Fluss x ist genau dann maximal, wenn es keinen augmentierenden $[s, t]$ -Weg bzgl. x gibt.

Beweis. Wir zeigen zuerst, dass wir für einen gegebenen Fluss x und einen augmentierenden $[s, t]$ -Weg P einen Fluss mit höherem Wert konstruieren können. Wir setzen

$$\varepsilon_{ij} = \begin{cases} c_{ij} - x_{ij}, & \text{falls } (i, j) \in P \text{ ist Vorwärtsbogen,} \\ x_{ij}, & \text{falls } (i, j) \in P \text{ ist Rückwärtsbogen.} \end{cases}$$

Ferner sei $\varepsilon := \min\{\varepsilon_{ij}\}$ und es gilt $\varepsilon > 0$. Dann ist

$$\hat{x}_{ij} = \begin{cases} x_{ij} + \varepsilon, & \text{falls } (i, j) \in P \text{ Vorwärtsbogen,} \\ x_{ij} - \varepsilon, & \text{falls } (i, j) \in P \text{ Rückwärtsbogen,} \\ x_{ij}, & \text{falls } (i, j) \in A \setminus P, \end{cases}$$

ein zulässiger (s, t) -Fluss mit

$$\text{val}(\hat{x}) = \text{val}(x) + \varepsilon.$$

Also war x nicht maximal.

Jetzt zeigen wir noch, dass ein Fluss x , für den es keinen augmentierenden $[s, t]$ -Weg gibt, maximal ist. Angenommen, x besitzt keinen augmentierenden $[s, t]$ -Weg. Es sei

$$W = \{v \in V : \exists \text{ augmentierenden } [s, v]\text{-Weg bzgl. } x\} \cup \{s\}.$$

Es ist $s \in W$ und (nach Annahme) $t \notin W$. Ferner gilt nach Definition von W :

$$x_a = \begin{cases} c_a, & \text{für alle } a \in \delta^{\text{out}}(W), \\ 0, & \text{für alle } a \in \delta^{\text{in}}(W). \end{cases}$$

Mit Lemma 6.4, Teil 1, erhalten wir schließlich

$$\begin{aligned}\text{val}(x) &= x(\delta^{\text{out}}(W)) - x(\delta^{\text{in}}(W)) \\ &= c(\delta^{\text{out}}(W)).\end{aligned}$$

Also ist x maximal nach Lemma 6.4, Teil 2. \square

Satz 6.8 (Das Max-Flow-Min-Cut-Theorem). Der maximale Wert eines (s, t) -Flusses ist gleich der minimalen Kapazität eines (s, t) -Schnittes.

Beweis. Sei x maximaler (s, t) -Fluss. Nach Lemma 6.4 genügt es einen (s, t) -Schnitt W zu finden mit

$$\text{val}(x) = c(\delta^{\text{out}}(W)).$$

Offenbar ist $\delta^{\text{out}}(W)$ mit W wie im Beweis von Satz 6.7 ein Schnitt mit dieser Eigenschaft. \square

Satz 6.9 (Ganzzahligkeitssatz). Sei D ein Digraph mit ganzzahligen Kapazitäten $c_a \geq 0$ und $s, t \in V$ mit $s \neq t$. Dann gibt es einen maximalen (s, t) -Fluss, der ganzzahlig ist.

Beweis. Wir führen den Beweis per Induktion über die Anzahl der durchgeführten Augmentierungen. Wir starten mit dem Nullfluss $x = 0$, der ganzzahlig ist.

Haben wir nun einen ganzzahligen Fluss x , der nicht maximal ist, so bestimmen wir einen augmentierenden Weg P und ε wie im Beweis von Satz 6.7. Nach Voraussetzung ist ε ganzzahlig und damit auch der neue Fluss \hat{x} .

Bei jeder Augmentierung erhöhen wir den Wert des Flusses um mindestens eins. Da der maximale Fluss endlich ist, folgt die Behauptung. \square

6.2 Augmentierende-Wege-Algorithmus

Der Satz 6.7 und sein Beweis geben uns bereits die Idee für einen ersten Algorithmus. Für die folgenden Überlegungen ist der sogenannte Residualgraph essentiell:

Definition 6.10 (Residualgraph). Gegeben sei ein Digraph $D = (V, A)$ mit $s, t \in V$ mit $s \neq t$ und x sei ein zulässiger (s, t) -Fluss. Dann nennen wir

$$G(x) = (V, A(x)),$$

mit

$$(i, j) \in A(x) \text{ mit Bogenkapazität } c_{ij} - x_{ij}, \text{ falls } (i, j) \in A \text{ und } x_{ij} < c_{ij}$$

Algorithmus 16 Augmentierender-Wege-Algorithmus (von Ford & Fulkerson)

Eingabe: Ein Digraph $D = (V, A)$ mit Bogenkapazitäten $c_a \geq 0, a \in A$, und zwei Knoten $s, t \in V$ mit $s \neq t$.

Ausgabe: Ein zulässiger (s, t) -Fluss x mit maximalem Wert $\text{val}(x)$ und ein kapazitätsminimaler (s, t) -Schnitt $\delta^{\text{out}}(W)$.

- 1: Bestimme einen zulässigen Fluss, z. B. $x = 0$.
- 2: **while** Es existiert ein augmentierender Weg von s nach t **do**
- 3: Identifiziere einen augmentierenden $[s, t]$ -Weg P .
- 4: Setze

$$\varepsilon = \min_{(i,j) \in P} \begin{cases} c_{ij} - x_{ij}, & \text{falls } (i, j) \in P \text{ Vorwärtsbogen,} \\ x_{ij}, & \text{falls } (i, j) \in P \text{ Rückwärtsbogen.} \end{cases}$$

- 5: Erhöhe x entlang P um ε , d. h., setze

$$x_{ij} \leftarrow \begin{cases} x_{ij} + \varepsilon, & \text{falls } (i, j) \in P \text{ Vorwärtsbogen,} \\ x_{ij} - \varepsilon, & \text{falls } (i, j) \in P \text{ Rückwärtsbogen,} \\ x_{ij}, & \text{sonst.} \end{cases}$$

- 6: Bestimme W wie im Beweis des Optimalitätskriteriums.
 - 7: **return** $x, \text{val}(x)$ und W .
-

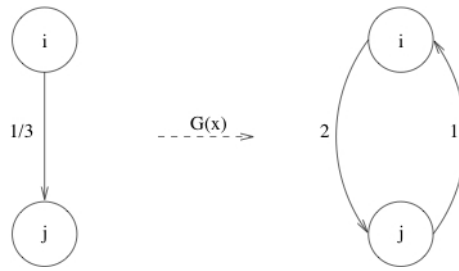


Abbildung 6.2: Beispiel für einen Residualgraph

und

$(j, i) \in A(x)$ mit Bogenkapazität x_{ij} , falls $(i, j) \in A$ und $x_{ij} > 0$

den *Residualgraph* von D bzgl. x .

Ein Beispiel für einen Residualgraphen sehen Sie in Abbildung 6.2.

Bemerkung 6.11. In Satz 2.7 haben wir für einen beliebigen Graphen $G = (V, E)$ mit $n = |V|$ und $m = |E|$ gezeigt, dass Tiefensuche (bzw. Breitensuche) eine Laufzeit von $\mathcal{O}(n + m)$ hat. Ist G zusammenhängend gilt $n \leq m + 1$ und somit $n + m \leq 2m + 1 \in \mathcal{O}(m)$.

Satz 6.12. Algorithmus 16 arbeitet korrekt. Die Laufzeit beträgt $\mathcal{O}(m\nu)$, wobei ν der Wert des maximalen Flusses ist, falls c ganzzahlig ist.

Beweis. Satz 6.7 aus Abschnitt 6.1 impliziert, dass Algorithmus 16 mit einem maximalen Fluss in Schritt 7 endet. Da c ganzzahlig ist, sind nach Satz 6.9 der Wert ε in Schritt 4 und ν ganzzahlig, d. h., die Schritte 2–5 werden maximal ν -mal durchlaufen. Es bleibt zu zeigen, dass die Schritte 3, 4 und 5 jeweils maximal $\mathcal{O}(m)$ Operationen benötigen.

Für die Schritt 4 und 5 ist das offensichtlich. Zur Bestimmung eines augmentierenden Weges P sei

$$G(x) = (V, A(x)),$$

der aktuelle Residualgraph. Der Aufbau von $G(x)$ benötigt $\mathcal{O}(m)$ Operationen. Offensichtlich entspricht jeder (s, t) -Weg in $G(x)$ einem augmentierenden $[s, t]$ -Weg bzgl. x in D und umgekehrt. Ein gerichteter (s, t) -Weg in $G(x)$ kann in $\mathcal{O}(m)$ bestimmt werden; z. B. mit Breiten- oder Tiefensuche. Damit ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(\nu m)$. \square

Man beachte, dass die Laufzeit von Algorithmus 16 nicht polynomiell ist. Polynomielle Laufzeit ist möglich unter Verwendung sogenannter kürzester augmentierender Wege. Wir nennen einen augmentierenden Weg P einen

kürzesten augmentierenden Weg, wenn er die kleinstmögliche Anzahl an Bögen enthält.

Satz 6.13. Wird in Schritt 3 von Algorithmus 16 immer ein kürzester augmentierender Weg gewählt, so sind höchstens nm Augmentierungen notwendig.

Die im letzten Satz beschriebene Variante des allgemeinen Augmentierende-Wege-Algorithmus 16 ist als Edmonds–Karp-Algorithmus bekannt. Sobald wir Satz 6.13 bewiesen haben, erhalten wir direkt das folgende Korollar.

Korollar 6.14. Algorithmus 16 mit Breitensuche in Schritt 3 bestimmt einen maximalen Fluss in $\mathcal{O}(nm^2)$.

Beweis. Ein kürzester augmentierender Weg (in Schritt 3) kann per Breitensuche in $G(x)$ in $\mathcal{O}(m)$ bestimmt werden. Zusammen mit Satz 6.13 folgt die Behauptung. \square

Es bleibt noch der Beweis von Satz 6.13. Hierfür betrachten wir eine Augmentierung von Fluss x zu Fluss x' entlang eines kürzesten Weges P mit den Knoten $s = v_0, v_1, \dots, v_k = t$.

Sei $d_x(v, w)$ die kürzeste Länge (hier gleich der Anzahl der Bögen) eines gerichteten (v, w) -Weges in $G(x)$. Wie üblich, setzen wir $d_x(v, w) = +\infty$, falls kein Weg von v nach w in $G(x)$ existiert. Offensichtlich gilt $d_x(s, v_i) = i$ und $d_x(v_i, t) = k - i$ für alle i . Falls $wv \in A(x')$ und $wv \notin A(x)$, dann ist $w = v_i$ und $v = v_{i-1}$ für ein passendes i .

Lemma 6.15. Für alle $v \in V$ gilt

$$d_{x'}(s, v) \geq d_x(s, v) \quad \text{und} \quad d_{x'}(v, t) \geq d_x(v, t).$$

Beweis. Angenommen, es existiert ein $v \in V$ mit $d_{x'}(s, v) < d_x(s, v)$ und wähle v derart, dass $d_{x'}(s, v)$ so klein wie möglich ist. Offensichtlich gilt $d_{x'}(s, v) > 0$. Sei P' ein kürzester (s, v) -Weg in $G(x')$ und w der vorletzte Knoten. Dann gilt

$$d_x(s, v) > d_{x'}(s, v) = d_{x'}(s, w) + 1 \geq d_x(s, w) + 1. \quad (6.3)$$

Daraus folgt $wv \notin A(x)$, denn sonst wäre $d_x(s, v) \leq d_x(s, w) + 1$. Also ist mit unseren Vorüberlegungen $w = v_i$ und $v = v_{i-1}$ für ein $i \in \{1, \dots, k\}$, da $(w, v) \in A(x')$ nach Konstruktion von P' . Aus Ungleichung (6.3) erhalten wir jedoch

$$i - 1 = d_x(s, v) > d_x(s, w) + 1 = i + 1,$$

was offensichtlich ein Widerspruch ist. Die zweite Ungleichung $d_{x'}(v, t) \geq d_x(v, t)$ zeigt man analog. \square

Aus Lemma 6.15 folgt, dass der Kürzeste-Augmentierende-Wege-Algorithmus in Stufen eingeteilt werden kann, in denen $d_x(s, t)$ konstant gleich k bleibt. Von diesen Stufen gibt es höchstens $n - 1$ Stück. Die offene Frage ist also, wie lange eine dieser Stufen maximal dauern kann. Sei dazu

$$\tilde{A}(x) := \{a \in A(x) : a \in P \text{ und } P \text{ ist ein kürzester augmentierender Weg}\}.$$

Lemma 6.16. Ist $d_{x'}(s, t) = d_x(s, t)$, dann gilt $\tilde{A}(x') \subset \tilde{A}(x)$.

Beweis. Es sei $k = d_{x'}(s, t) = d_x(s, t)$ und betrachte $a \in \tilde{A}(x')$. Angenommen $a \notin \tilde{A}(x)$. Daraus folgt, dass $x_a \neq x'_a$ gelten muss, was wiederum $a \in P$ für einen passenden kürzesten Weg P in $A(x)$ impliziert und ein Widerspruch zu $a \notin \tilde{A}(x)$ ist. Also gilt $\tilde{A}(x') \subseteq \tilde{A}(x)$. Nun muss es einen Bogen $a \in P$ geben mit

$$a \text{ ist Vorwärtsbogen und } x'_a = c_a$$

oder

$$a \text{ ist Rückwärtsbogen und } x'_a = 0.$$

Sei $a = (v, w) \in \tilde{A}(x)$. Dann gilt

$$d_x(s, v) = i - 1 \quad \text{und} \quad d_x(w, t) = k - i \quad \text{für ein } i \in \{1, \dots, k\},$$

d. h., die Augmentierung bzgl. x muss in umgekehrter Reihenfolge erfolgen, wie die Augmentierung bzgl. x' . Die Länge dieses augmentierenden Weges bzgl. x' ist

$$d_{x'}(s, w) + 1 + d_{x'}(v, t) \geq d_x(s, w) + 1 + d_x(v, t) = i + 1 + (k - i + 1) = k + 2.$$

Damit ist $a \notin \tilde{A}(x')$ und die Behauptung ist gezeigt. \square

Damit können wir Satz 6.13 beweisen:

Beweis von Satz 6.13. Nach Lemma 6.16 gibt es höchstens m Augmentierungen pro Stufe. Da es höchstens $n - 1$ Stufen gibt, folgt Satz 6.13. \square

6.3 Ein allgemeiner Push-Relabel-Algorithmus

In diesem Abschnitt lernen wir eine weitere, zunächst vollkommen anders erscheinende, Idee kennen, um maximale Flüsse zu bestimmen. Die resultierenden Algorithmen heißen *Push-Relabel* oder *Preflow-Push* Algorithmen.

Zunächst schauen wir uns ein Beispiel an, für das Augmentierende-Wege-Algorithmen ineffizient sind.

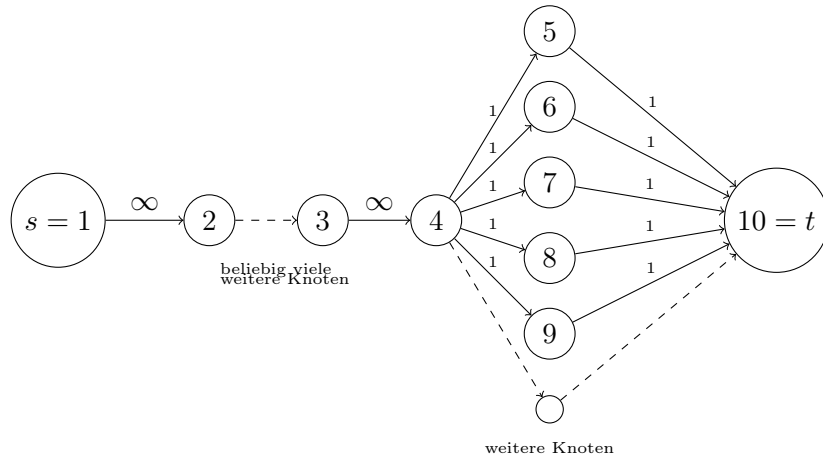


Abbildung 6.3: “Lollipop”-Graph aus Beispiel 21

Beispiel 21. Wir betrachten den “Lollipop”-Graph aus Abbildung 6.3. Der Nachteil von Augmentierende-Wege-Algorithmen bei Graphen dieser Struktur ist, dass der Fluss immer entlang von $[s, t]$ -Wegen geschickt wird. Dies ist aber teuer, da es $\mathcal{O}(n)$ bzw. $\mathcal{O}(m)$ Augmentierungen benötigt, bis die Bögen $(4, i)$ und $(i, 10)$ für $i \in \{5, \dots, 9\}$ saturiert sind. \triangle

Eine Idee, um dieses Verhalten zu umgehen, lautet wie folgt: Schiebe (engl. push) soviel wie möglich durch das Netz unter Verletzung der Flusserhaltungsbedingung (6.2). Dadurch entstehen Knoten mit sogenanntem Überschuss. Dieser Überschuss muss abgebaut werden durch Schieben von Fluss in Richtung der Senke t .

Zur Vereinfachung der Darstellung treffen wir in diesem Kapitel o. B. d. A. folgende Annahme:

$$c_{wv} = x_{wv} = 0 \quad \text{falls} \quad vw \in A \text{ und } wv \notin A.$$

Im Folgenden sei $G(x) = (V, A(x))$ wieder der Residualgraph bzgl. x mit $x \in \mathbb{R}^{|A|}$ und $0 \leq x \leq c$. Falls $(i, j) \in A(x)$ ist, kann maximal $\tilde{c}_{ij} = c_{ij} - x_{ij} + x_{ji}$ geschoben werden. Der Wert \tilde{c}_{ij} heißt *Residualkapazität* des Bogens (i, j) .

Definition 6.17 (Pseudofluss, Überschuss, aktive Knoten). Ein *Pseudofluss* (engl. preflow) auf einem Digraphen $D = (V, A)$ ist eine Abbildung $x : A \rightarrow \mathbb{R}$ mit folgenden Eigenschaften:

1. $0 \leq x_a \leq c_a$ für alle $a \in A$.
2. $e_x(v) = x(\delta^{\text{in}}(v)) - x(\delta^{\text{out}}(v)) \geq 0$ für alle $v \in V \setminus \{s, t\}$. Die Abbildung $e_x : V \rightarrow \mathbb{R}_{\geq 0}$ heißt *Überschuss* (engl. excess) bzgl. x .

Ein Knoten v heißt *aktiv* (bzgl. x), falls $e_x(v) > 0$ gilt.

Bemerkung 6.18. Ein Pseudofluss ist genau dann ein zulässiger Fluss, wenn er keine aktiven Knoten hat.

Definition 6.19 (Gültige Markierungen). Eine Funktion $d : V \rightarrow \mathbb{Z}_{\geq 0} \cup \{\infty\}$ heißt *gültige Markierung* bzgl. des Pseudoflusses x , falls

1. $d(t) = 0$ und
2. $d(v) \leq d(w) + 1$ für alle $vw \in A(x)$

gilt.

Es bezeichne $\text{dist}_x(v, w)$ die Länge eines kürzesten (v, w) -Weges in $G(x)$ bzgl. des Flusses x und der Bogengewichte $c_a = 1$ für alle $a \in A$. Man beachte, dass $d(v) = \text{dist}_x(v, t)$ die zweite Bedingung aus Definition 6.19 erfüllt; vgl. Satz 5.6 in Kapitel 5.1. Außerdem haben wir $d(t) = \text{dist}_x(t, t) = 0$.

Es lässt sich wie folgt ein Pseudofluss x und eine dafür gültige Markierung d konstruieren vermöge

$$\begin{aligned} x_{uv} &= \begin{cases} c_{uv}, & \text{falls } uv \in \delta^{\text{out}}(s), \\ 0, & \text{sonst,} \end{cases} \\ d(v) &= \begin{cases} n, & \text{falls } v = s, \\ k, & \text{wobei } k \text{ kürzeste } (v, t)\text{-Weglänge in } G(x) \text{ ist, sonst.} \end{cases} \end{aligned} \quad (6.4)$$

Der folgende Satz zeigt, dass der Wert eines Pseudoflusses dem Wert eines Schnittes entspricht.

Lemma 6.20. Sei x ein Pseudofluss und d eine gültige Markierung bzgl. x wie in (6.4). Dann existiert ein $W \subseteq V, s \in W, t \notin W$ mit

$$x_{uv} = \begin{cases} c_{uv} & , \text{ für alle } uv \in \delta^{\text{out}}(W) , \\ 0 & , \text{ für alle } uv \in \delta^{\text{in}}(W) . \end{cases}$$

Beweis. Angenommen, es existiert kein Schnitt W mit dieser Eigenschaft. Da es n Knoten gibt, existiert eine Zahl k (siehe *Pigeon Hole Principle*) mit $0 < k < n$, so dass $d(v) \neq k$ für alle $v \in V$. Setze

$$W = \{v \in V \mid d(v) > k\} .$$

Offensichtlich gilt $s \in W, t \notin W$. Definition 6.19 impliziert, dass kein Bogen von $A(x)$ die Knotenmenge W verlässt, d.h.

$$\delta^+(W) \cap A(x) = \emptyset .$$

□

Bemerkung 6.21. Damit sind Push–Relabel–Algorithmen in gewissem Sinne dual zu Augmentierenden–Wege–Algorithmen. Sie behalten einen Pseudofluss x und eine gültige Markierung (und damit einen saturierten Schnitt) und terminieren, wenn x zulässig ist, während Augmentierende–Wege–Algorithmen einen zulässigen Fluss behalten und terminieren, wenn ein Schnitt saturiert wird.

Wir zeigen jetzt, dass eine gültige Markierung $d(v)$ eine untere Schranke an die Länge eines kürzesten gerichteten Weges vom Knoten v zum Knoten t in $G(x)$ ist.

Lemma 6.22. Für einen Pseudofluss x und eine gültige Markierung d bzgl. x gilt

$$\text{dist}_x(v, w) \geq d(v) - d(w) \quad \text{für alle } v, w \in V.$$

Beweis. Ist $\text{dist}_x(v, w) = +\infty$, d. h., es gibt keinen Weg von v nach w , so ist die Ungleichung offensichtlich erfüllt.

Andernfalls sei $P = (v = i_0, i_1, \dots, i_k = w)$ ein kürzester (v, w) -Weg der Länge k in $G(x)$. Aus dem zweiten Teil von Definition 6.19 folgt dann

$$d(v) = d(i_0) \leq d(i_1) + 1 \leq d(i_2) + 2 \leq \dots \leq d(i_k) + k = d(w) + k. \quad \square$$

Definition 6.23. Es sei $d : V \rightarrow \mathbb{Z}_{\geq 0} \cup \{\infty\}$ eine gültige Markierung. Wir nennen einen Bogen (v, w) *zulässigen Bogen*², falls $d(v) = d(w) + 1$ gilt. Ein Pfad P von s nach t wird *zulässiger Pfad*³ genannt, falls alle $(v, w) \in P$ zulässige Bögen sind.

Die Idee des Push-Relabel-Algorithmus ist jetzt die folgende: Wenn wir einen Pseudofluss haben, wissen wir, dass er zulässig ist, wenn der Überschuss an allen Knoten 0 ist. Angenommen, wir haben noch Knoten mit Überschuss. Dann wählen wir einen solchen und versuchen seinen Überschuss abzubauen, indem wir mehr Fluss zu seinen Nachbarn schicken. Die Frage ist: “Zu welchen Nachbarn?”. Da wir grundsätzlich Fluss von der Quelle zur Senke schicken wollen, wählen wir diejenigen Nachbarn, die “näher” an der Senke sind, wobei wir gültige Markierungen nutzen, um eben diese Nähe zu messen. Wir erreichen es, Fluss näher zur Senke zu schicken (Push) genau dann, wenn wir Fluss über zulässige Bögen schicken. Dies überlegen wir uns noch etwas formaler: Aus Lemma 6.22 folgt, dass $d(v)$ eine untere Schranke für $\text{dist}_x(v, t)$ ist. Daher versuchen wir, Fluss entlang derjenigen Bögen (v, w) zu schieben, für die $d(w) < d(v)$ gilt. Der zweite Teil von Definition 6.19 impliziert dann aber, dass $d(v) = d(w) + 1$ gelten muss, d. h. wir schieben

²Engl. *admissible arcs*

³Engl. *admissible path*

den Fluss nur entlang (v, w) , falls $d(v) = d(w) + 1$ gilt – also nur entlang zulässiger Bögen. Falls wir einen aktiven Knoten gefunden haben, für den es keine zulässigen Bögen gibt, erhöhen wir seine Markierung (Relabel) derart, dass es mindestens einen zulässigen Bogen gibt. Formal ist das Verfahren in Algorithmus 17 gegeben.

Algorithmus 17 Push-Relabel-Algorithmus (Goldberg & Tarjan, 1985)

Eingabe: Ein Digraph $D = (V, A)$ mit Bogenkapazitäten $c_a \geq 0, a \in A$, und zwei Knoten $s, t \in V$ mit $s \neq t$.

Ausgabe: Ein maximaler (s, t) -Fluss x .

- 1: Setze x und d gemäß Gleichung (6.4).
 - 2: **while** es gibt einen aktiven Knoten **do**
 - 3: Wähle einen aktiven Knoten v .
 - 4: **if** es gibt einen zulässigen Bogen (v, w) **then**
 - 5: Push: Schiebe $\delta = \min\{e_x(v), c_{vw} - x_{vw} + x_{wv}\}$ von v nach w .
 - 6: **else**
 - 7: Relabel: Setze $d(v) \leftarrow 1 + \min\{d(w) : (v, w) \in A(x)\}$.
 - 8: **return** x .
-

Ein Schub von δ Flusseinheiten von Knoten v zu Knoten w in Schritt 5 von Algorithmus 17 verringert sowohl den Überschuss $e(v)$ als auch die Residualkapazität \tilde{c}_{vw} um den Wert δ . Außerdem wird der Überschuss $e(w)$ und die Residualkapazität \tilde{c}_{wv} um δ erhöht.

Definition 6.24 ((Nicht-)Saturierte Schübe). Ein Schub entlang (v, w) heißt *saturiert*, falls $\tilde{c}_{vw} \leq e_x(v)$ gilt, d. h. es werden $\delta = \tilde{c}_{vw} = c_{vw} - x_{vw} + x_{wv}$ Einheiten geschoben, und der Bogen (v, w) wird aus $G(x)$ entfernt. Andernfalls ($\tilde{c}_{vw} > e_x(v)$) heißt der Schub *nicht-saturiert*.

Beispiel 22 (Push-Relabel-Algorithmus). Ein Beispiel für Algorithmus 17 ist in Abbildung 6.4 gegeben. \triangle

Wir zeigen nun die Korrektheit des Verfahrens.

Lemma 6.25. Nach der Initialisierung der Markierungen in Schritt 1 von Algorithmus 17 sind die Markierungen immer gültig.

Beweis. Die Initialisierung führt zu einem Residualgraph in dem d eine gültige Markierung ist.

Angenommen wir führen die Push-Operation für den Bogen $(v, w) \in G(x)$ durch. Dann gilt $d(v) = d(w) + 1$. Da sich die Markierungen von v und w nicht ändern, bleibt diese Markierung gültig. Falls der Bogen (w, v) bereits

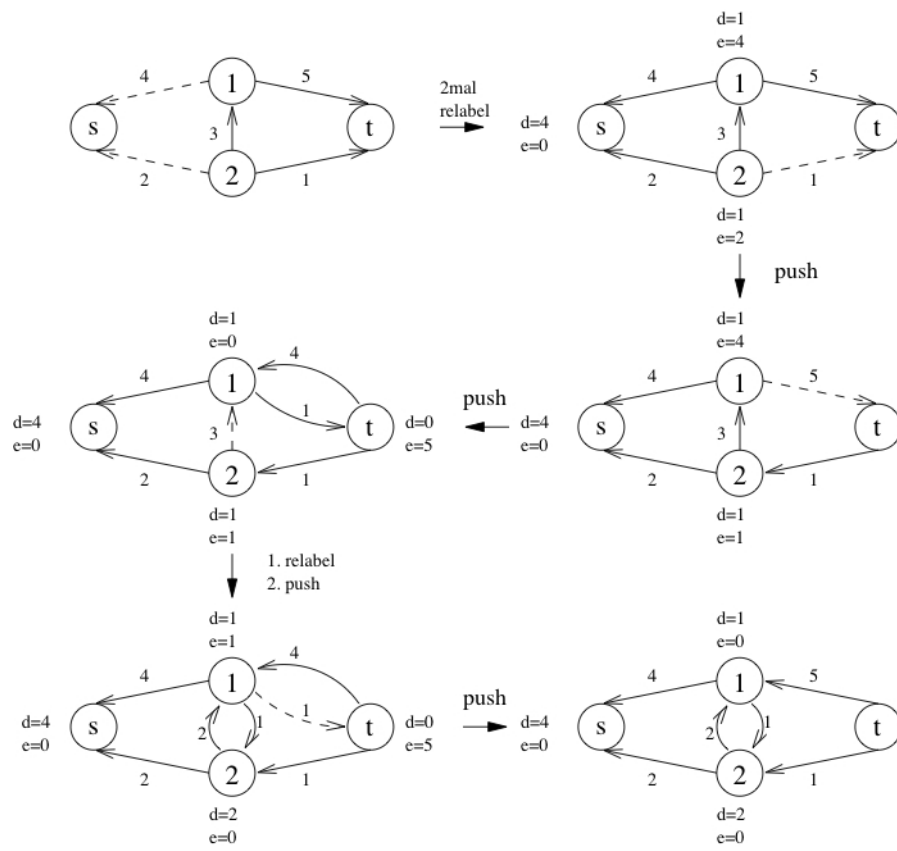


Abbildung 6.4: Graphische Darstellung des Push-Relabel-Algorithmus 17

ebenfalls in $G(x)$ ist oder in $G(x')$ aufgenommen wird, ist die Markierung auch gültig, da $d(w) = d(v) - 1 \leq d(v) + 1$ gilt.

Bleibt der Fall zu diskutieren, dass v aktiv ist, also Überschuss hat, es aber keinen zulässigen Bogen gibt, der v verlässt. In diesem Fall setzen wir in Schritt 7

$$d(v) = 1 + \min\{d(w) : (v, w) \in A(x)\},$$

was die Gültigkeit ebenfalls erhält. \square

Satz 6.26. Wenn der Push-Relabel-Algorithmus 17 terminiert, ist der zurückgegebene Pseudofluss x ein maximaler zulässiger (s, t) -Fluss. In anderen Worten: Algorithmus 17 arbeitet korrekt, falls er terminiert.

Beweis. Der Algorithmus terminiert, wenn es keinen aktiven Knoten, d. h. keinen Knoten mit Überschuss, mehr gibt. Also ist x ein zulässiger (s, t) -Fluss. Da $d(s) = n$ initialisiert wird, die Markierungen im Lauf des Algorithmus nicht kleiner werden, sie nach Lemma 6.25 immer gültig sind und daher nach Lemma 6.22 eine untere Schranke an den kürzesten Weg sind, kann es keinen gerichteten kürzesten Weg von s nach t in $G(x)$ geben. Damit ist x nach Satz 6.7 maximal. \square

Es verbleibt, die Laufzeit zu analysieren.

Lemma 6.27. Ist x ein Pseudofluss und v aktiv, so gibt es einen gerichteten (v, s) -Weg in $G(x)$.

Beweis. Da x ein Pseudofluss ist, gilt $e_x(s) \leq 0$ und $e_x(v) \geq 0$ für alle $v \in V \setminus \{s\}$. Nach der Flusszerlegung in Wege und Kreise (vgl. Lemma 6.3) können wir den Pseudofluss x bzgl. des Originalgraphen D zerlegen in

1. nicht-negative Flüsse entlang von (s, t) -Wegen,
2. Flüsse auf Wege von s zu aktiven Knoten und
3. Flüsse in Kreisen.

Es sei jetzt v ein aktiver Knoten bzgl. des Pseudoflusses x . Also muss es einen gerichteten Weg P von s nach v in D geben, da (s, t) - und Kreisflüsse nicht zum Überschuss in v beitragen. Der Residualgraph enthält dann aber nach Konstruktion einen gerichteten Weg Q , der P rückwärts durchlaufen ist. \square

Das Lemma 6.27 impliziert insbesondere, dass das Minimum in Schritt 7 nicht über die leere Menge gebildet wird.

Lemma 6.28. Für alle $v \in V$ gilt $d(v) \leq 2n - 1$. Damit wird die Markierung jedes Knotens höchstens $2n - 1$ mal geändert und die Gesamtanzahl der Relabel-Operationen ist höchstens $\mathcal{O}(n^2)$.

Beweis. Aus Lemma 6.27 folgt $\text{dist}_x(v, s) \leq n - 1$ und Lemma 6.22 impliziert $\text{dist}_x(v, s) \geq d(v) - n$. Daraus folgt $d(v) \leq 2n - 1$.

Die zweite Aussage folgt aus der ersten, da bei jedem Relabel $d(v)$ um mindestens 1 steigt; vgl. Schritt 7 in Algorithmus 17.

Über alle Knoten $v \in V$ ist die Anzahl der Relabel-Operationen also höchstens $\mathcal{O}(n^2)$. \square

Zuletzt analysieren wir die Anzahl der Push-Operationen. Hierfür unterscheiden wir zwischen saturierten und nicht-saturierten Schüben.

Lemma 6.29. Die Anzahl saturierter Schübe ist höchstens $2nm \in \mathcal{O}(nm)$.

Beweis. Betrachte ein festes Paar (v, w) , so dass $vw \in A$ oder $wv \in A$. Zwischen zwei saturierten Schüben entlang vw muss es einen Schub entlang wv geben, denn andernfalls wäre $vw \notin A(x)$. Da

$$\begin{aligned} d(v) &= d(w) + 1 && \text{für einen Schub entlang } vw, \\ d(w) &= d(v) + 1 && \text{für einen Schub entlang } wv \end{aligned}$$

gilt und $d(v)$ nicht abnimmt, muss es eine Relabel-Operation für w geben vor dem Schub entlang wv . Das heißt, zwischen zwei saturierten Schüben entlang vw muss das Label von $d(w)$ um mindestens 2 erhöht worden sein. Dies kann nach Lemma 6.28 höchstens

$$\left\lceil \frac{2n - 1}{2} \right\rceil = n$$

mal vorkommen. Daraus folgt, dass die Anzahl saturierter Schübe entlang vw höchstens n ist. Also gibt es für einen Bogen $(v, w) \in A$ höchstens $2n$ saturierte Schübe – jeweils höchstens n entlang vw und wv in $G(x)$ – und damit insgesamt $2nm$. \square

Lemma 6.30. Die Anzahl nicht-saturierter Schübe ist $\mathcal{O}(n^2m)$.

Beweis. Sei I die Menge der aktiven Knoten und $\Phi = \sum_{v \in I} d(v)$. Da $|I| < n$ und nach Lemma 6.28 $d(v) < 2n$ gilt, ist zu Beginn des Algorithmus $\Phi < 2n^2$. Am Ende des Algorithmus ist $\Phi = 0$. Außerdem wissen wir, dass Φ nie negativ wird. Wir analysieren nun die Operationen von Algorithmus 17, die zu einer Erhöhung oder Verringerung von Φ führen.

1. Jede Relabel-Operation erhöht Φ um mindestens 1. Da $d(v) < 2n$ für alle $v \in V$ gilt ist das Wachstum von Φ durch Erhöhungen von Markierungen nach oben beschränkt durch $\mathcal{O}(n^2)$.

2. Ein saturierter Schub entlang (v, w) erhöht Φ um höchstens $2n$, denn nach dem Schub könnte $w \in I$ und $v \in I$ gelten, d. h. v bleibt in I und w kommt hinzu. Über alle saturierten Schübe kann Φ nach Lemma 6.29 also um höchstens $\mathcal{O}(n^2m)$ wachsen.
3. Jeder nicht-saturierte Schub verringert Φ um $d(v)$, aber erhöht Φ gleichzeitig um $d(w) = d(v) - 1$, falls Knoten w aktiv wird. Dies entspricht insgesamt einer Verringerung von 1. Falls der Knoten w bereits aktiv war, verringern wir Φ um $d(v)$. Insgesamt verringern wir Φ also mindestens um den Wert 1 pro nicht-saturiertem Schub.

Das heißt, die maximale Erhöhung von Φ ist

$$\mathcal{O}(n^2) + \mathcal{O}(n^2m) = \mathcal{O}(n^2m).$$

Jeder nicht-saturierte Schub verringert Φ mindestens um 1. Da die Summe der Verringerungen höchstens der Summe der Erhöhungen sein kann, gibt es höchstens $\mathcal{O}(n^2m)$ nicht-saturierte Schübe. \square

Insgesamt haben wir also den folgenden Satz bewiesen.

Satz 6.31. Algorithmus 17 führt maximal $\mathcal{O}(n^2)$ Relabel-Operationen und maximal $\mathcal{O}(n^2m)$ Push-Operationen durch.

Um nun tatsächlich eine Gesamtlaufzeit von $\mathcal{O}(n^2m)$ zu erhalten, müssen wir uns Gedanken über geeignete Datenstrukturen machen, um insbesondere Schritt 2 (Finden eines aktiven Knotens) und Schritt 4 (Finden eines zulässigen Bogens) schnell ausführen zu können. Darauf möchten wir an dieser Stelle aber nicht näher eingehen und den interessierten Leser auf Ahuja et al. [1] verweisen. Damit erhalten wir den folgenden Satz.

