Diskrete Optimierung

TU Bergakademie Freiberg
Fakultät für Mathematik und Informatik
Institut für Numerische Mathematik und Optimierung
Dr.rer.nat. H. Schreier
Sommersemester 2013

Die wesentlichen Eigenschaften linearer Optimierungsaufgaben waren neben Linearität von Zielfunktion und Nebenbedingungen, dass ausschließlich kontinuierliche Variable vorkamen. Aber schon bei Transport- und Zuordnungsproblemen spielten ganzzahlige Werte der Variablen eine Rolle, da andere Werte keine sinnvollen Lösungen ergaben. Die Ganzzahligkeit ergab sich dabei zwangsläufig aus der mathematischen Struktur dieser Modelle und musste deshalb nicht als zusätzliche Restriktion gefordert werden. Somit konnten diese Modelle als Spezialfälle der linearen Optimierung abgehandelt werden. Eine zusätzlich auftretende Forderung nach Ganzzahligkeit von Lösungsvektoren zu einer beliebigen linearen Optimierungsaufgabe kann jedoch zu erheblichen theoretischen und rechnerischen Problemen führen. Deshalb sind auch Kenntnisse zur Komplexitätstheorie nützlich.

Viele *diskrete Optimierungsprobleme* haben eine Interpretation als ganzzahlige lineare oder gemischt-ganzzahlige lineare Optimierungsaufgabe. Diskrete Optimierungsprobleme treten immer dann auf, wenn die betrachteten Größen nur endlich oder abzählbar unendlich viele Werte annehmen. Im ersten Fall spricht man auch von *kombinatorischen Optimierungsproblemen*. Die Kombinatorik befasst sich bekanntlich mit den verschiedenen Möglichkeiten der Anordnung endlich vieler Objekte. Die *Graphentheorie* stellt in diesem Bereich viele Begriffe zur Beschreibung zugehöriger Optimierungsprobleme bereit.

Der Problemkreis der diskreten Optimierung erscheint bei unscharfer Betrachtung als Vielzahl einzelner Probleme, über die mittels tiefgreifender Überlegung Aussagen gewonnen werden können, die Beziehungen zu vielen anderen mathematischen Disziplinen aufweisen. Man kann aber feststellen, dass für diese Probleme allgemeine formalisierte Lösungsmethoden vorliegen. Die Untersuchungen der letzten Jahrzehnte zeigen, dass sich die diskrete Optimierung zu einem eigenständigen und perspektivreichen Teilgebiet der mathematischen Optimierung mit sehr spezifischen Denk- und Arbeitsweisen entwickelt hat.

Die Behandlung dieses extrem umfangreichen und vielschichtigen Gebietes kann in diesem Rahmen nur exemplarisch erfolgen. In der Lehrveranstaltung erfolgt die Darlegung grundlegender Begriffe, eine Vorstellung von Modellierungstechniken an Hand von Beispielen, die zu speziellen ganzzahligen Optimierungsaufgaben führen, die Beschreibung von Prinzipien zur Konstruktion exakter Lösungsverfahren und die Charakterisierung von Näherungsverfahren. Für weiterführende Betrachtungen sei auf die umfangreichen Monografien von [NeWo88] [KoVy00] und [Schr03] zur diskreten und kombinatorischen Optimierung verwiesen, die sich tiefgründig mit einer Vielzahl von Problemen auseinandersetzen.

Inhaltsverzeichnis

Kapitel 1

Begriffe zur diskreten Optimierung

Zur Formulierung der Aufgabenstellung wird zunächst der Begriff der diskreten Menge eingeführt. Es sei vorausgesetzt, dass sich eine diskrete Menge S in einen metrischen Raum H, zum Beispiel in den n-dimensionalen Vektorraum $\mathbb{R}^n$ mit der über die Euklidische oder eine andere Norm erklärten Metrik einbetten lässt.

Definition 1.1
Eine Menge S heißt *diskret*, wenn eine der beiden folgenden Forderungen erfüllt ist:

a) S ist endlich.

b) $S \subset H$, wobei H ein metrischer Raum ist, und zu S existiert ein $\varepsilon > 0$, so dass für jedes $x \in S$ die Eigenschaften $K(x, \varepsilon) \subset H$ und $K(x, \varepsilon) \cap S = \{x\}$ gelten, wobei $K(x, \varepsilon)$ eine Kugel um $x \in H$ mit Radius $\varepsilon > 0$ ist.

Bemerkung 1.1
Eine Menge S mit $S = S_1 \times S_2$, $S_1 \neq \emptyset$, $S_2 \neq \emptyset$ und S_1 diskret, heißt *gemischt diskret*.
Eine Menge $S \subset \mathbb{R}^n$ ist demzufolge gemischt diskret, wenn mindestens eine aber nicht alle Komponenten von $x \in S$ diskret sind.

Es sei $S \neq \emptyset$ eine diskrete Menge und $f : S \longmapsto \mathbb{R}^1$ eine reellwertige Funktion.

Modell DOP (Diskretes Optimierungsproblem):

$$\begin{aligned} f(x) &\longrightarrow \min \\ x &\in S \end{aligned} \tag{1.1}$$

Gesucht ist das Minimum von f über S oder die Aussage "Aufgabe unlösbar".
Gilt $f^* = \min_{x \in S} f(x)$, dann ist die Optimalmenge $S^* = \{x \mid x \in S \wedge f(x) = f^*\}$ oder wenigstens ein Element $x^* \in S^*$ zu bestimmen.

Bemerkung 1.2
Ein *kombinatorisches Optimierungsproblem* liegt genau dann vor, wenn die Menge S in (1.1) *nichtleer* und *endlich* ist. Für eine kombinatorische Optimierungsaufgabe gilt stets $S^* \neq \emptyset$.

Kombinatorische Optimierungsprobleme sind oft in einer konkreten Struktur gegeben. Die Menge S wird nicht dadurch beschrieben, dass man alle Elemente explizit aufzählen kann. Sie ist meist implizit durch die Angabe von gewissen Eigenschaften charkterisiert. Auch die Funktion f wird konkret durch eine Vorschrift gegeben sein.

Viele Probleme der kombinatorischen Optimierung werden durch folgende Modellierung erfasst: Eine Grundmenge werde durch die Indexmenge $N = \{1, ..., n\}$ beschrieben. Mit $c = (c_1, ..., c_n)^\top$ seien die Elemente von N bewertet. Einer Teilmenge $F \subseteq N$ wird die Bewertung $c(F) = \sum_{j \in F} c_j$ zugeordnet. Weiter sei $\mathcal{F}$ eine Familie von Teilmengen von N. Die Menge aller Teilmengen von N ist die mit $Pot(N)$ bezeichnete Potenzmenge. Dann gilt $\mathcal{F} \subseteq Pot(N)$. Die Familie $\mathcal{F}$ beschreibt den zulässigen Bereich und entspricht der diskreten Menge S. Damit wird durch

$$\begin{aligned} c(F) &\longrightarrow \min \\ F &\in \mathcal{F} \end{aligned} \tag{1.2}$$

eine kombinatorische Optimierungsaufgabe mit *linearer* Zielfunktion dargestellt.

Es sei S eine diskrete Menge und $A \subseteq S$. Durch die Frage

$$\text{Gehört ein } y \in S \text{ zu } A \text{ ?}$$

wird ein *diskretes Entscheidungsproblem* formuliert.
Dem diskreten Entscheidungsproblem kann mittels der charakteristischen Funktion

$$f(x) = \begin{cases} 0 & , \quad x \in A \\ 1 & , \quad x \in S \setminus A \end{cases}$$

eine diskrete Optimierungsaufgabe zugeordnet werden.

Für die Beschreibung von Prinzipien zur Konstruktion von Lösungsverfahren sind noch einige weitere Begriffe einzuführen.

Definition 1.2
Gegeben seien die Optimierungsaufgaben

$$P: \qquad f(x) \to \min\,, \quad x \in S$$
$$Q: \qquad g(x) \to \min\,, \quad x \in R \cap S$$

Q heißt *Einbettung* von P, wenn folgende Eigenschaften gelten:

a) $f^* = \min\limits_{x \in S} f(x) \quad \Longrightarrow \quad g^* = \min\limits_{x \in R \cap S} g(x)$

b) $\tilde{x} \in R \cap S \ \wedge \ g(\tilde{x}) = g^* \quad \Longrightarrow \quad f(\tilde{x}) = f^*$

Aus der Lösbarkeit der Aufgabe P soll mittels der ersten Forderung auch die Lösbarkeit der Aufgabe Q folgen. Mit dem Optimalwert g^* kann man dann formal die Optimalmenge $R^* = \{x \mid x \in R \cap S \wedge g(x) = g^*\}$ bestimmen. Die zweite Forderung garantiert die Inklusion $R^* \subseteq S^*$. Im Falle der Lösbarkeit von P wird bei einer Einbettung wenigstens eine Optimallösung von P mit Hilfe der Optimierungsaufgabe Q bestimmt.

Definition 1.3
Gegeben seien die Optimierungsaufgaben

$$P: \qquad f(x) \to \min\,, \quad x \in S$$
$$Q: \qquad g(x) \to \min\,, \quad x \in R$$

Q heißt *Relaxation* von P, wenn folgende Eigenschaften gelten:

a) $S \subseteq R$

b) $g(x) \leq f(x)\,, \quad \forall x \in S$

Die Funktion g ist eine *Minorante* zur Funktion f für Elemente $x \in S$.
Im Allgemeinen erhält man eine Relaxation zu P, wenn aus der impliziten Beschreibung der Menge S nicht alle Informationen auf die Menge R übertragen werden. Dies kann zum Beispiel dadurch geschehen, dass man explizit angegebene Diskretheitsforderungen wie die Ganzzahligkeit von Variablenwerten fallen lässt.

Satz 1.1
Gegeben sind die Optimierungsaufgabe P und eine zugehörige Relaxation Q. Beide Aufgaben seien lösbar, das heißt $f^* = \min\limits_{x \in S} f(x)$ und $g^* = \min\limits_{x \in R} g(x)$. Dann gilt:

(i) $g^* \leq f^*$

(ii) $g(\tilde{x}) = g^* \ \wedge \ \tilde{x} \in S \ \wedge \ g(\tilde{x}) = f(\tilde{x}) \quad \Longrightarrow \quad f(\tilde{x}) = f^*$

Die erste Beziehung besagt, dass g^* eine untere Schranke für den gesuchten Optimalwert der Optimierungsaufgabe P ist. Das in der zweiten Beziehung aufgeführte naive Optimalitätskriterium für die Lösung $\tilde{x} \in S$ wird oft in Lösungsprinzipien zur diskreten Optimierung verwendet.

Definition 1.4
Gegeben seien die Aufgaben

$$P: \qquad f(x) \to \min\,, \quad x \in S$$
$$P_k \qquad f(x) \to \min\,, \quad x \in S_k\,, \quad k = 1, ..., l$$

Gilt $S = \bigcup\limits_{k=1}^{l} S_k$, dann ist P_k, $k = 1, ..., l$, eine *Separation* von P.
Gilt zusätzlich $S_\mu \cap S_\nu = \emptyset$, $\mu, \nu = 1, ..., l$, $\mu \neq \nu$, dann liegt eine strenge Separation, auch *Partition* genannt, vor.

Mittels Separation wird die Aufgabe P in eine endliche Anzahl "kleinerer" Aufgaben P_k zerlegt. Bei der Übertragung geht keine Lösung aus S verloren. Wünschenswert wäre eine (implizite) Beschreibung der Mengen S_k , $k = 1, ..., l$, so, dass jede Lösung $x \in S$ in genau einer der Mengen S_k , $k = 1, ..., l$, enthalten ist.

Kapitel 2

Modellierung von Diskretheitsbedingungen

2.1 Optimierungsprobleme mit Ganzzahligkeitsforderungen

Diskrete Optimierungsaufgaben über dem Raum $\mathbb{R}^n$ erhält man formal durch die Beschreibung mittels skalarer Funktionen und diskreter Mengen für einzelne Komponenten des Lösungsvektors. Es seien D_j , $j = 1, ..., n$, diskrete Mengen im $\mathbb{R}^1$. Dann wird durch

$$\begin{array}{rcl} f(x_1, ..., x_n) & \rightarrow & \min \\ g_i(x_1, ..., x_n) & \geq & 0 \quad , \quad i = 1, ..., m \\ x_j & \in & D_j \ , \quad j = 1, ..., n \end{array} \tag{2.1}$$

eine allgemeine diskrete Optimierungsaufgabe im $\mathbb{R}^n$ beschrieben.
Im Falle $D_j = \mathbb{N} \cup \{0\}$, $j = 1, ..., n$, nehmen alle Variable nur nichtnegative ganzzahlige Werte an und man spricht von einer ganzzahligen Optimierungsaufgabe. Erfolgt eine zusätzliche Einschränkung auf $D_j = \{0\,;\,1\}$, $j = 1, ..., n$, dann liegt eine *binäre* Optimierungsaufgabe vor.

Zur Formulierung der folgenden beiden ganzzahligen Optimierungssprobleme wird mit $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ die Menge der nichtnegative ganzen Zahlen bezeichnet.

Modell GGLOA (gemischt ganzzahlige lineare Optimierungsaufgabe):

$$\begin{array}{rcl} z = \sum\limits_{j=1}^{n} c_j x_j + \sum\limits_{k=1}^{l} d_k y_k & \rightarrow & \max \\ \sum\limits_{j=1}^{n} a_{ij} x_j + \sum\limits_{k=1}^{l} b_{ik} y_k & = & b_i \,, \quad i = 1, ..., m \\ x_j & \in & \mathbb{N}_0 \,, \quad j = 1, ..., n \\ y_k & \geq & 0 \,, \quad k = 1, ..., l \end{array} \tag{2.2}$$

Die gemischt ganzzahlige lineare Optimierungsaufgabe kann auch als Minimierungsaufgabe sowie auch mit Ungleichungsrestriktionen dargestellt werden. Um zu den in (2.2) formulierten Gleichungen zu kommen, muss man nur Schlupfvariable einführen und diese zu den kontinuierlichen Variablen zählen.