Satz 6.32. Der Push-Relabel-Algorithmus 17 kann in $\mathcal{O}(n^2m)$ implementiert werden.

Kapitel 7

Minimalkosten-Fluss-Probleme

Es sei $D = (V, A)$ ein Digraph mit Bogenkapazitäten $c_a \geq 0$ für alle $a \in A$ und Kosten $w_a \in \mathbb{R}$ für alle $a \in A$. Ferner sei $b : V \rightarrow \mathbb{R}$ eine Bedarfsfunktion mit

$$\sum_{v \in V} b(v) = 0.$$

Das Problem

$$\begin{aligned} \min_x \quad & \sum_{a \in A} w_a x_a \\ \text{s.t.} \quad & x(\delta^{\text{out}}(v)) - x(\delta^{\text{in}}(v)) = b(v) \quad \text{für alle } v \in V, \quad (7.1a) \\ & 0 \leq x_a \leq c_a \quad \text{für alle } a \in A \quad (7.1b) \end{aligned}$$

heißt *Minimalkosten-Fluss-Problem*¹. Ein Vektor $x \in \mathbb{R}^{|A|}$ heißt *zulässiger Fluss* bzgl. b und c , falls er die Bedingungen (7.1a) und (7.1b) erfüllt.

Man kann sich auf Vektoren $b \in \mathbb{R}^{|V|}$ beschränken mit $b(v) = 0$ für alle $v \in V \setminus \{s, t\}$ und $b(s) > 0$ und $b(t) = -b(s) < 0$; vgl. Abbildung 7.1. Betrachtet man nur solche Vektoren, ist obiges Problem äquivalent zum Finden eines kostenminimalen (s, t) -Flusses mit Wert $b(s)$. Dieses Problem nennt man das *Minimalkosten- (s, t) -Fluss-Problem*.

Wir nehmen noch eine weitere Umformulierung vor, so dass das Problem äquivalent ist zu einer sogenannten kostenminimalen Zirkulation.

Definition 7.1 (*b-transshipment, Zirkulation*). Einen Fluss x , der die Bedingungen (7.1a) und (7.1b) erfüllt, nennt man *b-transshipment*. Ist $b = 0$ der Nullvektor, so nennt man das *b-transshipment* eine *Zirkulation*.

¹Engl.: *minimum cost flow problem*; oder kurz *Min-cost-flow problem*

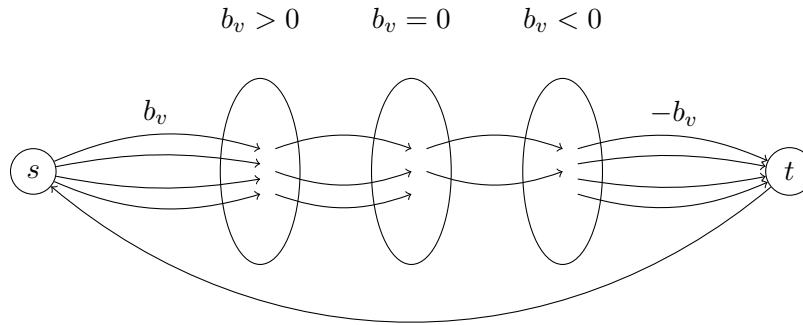


Abbildung 7.1: Reformulierung von Minimalkosten-Fluss-Problemen

Definition 7.2 (Minimalkosten-Zirkulations-Problem). Das Problem

$$\begin{aligned} \min_x \quad & \sum_{a \in A} w_a x_a \\ \text{s.t.} \quad & x(\delta^{\text{out}}(v)) - x(\delta^{\text{in}}(v)) = 0 \quad \text{für alle } v \in V, \\ & d_a \leq x_a \leq c_a \quad \text{für alle } a \in A \end{aligned} \quad \begin{aligned} (7.2a) \\ (7.2b) \end{aligned}$$

heißt *Minimalkosten-Zirkulations-Problem*².

Man kann sich leicht überlegen, dass man das Minimalkosten-Fluss-Problem auf das Minimalkosten-Zirkulations-Problem reduzieren kann. Dazu reduzieren wir das Minimalkosten-Fluss-Problem zunächst auf ein Minimalkosten- (s, t) -Fluss-Problem wie oben beschrieben. Danach führen wir einen weiteren Bogen $a' = (t, s)$ mit $d_{a'} = c_{a'} = b(s) = -b(t)$ ein und setzen $w_{a'} = 0$; vgl. nochmal Abbildung 7.1. In diesem Fall entspricht eine kostenminimale Zirkulation genau einem kostenminimalen Fluss im Originalgraphen.

Man kann sich leicht überlegen³, dass sowohl das Kürzeste-Wege-Problem als auch das Maximal-Fluss-Problem Spezialfälle des Minimalkosten-Fluss-Problems sind.

Wir treffen für dieses Kapitel folgende Annahme:

$$w_a \geq 0 \quad \text{für alle } a \in A.$$

Diese Annahme stellt keine Einschränkung der Allgemeinheit dar, da jedes Minimalkosten-Fluss-Problem mit negativen Kosten in ein äquivalentes Problem mit nicht-negativen Gewichten transformiert werden kann; vgl. Abbildung 7.2.

²Engl.: *minimum cost circulation problem*

³Das machen wir in der Übung.

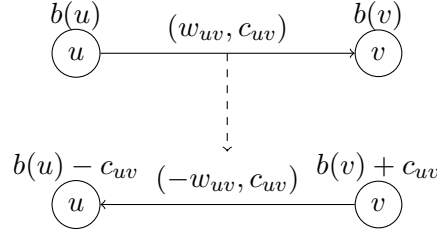


Abbildung 7.2: Reformulierung von Minimalkosten-Fluss-Problemen mit negativen Kosten

7.1 Die Negative-Kreise-Optimalitätsbedingung

Sei $G(x) = (V, A(x))$ der Residualgraph (siehe Kapitel 6.2) mit

$$\begin{aligned} A_1(x) &:= \{ij \in A(x) : ij \in A \text{ und } x_{ij} < c_{ij}\}, \\ A_2(x) &:= \{ij \in A(x) : ji \in A \text{ und } x_{ji} > 0\} \end{aligned}$$

und $A(x) = A_1(x) \cup A_2(x)$. Ferner sei

$$r_{ij} = \begin{cases} c_{ij} - x_{ij}, & \text{falls } ij \in A_1(x), \\ x_{ji}, & \text{falls } ij \in A_2(x), \end{cases}$$

die Residualkapazität.

Im Fall von allgemeinen unteren Flussschranken $d \in \mathbb{R}^{|A|}$ lautet die analoge Definition

$$\begin{aligned} A_1(x) &:= \{ij \in A(x) : ij \in A \text{ und } x_{ij} < c_{ij}\}, \\ A_2(x) &:= \{ij \in A(x) : ji \in A \text{ und } x_{ji} > d_{ji}\}. \end{aligned}$$

Es sei C ein gerichteter Kreis in $G(x)$. Bekanntermaßen definiert jeder gerichtete Kreis in $G(x)$ einen ungerichteten Kreis in D . Wir definieren

$$\chi_{ij}^C := \begin{cases} 1, & \text{falls } C \text{ den Bogen } ij \text{ von } D \text{ durchläuft,} \\ -1, & \text{falls } C \text{ den Bogen } ji \text{ von } D \text{ durchläuft,} \\ 0, & \text{sonst.} \end{cases}$$

Satz 7.3. Sei x ein Zirkulation. Dann ist x optimal für das Minimalkosten-Zirkulations-Problem genau dann, wenn $G(x)$ keinen (gerichteten) negativen Kreis bzgl. der Kostenfunktion

$$\bar{w}_{ij} = \begin{cases} w_{ij}, & \text{falls } ij \in A_1(x), \\ -w_{ji}, & \text{falls } ij \in A_2(x) \end{cases}$$

enthält.

Beweis. Wir zeigen zunächst die Notwendigkeit der Bedingung. Sei dafür C ein gerichteter Kreis in $G(x)$ mit negativen Kosten. Dann gibt es ein $\varepsilon > 0$, so dass $x' := x + \varepsilon \chi^C$ wieder eine Zirkulation ist, d. h., x' erfüllt sowohl (7.2a) als auch (7.2b). Außerdem gilt

$$\begin{aligned} \sum_{a \in A} w_a x_a &= \sum_{a \in C} w_a x_a + \sum_{a \in A \setminus C} w_a x_a \\ &> \sum_{a \in C} w_a x'_a + \sum_{a \in A \setminus C} w_a x_a \\ &= \sum_{a \in C} w_a x'_a + \sum_{a \in A \setminus C} w_a x'_a \\ &= \sum_{a \in A} w_a x'_a, \end{aligned}$$

also war x nicht optimal.

Wir zeigen jetzt noch, dass die Bedingung auch hinreichend ist. Dazu habe jeder gerichtete Kreis in $G(x)$ nicht-negative Kosten und es sei x' eine beliebige Zirkulation. Dann erfüllt $x' - x$ ebenfalls die Flusserhaltungsbedingung (7.2a) und es gibt gerichtete Kreise C_1, \dots, C_m in $G(x)$ mit $\lambda_1, \dots, \lambda_m > 0$, so dass

$$x' - x = \sum_{j=1}^m \lambda_j \chi^{C_j}$$

gilt. Weiter gilt

$$\sum_{a \in A} w_a x'_a - \sum_{a \in A} w_a x_a = \sum_{a \in A} w_a (x'_a - x_a) = \sum_{j=1}^m \lambda_j w(C_j) \geq 0$$

mit

$$w(C_j) = \sum_{a \in C_j} w_a x_a.$$

Also ist x optimal. □

7.2 Der Kreis-Löschungs-Algorithmus

- Bemerkung 7.4.**
1. Die Bestimmung eines zulässigen Flusses in Schritt 1 kann mithilfe eines Maximalfluss-Algorithmus durchgeführt werden.
 2. Die Bestimmung eines negativen Kreises in Schritt 4 ist beispielsweise mit Verfahren aus Kapitel 5 möglich.

Algorithmus 18 Kreis-Löschungs-Algorithmus

Eingabe: Ein Digraph $D = (V, A)$ mit Bogenkapazitäten $c_a \in \mathbb{R}_{\geq 0}$ und Kosten w_a für alle $a \in A$, Knotenbedarfe b_v für alle $v \in V$.

Ausgabe: Ein Minimalkostenfluss x oder die Meldung, dass es keinen zulässigen Fluss gibt.

- 1: Bestimme einen zulässigen Fluss x .
 - 2: Falls es keinen zulässigen Fluss gibt: **return** "Problem ist unzulässig".
 - 3: **while** $G(x)$ enthält einen negativen Kreis **do**
 - 4: Bestimme negativen Kreis C .
 - 5: Setze $\varepsilon = \min\{r_{ij} : ij \in C\}$.
 - 6: Augmentiere ε Flusseinheiten entlang C und aktualisiere $G(x)$.
 - 7: **return** x
-

Satz 7.5. Algorithmus 18 arbeitet korrekt. Für $\theta = \max_{a \in A} c_a$, $\nu = \max_{a \in A} w_a$ und ganzzahlige Eingabedaten beträgt seine Laufzeit $\mathcal{O}(nm^2 + nm^2\theta\nu)$.

Beweis. Die Korrektheit folgt direkt aus Satz 7.3.

Schritt 1 wird durch Verwendung des Edmonds–Karp-Algorithmus in $\mathcal{O}(nm^2)$ durchgeführt; vgl. Satz 6.32. Die Bestimmung eines negativen Kreises kann in $\mathcal{O}(mn)$ durchgeführt werden. Die Schritte 5 und 6 benötigen jeweils $\mathcal{O}(m)$.

Es bleibt also noch die Anzahl der Iteration von Schritt 3–6 abzuschätzen. Der Startfluss hat Kosten von höchstens $m\theta\nu$ und der kostenminimale Fluss hat Kosten von mindestens $-m\theta\nu$. Da $\varepsilon \geq 1$ gilt nach Schritt 5 folgt daraus, dass die maximale Anzahl der Iterationen höchstens $\mathcal{O}(m\theta\nu)$ ist.

Insgesamt haben wir also eine Laufzeit von $\mathcal{O}(nm^2 + nm^2\theta\nu)$. \square

Algorithmus 18 ist wieder sehr generisch formuliert und ist kein Polynomi-
alzeitalgorithmus. Wählt man aber in Schritt 4 nicht irgendeinen negativen
Kreis, sondern einen Kreis, der $\bar{w}(C)/|C|$ minimiert, so kann man zeigen, dass
höchstens $\mathcal{O}(nm^2 \log n)$ Durchläufe der Schritte 3–6 benötigt werden. Einen
sogenannten minimalen Durchschnittskreis kann man in $\mathcal{O}(mn)$ bestimmen.
Beweise für diese Aussagen finden Sie beispielsweise in Kapitel 10 von Ahuja
u. a. [1]. Damit gilt der folgende Satz.

Satz 7.6. Wird in Schritt 4 ein minimaler Durchschnittskreis gewählt, so
beträgt die Gesamtlaufzeit $\mathcal{O}(n^2m^3 \log n)$.

Satz 7.7. Sind die Bogenkapazitäten c und die Bedarfe b ganzzahlig, so
berechnet Algorithmus 18 einen ganzzahligen Minimalkosten-Fluss.

Beweis. Aus Satz 6.9 folgt, dass der Startfluss nach Schritt 1 ganzzahlig ist.

Da alle Daten ganzzahlig sind, ist ε in Schritt 5 ebenfalls ganzzahlig und damit auch der neue Fluss nach Schritt 6. Also bleibt x während des gesamten Algorithmus ganzzahlig. \square

Es gibt eine Vielzahl unterschiedlicher Algorithmen für Minimalkosten-Fluss-Probleme. Aus zeitlichen Gründen möchten wir hier aber nicht näher darauf eingehen und den interessierten Leser abermals auf Ahuja et al. [1] verweisen.

Kapitel 8

Unabhängigkeitssysteme und Matroide

Viele kombinatorische Optimierungsprobleme können folgendermaßen formuliert werden. Für ein gegebenes Mengensystem (E, \mathcal{I}) (d.h. eine endliche Menge E mit Potenzmenge 2^E und eine Familie $\mathcal{I} \subseteq 2^E$) mit einer Gewichtsfunktion $c : 2^E \rightarrow \mathbb{R}$, bestimme man ein Element aus \mathcal{I} mit minimalem bzw. maximalem Gewicht.

In diesem Kapitel beschränken wir uns auf diejenigen kombinatorischen Optimierungsprobleme, wo (E, \mathcal{I}) ein Unabhängigkeitssystem oder sogar ein Matroid ist. Hassler Whitney gebrauchte 1935 in seinem grundlegenden Artikel [21] den Begriff Matroid. Wie das Wort bereits andeutet, konzipierte er ein Matroid als eine abstrakte Verallgemeinerung einer Matrix.

Zuerst führen wir Unabhängigkeitssysteme und anschließend Matroide ein und zeigen, dass viele kombinatorische Optimierungsprobleme in diesem Kontext beschrieben werden können. Es gibt mehrere äquivalente Axiomensysteme für Matroide und eine interessante Dualitätsrelation. Der Hauptgrund, weswegen Matroide wichtig sind, ist, dass ein einfacher Greedy-Algorithmus zur Optimierung in Matroiden benutzt werden kann. Abschließend betrachten wir den Schnitt von Unabhängigkeitssystemen und Matroiden.

8.1 Unabhängigkeitssysteme

Zunächst definieren wir den Begriff Unabhängigkeitssystem. Später werden wir diese Definition erweitern, um die Definition eines Matroids zu erhalten.

Definition 8.1 (Unabhängigkeitssystem). Ein Mengensystem (E, \mathcal{I}) mit

$\mathcal{I} \subseteq 2^E$ heißt *Unabhängigkeitssystem* auf E , wenn es die folgenden Axiome erfüllt:

(I1) $\emptyset \in \mathcal{I}$,

(I2) $I_1 \subseteq I_2 \in \mathcal{I} \Rightarrow I_1 \in \mathcal{I}$.

Die Teilmengen von E , die in \mathcal{I} enthalten sind, heißen unabhängig und alle übrigen Teilmengen von E heißen abhängig.

Mit jedem Unabhängigkeitssystem (E, \mathcal{I}) sind auf kanonische Weise andere Mengensysteme verbunden, welche wir nun einführen werden.

Definition 8.2 (Kreis, Kreissystem). Eine inklusionsminimale abhängige Teilmenge von E heißt *Kreis*. Die Menge aller Kreise eines Unabhängigkeitssystems (E, \mathcal{I}) heißt *Kreissystem* und wird mit \mathcal{C} bezeichnet.

Definition 8.3 (Basis, Basissystem). Eine inklusionsmaximale unabhängige Teilmenge von E heißt *Basis*. Die Menge aller Basen eines Unabhängigkeitssystems (E, \mathcal{I}) heißt *Basissystem* und wird mit \mathcal{B} bezeichnet.

Definition 8.4 (Antikette). Eine *Antikette* ist eine Menge paarweise nicht vergleichbarer Elemente.

Kreissysteme und Basissysteme sind Antiketten, d.h. Systeme von Mengen, so dass keine zwei Mengen ineinander enthalten sind.

Offenbar induziert jedes Unabhängigkeitssystem (E, \mathcal{I}) ein eindeutig bestimmtes Kreissystem und ein eindeutig bestimmtes Basissystem. Es gilt auch die Umkehrung, wie wir nachfolgend (ohne Beweis) für Kreis- und Basissysteme skizzieren.

Ist \mathcal{B} eine Antikette auf E , so ist

$$\mathcal{I} := \{I \subseteq E : \exists B \in \mathcal{B} \text{ mit } I \subseteq B\}$$

ein Unabhängigkeitssystem, und \mathcal{B} ist das zu (E, \mathcal{I}) gehörige Basissystem.

Ist $\mathcal{C} \neq \{\emptyset\}$ eine Antikette auf E , so ist

$$\mathcal{I} := \{I \subseteq E : I \text{ enthält kein Element von } \mathcal{C}\} \quad (8.1)$$

ein Unabhängigkeitssystem, und \mathcal{C} ist das zu (E, \mathcal{I}) gehörige Kreissystem.

Sei für jedes Element $e \in E$ ein Gewicht $c_e \in \mathbb{R}$ gegeben. Für $F \subseteq E$ setzen wir $c(F) := \sum_{e \in F} c_e$.

Definition 8.5 (Optimierungsproblem über einem Unabhängigkeitssystem). Ist ein Unabhängigkeitssystem (E, \mathcal{I}) mit Gewichtsfunktion $c : 2^E \rightarrow \mathbb{R}_{\geq 0}$

gegeben, so nennen wir

$$\max\{c(I) : I \in \mathcal{I}\} \quad (8.2)$$

Optimierungsproblem über einem Unabhängigkeitssystem (E, \mathcal{I}) .

Offenbar macht es hier keinen Sinn, Gewichte c_e zu betrachten, die nicht positiv sind. Denn wenn $I^* \in \mathcal{I}$ optimal ist, gilt $I' := I^* \setminus \{e \in E : c_e \leq 0\} \in \mathcal{I}$ und $c(I') \geq c(I^*)$, also ist I' ebenfalls eine optimale unabhängige Menge.

Definition 8.6 (Optimierungsproblem über einem Basissystem). Ist ein Basissystem \mathcal{B} auf E mit Gewichtsfunktion $c : 2^E \rightarrow \mathbb{R}$ gegeben, so nennen wir

$$\min\{c(B) : B \in \mathcal{B}\} \quad (8.3)$$

Optimierungsproblem über einem Basissystem \mathcal{B} .

Im folgenden Beispiel wollen wir einige Optimierungsprobleme über Unabhängigkeits- bzw. Basissysteme auflisten.

- Beispiel 23.** (a) Sei $G = (V, E)$ ein Graph. Eine Knotenmenge $S \subseteq V$ heißt *stabile Menge*, falls je zwei Knoten aus S nicht benachbart sind. Die Menge der stabilen Mengen ist ein Unabhängigkeitssystem. Eine gewichtsmaximale stabile Menge zu finden, ist ein Optimierungsproblem über einem Unabhängigkeitssystem.
- (b) Sei $G = (V, E)$ ein Graph. Eine Knotenmenge $S \subseteq V$ heißt *Clique*, falls je zwei Knoten aus S benachbart sind. Die Menge der Cliques ist ein Unabhängigkeitssystem. Eine gewichtsmaximale Clique zu finden, ist ein Optimierungsproblem über einem Unabhängigkeitssystem.
- (c) Sei $G = (V, E)$ ein Graph. Die Menge der *Wälder* von G ist ein Unabhängigkeitssystem. Einen maximalen Wald mit minimalem Gewicht zu finden ist somit ein Optimierungsproblem über einem Basissystem.
- (d) Sei $G = (V, E)$ ein Graph. Ein *Matching* M in G ist eine Kantenmenge $M \subseteq E$, so dass jeder Knoten aus V höchstens zu einer Kante aus M inzident ist. Die Menge aller Matchings ist ein Unabhängigkeitssystem. Ein maximales Matching zu bestimmen, ist ein Optimierungsproblem über einem Unabhängigkeitssystem.
- (e) Gegeben seien Zahlen $n \in \mathbb{N}$, $c_j \geq 0$, $a_j \geq 0$ für $1 \leq j \leq n$ und $b \geq 0$. Das Problem eine Teilmenge $S \subseteq \{1, \dots, n\}$ mit $\sum_{j \in S} a_j \leq b$ und maximalen Zielfunktionswert $\sum_{j \in S} c_j$ zu bestimmen, wird als *Rucksack-Problem* (engl. *Knapsack-Problem*) bezeichnet. Die Menge aller zulässigen Lösungen $\mathcal{I} = \{I \subseteq E : \sum_{j \in I} a_j \leq b\}$ mit $E = \{1, \dots, n\}$ ist ein Unabhängigkeitssystem (E, \mathcal{I}) . Dabei handelt es sich um ein Optimierungsproblem über einem Unabhängigkeitssystem.

- (f) Gegeben sei ein vollständiger Digraph $D = (V, A)$ mit Bogengewichten c_{ij} für alle $(i, j) \in A$. Die Aufgabe einen gerichteten Hamiltonkreis mit minimalem Gewicht zu finden, heißt *asymmetrisches Travelling-Salesman-Problem* (ATSP). Die Menge aller gerichteten Hamiltonkreise ist ein Basissystem eines Unabhängigkeitssystems. Das ATSP ist somit ein Optimierungsproblem über einem Basissystem.

△

Unabhängigkeitssysteme sind sehr allgemeine Objekte und besitzen häufig zu wenig Struktur, um wirklich tiefliegende Aussagen über sie machen zu können. Deshalb wollen wir im nächsten Abschnitt eine spezielle Klasse von Unabhängigkeitssystemen betrachten.

8.2 Matroide

Wie die Beispiele aus dem vorigen Abschnitt zeigen, enthalten Optimierungsprobleme über Unabhängigkeits- oder Basissysteme sowohl polynomial lösbare Probleme (z.B. Bestimmung minimaler Spannbaum) als auch Probleme für die keine polynomialen Lösungsverfahren bekannt sind (z.B. ATSP). Man wird daher nicht erwarten können, dass für diese Probleme eine "gute" Lösungstheorie existiert. Wir wollen nun eine Spezialklasse von Unabhängigkeitssystemen einführen, für die es so etwas gibt.

Definition 8.7 (Matroid). Ein Unabhängigkeitssystem (E, \mathcal{I}) , das das zusätzliche Axiom

$$(I3) \quad I, J \in \mathcal{I} \text{ mit } |I| < |J| \implies \exists j \in J \setminus I \text{ mit } I \cup \{j\} \in \mathcal{I}$$

erfüllt, nennt man *Matroid* und bezeichnet es mit $M = (E, \mathcal{I})$.

Satz 8.8. Die folgenden Unabhängigkeitssysteme (E, \mathcal{I}) sind Matroide:

- (a) E ist die Spaltenmenge einer Matrix A über irgendeinem Körper \mathbb{K} und $\mathcal{I} := \{F \subseteq E : \text{die Spalten in } F \text{ sind linear unabhängig bzgl. } \mathbb{K}\}$.
- (b) E ist die Kantenmenge eines ungerichteten Graphen $G = (V, E)$ und $\mathcal{I} := \{F \subseteq E : (V, F) \text{ ist ein Wald}\}$.
- (c) E ist die Kantenmenge eines ungerichteten Graphen G , S ist eine stabile Menge in G , $k \in \mathbb{Z}_{\geq 0}$ und $\mathcal{I} := \{F \subseteq E : |\delta(s) \cap F| \leq k \forall s \in S\}$.
- (d) E ist die Bogenmenge A eines Digraphen $D = (V, A)$, $S \subseteq V$, $k \in \mathbb{Z}_{\geq 0}$ und $\mathcal{I} := \{F \subseteq E : |\delta^{\text{in}}(s) \cap F| \leq k \forall s \in S\}$.

- (e) E ist die Bogenmenge A eines Digraphen $D = (V, A)$, $S \subseteq V$, $k \in \mathbb{Z}_{\geq 0}$ und $\mathcal{I} := \{F \subseteq E : |\delta^{\text{out}}(s) \cap F| \leq k \ \forall s \in S\}$.

Beweis. In allen Fällen ist klar, dass (E, \mathcal{I}) jeweils ein Unabhängigkeitssystem ist. Somit bleibt zu zeigen, dass Definition 8.7 gilt.

- (a) Dies folgt aus dem Basisergänzungssatz der linearen Algebra.
- (b) Ang., es seien $I, J \in \mathcal{I}$ mit $|I| < |J|$ und es wäre $I \cup \{j\} \notin \mathcal{I}$ für alle $j \in J \setminus I$. Für jede Kante $j = \{v, w\} \in J$ sind v und w in derselben Zusammenhangskomponente von (V, I) . Also ist jede Zusammenhangskomponente von (V, J) eine Teilmenge einer Zusammenhangskomponente von (V, I) . Somit ist die Anzahl p der Zusammenhangskomponenten des Waldes (V, J) nicht kleiner als die Anzahl der Zusammenhangskomponenten q des Waldes (V, I) . Daraus folgt $|V| - |J| = p \geq q = |V| - |I|$ oder $|J| \leq |I|$, was ein Widerspruch zur Annahme ist.
- (c) Seien $I, J \in \mathcal{I}$ mit $|I| < |J|$ und $S' := \{s \in S : |\delta(s) \cap I| = k\}$. Da $|I| < |J|$ und $|\delta(s) \cap J| \leq k$ für alle $s \in S'$, gibt es ein $j \in J \setminus I$ mit $j \notin \delta(s)$ für $s \in S'$. Somit ist $I \cup \{j\} \in \mathcal{I}$.
- (d) Der Beweis verläuft, bis auf ein Ersetzen von δ durch δ^{in} , analog zu (c).
- (e) Analog zu (c).

□

Zwei der Matroide aus Satz 8.8 haben besondere Namen. Das Matroid in (a) heißt *Vektormatroid* von A und wird mit $M[A]$ bezeichnet. Das in (b) dargestellte Matroid wird *Kreismatroid* von G genannt und mit $M(G)$ bezeichnet.

Beispiel 24. Es sei die Matrix $A \in \mathbb{R}^{2 \times 4}$ mit

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

gegeben. Die Menge der Spalten von A ist $E = \{1, 2, 3, 4, 5\}$ und mit

$$\mathcal{I} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{5\}, \{1, 2\}, \{1, 3\}, \{1, 5\}, \{2, 5\}, \{3, 5\}\}$$

erhalten wir das Vektormatroid $M[A]$. Das Basissystem von $M[A]$ ist $\mathcal{B} = \{\{1, 2\}, \{1, 3\}, \{1, 5\}, \{2, 5\}, \{3, 5\}\}$ und die Familie der abhängigen Mengen ist

$$\{\{4\}, \{1, 4\}, \{2, 4\}, \{3, 4\}, \{4, 5\}, \{2, 3\}\} \cup \{X \subseteq E \mid |X| \geq 3\}.$$

Somit sind

$$\Theta := \{\{4\}, \{2, 3\}, \{1, 2, 5\}, \{1, 3, 5\}\}$$

die minimal abhängigen Mengen, deren Teilmengen alle unabhängig sind.

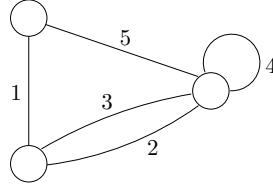


Abbildung 8.1: Kreismatroid $M(G)$

Sei $G = (V, E)$ der Graph aus Abb. 8.1 und $M(G)$ das entsprechende Kreismatroid. Dann gilt $E = \{1, 2, 3, 4, 5\}$ und das Kreissystem ist $\mathcal{C} = \{\{4\}, \{2, 3\}, \{1, 2, 5\}, \{1, 3, 5\}\}$. Es ist zu sehen, dass \mathcal{C} den minimal abhängigen Mengen Θ von $M[A]$ entspricht. \triangle

8.2.1 Kreisaxiome und Basisaxiome

In der Matroidtheorie gibt es kein Standardaxiomensystem. Dies liegt daran, dass bei unterschiedlichem Zugang sich verschiedene Strukturen in den Matroiden zur Axiomatisierung anbieten. Neben Definition 8.7 werden wir nun zwei weitere Axiomensysteme kennenlernen. Zunächst betrachten wir die Kreisaxiome, die bei einem graphentheoretischen Zugang näherliegend sind. Anschließend gehen wir auf die Basisaxiome, die durch die lineare Algebra motiviert sind, ein.

Lemma 8.9. Sei \mathcal{C} das Kreissystem eines Matroids $M = (E, \mathcal{I})$. Dann gilt

- (C1) $\emptyset \notin \mathcal{C}$,
- (C2) $C_1, C_2 \in \mathcal{C}$ und $C_1 \subseteq C_2 \Rightarrow C_1 = C_2$,
- (C3) $C_1, C_2 \in \mathcal{C}, C_1 \neq C_2, z \in C_1 \cap C_2 \Rightarrow \exists C_3 \in \mathcal{C}$ mit $C_3 \subseteq (C_1 \cup C_2) \setminus \{z\}$

Beweis. (C1) und (C2) folgen aus der Definition eines Kreises. Seien also C_1 und C_2 verschiedene Kreise und $e \in C_1 \cap C_2$. Angenommen, $(C_1 \cup C_2) \setminus \{e\} \in \mathcal{I}$. Wegen (C2) ist $C_2 \setminus C_1$ nicht leer und wir wählen $f \in C_2 \setminus C_1$. Da C_2 minimal abhängig ist, gilt $C_2 \setminus \{f\} \in \mathcal{I}$. Wir wählen eine maximale unabhängige Menge I mit $C_2 \setminus \{f\} \subseteq I \subseteq C_1 \cup C_2$. Offensichtlich gilt $f \notin I$. Da C_1 ein Kreis ist, existiert ein Element $g \in C_1$ mit $g \notin I$. Aufgrund von $f \in C_2 \setminus C_1$, sind f und g unterschiedlich. Somit gilt

$$|I| \leq |(C_1 \cup C_2) \setminus \{f, g\}| = |C_1 \cup C_2| - 2 < |(C_1 \cup C_2) \setminus \{e\}|.$$

Nun wenden wir (I3) mit $I_1 = I$ und $I_2 = (C_1 \cup C_2) \setminus \{e\}$ an. Daraus folgt ein Widerspruch zur Maximalität von I . \square

Wie der folgende Satz zeigt, können Matroide über (C1), (C2) und (C3) charakterisiert werden.

Satz 8.10. Sei E endlich und $\mathcal{C} \subseteq 2^E$ eine Mengenfamilie, die (C1), (C2) und (C3) erfüllt. Sei $\mathcal{I} := \{I \subseteq E : I \text{ enthält kein Element von } \mathcal{C}\}$, dann ist (E, \mathcal{I}) ein Matroid und \mathcal{C} sein Kreissystem.

Beweis. Siehe Übung. □

Ein Kreis, der aus einer unabhängigen Menge durch Hinzunahme eines Elementes entsteht, ist eindeutig:

Lemma 8.11. Sei $M = (E, \mathcal{I})$ ein Matroid, $I \in \mathcal{I}$, $e \in E$ und $I \cup \{e\} \notin \mathcal{I}$. Dann gibt es genau einen Kreis $C \subseteq I \cup \{e\}$ und es gilt $e \in C$.

Beweis. Zu zeigen ist die Eindeutigkeit. Angenommen, es gäbe zwei solcher Kreise C_1 und C_2 mit $C_1 \neq C_2$, dann würde nach (C3) die Menge $((C_1 \cup C_2) \setminus \{e\}) \subseteq I$ einen Kreis enthalten, was ein Widerspruch zu $I \in \mathcal{I}$ ist. □

Lemma 8.12. Sind B_1 und B_2 Basen eines Matroids $M = (E, \mathcal{I})$, so ist $|B_1| = |B_2|$.

Beweis. Angenommen, es gilt $|B_1| < |B_2|$. Da $B_1, B_2 \in \mathcal{I}$, existiert nach (I3) ein Element $e \in B_2 \setminus B_1$ mit $B_1 \cup \{e\} \in \mathcal{I}$. Dies ist aber ein Widerspruch zur Maximalität von B_1 . Somit gilt $|B_1| \geq |B_2|$ und auch $|B_2| \geq |B_1|$. □

Lemma 8.13. Sei \mathcal{B} das Basissystem eines Matroids $M = (E, \mathcal{I})$. Dann gilt

(B1) $\mathcal{B} \neq \emptyset$,

(B2) $B_1, B_2 \in \mathcal{B}, e \in B_1 \setminus B_2 \Rightarrow \exists f \in B_2 \setminus B_1 \text{ mit } (B_1 \setminus \{e\}) \cup \{f\} \in \mathcal{B}$.

Beweis. Da $\emptyset \in \mathcal{I}$ ist, gilt (B1). Sowohl $B_1 \setminus \{e\}$ als auch B_2 sind unabhängige Mengen. Wegen Lemma 8.12 gilt $|B_1 \setminus \{e\}| < |B_2|$. Also gibt es nach (I3) ein $f \in B_2 \setminus (B_1 \setminus \{e\})$ mit $(B_1 \setminus \{e\}) \cup \{f\} \in \mathcal{I}$ und somit existiert eine maximal unabhängige Menge B'_1 mit $(B_1 \setminus \{e\}) \cup \{f\} \subseteq B'_1$. Aus Lemma 8.12 folgt $|B'_1| = |B_1|$ und weiter $|B_1| = |(B_1 \setminus \{e\}) \cup \{f\}|$. Da $(B_1 \setminus \{e\}) \cup \{f\} = B'_1$ ist $(B_1 \setminus \{e\}) \cup \{f\}$ eine Basis von M . □

Satz 8.14. Sei E endlich und $\mathcal{B} \subseteq 2^E$ eine Mengenfamilie, die (B1) und (B2) erfüllt. Sei $\mathcal{I} := \{I \subseteq E : \exists B \in \mathcal{B} \text{ und } I \subseteq B\}$, dann ist (E, \mathcal{I}) ein Matroid und \mathcal{B} sein Basissystem.

Beweis. Siehe Übung. □

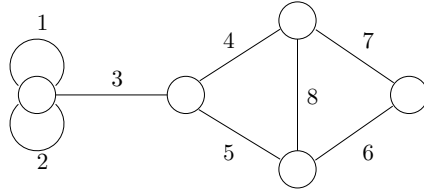


Abbildung 8.2: Kreismatroid $M(G)$

Betrachte ein Matroid $M = (E, \mathcal{I})$ und $X \subseteq E$. Sei $\mathcal{I}_X := \{I \subseteq X : I \in \mathcal{I}\}$, dann ist das Paar $M|_X = (X, \mathcal{I}_X)$ ebenfalls ein Matroid, welches als *Restriktion* von M auf X bezeichnet wird.

Definition 8.15 (Rang). Sei $M = (E, \mathcal{I})$ ein Matroid, $X \subseteq E$ und $B \in \mathcal{I}_X$ eine Basis von $M|_X$. Dann bezeichnen wir die Funktion $r : 2^E \rightarrow \mathbb{N}_{\geq 0}$ mit

$$r(X) := |B|$$

als *Rang* von X .

Falls in Definition 8.15 $X = E$ ist, schreibt man statt $r(E)$ oft auch nur $r(M)$.

Bemerkung 8.16. Sei $G = (V, E)$ ein Graph mit Kreismatroid $M = M(G)$ und $\omega(G)$ die Anzahl seiner Zusammenhangskomponenten, dann gilt

$$r(M) = |V| - \omega(G).$$

Beispiel 25. Sei $M = M(G)$ mit G der Graph in Abbildung 8.2. G ist zusammenhängend, also $\omega(G) = 1$, und $r(M) = |V| - 1 = 4$. Würden wir nur den induzierten Subgraphen für $X = \{4, 5, 6, 7, 8\}$ betrachten, dann ist $\{4, 5, 6\}$ eine Basis und $r(X) = 3$. \triangle

Auch die Rangfunktion liefert ein Axiomensystem. Darauf möchten wir an dieser Stelle aber nicht weiter eingehen.

8.2.2 Dualität und Isomorphie

Wir haben in dieser Vorlesung bereits duale Graphen von planaren Graphen und duale algorithmische Konzepte im Kapitel über maximale Flüsse kennengelernt. Hier wollen wir nun die Dualität von Matroiden betrachten.

In Lemma 8.13 (B2) wir zunächst die Basis verkleinert und dann wieder aufgefüllt. Gehen wir umgekehrt vor, so erhalten wir einen Kreis, aus dem wir ein Element entfernen können.

Lemma 8.17. Sei \mathcal{B} das Basissystem eines Matroids. Dann gilt

$$(B2)^* \quad B_1, B_2 \in \mathcal{B}, e \in B_2 \setminus B_1 \Rightarrow \exists f \in B_1 \setminus B_2 \text{ mit } (B_1 \cup \{e\}) \setminus \{f\} \in \mathcal{B}.$$

Beweis. Dies folgt aus Lemma 8.11. \square

Satz 8.18. Sei $M = (E, \mathcal{I})$ ein Matroid mit Basissystem \mathcal{B} und $\mathcal{B}^* := \{E \setminus B : B \in \mathcal{B}\}$. Dann ist \mathcal{B}^* das Basissystem eines Matroids.

Beweis. Da $\mathcal{B} \neq \emptyset$, ist auch $\mathcal{B}^* \neq \emptyset$ und \mathcal{B}^* erfüllt (B1). Sei $B_1^*, B_2^* \in \mathcal{B}^*$, $e \in B_1^* \setminus B_2^*$, $B_1 = E \setminus B_1^*$ und $B_2 = E \setminus B_2^*$, dann gilt $B_1, B_2 \in \mathcal{B}$ und

$$B_1^* \setminus B_2^* = (E \setminus B_1) \setminus (E \setminus B_2) = B_2 \setminus B_1.$$

Wegen (B2)* aus Lemma 8.17 mit $e \in B_2 \setminus B_1$ existiert $f \in B_1 \setminus B_2$ mit $(B_1 \cup \{e\}) \setminus \{f\} \in \mathcal{B}$. Daraus folgt, dass $f \in B_2^* \setminus B_1^*$ und

$$\mathcal{B}^* \ni E \setminus ((B_1 \cup \{e\}) \setminus \{f\}) = ((E \setminus B_1) \setminus \{e\}) \cup \{f\} = (B_1^* \setminus \{e\}) \cup \{f\}.$$

Folglich erfüllt \mathcal{B}^* die Eigenschaft (B2) von Lemma 8.13. \square

Das Matroid aus Satz 8.18 heißt das *duale Matroid* von M und wird mit M^* bezeichnet.

Bemerkung 8.19. Aus Satz 8.18 folgen die Eigenschaften

$$(M^*)^* = M$$

und

$$r(M) + r(M^*) = |E|.$$

Definition 8.20 (Kokreis). Gegeben sei ein Graph $G = (V, E)$. Ein *Kokreis* ist eine Kantenmenge, deren Entfernung aus G die Anzahl der Zusammenhangskomponenten erhöht und die mengeninklusionsweise minimal bezüglich dieser Eigenschaft ist.

Beispiel 26. Betrachten wir den Graphen $G = (V, E)$ in Abb. 8.3. $\{1, 5, 6\}$ oder $\{2, 3\}$ wären Kantenmengen bei deren Entfernung G in zwei Zusammenhangskomponenten zerfällt. Die Kantenmengen $\{1, 6\}$ und $\{3\}$ sind Kokreise. \triangle

Jeder Kokreis ist offenbar ein Schnitt, und es ist einfach einzusehen, dass die Kokreise gerade die minimalen nicht-leeren Schnitte von G sind.

Definition 8.21 (Kokreismatroid). Sei $G = (V, E)$ ein Graph. Das duale Matroid $M^*(G)$ von $M(G)$ nennen wir *Kokreismatroid*.

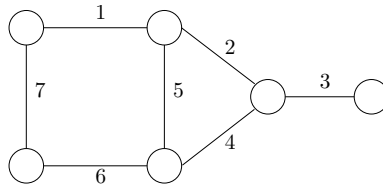


Abbildung 8.3: Graph $G = (V, E)$ mit 7 Kanten

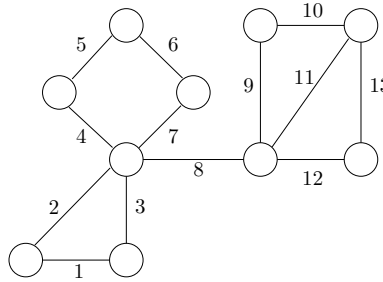


Abbildung 8.4: Graph $G = (V, E)$ mit 13 Kanten

Die minimal abhängigen Mengen des Vektormatroids entsprechen den Kreisen im Kreismatroid und die unabhängigen Mengen des Kreismatroids entsprechen den unabhängigen Spalten im Vektormatroid. Dies führt uns auf den Begriff der Isomorphie von Matroiden.

Definition 8.22. Zwei Matroide $M_1 = (E_1, \mathcal{I}_1)$ und $M_2 = (E_2, \mathcal{I}_2)$ heißen isomorph, in Zeichen $M_1 \cong M_2$, wenn es eine bijektive Abbildung $\psi : 2^{E_1} \rightarrow 2^{E_2}$ gibt mit

$$X \in \mathcal{I}_1 \iff \psi(X) \in \mathcal{I}_2 \quad \forall X \subseteq E_1.$$

Ein Matroid, das isomorph zu einem Kreismatroid ist, heißt *graphisch* und ein zu einem Kokreismatroid isomorphes Matroid wird *kographisch* genannt.

Beispiel 27. Betrachten wir den folgenden, in Abb. 8.4 dargestellten Graphen $G = (V, E)$ mit $E = \{1, 2, \dots, 13\}$. Das Kreissystem \mathcal{C} von $M(G)$ ist

$$\mathcal{C} = \{\{1, 2, 3\}, \{4, 5, 6, 7\}, \{9, 10, 11\}, \{11, 12, 13\}, \{9, 10, 12, 13\}\}.$$

Das Kreissystem \mathcal{C}^* des kographischen Matroids $M^*(G)$ auf E ist gegeben durch die Menge

$$\mathcal{C}^* = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{4, 5\}, \{4, 6\}, \{4, 7\}, \{5, 6\}, \{5, 7\}, \{6, 7\}, \{8\}, \{9, 10\}, \{9, 11, 12\}, \{9, 11, 13\}, \{10, 11, 12\}, \{10, 11, 13\}, \{12, 13\}\}.$$

△

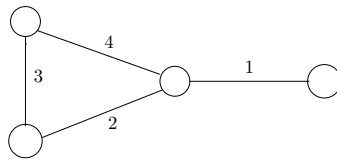


Abbildung 8.5: Graph $G = (V, E)$ mit 4 Kanten

Beispiel 28. Das Basissystem \mathcal{B} des graphischen Matroids aus Abb. 8.5) mit $E = \{1, 2, 3, 4\}$ ist gegeben durch

$$\mathcal{B} = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}\}$$

und das Basissystem \mathcal{B}^* des kographischen Matroids bzgl. G ist die Antikette $\mathcal{B}^* = \{\{4\}, \{3\}, \{2\}\}$. \triangle

Wir wenden uns nun zwei Graphen zu, die im Kontext planarer Graphen eine wichtige Rolle spielen.

Lemma 8.23. Sei M ein graphisches Matroid, dann gilt $M \cong M(G)$ für einen zusammenhängenden Graphen $G = (V, E)$.

Beweis. Siehe Übung. \square

Satz 8.24. Weder $M^*(K_5)$ noch $M^*(K_{3,3})$ sind graphisch.

Beweis. Siehe Übung. \square

Dass K_5 und $K_{3,3}$ nicht planar sind, ist kein zufälliges Zusammentreffen. Abschließend geben wir noch zwei Resultate zu planaren Graphen ohne Beweis an.

Satz 8.25. Ist $G = (V, E)$ planar, so ist $M^*(G)$ graphisch.

Korollar 8.26. Ist $G = (V, E)$ ein planarer Graph und $G^* = (V^*, E)$ der duale Graph, dann ist $M^*(G) = M(G^*)$.

Aus Korollar 8.26 ist ersichtlich, dass Matroide das Konzept von Dualität planarer Graphen verallgemeinern.

8.2.3 Greedy-Algorithmus

Wir werden nun zeigen, dass die mathematische Struktur eines Matroids vollständig durch ein algorithmisches Vorgehen und umgekehrt charakterisiert werden kann, was ein seltener Fall in der Mathematik ist.

Ein Greedy-Algorithmus für ein Optimierungsproblem über einem Unabhängigkeitssystem (8.2) ist in Algorithmus 19 gegeben. Möchte man Minimie-

Algorithmus 19 Greedy-Max für Unabhängigkeitssysteme

Eingabe: Ein Problem der Form (8.2), eine Grundmenge E und eine Funktion $c : 2^E \rightarrow \mathbb{R}_{\geq 0}$.

Ausgabe: Eine zulässige Lösung $I_G \in \mathcal{I}$.

1: Sortiere die Elemente in E derart, dass

$$c(I_1) \geq c(I_2) \geq \dots \geq c(I_m)$$

gilt, wobei $|E| = m$ gelte.

2: Setze $I_G \leftarrow \emptyset$.

3: **for** $k = 1 : m$ **do**

4: **if** $I_G \cup \{I_k\} \in \mathcal{I}$ **then**

5: Setze $I_G \leftarrow I_G \cup \{I_k\}$.

6: **return** I_G .

rungsprobleme anstelle von Maximierungsproblemen über Unabhängigkeitssystemen betrachten, so erhält man einen entsprechenden Algorithmus, wenn man in Schritt 1 derart sortiert, dass

$$c(I_1) \leq c(I_2) \leq \dots \leq c(I_m)$$

gilt.

Ziel des Rests dieses Kapitels ist zu charakterisieren, wann dieser Algorithmus das Maximierungsproblem über Unabhängigkeitssystemen tatsächlich löst. Dazu betrachten wir zunächst zwei Beispiele.

Beispiel 29 (Maximaler Wald minimalen Gewichts). Wir betrachten das Problem, in einem Graphen $G = (V, E)$ mit Gewichtsfunktion $c : E \rightarrow \mathbb{R}_{\geq 0}$ einen maximalen Wald minimalen Gewichts zu bestimmen. Wir haben uns in Beispiel 23 (c) bereits davon überzeugt, dass die Menge der Wälder von G ein Unabhängigkeitssystem ist. Wir können also Algorithmus 19 anwenden. Die Bedingung in Schritt 4 übersetzt sich dann in "Falls $I_G \cup \{I_k\}$ kreisfrei". In diesem Fall liefert Algorithmus 19 tatsächlich eine Optimallösung. \triangle

Satz 8.27. Algorithmus 19 mit aufsteigender Sortierung in Schritt 1 löst das Problem der Bestimmung eines maximalen Waldes mit minimalem Gewicht.

Beweis. Es handelt sich exakt um Algorithmus 9 von Kruskal. Demnach folgt der Satz direkt aus Satz 4.7. \square

Beispiel 30. Wir betrachten das Rucksack-Problem

$$\begin{aligned} \max_x \quad & \sum_{j=1}^n c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^n a_j x_j \leq b, \\ & x_j \in \{0, 1\} \quad \text{für alle } j = 1, \dots, n \end{aligned}$$

mit $c_j \geq 0$, $a_j \geq 0$ für alle j und $b \geq 0$. Wir haben uns in Beispiel 23 (e) davon überzeugt, dass es sich hierbei um ein Unabhängigkeitssystem handelt. Wenden wir Algorithmus 19 auf das Rucksackproblem an, so kann der Algorithmus allerdings beliebig schlechte Lösungen liefern. Dazu betrachten wir folgende Beispielinstantz des Rucksackproblems:

$$\begin{aligned} \max_x \quad & nx_1 + (n-1)x_2 + \dots + (n-1)x_n + (n-1)x_{n+1} \\ \text{s.t.} \quad & nx_1 + x_2 + \dots + x_n + x_{n+1} \leq n, \\ & x_j \in \{0, 1\} \quad \text{für alle } j = 1, \dots, n+1. \end{aligned}$$

Algorithmus 19 liefert die Lösung $I_G = \{1\}$ mit Zielfunktionswert n , während die Optimallösung den Wert $n(n-1)$ hat. \triangle

Wir haben also ein Beispiel gesehen, wo der Greedy-Algorithmus eine optimale Lösung liefert und ein Beispiel, bei dem er sehr schlechte Lösungen liefert. Wir charakterisieren jetzt, für welche Unabhängigkeitssysteme der Greedy-Algorithmus immer optimal funktioniert.

Lemma 8.28. Ist $M = (E, \mathcal{I})$ ein Matroid und $c : 2^E \rightarrow \mathbb{R}$, so liefert der Greedy-Algorithmus eine Optimallösung eines Optimierungsproblems über einem Unabhängigkeitssystem (E, \mathcal{I}) .

Beweis. Sei $B_G = \{b_1, \dots, b_r\}$ die vom Greedy-Algorithmus gefundene Lösung mit $c(b_1) \geq \dots \geq c(b_r)$. Angenommen, B_G ist keine Optimallösung. Sei dann $\bar{B} = \{\bar{b}_1, \dots, \bar{b}_{r'}\} \in \mathcal{I}$ eine solche und so gewählt, dass $|B_G \cap \bar{B}|$ maximal ist. Sei $b_\ell \in B_G \setminus \bar{B}$ mit kleinstem Index ℓ und $I_\ell := \{b_i \in B_G : i \leq \ell\} \in \mathcal{I}$. Da \bar{B} maximal unabhängig ist und b_ℓ nicht in \bar{B} ist, muss $\bar{B} \cup \{b_\ell\}$ abhängig sein. Nach Lemma 8.11 gibt es genau einen Kreis $C \subseteq \bar{B} \cup \{b_\ell\}$. Da $\mathcal{I} \ni I_\ell \subseteq \bar{B} \cup \{b_\ell\} \notin \mathcal{I}$, gibt es ein $\bar{b}_j \in C$ mit $j > \ell$ und $c(\bar{b}_j) \leq c(b_\ell)$. Nun ist aber $(\bar{B} \cup \{b_\ell\}) \setminus \{\bar{b}_j\} \in \mathcal{I}$ und $c((\bar{B} \cup \{b_\ell\}) \setminus \{\bar{b}_j\}) = c(\bar{B}) + c(b_\ell) - c(\bar{b}_j) \geq c(\bar{B})$ im Widerspruch zur Maximalität von $|B_G \cap \bar{B}|$. \square

Matroide sind also genau die Unabhängigkeitssysteme, bei denen der Greedy-Algorithmus die Optimallösung liefert. Man kann also das Axiom (I3) durch ein Greedyoptimalitätsaxiom ersetzen.

Satz 8.29. Sei (E, \mathcal{I}) eine Mengenfamilie. Dann ist \mathcal{I} die Familie der unabhängigen Mengen eines Matroids genau dann, wenn

- (I1) $\emptyset \in \mathcal{I}$,
- (I2) $I_1 \subseteq I_2 \in \mathcal{I} \Rightarrow I_1 \in \mathcal{I}$,
- (G) Für $c : 2^E \rightarrow \mathbb{R}$ liefert der Greedy-Algorithmus eine maximale Menge in \mathcal{I} mit maximalem Gewicht.

Beweis. Falls (E, \mathcal{I}) ein Matroid ist, gelten (I1) und (I2). Wir zeigen, dass unter (I1) und (I2) die Bedingungen (I3) und (G) äquivalent sind. Die Hinrichtung (I3) \Rightarrow (G) folgt aus Lemma 8.28. Wir wollen nun zeigen, dass (G) \Rightarrow (I3) gilt. Angenommen, dies wäre nicht der Fall. Seien $I_1, I_2 \in \mathcal{I}$ mit $|I_1| < |I_2|$, so dass $I_1 \cup \{e\} \notin \mathcal{I}$ für alle $e \in I_2 \setminus I_1$. Da $|I_1 \setminus I_2| < |I_2 \setminus I_1|$ und $I_1 \setminus I_2 \neq \emptyset$, können wir ein $\varepsilon > 0$ wählen mit

$$0 < (1 + \varepsilon)|I_1 \setminus I_2| < |I_2 \setminus I_1|.$$

Betrachte die Gewichtsfunktion

$$c(e) = \begin{cases} 2, & \text{falls } e \in I_1 \cap I_2, \\ 1/|I_1 \setminus I_2|, & \text{falls } e \in I_1 \setminus I_2, \\ (1 + \varepsilon)/|I_2 \setminus I_1|, & \text{falls } e \in I_2 \setminus I_1, \\ 0, & \text{sonst.} \end{cases}$$

Damit wählt der Greedy-Algorithmus zuerst die Elemente von $I_1 \cap I_2$ und dann alle Elemente von $I_1 \setminus I_2$. Per Annahme, werden keine Elemente von $I_2 \setminus I_1$ gewählt. Damit sind die übrigen Elemente der Greedy-Lösung B_G in $E \setminus (I_1 \cup I_2)$ enthalten. Daraus folgt

$$\begin{aligned} c(B_G) &= 2|I_1 \cap I_2| + |I_1 \setminus I_2|(1/|I_1 \setminus I_2|) \\ &= 2|I_1 \cap I_2| + 1. \end{aligned} \tag{8.4}$$

Aber wegen (I2) ist I_2 enthalten in einem maximalen Element $I'_2 \in \mathcal{I}$ und

$$\begin{aligned} c(I'_2) \geq c(I_2) &= 2|I_1 \cap I_2| + |I_2 \setminus I_1|((1 + \varepsilon)/|I_2 \setminus I_1|) \\ &= 2|I_1 \cap I_2| + 1 + \varepsilon. \end{aligned} \tag{8.5}$$

Aus (8.4) und (8.5) folgt $c(I'_2) > c(B_G)$ und somit ein Widerspruch. \square

8.2.4 Schnitt von Matroiden

Wir haben bereits gesehen, dass Matroide fundamentale mathematische Strukturen sind mit deren Hilfe der Begriff der linearen Unabhängigkeit verallgemeinert wird. Nun stellt sich natürlich die Frage, ob der Schnitt von Matroiden selbst wieder ein Matroid ist. Um dieser Fragestellung auf den Grund gehen zu können, müssen wir zunächst den Schnitt von Unabhängigkeitssystemen betrachten.

Satz 8.30. Der Schnitt von Unabhängigkeitssystemen $(E, \bigcap_i \mathcal{I}_i)$ für $i = 1, \dots, n$ ist ein Unabhängigkeitssystem.

Beweis. Sei $I \in \bigcap_i \mathcal{I}_i$ und $J \subseteq I$. Für alle $i = 1, \dots, n$ gilt daher $I \in \mathcal{I}_i$. Da (E, \mathcal{I}_i) abgeschlossen unter Inklusion ist, folgt $J \in \mathcal{I}_i$ für alle $i = 1, \dots, n$, insgesamt also $J \in \bigcap_i \mathcal{I}_i$. \square

Korollar 8.31. Jedes Unabhängigkeitssystem (E, \mathcal{I}) ist der Schnitt endlich vieler Matroide.

Beweis. Jedes Matroid ist per Definition ein Unabhängigkeitssystem. Mit Satz 8.30 folgt die Behauptung. \square

Das folgende Beispiel zeigt hingegen, dass der Schnitt von Matroiden i.A. kein Matroid ist.

Beispiel 31. Gegeben sind zwei Matroide:

$M_1 = (\{1, 2, 3, 4\}, \mathcal{I}_1)$ mit $\mathcal{I}_1 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 4\}\}$,
 $M_2 = (\{1, 2, 3, 4\}, \mathcal{I}_2)$ mit $\mathcal{I}_2 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 4\}, \{2, 3\}, \{3, 4\}\}$.

Wir erhalten als Schnitt

$$(E, \mathcal{I}_1 \cap \mathcal{I}_2) = (\{1, 2, 3, 4\}, \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{3, 4\}\}),$$

was selbst kein Matroid ist. \triangle

Besonders interessant sind Unabhängigkeitssysteme, die der Schnitt zweier Matroide sind. Das vielleicht wichtigste Beispiel ist hier das Matching-Problem in einem bipartiten Graphen $G = (A \dot{\cup} B, E)$. Betrachte die unabhängigen Mengen

$$\begin{aligned} \mathcal{I}_1 &:= \{F \subseteq E : |\delta(s) \cap F| \leq 1 \ \forall s \in A\}, \\ \mathcal{I}_2 &:= \{F \subseteq E : |\delta(s) \cap F| \leq 1 \ \forall s \in B\}. \end{aligned}$$

Nach Satz 8.8(c) sind $(E, \mathcal{I}_1), (E, \mathcal{I}_2)$ Matroide und $\mathcal{I}_1 \cap \mathcal{I}_2 = \{F \subseteq E : F \text{ ist ein Matching in } G\}$.

Definition 8.32 (Matroid-Intersektions-Problem). Sind zwei Matroide $(E, \mathcal{I}_1), (E, \mathcal{I}_2)$ durch Unabhängigkeitsorakel¹ gegeben und möchte man eine Menge $F \in \mathcal{I}_1 \cap \mathcal{I}_2$ mit maximalem $|F|$ bestimmen, spricht man vom *Matroid-Intersektions-Problem*.

¹Ein *Orakel* kann man als eine Art Subroutine interpretieren, welches Fragen eines speziellen Typs beantwortet. Das Unabhängigkeitsorakel bekommt als Eingabe $F \subseteq E$ und beantwortet die Frage "Ist F unabhängig?" mit ja/nein.

Edmonds [7] hat gezeigt, dass das Matroid-Intersektions-Problem in $O(|E|^3\theta)$ Zeit, wobei θ die maximale Komplexität der beiden Unabhängigkeitsorakel ist, gelöst werden kann.

Man fragt sich nun, ob auch über dem Durchschnitt von 3 Matroiden in polynomialer Zeit optimiert werden kann. Dies geht vermutlich i.A. nicht, wie das folgende Beispiel zeigt.

Wir betrachten das asymmetrische Travelling-Salesman-Problem (ATSP). Das zugehörige Unabhängigkeitssystem ist wie folgt definiert:

$$\mathcal{H} := \{S \subseteq A_n : \\ S \text{ ist Teilmenge eines gerichteten hamiltonschen Kreises in } D_n = (V, A_n)\},$$

wobei D_n der vollständige gerichtete Graph auf n Knoten ist. Wir modifizieren D_n zu dem Graphen $D'_n = (V', A'_n)$, indem wir den Knoten $1 \in V$ in zwei neue Knoten $1, n+1 \in V'$ aufspalten. Alle Bögen aus A_n , die den Knoten 1 aus D_n als Anfangsknoten haben, haben auch $1 \in V'$ als Anfangsknoten. Die Bögen aus A_n , die 1 $\in V$ als Endknoten haben, bekommen den neuen Knoten $n+1 \in V'$ als Endknoten zugewiesen. Diese Konstruktion bewirkt, dass jedem gerichteten hamiltonschen Kreis in D_n ein gerichteter hamiltonscher Weg von 1 nach $n+1$ in D'_n entspricht und umgekehrt. Das ATSP-Unabhängigkeitssystem \mathcal{H} ist dann identisch mit

$$\mathcal{H}' := \{S \subseteq A'_n : \\ S \text{ ist Teilmenge eines gerichteten hamiltonschen } (1, n+1)\text{-Wegs in } D'_n\}.$$

Betrachte auf A'_n die drei unabhängigen Mengen

$$\begin{aligned} \mathcal{I}_1 &:= \{W \subseteq A'_n : W \text{ ist Wald}\}, \\ \mathcal{I}_2 &:= \{F \subseteq A'_n : |\delta^{\text{in}}(v) \cap F| \leq 1 \ \forall v \in V'\}, \\ \mathcal{I}_3 &:= \{F \subseteq A'_n : |\delta^{\text{out}}(v) \cap F| \leq 1 \ \forall v \in V'\}, \end{aligned}$$

so sind nach Satz 8.8 $(A'_n, \mathcal{I}_1), (A'_n, \mathcal{I}_2), (A'_n, \mathcal{I}_3)$ Matroide, und es gilt $\mathcal{H}' = \mathcal{I}_1 \cap \mathcal{I}_2 \cap \mathcal{I}_3$.

Hiermit möchten wir die Betrachtung von Matroiden im Rahmen dieser Vorlesung abschließen. Wer sich tiefergehend mit Matroiden auseinandersetzen möchte, dem sei das Buch von Oxley [15] empfohlen.

Teil II

Lineare Optimierung und der Simplex-Algorithmus

Kapitel 9

Das Simplex-Verfahren

Ein motivierendes Beispiel

Auf einer Studentenparty werden drei alkoholische Getränke mit verschiedenen Preisen und mit unterschiedlichem Zuckergehalt zum Mixen angeboten; siehe Tabelle 9.1.

Ein Austauschstudent hat nur noch 3€ übrig, die er für ein Mischgetränk aus den drei Angeboten vollständig ausgeben möchte. Da der Student zudem Diabetiker ist, muss er auf seinen Zuckerspiegel achten und dafür sorgen, dass er noch genau 2 Broteinheiten (BE) zu sich nimmt.

Wir bezeichnen mit x_1, x_2 und x_3 die Menge (in 10 ml) der drei jeweiligen Getränke. Der Student erhält damit für diese Variablen das folgende lineare Gleichungssystem:

$$2x_1 + x_2 + 2x_3 = 3, \quad (9.1a)$$

$$2x_2 + x_3 = 2. \quad (9.1b)$$

Hierbei handelt es sich um ein Gleichungssystem der Form $Ax = b$. Wir bringen dieses Gleichungssystem mithilfe des Gauß-Algorithmus in eine für unsere Zwecke besser geeignete Form. Dazu subtrahieren wir das 1/2-fache

	Sugarfree Obstler	Bubblegum Sprit	Machfasch Flash
Kosten / 10 ml	2€	1€	2€
Zuckergehalt / 10 ml	0 BE	2 BE	1 BE

Tabelle 9.1: Getränke auf der Studentenparty

der Gleichung (9.1b) von der Gleichung (9.1a) und erhalten

$$\begin{aligned}2x_1 + \frac{3}{2}x_3 &= 2, \\ 2x_2 + x_3 &= 2.\end{aligned}$$

Anschließend multiplizieren wir noch beide Zeilen mit $1/2$. Dies liefert:

$$\begin{aligned}x_1 + \frac{3}{4}x_3 &= 1, \\ x_2 + \frac{1}{2}x_3 &= 1.\end{aligned}$$

Die Lösungsmenge des ursprünglichen Systems (9.1) ist damit durch

$$x_1 = 1 - \frac{3}{4}x_3, \quad (9.2a)$$

$$x_2 = 1 - \frac{1}{2}x_3 \quad (9.2b)$$

beschrieben. Wir sehen, dass das System beliebig viele Lösungen hat, die sich mithilfe von x_3 parametrisieren lassen. Für $x_3 = 0$ erhalten wir beispielsweise die spezielle Lösung $\bar{x} = (\bar{x}_1, \bar{x}_2, \bar{x}_3)^\top = (1, 1, 0)^\top$. Nachdem wir mehr als eine zulässige Lösung zur Verfügung haben, stellt sich die Frage, welche dieser Lösungen eine gegebene Zielfunktion optimiert, d. h., minimiert oder maximiert. In dieser Vorlesung beschäftigen wir uns nur mit linearen Problemen, d. h., wir beschäftigen uns auch nur mit linearen Zielfunktionen. In unserem aktuellen Problem haben wir also eine Zielfunktion

$$c^\top x = \sum_{i=1}^3 c_i x_i$$

zu optimieren.

Der beste Cocktail (Teil 1)

In unserem speziellen Fall nehmen wir an, dass der Student gerne möglichst nüchtern bleiben und daher den Alkoholgehalt seines Mischgetränks minimieren möchte. Sei dazu x eine beliebige zulässige Lösung für unser Zulässigkeitssystem (9.1). Dann können wir unter Zuhilfenahme der Gleichungen (9.2) die Zielfunktion wie folgt umschreiben:

$$\begin{aligned}c^\top x &= c_1 \left(1 - \frac{3}{4}x_3\right) + c_2 \left(1 - \frac{1}{2}x_3\right) + c_3 x_3 \\ &= c_1 + c_2 + \left(-\frac{3}{4}c_1 - \frac{1}{2}c_2 + c_3\right) x_3 \\ &= \alpha + \beta x_3\end{aligned} \quad (9.3)$$

mit

$$\alpha = c_1 + c_2, \quad \beta = -\frac{3}{4}c_1 - \frac{1}{2}c_2 + c_3.$$

Mithilfe der Darstellung in (9.2) erhalten wir also die folgende äquivalente Formulierung des Problems:

$$\begin{aligned} \min_x \quad & \alpha + \beta x_3 \\ \text{s.t.} \quad & x_1 = 1 - \frac{3}{4}x_3, \\ & x_2 = 1 - \frac{1}{2}x_3. \end{aligned}$$

Ist $\beta = 0$, so ist die Zielfunktion auf der Menge der zulässigen Lösungen konstant gleich α . Insbesondere ist dann die spezielle Lösung $\bar{x} = (1, 1, 0)^\top$ optimal. Ist $\beta > 0$, so ist die Zielfunktion auf der Menge der zulässigen Lösungen nach unten unbeschränkt, da wir $x_3 < 0$ beliebig klein wählen können. Auch für $\beta < 0$ ist die Zielfunktion nach unten unbeschränkt, da hier $x_3 > 0$ beliebig groß gewählt werden kann. Unser lineares Optimierungsproblem

$$\min \{c^\top x : x \text{ erfüllt (9.1)}\}$$

ist damit vollständig analysiert. Lineare Optimierung über linearen Gleichungssystemen ist nicht besonders spannend: Entweder sind alle zulässigen Lösungen optimal oder es gibt überhaupt keine Optimallösung, da das Problem nicht nach unten beschränkt ist. Wir werden später sehen, dass dies nicht nur für unser spezielles Beispiel, sondern auch im Allgemeinen, gilt.

Der beste Cocktail (Teil 2)

Wenn wir unsere Modellierung des Optimierungsproblems nochmal etwas genauer betrachten, stellen wir fest, dass wir wichtige Nebenbedingungen übersehen haben: Die Mengen x_1, x_2 und x_3 der Getränke sollten nicht-negativ sein! Damit sieht das Optimierungsproblem wie folgt aus:

$$\min \{c^\top x : x \text{ erfüllt (9.1) und } x \geq 0\},$$

oder, nochmal voll ausgeschrieben:

$$\begin{aligned} \min_x \quad & c_1x_1 + c_2x_2 + c_3x_3 \\ \text{s.t.} \quad & 2x_1 + x_2 + 2x_3 = 3, \\ & 2x_2 + x_3 = 2, \\ & x_1, x_2, x_3 \geq 0. \end{aligned} \tag{9.4}$$

Unsere Überlegungen von oben, die uns zu dem umformulierten linearen Gleichungssystem (9.2) geführt haben, bleiben nach wie vor gültig: Wir

können wieder nach x_1 und x_2 auflösen und in die Zielfunktion einsetzen, womit wir die folgende äquivalente Formulierung des Problems (9.4) erhalten:

$$\min_x \quad \alpha + \beta x_3 \quad (9.5a)$$

$$\text{s.t.} \quad x_1 = 1 - \frac{3}{4}x_3, \quad (9.5b)$$

$$x_2 = 1 - \frac{1}{2}x_3, \quad (9.5c)$$

$$x_1, x_2, x_3 \geq 0. \quad (9.5d)$$

Die spezielle Lösung $\bar{x} = (\bar{x}_1, \bar{x}_2, \bar{x}_3) = (1, 1, 0)^\top$, die wir für $\bar{x}_3 = 0$ aus Gleichung (9.2) erhalten, ist wieder zulässig und hat den Zielfunktionswert α . Das Einzige, das sich gegenüber der Variante ohne Vorzeichenrestriktion ändert, ist, dass wir unsere Fallunterscheidungen nach β neu durchführen müssen. Falls $\beta = 0$, so gilt für alle Lösungen des Systems (9.1), also insbesondere auch für alle nicht-negativen Lösungen x von Gleichung (9.1), weiterhin $c^\top x = \alpha$. Die Lösung $\bar{x} = (1, 1, 0)^\top$ mit $c^\top \bar{x} = \alpha$ bleibt optimal. Für $\beta > 0$ konnten wir vorher $x_3 < 0$ beliebig klein wählen. Dies ist nun nicht mehr der Fall. In der Tat gilt für jede zulässige Lösung x mit $x_3 \geq 0$ aufgrund der Gleichung (9.3) jetzt $c^\top x \geq \alpha$. Damit ist \bar{x} erneut als optimal nachgewiesen. Es verbleibt der Fall, dass $\beta < 0$ gilt. Wie anfangs führt eine Erhöhung von x_3 dann zu einer Verringerung der Zielfunktion. Allerdings wird diese Erhöhung durch $x_1, x_2 \geq 0$ und die Gleichungen (9.2) (bzw. durch die Gleichungen (9.5b) und (9.5c)) begrenzt. Wegen Gleichung (9.5b) muss

$$1 - \frac{3}{4}x_3 \geq 0,$$

also $x_3 \leq 4/3$ gelten. Entsprechend erhalten wir aus Gleichung (9.5c) die Bedingung

$$1 - \frac{1}{2}x_3 \geq 0,$$

also das $x_3 \leq 2$ gelten muss. Insgesamt ist die maximale Erhöhung γ von x_3 gegeben durch

$$\gamma = \min \left\{ \frac{4}{3}, 2 \right\} = \frac{4}{3}. \quad (9.6)$$

Wenn wir x_3 auf $\gamma = 4/3$ erhöhen und die Werte von x_1, x_2 gemäß der Gleichungen (9.2) anpassen, so ergibt sich die neue spezielle Lösung $\bar{x}^+ = (0, 1/3, 4/3)^\top$ mit dem Zielfunktionswert $c^\top \bar{x}^+ = \alpha + 4/3\beta$. Da $\beta < 0$ war, gilt insbesondere $c^\top \bar{x}^+ < c^\top \bar{x}$ und wir haben eine bessere Lösung gefunden. In unserem konkreten Beispiel nehmen wir an, dass die Alkoholgehalte der drei Getränke $c_1 = 4, c_2 = 2$ und $c_3 = 3$ sind. Tabelle 9.2 fasst die Daten der Getränke noch einmal zusammen. Dann erhalten wir α und β für die

	Sugarfree Obstler	Bubblegum Sprit	Machfasch Flash
Alkoholgehalt / 10 ml	4	2	3
Kosten / 10 ml	2€	1€	2€
Zuckergehalt / 10 ml	0 BE	2 BE	1 BE

Tabelle 9.2: Getränke auf der Studentenparty und ihr Alkoholgehalt

umgeschriebene Zielfunktion nach Gleichung (9.3) als:

$$\alpha = c_1 + c_2 = 4 + 2 = 6,$$

$$\beta = -\frac{3}{4}c_1 - \frac{1}{2}c_2 + c_3 = -3 - 1 + 3 = -1 < 0.$$

Hier lohnt es sich demnach, den Wert der Variablen x_3 zu erhöhen, da $\beta < 0$ ist. Wie wir bereits in Gleichung (9.6) ausgerechnet hatten, können wir x_3 maximal auf den Wert $4/3$ erhöhen und wir erhalten die neue Lösung $\bar{x}^+ = (0, 1/3, 4/3)^\top$ mit dem Zielfunktionswert

$$c^\top \bar{x}^+ = \alpha + \frac{4}{3}\beta = 6 - \frac{4}{3} = \frac{14}{3}.$$

Wie können wir jetzt weiterarbeiten? Ist \bar{x}^+ bereits eine optimale Lösung? Was die Vorgehensweise bei \bar{x} so einfach machte, war, dass wir die zwei Variablen x_1 und x_2 in Abhängigkeit der Variablen x_3 ausgedrückt hatten, wobei wir x_3 in $\bar{x} = (1, 1, 0)^\top$ den Wert 0 besaß. Im Prinzip vollzieht sich beim Übergang von \bar{x} zu \bar{x}^+ ein Rollentausch der Variablen x_1 und x_3 : x_1 erhält den Wert 0 und x_3 wird auf $4/3$ erhöht. Wir stellen daher das System der linearen Gleichungen derart um, dass x_2 und x_3 in Abhängigkeit von x_1 ausgedrückt werden (der genaue Rechenweg ist hier momentan nicht ganz so interessant, wir werden im nächsten Abschnitt das systematische Vorgehen genauer untersuchen):

$$x_3 = \frac{4}{3} - \frac{4}{3}x_1, \quad (9.7a)$$

$$x_2 = \frac{1}{3} + \frac{2}{3}x_1. \quad (9.7b)$$

Für jede zulässige Lösung $x = (x_1, x_2, x_3)^\top$ gilt also

$$\begin{aligned} c^\top x &= c_1 x_1 + c_2 \left(\frac{1}{3} + \frac{2}{3}x_1 \right) + c_3 \left(\frac{4}{3} - \frac{4}{3}x_1 \right) \\ &= 4x_1 + 2 \left(\frac{1}{3} + \frac{2}{3}x_1 \right) + 3 \left(\frac{4}{3} - \frac{4}{3}x_1 \right) \\ &= \frac{14}{3} + \left(4 + \frac{4}{3} - 4 \right) x_1 \\ &= \frac{14}{3} + \frac{4}{3}x_1. \end{aligned} \quad (9.8)$$

Außerdem erfüllt jede zulässige Lösung $x = (x_1, x_2, x_3)^\top$ auch die Vorzeichenbedingung $x_1 \geq 0$. Daher gilt dann aufgrund von Gleichung (9.8) auch $c^\top x \geq 14/3 = c^\top \bar{x}^+$ für jede zulässige Lösung x . Die spezielle Lösung $\bar{x}^+ = (0, 1/3, 4/3)^\top$ ist damit eine Optimallösung und unsere Vorgehensweise liefert dafür einen mathematischen Beweis.