Modell GLOA (rein ganzzahlige lineare Optimierungsaufgabe):

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &\rightarrow \max \\ \sum_{j=1}^{n} a_{ij} x_j &= b_i\,, \quad i = 1, ..., m \\ x_j &\in \mathbb{N}_0\,, \quad j = 1, ..., n \end{aligned} \tag{2.3}$$

Eine durch Ungleichungen formulierte rein ganzzahlige lineare Optimierungsaufgabe der Form

$$\begin{aligned} z = c^\top x &\rightarrow \max \\ Ax &\leq b \\ x &\geq 0\,, \quad \text{ganzzahlig} \end{aligned}$$

ist nach Einführung des Vektors der Schlupfvariablen u äquivalent zur folgenden gemischt ganzzahligen linearen Optimierungsaufgabe:

$$\begin{aligned} z = c^\top x + 0^\top u &\rightarrow \max \\ Ax + Eu &= b \\ x &\geq 0\,, \quad \text{ganzzahlig} \\ u &\geq 0 \end{aligned}$$

Man kann nicht automatisch auch an den Vektor u die Ganzzahligkeit fordern.

Ganzzahlige lineare Optimierungsmodelle besitzen sehr viele praktische Anwendungen. An dieser Stelle soll stellvertretend ein Modell zur *Zuschnittoptimierung* angegeben werden. Man findet derartige Probleme bei industriellen Fertigungsvorgängen, wo Material für die weitere Verarbeitung oder den Verkauf zugeschnitten wird. Dabei wird ein Zuschnittplan gesucht, bei dem entweder der Abfall oder der Verbrauch des eingesetzten Ausgangsmaterials minimiert wird. Hier soll ein eindimensionales Zuschnittproblem betrachtet werden, das heißt man hat nur eine Abmessung wie etwa beim Zuschnitt von Eisenträgern im Bauwesen zu berücksichtigen.

Es sei L die Länge der zu zerschneidenden Ausgangsstücke. Die Längen der benötigten Teilstücke sei mit $l_i\,,\ i = 1, ..., m$, bezeichnet. Ohne Beschränkung der Allgemeinheit gelte $l_1 > l_2 > ... > l_m$. Es ist bekannt, dass mindestens $b_i\,,\ i = 1, ..., m$, Teilstücke der Nummer i für gewisse Aufträge benötigt werden. Jetzt braucht man noch zulässige beziehungsweise sinnvolle Zuschnittkombinationen.
Für eine Zuschnittkombination mit Nummer $j \in \{1, ..., n\}$ seien mit $a_{ij}\,, i = 1, ..., m$, die Anzahl der Längen l_i gegeben, die man aus der Ausgangslänge L abschneidet. Eine Zuschnittkombination j ist zulässig und sinnvoll, wenn

$$0 \leq L - \sum_{i=1}^{m} a_{ij}\, l_i < l_m\,,\ \ j = 1, ..., n$$

gilt. Verzichtet man auf den zweiten Teil der Ungleichungen, dann entstehen zwar zulässige Zuschnittkombinationen, aber vom verbleibenden Rest könnten noch benötigte Teile abgeschnitten werden.
Die zu bestimmenden Werte der Variablen x_j , $j = 1, ..., n$, geben an, wie oft eine Länge L nach der Zuschnittkombination j geschnitten werden soll. Will man die Anzahl der für die Aufträge benötigten Ausgangsstücke minimieren, dann erhält man das folgende rein ganzzahlige Optimierungsmodell in Ungleichungsform:

$$\begin{aligned} z_{MV} &= \sum_{j=1}^{n} x_j \;\rightarrow\; \min \\ \sum_{j=1}^{n} a_{ij} x_j &\geq b_i\,, \quad i = 1, ..., m \\ x_j &\in \mathbb{N}_0\,, \quad j = 1, ..., n \end{aligned}$$

Für die Minimierung des Verschnittes wird die Zielfunktion

$$z_{AF} = \sum_{j=1}^{n} \left(L - \sum_{i=1}^{m} a_{ij}\, l_i \right) x_j \;\longrightarrow\; \min$$

verwendet. Bei diesem Zielkriterium kann es aber zur Überproduktion einzelner Teilstücke kommen. Dies ist unproblematisch, wenn diese Längen noch zu einem späteren Zeitpunkt verwendet werden können. Es wäre aber auch denkbar, eine Überproduktion durch zusätzliche Restriktionen einzuschränken.

2.2 Modellierung mit Hilfe von 0-1-Variablen

In vielen praktischen Situationen sind Entscheidungen zu treffen, ob ein Projekt realisiert werden soll, oder nicht. Können die Variablen in (2.3) nur die Werte Null oder Eins annehmen, dann spricht man von einer binären linearen Optimierungsaufgabe oder einer linearen 0-1-Optimierungsaufgabe. Sie lässt sich in allgemeiner Form wie folgt darstellen:

Modell L-01-OA (lineare 0-1-Optimierungsaufgabe):

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j \quad &\longrightarrow \quad \left\{ \begin{array}{c} \max \\ \min \end{array} \right\} \\ \sum_{j=1}^{n} a_{ij} x_j \quad &\left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} \quad b_i\,, \qquad i = 1, ..., m \\ x_j \quad &\in \quad \{0; 1\}\,, \quad j = 1, ..., n \end{aligned} \tag{2.4}$$

Bei binären Variablen ist die Linearität von Funktionen, die bei der Beschreibung dieses Modells vorliegt, nicht mehr so wesentlich. Prinzipiell kann (2.4) auch durch eine kombinatorische Optimierungsaufgabe der Gestalt (1.2) beschrieben werden. Damit liegt ein qualitativ anderer Problemtyp vor.

Ein besonders anschaulisches Beispiel für eine lineare 0-1-Optimierungsaufgabe ist das *Rucksackproblem*, auch unter dem Namen *Knapsack Problem* bekannt. Gegeben sei eine Menge von auswählbaren Gegenständen $j \in \{1, ..., n\}$, die ein Wanderer in seinen Rucksack packen möchte. Der Gegenstand j besitze den Nutzen c_j und das Gewicht g_j. Mit G wird das Höchstgewicht der mitzunehmenden Gegenstände vorgegeben. Bezeichnet man mit

$$x_j = \begin{cases} 1\,, & \text{Gegenstand j wird eingepackt} \\ 0\,, & \text{sonst} \end{cases} \quad , \quad j = 1, ..., n$$

die binären Variablen, so lässt sich das Modell wie folgt formulieren:

Modell RSP (Rucksackproblem):

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &\rightarrow \max \\ \sum_{j=1}^{n} g_j\, x_j &\leq G \\ x_j &\in \{0; 1\}\,, \quad j = 1, ..., n \end{aligned} \tag{2.5}$$

Üblicherweise setzt man für die Daten eines Rucksackproblems $c_j > 0$, $g_j > 0$, $g_j \leq G$, $j = 1, ..., n$, und $\sum_{j=1}^{n} g_j > G$ voraus.

Anwendungen zu diesem auf dem ersten Blick recht einfach erscheinenden Modell findet man unter anderem in der Investitionsplanung. Die Größe G kann als Budget interpretiert werden, das in n Projekte mit den Kosten g_j und zu erwartenden Gewinnen c_j investiert werden soll.

Ein klassisches Beispiel für eine kombinatorische Optimierungsaufgabe, die sich auch mit Hilfe von binären Variablen formulieren lässt, ist das sogenannte *Mengenüberdeckungsproblem* (Set Covering).

Gegeben sei eine Familie von n Teilmengen $M_j\,,\ j \in N = \{1, 2, ..., n\}$, einer Grundmenge $M = \{1, 2, ..., m\}$, das heißt es gilt $M_j \subseteq M\,,\ j \in N$. Jeder Menge M_j sind Kosten c_j zugeordnet. Gesucht ist eine kostenminimale Überdeckung der Menge M durch ein System von ausgewählten Teilmengen $M_j\,,\ j \in J,\ J \subseteq N$.

Damit kann folgendes Modell aufgestellt werden:

Modell MÜP (Mengenüberdeckungsproblem):

$$\begin{aligned} \sum_{j \in J} c_j &\rightarrow \min \\ \bigcup_{j \in J} M_j &= M \\ J &\subseteq N \end{aligned} \tag{2.6}$$

Die Mengen M_j , $j \in N$, können durch Spaltenvektoren einer Matrix A repräsentiert werden:

$$a_{ij} = \begin{cases} 1\,, & \text{falls } i \in M_j \\ 0\,, & \text{falls } i \notin M_j \end{cases}, \quad j \in N$$

Mit den binären Variablen

$$x_j = \begin{cases} 1\,, & j \in J \\ 0\,, & j \notin J \end{cases}$$

lässt sich die folgende spezielle lineare 0-1-Optimierungsaufgabe zuordnen:

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &\rightarrow \min \\ \sum_{j=1}^{n} a_{ij}\, x_j &\geq 1\,, \qquad i = 1, ..., m \\ x_j &\in \{0; 1\}\,, \quad j = 1, ..., n \end{aligned}$$

Hat man eine Optimallösung x^* dieser Aufgabe bestimmt, dann ist ein optimales Mengensystem J^* durch die Zuordnung $x_j^* = 1 \Longleftrightarrow j \in J^*$ bestimmt.
Beim *Mengenaufteilungsproblem* (Set Partitioning) ist zusätzlich die Forderung

$$M_k \cap M_l = \emptyset\,,\ k \in J\,,\ l \in J\,,\ k \neq l$$

dem Modell (2.6) beizufügen. In der zugeordneten linearen 0-1-Optimierungsaufgabe sind die Ungleichungsnebenbedingungen als Gleichungen zu formulieren.

Praktische Anwendungen treten bei der Routenplanung in einem Fuhrpark auf. Dann stellt die Menge M die anzusteuernden Ziele dar und die Mengen M_j , $j \in N$, repräsentieren Kombinationen möglicher Ziele, die ein Fahrzeug bewältigen kann. Das Mengenaufteilungsproblem bestimmt einen kostenminimalen Einsatzplan unter der Annahme, dass jedes Ziel genau einmal angesteuert wird. Beim Mengenüberdeckungsproblem wird das mehrfache Anfahren eines Ziels zugelassen.

Im Folgenden wird demonstriert, wie man mit Hilfe von binären Variablen Diskretheitsforderungen der verschiedensten Art modellieren kann. Dabei werden auch Modelltypen vorgestellt, bei denen die Werte der binären Variablen kein Maß für eine Menge sind, sondern einfach nur logische Entscheidungen darstellen.

Wenn Variable nur *endlich* viele Werte annehmen können, dann haben die entsprechenden diskreten Mengen die Gestalt $D_j = \{d_{jk} \mid k = 1, ..., r_j\}$, $j = 1, ..., n$, wobei $r_j \geq 2$, ganzzahlig, $j = 1, ..., n$, vorausgesetzt wird. Die Bedingungen $x_j \in D_j$, $j = 1, ..., n$, lassen sich mit Hilfe von binären Variablen wie folgt umformulieren:

$$\left.\begin{aligned} x_j - \sum_{k=1}^{r_j} d_{jk} y_{jk} &= 0 \\ \sum_{k=1}^{r_j} y_{jk} &= 1 \\ y_{jk} &\in \{0\,;\,1\}\,, \quad k = 1, ..., r_j \end{aligned}\right\} \quad j = 1, ..., n,$$

Repräsentieren Variable $x_j \in \{0\,;\,1\}\,,\ j = 1, ..., n$, Entscheidungen über das Nichtausführen beziehungsweise Ausführen einer Aktivität j, dann gibt die Restriktion

$$\sum_{j=1}^{n} x_j \left\{ \begin{array}{l} \leq \\ = \\ \geq \end{array} \right\} t$$

an, dass von den n möglichen Aktivitäten *höchstens*, *genau* beziehungsweise *mindestens* t Aktivitäten auszuführen sind. Die Forderung

$$x_k \leq \sum_{j \in T_k} x_j \ , \quad T_k \subseteq \{1, ..., n\} \setminus \{k\}$$

bedeutet, dass Aktivität k nur dann ausgeführt werden kann, wenn mindestens eine der Aktivitäten aus der Menge T_k realisiert wird.

Die im Folgenden zu beschreibenden *alternativen Bedingungen* führen zu nicht zusammenhängenden Lösungsbereichen. Dies drückt sich auch in der möglichen Modellierung mit Hilfe von binären Variablen aus.

Für ein Optimierungsproblem mit einem nichtleeren und beschränkten Grundbereich G sollen zusätzlich *alternativ* die Restriktionen

$$\sum_{j=1}^{n} a_{ij} x_j \leq a_i \,, \quad i = 1, ..., m$$

oder die Restriktionen

$$\sum_{j=1}^{n} b_{kj} x_j \leq b_k \,, \quad k = 1, ..., l$$

erfüllt werden.
Wegen der Beschränktheit des Grundbereichs G existiert eine hinreichend große Zahl M, so dass für alle $x \in G$ stets $\sum_{j=1}^{n} a_{ij} x_j \leq M$, $i = 1, ..., m$, und auch $\sum_{j=1}^{n} b_{kj} x_j \leq M$, $k = 1, ..., l$, gilt.

Der alternative Bereich kann mit Hilfe einer binären Variable modelliert werden:

$$\begin{array}{lll} \displaystyle\sum_{j=1}^{n} a_{ij} x_j - (M - a_i)\, y & \leq & a_i \ , \quad i = 1, ..., m \\ \displaystyle\sum_{j=1}^{n} b_{kj} x_j - (M - b_k)\,(1 - y) & \leq & b_k \ , \quad k = 1, ..., l \\ y \in \{0\,;\,1\} & & \end{array}$$

Die binäre Variable y geht mit dem Wert Null in die Zielfunktion des Optimierungsproblems ein. Für $y = 0$ wird die Erfüllung der ersten m Restriktionen gefordert. Die anderen müssen nicht unbedingt erfüllt werden. Analog sind im Fall $y = 1$ die letzen l Restriktionen zu erfüllen, aber nicht notwendig die anderen.

Um eine echte "entweder-oder" Entscheidung geht es bei dem folgenden Problem. Verschiedene Lastkraftwagen haben eine maximale Ladekapazität der Größe s_j , $j = 1, ..., n$. Wenn ein Lastkraftwagen für einen Einsatz vorgesehen ist, dann soll er eine Mindestmenge u_j , $0 < u_j \leq s_j$, $j = 1, ..., n$, aufladen. Bezeichnet man mit x_j , $j = 1, ..., n$, die zu transportierende Mengen für den j-ten Lastkraftwagen, dann stellen sich die Forderungen

$$\left.\begin{array}{ll} \text{entweder} & x_j = 0 \\ \text{oder} & u_j \leq x_j \leq s_j \end{array}\right\} \quad j = 1, ..., n\,.$$

Diese Alternativen lassen sich auch mit Hilfe von binären Variablen modellieren:

$$\left.\begin{array}{rcl} -x_j + u_j\, y_j & \leq & 0 \\ x_j - s_j\, y_j & \leq & 0 \\ y_j & \in & \{0; 1\} \end{array}\right\} \quad j = 1, ..., n$$

Offensichtlich folgt aus $y_j = 0$ sofort $x_j = 0$ und der entsprechend Lastkraftwagen kommt nicht zum Einsatz. Aus $y_j = 1$ resultiert die Forderung $u_j \leq x_j \leq s_j$. Der entsprechende Lastkraftwagen kommt zum Einsatz und transportiert nicht nur geringfügige Mengen.

In vielen praktischen Problemen tritt der Fall ein, dass eine Vielzahl von zu berücksichtigenden Restriktionen zur Nichtexistenz zulässiger Lösungen führt. Dann ist der Verzicht auf die Erfüllung einer festen Anzahl von Restriktionen angebracht. Bei einem Optimierungsproblem seien die Restriktionen

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad , \quad i = 1, ..., m$$

gegeben. Es sollen *mindestens* t dieser Restriktionen, erfüllt werden, wobei $0 < t < m$ gilt. Es handelt sich jetzt um ein Optimierungsproblem mit mehrfachen Alternativen. Mit M sei wieder eine hinreichend große Zahl bekannt, die als obere Schranke für alle Restriktionen für die auszuwählenden Lösungen aus einer Grundmenge G fungiert. Jeder Restriktion wird eine binäre Variable y_i , $i = 1, ..., m$, zugeordnet und das folgende System aufgestellt:

$$\begin{array}{rcl} \displaystyle\sum_{j=1}^{n} a_{ij} x_j & \leq & b_i + (M - b_i)\, y_i \quad , \quad i = 1, ..., m \\ \displaystyle\sum_{i=1}^{m} y_i & \leq & m - t \\ y_i & \in & \{0\,;\, 1\} \quad , \quad i = 1, ..., m \end{array}$$

Für ein $y_i = 0$ wird die Erfüllung der Restriktion i gefordert. Sollen mindestens t Restriktionen erfüllt werden, dann müssen mindesten t binäre Variable den Wert Null haben. Das entspricht aber der Bedingung, dass höchstens $m - t$ binäre Variable auf Eins zu setzen sind.
Zu beachten ist, dass bei dieser Modellierung durch $\sum_{i=1}^{m} y_i = m - t$ nicht die Erfüllung von genau t Restriktionen realisiert wird. Für ein $y_i = 1$ kann die zugehörige Restriktion i durchaus erfüllt sein.

Die folgende Problemformulierung beschreibt einen ähnlichen Sachverhalt. Von den Variablen $x_j\,,\ j=1,...,n$, mit vorgegebenenen Beschränkungen $0 \le x_j \le s_j\,,\ j=1,...,n$, wobei $s_j > 0\,,\ j=1,...,n$, vorausgesetzt wird, dürfen in einem Optimierungsproblem in der Endlösung nicht mehr als k Variable positive Werte annehmen. Hier ist natürlich $0 < k < n$ vorauszusetzen. In der Transportoptimierung kann man auf diese Weise die Zahl der zu benutzenden Transportverbindungen einschränken. Eine zu scharfen Vorgabe der Zahl k kann allerdings schnell zur Unlösbarkeit einer konkreten Aufgabe führen.
Mit Hilfe von binären Variablen erhält man die folgende Darstellung:

$$\begin{aligned} x_j &\ge 0\,, && j=1,...,n \\ x_j - s_j y_j &\le 0\,, && j=1,...,n \\ \sum_{j=1}^{n} y_j &\le k \\ y_j &\in \{0\,;\,1\}\,, && j=1,...,n \end{aligned}$$

Auch hier ist über diesen Zugang durch die Gleichung $\sum_{j=1}^{n} y_j = k$ nicht die Forderung realisierbar, dass von den vorgegebenen n Variable genau k Variable positive Werte erhalten.

Ein häufig in der Produktion auftretendes Problem ist das sogenannte *Fixkostenproblem* (Fixed Charge Problem). Bei einer Serienproduktion ist die Annahme eines linearen Kostenverlaufs in Abhängigkeit von der produzierten Menge nur realistisch, wenn gewisse Anfangs- oder Rüstkosten berücksichtigt werden. Sie fallen beim Anlaufen der Produktion an und werden auch *auflagefixe* Kosten genannt.

Der zulässige Bereich einer Optimierungsaufgabe sei durch das Restriktionensystem

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j &\ge b_i\,, && i=1,...,m \\ 0 \le x_j &\le s_j\,, && j=1,...,n \end{aligned}$$

vorgegeben. Er ist offensichtlich beschränkt. Die Existenz von zulässigen Lösungen sei gegeben. Über diesem zulässigen Bereich soll eine separable Funktion $F(x)$ der Gestalt

$$F(x) = \sum_{j=1}^{n} f_j(x_j)$$

mit den speziellen Fixkostenfunktionen in einer Veränderlichen

$$f_j(x_j) = \begin{cases} 0 & , \quad \text{falls } x_j = 0 \\ d_j + c_j x_j & , \quad \text{falls } x_j > 0 \end{cases} \quad , \quad j=1,...,n$$

minimiert werden. Dabei gelte $c_j \ge 0$, und $d_j > 0$, $j=1,...,n$. Jede der Funktionen $f_j(x_j)$ ist monoton nicht fallend über dem Bereich $x_j \in [0, s_j]$ und unstetig an der Stelle $x_j = 0$.

Hinsichtlich der Optimalmengen ist zum beschriebenen Optimierungsproblem das folgende Modell äquivalent:

$$\begin{aligned} z = \sum_{j=1}^{n} (c_j x_j + d_j y_j) &\rightarrow \min \\ \sum_{j=1}^{n} a_{ij} x_j &\geq b_i\,, \qquad i = 1, ..., m \\ x_j - s_j y_j &\leq 0\,, \qquad j = 1, ..., n \\ x_j &\geq 0\,, \qquad j = 1, ..., n \\ y_j &\in \{0\,;\,1\}\,, \qquad j = 1, ..., n \end{aligned}$$

In der zugeordneten gemischt ganzzahligen linearen 0-1-Optimierungsaufgabe setzt ein Wert $y_j = 0$ automatisch den Wert $x_j = 0$ fest und es ergibt sich korrekterweise $f_j(x_j) = 0$. Ein kleines Problem gibt es im Fall $y_j = 1$. Dies zieht nicht automatisch die Forderung $x_j > 0$ nach sich, womit für $x_j = 0$ nicht der korrekte Wert von $f_j(x_j) = 0$, sondern der Wert $f_j(x_j) = d_j$ wiedergegeben wird.
Die eventuell auftretende, aber nicht gewollte Lösung mit den Komponenten $y_k = 1$ und $x_k = 0$ ist wegen $d_k > 0$ nicht optimal. Ändert man in dieser Lösung nur den Wert der binären Variable auf $y_k = 0$ und lässt $x_k = 0$, dann wird der Sachverhalt bezüglich der Funktion $f_k(x_k)$ jetzt korrekt wiedergegeben und im zugeordneten Optimierungsproblem hat sich der Zielfunktionswert nach dieser Änderung um den Wert d_k verbessert.

2.3 Permutationsprobleme

Ein *lineares Zuordnungsproblem* entsteht durch die mathematische Modellierung der folgenden Problemsituation:

> Zur Erledigung von n verschiedenen Aufträgen A_i stehen n Maschinen B_j zur Verfügung, von denen jede zur Erledigung jedes Auftrages eingesetzt werden kann. Ist für jedes Paar (i, j) der Aufwand c_{ij} bekannt, der beim Einsatz der Maschine B_j zur Erledigung des Auftrages A_i entsteht, dann soll eine solche Zuordnung je einer Maschine zu je einem Auftrag ermittelt werden, für welche die Aufwandssumme minimal ist.

Die beschriebenen Aufwandsgrößen lassen sich in einer quadratischen n-reihigen Matrix C anordnen.
Eine Auswahl von Elementen einer Matrix C heißt *unabhängig*, wenn in jeder Zeile und in jeder Spalte der Matrix C höchstens eines der ausgewählten Elemente steht.
In einer quadratischen Matrix der Ordnung n existieren maximal n unabhängige Elemente. Ist $(j_1, j_2, ..., j_n)$ eine beliebige Permutation der Zahlen $\{1, 2, ..., n\}$, dann bilden die Elemente $c_{1j_1}, c_{2j_2}, ..., c_{nj_n}$ eine Auswahl von n unabhängigen Elementen der Matrix C.
Unter dem linearen Zuordnungsproblem versteht man die Bestimmung von n unabhängigen Elementen einer quadratischen Matrix C, so dass deren Summe minimal ausfällt.

Es sei S_n die Menge aller Permutationen der Zahlen $\{1, ..., n\}$. Das lineare Zuordnungsproblem kann durch das folgende diskrete Optimierungsmodell ausgedrückt werden:

Modell L-ZOP (Lineares Zuordnungsproblem):

$$\begin{aligned} z = \sum_{i=1}^{n} c_{ij_i} &\longrightarrow \min \\ (j_1, ..., j_n) &\in S_n \end{aligned} \tag{2.7}$$

Eine andere mathematische Formulierung erhält man, wenn für jedes Paar (i, j) die folgenden binären Variablen eingeführt werden:

$$x_{ij} = \begin{cases} 1\,, & \text{Auftrag } A_i \text{ wird Maschine } B_j \text{ zugeordnet} \\ 0\,, & \text{anderenfalls} \end{cases}$$

Eine derartige Matrix X repräsentiert genau dann eine *zulässige Zuordnung*, wenn sich in jeder Zeile und in jeder Spalte genau ein Eintrag mit dem Wert Eins befindet.

$$\begin{aligned} z = \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \quad \to \quad \min \\ & \sum_{j=1}^{n} x_{ij} = 1\,, \qquad i = 1, ..., n \\ & \sum_{i=1}^{n} x_{ij} = 1\,, \qquad j = 1, ..., n \\ & x_{ij} \in \{0; 1\}\,, \qquad i = 1, ..., n\ ,\ j = 1, ..., n \end{aligned} \tag{2.8}$$

Wegen der totalen Unimodularität der zugehörigen Restriktionsmatrix kann man die Diskretheitsbedingungen an die Variablen durch Nichtnegativitätsbedingungen ersetzen. Jede zu einem Eckpunkt gehörige zulässige Basislösung der so entstandenen linearen Optimierungsaufgabe erfüllt automatisch die eigentlich zu fordernden Diskretheitsbedingungen.

Bei dem *quadratischen Zuordnungsproblem* handelt es sich um eine wesentlich schwierigere Aufgabe. Im Bereich der innerbetrieblichen Standortplanung sollen n gleich große Maschinen auf n gleichartigen Plätzen so angeordnet werden, dass die Summe der Transportkosten zwischen den Maschinen, also nach Zuordnung der Maschinen zu den einzelnen Plätzen, minimal wird. Dabei wird vorausgesetzt, dass die Transportkosten proportional zu der zwischen den einzelnen Maschinen zu transportierenden Mengen als auch proportional zu den zurückzulegenden Entfernungen zwischen den einzelnen Plätzen ist.

Zur Formulierung eines Permutationsmodells sind zwei Matrizen G und H festzulegen. Mit g_{ik}, $i = 1, ..., n$, $k = 1, ..., n$, seien die von Maschine i zu Maschine k zu transportierenden Mengen bekannt. Es wird dabei $g_{ii} = 0$, $i = 1, ..., n$, vereinbart. Die Entfernungen vom Platz j zum Platz l wird durch die Größe h_{jl}, $j = 1, ..., n$, $l = 1, ..., n$, erfasst. Auch hier sei $h_{jj} = 0$, $j = 1, ..., n$. Wird die Maschine i auf den Platz j_i und die Maschine k auf den Platz j_k gestellt, dann entstehen zwischen diesen beiden Maschinen Transportkosten in Höhe von $g_{ik} \cdot h_{j_i j_k}$.