9.1 Lineare Optimierungsprobleme mit Gleichungsrestriktionen

Wir verallgemeinern nun unser Beispiel aus dem letzten Abschnitt auf beliebige lineare Gleichungsrestriktionen der Form

$$Ax = b, \quad (9.9)$$

wobei A eine $m \times n$ Matrix und $b \in \mathbb{R}^m$ ist. Wir nehmen zunächst an, dass $\text{Rang } A = m$ (und damit insbesondere $m \leq n$) gilt. Dadurch ist das System (9.9) unterbestimmt und es gibt Optimierungspotential. In unserem Beispiel oben war $n = 3$ und $m = 2$, siehe Gleichungen (9.1), und wir hatten

$$A = \begin{bmatrix} 2 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix}, \quad b = \begin{pmatrix} 3 \\ 2 \end{pmatrix}.$$

Wegen $\text{Rang } A = m$ gibt es eine $m \times m$ Teilmatrix A' von A , die nicht-singulär ist. Die Indizes der Spalten dieser Teilmatrix A' und ihre Reihenfolge spielen eine besondere Rolle. Wir führen daher folgende Notation ein.

Für eine $m \times n$ Matrix A mit Spalten A_1, \dots, A_n und ein Tupel $J = (j_1, \dots, j_k)$ von paarweise verschiedenen Spaltenindizes sei $A_J = (A_{j_1}, \dots, A_{j_k})$ die Matrix, die aus den Spalten mit Indizes in J (in dieser Reihenfolge) entsteht. Wir schreiben dann auch $|J| = k$ für die Anzahl der Indizes in J und $J \subseteq \{1, \dots, n\}$.

Definition 9.1 (Basis, Basismatrix). Ist $B \subseteq \{1, \dots, n\}$ mit $|B| = m$ und A_B nicht-singulär, so nennen wir B eine *Basis* von A und A_B *Basismatrix*.

In unserem Beispiel-Problem ist beispielsweise $B = (1, 2)$ eine Basis und

$$A' = A_B = \begin{bmatrix} 2 & 1 \\ 0 & 2 \end{bmatrix}$$

eine Basismatrix. Ist B eine Basis von A , so können wir jeden Vektor $x \in \mathbb{R}^n$ in die *Basisvariablen* x_B und die *Nicht-Basisvariablen* x_N partitionieren:

$x = (x_B, x_N)^\top$.¹ Es gilt dann:

$$\begin{aligned} Ax = b &\iff A_B x_B + A_N x_N = b \\ &\iff A_B x_B = b - A_N x_N \\ &\iff x_B = A_B^{-1}(b - A_N x_N). \end{aligned} \quad (9.10)$$

Insbesondere ist $\bar{x} = (x_B, x_N)^\top = (A_B^{-1}b, 0)^\top$ eine spezielle Lösung des Systems $Ax = b$ aus Gleichung (9.9).

Definition 9.2 (Basislösung). Ist B eine Basis von A , so heißt der Vektor $\bar{x} = (\bar{x}_B, \bar{x}_N)^\top$ mit

$$\bar{x}_B = A_B^{-1}b, \quad \bar{x}_N = 0$$

die zur Basis B gehörende *Basislösung*.

In unserem motivierenden Beispiel des letzten Abschnitts ist $B = (1, 2)$ eine Basis mit zugehöriger Basislösung $\bar{x} = (\bar{x}_1, \bar{x}_2, \bar{x}_3)^\top = (1, 1, 0)^\top$. Wie in Gleichung (9.3) können wir die Zielfunktion für eine Lösung $x = (x_B, x_N)^\top$ von Gleichung (9.9) umschreiben:

$$\begin{aligned} c^\top x &= c_B^\top x_B + c_N^\top x_N \\ &= c_B^\top A_B^{-1}(b - A_N x_N) + c_N^\top x_N \\ &= c_B^\top A_B^{-1}b + (c_N^\top - c_B^\top A_B^{-1}A_N)x_N \\ &= \begin{pmatrix} c_B \\ c_N \end{pmatrix}^\top \begin{pmatrix} A_B^{-1}b \\ 0 \end{pmatrix} + z_N^\top x_N \\ &= c^\top \bar{x} + z_N^\top x_N, \end{aligned} \quad (9.11)$$

wobei \bar{x} die zu B gehörende Basislösung ist. Der Vektor

$$z_N = c_N - A_N^\top (A_B^\top)^{-1} c_B$$

heißt Vektor der *reduzierten Kosten* (oder einfach nur *reduzierte Kosten*). In unserem Beispiel-Problem ergaben sich unterschiedliche Fälle in Abhängigkeit von z_N . Ist $z_N = 0$ der Nullvektor, so ist die Zielfunktion auf der Menge der Lösungen des Gleichungssystems (9.9) konstant gleich $c^\top \bar{x}$. Insbesondere ist dann die Basislösung \bar{x} optimal. Sei daher nun $z_N \neq 0$, also $z_j \neq 0$ für mindestens ein $j \in N$. Wir betrachten für einen Skalar t den Vektor $x(t)$, definiert durch

$$\begin{aligned} x_j(t) &= t, \\ x_i(t) &= 0 \quad \text{für alle } i \in N \setminus \{j\}, \\ x_B(t) &= A_B^{-1}(b - A_N x_N(t)). \end{aligned}$$

¹An dieser Stelle sind wir notationell nicht ganz exakt, da wir eigentlich $(x_B^\top, x_N^\top)^\top$ schreiben müssten.

Offenbar gilt $Ax(t) = b$ und nach unserer Rechnung in Gleichung (9.11) gilt demnach

$$c^\top x(t) = c^\top \bar{x} + z_N^\top x_N(t) = c^\top \bar{x} + z_j t. \quad (9.12)$$

Gilt $z_j > 0$, so ist die Zielfunktion auf der Menge der zulässigen Lösungen nach unten unbeschränkt, da wir $x_j(t) = t < 0$ beliebig klein wählen können. Analog ist die Zielfunktion auch nach unten unbeschränkt, falls $z_j < 0$ für ein $j \in N$, da hier $x_j(t) = t > 0$ beliebig groß gewählt werden kann. Damit ergibt sich im allgemeinen Fall die gleiche Situation wie in unserem einführenden Beispiel-Problem: Das Optimierungsproblem

$$\min \{c^\top x : Ax = b\}$$

ist entweder unbeschränkt oder die Zielfunktion ist auf der Menge $\{x \in \mathbb{R}^n : Ax = b\}$ der zulässigen Lösungen konstant. Beide Fälle sind nicht sonderlich spannend.

9.2 Das Simplex-Verfahren für lineare Optimierungsprobleme in Standardform

Im einführenden Beispiel haben wir als nächstes die (für die konkrete Anwendung überaus sinnvollen) Vorzeichenrestriktionen $x \geq 0$ eingeführt. Im allgemeinen Fall führt dies auf das Optimierungsproblem

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0. \end{aligned} \quad (9.13)$$

Definition 9.3 (Lineare Optimierungsprobleme in Standardform). Ein lineares Optimierungsproblem der Form (9.13) nennen wir ein *lineares Optimierungsproblem in Standardform*.

Für die Basislösungen ergibt sich eine neue Situation, da nun nicht mehr jede solche Lösung automatisch zulässig ist. Beispielsweise liefert die Basis $B = (1, 3)$ aufgrund von

$$x_B = \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} = A_B^{-1}b = \begin{bmatrix} 2 & 2 \\ 0 & 1 \end{bmatrix}^{-1} \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} \\ 2 \end{pmatrix}$$

die Basislösung $(-1/2, 0, 2)^\top$, die negative Einträge besitzt.

Definition 9.4 ((Primal) zulässige Basis). Eine Basis B heißt (*primal*) *zulässige Basis* und die zugehörige Basislösung $(x_B, x_N)^\top$ heißt *zulässige Basislösung*, falls $x_B \geq 0$ gilt.

Wie wir bereits im Beispiel weiter vorne gesehen haben, ist die Basis $B = (1, 2)$ zulässig, da die zugehörige Basislösung $\bar{x} = (1, 1, 0)^\top$ nicht-negativ ist. Wir gehen zunächst davon aus, dass wir eine (primal) zulässige Basis B gegeben haben. Wie wir eine solche erhalten, ist Thema eines späteren Abschnitts. Sei \bar{x} die zugehörige zulässige Basislösung. Wie im Beispiel bleiben unsere Überlegungen, die uns zu Gleichung (9.11) führten nach wie vor gültig – nur die Fallunterscheidung bzgl. z_N muss neu überdacht werden. Falls $z_N = 0$, so ist die Zielfunktion wieder konstant auf der Menge aller zulässigen Lösungen und die Basislösung \bar{x} ist insbesondere optimal. Ist $z_N \geq 0$, so gilt wegen $x_N \geq 0$ für jede zulässige Lösung $x = (x_B, x_N)^\top$ des Problems (9.13):

$$c^\top x = c^\top \bar{x} + z_N^\top x_N \geq c^\top \bar{x}. \quad (9.14)$$

Daher folgt aus $z_N \geq 0$ wieder die Optimalität der Basislösung \bar{x} . Ist andererseits $z_j < 0$ für ein mindestens $j \in N$, so bringt nach Gleichung (9.12) jede Erhöhung des Wertes von x_j eine Verbesserung der Zielfunktion. Wir wollen x_j nun maximal erhöhen, so dass der resultierende Vektor noch zulässig ist. Dabei lassen wir die anderen Nicht-Basiseinträge in $N \setminus \{j\}$ auf 0 fixiert. Wegen Gleichung (9.10) gilt

$$\begin{aligned} x_B &= A_B^{-1}b - A_B^{-1}A_N x_N \\ &= A_B^{-1}b - A_B^{-1}A_j x_j - A_B^{-1}A_{N \setminus \{j\}} x_{N \setminus \{j\}} \\ &= \bar{x}_B - A_B^{-1}A_j x_j. \end{aligned} \quad (9.15)$$

Der Einfluss von x_j auf die Basisvariablen x_B wird daher durch den Vektor

$$w = A_B^{-1}A_j \in \mathbb{R}^m \quad (9.16)$$

gesteuert. Gilt $w \leq 0$, so können wir x_j beliebig erhöhen, ohne die Zulässigkeit zu verlieren. Da zusätzlich $z_j < 0$ gilt, wird dadurch auch die Zielfunktion beliebig verringert (vgl. Gleichung (9.11)), d. h. das Optimierungsproblem (9.13) ist nach unten unbeschränkt. Nehmen wir deshalb $w \not\leq 0$ an und betrachten Indizes $i \in \{1, \dots, m\}$ mit $B_i \in B$ und $w_i > 0$. Damit x zulässig bleibt, muss $x_B \geq 0$ bleiben, d. h. wir können x_j maximal soweit erhöhen, bis eine der Variablen aus x_B Null erreicht. Das heißt, es muss

$$x_B = \bar{x}_B - x_j w \geq 0$$

gelten. Für alle $i \in \{1, \dots, m\}$ mit $B_i \in B$ und $w_i > 0$ muss also

$$x_j w_i \leq \bar{x}_{B_i} \iff x_j \leq \frac{\bar{x}_{B_i}}{w_i}$$

gelten. Die maximale Erhöhung γ lässt sich in folgender Form schreiben:

$$\gamma = \min \left\{ \frac{\bar{x}_{B_i}}{w_i} : w_i > 0 \text{ und } i \in \{1, \dots, m\} \right\}. \quad (9.17)$$

Wir betrachten nun den entsprechenden Vektor \bar{x}^+ mit $\bar{x}_j^+ = \gamma$ und $\bar{x}_B^+ = \bar{x} - \gamma w$, den wir erhalten haben. Es gilt nach Konstruktion

$$\begin{aligned} A\bar{x}^+ &= b, \\ \bar{x}^+ &\geq 0, \\ c^\top \bar{x}^+ &= c^\top \bar{x} + \gamma z_j \leq c^\top \bar{x}, \end{aligned} \tag{9.18}$$

wobei in der letzten Ungleichung sogar strikte Ungleichung gilt, falls $\gamma > 0$ gewählt werden konnte, da $z_j < 0$ ist.²

Ist nun \bar{x}^+ eine optimale Lösung unseres Optimierungsproblems (9.13)? Um dies festzustellen bzw. um eine bessere Lösung zu finden, würden wir gerne mit \bar{x}^+ genauso verfahren, wie wir dies bei \bar{x} getan haben; vgl. Gleichung (9.14). Allerdings ist nicht klar, wie dies geschehen soll. Was die Vorgehensweise bei \bar{x} so einfach machte, war, dass \bar{x} eine Basislösung (zur Basis B) war und wir dadurch die Basisvariablen in Abhängigkeit von den Nicht-Basisvariablen ausdrücken konnten; siehe Gleichung (9.10). Dies ermöglichte uns das Umschreiben der Zielfunktion in Gleichung (9.11) und schließlich das Finden von \bar{x}^+ . Im einführenden Beispiel hatten wir einfach die Rollen der beiden Variablen x_1 und x_3 beim Übergang von $\bar{x} = (1, 1, 0)^\top$ zu $\bar{x}^+ = (0, 1/3, 4/3)^\top$ vertauscht, da x_1 in der neuen Lösung \bar{x}^+ den Wert 0 erhielt. Das folgende Lemma zeigt, dass diese Vorgehensweise auch im allgemeinen Fall gilt: Wenn wir die neue Variable x_j in die Basis aufnehmen und eine beliebige Variable x_i aus der Basis entfernen, die in \bar{x}^+ den Wert 0 hat, so haben wir eine neue Basis B^+ und \bar{x}^+ ist die zugehörige Basislösung.

Lemma 9.5. Sei $\text{Rang } A = m$ und $i \in \{1, \dots, m\}$ beliebig mit $B_i \in B$, $w_i > 0$ sowie $\bar{x}_{B_i}^+ = 0$. Sei ferner $B^+ = B \setminus \{B_i\} \cup \{j\}$. Dann ist B^+ eine zulässige Basis von A und der Vektor \bar{x}^+ die zugehörige Basislösung.

Beweis. Sei $N^+ := N \setminus \{j\} \cup \{B_i\}$. Wir müssen folgende Eigenschaften zeigen:

- (i) A_{B^+} ist nicht-singulär,
- (ii) $\bar{x}_{B^+}^+ \geq 0$ und $\bar{x}_{N^+}^+ = 0$,
- (iii) $\bar{x}_{B^+}^+ = A_{B^+}^{-1}b$.

Wir zeigen zunächst (i). Angenommen, A_{B^+} wäre singulär. Da A_B nicht-singulär war und $B^+ \setminus \{j\} = B \setminus \{B_i\}$ gilt, sind die Spalten aus $A_{B^+ \setminus \{j\}}$ linear unabhängig. Daher folgt, dass A_j eine Linearkombination der Spalten aus $A_{B^+ \setminus \{j\}}$ sein muss und es wegen der linearen Unabhängigkeit der Spalten aus $A_{B^+ \setminus \{j\}}$ einen eindeutigen Vektor λ gibt mit $A_{B^+ \setminus \{j\}}\lambda = A_j$. Setzen wir $\bar{\lambda}_i := 0$ und $\bar{\lambda}_k := \lambda_k$ für $k \in \{1, \dots, m\} \setminus \{i\}$ sonst, so haben wir dann

²Wir werden später noch sehen, dass dem Fall $\gamma = 0$ eine besondere Bedeutung innerhalb des Simplex-Verfahrens zukommt.

$A_B \bar{\lambda} = A_j$, also $\bar{\lambda} = A_B^{-1} A_j = w$, wobei w wie in Gleichung (9.16) definiert ist. Dann ist aber $w_i = \bar{\lambda}_i = 0$ im Widerspruch zur Voraussetzung $w_i > 0$.

Als nächstes zeigen wir (ii). Die Aussage $\bar{x}_{N^+}^+ = 0$ ist offensichtlich nach Konstruktion richtig, da nach Voraussetzung $\bar{x}_{B_i}^+ = 0$ gilt und alle Nicht-Basisvariablen in $N \setminus \{j\}$ den Wert 0 behalten.

Wegen $\bar{x}_j^+ = \gamma$ und $\gamma \geq 0$ folgt $\bar{x}_j^+ \geq 0$. Wir müssen also nur noch zeigen, dass $\bar{x}_{B_k}^+ \geq 0$ für alle $k \in \{1, \dots, m\}$ mit $B_k \in B^+ \setminus \{j\} = B \setminus \{B_i\}$. Für diese B_k gilt nach (9.15) und (9.16)

$$\bar{x}_{B_k}^+ = \bar{x}_{B_k} - \gamma w_k.$$

Ist $w_k \leq 0$, dann gilt $\bar{x}_{B_k}^+ \geq 0$, da $\bar{x}_{B_k} \geq 0$. Falls $w_k > 0$ so haben wir nach der Wahl von γ in (9.17) die Ungleichung $\gamma \leq \bar{x}_{B_k}/w_k$ und damit

$$\bar{x}_{B_k}^+ = \bar{x}_{B_k} - \gamma w_k \geq \bar{x}_{B_k} - \frac{\bar{x}_{B_k}}{w_k} w_k = 0. \quad (9.19)$$

Eigenschaft (iii) ergibt sich wie folgt: Nach Konstruktion von \bar{x}^+ in (9.15) gilt $A\bar{x}^+ = b$. Da nach (ii) $\bar{x}_{N^+}^+ = 0$ und nach (i) A_{B^+} regulär ist, folgt $\bar{x}_{B^+}^+ = A_{B^+}^{-1} b$ aus $b = A_{B^+} \bar{x}_{B^+}^+ + A_{N^+} \bar{x}_{N^+}^+ = A_{B^+} \bar{x}_{B^+}^+$. \square

Lemma 9.5 ermöglicht nun eine Fortsetzung des Verfahrens mit B^+ und \bar{x}^+ anstelle von B und \bar{x} . Das resultierende Verfahren ist in Algorithmus 20 formal dargestellt:

Beispiel 32. Als Beispiel betrachten wir folgendes lineare Optimierungsproblem:

$$\begin{aligned} \min \quad & -3x_1 - 2x_2 - 2x_3 \\ \text{s.t.} \quad & x_1 + x_3 \leq 8, \\ & x_1 + x_2 \leq 7, \\ & x_1 + 2x_2 \leq 12, \\ & x_1, x_2, x_3 \geq 0. \end{aligned}$$

Die Menge der zulässigen Lösungen ist in Abbildung 9.1 dargestellt. Wir bringen das LP in Standardform, indem wir sogenannte Schlupfvariablen x_4 , x_5 und x_6 einführen:

$$\begin{aligned} \min \quad & -3x_1 - 2x_2 - 2x_3 \\ \text{s.t.} \quad & x_1 + x_3 + x_4 = 8, \\ & x_1 + x_2 + x_5 = 7, \\ & x_1 + 2x_2 + x_6 = 12, \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0. \end{aligned}$$

Algorithmus 20 Das Simplex-Verfahren für LPs in Standardform

Eingabe: A, b, c und eine primal zulässige Basis $B = (B_1, \dots, B_m)$.

Ausgabe: Eine Optimallösung \bar{x} oder die Meldung, dass das LP unbeschränkt ist.

- 1: BTRAN (Backward Transformation): Löse $\bar{y}^\top A_B = c_B^\top$.
- 2: Pricing: Berechne $z_N = c_N - A_N^\top \bar{y}$.
- 3: **if** $z_N \geq 0$ **then**
- 4: **return** optimale Basis B und Optimallösung $\bar{x} = (\bar{x}_B, \bar{x}_N)^\top$.
- 5: Wähle ein $j \in N$ mit $z_j < 0$.
- 6: FTRAN (Forward Transformation): Löse $A_B w = A_j$.
- 7: **if** $w \leq 0$ **then**
- 8: **return** "Das LP ist unbeschränkt."
- 9: Ratio-Test (Quotiententest): Berechne

$$\gamma = \frac{\bar{x}_{B_i}}{w_i} = \min \left\{ \frac{\bar{x}_{B_k}}{w_k} : w_k > 0 \text{ und } k \in \{1, \dots, m\} \right\}.$$

- 10: Update: Setze

$$\bar{x}_B = \bar{x}_B - \gamma w, \quad N = N \setminus \{j\} \cup \{B_i\}, \quad B_i = j, \quad \bar{x}_j = \gamma$$

und gehe zu Schritt 1.

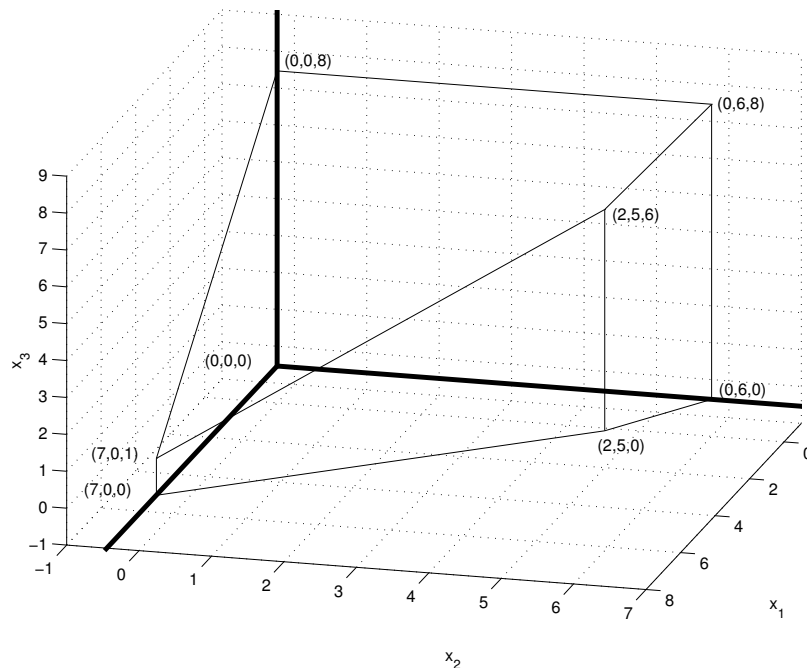


Abbildung 9.1: Graphische Darstellung zu Beispiel 32

Eine primal zulässige Basis ist $B = (4, 5, 6)$ und die zugehörige Basislösung ist gegeben durch

$$\begin{aligned}\bar{x}_B = \begin{pmatrix} \bar{x}_4 \\ \bar{x}_5 \\ \bar{x}_6 \end{pmatrix} &= A_B^{-1}b = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{pmatrix} 8 \\ 7 \\ 12 \end{pmatrix} = \begin{pmatrix} 8 \\ 7 \\ 12 \end{pmatrix}, \\ \bar{x}_N &= \begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix},\end{aligned}$$

d. h., alle Schlupfvariablen sind in der Basis und der Zielfunktionswert ist

$$c^\top \bar{x} = c_B^\top \bar{x}_B = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}^\top \begin{pmatrix} 8 \\ 7 \\ 12 \end{pmatrix} = 0.$$

Die einzelnen Teilschritte des Algorithmus 20 sehen dann wie folgt aus:

1. BTRAN: Wir lösen $\bar{y}^\top A_B = c_B^\top$, d. h.

$$\bar{y}^\top \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = (0, 0, 0) \implies \bar{y}^\top = (0, 0, 0).$$

2. Pricing: Wir berechnen $z_N = c_N - A_N^\top \bar{y}$:

$$z_N = \begin{pmatrix} -3 \\ -2 \\ -2 \end{pmatrix} - \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -3 \\ -2 \\ -2 \end{pmatrix}.$$

Für $j = 1$ gilt $z_j = -3 < 0$.

3. FTRAN: Wir lösen $A_B w = A_j$:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} w = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \implies w = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

4. Ratio-Test: Wir berechnen

$$\begin{aligned}\gamma &= \min \left\{ \frac{\bar{x}_{B_k}}{w_k} : w_k > 0 \text{ und } k \in \{1, \dots, m\} \right\} \\ &= \min \left\{ \frac{8}{1}, \frac{7}{1}, \frac{12}{1} \right\} = 7 = \frac{\bar{x}_5}{w_2} = \frac{\bar{x}_{B_i}}{w_i}\end{aligned}$$

mit $i = 2$ und $B_2 = 5$.

Iteration	Basis	Zielfunktionswert	Eintretende Variable	Verlassende Variable
1	$B = (4, 5, 6)$	0	x_1	x_5
2	$B = (4, 1, 6)$	-21	x_3	x_4
3	$B = (3, 1, 6)$	-23	x_2	x_6
4	$B = (3, 1, 2)$	-28		

Tabelle 9.3: Verlauf des Simplex-Algorithmus bei dem LP aus Beispiel 32

5. **Update:** Wir setzen

$$\begin{aligned}\bar{x}_B &= \begin{pmatrix} \bar{x}_4 \\ \bar{x}_5 \\ \bar{x}_6 \end{pmatrix} - \gamma w = \begin{pmatrix} 8 \\ 7 \\ 12 \end{pmatrix} - 7 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 5 \end{pmatrix}, \\ N &= N \setminus \{j\} \cup \{B_i\} = \{1, 2, 3\} \setminus \{1\} \cup \{5\} = \{2, 3, 5\}, \\ B_2 &= 1, \\ \bar{x}_1 &= \gamma = 7.\end{aligned}$$

Wir erhalten die neue Basislösung \bar{x} zur Basis $B = (4, 1, 6)$ mit

$$\bar{x}_B = \begin{pmatrix} \bar{x}_4 \\ \bar{x}_1 \\ \bar{x}_6 \end{pmatrix} = \begin{pmatrix} 1 \\ 7 \\ 5 \end{pmatrix}, \quad \bar{x}_N = \begin{pmatrix} \bar{x}_2 \\ \bar{x}_3 \\ \bar{x}_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Der Zielfunktionswert verringert sich um $\gamma z_j = 7 \cdot (-3) = -21$, so dass nun $c^\top \bar{x} = -21$ gilt.

Die Fortsetzung des Simplex-Algorithmus liefert die Ergebnisse aus Tabelle 9.3. Nach der dritten Iteration gilt $B = (3, 1, 2)$ und $N = \{4, 5, 6\}$. Die aktuelle Basislösung ist

$$\bar{x}_B = \begin{pmatrix} \bar{x}_3 \\ \bar{x}_1 \\ \bar{x}_2 \end{pmatrix} = \begin{pmatrix} 6 \\ 2 \\ 5 \end{pmatrix}, \quad \bar{x}_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

In der vierten Iteration lauten die einzelnen Teilschritte dann wie folgt:

1. **BTRAN:** Wir lösen $\bar{y}^\top A_B = c_B^\top$:

$$\bar{y}^\top \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix} = (-2, -3, -2) \implies \bar{y} = \begin{pmatrix} -2 \\ 0 \\ -1 \end{pmatrix}.$$

2. **Pricing:** Wir berechnen $z_N = c_N - A_N^\top \bar{y}$:

$$z_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} - \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} -2 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}.$$

Es gilt nun $z_N \geq 0$ und der Algorithmus terminiert daher mit der Optimallösung $x^* = (2, 5, 6, 0, 0, 0)^\top$. \triangle

Satz 9.6. Terminiert Algorithmus 20, so liefert er das korrekte Ergebnis.

Beweis. Der Algorithmus terminiert entweder in Schritt 4 mit einer Lösung oder in Schritt 8 mit der Meldung, dass das Problem unbeschränkt ist. Stoppt das Verfahren in Schritt 4 mit der aktuellen Basislösung \bar{x} , so gilt $z_N \geq 0$. Wie wir bereits mit Gleichung (9.14) gezeigt haben, gilt für jede zulässige Lösung x des linearen Problems die Gleichung (9.13):

$$c^\top x = c^\top \bar{x} + z_N^\top x_N \geq c^\top \bar{x}.$$

Somit ist die Basislösung \bar{x} auch tatsächlich eine optimale Lösung. Falls das Verfahren in Schritt 8 terminiert, so ist der Vektor $w = A_B^{-1}A_j$, der im FTRAN-Schritt berechnet wird, nicht-positiv: $w \leq 0$. Wie wir bei der Herleitung in Gleichung (9.16) gezeigt haben, ist in diesem Fall das Optimierungsproblem (9.13) nach unten unbeschränkt. \square

Der Simplex-Algorithmus 20 liefert beim Abbruch also entweder eine Optimallösung oder die korrekte Information, dass das Optimierungsproblem unbeschränkt ist. Aber terminiert der Algorithmus immer? Dies ist eine der Fragen, mit denen wir uns noch beschäftigen müssen. Insgesamt müssen wir noch folgende Punkte klären:

Terminierung: Bricht der Simplex-Algorithmus immer ab oder kann es vorkommen, dass unendlich viele Iterationen durchgeführt werden?

Initialisierung: Wie finden wir eine erste zulässige Basis? In unserer bisherigen Grundversion des Simplex-Algorithmus sind wir davon ausgegangen, dass zu Beginn bereits eine primal zulässige Basis vorliegt. Im Allgemeinen kann man aber eine zulässige Basis nicht einfach ablesen bzw. es ist nicht klar, ob eine solche Basis überhaupt existiert.

9.3 Die Standardform ist keine Einschränkung

Bisher haben wir uns mit linearen Optimierungsproblemen in der Standardform

$$\begin{array}{ll} \min_x & c^\top x \\ \text{s.t.} & Ax = b, \\ & x \geq 0 \end{array}$$

beschäftigt. Wir zeigen in diesem Abschnitt, dass man jedes “allgemeine lineare Optimierungsproblem” auf diese Standardform bringen kann. Im Prinzip genügt es daher, sich mit Lösungsmethoden für lineare Optimierungsprobleme in Standardform zu beschäftigen. Es sollte aber erwähnt werden, dass man in der Praxis durchaus die Struktur von bestimmten linearen Optimierungsproblemen ausnutzt, um spezielle Formen der Simplex-Methode zu erhalten, die möglichst effizient arbeiten.

9.3.1 Minimieren vs. Maximieren

Da für jede Menge S die Beziehung

$$\min \{c^\top x : x \in S\} = -\max \{-c^\top x : x \in S\}$$

gilt, sind Minimierungs- und Maximierungsprobleme äquivalent. Wir können uns also ohne Einschränkung auf lineare Optimierungsprobleme beschränken, bei denen die Zielfunktion minimiert wird.

9.3.2 Lineare Optimierungsprobleme in allgemeiner Form

Wir betrachten ein lineares Optimierungsproblem in allgemeiner Form, das heißt mit linearen Ungleichungen und Gleichungen sowie vorzeichenbeschränkten und freien Variablen, bei dem die Zielfunktion minimiert wird:

$$\begin{aligned} \min_{x,y} \quad & c^\top x + d^\top y \\ \text{s.t.} \quad & Ax + By = b, \\ & Cx + Dy \leq f, \\ & Ex + Fy \geq e, \\ & x \geq 0. \end{aligned} \tag{9.20}$$

Die Variablen x , bei denen wir Nicht-Negativität $x \geq 0$ fordern, nennt man *vorzeichenbeschränkte Variablen*, die y -Variablen heißen *freie Variablen*.

Schlupf- und Überschussvariablen

Zunächst beschäftigen wir uns mit den Ungleichungen $Cx + Dy \leq f$ und $Ex + Fy \geq e$ und zeigen, wie wir sie durch Einführen von neuen Variablen in Gleichungsrestriktionen überführen können. Für eine Ungleichung der Form

$$C_{i,\cdot}x + D_{i,\cdot}y \leq f_i$$

können wir eine neue vorzeichenbeschränkte Variable $s_i \geq 0$ einführen, und die Ungleichung somit äquivalent als

$$C_{i,\cdot}x + D_{i,\cdot}y + s_i = f_i, \quad s_i \geq 0$$

schreiben. Die Variable s_i nennt man *Schlupfvariable*³. Sie misst den “freien Platz” (oder “Schlupf”) bis zur oberen Schranke f_i . Fassen wir die s_i -Variablen in einem Vektor $s \geq 0$ zusammen, so erhalten wir die Gleichungsrestriktionen

$$Cx + Dy + s = f.$$

Analog können wir für jede Ungleichung

$$E_{i,\cdot}x + F_{i,\cdot}y \geq e_i$$

eine Schlupfvariable $u_i \geq 0$ einführen, so dass wir die Ungleichungen $Ex + Fy \geq e$ äquivalent als

$$Ex + Fy - u = e$$

formulieren können. Manchmal nennt man die u_i -Variablen auch *Überschussvariablen*, da u_i den Überschuss bis zur unteren Schranke in der i -ten Restriktion misst. Mithilfe der Schlupfvariablen erhalten wir folgende äquivalente Form des Optimierungsproblems (9.20):

$$\begin{aligned} \min_{x,y,u,s} \quad & c^\top x + d^\top y \\ \text{s.t.} \quad & Ax + By = b, \\ & Cx + Dy + s = d, \\ & Ex + Fy - u = e, \\ & x, u, s \geq 0. \end{aligned} \tag{9.21}$$

Das Problem (9.21) ist fast in Standardform. Der einzige Unterschied besteht darin, dass die Variablen y freie und keine vorzeichenbeschränkte Variablen sind. Es hat also die Form

$$\begin{aligned} \min_{\hat{x}} \quad & \hat{c}^\top \hat{x} \\ \text{s.t.} \quad & \hat{A}\hat{x} = \hat{b}, \\ & \hat{x}_i \geq 0 \quad \text{für } i \in I, \end{aligned} \tag{9.22}$$

wobei $I \subset \{1, \dots, n\}$ eine echte Teilmenge der Variablen indiziert.

³Engl.: *slack variable*.

Freie Variablen (1. Möglichkeit)

Eine einfache Möglichkeit, eine freie Variable \hat{x}_j zu eliminieren, ist sie durch zwei vorzeichenbeschränkte Variablen $\hat{x}_j^+ \geq 0$ und $\hat{x}_j^- \geq 0$ zu ersetzen. Wir setzen hierfür

$$\hat{x}_j = \hat{x}_j^+ - \hat{x}_j^-. \quad (9.23)$$

Wenn wir überall im linearen Optimierungsproblem (9.22) jedes \hat{x}_j mit $j \in \{1, \dots, n\} \setminus I$ durch $\hat{x}_j^+ - \hat{x}_j^-$ ersetzen, bleibt die lineare Struktur in allen Nebenbedingungen sowie in der Zielfunktion erhalten.

Freie Variablen (2. Möglichkeit)

Bei der oben genannten einfachen Methode zur Elimination von freien Variablen erhöht sich die Anzahl der Variablen – ein Umstand, den man gerne vermeiden möchte, um das Problem “so klein wie möglich” zu halten. Außerdem führt die Transformation (9.23) zu einer gewissen Art von Redundanz: Wenn man sowohl zu \hat{x}_j^+ als auch zu \hat{x}_j^- eine Konstante addiert, ändert sich die Differenz $\hat{x}_j^+ - \hat{x}_j^-$ nicht. Mit anderen Worten: Die Darstellung von \hat{x}_j als Differenz von zwei nicht-negativen Werten ist nicht eindeutig. Dies stört zwar in der Simplex-Methode nicht, ist aber aus theoretischer Sicht etwas “unschön”. Eine zweite elegantere Möglichkeit zur Elimination einer freien Variable \hat{x}_j besteht darin, die Nebenbedingungen $\hat{A}\hat{x} = \hat{b}$ zu benutzen. Wir betrachten dazu die j -te Spalte \hat{A}_j der Matrix \hat{A} . Falls $\hat{A}_j = 0$ ist, also die Variable \hat{x}_j in keiner Nebenbedingung vorkommt, so unterscheiden wir zwei Fälle: $\hat{c}_j = 0$ und $\hat{c}_j \neq 0$. Ist $\hat{c}_j = 0$, so sind nicht nur die Nebenbedingungen von \hat{x}_j unabhängig, sondern auch die Zielfunktion. Wir können daher die Variable \hat{x}_j aus dem linearen Optimierungsproblem (9.22) entfernen und erhalten ein kleineres äquivalentes lineares Optimierungsproblem. Ist hingegen $\hat{c}_j \neq 0$ und besitzt das Problem (9.22) eine zulässige Lösung \bar{x} , so ist wegen $\hat{A}_j = 0$ für jeden Wert $t \in \mathbb{R}$ der Vektor $x(t)$, definiert durch

$$x_k(t) = \begin{cases} \bar{x}_k, & \text{falls } k \neq j, \\ \bar{x}_k + t, & \text{falls } k = j \end{cases}$$

eine zulässige Lösung. Es gilt dann $\hat{c}^\top x(t) = \hat{c}^\top \bar{x} + t\hat{c}_j$. Da $\hat{c}_j \neq 0$ können wir die Zielfunktion durch geeignete Wahl von $t \in \mathbb{R}$ beliebig klein machen. Wir können also schließen, dass das Problem (9.22) keine Optimallösung hat, da es entweder unzulässig oder unbeschränkt ist. Es verbleibt noch der Fall, dass $\hat{A}_j \neq 0$. Dann gibt es aber mindestens eine Gleichung von $\hat{A}\hat{x} = \hat{b}$, in der \hat{x}_j mit einem Koeffizienten ungleich 0 vorkommt, beispielsweise

$$\hat{A}_{k,\cdot}\hat{x} = \hat{b}_k \iff a_{k,1}\hat{x}_1 + a_{k,2}\hat{x}_2 + \dots + a_{k,j}\hat{x}_j + \dots + a_{k,n}\hat{x}_n = \hat{b}_k, \quad (9.24)$$

wobei $a_{k,j} \neq 0$ gilt. Wir können nun (9.24) nach \hat{x}_j auflösen und erhalten

$$\hat{x}_j = \frac{1}{a_{k,j}} \left(\hat{b}_k - \sum_{l \neq j} a_{k,l} \hat{x}_l \right). \quad (9.25)$$

Somit lässt sich \hat{x}_j in jeder Gleichung von $\hat{A}\hat{x} = \hat{b}$ durch die rechte Seite von (9.25) ersetzen. Wir erhalten schließlich ein reduziertes äquivalentes lineares Problem ohne die freie Variable \hat{x}_j , das sogar eine Variable weniger besitzt als das ursprüngliche Problem (9.22). Des Weiteren kann jede Optimallösung des neuen Problems mittels (9.25) zu einer Optimallösung des Originalproblems (9.22) erweitert werden. Es sei letztlich noch erwähnt, dass derartigen Variablensubstitutionen in der Praxis mit Sorgfalt durchgeführt werden müssen um keine numerischen Probleme zu bekommen.