Mit S_n als Menge aller Permutationen der Zahlen $\{1, ..., n\}$ und den beiden n-reihigen Transportmengen- beziehungsweise Entfernungsmatrizen G und H kann die folgende Modellvariante formuliert werden:

Modell Q-ZOP (Quadratisches Zuordnungsproblem):

$$z = \sum_{i=1}^{n} \sum_{k=1}^{n} g_{ik}\, h_{j_i j_k} \longrightarrow \min \qquad (j_1, ..., j_n) \in S_n \tag{2.9}$$

Zur Formulierung einer äquivalenten 0-1-Optimierungsaufgabe mit quadratischer Zielfunktion sind die folgenden Variablen zu vereinbaren:

$$x_{ij} = \begin{cases} 1\,, & \text{Maschine i wird dem Platz j zugeordnet} \\ 0\,, & \text{sonst} \end{cases} \quad , \quad i = 1, ..., n\,,\ j = 1, ..., n$$

Das Restriktionensystem wird vom linearen Zuordnungsproblem übernommen. Die Diskretheitforderungen an die Variablen kann man hier allerdings nicht fallen lassen.
Für die Formulierung der Zielfunktion ist folgendes zu beachten: Nur dann, wenn die Maschine i dem Platz j und gleichzeitig die Maschine k dem Platz l zugeordnet wird, also wenn $x_{ij} = 1$ und $x_{kl} = 1$ gilt, dann fallen die Kosten $g_{ik} \cdot h_{jl}$ an.
Damit ergibt sich die folgende äquivalente binäre Formulierung des quadratischen Zuordnungsproblems:

$$\begin{aligned} z = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} g_{ik} h_{jl} x_{ij} x_{kl} \ &\rightarrow \ \min \\ \sum_{j=1}^{n} x_{ij} = 1\,, &\qquad i = 1, ..., n \\ \sum_{i=1}^{n} x_{ij} = 1\,, &\qquad j = 1, ..., n \\ x_{ij} \in \{0; 1\}\,, &\qquad i = 1, ..., n\ ,\ j = 1, ..., n \end{aligned} \tag{2.10}$$

Ein weiteres, sehr schwieriges Optimierungsproblem ist das sogenannte *Rundreiseproblem*, dass sich vielfältig modellieren lässt. An dieser Stelle wird ein Permutationsmodell und ein binäres lineares Optimierungsmodell entwickelt.

Das Rundreiseproblem, auch *Traveling Salesman Problem* genannt, erhielt seinen Namen durch die folgende Aufgabenstellung:

> Gegeben sind n Orte, die – ausgehend von einem von ihnen – jeder genau einmal aufgesucht werden sollen, so dass der Ausgangspunkt erst am Ende der Reise wieder erreicht wird und die Länge des zurückgelegten Weges minimal ist.

Um das Rundreiseproblem zu modellieren, müssen die Entfernungen zwischen den einzelnen Orten $c_{ij}\,,\ i = 1, ..., n\,,\ j = 1, ..., n\,,\ i \neq j$, erfasst werden. Dabei ist durchaus $c_{ij} \neq c_{ji}$ denkbar, was zum Beispiel durch Einbahnsstraßen verursacht werden kann. Dann liegt ein *asymmetrisches* Rundreiseproblem vor.

Die "Entfernungen" $c_{ii}\,,\ i = 1, ..., n$, sind natürlich stets Null. Fahrten vom Ort i zum Ort i sind sinnlos. Um diese in einem noch zu formulierenden Modell auszuschließen, wird $c_{ii} = M\,,\ i = 1, ..., n$, gesetzt, wobei M eine hinreichend große Zahl darstellt. Mit dieser Manipulation liegt nun wie beim linearen Zuordnungsproblem eine n-reihige Kostenmatrix C vor.

Eine Rundreise wird nicht allein durch die Festlegung von n unabhängigen Elementen c_{ij_i}, $i = 1, ..., n$, $(j_1, ..., j_n) \in S_n$, der Matrix C bestimmt. Es reicht also nicht aus, in jeder Zeile und in jeder Spalte der Matrix C genau ein Element auszuwählen. Es können dann noch sogenannte *Kurzzyklen* der Art

$$\{(j_1, j_2), (j_2, j_3), ..., (j_k, j_1)\} \quad \text{mit} \quad k < n$$

auftreten. Ein solcher Kurzzyklus verursacht die vorzeitige Rückkehr zum Ausgangsort j_1, obwohl $n - k$ Orte noch nicht besucht wurden. Eine vollständige Rundreise bilden also nur Indexpaare der Form $\{(j_1, j_2), (j_2, j_3,)..., (j_{n-1}, j_n), (j_n, j_1)\}$, wobei $(j_1, ..., j_n)$ eine Permutation der Zahlen $\{1, ..., n\}$ ist. Dabei kann noch ohne Beschränkung der Allgemeinheit durch $j_1 = 1$ der erste Ort als Ausgang- und Endpunkt einer Rundreise festgelegt werden.

Mit der quadratischen Matrix C der Ordnung n und der besprochenen Charakterisierung einer Rundreise kann das Rundreiseproblem als Permutationsmodell formuliert werden:

Modell TSP (Rundreiseproblem):

$$\sum_{i=1}^{n-1} c_{j_i j_{i+1}} + c_{j_n j_1} \longrightarrow \min$$
$$(j_1, ..., j_n) \in S_n\,,\ j_1 = 1 \tag{2.11}$$

Das lineare Zuornungsproblem ist eine Relaxation des Rundreiseproblems. Damit ist auch klar, dass zur Formulierung eines binären Optimierungsmodells zum Rundreiseproblem das in binären Variablen formulierte Zuordnungsproblem (2.8) verwendet werden kann. Dabei haben die Variablen die folgende Bedeutung:

$$x_{ij} = \begin{cases} 1\,, & \text{Rundreise enthält die Strecke von i nach j} \\ 0\,, & \text{sonst} \end{cases} \quad , \quad i = 1, ..., n\,,\ j = 1, ..., n$$

In Verbindung mit den 0-1-Forderungen an die Variablen garantieren die ersten n Gleichungen in (2.8), dass jeder Ort genau einmal verlassen wird. Die letzten n Gleichungen verlangen, dass jeder Ort genau einmal angefahren wird.

Zur endgültigen Formulierung des Rundreiseproblems in binären Variablen müssen noch Bedingungen zur Kurzzyklenfreiheit hinzugefügt werden.
Um zu verhindern, dass eine Variable x_{ii} den Wert Eins annimmt, wurden bereits die Bewertungen in der Hauptdiagonale der Kostenmatrix hinreichend groß gewählt. Zur Vermeidung von Kurzzyklen unter Einbeziehung von mindestens zwei Orten ist für jede Permutation $(j_1, j_2, ..., j_k)$ von k Zahlen aus der Menge $\{1, ..., n\}$, $1 < k < n$, die Forderung

$$x_{j_1 j_2} + x_{j_2 j_3} + ... + x_{j_k j_1} \leq k - 1$$

zu erfüllen.

Diese Ungleichungen können durch die folgenden speziellen linearen Restriktionen realisiert werden:

$$\sum_{i \in U} \sum_{j \in U} x_{ij} \quad \leq \quad |U| - 1 \ , \qquad \forall\, U : U \subseteq \{1, ..., n\} \ , \qquad \text{mit } 2 \leq |U| \leq \lfloor \tfrac{n}{2} \rfloor \tag{2.12}$$

Mit der Bezeichnung $|U|$ wird die Mächtigkeit der Menge U erfasst. In der durch die Menge U induzierten Teilmatrix C_U der Kostenmatrix C dürfen höchstens $|U| - 1$ unabhängige Elemente ausgewählt werden. Ein Auswahl von $|U|$ unabhängigen Elementen liefert mit Sicherheit einen Kurzzyklus.
Durch ein Verbot von Kurzzyklen der Länge $k \leq \lfloor \frac{n}{2} \rfloor$ können keine Kurzzyklen der Länge $n - k$ auftreten. Damit genügt es in (2.12) die zu untersuchenden Teilmengen U auf die Mächtigkeit von maximal $\frac{n}{2}$, falls n eine gerade Zahl beziehungsweise auf $\frac{n-1}{2}$, falls n eine ungerade Zahl ist, einzuschränken.

Problematisch ist hier allerdings die Anzahl der Nebenbedingungen. Die *exponentiell* vielen Zyklusbedingungen machen die Lösung des Rundreiseproblems äußerst schwierig.

Eine weitere Variante zur Verhinderung von Kurzzyklen wird durch die folgenden linearen Ungleichungen beschrieben:

$$\sum_{i \in U} \sum_{j \notin U} x_{ij} \quad \geq \quad 1 \ , \qquad \forall\, U : U \subseteq \{1, ..., n\} \ , \qquad \text{mit } 2 \leq |U| \leq \lfloor \tfrac{n}{2} \rfloor \tag{2.13}$$

Die Formulierung des Rundreiseproblems als binäre lineare Optimierungsaufgabe der Form (2.8), (2.12) beziehungsweise (2.8), (2.13) mit dem *vollständigen* System von Restriktionen zur Verhinderung von Kurzzyklen hat demzufolge nur eine formale Bedeutung, da selbst bei mittlerer Problemgröße der immense Umfang an zusätzlichen Ungleichungen so nicht verarbeitet werden kann.

Kapitel 3

Komplexitätstheorie

Durch den Begriff *Komplexität* wird der Aufwand zur Abarbeitung eines Algorithmus beziehungsweise zur Lösung eines Problems in Abhängigkeit vom Umfang der Eingangsinformation *größenordnungsmäßig* erfasst. Effektiv arbeitende Algorithmen und in vernünftiger Zeit lösbare Probleme werden durch den Begriff "*polynomial*" charakterisiert. Mit dem Begriff "*nichtdeterministisch polynomial*" ist es möglich, Klassen komplizierterer Algorithmen und Probleme zu untersuchen.

Das Ziel der Komplexitätstheorie ist es, für grundlegende Probleme nachzuweisen, dass zu ihrer Lösung bestimmte Mindestressourcen nötig sind. Die Hauptschwierigkeit liegt darin, dass *alle* Algorithmen für das Problem betrachtet werden müssen.

Von den wichtigsten Problemstellungen der Komplexitätstheorie ist noch kein einziges gelöst. Viele der erzielten Ergebnisse setzen solide begründete, aber unbewiesene Hypothesen voraus. Für Probleme, die als schwierig angesehen werden, wurde *nicht* bewiesen, dass sie schwierig sind. Es konnte aber gezeigt werden, das Tausende von Problemen in Wesentlichen gleich schwierig sind. Ein effizienter Algorithmus für eines dieser Probleme würde effiziente Algorithmen für alle anderen Probleme zulassen. Würde der Nachweis gelingen, dass eines dieser Probleme nicht effizient gelöst werden kann, dann wäre keines dieser Probleme effizient lösbar.

Einen Meilenstein der Komplexitätstheorie stellt das Buch *Computers and Intractability* von M.R. Garey, D.S. Johnson [GaJo79] dar, das eine der ersten Monographien zu diesem Thema ist.

3.1 Grundbegriffe und Notationen

Definition 3.1
Es sei $\mathcal{M} = \{f \mid f : \mathbb{N} \longmapsto \mathbb{R}\}$ die Menge der reellwertigen Funktionen über dem Bereich der natürlichen Zahlen. Für $g \in \mathcal{M}$ sei

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

Die Menge $\mathcal{O}(g)$ (gesprochen: "groß Oh"von g) enthält alle Funktionen, die asymptotisch nicht schneller wachsen als die Funktion g. Die Menge $\Omega(g)$ enthält alle Funktionen, die asymptotisch mindestens so schnell wachsen wie die Funktion g. Funktionen aus der Menge $\Theta(g)$ haben die Eigenschaft, dass sie die gleiche Wachstumsrate besitzen. Durch obige Definitionen wird der Einfluss von Faktoren und Konstanten eliminiert.

Es sei $f(n)$ die Zahl der elementaren Schritte eines Algorithmus in Abhängigkeit der Problemgröße n. Gilt $f \in \mathcal{O}(n)$, dann führt der Algorithmus maximal $\mathcal{O}(n)$ Schritte aus; er arbeitet in *linearer* Zeit. Wenn $f \in \mathcal{O}(1)$ ist, führt der Algorithmus nur eine konstante Zahl von Schritten unabhängig von der Problemgröße n aus. Er arbeitet in *konstanter* Zeit. Bei $f \in \Omega(n)$ arbeitet der Algorithmus mindestens in linearer Zeit, keinesfalls in konstanter Zeit.

Definition 3.2
Eine Funktion $f \in \mathcal{M}$ ist von polynomialer Größenordnung oder einfach *polynomial*, wenn es ein Polynom $g \in \mathcal{M}$ gibt, so dass $f \in \mathcal{O}(g)$ gilt.

Für ein Polynom $f(n) = \sum_{j=0}^{p} a_j n^j$, $a_p \neq 0$, werden mit der Konvention $f \in \mathcal{O}(n^p)$ jeder Term vom Grad kleiner p und all seine Konstanten ignoriert. Nur für "kleine" Werte n können die unterdrückten Terme in Abhängigkeit von ihren Konstanten die Funktion dominieren.

Eine Funktion $f \in \mathcal{M}$ ist dagegen *exponentiell*, wenn es Konstanten $c_1 > 0$, $c_2 > 0$, $d_1 > 1$, $d_2 > 1$ gibt, so dass $c_1 d_1^n \leq f(n) \leq c_2 d_2^n$ für $n \geq n_0$ gilt. Durch $\Omega(2^{\varepsilon n})$, $\varepsilon > 0$, wird eine untere Schranke für das asymptotische Verhalten beschrieben. Ein typisches Beispiel für einen exponentiellen Zeitaufwand ist die Erzeugung aller 0-1-Vektoren der Dimension n mittels vollständiger Enumeration.

Es hat sich eingebürgert, Algorithmen mit polynomialer Laufzeit schnell zu nennen. Dabei handelt es sich allerdings um ein grobes Raster, denn eine Laufzeit von n^{100} ist für große Zahlen n praktisch nicht vertretbar.

Die vorliegenden Ergebnisse der Zeitanalysen für Optimierungsprobleme teilt sie in zwei Klassen ein:

(i) Optimierungsprobleme, für die es Algorithmen gibt, deren Zeitaufwand im *ungünstigsten* Fall von der Größenordnung $\mathcal{O}(p(n))$ ist, wobei $p(n)$ ein Polynom in der Problemgröße n darstellt. Dazu zählen die linearen Optimierungsprobleme und viele grafentheoretische Probleme.

(ii) Optimierungsprobleme, zu deren Lösung Algorithmen notwendig sind, die nicht ohne Durchmusterung größerer Teile des zulässigen Bereichs auskommen. Das Zeitverhalten kann somit im schlechtesten Fall exponentiell von der Problemgröße n abhängen. Dies ist bei den meisten diskreten Optimierungsproblemen der Fall.