Kapitel 10

Geometrische Aspekte der linearen Optimierung

Bei dem linearen Optimierungsproblem in Beispiel 32 gab es eine Optimallösung, die in einer “Ecke” des zulässigen Bereichs lag. In diesem Kapitel werden wir den Begriff der “Ecke” formalisieren und beweisen, dass diese Beobachtung kein Zufall war.

10.1 Polyeder und Polytope

Wir betrachten das lineare Optimierungsproblem

$$\begin{array}{llll} \min_x & 3x_1 & + & 5x_2 \\ \text{s.t.} & -2x_1 & - & x_2 \leq -3, \quad (1) \\ & -2x_1 & - & 2x_2 \leq -5, \quad (2) \\ & -x_1 & - & 4x_2 \leq -4, \quad (3) \\ & x_1, x_2 & \geq & 0. \end{array} \quad (10.1)$$

In diesem linearen Optimierungsproblem sind alle Nebenbedingungen Ungleichungen. Diese Ungleichungen beschreiben jeweils sogenannte Halbräume, mit denen wir uns jetzt genauer beschäftigen. Sei dazu $\mathbb{K} = \mathbb{R}$ oder $\mathbb{K} = \mathbb{Q}$.

Definition 10.1 (Hyperebene, Halbraum). Für einen Vektor $a \in \mathbb{K}^n$ mit $a \neq 0$ und $\alpha \in \mathbb{K}$ heißt die Menge

$$G = G_{a,\alpha} = \{x \in \mathbb{K}^n : a^\top x = \alpha\}$$

die durch a und α bestimmte *Hyperebene*. Analog nennen wir die Menge

$$H = H_{a,\alpha} = \{x \in \mathbb{K}^n : a^\top x \leq \alpha\}$$

den durch a und α bestimmten *Halbraum*.

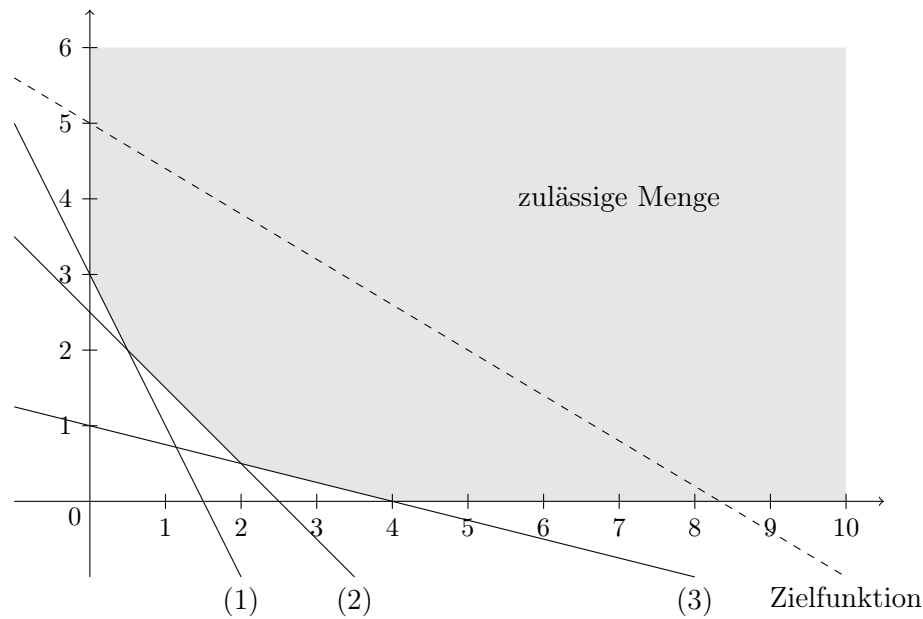


Abbildung 10.1: Illustration der zulässigen Menge des linearen Optimierungsproblems (10.1)

In unserem Beispiel bildet der Schnitt der fünf Halbräume

$$\begin{aligned}
 H\begin{pmatrix} -2 \\ -1 \end{pmatrix}, -3 &= \{x \in \mathbb{R}^2: -2x_1 - x_2 \leq -3\}, \\
 H\begin{pmatrix} -2 \\ -2 \end{pmatrix}, -5 &= \{x \in \mathbb{R}^2: -2x_1 - 2x_2 \leq -5\}, \\
 H\begin{pmatrix} -1 \\ -4 \end{pmatrix}, -4 &= \{x \in \mathbb{R}^2: -x_1 - 4x_2 \leq -4\}, \\
 H\begin{pmatrix} -1 \\ 0 \end{pmatrix}, 0 &= \{x \in \mathbb{R}^2: -x_1 \leq 0\}, \\
 H\begin{pmatrix} 0 \\ -1 \end{pmatrix}, 0 &= \{x \in \mathbb{R}^2: -x_2 \leq 0\},
 \end{aligned}$$

die Menge der zulässigen Lösungen für das lineare Optimierungsproblem (10.1). Den Schnitt von endlich vielen Halbräumen nennen wir einen *Polyeder*. Allerdings stellt es sich als nützlich heraus, auch den ganzen Raum als Polyeder zu bezeichnen. Wir haben also folgende Definition.

Definition 10.2 (Polyeder, Polytop). Wir nennen eine Menge $P \subseteq \mathbb{K}^n$ einen *Polyeder*, falls entweder $P = \mathbb{K}^n$ gilt oder P der Durchschnitt endlich vieler

Halbräume ist. Ein Polyeder P heißt *Polytop*, wenn es beschränkt ist, d. h., falls es eine Zahl $K > 0$ gibt, so dass $P \subseteq \{x \in \mathbb{K}^n : \|x\|_2 \leq K\}$ gilt.

Ist $P \neq \mathbb{K}^n$ ein Polyeder, also

$$P = \bigcap_{i=1}^m \{x \in \mathbb{K}^n : a_i^\top x \leq \alpha_i\}, \quad (10.2)$$

so können wir die Vektoren $a_1^\top, \dots, a_m^\top$ als Zeilen einer $m \times n$ Matrix

$$A = \begin{bmatrix} a_1^\top \\ \vdots \\ a_m^\top \end{bmatrix}$$

auffassen und den Vektor $b = (\alpha_1, \dots, \alpha_m)^\top \in \mathbb{K}^m$ definieren. Damit können wir das Polyeder P aus (10.2) äquivalent als

$$P = P(A, b) = \{x \in \mathbb{K}^n : Ax \leq b\} \quad (10.3)$$

schreiben. Ist $P = \mathbb{K}^n$, so können wir P ebenfalls in der Form (10.3) schreiben, indem wir die $1 \times n$ -Nullmatrix und den Nullvektor $0 \in \mathbb{K}^1$ benutzen:

$$\mathbb{K}^n = \{x \in \mathbb{K}^n : 0^\top x \leq 0\}.$$

Daher ist jedes Polyeder in der Form (10.3) darstellbar. Umgekehrt ist natürlich auch jede Menge der Form (10.3) ein Polyeder, da

$$P(A, b) = \bigcap_{\{i : A_{i,\cdot} \neq 0\}} \{x \in \mathbb{K}^n : A_{i,\cdot} x \leq b_i\}$$

gilt, wobei $A_{i,\cdot}$ die i -te Zeile der Matrix A bezeichnet. Damit erhalten wir das folgende grundlegende Ergebnis.

Satz 10.3. Eine Menge $P \subseteq \mathbb{K}^n$ ist genau dann ein Polyeder, wenn es eine $m \times n$ Matrix A und einen Vektor $b \in \mathbb{K}^m$ gibt, so dass

$$P = P(A, b) = \{x \in \mathbb{K}^n : Ax \leq b\} \quad (10.4)$$

gilt.

Wir schreiben im Folgenden kurz $P(A, b)$, um das in Gleichung (10.4) über eine Matrix A und einen Vektor b definierte Polyeder zu bezeichnen. Man beachte, dass es für ein Polyeder P immer mehr als eine Matrix A und einen Vektor b gibt, so dass $P = P(A, b)$ gilt. So sind für jeden Skalar $\lambda > 0$ die Polyeder $P(A, b)$ und $P(\lambda A, \lambda b)$ identisch.¹ Darüber hinaus sind auch die Dimensionen von A und b nicht eindeutig.²

¹Warum?

²Warum?

10.2 Konvexität, Extremalmengen und Ecken von Polyedern

Wir haben bereits in Abschnitt 1 den Begriff einer konvexen Menge spezifiziert (siehe Definition 1.2). Darauf aufbauend werden wir nun die zulässigen Mengen linearer Optimierungsprobleme beschreiben.

Jeder Halbraum $H = H_{a,\alpha} = \{x \in \mathbb{K}^n : a^\top x \leq \alpha\}$ ist konvex, da für $x, y \in H$ und $0 \leq \lambda \leq 1$ die Ungleichung

$$a^\top (\lambda x + (1 - \lambda)y) = \lambda a^\top x + (1 - \lambda)a^\top y \leq \lambda \alpha + (1 - \lambda)\alpha = \alpha$$

gilt.

Lemma 10.4. Der Schnitt beliebig vieler konvexer Mengen ist eine konvexe Menge.

Beweis. Seien K_i mit $i \in I$ konvex und $K = \bigcap_{i \in I} K_i$. Für $x, y \in K$ gilt also $x, y \in K_i$ für alle $i \in I$. Somit ist dann für $0 \leq \lambda \leq 1$ wegen der Konvexität jeder Menge K_i auch $z = \lambda x + (1 - \lambda)y \in K_i$ für alle i . Also gilt $z \in K$. \square

Korollar 10.5. Jedes Polyeder ist eine konvexe Menge.

Beweis. Folgt aus Lemma 10.4, da jedes Polyeder entweder gleich \mathbb{K}^n oder der Schnitt endlich vieler Halbräume ist, die konvex sind. \square

Beispiel 33. Sei A eine $m \times n$ Matrix und die Menge K definiert durch

$$\begin{aligned} K &= \{z : \text{es gibt ein } x \in \mathbb{R}^n \text{ mit } x \geq 0 \text{ und } Ax = z\} \\ &= \{Ax : x \in \mathbb{R}^n, x \geq 0\} \subseteq \mathbb{R}^m. \end{aligned}$$

Dann ist K konvex. Sind nämlich $z_1 = Ax_1 \in K$ und $z_2 = Ax_2 \in K$ mit $x_1, x_2 \geq 0$ und $0 \leq \lambda \leq 1$, so haben wir

$$\lambda x_1 + (1 - \lambda)x_2 \geq 0$$

und

$$A(\lambda x_1 + (1 - \lambda)x_2) = \lambda Ax_1 + (1 - \lambda)Ax_2 = \lambda z_1 + (1 - \lambda)z_2,$$

also $\lambda z_1 + (1 - \lambda)z_2 \in K$. \triangle

Definition 10.6 (Extremalmenge, Extrempunkt). Eine konvexe Teilmenge E einer konvexen Menge $K \subseteq \mathbb{K}^n$ heißt *Extremalmenge* von K , falls aus $z \in E$ mit $z = \lambda x + (1 - \lambda)y$ und $0 < \lambda < 1$, $\lambda \in \mathbb{K}$, für $x, y \in K$, folgt, dass $x, y \in E$. Ein *Extrempunkt* (*Ecke*) ist eine einelementige Extremalmenge.

Ein Extrempunkt z eines Polyeders ist also ein Punkt $z \in P$, so dass aus $z = \lambda x + (1 - \lambda)y$ mit $x, y \in P$ und $0 < \lambda < 1$ folgt, dass $x = y = z$. Ein Extrempunkt eines Polyeders P lässt sich also nicht als echte Konvexkombination verschiedener Punkte $x, y \in P$ schreiben.

Satz 10.7. Sei $P \subseteq \mathbb{K}^n$ ein Polyeder. Falls das lineare Optimierungsproblem

$$\min \{c^\top x : x \in P\}$$

Optimallösungen hat, so ist die Menge der Optimallösungen eine Extrempunkte von P .

Beweis. Falls das lineare Optimierungsproblem eine Optimallösung hat, so existiert

$$\gamma = \min \{c^\top x : x \in P\},$$

d. h., es gibt ein $x^* \in P$ mit $c^\top x^* = \gamma$. Die Menge der Optimallösungen ist dann

$$E = \{x \in P : c^\top x = \gamma\} = P \cap \{x \in \mathbb{K}^n : -c^\top x \leq -\gamma\} \cap \{x \in \mathbb{K}^n : c^\top x \leq \gamma\}.$$

Wir haben also E als Schnitt von P und von Halbräumen dargestellt. Somit ist E wieder ein Polyeder und damit konvex. Sei jetzt $z \in E$. Wir nehmen an, dass es $x, y \in P$ gibt, die nicht beide in E liegen, und $0 < \lambda < 1$ mit $z = \lambda x + (1 - \lambda)y$. Ohne Beschränkung der Allgemeinheit sei $x \notin E$, also $c^\top x > \gamma$. Dann folgt mit $c^\top y \geq \gamma$

$$c^\top z = \lambda c^\top x + (1 - \lambda)c^\top y > \lambda\gamma + (1 - \lambda)\gamma = \gamma,$$

was im Widerspruch zu $z \in E$ steht. Also muss $x, y \in E$ gelten. \square

10.3 Basislösungen und Optimierung

Wir betrachten das lineare Optimierungsproblem in Standardform

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0. \end{aligned} \tag{10.5}$$

Wir nehmen dabei zunächst an, dass die Matrix $A \in \mathbb{K}^{m \times n}$ vollen Zeilenrang besitzt, also $\text{Rang } A = m$ gilt. Es besteht ein wichtiger Zusammenhang zwischen den im letzten Abschnitt definierten Extrempunkten eines Polyeders und Basislösungen aus Definition 9.2.

Lemma 10.8. Sei $P = \{x \in \mathbb{K}^n : Ax = b, x \geq 0\}$ das Polyeder der zulässigen Lösungen des linearen Optimierungsproblems (10.5) und $\text{Rang } A = m$. Dann ist x genau dann ein Extrempunkt von P , wenn x eine Basislösung von (10.5) ist.

Beweis. Wir starten mit der Hinrichtung. Sei dazu x ein Extrempunkt von P und $I(x) = \{i : x_i > 0\}$ die Menge der Indizes i , für die x_i strikt positiv ist. Falls die Vektoren $\{A_i : i \in I(x)\}$ linear unabhängig sind, dann können wir wegen $\text{Rang } A = m$ noch $m - |I(x)|$ weitere Spaltenindizes I' finden, so dass $B = I(x) \cup I'$ eine Basis von A ist. Da $A_B x_B = b$ und A_B nicht-singulär ist, ist x die eindeutige zu B gehörende Basislösung.

Falls die Vektoren $\{A_i : i \in I(x)\}$ linear abhängig sind, dann finden wir Skalare $\lambda_i \in \mathbb{K}$ mit $i \in I(x)$, die nicht alle 0 sind, so dass $\sum_{i \in I(x)} \lambda_i A_i = 0$ gilt. Wir definieren jetzt einen Vektor $y \in \mathbb{K}^n$ durch

$$y_i := \begin{cases} \lambda_i, & \text{falls } i \in I(x), \\ 0, & \text{sonst.} \end{cases}$$

Dann gilt nach Konstruktion

$$Ay = \sum_{i=1}^n A_i y_i = \sum_{i \in I(x)} A_i \lambda_i = 0.$$

Wir betrachten für $\delta \in \mathbb{K}$ den Vektor $x + \delta y$, für den gilt:

$$x_i + \delta y_i = \begin{cases} x_i + \delta \lambda_i, & \text{falls } i \in I(x), \\ 0, & \text{sonst.} \end{cases} \quad (10.6)$$

Dann gilt $A(x + \delta y) = Ax + \delta Ay = Ax = b$ für alle $\delta \in \mathbb{K}$. Da $x_i > 0$ für $i \in I(x)$ finden wir $\bar{\delta} \neq 0$, so dass

$$x + \bar{\delta} y \geq 0 \quad \text{und} \quad x - \bar{\delta} y \geq 0$$

gilt. Also sind $x + \bar{\delta} y \in P$ und $x - \bar{\delta} y \in P$ wegen $y \neq 0$ zwei von x verschiedene Vektoren mit

$$x = \frac{1}{2}(x + \bar{\delta} y) + \frac{1}{2}(x - \bar{\delta} y),$$

was im Widerspruch zur Voraussetzung steht, dass x ein Extrempunkt von P ist.

Es bleibt noch die Rückrichtung zu zeigen. Sei dazu $x = (x_B, x_N)^\top$ Basislösung zur Basis B . Wir nehmen an, dass $x = \lambda y + (1 - \lambda)z$ mit $0 < \lambda < 1$ und $y, z \in P$ gilt. Für $j \in N$ gilt

$$0 = x_j = \lambda y_j + (1 - \lambda)z_j.$$

Da sowohl $y_j \geq 0$ als auch $z_j \geq 0$ ist, folgt $y_N = z_N = 0$. Dann muss aber wegen

$$b = Ay = A_B y_B + A_N y_N = A_B y_B$$

und der Nichtsingularität von A_B auch $y_B = A_B^{-1}b = x_B$ gelten. Somit ist $y = x$. Analog folgt $z = x$, also ist x Extrempunkt von P . \square

Satz 10.9 (Fundamentalsatz der linearen Optimierung I). Für das lineare Optimierungsproblem in Standardform (10.5) mit $\text{Rang } A = m$ gelten folgende Aussagen:

- (i) Falls (10.5) eine zulässige Lösung besitzt, dann hat das lineare Optimierungsproblem auch eine zulässige Basislösung.
- (ii) Falls (10.5) eine optimale Lösung besitzt, dann hat das lineare Optimierungsproblem auch eine optimale Basislösung, d. h., eine optimale Lösung die auch Basislösung ist.

Beweis.

- (i) Der Beweis verläuft ähnlich wie der Beweis von Lemma 10.8. Sei x eine zulässige Lösung von (10.5) und $I(x) = \{i: x_i > 0\}$. Falls nun die Vektoren $\{A_i: i \in I(x)\}$ linear unabhängig sind, so können wir wie im Beweis von Lemma 10.8 schließen, dass x eine Basislösung ist. Falls die Vektoren $\{A_i: i \in I(x)\}$ nicht linear unabhängig sind, dann finden wir Skalare $\lambda_i \in \mathbb{K}$ mit $i \in I(x)$, die nicht alle 0 sind, so dass $\sum_{i \in I(x)} \lambda_i A_i = 0$ ist. Wir können o. B. d. A. annehmen, dass mindestens ein Wert λ_i strikt größer als 0 ist.³ Wir betrachten für $\delta \in \mathbb{K}$ den Vektor $x + \delta y$ aus (10.6). Wieder gilt $A(x + \delta y) = b$ für alle $\delta \in \mathbb{K}$. Sei

$$\bar{\delta} = \min \left\{ \frac{x_i}{y_i} : i \in I(x) \text{ und } y_i > 0 \right\}. \quad (10.7)$$

Dann gilt $x - \bar{\delta}y \geq 0$ und $x - \bar{\delta}y$ hat höchstens $|I(x)| - 1$ positive Komponenten. Wir setzen $x \leftarrow x - \bar{\delta}y$. Wenn wir diesen Prozess fortsetzen, so gelangen wir nach höchstens n Schritten in die Situation, dass die Vektoren $\{A_i: i \in I(x)\}$ linear unabhängig sind und erhalten wie oben eine zulässige Basislösung.

- (ii) Sei nun x eine Optimallösung von (10.5). Falls die Vektoren $\{A_i: i \in I(x)\}$ linear unabhängig sind, so ist x wie oben bereits eine Basislösung und es ist nichts mehr zu zeigen. Im zweiten Fall ist $x + \delta y$ für alle hinreichend kleinen $|\delta|$ zulässig für (10.5). Wir haben also

$$c^\top(x + \delta y) = c^\top x + \delta \sum_{i \in I(x)} c_i y_i =: c^\top x + \delta t. \quad (10.8)$$

³Wir können notfalls alle Werte λ_i mit -1 multiplizieren.

Es folgt $t = 0$, da ansonsten entweder $c^\top(x + \delta y) < c^\top x$ oder $c^\top(x - \delta y) < c^\top x$ für ein hinreichend kleines $\delta > 0$ wäre. Damit ist dann aber der Vektor $x + \bar{\delta}y$ mit $\bar{\delta}$ aus Gleichung (10.7) wegen $c^\top(x + \bar{\delta}y) = c^\top x$ ebenfalls eine Optimallösung von (10.5), die weniger positive Komponenten als x hat. Wir können also wie in (i) verfahren und gelangen nach höchstens n Schritten zu einer optimalen Lösung, die auch Basislösung ist. \square

Aus dem obigen Fundamentalsatz und der Charakterisierung von Extrempunkten eines Polyeders in Lemma 10.8 ergibt sich nun unmittelbar:

Korollar 10.10. Falls das lineare Optimierungsproblem in Standardform (10.5) mit $\text{Rang } A = m$ eine Optimallösung hat, so hat es auch eine Optimallösung, die Ecke des Polyeders $P = \{x \in \mathbb{K}^m : Ax = b, x \geq 0\}$ ist.

Kapitel 11

Dualität

11.1 Untere Schranken für optimale Zielfunktionswerte eines linearen Problems

Versetzen wir uns noch in die Lage unseres Austauschstudenten aus Kapitel 9, der auf der Studentenparty seinen Alkoholkonsum minimieren möchte. Wie wir bereits gesehen haben, ließ sich dies durch das lineare Optimierungsproblem

$$\min_x \quad 12x_1 + 2x_2 + 5x_3 \quad (11.1a)$$

$$\text{s.t.} \quad 2x_1 + x_2 + 2x_3 = 3, \quad (11.1b)$$

$$2x_2 + x_3 = 2, \quad (11.1c)$$

$$x_1, x_2, x_3 \geq 0, \quad (11.1d)$$

modellieren, wobei wir hier im Gegensatz zu Kapitel 9 andere Werte $c_1 = 12$, $c_2 = 2$ und $c_3 = 5$ für die Zielfunktionskoeffizienten nutzen, um uns die folgenden motivierenden Rechnungen zu vereinfachen. Nehmen wir an, dass der Student vorerst gar nicht die wirkliche Optimallösung von Problem (11.1) bestimmen möchte, sondern zunächst abschätzen möchte, wie viel Alkohol er mindestens zu sich nehmen wird, wenn er dem Ergebnis des LPs (11.1) folgt. Mit anderen Worten, er möchte zunächst eine *untere Schranke* für den optimalen Zielfunktionswert z^* von Problem (11.1) bestimmen.

Wenn wir die Gleichung (11.1c) mit $1/2$ multiplizieren und zu Gleichung (11.1b) addieren, so erhalten wir

$$1 \cdot (2x_1 + x_2 + 2x_3) + \frac{1}{2} \cdot (2x_2 + x_3) = 1 \cdot 3 + \frac{1}{2} \cdot 2 = 4.$$

Somit gilt für jede zulässige Lösung $x = (x_1, x_2, x_3)^\top$ von Problem (11.1) die Gleichung

$$2x_1 + 2x_2 + \frac{5}{2}x_3 = 4. \quad (11.2)$$

Wir betrachten nun die Koeffizienten vor den Variablen in der Zielfunktion (11.1a). Da $2 \leq 12 = c_1$, $2 \leq 2 = c_2$, $5/2 \leq 5 = c_3$ und $x \geq 0$ gilt, folgt aus Gleichung (11.2), dass für jede zulässige Lösung

$$\begin{aligned} c^\top x &= c_1 x_1 + c_2 x_2 + c_3 x_3 \\ &= 12x_1 + 2x_2 + 5x_3 \\ &\geq 2x_1 + 2x_2 + \frac{5}{2}x_3 \\ &= 4 \end{aligned}$$

gilt. Wir haben damit gezeigt, dass $c^\top x \geq 4$ für jede zulässige Lösung x von Problem (11.1) ist. Insbesondere gilt dies natürlich auch für die Optimallösung x^* von (11.1), so dass unser Austauschstudent durch diese kurze Rechnung sicher sein kann, dass er mindestens 4 Alkoholeinheiten zu sich nehmen wird.

Können wir eine noch bessere untere Schranke herleiten? Wenn wir Gleichung (11.1b) mit 3 multiplizieren, Gleichung (11.1c) mit -2 multiplizieren und die beiden erhaltenen Gleichungen wieder addieren, so erhalten wir

$$3 \cdot (2x_1 + x_2 + 2x_3) - 2 \cdot (2x_2 + x_3) = 3 \cdot 3 - 2 \cdot 2 = 5.$$

Also folgt

$$6x_1 - x_2 + 4x_3 = 5 \quad (11.3)$$

für jede zulässige Lösung x von Problem (11.1). Wir haben wieder die Situation, dass die Koeffizienten vor den Variablen in Gleichung (11.3) kleiner oder gleich den entsprechenden Koeffizienten in der Zielfunktion von Problem (11.1) sind: $6 \leq 12 = c_1$, $-1 \leq 2 = c_2$ und $4 \leq 5 = c_3$. Wie oben können wir daher schließen, dass für jede zulässige Lösung x von Problem (11.1)

$$c^\top x = 12x_1 + 2x_2 + 5x_3 \geq 6x_1 - x_2 + 4x_3 = 5$$

gilt. Der Student kann durch die zweite Rechnung also sogar beweisen, dass er mindestens 5 Alkoholeinheiten zu sich nehmen wird. Wir systematisieren nun diesen ad-hoc-Ansatz zur Berechnung unterer Schranken an den optimalen Zielfunktionswert eines linearen Optimierungsproblems.

Im Prinzip haben wir Linearkombinationen der Nebenbedingungen des linearen Optimierungsproblems betrachtet: Wir haben die erste Gleichung (11.1b) mit einem Wert $y_1 \in \mathbb{R}$ multipliziert, die zweite Gleichung (11.1c) mit einem Wert $y_2 \in \mathbb{R}$ multipliziert und schließlich beide addiert. Dies ergibt dann im allgemeinen Fall die Gleichung

$$y_1(2x_1 + x_2 + 2x_3) + y_2(2x_2 + x_3) = 3y_1 + 2y_2.$$

Nach Umsortieren der linken Seite erhalten wir dann

$$(2y_1 + 0y_2)x_1 + (y_1 + 2y_2)x_2 + (2y_1 + y_2)x_3 = 3y_1 + 2y_2. \quad (11.4)$$

Damit wir den Zielfunktionswert jeder zulässigen Lösung x von Problem (11.1) durch die linke Seite der obigen Gleichung nach unten abschätzen können, müssen die Koeffizienten auf der linken Seite von Gleichung (11.4) kleiner oder gleich den Koeffizienten in der Zielfunktion sein. Das heißt, es muss

$$\begin{aligned} 2y_1 + 0y_2 &\leq c_1, \\ y_1 + 2y_2 &\leq c_2, \\ 2y_1 + y_2 &\leq c_3 \end{aligned} \tag{11.5}$$

gelten.

Haben wir also Werte $y_1, y_2 \in \mathbb{R}$ gefunden, welche die Ungleichungen (11.5) erfüllen, so ist nach Gleichung (11.4) durch $3y_1 + 2y_2$ eine untere Schranke für den optimalen Zielfunktionswert z^* von Problem (11.1) gegeben. Bei unserem ersten Versuch hatten wir $y_1 = 1$ und $y_2 = 1/2$, was die untere Schranke $3 \cdot 1 + 2 \cdot 1/2 = 4$ ergab. Im zweiten Versuch fanden wir mit $y_1 = 3$ und $y_2 = -2$ die bessere Schranke $3 \cdot 3 + 2 \cdot (-2) = 5$. Wir können nun das Problem, eine größtmögliche untere Schranke mit unserem Linearkombinationsansatz zu finden, als Optimierungsproblem formulieren: Wir wollen $y_1, y_2 \in \mathbb{R}$ finden, so dass $3y_1 + 2y_2$ möglichst groß ist und die Ungleichungen (11.5) erfüllt sind:

$$\begin{aligned} \max_y \quad & 3y_1 + 2y_2 \\ \text{s.t.} \quad & 2y_1 + 0y_2 \leq 12, \\ & y_1 + 2y_2 \leq 2, \\ & 2y_1 + y_2 \leq 5. \end{aligned}$$

Dieses Problem ist wieder ein lineares Optimierungsproblem!

11.2 Das duale Problem

Wir betrachten nun ein allgemeines lineares Optimierungsproblem in Standardform

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0 \end{aligned} \tag{11.6}$$

mit $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ und $c \in \mathbb{R}^n$. In unserem Ansatz zur Bestimmung unterer Schranken im letzten Abschnitt haben wir Linearkombinationen der Gleichungen in (11.6) gebildet. Wir bezeichnen für $i = 1, \dots, m$ mit y_i den Multiplikator für die i -te Gleichung $A_{i,\cdot}x = b_i$ und setzen $y = (y_1, \dots, y_m)^\top$.

Dann ergibt die Linearkombination mit den Koeffizienten y_1, \dots, y_m :

$$\begin{aligned}
& \sum_{i=1}^m y_i A_{i,\cdot} x = \sum_{i=1}^m y_i b_i \\
\iff & \left(\sum_{i=1}^m y_i A_{i,\cdot} \right) x = b^\top y \\
\iff & (y^\top A) x = b^\top y \\
\iff & (A^\top y)^\top x = b^\top y.
\end{aligned} \tag{11.7}$$

Die Gleichung $(A^\top y)^\top x = b^\top y$ gilt also für alle für Problem (11.6) zulässigen x . Damit wir $b^\top y$ als untere Schranke für $c^\top x$ verwenden können, muss der Vektor $A^\top y$ komponentenweise kleiner oder gleich dem Vektor c sein. Es muss also

$$A^\top y \leq c \tag{11.8}$$

gelten. Haben wir einen Vektor y , der die Ungleichung (11.8) erfüllt, so folgt für jede zulässige Lösung x von Gleichung (11.6) wegen $x \geq 0$:

$$c^\top x \geq (A^\top y)^\top x = y^\top A x = y^\top b = b^\top y. \tag{11.9}$$

Die bestmögliche untere Schranke erhalten wir schließlich, wenn wir y derart wählen, dass Ungleichung (11.8) gilt und $b^\top y$ möglichst groß wird, d. h., indem wir das lineare Optimierungsproblem

$$\begin{aligned}
& \max_y \quad b^\top y \\
& \text{s.t.} \quad A^\top y \leq c
\end{aligned} \tag{11.10}$$

lösen. Das lineare Optimierungsproblem (11.10) wird *duales Optimierungsproblem* zum *primalen Problem* (11.6) genannt. Unsere Herleitung von Problem (11.10) zeigt das folgende wichtige Ergebnis.

Lemma 11.1 (Schwacher Dualitätssatz). Ist x eine zulässige Lösung des primalen Optimierungsproblems (11.6) und y eine zulässige Lösung des dualen Optimierungsproblems (11.10), so gilt

$$b^\top y \leq c^\top x.$$

Beweis. Siehe Gleichung (11.9). □

Aus dem schwachen Dualitätssatz ergibt sich sofort folgende Konsequenz:

Korollar 11.2. Ist x zulässig für das primale Optimierungsproblem (11.6), y zulässig für das duale Optimierungsproblem (11.10) und gilt $c^\top x = b^\top y$, so sind x und y optimal für die entsprechenden Probleme.

11.3 Das Farkas-Lemma

Bevor wir das Farkas-Lemma zeigen, beweisen wir zunächst den folgenden Alternativsatz.

Satz 11.3. Es sei $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$. Dann hat genau eines der beiden Systeme

$$Ax = b \quad (11.11)$$

und

$$\begin{aligned} A^\top y &= 0, \\ b^\top y &\neq 0 \end{aligned} \quad (11.12)$$

eine Lösung.

Beweis. Man sieht leicht ein, dass nicht beide Systeme eine Lösung haben können. Ist nämlich $Ax = b$ und $A^\top y = 0$, so folgt nämlich der Widerspruch

$$b^\top y = (Ax)^\top y = x^\top (A^\top y) = x^\top 0 = 0.$$

Aus der linearen Algebra ist bekannt, dass der Nullraum $N(A^\top) = \{y \in \mathbb{R}^m : A^\top y = 0\}$ von A^\top und der Bildraum $R(A) = \{Ax : x \in \mathbb{R}^n\}$ orthogonale Komplemente sind. Daher können wir b eindeutig zerlegen in $b = b_1 + b_2$ mit $b_1 \in N(A^\top)$ und $b_2 \in R(A)$. Wir nehmen nun an, dass das System (11.11) nicht lösbar ist. Das heißt, es gilt $b \notin R(A)$ und für obige Zerlegung gilt $b_1 \neq 0$ mit $A^\top b_1 = 0$, da $b_1 \in N(A^\top)$. Außerdem gilt $b_1^\top b = b_1^\top b_1 + b_1^\top b_2 = \|b_1\|_2^2 > 0$. Also haben wir mit b_1 eine Lösung des Systems (11.12) gefunden. \square

Das Ziel ist es jetzt, vergleichbare Resultate für Systeme von linearen Ungleichungen zu beweisen.

Lemma 11.4 (Farkas-Lemma). Es sei $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$. Dann hat genau eines der beiden Systeme

$$\begin{aligned} Ax &= b, \\ x &\geq 0 \end{aligned} \quad (11.13)$$

und

$$\begin{aligned} A^\top y &\geq 0, \\ b^\top y &< 0 \end{aligned} \quad (11.14)$$

eine Lösung.

Beweis. Offensichtlich können nicht beide Systeme (11.13) und (11.14) eine Lösung haben, denn andernfalls wäre

$$0 > b^\top y = (Ax)^\top y = x^\top (A^\top y) \geq 0.$$

Es bleibt also zu zeigen, dass immer mindestens eines der beiden Systeme eine Lösung besitzt. Wir nehmen an, das System (11.13) ist nicht lösbar, d. h., es gibt kein x mit $Ax = b$ und $x \geq 0$. Wir bemerken zunächst, dass wir o. B. d. A. annehmen können, dass in der Matrix A keine Spalte mehr als einmal vorkommt. Wir nennen nun eine Spalte $A_{\cdot,j}$ *komplementär*, wenn auch $-A_{\cdot,j}$ eine Spalte von A ist und beweisen die Behauptung durch Induktion nach der Anzahl k der nicht-komplementären Spalten.

Ist $k = 0$, so können wir die jeweils komplementären Spalten zusammenfassen und erhalten ein äquivalentes System $\tilde{A}\tilde{x} = b$, in dem \tilde{x} keine Vorzeichenrestriktionen hat.¹ Nach Annahme ist dieses System unlösbar und damit gibt es nach Satz 11.3 ein y mit $\tilde{A}^\top y = 0$ und $b^\top y = \alpha \neq 0$. Führt man nun \tilde{A}^\top wieder die komplementären Zeilen ein, so erhalten wir eine Lösung des Systems $A^\top y = 0$ und $b^\top y = \alpha \neq 0$. Wenn $\alpha < 0$ ist, so haben wir eine Lösung des Systems (11.14) gefunden. Ist $\alpha > 0$, so löst $-y$ das System (11.14).

Wir nehmen nun an, dass die Aussage für alle Systeme mit höchstens $k \geq 0$ nicht-komplementären Spalten bereits bewiesen und A eine Matrix mit $k + 1$ nicht-komplementären Spalten ist. Ohne Beschränkung der Allgemeinheit sei die letzte Spalte $A_{\cdot,n}$ nicht-komplementär. Wir setzen $J = \{1, \dots, n-1\}$. Nach Konstruktion hat dann die Matrix $A_{\cdot,J}$ genau k nicht-komplementäre Spalten. Dann ist das System $A_{\cdot,J}x = b$, $x \geq 0$ nicht lösbar, da sich jede Lösung dieses Systems mit $x_n = 0$ zu einer Lösung des Systems (11.13) ergänzen lässt. Daher existiert nach Induktionsvoraussetzung ein y mit $A_{\cdot,J}^\top y \geq 0$ (also $A_{\cdot,j}^\top y \geq 0$ für alle $j \in J$) und $b^\top y < 0$. Wenn auch $A_{\cdot,n}^\top y \geq 0$ gilt, dann erfüllt y die Bedingungen $A^\top y \geq 0$ und $b^\top y < 0$, ist also die geforderte Lösung für das System (11.14), und wir sind fertig. Andernfalls ist y eine Lösung von

$$\begin{aligned} (A_{\cdot,1}, \dots, A_{\cdot,n-1}, -A_{\cdot,n})^\top y &\geq 0 \\ b^\top y &< 0. \end{aligned} \tag{11.15}$$

Wir betrachten nun die Matrix

$$\bar{A} = [A_{\cdot,1}, \dots, A_{\cdot,n-1}, A_{\cdot,n}, -A_{\cdot,n}].$$

Nach Konstruktion hat \bar{A} höchstens k nicht-komplementäre Spalten. Nach Induktionsvoraussetzung tritt genau einer der beiden folgenden Fälle ein:

1. Es gibt ein $\bar{x} \in \mathbb{R}^{n+1}$ mit $\bar{A}\bar{x} = b$ und $\bar{x} \geq 0$.

¹Warum?