Um die wesentlichen Ideen der Komplexitätstheorie vorstellen zu können, müssen die Begriffe *Problem*, *Optimierungsproblem*, *Entscheidungsproblem*, *Algorithmus* und *Kodierung* etwas näher erklärt werden.

Definition 3.3
Unter einem *Problem* $P(\mathcal{E}, \mathcal{X})$ versteht man den Operator P, der der Menge der Eingangsinformationen $\mathcal{E}$ eine Menge von Ausgangsinformationen $\mathcal{X}$ zuordnet. Die Ausgangsinformationen beschreiben die Lösungsmenge des Problems.
Ein Problem $P(\mathcal{E}_1, \mathcal{X}_1)$ mit $\mathcal{E}_1 \subset \mathcal{E}$ und $\mathcal{X}_1 \subseteq \mathcal{X}$ heißt *Teilproblem* zum Problem $P(\mathcal{E}, \mathcal{X})$.
Wenn die Ausgangsinformation eines Problems entweder aus der Antwort "ja" oder aus der Antwort "nein" besteht, so nennt man es *Entscheidungsproblem*.
Speziell wird $P(E, X)$ mit $E \in \mathcal{E}$ und $X \in \mathcal{X}$ ein *Beispiel*, oft auch konkrete Aufgabe, Instance oder Zustand genannt.

Ein Problem ist durch eine allgemeine Beschreibung all seiner Parameter und durch die Beschreibung der Eigenschaften, die die Lösung haben soll, gegeben. Die Beschreibung selbst erfolgt in einer formalen Sprache. Eine konkrete Beschreibung wird als *mathematisches Modell* bezeichnet.

Optimierungsprobleme lassen sich formal als eine Menge von Aufgaben definieren. Jede Aufgabe kann in der Form

$$\begin{array}{c} f(x) \longrightarrow \min \\ x \in M \end{array} \tag{3.1}$$

geschrieben werden, wobei M der zulässsige Bereich ist und f ein Ziel repräsentiert. Für jede Aufgabe eines Optimierungsproblems haben M und f gleiche strukturelle Eigenschaften. Sie werden durch die Festlegung von Parametern und numerischen Eingabedaten spezifiziert.

Bei der Lösung einer Optimierungsaufgabe kann man formal zwei algorithmisch zu lösende Aufgaben unterscheiden: die Bestimmung einer optimalen Lösung x^* aus der vorgegebenen Menge der zulässigen Lösungen M, auch *Suchproblem* genannt, und die Berechnung des optimalen Zielfunktionswertes f^*, das *Auswertungsproblem*. Bei den meisten Problemen ist die Auswertung der Zielfunktion für eine bekannte zulässige Lösung recht einfach. Dagegen ist die Suche nach einer zulässigen Lösung mit gewissen Eigenschaften, zum Beispiel einem vorgegebenen Zielfunktionswert oder einer optimalen Lösung, im Allgemeinen aufwendiger.

Von zentraler Bedeutung für die Komplexitätstheorie ist die Reformulierung einer Optimierungsaufgabe als Entscheidungsproblem, oder genauer als eine Folge von Entscheidungsproblemen. Unter der Formulierung als Entscheidungsproblem versteht man die Beantwortung der folgenden Frage:

$$\text{Gibt es zu einer Zahl } \alpha \text{ ein } x \in M \text{ mit } f(x) \leq \alpha \text{ ?} \tag{3.2}$$

Ein so formuliertes Entscheidungsproblem wird auch *Zielfunktionsseparierungsproblem* genannt.

Durch $M(\alpha) = \{x \in M \mid f(x) \leq \alpha\}$ wird eine Teilmenge des zulässigen Bereichs M definiert. Das Zielfunktionsseparierungsproblem fragt nach der Existenz einer zulässigen Lösung von $M(\alpha)$.

Bei Kenntnis des optimalen Zielfunktionswertes f^* von (3.1) kann jedes Zielfunktionsseparierungsproblem (3.2) sofort entschieden werden. Umgekehrt lässt sich das Auswertungsproblem, also die Bestimmung des optimalen Zielfunktionswertes f^*, durch eine Folge von Zielfunktionsseparierungsproblemen lösen.

Im Folgenden sei mit $\lceil z \rceil$ die kleinste ganze Zahl, die nicht kleiner als z ist, vereinbart.

Satz 3.1
Ist eine Einschließung des optimalen Zielfunktionswertes f^* des Optimierungsproblems (3.1) durch $\underline{f} \leq f^* \leq \overline{f}$ bekannt, dann kann f^* durch Lösen von

$$k(\varepsilon) = \left\lceil \log_2 \frac{\overline{f} - \underline{f}}{\varepsilon} \right\rceil \tag{3.3}$$

Entscheidungsproblemen (3.2) mit einer Genauigkeit von $\varepsilon > 0$ bestimmt werden.

Bemerkung 3.1
Mit Hilfe der *binären Suche* gelingt es, das Intervall $[\,\underline{f}, \overline{f}\,]$ durch die Frage "$M(\alpha) \neq \emptyset\,?$" in jedem Schritt zu halbieren. Sind f^*, $\underline{f}$ und $\overline{f}$ ganzzahlig, so wird der Wert f^* unabhängig von ε nach $k = \lceil \log_2(\overline{f} - \underline{f} + 1) \rceil$ Schritten gefunden. Die Anzahl der dann zur Bestimmung des optimalen Zielfunktionswertes f^* zu lösenden Zielfunktionsseparierungsprobleme ist durch $\mathcal{O}(\log_2(\overline{f} - \underline{f}))$ beschränkt.

Beispiel 3.1
Gegeben sei das Rucksackproblem

$$\begin{array}{rrcl} z = & c^\top x & \longrightarrow & \max \\ & g^\top x & \leq & G \\ & x & \in & \{0,1\}^n \end{array}$$

mit positiv ganzzahligen Daten c, g und G und eine zulässige Lösung $\tilde{x}$. Sucht man für das Rucksackproblem eine zulässige Lösung x mit besserem Zielfunktionswert, dann entspricht dies der Ungleichung $c^\top x > c^\top \tilde{x}$. Wegen der Ganzzahligkeit der Daten und Variablen kann diese Forderung durch die Ungleichung $c^\top x \geq c^\top \tilde{x} + 1$ verschärft werden.
Für $K \in \mathbb{N}$ lautet das zugehörige Zielfunktionsseparierungsproblem:

$$\text{Gibt es ein } x \in \{0,1\}^n \text{ mit } -c^\top x \leq -K \text{ und } g^\top x \leq G \ ?$$

Eine obere Schranke $\overline{z}$ für den optimalen Zielfunktionswert ist durch $\overline{z} = 1^\top c$ gegeben. Die untere Schranke ist trivialerweise Null. Soll das Rucksackproblem durch eine Folge von Zielfunktionsseparierungsproblemen gelöst werden, so ist deren Zahl größenordnungsmäßig durch $\mathcal{O}(\log \overline{z})$ beschränkt.

Um eine konkrete Aufgabe eines Problems zu lösen, verwendet man Algorithmen.

- Ein *Algorithmus* ist eine *endliche* Folge von ausführbaren Operationen, die jeder Eingangsinformation eine Ausgangsinformation zuordnet.
- Unter *elementaren Operationen* versteht man Lesen, Schreiben, Löschen, Additionen, Subtraktionen, Multiplikationen, Divisionen und Vergleiche von ganzen und rationalen Zahlen. Funktionswertberechnungen, Pivotisierungsschritte und insbesondere Computerbefehle können als allgemeinere Operationen aufgefasst werden.

Eine exakte formale Beschreibung eines Algorithmus erfolgte erstmals von A.M. Turing [Tur36] durch den Begriff der Turing-Maschine. Dazu ist das Sprachkonzept von entscheidender Bedeutung.

Definition 3.4
Eine endliche Menge Σ von Symbolen heißt *Alphabet*. Eine geordnete und endliche Folge von Symbolen aus Σ ist eine *Zeichenkette* (String) oder ein Wort. Mit Σ^* wird die Menge aller Zeichenketten aus Σ bezeichnet. Eine Teilmenge $L \subset \Sigma^*$ wird *Sprache* über dem Alphabet Σ genannt.

Durch ein *Kodierungsschema* C wird jede Eingangsinformation E eines Entscheidungsproblems P als Zeichenkette S der Länge $|S|$ über dem Alphabet Σ geschrieben. Die Länge $|S|$ bezeichnet man als *Größe* der Aufgabe.

Die zum Entscheidungsproblem P, dem Kodierungsschema C und dem Alphabeth Σ gehörige Sprache $L(P, C)$ umfasst alle Zeichenketten, die zu einer Eingangsinformation E des Entscheidungsproblems P gehören.

Eine *deterministische* **TURING-Maschine** kann als abstraktes Modell eines Algorithmus aufgefasst werden. Sie besteht aus einer Zustandskontrolle, einem Lese-Schreib-Kopf und einem beidseitig unendlichen Band. Das Band ist in Speicherplätze eingeteilt, die mit $..., -2, -1, 0, 1, 2, ...$ indiziert sind. Ein Algorithmus wird durch die folgenden Informationen realisiert:

1. Eine endliche Menge von Bandsymbolen Γ mit einer Teilmenge $\Sigma \subset \Gamma$ von Eingabesymbolen und einem Leezeichen $\sqcup \in \Gamma \setminus \Sigma$.

2. Eine endliche Menge Q von Zuständen mit einem ausgezeichneten Anfangszustand q_0 und zwei verschiedenen Haltezuständen q_Y und q_N.

3. Eine Überführungsfunktion $F : \Gamma \times Q \setminus \{q_Y, q_N\} \to \Gamma \times Q \times \{-1, +1\}$, gegeben in Tabellenform, wobei "-1" eine Links- und "$+1$" eine Rechtsbewegung des Lese-Schreib-Kopfes bedeuten.

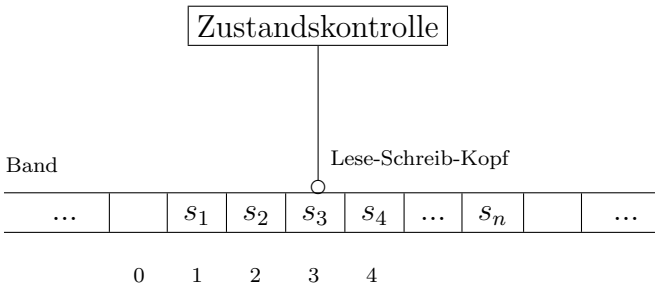


Abbildung 3.1: Schematische Darstellung einer deterministischen TURING-Maschine

Eine Aufgabe sei mit Hilfe des Alphabets Σ in die Zeichenkette $S = \{s_1, ..., s_n\} \in \Sigma^*$ kodiert. Diese Zeichenkette wird in die Speicherplätze 1 bis n des Bandes geschrieben, die übrigen Speicherplätze enthalten das Leerzeichen $\sqcup$. Der Anfangszustand q_0 wird gesetzt und der Lese-Schreib-Kopf über dem ersten Speicherplatz des Bandes positioniert. Die Berechnung übernimmt die Überführungsfunktion F. Befindet sich die deterministische TURING-Maschine im Zustand q und steht der Lese-Schreib-Kopf über einem Speicherplatz des Bandes mit dem Inhalt $s \in \Sigma$ (Abbildung 3.1), wird entsprechend dem Wert

$F(s,q) = (s',q',\triangle)$ der Folgezustand q' zugeordnet, in dem betreffenden Speicherplatz des Bandes das Symbol s' eingetragen und der Lese-Schreib-Kopf um einen Speicherplatz nach links bewegt, falls $\triangle = -1$, oder nach rechts bewegt, falls $\triangle = +1$. Der Algorithmus stoppt mit der Entscheidung "ja", wenn der Endzustand q_Y und mit "nein", wenn der Endzustand q_N angenommen wird.

Eine deterministische TURING-Maschine DTM *löst* ein Entscheidungsproblem P unter einem Kodierschema C, wenn sie bei Eingabe jeder Zeichenkette aus der Sprache $L(P,C)$ nach endlich vielen Schritten in einem der beiden Endzustände hält.

Definition 3.5
Es sei DTM eine deterministische TURING-Maschine zur Lösung des Entscheidungsproblems P, $g(S)$ die notwendige Zeit zur Berechnung der Lösung für eine in die Zeichenkette S transformierten Eingangsinformation E und $h(S)$ die Anzahl der Speicherplätze, die bis zur Lösung von P mindestens einmal besetzt werden. Die *Zeitkomplexität* von DTM für das Entscheidungsproblem P wird durch die Funktion

$$comple_{ze}(n) = \max\{g(S) \mid S \in L(P,C) \wedge |S| = n\} \tag{3.4}$$

und die *Speicherplatzkomplexität* durch die Funktion

$$comple_{sp}(n) = \max\{h(S) \mid S \in L(P,C) \wedge |S| = n\} \tag{3.5}$$

beschrieben. Gibt es ein Polynom $p(n)$ derart, dass $comple_{ze}(n) \leq p(n)$ für alle $n \in \mathbb{N}$ gilt, dann heißt DTM von *polynomialer Zeitkomplexität*. Gibt es ein Polynom $q(n)$ derart, dass $comple_{sp}(n) \leq q(n)$ für alle $n \in \mathbb{N}$ gilt, dann heißt DTM von *polynomialer Speicherplatzkomplexität*.

Die Zeit $g(S)$ kann als die Anzahl der Bewegungen des Lese-Schreib-Kopfes der TURING-Maschine bis zum Haltezustand interpretiert werden.

Bemerkung 3.2
Die Funktionen $comple_{ze}(n)$ und $comple_{sp}(n)$ beschreiben das Verhalten im *schlechtesten* Fall *(worst case)*. Sie sind monoton wachsend in n, da kleinere Probleme durch Verwendung überflüssiger Speicherplätze in größere eingebettet werden können.
Voraussetzung für eine polynomiale Zeitkomplexität ist die polynomiale Speicherplatzkomplexität. Die Besetzung und Benutzung aller Speicherplätze beinhaltet mindestens so viele Bewegungen des Lese-Schreib-Kopfes der TURING-Maschine, wie Speicherplätze benötigt werden.

Die Definitionen zur Komplexität hängen wegen der Beschreibung der Überführungsfunktion F wesentlich von der Wahl des Alphabets $\Sigma \subset \Gamma$ ab.

- In Zusammenhang mit realen Rechnern ist die Verwendung der *binären* Kodierung, das heißt einem Zwei-Symbol-Alphabet $\Sigma = \{0,1\}$ angebracht.
- Weiterhin sollen alle Daten als ganzzahlig oder rational vereinbart sein.

Eine natürliche Zahl n mit $2^r \leq n < 2^{r+1}$ wird durch die Zeichenkette $S = \{s_0, s_1, ..., s_r\}$ mit

$$n = \sum_{j=0}^{r} s_j \, 2^j \,, \quad s_j \in \{0,1\} \,, \quad j = 0, ..., r$$

dargestellt.
Für die *Kodierungslänge* $\langle n \rangle := |S|$ gilt

$$\langle n \rangle = r + 1 = \lceil \log_2(n+1) \rceil \in \mathcal{O}(\log n) \,.$$

Unter Beachtung von Vorzeichen (aber Vernachlässigung von Trennungen) ergeben sich als Kodierungslängen

$$\begin{array}{lcl}
\text{für eine ganze Zahl } g & : & \langle g \rangle := \lceil \log_2(|g|+1) \rceil + 1 \,, \\
\text{für eine rationale Zahl } r = \dfrac{p}{q} \text{ mit } p, q \in \mathbb{Z} & : & \langle r \rangle := \langle p \rangle + \langle q \rangle \,, \\
\text{für einen Vektor } c \in \mathbb{Q}^n & : & \langle c \rangle := \displaystyle\sum_{j=1}^{n} \langle c_j \rangle \,, \\
\text{für eine Matrix } A \in \mathbb{Q}^{m \times n} & : & \langle A \rangle := \displaystyle\sum_{i=1}^{m} \sum_{j=1}^{n} \langle a_{ij} \rangle \,.
\end{array}$$

Wählt man ein *geeignetes* Alphabet, dann ist die Länge der zur Problemdarstellung notwendigen Zeichenkette stets ein Wert, der polynomial von einem einfach zu bestimmenden Größenparameter abhängt. Zur Kodierung von Problemen sollen deshalb nur solche Alphabete zugelassen werden, die der folgenden Äquivalenzbedingung genügen:

> Sind S_1 und S_2 zwei Kodierungen mit den Kodierungsschemata C_1 beziehungsweise C_2 einer Eingangsinformation E des Entscheidungsproblems P mit den Kodierungslängen $|S_1|$ und $|S_2|$, dann gibt es Polynome $p(n)$ und $q(n)$, so dass $|S_1| \leq p(|S_2|)$ und $|S_2| \leq q(|S_1|)$ gelten.

Damit hängt der Begriff der polynomialen Komplexität nicht von der konkret gewählten Kodierung ab. Die Vorgehensweise rechtfertigt auch die Beschränkung auf eine einfache Kennzeichnung oder Beschreibung der Problemgröße.

Beispiel 3.2
Gegeben sei der Spezialfall $c = g$ des Rucksackproblems (siehe Beispiel 3.1):

$$\begin{array}{rccl}
z = & g^\top x & \longrightarrow & \max \\
& g^\top x & \leq & G \\
& x & \in & \{0,1\}^n
\end{array}$$

Das Zielfunktionsseparierungsproblem für den Spezialfall $c = g$ und der speziellen Wahl $K = G$ lautet:

$$\text{Gibt es ein } x \in \{0,1\}^n \text{ mit } g^\top x = G \ ?$$

Das ist das bekannte SUBSET SUM Problem. Es fragt nach der Darstellbarkeit einer ganzen Zahl durch ein System von Basiszahlen:

$$\text{Gibt es eine Teilmenge } J \subseteq \{1, ..., n\} \quad \text{mit} \quad \sum_{j \in J} g_j = G \ ?$$

Jeder Algorithmus für das Rucksackproblem löst auch das SUBSET SUM Problem durch Eingabe von Daten $c = g$. Die Antwort lautet "ja", wenn für den optimalen Zielfunktionswert $z^* = G$ gilt.
Ein Datensatz des SUBSET SUM Problems besteht aus $n + 1$ natürlichen Zahlen, wobei n der Größenparameter ist. Für die Kodierungslänge l gilt

$$l = \langle n \rangle + \sum_{j=1}^{n} \langle g_j \rangle + \langle G \rangle \, .$$

Im Allgemeinen ist G die größte darzustellende Zahl. Die Kodierungslänge ist dann durch $l \leq \langle n \rangle + (n + 1)\langle G \rangle$ beschränkt. Hat G eine Größenordnung von mindestens $\mathcal{O}(2^{2^n})$, dann erreicht die Kodierungslänge die exponentielle Größenordnung $\mathcal{O}(2^n)$. Das entspricht bereits der Mächtigkeit aller Teilmengen von $\{g_1, ..., g_n\}$.

Alle gewöhnlichen Rechnermodelle können prinzipiell auf die deterministische TURING-Maschine zurückgeführt werden. Die *Churchsche These* besagt, dass sich alle Rechnermodelle gegenseitig simulieren können. Damit ist die Menge der algorithmisch lösbaren Probleme vom Rechnermodell unabhängig. Die *erweiterte Churchsche These* postuliert, dass für je zwei Rechnermodelle R_1 und R_2 ein Polynom p existiert, so dass g Rechenschritte auf R_1 durch $p(g)$ Rechenschritte auf R_2 simuliert werden können. Bei Algorithmen in einer Programmiersprache muss man im Allgemeinen das im Computer erzeugte Maschinenprogramm als den eigentlichen Algorithmus ansehen.

Im Folgenden wird der Begriff der Laufzeit abstrahiert. Statt der Bewegungen des Lese-Schreib-Kopfes der TURING-Maschine werden elementare Rechenoperationen gezählt. Zur Berechnung der Laufzeit muss jede elementare Operation mit den Kodierungslängen der beteiligten Zahlen multipliziert werden. Der Begriff der Komplexität wird jetzt auf ein allgemeines Problem übertragen. Des Weiteren werden nur Algorithmen mit endlicher Laufzeit betrachtet.

Formal heißt ein Algorithmus A zur Lösung eines Entscheidungsproblems P *polynomial*, wenn seine Darstellung als deterministische TURING-Maschine von polynomialer Zeitkomplexität ist. Dies setzte bereits die polynomiale Speicherplatzkomplexität voraus.

Definition 3.6
Es sei A ein Algorithmus zur Lösung eines Problems P. Die Funktion $g_A : \mathcal{E} \longmapsto \mathbb{R}_+$ beschreibe den Aufwand $comple(A, E)$, auch Laufzeit genannt, den der Algorithmus A zur Lösung eines Beispiels mit Eingangsinformation E und Kodierungslänge $l(E)$ hat.
Die *worst case Komplexität* eines Algorithmus A zur Lösung eines Problems P wird durch

$$compl(A, P) := f_A(d) := \max\{g_A(E) \,|\, E \in \mathcal{E} \wedge l(E) = d\} \tag{3.6}$$

beschrieben. Der Algorithmus A heißt polynomial, wenn der Aufwand f_A von polynomialer Größenordnung ist.

Definition 3.7
Es sei AP die Menge aller Algorithmen zur Lösung eines Problems P. Die *worst case Komplexität* des Problems P wird durch

$$compl(P) := \min_{A \in AP} compl(A, P) \tag{3.7}$$

beschrieben. Ein Problem P heißt *polynomial*, wenn es einen polynomialen Algorithmus zur Lösung von P gibt. Mit $\mathcal{P}$ wird die Klasse der *polynomial lösbaren Probleme* bezeichnet.

Beispiel 3.3
Das Sortieren einer Datenfolge ist eines der am häufigsten auftretenden algorithmischen Probleme. In vielen Algorithmen steht am Anfang das Problem *SORT* des Sortierens von endlich vielen Zahlen.
Gegeben sei eine Liste $S = \{s_1, ..., s_n\}$ mit rationalen Zahlen. Gesucht ist eine Permutation $\pi : \{1, ..., n\} \rightarrowtail \{1, ..., n\}$ mit $s_{\pi(i)} \leq s_{\pi(i+1)}$, $i = 1, ..., n-1$.
Der Algorithmus MERGE (siehe u.a. [Lan02] oder [KoVy00]) teilt die zu sortierende Folge zunächst in zwei Hälften auf (Divide), die jeweils für sich sortiert werden (Conquer). Dann werden die sortierten Hälften zu einer insgesamt sortierten Folge verschmolzen (Combine). Unter der Voraussetzung, dass je zwei Zahlen in konstanter Zeit verglichen werden können, gilt $comple(\text{MERGE}, SORT) \in \mathcal{O}(n \log n)$. Unter der Annahme, dass man die Sortierung nur über den Vergleich von zwei Zahlen erhält, benötigt *jeder* Algorithmus mindestens $\Omega(n \log n)$ Vergleichsoperationen. Folglich gilt $comple(SORT) \in \Theta(n \log n)$.
Damit ist der Algorithmus MERGE bis auf konstante Faktoren sogar optimal. Ein Sortieren in linearer Zeit ist also nicht möglich.

In der Komplexitätstheorie nimmt der Begriff der *polynomialen Transformation* zwischen Problemen eine Schlüsselstellung ein.

Definition 3.8
Es seien P_1 und P_2 zwei Probleme mit den Eingangsinformationen $\mathcal{E}_1$ und $\mathcal{E}_2$, den Ausgangsinformationen $\mathcal{X}_1 = \bigcup_{E \in \mathcal{E}_1} X_1(E)$ und $\mathcal{X}_2 = \bigcup_{E \in \mathcal{E}_2} X_2(E)$.
Eine *polynomiale Transformation* $\propto$ des Problems P_1 auf das Problem P_2 ist eine Funktion $\varphi : \mathcal{E}_1 \rightarrow \mathcal{E}_2$ mit folgenden Eigenschaften:

1. Die Berechnung von $\varphi(E)$ erfolgt mit polynomialen Aufwand.
2. Für alle $E_1 \in \mathcal{E}_1$ gilt
   a) $X_2(\varphi(E_1)) = \emptyset \implies X_1(E_1) = \emptyset$,
   b) $X_2(\varphi(E_1)) \neq \emptyset \implies$ Es existiert eine Funktion $\xi : X_2 \rightarrow X_1$ mit $x^2 \in X_2(\varphi(E_1)) \wedge x^1 = \xi(x^2) \Rightarrow x^1 \in X_1(E_1)$ und $\xi(x^2)$ ist mit polynomialen Aufwand berechenbar.

$P_1 \propto P_2$ bedeuted: P_1 kann polynomial auf P_2 transformiert werden.

Betrachtet man nur Entscheidungsprobleme, dann kann eine einfachere Beschreibung für polynomiale Transformationen angegeben werden.

Definition 3.9
Ein Entscheidungsproblem P_1 ist polynomial auf ein Entscheidungsproblem P_2 transformierbar, wenn jede Kodierung S_1 einer Aufgabe von P_1 in polynomialer Zeit auf die Kodierung S_2 einer Aufgabe von P_2 transformiert werden kann und S_1 genau dann als wahr akzeptiert wird, wenn auch S_2 als wahr akzeptiert wird.

Beispiel 3.4
Gegeben seien die folgenden Entscheidungsprobleme:

HC	:	Enthält ein ungerichteter Graf $G = (V, E)$ einen Hamilton-Kreis ?
$TSP\text{-}\alpha$	:	Gibt es für ein gegebenes Rundreiseproblem eine Rundreise mit Länge $L \leq \alpha$?

Das Problem $TSP\text{-}\alpha$ ist das zum Rundreiseproblem gehörige Zielfunktionsseparierungsproblem. Es soll durch einen vollständigen Grafen beschrieben sein.
Ein Zustand S_1 von HC ist ein Graf $G = (V, E)$ mit $|V| = n$, ohne Schlingen und ohne Kantenbewertung. Zur Transformation auf das Problem $TSP\text{-}\alpha$ sind eine Matrix D und der Wert α anzugeben. Durch

$$d_{ij} := \begin{cases} 1\,, & (i,j) \in E \\ 2\,, & (i,j) \notin E \end{cases}$$

und $\alpha := n$ wird eine Überführungsfunktion φ beschrieben, die einen Zustand S_1 von HC in einen Zustand S_2 von $TSP\text{-}\alpha$ transformiert und es gilt $\varphi \in \mathcal{O}(n^2)$.
Besitzt der Graf G einen Hamilton-Kreis, dann existiert für das zugeordnete Rundreiseproblem eine Rundreise der Länge $L = n$, anderenfalls haben alle Rundreisen eine Länge $L \geq n + 1$.

Über die Komplexität der eben betrachteten Entscheidungsprobleme gibt die angegebene polynomiale Transformation keine Auskunft. Der folgende Satz zeigt aber eine praktische Bedeutung der polynomialen Transformation.

Satz 3.2
Kann ein Problem P_1 polynomial auf ein Problem P_2 transformiert werden und gehört P_2 zur Klasse der polynomial lösbaren Probleme $\mathcal{P}$, dann gehört auch P_1 zur Klasse $\mathcal{P}$.

$P_1 \propto P_2$ bedeuted auch, dass P_2 mindestens so schwierig ist wie P_1. Ist aber P_2 nicht schwierig, dann kann auch P_1 nicht schwierig sein.

Satz 3.3
Die polynomiale Transformation zwischen Problemen ist transitiv.

Definition 3.10
Zwei Probleme P_1 und P_2 heißen *polynomial äquivalent*, falls $P_1 \propto P_2 \wedge P_2 \propto P_1$ gilt.