2. Es gibt ein $\bar{y} \in \mathbb{R}^m$, so dass $\bar{A}^\top \bar{y} \geq 0$ und $b^\top \bar{y} < 0$ gilt.

Im zweiten Fall gilt $A_{\cdot,j}^\top \bar{y} \geq 0$ für $j = 1, \dots, n-1$ sowie $A_{\cdot,n}^\top \bar{y} \geq 0$ und $-A_{\cdot,n}^\top \bar{y} \geq 0$, also $A_{\cdot,n}^\top \bar{y} = 0$. Daher ist \bar{y} eine Lösung des Systems (11.14) und wir sind fertig.

Es verbleibt also nur noch der erste Fall. Wir definieren den Vektor $x \in \mathbb{R}^n$ durch

$$x_i = \begin{cases} \bar{x}_i, & \text{für } i = 1, \dots, n-1, \\ \bar{x}_n - \bar{x}_{n+1}, & \text{für } i = n. \end{cases}$$

Dann gilt $Ax = b$. Es kann nicht $x \geq 0$ gelten, da wir sonst eine Lösung des Systems (11.13) gefunden hätten (im Widerspruch zu unserer Annahme). Wegen $\bar{x} \geq 0$ muss $x_n < 0$, also $-x_n > 0$ gelten. Für die Lösung y aus Gleichung (11.15) folgt dann

$$\begin{aligned} 0 &> b^\top y = y^\top b \\ &= y^\top Ax = y^\top [A_{\cdot,1}, \dots, A_{\cdot,n-1}, A_{\cdot,n}](x_1, \dots, x_n)^\top \\ &= y^\top [A_{\cdot,1}, \dots, A_{\cdot,n-1}, -A_{\cdot,n}](x_1, \dots, x_{n-1}, -x_n)^\top \\ &= \underbrace{(x_1, \dots, x_{n-1})}_{\geq 0} \underbrace{(-x_n)}_{> 0} [A_{\cdot,1}, \dots, A_{\cdot,n-1}, -A_{\cdot,n}]^\top y, \\ &\geq 0, \end{aligned}$$

also ein Widerspruch. \square

Definition 11.5 (Kegel, Konvexer Kegel). Eine Teilmenge C eines \mathbb{K} -Vektorraums heißt *Kegel*, wenn mit $x \in C$ auch $\alpha x \in C$ für $\alpha \geq 0$. Gilt für beliebige $x, y \in C$ auch $x + y \in C$, dann ist C ein *konvexer Kegel*.

Das Farkas-Lemma hat auch eine geometrische Interpretation:

Entweder liegt b im konvexen Kegel

$$\left\{ \sum_{j=1}^n A_{\cdot,j} x_j \in \mathbb{R}^m : x_j \geq 0 \ \forall j \in \{1, \dots, n\} \right\},$$

der von den Spalten $A_{\cdot,1}, \dots, A_{\cdot,n}$ der Matrix A aufgespannt wird, oder es gibt eine Hyperebene, die b von eben diesem Kegel trennt.

Beispiel 34. Wir betrachten die Matrix

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}.$$

1. Für $b_1 = (3, 3)^\top$ ist $\bar{x} = (1, 1)^\top$ eine Lösung von $Ax = b_1$, das heißt, b_1 liegt im Kegel, der von $A_{\cdot,1} = (1, 2)^\top$ und $A_{\cdot,2} = (2, 1)^\top$ aufgespannt wird, vgl. Abbildung 11.1.

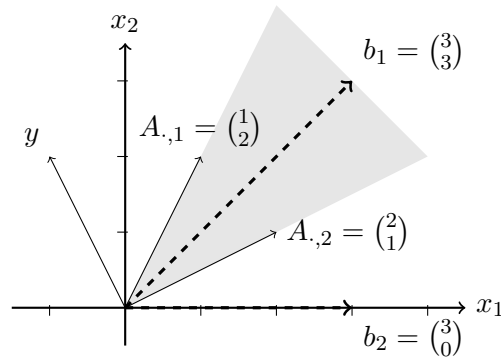


Abbildung 11.1: Geometrische Interpretation des Farkas-Lemmas

2. Für $b_2 = (3, 0)^\top$ hat das System $Ax = b$ keine Lösung, für die $x \geq 0$ gilt. Wir betrachten den Vektor $y = (-1, 2)^\top$. Dann hat y mit $A_{1,1} = (1, 2)^\top$ und $A_{1,2} = (2, 1)^\top$ ein nicht-negatives Skalarprodukt (d. h., einen spitzen oder rechten Winkel) und mit b_2 ein negatives Skalarprodukt (d. h., einen stumpfen Winkel); vgl. wieder Abbildung 11.1. \triangle

Korollar 11.6 (Variante des Farkas-Lemmas). Es sei $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$. Dann hat genau eines der beiden Systeme

$$Ax \leq b$$

und

$$\begin{aligned} A^\top y &= 0, \\ y &\geq 0, \\ b^\top y &< 0 \end{aligned} \tag{11.16}$$

eine Lösung.

Beweis. Wir führen im System $Ax \leq b$ eine Schlupfvariable $s \geq 0$ ein und ersetzen x durch $x^+ - x^-$, wobei $x^+, x^- \geq 0$. Dann ist $Ax \leq b$ genau dann lösbar, wenn $Ax^+ - Ax^- + s = b$, $x^+, x^-, s \geq 0$ lösbar ist. Wir definieren $A' := [A, -A, I]$. Nach obigen Überlegungen hat $Ax \leq b$ genau dann eine Lösung, wenn

$$A' \begin{pmatrix} x^+ \\ x^- \\ s \end{pmatrix} = b, \quad x^+, x^-, s \geq 0$$

eine Lösung besitzt. Dies ist nach der ersten Variante des Farkas-Lemmas 11.4 genau dann der Fall, wenn $(A')^\top y \geq 0$, $b^\top y < 0$ unlösbar ist. Dies ist aber mit $A' = [A, -A, I]$ wiederum äquivalent zu $A^\top y \geq 0$, $-A^\top y \geq 0$, $y \geq 0$, $b^\top y < 0$, also zum System (11.16). \square

11.4 Der Dualitätssatz der linearen Optimierung

Wir haben jetzt alle Hilfsmittel beisammen, um den starken Dualitätssatz der linearen Optimierung zu beweisen.

Satz 11.7 (Starker Dualitätssatz). Für das Paar

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0 \end{aligned} \tag{P}$$

und

$$\begin{aligned} \max_y \quad & b^\top y \\ \text{s.t.} \quad & A^\top y \leq c \end{aligned} \tag{D}$$

von linearen Optimierungsproblemen sind folgende Aussagen äquivalent:

- (i) (P) und (D) haben beide zulässige Lösungen.
- (ii) (P) und (D) haben beide optimale Lösungen x^* bzw. y^* und es gilt $c^\top x^* = b^\top y^*$.
- (iii) (P) oder (D) hat eine endliche Optimallösung.

Beweis. “(i) \implies (ii)”: Nach Korollar 11.2 genügt es zu zeigen, dass es zulässige Lösungen x^* von (P) und y^* von (D) gibt mit $c^\top x^* = b^\top y^*$. Da nach dem schwachen Dualitätssatz (Lemma 11.1, Kapitel 11.2) sowieso $c^\top x^* \geq b^\top y^*$ gilt, reicht es aus zu zeigen, dass für geeignete zulässige Lösungen x^* und y^* die Ungleichung $c^\top x^* \leq b^\top y^*$ gilt. Dies ist genau dann der Fall, wenn folgendes System lösbar ist:

$$Ax = b, \tag{11.17a}$$

$$A^\top y \leq c, \tag{11.17b}$$

$$c^\top x - b^\top y \leq 0, \tag{11.17c}$$

$$x \geq 0. \tag{11.17d}$$

Ersetzen wir y durch $y^+ - y^-$ mit $y^+, y^- \geq 0$, führen wir in (11.17b) den Vektor von Schlupfvariablen $s \geq 0$ und in (11.17c) die Schlupfvariable $\alpha \geq 0$ ein, so erhalten wir das zu (11.17) äquivalente System

$$Ax = b,$$

$$A^\top y^+ - A^\top y^- + s = c,$$

$$c^\top x - b^\top y^+ + b^\top y^- + \alpha = 0,$$

$$x, y^+, y^-, s, \alpha \geq 0,$$

was wiederum äquivalent ist zu

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 \\ 0 & A^\top & -A^\top & I & 0 \\ c^\top & -b^\top & b^\top & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y^+ \\ y^- \\ s \\ \alpha \end{pmatrix} = \begin{pmatrix} b \\ c \\ 0 \end{pmatrix}, \quad (11.18)$$

$$x, y^+, y^-, s, \alpha \geq 0.$$

Wir nehmen jetzt an, dass System (11.18) keine Lösung besitzt. Nach dem Farkas-Lemma 11.4 ist dann das System

$$\begin{bmatrix} A & 0 & 0 & 0 & 0 \\ 0 & A^\top & -A^\top & I & 0 \\ c^\top & -b^\top & b^\top & 0 & 1 \end{bmatrix}^\top \begin{pmatrix} u \\ v \\ \gamma \end{pmatrix} \geq 0,$$

$$\begin{pmatrix} b \\ c \\ 0 \end{pmatrix}^\top \begin{pmatrix} u \\ v \\ \gamma \end{pmatrix} < 0$$

lösbar, was wiederum äquivalent ist zu

$$\left\{ \begin{array}{l} A^\top u + c\gamma \geq 0 \\ Av - b\gamma \geq 0 \\ -Av + b\gamma \geq 0 \\ v \geq 0 \\ \gamma \geq 0 \\ b^\top u + c^\top v < 0 \end{array} \right\} \iff \left\{ \begin{array}{l} A^\top u + c\gamma \geq 0 \\ Av - b\gamma = 0 \\ v \geq 0 \\ \gamma \geq 0 \\ b^\top u + c^\top v < 0 \end{array} \right\}. \quad (11.19)$$

Sei u, v, γ eine Lösung von System (11.19). Ist $\gamma = 0$, so bilden u und v eine Lösung von

$$\left\{ \begin{array}{l} A^\top u \geq 0 \\ Av \geq 0 \\ -Av \geq 0 \\ v \geq 0 \\ b^\top u + c^\top v < 0 \end{array} \right\} \iff \begin{bmatrix} A^\top & 0 \\ 0 & A \\ 0 & -A \\ 0 & I \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} \geq 0, \quad \begin{pmatrix} b \\ c \end{pmatrix}^\top \begin{pmatrix} u \\ v \end{pmatrix} < 0,$$

und nach dem Farkas-Lemma 11.4 wäre dann das System

$$\begin{aligned} Ax &= b, \\ A^\top y^+ - A^\top y^- + s &= c, \\ x, y^+, y^-, s &\geq 0 \end{aligned}$$

unlösbar, was der Voraussetzung widerspricht, dass sowohl (P) als auch (D) zulässige Lösungen haben. Es muss also $\gamma > 0$ gelten. Wir können dann in System (11.19) jede Zeile durch $\gamma > 0$ teilen und sehen, dass $u\gamma$, $v\gamma$, 1 ebenfalls eine Lösung von System (11.19) bildet. Daher können wir ohne Einschränkung $\gamma = 1$ annehmen, so dass u und v folgendes System lösen:

$$A^\top u \geq -c, \quad (11.20a)$$

$$Av = b, \quad (11.20b)$$

$$v \geq 0, \quad (11.20c)$$

$$b^\top u + c^\top v < 0. \quad (11.20d)$$

Dann folgt aber

$$\begin{aligned} 0 &> b^\top u + c^\top v && \text{(mit Ungleichung (11.20d))} \\ &= u^\top Av + c^\top v && \text{(mit Gleichung (11.20b))} \\ &\geq u^\top Av - u^\top Av = 0, && \text{(mit Ungleichungen (11.20a), (11.20c))} \end{aligned}$$

also ein Widerspruch. Folglich ist System (11.19) unlösbar und damit System (11.18), wie gewünscht, lösbar.

“(ii) \implies (iii)”: trivial.

“(iii) \implies (i)”: Es habe zunächst (D) eine endliche Optimallösung y^* . Wäre (P) unzulässig, so gäbe es nach dem Farkas-Lemma 11.4 ein y mit $A^\top y \geq 0$ und $b^\top y < 0$. Für $u = -y$ gilt dann $A^\top u \leq 0$ und $b^\top u > 0$. Dann ist aber für beliebiges $\lambda > 0$ der Vektor $y^* + \lambda u$ wegen

$$A^\top(y^* + \lambda u) = \underbrace{A^\top y^*}_{\leq c} + \lambda \underbrace{A^\top u}_{\leq 0} \leq c$$

zulässig für (D) und es gilt

$$b^\top(y^* + \lambda u) = b^\top y^* + \lambda b^\top u > b^\top y^*,$$

im Widerspruch zur Optimalität von y^* . Also muss (P) zulässig sein. Es habe nun (P) eine Optimallösung x^* . Hätte (D) keine zulässige Lösung, so wäre $A^\top y \leq c$ unlösbar. Nach Korollar 11.6 gibt es dann ein $x \geq 0$ mit $Ax = 0$ und $c^\top x < 0$. Für $\lambda > 0$ ist dann $x^* + \lambda x$ zulässig für (P) und

$$c^\top(x^* + \lambda x) = c^\top x^* + \lambda c^\top x < c^\top x^*$$

im Widerspruch zur Optimalität von x^* . Also muss (D) zulässig sein. Da aus der Existenz einer endlichen Optimallösung von (P) bzw. (D) jeweils auch die Zulässigkeit von (P) bzw. (D) folgt, sind wir fertig. \square

Korollar 11.8. Es seien P bzw. D die Mengen der zulässigen Lösungen von (P) bzw. (D). Setzen wir

$$z^* = \begin{cases} -\infty, & \text{falls (P) unbeschränkt ist,} \\ +\infty, & \text{falls } P = \emptyset, \\ \min \{c^\top x : x \in P\}, & \text{sonst} \end{cases}$$

und

$$w^* = \begin{cases} +\infty, & \text{falls (D) unbeschränkt,} \\ -\infty, & \text{falls } D = \emptyset, \\ \max \{b^\top y : y \in D\}, & \text{sonst,} \end{cases}$$

so gilt

- (a) $z^* = -\infty \implies D = \emptyset$,
- (b) $w^* = +\infty \implies P = \emptyset$,
- (c) $P = \emptyset \implies (D = \emptyset \text{ oder } w^* = +\infty)$,
- (d) $D = \emptyset \implies (P = \emptyset \text{ oder } z^* = -\infty)$.

Beweis. (a) Wäre $y \in D \neq \emptyset$, so folgt aus dem schwachen Dualitätssatz 11.1, dass $z^* \geq b^\top y$ gilt, was $z^* = -\infty$ widerspricht.

(b) Der Beweis erfolgt analog zu (a).

(c) Nach dem Dualitätssatz 11.7 kann nicht gleichzeitig $w^* < \infty$ und $P = \emptyset$ eintreten. Daraus folgt die Behauptung.

(d) Der Beweis erfolgt analog zu (c). □

Als Ergänzung zu den beiden Fällen (c) und (d) von Korollar 11.8 betrachten wir nun ein Beispiel mit $P = \emptyset$ und $D = \emptyset$.

Beispiel 35. Es sei

$$P = \{x \in \mathbb{R}^2 : x_1 - x_2 = 1, -x_1 + x_2 = 1, x \geq 0\},$$

$$D = \{y \in \mathbb{R}^2 : y_1 - y_2 \leq 0, -y_1 + y_2 \leq -1\}.$$

Dann sind die linearen Optimierungsprobleme

$$\min \{-x_2 : x \in P\} \quad \text{und} \quad \max \{y_1 + y_2 : y \in D\}$$

dual zueinander und es gilt:

- (a) $D = \emptyset$, da nach der Variante des Farkas-Lemmas aus Korollar 11.6 das System

$$\begin{aligned} u_1 - u_2 &= 0, \\ -u_1 + u_2 &= 0, \\ u_1, u_2 &\geq 0, \\ -u_2 &< 0 \end{aligned}$$

eine Lösung hat, nämlich $u_1 = u_2 = 1$.

- (b) $P = \emptyset$, da nach dem Farkas-Lemma 11.4 das System

$$\begin{aligned} u_1 - u_2 &\geq 0, \\ -u_1 + u_2 &\geq 0, \\ u_1 + u_2 &< 0 \end{aligned}$$

eine Lösung hat, nämlich $u_1 = u_2 = -1$.

\triangle

Betrachten wir noch einmal das Paar (P) und (D) von linearen Optimierungsproblemen. Wir sind in unserer Herleitung der Dualität von (P) ausgegangen und haben das duale Optimierungsproblem (D) erhalten. Wir zeigen jetzt, dass die Dualisierung in gewisser Weise symmetrisch ist – wir also umgekehrt bei der Dualisierung von (D) auch wieder (P) erhalten. Dazu schreiben wir (D) äquivalent durch Einführung von Schlupfvariablen $s \geq 0$ um und ersetzen y durch $y = y^+ - y^-$:

$$\begin{aligned} - \min_{y^+, y^-, s} \quad & \begin{pmatrix} -b \\ b \\ 0 \end{pmatrix}^\top \begin{pmatrix} y^+ \\ y^- \\ s \end{pmatrix} \\ \text{s.t.} \quad & [A^\top \quad -A^\top \quad I] \begin{pmatrix} y^+ \\ y^- \\ s \end{pmatrix} = c, \\ & \begin{pmatrix} y^+ \\ y^- \\ s \end{pmatrix} \geq 0. \end{aligned}$$

Damit erhalten wir ein lineares Optimierungsproblem in Standardform, zu dem wir (rein mechanisch) das duale lineare Optimierungsproblem bilden können. Dieses lautet dann

$$\begin{aligned} - \max_x \quad & c^\top x \\ \text{s.t.} \quad & \begin{bmatrix} A \\ -A \\ I \end{bmatrix} x \leq \begin{pmatrix} -b \\ b \\ 0 \end{pmatrix}, \end{aligned}$$

was äquivalent ist zu

$$\begin{aligned} & - \max_x \quad -c^\top(-x) \\ & \text{s.t.} \quad A(-x) = b, \\ & \quad \quad (-x) \geq 0, \end{aligned}$$

was wiederum äquivalent ist zu

$$\begin{aligned} & \min_{\hat{x}} \quad c^\top \hat{x} \\ & \text{s.t.} \quad A\hat{x} = b, \\ & \quad \quad \hat{x} \geq 0. \end{aligned}$$

Dies zeigt, dass das Duale zum dualen Optimierungsproblem wieder das primale lineare Optimierungsproblem (P) ist. Das bislang behandelte Konzept der Dualität lässt sich mit den oben benutzten “Tricks” wie dem Einführen von Schlupfvariablen und Variablensplitting auch auf beliebige lineare Optimierungsprobleme verallgemeinern.

Satz 11.9. Gegeben seien vier dimensionsverträgliche Matrizen A, B, C, D , und Vektoren a, b, c, d und das (primale) lineare Optimierungsproblem

$$\begin{aligned} & \min_{x,y} \quad c^\top x + d^\top y \\ & \text{s.t.} \quad Ax + By \geq a, \\ & \quad \quad Cx + Dy = b, \\ & \quad \quad x \geq 0. \end{aligned} \tag{11.21}$$

Dann gilt:

(a) Das zu (11.21) duale Optimierungsproblem lautet

$$\begin{aligned} & \max_{u,v} \quad a^\top u + b^\top v \\ & \text{s.t.} \quad A^\top u + C^\top v \leq c, \\ & \quad \quad B^\top u + D^\top v = d, \\ & \quad \quad u \geq 0. \end{aligned} \tag{11.22}$$

(b) Das zu (11.22) duale Optimierungsproblem ist (11.21).

(c) Satz 11.7 und Korollar 11.8 gelten analog für (11.21) und (11.22).

Beweis. (a) Mithilfe der Transformationsregeln schreiben wir (11.21) als

$$\begin{aligned} & \min_{x,y^+,y^-,s} \quad c^\top x + d^\top y^+ - d^\top y^- \\ & \text{s.t.} \quad Ax + By^+ - By^- - s = a, \\ & \quad \quad Cx + Dy^+ - Dy^- = b, \\ & \quad \quad x, y^+, y^-, s \geq 0, \end{aligned}$$

Primal		Dual
Ungleichung	\leftrightarrow	nicht-negative Variable
Gleichung	\leftrightarrow	freie Variable
nicht-negative Variable	\leftrightarrow	Ungleichung
freie Variable	\leftrightarrow	Gleichung

Tabelle 11.1: Transformationsregeln

also in Standardform. Das hierzu duale lineare Optimierungsproblem lautet

$$\begin{aligned}
\max_{u,v} \quad & a^\top u + b^\top v \\
\text{s.t.} \quad & A^\top u + C^\top v \leq c, \\
& B^\top u + D^\top v \leq d, \\
& -B^\top u - D^\top v \leq -d, \\
& -u \leq 0.
\end{aligned}$$

Dies entspricht (11.22).

(b) Der Beweis erfolgt analog zu (a).

(c) Die Behauptung folgt aus Teil (a). □

In Tabelle 11.1 sind einige Transformationsregeln zusammengefasst, die angeben, wie man aus einem primalen linearen Optimierungsproblem das zugehörige duale Problem ableitet und umgekehrt.

11.5 Komplementärer Schlupf

Satz 11.10 (Satz vom schwachen komplementären Schlupf für allgemeine LPs). Gegeben seien dimensionsverträgliche Matrizen A, B, C, D und Vektoren a, b, c, d . Wir betrachten die zueinander dualen linearen Optimierungsprobleme

$$\begin{aligned}
\min_{x,y} \quad & c^\top x + d^\top y \\
\text{s.t.} \quad & Ax + By \geq a, \\
& Cx + Dy = b, \\
& x \geq 0
\end{aligned} \tag{P}$$

und

$$\begin{aligned}
\max_{u,v} \quad & a^\top u + b^\top v \\
\text{s.t.} \quad & A^\top u + C^\top v \leq c, \\
& B^\top u + D^\top v = d, \\
& u \geq 0,
\end{aligned} \tag{D}$$

vgl. (11.21) und (11.22) im letzten Abschnitt. Die Vektoren $(\bar{x}, \bar{y})^\top$ und $(\bar{u}, \bar{v})^\top$ seien zulässig für (P) bzw. (D). Dann sind folgende Aussagen äquivalent:

- (i) $(\bar{x}, \bar{y})^\top$ ist optimal für (P) und $(\bar{u}, \bar{v})^\top$ ist optimal für (D).
- (ii) Es gilt

$$\left(c - (A^\top \bar{u} + C^\top \bar{v}) \right)^\top \bar{x} + \bar{u}^\top (A\bar{x} + B\bar{y} - a) = 0.$$

- (iii) Für alle Komponenten \bar{u}_i der Duallösung gilt

$$\bar{u}_i > 0 \implies A_{i,\cdot} \bar{x} + B_{i,\cdot} \bar{y} = a_i,$$

d. h., ist $\bar{u}_i > 0$, so ist die i -te primale Ungleichung in $A\bar{x} + B\bar{y} \geq a$ mit Gleichheit erfüllt. Für alle Komponenten \bar{x}_j der Primallösung gilt

$$\bar{x}_j > 0 \implies A_{\cdot,j}^\top \bar{u} + C_{\cdot,j}^\top \bar{v} = c_j,$$

d. h., ist $\bar{x}_j > 0$, so ist die j -te duale Ungleichung in $A^\top \bar{u} + C^\top \bar{v} \leq c$ mit Gleichheit erfüllt.

- (iv) Für alle Zeilenindizes i von A und B gilt

$$A_{i,\cdot} \bar{x} + B_{i,\cdot} \bar{x} > a_i \implies \bar{u}_i = 0,$$

d. h., ist die i -te primale Ungleichung strikt erfüllt, so ist die zugehörige Dualvariable \bar{u}_i gleich 0. Für alle Spaltenindizes j von A und C gilt

$$A_{\cdot,j}^\top \bar{u} + C_{\cdot,j}^\top \bar{v} < c_j \implies \bar{x}_j = 0,$$

d. h., ist die j -te duale Ungleichung strikt erfüllt, so ist die zugehörige primale Variable \bar{x}_j gleich 0.

Beweis. “(i) \iff (ii)”: Nach Satz 11.9 ist $(\bar{x}, \bar{y})^\top$ optimal für (P) und $(\bar{u}, \bar{v})^\top$ optimal für (D) genau dann, wenn

$$c^\top \bar{x} + d^\top \bar{y} = a^\top \bar{u} + b^\top \bar{v}$$

gilt. Also haben wir

$$\begin{aligned}
\text{(i)} & \iff c^\top \bar{x} + d^\top \bar{y} = a^\top \bar{u} + b^\top \bar{v} \\
& \iff c^\top \bar{x} + (B^\top \bar{u} + D^\top \bar{v})^\top \bar{y} = a^\top \bar{u} + (C\bar{x} + D\bar{y})^\top \bar{v} \\
& \iff (c - C^\top \bar{v})^\top \bar{x} + \bar{u}^\top (B\bar{y} - a) = 0 \\
& \iff (c - C^\top \bar{v})^\top \bar{x} - (A^\top \bar{u})^\top \bar{x} + \bar{u}^\top (B\bar{y} - a) + \bar{u}^\top A\bar{x} = 0 \\
& \iff (c - C^\top \bar{v} - A^\top \bar{u})^\top \bar{x} + \bar{u}^\top (B\bar{y} + A\bar{x} - a) = 0 \\
& \iff \text{(ii)}.
\end{aligned}$$

“(ii) \implies (iii)”: Sei $t = c - (A^\top \bar{u} + C^\top \bar{v})$ und $s = A\bar{x} + B\bar{y} - a$. Da $(\bar{x}, \bar{y})^\top$ zulässig für (P) ist, gilt $s \geq 0$. Analog folgt aus der Zulässigkeit von $(\bar{u}, \bar{v})^\top$ für (D), dass $t \geq 0$. Aus $\bar{x} \geq 0$, $\bar{u} \geq 0$, $s \geq 0$ und $t \geq 0$ folgt daher $t^\top \bar{x} \geq 0$ und $\bar{u}^\top s \geq 0$. Nach Voraussetzung ist $t^\top \bar{x} + \bar{u}^\top s = 0$, was aber nur gelten kann, falls $t^\top \bar{x} = \bar{u}^\top s = 0$ gilt. Dies impliziert (iii).

“(iii) \implies (ii)”: Klar.

“(iv) \iff (iii)”: Beweis folgt sofort durch Negation. \square

Der Name des Satzes ist durch die im Beweis definierten Größen t und s begründet. Diese beschreiben gerade den Schlupf in den primalen bzw. dualen Ungleichungen und der Satz sagt, dass dieser Schlupf komplementär zu den zugehörigen dualen bzw. primalen Variablen ist.

Für das lineare Optimierungsproblem in Standardform (11.6) und sein duales Problem (11.10) besitzen die Komplementaritätsbedingungen eine etwas einfachere Form.

Korollar 11.11 (Satz vom schwachen komplementären Schlupf für LPs in Standardform). Wir betrachten das Paar zueinander dualer linearer Optimierungsprobleme

$$\begin{aligned}
\min_{x} \quad & c^\top x \\
\text{s.t.} \quad & Ax = b, \\
& x \geq 0
\end{aligned} \tag{11.23}$$

und

$$\begin{aligned}
\max_{y} \quad & b^\top y \\
\text{s.t.} \quad & A^\top y \leq c.
\end{aligned} \tag{11.24}$$

Ferner sei \bar{x} zulässig für (11.23) und \bar{y} zulässig für (11.24). Dann sind folgende Aussagen äquivalent:

- (i) \bar{x} ist optimal für (11.23) und \bar{y} ist optimal für (11.24).

(ii) Es gilt

$$(c - A^\top \bar{y})^\top \bar{x} = 0.$$

(iii) Für alle Komponenten \bar{x}_j der Primallösung gilt

$$\bar{x}_j > 0 \implies A_{\cdot,j}^\top \bar{y} = c_j,$$

d. h., ist $\bar{x}_j > 0$, so ist die j -te duale Ungleichung in $A^\top \bar{y} \leq c$ mit Gleichheit erfüllt.

(iv) Für alle Spaltenindizes j von A gilt

$$A_{\cdot,j}^\top \bar{y} < c_j \implies \bar{x}_j = 0,$$

d. h., ist die j -te duale Ungleichung in $A^\top \bar{y} \leq c$ strikt erfüllt, so ist die zugehörige primale Variable \bar{x}_j gleich 0.

Beweis. Folgt unmittelbar aus Satz 11.10. □

Die Umkehrungen in (iii) und (iv) gelten im Allgemeinen nicht. Es gibt jedoch immer Paare von Lösungen, die diese erfüllen.

Satz 11.12 (Satz vom starken komplementären Schlupf). Wir betrachten das Paar zueinander dualer linearer Optimierungsprobleme (11.23) und (11.24) aus Korollar 11.11. Besitzt sowohl (11.23) als auch (11.24) eine zulässige Lösung, so existieren strikt komplementäre Optimallösungen \bar{x} von (11.23) und \bar{y} von (11.24) mit

$$\begin{aligned} \bar{x}_j > 0 &\iff A_{\cdot,j}^\top \bar{y} = c_j, \\ \bar{x}_j = 0 &\iff A_{\cdot,j}^\top \bar{y} < c_j. \end{aligned}$$

11.6 Dualität und das Simplex-Verfahren

In diesem Abschnitt analysieren wir den Zusammenhang des Konzepts der Dualität mit dem Simplex-Verfahren. Wir betrachten also wieder das Paar dualer zueinander linearer Optimierungsprobleme

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0 \end{aligned}$$

und

$$\begin{aligned} \max_y \quad & b^\top y \\ \text{s.t.} \quad & A^\top y \leq c, \end{aligned}$$

wobei wir wieder annehmen, dass $\text{Rang } A = m$ gilt, also A vollen Zeilenrang besitzt, und die Menge der zulässigen Lösungen des primalen Problems nicht leer ist. Unter diesen Voraussetzungen können wir den Fundamentalsatz 10.9 der linearen Optimierung anwenden, der besagt, dass (P) unter diesen Annahmen eine zulässige Basislösung besitzt. Wir wenden nun die Simplex-Methode auf (P) an, wobei wir in der existierenden zulässigen Basislösung starten. Der Simplex-Algorithmus 20 kann auf zwei Arten abbrechen:

- (a) Mit einer optimalen Basislösung $\bar{x} = (\bar{x}_B, \bar{x}_N)^\top = (A_B^{-1}b, 0)^\top$, wobei B die zugehörige Basis ist.
- (b) Mit der Information, dass das primale Problem (P) unbeschränkt ist.

Wir behandeln hier den ersten Fall. Wenn die Simplex-Methode mit der Information abbricht, dass die aktuelle Basislösung \bar{x} optimal ist, so gilt nach Konstruktion für den Vektor $z_N = c_N - A_N^\top (A_B^\top)^{-1} c_B$ die Bedingung $z_N \geq 0$, also

$$A_N^\top (A_B^\top)^{-1} c_B \leq c_N. \quad (11.25)$$

Wir betrachten den Vektor

$$\bar{y} = (A_B^\top)^{-1} c_B,$$

der in Algorithmus 20 im BTRAN-Schritt berechnet wird. Für diesen Vektor \bar{y} gilt

$$\begin{aligned} A^\top \bar{y} &= \begin{bmatrix} A_B^\top \\ A_N^\top \end{bmatrix} (A_B^\top)^{-1} c_B \\ &= \begin{pmatrix} A_B^\top (A_B^\top)^{-1} c_B \\ A_N^\top (A_B^\top)^{-1} c_B \end{pmatrix} \\ &= \begin{pmatrix} c_B \\ A_N^\top (A_B^\top)^{-1} c_B \end{pmatrix} \\ &\leq \begin{pmatrix} c_B \\ c_N \end{pmatrix} = c, \end{aligned}$$

wobei die letzte Ungleichung aus Ungleichung (11.25) folgt. Der Vektor \bar{y} ist also zulässig für das duale lineare Optimierungsproblem (D).

Für seinen Zielfunktionswert $b^\top \bar{y}$ ergibt sich

$$b^\top \bar{y} = b^\top ((A_B^\top)^{-1} c_B) = c_B^\top A_B^{-1} b = c_B^\top \bar{x}_B = c^\top \bar{x},$$

und daher ist \bar{y} nach Korollar 11.2 eine optimale Lösung des dualen Problems (D). Damit erhalten wir folgenden Satz.

Satz 11.13. Terminiert der Simplex-Algorithmus 20 mit einer optimalen Basislösung $\bar{x} = (\bar{x}_B, \bar{x}_N)^\top = (A_B^{-1}b, 0)^\top$ von (P), so ist

$$\bar{y} = (A_B^\top)^{-1} c_B$$

eine Optimallösung von (D).

11.7 Sensitivitätsanalyse I

Wir sind bisher davon ausgegangen, dass die Daten c , A und b eines linearen Optimierungsproblems der Form

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0 \end{aligned} \tag{11.26}$$

fest vorgegeben sind. Häufig ist es jedoch der Fall, dass sich diese Daten im Lauf der Zeit ändern, wenn Bedingungen und/oder Variablen hinzukommen und “nachoptimiert” werden muss. Wir gehen jetzt davon aus, dass wir ein lineares Optimierungsproblem der Form (11.26) bereits gelöst haben und eine optimale Basis B mit zugehöriger Basislösung $\bar{x}_B = A_B^{-1}b$ und $\bar{x}_N = 0$ kennen. Wir nehmen ferner an, dass die Basislösung \bar{x} nicht degeneriert ist, d. h., dass $\bar{x}_B > 0$ gilt. In diesem Fall gilt wie im letzten Abschnitt für den Vektor $z_N = c_N - A_N^\top(A_B^\top)^{-1}c_B$ der reduzierten Kosten, dass $z_N \geq 0$ ist, und dass der Vektor $\bar{y} = (A_B^\top)^{-1}c_B$ eine Optimallösung des dualen linearen Optimierungsproblems ist. Wir betrachten nun die Situation, in der sich die rechte Seite b um eine “kleine Störung” Δb ändert, d. h., wir betrachten das neue lineare Optimierungsproblem

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b + \Delta b, \\ & x \geq 0. \end{aligned} \tag{11.27}$$

Die alte Basis B ist immer noch eine Basis des gestörten linearen Optimierungsproblems (11.27) und

$$\begin{aligned} \bar{x}' &= \begin{pmatrix} \bar{x}'_B \\ \bar{x}'_N \end{pmatrix} = \begin{pmatrix} A_B^{-1}b + A_B^{-1}\Delta b \\ 0 \end{pmatrix} \\ &= \bar{x} + \begin{pmatrix} A_B^{-1}\Delta b \\ 0 \end{pmatrix} \\ &=: \bar{x} + \begin{pmatrix} \Delta x \\ 0 \end{pmatrix} \end{aligned}$$

ist eine Basislösung. Da nach unserer Voraussetzung B nicht degeneriert ist, also

$$\bar{x}_B = A_B^{-1}b > 0$$

gilt, ist für hinreichend kleines Δb auch $\bar{x}_B + \Delta x = \bar{x}_B + A_B^{-1}\Delta b \geq 0$, also \bar{x}' eine zulässige Basislösung für (11.27). Da der Vektor z_N der reduzierten Kosten unverändert bleibt, ist \bar{x}' somit für hinreichend kleine Störungen Δb eine

Optimallösung des gestörten Problems (11.27). Die entsprechende Änderung in der Zielfunktion ist gegeben durch

$$c_B^\top(\bar{x}_B + \Delta x) - c_B^\top \bar{x}_B = c_B^\top \Delta x = c_B^\top A_B^{-1} \Delta b = \bar{y}^\top \Delta b. \quad (11.28)$$

Gleichung (11.28) zeigt, dass die optimale Duallösung \bar{y} die *Sensitivität* bzgl. kleiner Änderungen in der rechten Seite b misst: Wenn wir b durch $b + \Delta b$ ersetzen, so ändert sich (für hinreichend kleines Δb) der optimale Zielfunktionswert um $\bar{y}^\top \Delta b$.

11.8 Eine ökonomische Interpretation von Dualvariablen

In vielen Anwendungen haben die Dualvariablen eine sinnvolle ökonomische Interpretation. Wir betrachten im Folgenden das LP

$$\max_x \quad c^\top x \quad (11.29a)$$

$$\text{s.t.} \quad Ax \leq b, \quad (11.29b)$$

$$x \geq 0, \quad (11.29c)$$

mit $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$. Dementsprechend hat unser LP m duale Variablen y_1, \dots, y_m , die (rein strukturell), mit den m primalen Nebenbedingungen

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = 1, \dots, m,$$

in Zusammenhang stehen. Unser Ziel ist es, jetzt auch einen ökonomischen Zusammenhang herzuleiten.

Dazu interpretieren wir das obige LP als Modell des folgenden Problems: Die primale Variable x_j bezeichnet die Produktion eines j -ten Erzeugnisses und b_i ist die zur Verfügung stehende Menge der i -ten Ressource. Also muss der Koeffizient a_{ij} interpretiert werden als die Menge der Ressource i , die zur Herstellung einer Einheit des j -ten Produkts benötigt wird. Schließlich ist c_j der Nettogewinn pro produzierter Menge des j -ten Produkts.

Das duale Problem enthält die Nebenbedingungen

$$\sum_{i=1}^m a_{ij} y_i \geq c_j, \quad j = 1, \dots, n.$$

Wenn a_{ij} also “Menge von Ressource i pro Produktion einer Einheit von Produkt j ” ist und die obigen Terme zur rechten Seite c_j (in der Einheit “Wert

pro produzierter j -ter Einheit”) passen sollen, müssen die y_j der Interpretation “Wert pro Ressource i ” genügen.

Die Erkenntnisse des letzten Abschnitts übertragen auf diese Situation ergeben den folgenden Satz.

Satz 11.14. Das primale LP (11.29) habe mindestens eine nicht-degenerierte optimale Basislösung. Dann existiert ein $\varepsilon > 0$ mit der folgenden Eigenschaft: Falls $|t_i| \leq \varepsilon$ für alle $i = 1, \dots, m$ gilt, dann hat das lineare Optimierungsproblem

$$\begin{aligned} \max_x \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b + t, \\ & x \geq 0 \end{aligned}$$

eine Optimallösung und der optimale Zielfunktionswert ist

$$z^* + \bar{y}^\top t,$$

wobei z^* der optimale Zielfunktionswert des ungestörten Problems (11.29) und \bar{y} die optimale Lösung des zu (11.29) gehörenden dualen Problems ist.

Beweis. Siehe Chvátal [5]. □

Dieser Satz formalisiert genau unsere vorherigen Überlegungen. Die dualen Variablen messen den Wert (hinreichend kleiner) Änderungen in den zur Verfügung stehenden Ressourcen: Mit jeder zusätzlichen Einheit der i -ten Ressource verändert sich der Gewinn um $\bar{y}_i t_i$.

Im ökonomischen Kontext stellt \bar{y}_i den Höchstpreis dar, wie viel ein Unternehmen in Ressource i investieren sollte. Dies ist auch der Grund, warum \bar{y}_i als *Schattenpreis* (engl. *shadow price*) bezeichnet wird.

Kapitel 12

Ein zweiter Blick auf das Simplex-Verfahren

In Kapitel 9.1 haben wir die Grundform des Simplex-Verfahrens zur Lösung linearer Optimierungsprobleme in Standardform

$$\min_x \quad c^\top x \quad (12.1a)$$

$$\text{s.t.} \quad Ax = b, \quad (12.1b)$$

$$x \geq 0 \quad (12.1c)$$

kennengelernt. Wir haben gesehen, wie wir ausgehend von einer zulässigen Basislösung $\bar{x} = (\bar{x}_B, \bar{x}_N)^\top$ mit $\bar{x}_B = A_B^{-1}b \geq 0$ und $\bar{x}_N = 0$ entweder eine neue Basislösung mit nicht schlechterem Zielfunktionswert finden oder korrekt schließen können, dass das Problem (12.1) unbeschränkt ist, d. h., Lösungen mit beliebig kleinem Zielfunktionswert besitzt. In diesem Kapitel beschäftigen wir uns mit den zwei wichtigen Punkten, die noch dazu fehlen, das Simplex-Verfahren zu einem vollständigen Optimierungsalgorithmus zu machen.

Terminiertheit: Bricht der Simplex-Algorithmus immer ab oder kann es vorkommen, dass unendlich viele Iterationen durchgeführt werden?

Mit dieser Frage beschäftigen wir uns in Abschnitt 12.1. Wir werden feststellen, dass die Wiederholung einer Basislösung die einzige Möglichkeit ist, die einen Abbruch nach endlich vielen Schritten verhindern kann. Wir zeigen dann, dass durch geeignete Wahl der eintretenden bzw. die Basis verlassenden Variablen dieses sogenannte *Kreiseln* ausgeschlossen werden kann.

Effizienz: Nachdem wir die Terminierung nach endlichen vielen Schritten behandelt haben, diskutieren wir in Abschnitt 12.2 kurz, wie effizient das Simplex-Verfahren ist.

Initialisierung: Wie finden wir eine erste zulässige Basis?

Die Initialisierung der Simplex-Methode ist Gegenstand von Kapitel 12.3. Wir zeigen, wie man für jedes Problem (12.1) in Standardform ein sogenanntes *Phase-1-Problem* aufstellen kann, das wiederum ein lineares Problem ist und für das man leicht eine zulässige Basis angeben kann. Die Lösung des Phase-1-Problems mit der Simplex-Methode liefert dann entweder eine zulässige Startbasis für (12.1) oder die Information, dass (12.1) keine zulässige Lösung besitzt.

12.1 Terminiertheit

12.1.1 Ein Negativ-Beispiel

Wir betrachten das folgende lineare Problem in Standardform:

$$\begin{aligned} \min \quad & -2.3x_1 - 2.15x_2 + 13.55x_3 + 0.4x_4 \\ \text{s.t.} \quad & 0.4x_1 + 0.2x_2 - 1.4x_3 - 0.2x_4 + x_5 = 0, \\ & -7.8x_1 - 1.4x_2 + 7.8x_3 + 0.4x_4 + x_6 = 0, \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0, \end{aligned}$$

In der Simplex-Methode gibt es im Allgemeinen mehrere Möglichkeiten, eine Variable in die Basis aufzunehmen bzw. aus der Basis zu entfernen. Für dieses Beispiel setzen wir folgende Regeln fest:

- Falls es beim **Pricing** mehrere Nichtbasisvariablen j mit $z_j < 0$ gibt, so wählen wir die Nichtbasisvariable mit kleinstem z_j aus, um in die Basis aufgenommen zu werden.
- Falls es beim **Ratio-Test** mehrere Basisvariablen gibt, für die das Minimum angenommen wird, so entfernen wir eine Basisvariable x_{B_k} mit größtem Wert w_k .

Für unser Beispiel ist $B = (5, 6)$ eine primal zulässige Basis und die zugehörige Basislösung ist gegeben durch

$$\begin{aligned} \bar{x}_B = \begin{pmatrix} \bar{x}_5 \\ \bar{x}_6 \end{pmatrix} &= A_B^{-1}b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \\ \bar{x}_N &= \begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \\ \bar{x}_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \end{aligned}$$

Der Zielfunktionswert ist $c^\top \bar{x} = 0$. Wir führen nun die einzelnen Schritte des Simplex-Algorithmus durch.

- (1) BTRAN: Löse $\bar{y}^\top A_B = c_B^\top$, d. h.

$$\bar{y}^\top \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = (0, 0) \implies \bar{y}^\top = (0, 0).$$

- (2) Pricing: Berechne $z_N = c_N - A_N^\top \bar{y}$. Wir haben

$$z_N = c_N - A_N^\top \begin{pmatrix} 0 \\ 0 \end{pmatrix} = c_N = \begin{pmatrix} -2.3 \\ -2.15 \\ 13.55 \\ 0.4 \end{pmatrix}.$$

Die Nichtbasisvariable mit dem kleinsten negativen Wert $z_1 = -2.3$ ist x_1 . Somit wird x_1 in die Basis aufgenommen und im Simplex-Algorithmus wird $j = 1$ gewählt.

- (3) FTRAN: Löse $A_B w = A_j$:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} w = \begin{pmatrix} 0.4 \\ -7.8 \end{pmatrix} \implies w = \begin{pmatrix} 0.4 \\ -7.8 \end{pmatrix}.$$

Es wird sich nachher als nützlich für die Darstellung herausstellen, wenn wir im FTRAN-Schritt nicht nur $w = A_B^{-1} A_j$ berechnen, sondern gleich die komplette Matrix $A_B^{-1} A_N$. Für diese gilt

$$A_B^{-1} A_N = \begin{bmatrix} \mathbf{0.4} & 0.2 & -1.4 & -0.2 \\ \mathbf{-7.8} & -1.4 & 7.8 & 0.4 \end{bmatrix}. \quad (12.2)$$

Den Vektor w haben wir dabei in der Matrix fett hervorgehoben.

- (4) Ratio-Test: Wir berechnen

$$\gamma = \min \left\{ \frac{\bar{x}_{B_k}}{w_k} : w_k > 0 \text{ und } k \in \{1, \dots, m\} \right\}.$$

Es gilt $\bar{x}_B = (\bar{x}_5, \bar{x}_6)^\top = (0, 0)^\top$. Da alle Basisvariablen den Wert 0 haben und nur $w_1 = 0.4 > 0$ gilt, haben wir $\gamma = 0$ und $x_{B_1} = x_5$ verlässt die Basis.

- (5) Update: Wir berechnen

$$\begin{aligned} \bar{x}_B &= \begin{pmatrix} \bar{x}_5 \\ \bar{x}_6 \end{pmatrix} - \gamma w = \begin{pmatrix} \bar{x}_5 \\ \bar{x}_6 \end{pmatrix} - 0 \cdot w = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \\ N &= \{1, 2, 3, 4\} \setminus \{1\} \cup \{5\} = \{2, 3, 4, 5\}, \\ B_1 &= 1, \\ \bar{x}_1 &= \gamma = 0. \end{aligned}$$

Wir erhalten die neue Basislösung \bar{x} mit

$$\bar{x}_B = \begin{pmatrix} \bar{x}_1 \\ \bar{x}_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \bar{x}_N = \begin{pmatrix} \bar{x}_2 \\ \bar{x}_3 \\ \bar{x}_4 \\ \bar{x}_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

zur Basis $B = (1, 6)$.

Im Beispiel haben wir zwar eine neue Basis erhalten, die Basislösung stimmt aber mit derjenigen aus der letzten Iteration überein, da im Ratio-Test $\gamma = 0$ gilt.

Definition 12.1 (Degenerierte Iteration, degenerierter Pivot). Eine Iteration des Simplex-Algorithmus, bei der im Ratio-Test $\gamma = 0$ gilt, heißt *degenerierte Iteration* oder *degenerierter Pivot*.

Bei einer degenerierten Iteration machen wir im Bezug auf die Zielfunktion keinen Fortschritt, da sich die eigentliche Lösung nicht verändert. Wir führen nun unser Beispiel weiter fort, in dem wir die nächste Iteration durchführen.

(1) BTRAN: Löse $\bar{y}^\top A_B = c_B^\top$, d. h.

$$\bar{y}^\top \begin{bmatrix} 0.4 & 0 \\ -7.8 & 1 \end{bmatrix} = (-2.3, 0) \implies \bar{y}^\top = (-5.75, 0).$$

(2) Pricing: Berechne $z_N = c_N - A_N^\top \bar{y}$:

$$z_N = \begin{pmatrix} -2.15 \\ 13.55 \\ 0.4 \\ 0 \end{pmatrix} - \begin{bmatrix} 0.2 & -1.4 \\ 1.4 & 7.8 \\ -0.2 & 0.4 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} -5.75 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 21.6 \\ -0.75 \\ 5.75 \end{pmatrix}.$$

Gemäß unserer Festlegungen wählen wir $j = 2$ mit $z_2 = -1$ und die zugehörige Nichtbasisvariable x_2 wird in die Basis aufgenommen.

(3) FTRAN: Löse $A_B w = A_j$:

$$\begin{bmatrix} 0.4 & 0 \\ -7.8 & 1 \end{bmatrix} w = \begin{pmatrix} 0.2 \\ -1.4 \end{pmatrix} \implies w = \begin{pmatrix} 0.5 \\ 2.5 \end{pmatrix}.$$

(4) Ratio-Test: Berechne

$$\gamma = \min \left\{ \frac{\bar{x}_{B_k}}{w_k} : w_k > 0 \text{ und } k \in \{1, \dots, m\} \right\}.$$

Wir haben $\bar{x}_B = (\bar{x}_1, \bar{x}_6)^\top = (0, 0)^\top$. Da wieder alle Basisvariablen den Wert 0 haben und $w_2 = 2.5 > 0$ der größte Wert ist, entfernen wir gemäß unserer Vorgaben die Variable $x_{B_2} = x_6$ aus der Basis.

- (5) **Update:** Wir erhalten die neue Basis $B = (1, 2)$ mit der zugehörigen Basislösung $\bar{x}_B = (\bar{x}_1, \bar{x}_2)^\top = (0, 0)^\top$, $\bar{x}_N = (\bar{x}_3, \bar{x}_4, \bar{x}_5, \bar{x}_6)^\top = (0, 0, 0, 0)^\top$.

Auch die zweite Iteration ist also degeneriert und wir haben keinen Fortschritt in Bezug auf die Zielfunktion gemacht. Die eigentliche Lösung $(0, 0, 0, 0, 0, 0)^\top$ hat sich bisher überhaupt nicht verändert, nur die Aufteilung in Basis- und Nichtbasisvariablen wurde modifiziert. Machen wir weiter:

- (1) **BTRAN:** Löse $\bar{y}^\top A_B = c_B^\top$, d. h.

$$\bar{y}^\top \begin{bmatrix} 0.4 & 0.2 \\ -7.8 & -1.4 \end{bmatrix} = (-2.3, -2.15) \implies \bar{y}^\top = (-13.55, -0.4).$$

- (2) **Pricing:** Berechne $z_N = c_N - A_N^\top \bar{y}$:

$$z_N = \begin{pmatrix} 13.55 \\ 0.4 \\ 0 \\ 0 \end{pmatrix} - \begin{bmatrix} -1.4 & 7.8 \\ -0.2 & 0.4 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} -13.55 \\ -0.4 \end{pmatrix} = \begin{pmatrix} -2.3 \\ -2.15 \\ 13.55 \\ 0.4 \end{pmatrix}.$$

Gemäß unsere Festlegungen wählen wir $j = 3$ mit $z_3 = -2.3$ und die zugehörige Nichtbasisvariable x_3 wird in die Basis aufgenommen.

In diesem Moment halten wir kurz inne und vergleichen die aktuelle Situation mit der Situation in der ersten Iteration. Die beiden Vektoren z_N stimmen überein (lediglich die Indizes der zugehörigen Variablen haben sich geändert). Anstelle im **FTRAN**-Schritt das Gleichungssystem $A_B w = A_j$ zu lösen (also $w = A_B^{-1} A_j$ zu berechnen), berechnen wir wie in der ersten Iteration $A_B^{-1} A_N$:

$$\begin{aligned} A_B^{-1} A_N &= \begin{bmatrix} 0.4 & 0.2 \\ -7.8 & -1.4 \end{bmatrix}^{-1} \begin{bmatrix} -1.4 & -0.2 & 1 & 0 \\ 7.8 & 0.4 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.4 & 0.2 & -1.4 & -0.2 \\ -7.8 & -1.4 & 7.8 & 0.4 \end{bmatrix}. \end{aligned} \tag{12.3}$$

Wenn wir Gleichung 12.3 mit der Situation in der ersten Iteration vergleichen (siehe Gleichung (12.2)), so sehen wir, dass sich wie im Vektor z_N nur die Namen der zugehörigen Variablen geändert haben. Genauer gesagt, wir haben eine zyklische Verschiebung der Variablen um zwei Stellen nach rechts: waren in der ersten Iteration von den sechs Variablen $(x_1, x_2, x_3, x_4, x_5, x_6)$ die letzten beiden Variablen Basisvariablen und der Rest Nichtbasisvariablen, so sind jetzt x_1, x_2 die Basisvariablen. Die aktuelle Iteration verläuft also genauso wie die erste, die nächste wie die zweite etc. Man sieht nun leicht ein, dass wir nach weiteren vier Iterationen wieder die Ausgangsbasis $B = (5, 6)$ erhalten. Die Simplex-Methode terminiert also bei diesem Beispiel nicht.

12.1.2 Kreiseln

Definition 12.2 (Kreiseln). Wir sagen, dass der Simplex-Algorithmus *kreiselt*, wenn er eine Basis wiederholt.

Wie der folgende Satz zeigt, ist Kreiseln die einzige Möglichkeit, die dazu führen kann, dass die Simplex-Methode nicht terminiert.

Satz 12.3. Wenn der Simplex-Algorithmus nicht terminiert, so kreiselt er.

Beweis. Eine Basis besteht aus m Basisvariablen. Es gibt nur endlich viele, genauer gesagt $\binom{n}{m}$, Möglichkeiten m Variablen aus n auszuwählen. Daher gibt es auch nur maximal $\binom{n}{m}$ mögliche Basen. Wenn der Simplex-Algorithmus unendlich viele Iterationen durchführt, muss daher eine Basis wiederholt werden. Also muss der Algorithmus kreiseln. \square

Satz 12.4. Das lineare Problem (9.13) in Standardform besitze zulässige Lösungen und erfülle die Eigenschaft, dass für alle primal zulässigen Basen B die entsprechende Basislösung $(\bar{x}_B, \bar{x}_N)^\top$ nicht degeneriert ist, d. h. $\bar{x}_B = A_B^{-1}b > 0$. Dann terminiert der Simplex-Algorithmus 20 nach endlich vielen Schritten.

Beweis. Wir betrachten zwei aufeinanderfolgende Basislösungen \bar{x}' und \bar{x}'' (zu Basen B' und B''). In Abschnitt 9.2 haben wir bereits gezeigt, dass

$$c^\top \bar{x}'' = c^\top \bar{x}' + \gamma z_j \leq c^\top \bar{x}', \quad (12.4)$$

gilt (vgl. System (9.18)). Unter den Voraussetzungen des Satzes gelten $z_j < 0$ und $0 < \gamma$, woraus

$$c^\top \bar{x}'' < c^\top \bar{x}'$$

folgt.

Ist nun B_1, B_2, \dots eine Folge von Basen, die von Algorithmus 20 erzeugt werden, so folgt daraus, dass keine Basis innerhalb der Folge doppelt auftreten kann, da die Zielfunktionswerte streng monoton fallen. Da es nur endlich viele Basen gibt, folgt damit die Behauptung. \square

12.1.3 Die Regel von Bland

Im Fall von degenerierten Basislösungen kann die Simplex-Methode kreiseln. Wir zeigen nun, dass wir dieses Kreiseln durch geeignete Wahl der eintretenden bzw. die Basis verlassenden Variablen verhindern können. Nach der *Regel von Bland* [3] wählt man sowohl für die eintretende als auch für die verlassende Variable unter allen möglichen Variablen diejenige mit dem kleinsten Index:

Eintretende Variable: Falls es beim Pricing mehrere Nichtbasisvariablen j mit $z_j < 0$ gibt, so wählen wir diejenige mit dem kleinsten Index j aus, um in die Basis aufgenommen zu werden.

Verlassende Variable: Falls es beim Ratio-Test mehrere Basisvariablen gibt, für die das Minimum angenommen wird, so entfernen wir die Basisvariable x_{B_k} mit dem kleinsten Index B_k .

Wir wollen nun den Satz formulieren, dass die Regel von Bland Kreisel verhindern.

Satz 12.5. Werden eintretende und verlassende Variablen nach der Regel von Bland gewählt, so terminiert die Simplex-Methode nach endlich vielen Iterationen entweder mit einer Optimallösung oder der Information, dass das Problem (12.1) unbeschränkt ist.

Beweis. Siehe Chvátal [5]. □

Beispiel 36. Wir betrachten nochmal das Beispiel aus Abschnitt 12.1.1 und wenden jetzt das Simplex-Verfahren mit der Regel von Bland an.

Die erste Iteration verläuft genau gleich und wir erhalten die neue Basis $B = (1, 6)$. In der zweiten Iteration bestimmen wir im FTRAN-Schritt den Vektor $w = (0.5, 2.5)^\top$. In dem Beispiel haben wir den Index mit dem maximalen Eintrag, also $B_2 = 6$ gewählt. Nach der Regel von Bland wählen wir jetzt aber den kleinsten möglichen Index, also $B_1 = 1$.

Die neue Basis für die dritte Iteration ist dann $B = (2, 6)$. Die an der dritten Iteration beteiligten Matrizen und Vektoren sind

$$A_B = \begin{bmatrix} 0.2 & 0 \\ -1.4 & 1 \end{bmatrix}, \quad A_N = \begin{bmatrix} 0.4 & -1.4 & -0.2 & 1 \\ -7.8 & 7.8 & 0.4 & 0 \end{bmatrix},$$

$$c_B = \begin{pmatrix} -2.15 \\ 0 \end{pmatrix}, \quad c_N = \begin{pmatrix} -2.3 \\ 13.55 \\ 0.4 \\ 0 \end{pmatrix}.$$

Damit liefert der BTRAN-Schritt den Vektor $\bar{y} = (-10.75, 0)^\top$ und der Vektor der reduzierten Kosten ist $z_N = (2, -1.5, -1.75, 10.75)^\top$. Nach der Regel von Bland kommt also der Index 3 in die Basis. FTRAN liefert dann mit $A_j = (-1.4, 7.8)^\top$ den Vektor $w = (-7, -2)^\top < 0$. Da alle Einträge von w negativ sind, terminiert das Simplex-Verfahren an dieser Stelle mit der Meldung, dass das LP unbeschränkt ist. △

Aus unseren Ergebnissen zur Regel von Bland können wir jetzt einen zweiten Fundamentalsatz der linearen Optimierung beweisen.

Satz 12.6 (Fundamentalsatz der linearen Optimierung II). Für das lineare Problem in Standardform

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0 \end{aligned} \tag{12.5}$$

mit $\text{Rang } A = m$ gilt genau eine der folgenden Aussagen:

- (i) Das Problem (12.5) ist unzulässig.
- (ii) Das Problem (12.5) ist unbeschränkt, d. h., es gilt

$$\min \{c^\top x : Ax = b, x \geq 0\} = -\infty.$$

- (iii) Das Problem (12.5) hat eine endliche Optimallösung, die zugleich Basislösung ist.

Beweis. Offenbar kann nur höchstens eine der drei Aussagen (i)–(iii) für das lineare Problem (12.5) gelten. Wir nehmen an, dass (i) und (ii) nicht gelten und zeigen, dass dann (iii) folgt. Da (i) nicht gilt, hat das Problem (12.5) eine zulässige Lösung. Nach dem ersten Fundamentalsatz 10.9 aus Kapitel 10.3 hat das Problem dann aber auch eine zulässige Basis B mit zugehöriger Basislösung \bar{x} . Wir wenden die Simplex-Methode mit der Regel von Bland auf das Problem (12.5) an, wobei wir mit der Basis B starten. Das Verfahren terminiert gemäß Satz 12.5 nach endlich vielen Schritten. Da (ii) nicht gilt, kann das Verfahren nicht mit der Information abbrechen, dass das Problem unbeschränkt ist. Also muss das Verfahren mit einer optimalen Lösung abbrechen, die gleichzeitig eine Basislösung ist, also folgt (iii). \square

Der letzte Satz sagt insbesondere aus, dass ein zulässiges LP, das nicht unbeschränkt ist, immer lösbar ist. Dies ist im Allgemeinen, also beispielsweise bei nichtlinearen Problemen, nicht der Fall. So ist

$$\min_x \quad \frac{1}{x} \quad \text{s.t.} \quad x \geq 1$$

zulässig, beschränkt (durch 0), aber es gibt kein x , für das das Optimum 0 angenommen wird.

12.2 Die Effizienz der Simplex-Methode

Aus praktischer Sicht stellt sich natürlich nicht nur die Frage, ob das Verfahren terminiert, sondern man ist auch an Aussagen über das Worst-Case-Verhalten

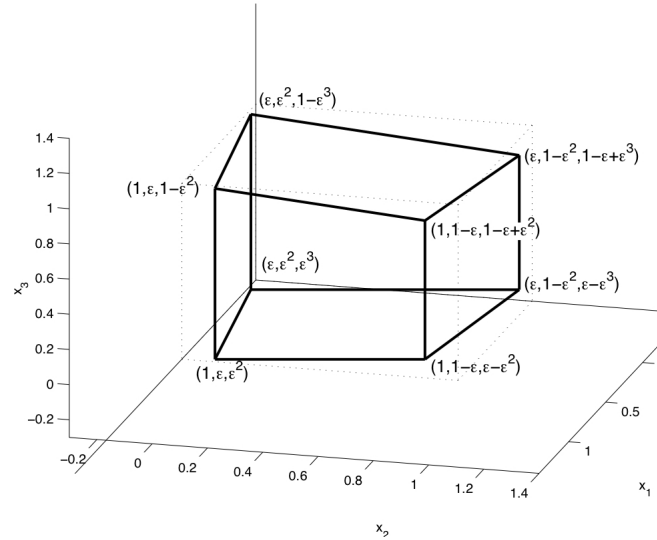


Abbildung 12.1: Der Klee-Minty-Würfel aus Beispiel 37

des Simplex-Verfahrens interessiert. Insbesondere stellt sich die Frage, ob es Pricing-Regeln und Regeln für die austretenden Basisvariablen gibt, die zu einer polynomiellen Worst-Case-Komplexität des Simplex-Verfahrens führen. Bislang sind solche Regeln nicht bekannt.

Beispiel 37. Für ε mit $0 < \varepsilon < 1/2$ sei das Problem

$$\begin{aligned} \min_x \quad & -x_n \\ \text{s.t.} \quad & \varepsilon \leq x_1 \leq 1, \\ & \varepsilon x_{j-1} \leq x_j \leq 1 - \varepsilon x_{j-1} \quad \text{für alle } j = 2, \dots, n \end{aligned}$$

gegeben. Die zulässige Menge ist in Abbildung 12.1 zu sehen. Dieses Beispiel ist in der Literatur unter dem Namen *Klee-Minty-Würfel* bekannt (benannt nach seinen Entdeckern). Man kann zeigen, dass der Simplex-Algorithmus 2^n Iterationen zur Lösung des obigen linearen Problems benötigt. Das heißt, das Verfahren läuft alle Ecken ab. Einen Beweis dieser Aussage findet man beispielsweise in Papadimitriou und Steiglitz [16]. \triangle

12.3 Die Phase-1 des Simplex-Verfahrens

Nachdem wir die Frage geklärt haben, ob bzw. wann das Simplex-Verfahren terminiert, wenden wir uns nun der Frage zu, wie wir eine erste zulässige

Basislösung für ein lineares Problem in Standardform

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0 \end{aligned} \tag{12.6}$$

finden bzw. feststellen, dass es keine zulässige Lösung gibt. Wir verzichten in diesem Abschnitt auf die Annahme, dass die Matrix A vollen Zeilenrang hat, d. h., wir lassen nun $\text{Rang}(A) \leq m$ zu. Dafür nehmen wir aber an, dass der Vektor b auf der rechten Seite der Gleichungen in (12.6) die Bedingung

$$b \geq 0 \tag{12.7}$$

erfüllt. Dies können wir einfach dadurch erreichen, dass wir eine Zeile $A_{i,\cdot}x = b_i$ mit $b_i < 0$ mit -1 multiplizieren und damit wie gewünscht $-A_{i,\cdot}x = -b_i \geq 0$ erhalten.

Wir betrachten nun das folgende Hilfsproblem, das ebenfalls ein lineares Problem in Standardform ist:

$$\begin{aligned} \min_{x,y} \quad & \sum_{i=1}^m y_i \\ \text{s.t.} \quad & Ax + y = b, \\ & x, y \geq 0. \end{aligned} \tag{12.8}$$

Dabei sei

$$e := (1, \dots, 1)^\top \in \mathbb{R}^m$$

der Vektor, der aus lauter Einsen besteht. Mit $u = (x, y)^\top$, $d = (0, e)^\top$ und $D = [A \ I]$ erhalten wir die zu (12.8) äquivalente Formulierung

$$\begin{aligned} \min_u \quad & d^\top u \\ \text{s.t.} \quad & Du = b, \\ & u \geq 0. \end{aligned}$$

Man beachte, dass die Matrix $D = [A \ I]$ der Nebenbedingungen des Problems (12.8) die Bedingung $\text{Rang } D = m$ erfüllt, da D die $m \times m$ -Einheitsmatrix als Teilmatrix enthält. Sind insbesondere $\{1, 2, \dots, n, n+1, \dots, n+m\}$ die Spaltenindizes von D , dann ist $B = (n+1, \dots, n+m)$ eine Basis von D mit $D_B = I$. Die Variablen des Vektors y werden als *künstliche Variablen* bezeichnet, die eigentlichen Variablen x unseres ursprünglichen Problems bezeichnen wir als *Strukturvariablen*. Die Basis $B = (n+1, \dots, n+m)$ enthält also genau die Indizes der künstlichen Variablen. Die zu B gehörende Basislösung ist $\bar{u} = (\bar{u}_B, \bar{u}_N)^\top$ mit

$$\begin{aligned} \bar{u}_B &= D_B^{-1}b = Ib = b \geq 0, \\ \bar{u}_N &= 0. \end{aligned}$$

Wegen der Annahme $b \geq 0$ aus Ungleichung (12.7) ist $B = (n+1, \dots, n+m)$ also eine zulässige Basis für das lineare Problem (12.8) und wir können den Simplex-Algorithmus 20 mit der Startbasis B auf das Problem (12.8) anwenden. Wenn wir die Regel von Bland aus dem letzten Abschnitt verwenden, so terminiert die Simplex-Methode nach Satz 12.5 entweder mit einer optimalen Lösung des linearen Problems (12.8) oder der Information, dass das Problem (12.8) unbeschränkt ist. Da für jede zulässige Lösung $u = (x, y)^\top$ von Problem (12.8) wegen $y \geq 0$ auch

$$d^\top u = e^\top y = \sum_{i=1}^m y_i \geq 0$$

gilt, ist Problem (12.8) nach unten beschränkt. Es verbleibt also nur die Möglichkeit, dass das Simplex-Verfahren eine optimale Lösung findet, die auch Basislösung ist. Sei $u^* = (x^*, y^*)^\top$ diese Lösung und B^* die zugehörige optimale Basis. Wir unterscheiden die folgenden Fälle:

Fall 1: Es gilt $d^\top u^* = e^\top y^* > 0$.

In diesem Fall kann das Ausgangsproblem (12.6) keine zulässige Lösung besitzen. Wäre nämlich x eine solche zulässige Lösung, so wäre $(x, 0)^\top$ eine zulässige Lösung von Problem (12.8) mit Zielfunktionswert 0. Dies würde aber im Widerspruch dazu stehen, dass der optimale Zielfunktionswert $d^\top u^*$ von Problem (12.8) strikt größer als Null ist.

Fall 2: Es gilt $d^\top u^* = e^\top y^* = 0$.

Diesen Fall untergliedern wir in zwei Unterfälle:

(2a) Es gilt $B^* \cap \{n+1, n+2, \dots, n+m\} = \emptyset$. Das heißt, B^* enthält keine künstliche Variable. In diesem Fall ist $D_{B^*} = A_{B^*}$. Insbesondere ist A_{B^*} nicht singulär und die zu B^* gehörende Basislösung $\bar{x} = (\bar{x}_{B^*}, \bar{x}_{N^*})^\top$ mit $N^* = \{1, \dots, n\} \setminus B^*$ erfüllt

$$\begin{aligned}\bar{x}_{B^*} &= A_{B^*}^{-1} b = D_{B^*}^{-1} b = x_{B^*}^* \geq 0, \\ \bar{x}_{N^*} &= 0.\end{aligned}$$

Also ist B^* eine zulässige Basis für das Ausgangsproblem (12.6) und wir können B^* als Startbasis verwenden, um mit dem Simplex-Verfahren das Problem (12.6) zu lösen.

(2b) Es gilt $B^* \cap \{n+1, n+2, \dots, n+m\} \neq \emptyset$. Das heißt, B^* enthält mindestens eine künstliche Variable. Wir versuchen nun, die künstlichen Variablen in der Basis durch Strukturvariablen zu ersetzen. Wir starten mit $B = B^*$, $N = \{1, \dots, n+m\} \setminus B$ und definieren

$$\begin{aligned}B_x &= B \cap \{1, \dots, n\}, & N_x &= N \cap \{1, \dots, n\}, \\ B_y &= B \cap \{n+1, \dots, n+m\}, & N_y &= N \cap \{n+1, \dots, n+m\}.\end{aligned}$$

Aufgrund des optimalen Zielfunktionswerts 0 gilt $y^* = 0$. Insbesondere besitzen alle künstlichen Variablen in der Basis den Wert 0 und die Basis B ist degeneriert. Sei $j \in \{1, \dots, n\} \cap N$ der Index einer Strukturvariablen, die nicht in der Basis ist. Wir betrachten den Vektor $w = D_B^{-1}D_j = D_B^{-1}A_j$. Falls $w_i \neq 0$, $i \in \{1, \dots, m\}$, $B_i \in B_y$, so führen wir eine (leicht modifizierte) Iteration des Simplex-Verfahrens mit u_j als eintretender und u_{B_i} als austretender Variable durch. Da $u_{B_i}^* = 0$ und $w_i \neq 0$ gilt

$$\gamma' = \min \left\{ \frac{u_{B_i}^*}{w_i} : w_i \neq 0 \text{ und } i \in \{1, \dots, m\} \right\} = 0.$$

Dabei ist $B \setminus \{B_i\} \cup \{j\}$ eine zulässige Basis, wobei sich in dieser degenerierten Iteration die Basislösung (bis auf die Zuordnung der Basis- und Nichtbasisvariablen) nicht ändert. Wir setzen also $B \leftarrow B \setminus \{B_i\} \cup \{j\}$ und $N \leftarrow N \setminus \{j\} \cup \{B_i\}$. Solange wir also Indizes $j \in \{1, \dots, n\} \cap N$ und $B_i \in B_y$ finden, so dass für $w = D_B^{-1}A_j$ die Bedingung $w_i \neq 0$ gilt, können wir wie oben beschrieben eine künstliche Variable gegen eine Strukturvariable in der Basis austauschen. Entweder können wir damit alle künstlichen Variablen aus der Basis entfernen und landen damit schließlich in Fall (2a) oder es gilt

$$(D_B^{-1}A_j)_i = 0 \text{ für alle } j \in \{1, \dots, n\} \cap N \text{ und } B_i \in B_y. \quad (12.9)$$

In diesem Fall können wir die zu B_y gehörenden Zeilen aus $Ax = b$ streichen, denn

$$(D_B^{-1}(A, b))_i = (D_B^{-1})_i(A, b) = 0.$$

Somit sind die Zeilen von (A, b) linear abhängig und A hatte nicht vollen Rang. Damit reduziert sich $Ax = b$ mit $I := \{i \in \{1, \dots, m\} : B_i \in B_y\}$, $S := \{1, \dots, m\} \setminus I$ zu $A_S x = b_S$ mit regulärer Matrix $D_{S, B_x} = A_{S, B_x}$. Darüber hinaus gilt für die zugehörige Basislösung

$$\bar{x}_{B_x} = \bar{u}_{B_x} = D_{S, B_x}^{-1} b_S \geq 0$$

und $\bar{x}_{N_x} = 0$. Daher ist B_x eine zulässige Basis und damit das, was wir erreichen wollten.

Kapitel 13

Der duale Simplex-Algorithmus

Die Grundidee des dualen Simplex-Algorithmus ist es, den primalen Simplex-Algorithmus 20 auf das duale Problem anzuwenden, ohne das primale Problem explizit zu dualisieren. Historisch war die Entwicklung jedoch anders: Man dachte zunächst, man hätte ein neues Verfahren gefunden, bevor man den obigen Zusammenhang erkannte.¹ Betrachten wir nochmals

$$\begin{array}{ll} \min_{x} & c^\top x \\ \text{s.t.} & Ax = b, \\ & x \geq 0 \end{array} \quad (\text{P})$$

und das zugehörige duale Problem

$$\begin{array}{ll} \max_{y,z} & b^\top y \\ \text{s.t.} & A^\top y + z = c, \\ & z \geq 0, \end{array} \quad (\text{D})$$

wobei wir das duale Problem durch Einführung der Schlupfvariablen z fast in Standardform überführt haben. Der einzige Unterschied zur Standardform sind die fehlenden Vorzeichenbeschränkungen an die dualen Variablen y .

Wir gehen im Folgenden wieder davon aus, dass A vollen Zeilenrang hat, d. h., $\text{Rang}(A) = \text{Rang}(A^\top) = m \leq n$. Mit $D = [A^\top, I] \in \mathbb{K}^{n \times (m+n)}$ erhält

¹Ein lesenswertes Interview mit Bob Bixby zur Geschichte der Entwicklung der Algorithmen für LPs (und auch für gemischt-ganzzahlige lineare Probleme) können Sie bei Kaibel u. a. [10] nachlesen.

(D) die Form

$$\begin{aligned} \max_{y,z} \quad & b^\top y \\ \text{s.t.} \quad & D \begin{pmatrix} y \\ z \end{pmatrix} = c, \\ & z \geq 0. \end{aligned}$$

Eine Basis H des dualen Problems in (D) hat also die Mächtigkeit n mit

$$H \subseteq \{1, 2, \dots, m, m+1, \dots, m+n\}.$$

Für die folgenden Überlegungen benötigen wir den (hier nicht bewiesenen) nächsten Satz.

Satz 13.1. Sei $A \in \mathbb{K}^{m \times n}$ mit vollem Zeilenrang und H eine zulässige Basis von (D). Dann ist (D) unbeschränkt oder es existiert eine optimale Basis H^* mit $\{1, \dots, m\} \subseteq H^*$.

Im Gegensatz zur Anwendung von Algorithmus 20 auf (P) haben wir bei Anwendung des Simplex-Verfahrens auf (D) die Besonderheit, dass nicht alle Variablen Vorzeichenbeschränkungen unterliegen, da die y -Variablen im dualen Problem freie Variablen sind. Satz 13.1 sagt aus, dass wir o. B. d. A. davon ausgehen können, dass alle Variablen in y in einer optimalen Basis H für (D) enthalten sind.