Jedem Problem $P(\mathcal{E}, \mathcal{X})$ kann man das Entscheidungsproblem

$$\overline{P}: \text{ Wird } E \in \mathcal{E} \text{ durch } P \text{ auf } X \in \mathcal{X} \text{ abgebildet?}$$

zuordnen und umgekehrt.

Offenbar gilt $P(\mathcal{E}, \mathcal{X}) \propto \overline{\mathcal{P}}(\mathcal{E}, \mathcal{X})$ und $\overline{P}(\mathcal{E}, \mathcal{X}) \propto \mathcal{P}(\mathcal{E}, \mathcal{X})$. Die Probleme $P(\mathcal{E}, \mathcal{X})$ und $\overline{P}(\mathcal{E}, \mathcal{X})$ sind polynomial äquivalent.

Eine polynomialer Lösungsalgorithmus A für das Problem $P(\mathcal{E}, \mathcal{X})$ löst gleichzeitig das Entscheidungsproblem $\overline{P}(\mathcal{E}, \mathcal{X})$. Ist $P(\mathcal{E}, \mathcal{X}) \in \mathcal{P}$, dann gilt auch $\overline{P}(\mathcal{E}, \mathcal{X}) \in \mathcal{P}$ und umgekehrt.

Bezeichnet man die Klasse der polynomial lösbaren Entscheidungsprobleme formal mit $\overline{\mathcal{P}}$, dann sind die Klassen $\mathcal{P}$ und $\overline{\mathcal{P}}$ als äquivalent anzusehen.

Im folgenden Abschnitt werden ausschließlich Entscheidungsprobleme betrachtet. Wird in diesem Zusammenhang die Bezeichnung $\mathcal{P}$ verwendet, so ist damit die Klasse der polynomial lösbaren Entscheidungsprobleme gemeint.

3.2 Klassifizierung von Entscheidungsproblemen

Es gibt sehr viele Entscheidungsprobleme, deren Zugehörigkeit zur Klasse $\mathcal{P}$ (noch) nicht nachgewiesen werden konnte. Nicht all diese Probleme sind über alle Maßen schwierig. Viele Entscheidungsprobleme haben die folgende bemerkenswerte und schöne Eigenschaft:

> Lautet die Antwort für ein Entscheidungsproblem "ja" und gibt irgend jemand eine zulässige Lösung vor, dann kann in polynomialer Zeit überprüft werden, ob die Antwort korrekt ist.

Entscheidungsprobleme mit der oben formulierten Eigenschaft der Abschwächung der Forderung nach polynomialer Lösbarkeit sollen im Folgenden etwas genauer untersucht werden.

Zu diesem Zweck wird der Begriff des *nichtdeterministischen* Algorithmus benötigt. Grob gesprochen ist es ein Algorithmus, der am Anfang "raten" kann. Nach diesem nichtdeterministischen Schritt arbeitet er wie üblich. Der Algorithmus kann seine Suchprozedur unter Modifikation der Suchwege von Anfang an immer wiederholen. Das entspricht einem Vervielfachen des Algorithmuszustandes an vorgegebenen Verzweigungsstellen. Die Zeitmessung beginnt bei jedem Neuanfang wieder von vorn. Man zählt nicht sequentiell die verstrichene Suchzeit, sondern die kürzeste Laufzeit einer erfolgreichen Wiederholung. Sie wird auch nichtdeterministische Rechenzeit genannt.

Eine exakte formale Beschreibung ist mit Hilfe der sogenannten *nichtdeterministischen* TURING-Maschine möglich. Sie kann als abstraktes Modell eines nichtdeterministischen Algorithmus aufgefasst werden und arbeitet wie folgt:
Einer deterministischen Turing-Maschine wird ein *Orakel*-Modul mit einem Schreibkopf hinzugefügt. Im ersten Teil schreibt das Orakel-Modul eine beliebige Zeichenkette aus Σ^*, beginnend vom Speicherplatz null nach links, auf das Band. Wie bei der deterministischen Turing-Maschine wird die in die Zeichenkette $S = \{s_1, ..., s_n\}$ kodierte Eingangsinformation in die Speicherplätze 1 bis n des Bandes eingetragen. Im zweiten Teil ist das Orakel-Modul

inaktiv. Beginnend mit Zustand q_0 arbeitet die deterministische Turing-Maschine ihr durch die Überführungsfunktion eingegebenes Programm ab. Dabei wird in der Regel das Orakel mitgelesen.
Eine Berechnung heißt *akzeptierend*, wenn sie im Zustand q_Y endet. Ein durch eine nichtdeterministische Turing-Maschine realisierter Algorithmus NDTM akzeptiert eine kodierte Eingabe S, falls gilt: Mindestens eine der unendlich vielen möglichen Berechnungen durch das Orakel aus Σ^* führt zu einer akzeptierenden Berechnung. Der Aufwand für den Orakelschritt wird gleich eins gesetzt.

Für ein Entscheidungsproblem $P(\mathcal{E}, \mathcal{X})$ sei

$$\mathcal{E}_Y := \{E \in \mathcal{E} \mid E \text{ führt auf eine “ja”-Antwort }\},$$
$$\mathcal{E}_N := \{E \in \mathcal{E} \mid E \text{ führt auf eine “nein”-Antwort }\}.$$

Für $E \in \mathcal{E}$ sei eine Zulässigkeits- oder Akzeptanzprüfung für $E \in \mathcal{E}_Y$ durch die Kenntnis einer Menge von problembezogenen *Prüfdaten* $\mathcal{D}(E)$ möglich. Die problembezogenen Prüfdaten werden auch als *Zertifikate* bezeichnet.

Definition 3.11
Ein *nichtdeterministischer* Algorithmus A zur Verifikation einer Eingangsinformation E eines Entscheidungsproblems $P(\mathcal{E}, \mathcal{X})$ besteht aus den folgenden beiden Teilen:

(a) Ratemodul: Rate $D \in \mathcal{D}(E)$.

(b) Testmodul: Prüfe $E \in \mathcal{E}_Y$ mit Hilfe von D.
Stopp, falls $E \in \mathcal{E}_Y$, anderenfalls gehe zu (a).

Der Algorithmus A heißt *nichtdeterministisch polynomial*, wenn folgende Bedingungen erfüllt sind:

(i) Gilt $E \in \mathcal{E}_Y$, dann gibt es ein $D^0 \in \mathcal{D}(E)$, dessen Kodierungslänge $l(D^0)$ polynomial beschränkt in der Kodierungslänge $l(E)$ bleibt, und der Aufwand zur Bescheinigung der Korrektheit von D^0 durch das Testmodul ist polynomial beschränkt in der Kodierungslänge $l(E) + l(D^0)$.

(ii) Gilt $E \in \mathcal{E}_N$, dann verwirft das Testmodul alle durch das Ratemodul verwendeten Prüfdaten $D \in \mathcal{D}(E)$ als inkorrekt für E.

Definition 3.12
Ein *Entscheidungsproblem* P heißt *nichtdeterministisch polynomial*, wenn es zur Lösung von P einen nichtdeterministisch polynomialen Algorithmus A gibt. Die Klasse der nichtdeterministisch polynomialen Entscheidungsprobleme sei mit $\mathcal{NP}$ bezeichnet.

Der Name $\mathcal{NP}$ steht als Abkürzung für nichtdeterministisch polynomial und darf keinesfalls als “nicht polynomial” interpretiert oder gelesen werden.

Bemerkung 3.3
In Definition 3.11 wird nichts über die Kenntnis eines Verfahrens zur Berechnung aller in Frage kommenden Prüfdaten vorausgesetzt. Insbesondere wird nichts darüber gesagt, ob $D^0 \in \mathcal{D}(E)$ mit einem polynomialen Algorithmus gefunden werden kann.
Die Definition der Klasse $\mathcal{NP}$ ist offenbar unsymmetrisch in Bezug auf die Mengen $\mathcal{E}_Y$ und $\mathcal{E}_N$. Der formulierte nichtdeterministisch polynomiale Algorithmus verlangt nicht die Existenz von Prüfdaten, die eine ”nein”-Antwort in polynomialer Zeit bestätigen.

Beispiel 3.5
Gegeben seien eine Matrix $A \in \mathbb{Z}^{m \times n}$ und ein Vektor $b \in \mathbb{Z}^m$. Das Entscheidungsproblem

$$BLUG \quad : \quad \text{Gibt es ein} \quad x \in \{0;1\}^n \text{ mit } Ax \leq b \; ?$$

fragt, ob ein lineares Ungleichungssystem in binären Variablen lösbar ist.
Ist θ die betragsmäßig größte Zahl der Matrix (A, b), dann gilt für die Eingabelänge eines Beispiels $d = \langle m \rangle + \langle n \rangle + \langle A \rangle + \langle b \rangle \leq c_0\, m\, (n+1)\, \log_2 \theta \in \mathcal{O}(mn \log \theta)$.
Der folgende nichtdeterministische Algorithmus

(a) Ratemodul: Rate ein $x \in \{0;1\}^n$.
(b) Testmodul: Prüfe $Ax \leq b$.

verursacht im Testmodul den Aufwand $f(d) \leq 2\, c_0\, m\, (n+1)\, \log_2 \theta \in \mathcal{O}(mn \log \theta)$, das heißt $f(d) \in \mathcal{O}(d)$. Der Algorithmus ist nichtdeterministisch linear. Damit gehört das Entscheidungsproblem $BLUG$ zur Klasse $\mathcal{NP}$.

Die meisten der in der diskreten Optimierung untersuchten Entscheidungsprobleme gehören zur Klasse $\mathcal{NP}$. Für sehr viele dieser Entscheidungsprobleme ist nicht bekannt, ob für sie polynomiale Algorithmen existieren.

Definition 3.13
Es sei $P(\mathcal{E}, \mathcal{X})$ ein Entscheidungsproblem mit den Eingangsinformationen $\mathcal{E} = \mathcal{E}_Y \cup \mathcal{E}_N$. Dann heißt das Entscheidungsproblem $\widetilde{P}(\widetilde{\mathcal{E}}, \widetilde{\mathcal{X}})$ mit $\widetilde{\mathcal{E}} = \mathcal{E}$, $\widetilde{\mathcal{E}}_Y = \mathcal{E}_N$ und $\widetilde{\mathcal{E}}_N = \mathcal{E}_Y$ *komplementär* zu $P(\mathcal{E}, \mathcal{X})$.

Das komplementäre Entscheidungsproblem zu dem binären linearen Ungleichungssystem $BLUG$ aus Beispiel 3.5 lautet:

$$\widetilde{BLUG} \quad : \quad \text{Gilt} \quad Ax \not\leq b \text{ für alle } x \in \{0;1\}^n \; ?$$

Es ist nicht bekannt, ob $\widetilde{BLUG}$ zu $\mathcal{NP}$ gehört.

Für Entscheidungsprobleme $P \in \mathcal{P}$ ist die Situation dagegen anders. Für sie gibt es einen polynomialen Algorithmus A mit $comple(A, P) = f_A(d) = p(d)$, wobei p ein Polynom ist. Der folgende Algorithmus $\widetilde{A}$ löst das komplementäre Entscheidungsproblem $\widetilde{P}$ in polynomialer Zeit:

(i) Wende Algorithmus A an.
(ii) Stoppe nach Überschreiten des Aufwandes $p(d) + 1$ mit der Antwort "ja".

Satz 3.4
Die Klasse der polynomial lösbaren Entscheidungsprobleme $\mathcal{P}$ ist gegenüber der Komplimentbildung abgeschlossen.

Mit Hilfe der komplemetären Entscheidungsprobleme lässt sich ein Analogon zur Klasse $\mathcal{NP}$ formulieren.

Definition 3.14
Mit co-$\mathcal{NP}$ wird die Klasse aller Entscheidungsprobleme P bezeichnet, deren komplementären Entscheidungsprobleme $\widetilde{P}$ zur Klasse $\mathcal{NP}$ gehören.

Für die Klasse co-$\mathcal{NP}$ kann also die "nein"-Antwort mittels Prüfdaten mit nichtdeterministisch polynomialem Aufwand nachgewiesen werden.

Aus Satz 3.4 folgt $\mathcal{P} = \text{co-}\mathcal{P}$, wobei co-$\mathcal{P}$ analog zur Klasse $\mathcal{P}$ zu definieren ist. Das oben betrachtete Paar von Entscheidungsproblemen zur Lösbarkeit eines linearen Ungleichungssystems in binären Variablen lässt vermuten, dass die Aussage dieses Satzes nicht auf die Klasse $\mathcal{NP}$ übertragen werden kann.

Die durchaus realistische Vermutung $\mathcal{NP} \neq \text{co-}\mathcal{NP}$ konnte noch nicht nachgewiesen werden. Durch die folgende Frage wird also ein noch offenes Entscheidungsproblem der Komplexitätstheorie formuliert:

$$\text{Gilt} \quad \mathcal{NP} = \text{co-}\mathcal{NP} \ ?$$

Ein offenes Problem der Komplexitätstheore und eine der wichtigsten Fragen der Mathematik und Informatik ist, ob die Klasse $\mathcal{NP}$ eine echte Erweiterung der Klasse $\mathcal{P}$ darstellt. Würde $\mathcal{P} = \mathcal{NP}$ gelten, dann *kann* der Nachweis dieser Behauptung gleichzeitig einen "revolutionären" polynomialen Algorithmus zur Lösung aller Entscheidungsprobleme aus $\mathcal{NP}$ liefern. Weit verbreitet und akzeptiert wird aber folgendes:

$$\boxed{\textbf{Vermutung:} \quad \mathcal{P} \neq \mathcal{NP}}$$

Würde sich diese Vermutung bestätigen, dann lassen sich für eine große Zahl praktischer Probleme keine polynomialen Algorithmen finden. Für diese Probleme muss man sich dann auf den Entwurf von schnellen Näherungsverfahren konzentrieren.

Gilt $\mathcal{P} \neq \mathcal{NP}$, dann kann ein Entscheidungsproblem $P_1 \in \mathcal{NP} \setminus \mathcal{P}$ **nicht** auf ein Problem $P_2 \in \mathcal{P}$ polynomial transformiert werden.