Da $|H| = n$ und y aus m Variablen besteht, muss H noch $n - m$ Variablen aus z enthalten. Wir bezeichnen mit $N \subseteq \{1, \dots, n\}$ diese $n - m$ Variablen und mit $B = \{1, \dots, n\} \setminus N$ die m Indizes für z , die nicht in der Basis sind. Da

$$D_H = \begin{bmatrix} (A_B)^\top & 0 \\ (A_N)^\top & I_N \end{bmatrix}$$

regulär ist, muss also auch $(A_B)^\top$ regulär sein. Also ist A_B regulär. Zur Vereinfachung der Notation schreiben wir im Folgenden A_B^\top anstelle von $(A_B)^\top$ und für $(A_N)^\top$ entsprechend A_N^\top . Nun gilt

$$\begin{aligned} D \begin{pmatrix} y \\ z \end{pmatrix} = c, \quad z \geq 0 &\iff A^\top y + z = c, \quad z \geq 0 \\ &\iff A_B^\top y + z_B = c_B, \quad z_B \geq 0, \\ &\quad A_N^\top y + z_N = c_N, \quad z_N \geq 0 \\ &\iff y = A_B^{-\top} (c_B - z_B), \\ &\quad z_N = c_N - A_N^\top A_B^{-\top} (c_B - z_B), \\ &\quad z_N, z_B \geq 0. \end{aligned} \tag{13.1}$$

Setzen wir die Nichtbasisvariablen aus $z_B = 0$, so ist H (primal) zulässig für (D), falls

$$z_N = c_N - A_N^\top A_B^{-\top} c_B \geq 0$$

gilt, was äquivalent dazu ist, dass B dual zulässig ist. Man beachte außerdem, dass H durch N und B eindeutig festgelegt ist. Daher ist es üblich von einer dual zulässigen Basis B für (P) zu sprechen und weniger von einer (primal) zulässigen Basis H für (D).

Betrachten wir also eine dual zulässige Basis B (Beachte: B sind die Nichtbasisvariablen im dualen Problem und N sind die Basisvariablen im dualen Problem) mit

$$\begin{aligned} z_N &= c_N - A_N^\top A_B^{-\top} c_B \geq 0, \\ z_B &= 0 \end{aligned}$$

und wenden Algorithmus 20 auf (D) an.

- (1) **BTRAN**: Im primalen Simplex-Verfahren wurde hier $\bar{y}^\top A_B = c_B^\top$ berechnet. Im dualen entspricht dies

$$\begin{aligned} \bar{x}^\top D_H = (b^\top, 0) &\iff \bar{x}^\top \begin{bmatrix} A_B^\top & 0 \\ A_N^\top & I_N \end{bmatrix} = (b^\top, 0) \\ &\iff \bar{x}_B^\top A_B^\top + \bar{x}_N^\top A_N^\top = b^\top, \quad \bar{x}_N^\top = 0, \\ &\iff \bar{x}_N = 0, \quad A_B \bar{x}_B = b, \\ &\iff \bar{x}_N = 0, \quad \bar{x}_B = A_B^{-1} b. \end{aligned}$$

- (2) Beim Pricing berechnen wir die reduzierten Kosten. Als wir den primalen Simplex auf das primale Problem angewendet haben, haben wir $c_N - A_N^\top \bar{y}$ berechnet. Dies entspricht angewandt auf das duale Problem

$$0_B - I_B \bar{x}_B = -\bar{x}_B.$$

Das Pricing basiert dann auf der Gleichung

$$b^\top y = b^\top (A_B^{-\top} (c_B - z_B)) = b^\top A_B^{-\top} c_B - (A_B^{-1} b)^\top z_B.$$

Da (D) ein Maximierungsproblem ist, ist H (bzw. B) dual optimal (bzw. B ist primal zulässig), falls $A_B^{-1} b = \bar{x}_B \geq 0$ gilt. Andernfalls wählen wir einen Index B_i mit $\bar{x}_{B_i} < 0$.

- (3) **FTRAN**: Im primalen haben wir bei diesem Schritt das System $A_B w = A_j$ gelöst. Dies entspricht im dualen

$$\begin{aligned} D_H \begin{pmatrix} w \\ \alpha_N \end{pmatrix} = \begin{pmatrix} e_i \\ 0 \end{pmatrix} &\iff \begin{bmatrix} A_B^\top & 0 \\ A_N^\top & I_N \end{bmatrix} \begin{pmatrix} w \\ \alpha_N \end{pmatrix} = \begin{pmatrix} e_i \\ 0 \end{pmatrix} \\ &\iff A_B^\top w = e_i \text{ und } A_N^\top w + \alpha_N = 0 \\ &\iff A_B^\top w = e_i \text{ und } \alpha_N = -A_N^\top w. \end{aligned}$$

- (4) **Ratio-Test:** Wir überprüfen, ob $\alpha_N \leq 0$ ist. Man beachte, dass das Vorzeichen von w egal ist, da die y freie Variablen sind. Ist dies der Fall, so ist (D) unbeschränkt, d. h. das primale Problem ist unbeschränkt. Andernfalls setze

$$\gamma = \frac{z_j}{\alpha_j} = \min \left\{ \frac{z_k}{\alpha_k} : \alpha_k > 0, k \in N \right\}$$

mit $j \in N$ und $\alpha_j > 0$. Damit verlässt nun j die duale Basis H bzw. tritt in die Basis B ein.

Dies zusammen ergibt Algorithmus 21.

Algorithmus 21 Der duale Simplex-Algorithmus

Eingabe: Eine dual zulässige Basis B und $z_N = c_N - A_N^\top A_B^{-\top} c_B \geq 0$.

Ausgabe: Eine Optimallösung \bar{x} für (P) bzw. eine Optimallösung \bar{y} für das zu (P) duale Problem (aber nicht in Standardform) bzw. eine Optimallösung \bar{y} , $z_N, z_B = 0$ für (D) oder die Meldung das primale Problem (P) unzulässig bzw. das duale Problem (D) unbeschränkt ist.

- 1: **BTRAN:** Löse $A_B \bar{x}_B = b$.
- 2: **Pricing:**
- 3: **if** $\bar{x}_B \geq 0$ **then**
- 4: **return** dual optimale Basis B , duale Optimallösung $\bar{y} = A_B^{-\top} c_B$.
- 5: Wähle ein $i \in \{1, \dots, m\}$ mit $x_{B_i} < 0$, d. h., B_i verlässt die Basis B .
- 6: **FTRAN:** Löse $A_B^\top w = e_i$ und berechne $\alpha_N = -A_N^\top w$.
- 7: **if** $\alpha_N \leq 0$ **then**
- 8: **return** "(D) ist unbeschränkt und (P) ist unzulässig."
- 9: **Ratio-Test:** Berechne

$$\gamma = \frac{z_j}{\alpha_j} = \min \left\{ \frac{z_k}{\alpha_k} : \alpha_k > 0, k \in N \right\}$$

mit $j \in N$ und $\alpha_j > 0$. Die Variable j kommt in die Basis.

- 10: **Update:** Setze

$$z_N = z_N - \gamma \alpha_N, \quad N = N \setminus \{j\} \cup \{B_i\}, \quad B_i = j, \quad z_{B_i} = \gamma.$$

- 11: Gehe zu Schritt 1.
-

Die Korrektheit von Algorithmus 21 und alle weiteren Konsequenzen gelten gemäß der Abschnitte über das primale Simplexverfahren und der Dualitätssätze. Dabei wird die Tatsache genutzt, dass Algorithmus 21 nichts anderes ist als die Anwendung von Algorithmus 20 auf (D).

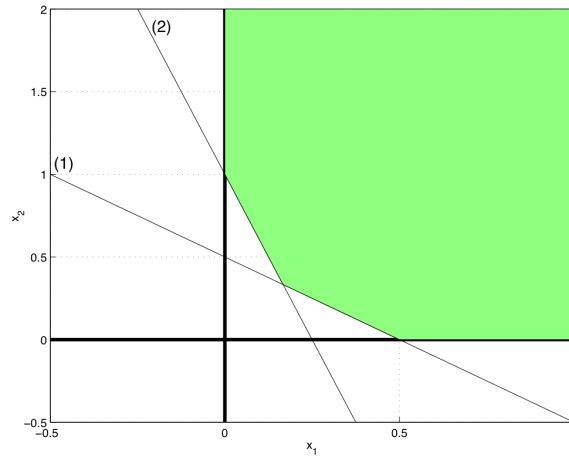


Abbildung 13.1: Grafische Darstellung zu Beispiel 38

Beispiel 38. Wir betrachten das LP

$$\begin{aligned} \min_x \quad & 2x_1 + x_2 \\ \text{s.t.} \quad & -x_1 - x_2 \leq -\frac{1}{2}, \\ & -4x_1 - x_2 \leq -1, \\ & x_1, x_2 \geq 0. \end{aligned}$$

Die zulässige Menge ist in Abbildung 13.1 dargestellt. In Standardform lautet das LP

$$\begin{aligned} \min_x \quad & 2x_1 + x_2 \\ \text{s.t.} \quad & -x_1 - x_2 + x_3 = -\frac{1}{2}, \\ & -4x_1 - x_2 + x_4 = -1, \\ & x_1, x_2, x_3, x_4 \geq 0. \end{aligned}$$

Wir starten mit $B = (3, 4)$, $N = \{1, 2\}$. Es gilt (mit $c_B^\top = (0, 0)^\top$)

$$z_N = \begin{pmatrix} 2 \\ 1 \end{pmatrix} - A_N^\top A_B^{-\top} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \geq 0.$$

Damit ist B dual zulässig.

(1.1) BTRAN

$$\begin{aligned} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \bar{x}_B &= \begin{pmatrix} -1/2 \\ -1 \end{pmatrix} \\ \implies \bar{x}_B = \begin{pmatrix} \bar{x}_3 \\ \bar{x}_4 \end{pmatrix} &= \begin{pmatrix} -1/2 \\ -1 \end{pmatrix}, \quad \bar{x}_N = \begin{pmatrix} \bar{x}_1 \\ \bar{x}_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}. \end{aligned}$$

(1.2) Pricing: Es gilt $\bar{x} \not\geq 0$. Wir wählen daher $i = 2$ mit $\bar{x}_{B_2} = \bar{x}_4 = -1 < 0$.

(1.3) FTRAN: Wir berechnen

$$A_B^\top w = e_2 \iff \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} w = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \iff w = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

und

$$\alpha_N = -A_N^\top w = \begin{bmatrix} 1 & 4 \\ 1 & 1 \end{bmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}.$$

(1.4) Ratio-Test: Wir berechnen

$$\gamma = \min \left\{ \frac{2}{4}, \frac{1}{1} \right\} = \frac{1}{2} \quad \text{mit } j = 1.$$

(1.5) Update

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = z_N = \begin{pmatrix} 2 \\ 1 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1/2 \end{pmatrix},$$

$$N = \{2, 4\}, \quad B = (3, 1) \quad z_4 = \frac{1}{2}.$$

(2.1) BTRAN: Wir berechnen

$$\begin{bmatrix} 1 & -1 \\ 0 & -4 \end{bmatrix} \bar{x}_B = \begin{pmatrix} -1/2 \\ -1 \end{pmatrix}$$

$$\implies \bar{x}_B = \begin{pmatrix} \bar{x}_3 \\ \bar{x}_1 \end{pmatrix} = \begin{pmatrix} -1/4 \\ 1/4 \end{pmatrix}, \quad \bar{x}_N = \begin{pmatrix} \bar{x}_2 \\ \bar{x}_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

(2.2) Pricing: Es gilt $\bar{x}_B \not\geq 0$ und wir wählen $i = 1$ mit $\bar{x}_{B_1} = \bar{x}_3 = -1/4 < 0$.

(2.3) FTRAN: Wir berechnen

$$A_B^\top w = e_1 \iff \begin{bmatrix} 1 & 0 \\ -1 & -4 \end{bmatrix} w = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \implies w = \begin{pmatrix} 1 \\ -1/4 \end{pmatrix}$$

und

$$\alpha_N = -A_N^\top w = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} 1 \\ -1/4 \end{pmatrix} = \begin{pmatrix} 3/4 \\ 1/4 \end{pmatrix}.$$

(2.4) Ratio-Test: Wir berechnen

$$\gamma = \min \left\{ \frac{1/2}{3/4}, \frac{1/2}{1/4} \right\} = \frac{2}{3} \quad \text{mit } j = 2.$$

(2.5) Update:

$$z_N = \begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix} - \frac{2}{3} \begin{pmatrix} 3/4 \\ 1/4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1/3 \end{pmatrix},$$
$$N = \{3, 4\}, \quad B = (2, 1), \quad z_{B_1} = z_3 = \frac{2}{3}.$$

(3.1) BTRAN: Noch einmal ...

$$\begin{bmatrix} -1 & -1 \\ -1 & -4 \end{bmatrix} \bar{x}_B = \begin{pmatrix} -1/2 \\ -1 \end{pmatrix}$$
$$\implies \bar{x}_B = \begin{pmatrix} \bar{x}_2 \\ \bar{x}_1 \end{pmatrix} = \begin{pmatrix} 1/3 \\ 1/6 \end{pmatrix}, \quad \bar{x}_N = \begin{pmatrix} \bar{x}_3 \\ \bar{x}_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

(3.2) Der Pricing-Schritt ergibt schließlich, dass $\bar{x}_B \geq 0$ gilt und

$$\bar{x}_1 = 1/6, \quad \bar{x}_2 = 1/3$$

optimal ist.

\triangle

Kapitel 14

Sensitivitätsanalyse II

Wir sind bisher davon ausgegangen, dass die Daten A, b und c eines linearen Problems der Form (P) fest vorgegeben sind. Häufig ist es jedoch der Fall, dass sich diese Daten im Lauf der Zeit ändern, wenn Bedingungen und/oder Variablen hinzukommen und damit ein sehr ähnliches Problem wie das vorherige gelöst werden muss. Gehen wir einmal davon aus, dass wir bereits ein lineares Problem der Form (P)

$$\begin{array}{ll} \min_x & c^\top x \\ \text{s.t.} & Ax = b, \\ & x \geq 0 \end{array}$$

gelöst haben und eine optimale Basis B mit der Lösung $\bar{x}_B = A_B^{-1}b$ und $\bar{x}_N = 0$ kennen. Wir wollen nun folgende Änderungen des linearen Problems betrachten und uns überlegen, wie wir ausgehend von B möglichst schnell eine Optimallösung des modifizierten Problems finden können:

1. Änderung der rechten Seite b ,
2. Änderung der Zielfunktion c ,
3. Änderung eines Eintrags in der Matrix A ,
4. Hinzufügen einer neuen Spalte bzw. Variablen,
5. Hinzufügen einer neuen Zeile bzw. Nebenbedingung.

Den Fall einer Änderung der rechten Seite b haben wir bereits in Abschnitt 11.7 behandelt.

Änderung der Zielfunktion

Der Zielfunktionsvektor c ändert sich zu \tilde{c} . Dann ist B weiterhin primal zulässig, da $\bar{x}_B = A_B^{-1}b$ unverändert bleibt. Wegen der Änderung der Zielfunktion ändern sich die reduzierten Kosten zu

$$\tilde{z}_N = \tilde{c}_N - A_N^T A_B^{-T} \tilde{c}_B.$$

1. Fall: $\tilde{z}_N \geq 0$. Die Basis bleibt demnach auch dual zulässig, somit bleibt B optimal. Der Zielfunktionswert ändert sich eventuell.

2. Fall: $\exists i \in N : \tilde{z}_i < 0$. Die Basis B ist nicht mehr dual zulässig, also ist B nicht mehr optimal. Der primale Simplex-Algorithmus kann mit B als primal zulässiger Basis gestartet werden.

Änderung eines Eintrags in der Matrix A in einer Nichtbasisspalte

Die Basis B bleibt primal zulässig, da $\bar{x}_B = A_B^{-1}b$ unverändert bleibt. Aufgrund der Änderung in der Nichtbasisspalte $A_{\cdot j}$ zu $\tilde{A}_{\cdot j}$ ($j \in N$) ändern sich die reduzierten Kosten bezüglich j zu

$$\tilde{z}_j = c_j - c_B^T A_B^{-1} \tilde{A}_{\cdot j}.$$

1. Fall: $\tilde{z}_j \geq 0$. Die Basis bleibt demnach dual zulässig, somit bleibt B optimal. Der Zielfunktionswert ändert sich nicht.

2. Fall: $\tilde{z}_j < 0$. Die Basis B ist nicht mehr dual zulässig, also ist B nicht mehr optimal. Der primale Simplex-Algorithmus kann mit B als primal zulässiger Basis gestartet werden.

Bei einer Änderung der Matrix in einer Basisspalte ist keine Vorhersage möglich, da sich sowohl die Basislösung \bar{x}_B als auch die reduzierten Kosten ändern.

Hinzufügen einer neuen Variablen

Eine neue Variable x_{n+1} mit Vorzeichenbeschränkung $x_{n+1} \geq 0$, Zielfunktionskoeffizient $c_{n+1} \in \mathbb{R}$ und Spalte $A_{\cdot, n+1} \in \mathbb{R}^m$ wird zum Problem hinzugefügt.

Wir fügen x_{n+1} zu den Nichtbasisvariablen hinzu, also $\tilde{N} = N \cup \{n+1\}$. Die Basis bleibt primal zulässig, da $\bar{x}_B = A_B^{-1}b$ unverändert bleibt. Für x_{n+1} erhalten wir die reduzierten Kosten

$$\tilde{z}_{n+1} = c_{n+1} - c_B^T A_B^{-1} A_{\cdot, n+1}.$$

1. Fall: $\tilde{z}_{n+1} \geq 0$. Die Basis bleibt demnach dual zulässig, somit bleibt B optimal. Der Zielfunktionswert ändert sich nicht.

2. Fall: $\tilde{z}_{n+1} < 0$. Die Basis B ist nicht mehr dual zulässig, also ist B nicht mehr optimal. Der primale Simplex-Algorithmus kann mit B als primal zulässiger Basis gestartet werden. Im ersten Schritt wird die neue Variable in die Basis aufgenommen, da die reduzierten Kosten für alle anderen Variablen unverändert (d.h. nicht negativ) bleiben.

Hinzufügen einer neuen Nebenbedingung

Dieser Fall wird insbesondere bei der Lösung ganzzahliger Optimierungsprobleme von Bedeutung sein, die in den Vorlesungen Diskrete Optimierung I & II betrachtet werden.

Wir fügen dem obigen LP also die neue Ungleichung

$$A_{m+1,\cdot}x \geq b_{m+1}$$

hinzu und unterscheiden die folgenden Fälle.

1. Fall: Falls unsere alte Lösung \bar{x} auch diese neue Ungleichung erfüllt, ist nichts zu tun.

2. Fall: Die Ungleichung ist nicht erfüllt. Das heißt, unsere alte Basis B ist primal unzulässig für das neue Problem

$$\begin{aligned} \min_x \quad & c^\top x \\ \text{s.t.} \quad & Ax = b, \\ & A_{m+1,\cdot}x \geq b_{m+1}, \\ & x \geq 0. \end{aligned}$$

Wir führen eine neue Schlupfvariable $x_{n+1} \geq 0$ ein, die wir direkt in die neue Basis B' aufnehmen. Die neue Basismatrix zur Basis B' hat die Form

$$\begin{bmatrix} A_B & 0 \\ A_{m+1,B} & -1 \end{bmatrix} \in \mathbb{K}^{(m+1) \times (m+1)}.$$

Die zugehörige inverse Basismatrix lautet dann

$$\begin{bmatrix} A_B^{-1} & 0 \\ A_{m+1,B}A_B^{-1} & -1 \end{bmatrix} \in \mathbb{K}^{(m+1) \times (m+1)}$$

und es gilt

$$\begin{aligned}
\bar{x}_{B'} &= \begin{bmatrix} A_B^{-1} & 0 \\ A_{m+1,B}A_B^{-1} & -1 \end{bmatrix} \begin{pmatrix} b \\ b_{m+1} \end{pmatrix} \\
&= \begin{pmatrix} A_B^{-1}b \\ A_{m+1,B}A_B^{-1}b - b_{m+1} \end{pmatrix} \\
&= \begin{pmatrix} \bar{x}_B \\ A_{m+1,B}\bar{x}_B - b_{m+1} \end{pmatrix},
\end{aligned}$$

wobei zwar $\bar{x}_B \geq 0$, aber auch $A_{m+1,B}\bar{x}_B - b_{m+1} < 0$ gilt, da wir angenommen haben, dass \bar{x} nicht zulässig für die neue Ungleichung ist. Das heißt, die neue Basis B' ist nicht primal zulässig. Sie ist aber dual zulässig, denn es gilt

$$\begin{aligned}
&c_N^\top - c_{B'}^\top A_{B'}^{-1} \begin{pmatrix} A_N \\ A_{m+1,N} \end{pmatrix} \\
&= c_N^\top - (c_B^\top, 0) \begin{bmatrix} A_B^{-1} & 0 \\ A_{m+1,B}A_B^{-1} & -1 \end{bmatrix} \begin{pmatrix} A_N \\ A_{m+1,N} \end{pmatrix} \\
&= c_N^\top - (c_B^\top, 0) \begin{bmatrix} A_B^{-1}A_N \\ A_{m+1,B}A_B^{-1}A_N - A_{m+1,N} \end{bmatrix} \\
&= c_N^\top - c_B^\top A_B^{-1}A_N \geq 0.
\end{aligned}$$

Wir können daher mit dem dualen Simplex-Algorithmus starten und erhalten entweder eine Optimallösung oder aber die Meldung, dass das duale Problem unbeschränkt ist und somit gilt für das entsprechende Polyeder

$$P\left(\begin{bmatrix} A \\ A_{m+1,\cdot} \end{bmatrix}, \begin{pmatrix} b \\ b_{m+1} \end{pmatrix}\right) = \emptyset.$$

Im zweiten Fall hat die Hyperebene $A_{m+1,\cdot}x = b_{m+1}$ einen leeren Schnitt mit $P(A, b)$.

Teil III

Exkurs: Gemischt-ganzzahlige lineare Optimierung

Kapitel 15

Branch-and-Bound

Wir wollen in diesem letzten Kapitel einen kurzen Ausblick geben auf Verfahren zur Lösung von (gemischt-)ganzzzahligen linearen Optimierungsproblemen. Dabei handelt es sich um Probleme, die ebenfalls durch eine lineare Zielfunktion und lineare Nebenbedingungen beschrieben werden können, die aber zusätzlich noch für alle oder einen Teil der Variablen Ganzzahligkeitsbedingungen enthalten. Werden an alle Variablen Ganzzahligkeitsforderungen gestellt, so nennt man das Problem ein *ganzzzahliges lineares Problem* (kurz: IP oder ILP).¹ Werden dahingegen Ganzzahligkeitsbedingungen nur an einen Teil der Variablen gestellt, so spricht man von einem *gemischt-ganzzzahligen linearen Problem* (kurz: MIP oder MILP).²

Moderne MIP-Löser benutzen im Kern ein Verfahren, das unter dem Namen “Branch-and-Cut” bekannt ist. Wir besprechen hier kurz die Grundzüge des dem Branch-and-Cut zugrunde liegenden “Branch-and-Bound”-Verfahrens. Dieses zuerst von Land und Doig [13] beschriebene Verfahren (auch wenn es in dem zitierten Artikel noch nicht so genannt wurde) bildet heute das Grundgerüst aller modernen MIP-Löser.

Im Kern ist es ein enumeratives Verfahren, d. h., es zählt die Lösungen der Reihe nach auf. Der Trick dabei ist, das Aufzählen so zu organisieren, dass wir möglichst vermeiden können, wirklich alle Lösungen zu enumerieren.

Wir diskutieren das Verfahren der Einfachheit halber für 0/1-IPs an – also

¹IP und ILP stehen für “integer program” bzw. “integer linear program”.

²MIP und MILP stehen für “mixed-integer program” bzw. “mixed-integer linear program”.

für Probleme der Form

$$\begin{aligned} \max_x \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b, \\ & x \in \{0, 1\}^n. \end{aligned} \tag{15.1}$$

Da wir später Probleme lösen wollen, bei denen gewisse ganzzahlige Variablen fixiert sind, führen wir gleich die dafür notwendige Notation ein:

$$\begin{aligned} \max_x \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b, \\ & x \in \{0, 1\}^n, \\ & x_i = 0 \quad \text{für alle } i \in Z, \\ & x_i = 1 \quad \text{für alle } i \in O. \end{aligned} \tag{15.2}$$

Das Problem (15.1) ist dann einfach der Spezialfall, bei dem $Z = O = \emptyset$ gilt. Um eine sinnvolle Reihenfolge zu erhalten, in der wir die Lösungen aufzählen, betrachten wir die sogenannte *LP-Relaxierung* des Problems. Das bedeutet, wir ignorieren die Ganzzahligkeitsbedingung und erhalten

$$\begin{aligned} \max_x \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b, \\ & x \in [0, 1]^n, \\ & x_i = 0 \quad \text{für alle } i \in Z, \\ & x_i = 1 \quad \text{für alle } i \in O. \end{aligned} \tag{15.3}$$

Wir schauen uns nun die zwei Hauptideen des Algorithmus an, die wir jeweils als Lemma formulieren.

Lemma 15.1. Es seien $Z, O \subseteq \{1, \dots, n\}$. Ferner sei z_{LP} der Zielfunktionswert einer Optimallösung der LP-Relaxierung (15.3) und z_{IP} der Zielfunktionswert von (15.2), wenn sie jeweils existieren und $-\infty$ wenn nicht. Dann gilt

$$z_{\text{IP}} \leq z_{\text{LP}}.$$

Außerdem gilt, dass aus der Unzulässigkeit der LP-Relaxierung (15.3) die Unzulässigkeit von (15.2) folgt.

Dieses Lemma ist zentral: Wenn wir die LP-Relaxierung lösen, können wir damit eine korrekte Abschätzung des Zielfunktionswerts des IPs angeben oder sogar die Unzulässigkeit des IPs zertifizieren.

Jetzt noch die zweite Beobachtung.

Lemma 15.2. Es seien $Z, O \subseteq \{1, \dots, n\}$. Ferner sei $x \in \{0, 1\}^n$ zulässig für (15.2) mit den Mengen Z, O und $i \in \{1, \dots, n\}$. Dann ist x zulässig entweder für (15.2) mit den Mengen $(Z \cup \{i\}, O)$ oder zulässig für (15.2) mit den Mengen $(Z, O \cup \{i\})$.

Diese beide Behauptungen zusammen ergeben auch den Namen des Verfahrens. Das Lemma 15.1 erlaubt uns den Zielfunktionswert zu beschränken (“bound”) und Lemma 15.2 erlaubt uns das Problem in Teilprobleme zu zerlegen (“branch”). Das Ganze führt zu Algorithmus 22.

Algorithmus 22 Branch-and-Bound für 0/1-IPs.

```

1:  $\ell \leftarrow -\infty$  und  $Q \leftarrow \{(\emptyset, \emptyset)\}$ .
2: while  $Q \neq \emptyset$  do
3:   Wähle ein  $(Z, O) \in Q$  und setze  $Q \leftarrow Q \setminus \{(Z, O)\}$ .
4:   Löse die LP-Relaxierung mit  $Z$  und  $O$ .
5:   if LP-Relaxierung mit  $Z$  und  $O$  ist unzulässig then
6:     Gehe zu Schritt 2.
7:   Setze  $\bar{x}$  auf die Optimallösung der LP-Relaxierung.
8:   if  $c^\top \bar{x} \leq \ell$  then
9:     Gehe zu Schritt 2.
10:  if  $\bar{x}$  ist ganzzahlig then
11:    Setze  $x^* \leftarrow \bar{x}$ ,  $\ell \leftarrow c^\top x^*$  und gehe zu Schritt 2.
12:  Wähle  $i$  mit  $\bar{x}_i \notin \{0, 1\}$ .
13:  Setze  $Q \leftarrow Q \cup \{(Z \cup \{i\}, O), (Z, O \cup \{i\})\}$ .
14: if  $\ell > -\infty$  then
15:   return Optimallösung  $x^*$ .
16: else
17:   return “Das Problem ist unzulässig.”

```

Grob gesprochen kann man das Verfahren wie folgt verstehen: Wir betrachten eine Relaxierung (am Anfang die LP-Relaxierung) und verstärken sie durch Branching. Die neue Schranke ist einfach das Maximum über die Schranken aller Teilprobleme. Es gibt aber noch andere Methoden, die Schranke zu verbessern; wir können Ungleichungen hinzufügen, die die Relaxierung verstärken, sogenannte Schnitte (“cuts”). Wenn wir dies systematisch tun, erhalten wir das tatsächliche Verfahren der MIP-Löser (“Branch-and-Cut”). Die Details dieses Verfahrens lernen Sie in den Vorlesungen Diskrete Optimierung 1 & 2 kennen.

Wir können dieses Verfahren auch auf MIPs verallgemeinern, bei denen wir “ $x_i \in \{0, 1\}$ ” durch “ $x_i \in X_i \subseteq \mathbb{Z}$ ” ersetzen und auch kontinuierliche Variablen zulassen. Der Übergang von 0/1-Problemen zu Problemen mit allgemeinen diskreten Variablen sorgt beim Branching-Schritt dazu, dass wir nicht mehr

Variablen fixieren, sondern Ungleichungen $x_i \leq n$ oder $x_i \geq n + 1$ den neu erzeugten Teilproblemen hinzufügen. Die Analyse des Verfahrens ist dann etwas komplizierter, da auch noch zu berücksichtigen ist, dass die zulässige Menge unbeschränkt sein kann. Diese Variante und viele der kleinen und großen Tricks, die das Verfahren praxistauglich machen, sind Thema der Veranstaltungen Diskrete Optimierung I & II. So spielt es aus praktischer Sicht eine große Rolle, welche Variable x_i wir zum Branching wählen und welches Teilproblem (Z, O) wir auswählen. Dies ändert aber nichts daran, dass wir (im 0/1-Fall) im schlimmsten Fall alle 2^n Varianten durchtesten müssen.

Abbildungsverzeichnis

1.1	Die Funktion $f(x) = x^3 - 7.5x^2 + 18x - 10.5$ im Intervall $[1, 5]$	9
1.2	Die Funktion $-f(x) = -(x^3 - 7.5x^2 + 18x - 10.5)$ im Intervall $[1, 5]$	10
2.1	Die Königsberger Brücken (anno 1736) (siehe Diestel [6])	15
2.2	Abstrakte Darstellung des Königsberger Brückenproblems	15
2.3	Das Haus vom Nikolaus	16
2.4	Ein Grundversorgungsproblem	16
2.5	Die Graphenbegriffe inzident, adjazent, parallele Kanten und Schlinge	17
2.6	Gerichteter Graph	17
2.7	Kanten- und Knotenmenge	18
2.8	Ein- und ausgehende Bögen δ^{out} und δ^{in}	19
2.9	Ein Graph mit isoliertem Knoten e	19
2.10	Der vollständige Graph K_5	20
2.11	Ein bipartiter Graph	20
2.12	Kette, Pfad und Weg	21
2.13	Dualer Graph (in grün)	21
2.14	Eulerpfad in G	22
2.15	Der Graph G aus Beispiel 8	23
2.16	Ein Beispielgraph zur Diskussion unterschiedlicher Speicherungen von Graphen	26
2.17	Grafische Darstellung der Breitensuche	29
2.18	Grafische Darstellung der Tiefensuche	31
3.1	Beispiele für einen Wald, einen Baum und für einen Graphen, der weder Wald noch Baum ist	37
4.1	Ausgangsgraph im Beispiel 14 zum Kruskal-Algorithmus	47
4.2	Bestimmung eines maximalen Waldes minimalen Gewichts nach Kruskal	47
4.3	Ausgangsgraph im Beispiel 15	49

4.4	Bestimmung eines minimal aufspannenden Baums mit Algorithmus 10	50
4.5	Ausgangsgraph im Beispiel 16	52
4.6	Bestimmung eines minimal aufspannenden Baums mit Algorithmus 11	53
5.1	Kürzeste-Wege-Baum	55
5.2	Beispiel für einen negativen Kreis	59
5.3	Graph aus dem Beispiel 18 zum Floyd–Warshall-Algorithmus 15	65
5.4	Ausgangsgraph in Beispiel 19	68
6.1	Ein Fluss mit Wert 3 in einem Graphen mit minimalem Schnitt 4	72
6.2	Beispiel für einen Residualgraph	75
6.3	“Lollipop”-Graph aus Beispiel 21	78
6.4	Graphische Darstellung des Push-Relabel-Algorithmus 17	82
7.1	Reformulierung von Minimalkosten-Fluss-Problemen	87
7.2	Reformulierung von Minimalkosten-Fluss-Problemen mit negativen Kosten	88
8.1	Kreismatroid $M(G)$	97
8.2	Kreismatroid $M(G)$	99
8.3	Graph $G = (V, E)$ mmit 7 Kanten	101
8.4	Graph $G = (V, E)$ mit 13 Kanten	101
8.5	Graph $G = (V, E)$ mit 4 Kanten	102
9.1	Graphische Darstellung zu Beispiel 32	120
10.1	Illustration der zulässigen Menge des linearen Optimierungsproblems (10.1)	129
11.1	Geometrische Interpretation des Farkas-Lemmas	143
12.1	Der Klee-Minty-Würfel aus Beispiel 37	166
13.1	Grafische Darstellung zu Beispiel 38	174

Tabellenverzeichnis

2.1	Landau-Symbole	25
9.1	Getränke auf der Studentenparty	109
9.2	Getränke auf der Studentenparty und ihr Alkoholgehalt . . .	113
9.3	Verlauf des Simplex-Algorithmus bei dem LP aus Beispiel 32	122
11.1	Transformationsregeln	150

Algorithmenverzeichnis

1	Breitensuche	29
2	Tiefensuche	30
3	<code>checkNeighbor(w, next)</code>	31
4	Selectionsort	34
5	Bubblesort	35
6	Quicksort	35
7	Heapsort	38
8	Bucketsort	44
9	Algorithmus von Kruskal, 1956	46
10	Bestimmung minimaler Spannbäume	49
11	Algorithmus von Prim, 1957	52
12	Dijkstra, 1959	57
13	Grundversion des Moore–Bellmann-Algorithmus	59
14	Bellmann, 1958	62
15	Floyd–Warshall, 1962	64
16	Augmentierender-Wege-Algorithmus (von Ford & Fulkerson)	74
17	Push-Relabel-Algorithmus (Goldberg & Tarjan, 1985)	81
18	Kreis-Löschungs-Algorithmus	90
19	Greedy-Max für Unabhängigkeitssysteme	103
20	Das Simplex-Verfahren für LPs in Standardform	120
21	Der duale Simplex-Algorithmus	173
22	Branch-and-Bound für 0/1-IPs.	184

Literatur

- [1] Ravindra K. Ahuja, Thomas L. Magnanti und James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.
- [2] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [3] R. G. Bland. „New finite pivoting rules for the simplex method“. In: *Mathematics of Operations Research* (1977), S. 103–107.
- [4] Stephen Boyd und Lieven Vandenberghe. *Convex Optimization*. Cambridge: Cambridge University Press, 2004, S. xiv+716.
- [5] Vasek Chvátal. *Linear Programming*. A Series of books in the mathematical sciences. New York (N. Y.): Freeman, 1983.
- [6] Reinhard Diestel. *Graph Theory*. 4. Aufl. Bd. 173. Graduate Texts in Mathematics. Springer-Verlag Berlin Heidelberg.
- [7] Jack Edmonds. „Submodular Functions, Matroids, and Certain Polyhedra“. In: New York: Gordon und Breach, 1970, S. 69–87.
- [8] Philip Faigle. *Der große Coup*. URL: <http://www.zeit.de/online/2007/46/schienenennetz>.
- [9] T. E. Harris und F. S. Ross. *Fundamentals of a Method for Evaluating Rail Net Capacities*. report Research Memorandum RM-1573. Santa Monica, California: The RAND Corporation, 1955. URL: <http://ntl.bts.gov/lib/13000/13200/13238/AD093458.pdf>.
- [10] Volker Kaibel, Jon Lee und Jeff Linderoth. *You have to figure out who your customer is going to be – An interview with Bob Bixby*. Sep. 2016. URL: <http://www.mathopt.org/Optima-Issues/optima101.pdf>.
- [11] Donald Ervin Knuth. *The art of computer programming: sorting and searching*. Bd. 3. Pearson Education, 1998.
- [12] Bernhard Korte und Jens Vygen. *Combinatorial Optimization*. Third. Bd. 21. Algorithms and Combinatorics. Berlin: Springer-Verlag, 2006, S. xvi+597.
- [13] A. H. Land und A. G. Doig. „An Automatic Method of Solving Discrete Programming Problems“. In: *Econometrica* 28.3 (1960), S. 497–520. URL: <http://www.jstor.org/stable/1910129>.
- [14] Jorge Nocedal und Stephen J. Wright. *Numerical Optimization*. 2nd. Berlin: Springer, 2006. DOI: 10.1007/978-0-387-40065-5.

- [15] J.G. Oxley. *Matroid Theory*. Oxford graduate texts in mathematics. Oxford University Press, 1992. URL: <https://books.google.de/books?id=Xo1t9YCJCI0C>.
- [16] Christos H. Papadimitriou und Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.
- [17] Alexander Schrijver. „On the History of the Shortest Path Problem“. In: *Optimization Stories*. Hrsg. von Martin Grötschel. Documenta Mathematica. 2012, S. 155–167. URL: http://www.math.uiuc.edu/documenta/vol-ismp/32_schrijver-alexander-sp.pdf.
- [18] Alexander Schrijver. „On the History of the Transportation and Maximum Flow Problems“. In: *Optimization Stories*. Hrsg. von Martin Grötschel. Documenta Mathematica. 2012, S. 169–180. URL: http://www.math.uiuc.edu/documenta/vol-ismp/33_schrijver-alexander-tmf.pdf.
- [19] Alexander Schrijver. *Combinatorial Optimization. Polyhedra and Efficiency*. Bd. A: *Paths, flows, matchings*. Berlin: Springer-Verlag, 2003, S. xxxviii+647.
- [20] Robert Sedgewick. *Algorithms*. Pearson Education India, 1988.
- [21] H. Whitney. „On the abstract properties of linear dependence“. In: *American Journal of Mathematics* 57 (1935), S. 509–533.