Ein polynomialer Algorithmus A für ein Entscheidungsproblem P liefert mit polynomial beschränktem Aufwand die richtige Antwort "ja" oder "nein" für P. Der Algorithmus A kommt dabei ohne Angabe von Prüfdaten aus und entscheidet sowohl über P als auch über das komplementäre Entscheidungsproblem $\widetilde{P}$ in polynomialer Zeit. Folglich gelten die Beziehungen $\mathcal{P} \subseteq \mathcal{NP}$ und $\mathcal{P} \subseteq \text{co-}\mathcal{NP}$ und damit auch $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$. Offen ist aber die Antwort auf folgende Frage:

$$\text{Gilt} \quad \mathcal{P} = \mathcal{NP} \cap \text{co-}\mathcal{NP} \ ?$$

Für Entscheidungsprobleme aus der Klasse $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ gibt es sowohl für die "ja"-Antwort als auch für die "nein"-Antwort Prüfdaten, die in polynomialer Zeit die jeweilige Antwort bestätigen. Man sagt, ein derartiges Problem $P \in \mathcal{NP} \cap \text{co-}\mathcal{NP}$ besitzt angenehme Eigenschaften (*good characterization*) [Ed65].

Auf Grund der Vermutung $\mathcal{P} \neq \mathcal{NP}$ ist man bemüht, Probleme in $\mathcal{NP}$ zu klassifizieren, die man als besonders schwierig ansehen kann.

Definition 3.15
Ein Entscheidungsproblem P heißt $\mathcal{NP}$-*vollständig*, wenn $P \in \mathcal{NP}$ gilt und jedes andere Entscheidungsproblem aus der Klasse $\mathcal{NP}$ polynomial auf P transformiert werden kann. Die Klasse aller $\mathcal{NP}$-vollständigen Entscheidungsprobleme wird mit $\mathcal{NPC}$ bezeichnet.

Falls ein $\mathcal{NP}$-vollständiges Entscheidungsproblem P in polynomialer Zeit gelöst werden kann, dann ist mit Satz 3.3 auch jedes andere Problem aus $\mathcal{NP}$ in polynomialer Zeit lösbar:

$$P \in \mathcal{NPC} \wedge P \in \mathcal{P} \iff \mathcal{P} = \mathcal{NP}$$

Dieses Eigenschaft besagt auch, dass kein Entscheidungsproblem in $\mathcal{NP}$ schwieriger ist als eins in $\mathcal{NPC}$. Die in der Definition gestellten Forderungen erscheinen äußerst restriktiv. Damit stellt sich sofort die Frage, ob es überhaupt $\mathcal{NP}$-vollständige Entscheidungsprobleme geben kann.

Cook konnte 1971 nachweisen, dass die Klasse $\mathcal{NPC}$ nicht leer ist. In der für die Komplexitätstheorie sehr bedeutenden Arbeit [Co71] wird das folgende Erfüllbarkeitsproblem SAT (*Satisfiability*) betrachtet:

> Gibt es für einen gegebenen logischen Ausdruck $B(x_1, ..., x_n)$ in konjunktiver Normalform (Konjunktionen, Disjunktionen und Negationen) eine derartige Belegung der n logischen Variablen, so dass der logische Ausdruck den Wert $TRUE$ hat ?

Zur Überprüfung eines beliebigen logischen Ausdrucks ist praktisch nur das Enumerationsverfahren mittels Wahrheitswerten bekannt. Das erfordert einen exponentiellen Aufwand. Der Test, ob eine vorgegebene Variablenbelegung den logischen Ausdruck erfüllt, kann in polynomialer Zeit in Abhängigkeit von der Länge des Ausdrucks durchgeführt werden. Damit gehört SAT zur Klasse $\mathcal{NP}$. Cook gab als erster eine Konstruktion zur polynomialen Transformation eines beliebigen Entscheidungsproblems aus $\mathcal{NP}$ auf das Erfüllbarkeitsproblem SAT an.

Satz 3.5
Das Entscheidungsproblem SAT ist $\mathcal{NP}$-vollständig.

Mit Hilfe von binären Variablen lässt sich SAT auch wie folgt formulieren:

$$\text{Gibt es ein } x \in \{0;1\}^n \text{ mit } \sum_{j \in C_i^+} x_j + \sum_{j \in C_i^-} (1 - x_j) \geq 1 \text{ für } i = 1, ..., m \text{ ?}$$

Dabei ist $C_i^+ \cup C_i^- \subseteq \{1, ..., n\}$ und $C_i^+ \cap C_i^- = \emptyset$ für $i = 1, ..., m$ vereinbart.
Der Spezialfall $3SAT$ liegt vor, wenn $|C_i^+ \cup C_i^-| = 3$ für alle $i = 1, ..., m$ gilt. Auch dieses Problem gehört zur Klasse $\mathcal{NPC}$.

SAT und $3SAT$ dienten als erste Referenzprobleme für den Nachweis, dass viele zu diskreten Optimierungsaufgaben gehörige Entscheidungsprobleme $\mathcal{NP}$-vollständig sind.

Bemerkung 3.4
Um für ein Problem P die Zugehörigkeit zur Klasse $\mathcal{NPC}$ zu zeigen, nutzt man die Transitivität der polynomialen Transformation aus:

(a) Zeige $P \in \mathcal{NP}$.

(b) Suche ein bekanntes Problem $\hat{P} \in \mathcal{NPC}$ und zeige $\hat{P} \propto P$.

Leider sind fast alle zu praxisrelevanten Optimierungsproblemen gehörenden Entscheidungsprobleme $\mathcal{NP}$-vollständig. Eine umfangreiche Liste von Entscheidungsproblemen, die zur Klasse $\mathcal{NPC}$ gehören, wurde erstmals in [GaJo79] publiziert. Ausführliche Referenzen findet man auch bei [Iba87]. Beweise für die $\mathcal{NP}$-Vollständigkeit verschiedener Entscheidungsprobleme kann man bei [Kar75] und [PaSt82] nachlesen.

Bemerkung 3.5
Unter der Voraussetzung $\mathcal{P} \neq \mathcal{NP}$ gilt $\mathcal{P} \cap \mathcal{NPC} = \emptyset$. In [Lad75] wurde der Nachweis geführt, dass es dann auch Entscheidungsprobleme in der Klasse $\mathcal{NP} \setminus \mathcal{P}$ gibt, die nicht $\mathcal{NP}$-vollständig sind. Damit gilt $\mathcal{P} \cup \mathcal{NPC} \neq \mathcal{NP}$.

Definition 3.16
Ein Entscheidungsproblem P heißt co-$\mathcal{NP}$-vollständig, wenn $P \in$ co-$\mathcal{NP}$ gilt und jedes andere Entscheidungsproblem aus der Klasse co-$\mathcal{NP}$ polynomial auf P transformiert werden kann. Die Klasse aller co-$\mathcal{NP}$-vollständigen Entscheidungsprobleme wird mit co-$\mathcal{NPC}$ bezeichnet.

Aus der Definition folgt, dass ein Entscheidungsproblem P genau dann co-$\mathcal{NP}$-vollständig ist, wenn das zugehörige komplementäre Entscheidungsproblem $\mathcal{NP}$-vollständig ist.

Bemerkung 3.6
Unter der Veraussetzung $\mathcal{NP} \neq$ co-$\mathcal{NP}$ kann man zeigen, dass kein co-$\mathcal{NP}$-vollständiges Problem in der Klasse $\mathcal{NP}$ liegt. Analog ist auch kein $\mathcal{NP}$-vollständiges Problem in der Klasse co-$\mathcal{NP}$ zu finden:

$$\mathcal{NP} \neq \text{co-}\mathcal{NP} \implies \mathcal{NPC} \cap \text{co-}\mathcal{NP} = \emptyset \ \wedge \ \text{co-}\mathcal{NPC} \cap \mathcal{NP} = \emptyset$$

Die Abbildung 3.2 zeigt die vermutliche Struktur der vorgestellten Komplexitätsklassen für Entscheidungsprobleme unter den Annahmen $\mathcal{NP} \neq$ co-$\mathcal{NP}$ und $\mathcal{P} \neq \mathcal{NP} \cap$ co-$\mathcal{NP}$.

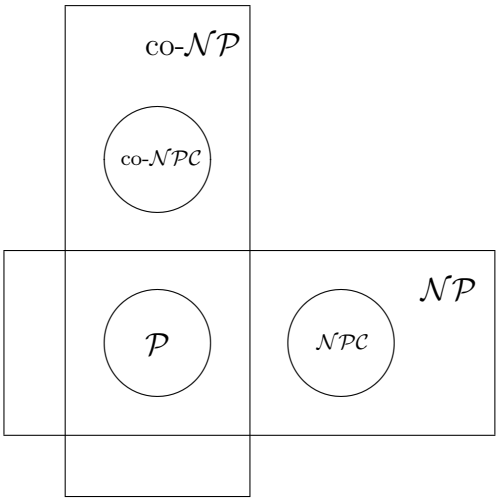


Abbildung 3.2: Klasseneinteilung für Entscheidungsprobleme

Abschließend soll noch ein Begriff ergänzt und auf eine weitere Verfeinerung der Klasseneinteilung hingewiesen werden. Betrachtet man Entscheidungsprobleme aus der Klasse $\mathcal{NPC}$, so kann man bezüglich ihrer algorithmischen Lösung noch bemerkenswerte Unterschiede entdecken. Bei vielen praktischen Anwendungen benötigt man nur Zahlen "moderater" Größe. Es stellt sich somit die Frage, ob $\mathcal{NP}$-vollständige Entscheidungsprobleme auch noch schwierig sind, wenn die in der Eingabe vorkommenden Zahlen polynomial in der Eingabelänge beschränkt sind.
Für das SUBSET SUM Problem (siehe Beispiel 3.2) gibt es einen auf der dynamischen Optimierung basierenden Algorithmus mit Aufwand $\mathcal{O}(nG)$ [KePfPi04]. Hat die Zahl G eine Größenordnung von mindestens $\mathcal{O}(2^{2^n})$, dann ist dies ein exponentieller Algorithmus. Hier spielt also die Größe von G ein entscheidende Rolle.

Definition 3.17
Es sei A ein Algorithmus zur Lösung eines Entscheidungsproblems P. Für ein Beispiel mit Eingangsinformation E und Kodierungslänge $l(E)$ sei $\delta(E)$ die betragsmäßig größte ganze Zahl in E. Der Algorithmus A heißt *pseudopolynomial*, wenn der Aufwand compl(A,P) durch ein Polynom in den Größen $l(E)$ und $\delta(E)$ beschränkt ist.

Anschaulich kann man dies deuten, in dem man die betragsmäßig größte Zahl in unärer Kodierung darstellt. Für das SUBSET SUM Problem ist in diesem Sinne der oben erwähnte Algorithmus mit Aufwand $\mathcal{O}(nG)$ pseudopolynomial, da er polynomial in der Problemgröße n und der Zahl G in unärer Kodierung ist.

Definition 3.18
Ein Entscheidungsproblem P, das auch dann $\mathcal{NP}$-vollständig bleibt, wenn die betragsmäßig größte kodierte Zahl durch ein Polynom in der Kodierungslänge der Eingangsinformation beschränkt ist, heißt *streng* $\mathcal{NP}$-vollständig.

Das Entscheidungsproblem HC (siehe Beispiel 3.4) ist streng $\mathcal{NP}$-vollständig, da zu seiner Darstellung keine großen Zahlen benötigt werden. Auch die wichtigen Referenzprobleme SAT und $3SAT$ sind streng $\mathcal{NP}$-vollständig. Das SUBSET SUM Problem ist zwar $\mathcal{NP}$-vollständig, aber nicht streng $\mathcal{NP}$-vollständig.

Ein pseudopolynomialer Algorithmus liefert für "polynomial kleine Zahlen" in der Eingabe stets einen polynomialen Algorithmus. Im Falle $\mathcal{P} \neq \mathcal{NP}$ bedeutet dies auch, dass es für streng $\mathcal{NP}$-vollständige Probleme nicht einmal pseudopolynomiale Algorithmen gibt.

Die Grenze zwischen einfachen und schwierigen Teilproblemen verläuft oft an überraschenden Stellen. So ist das analog zum Entscheidungsproblem $3SAT$ zu formulierende Entscheidungsproblem $2SAT$ bereits polynomial lösbar.
Bei $\mathcal{NP}$-vollständigen Problemen lohnt sich demzufolge die Überlegung, ob vielleicht nur Algorithmen für eine ganz spezielle Problemvariante benötigt werden. Die Untersuchung der Komplexität des spezielleren Problems kann durchaus erfolgreich verlaufen.

3.3 Klassifizierung von Optimierungsproblemen

Nach dem im letzten Abschnitt nur Entscheidungsprobleme klassifiziert wurden, sollen jetzt die Optimierungsprobleme mit einbezogen werden.
Dazu wird jetzt der Begriff der polynomialen Transformation erweitert.

Definition 3.19
Ein Problem P_1 heißt *polynomial reduzierbar* auf ein Problem P_2, wenn es einen Algorithmus A_1 zur Lösung von P_1 gibt, der auf einen Algorithmus A_2 zur Lösung von P_2 zurückgreift und folgende Eigenschaften hat:

1. Die Komplexität des Algorithmus A_1 zur Lösung von P_1 sei ohne die Aufrufe von Algorithmus A_2 zur Lösung von P_2 durch ein Polynom $p(d)$ in der Kodierungslänge $l(E) = d$ der Eingangsinformation E beschränkt.
2. Die Anzahl der Aufrufe von Algorithmus A_2 zur Lösung von P_2 ist durch ein Polynom $q(d)$ beschränkt.
3. Die Kodierungslänge der Eingangsinformation für jeden Aufruf des Algorithmus A_2 zur Lösung von P_2 ist durch ein Polynom $r(d)$ beschränkt.

$P_1 \prec P_2$ bedeuted: P_1 kann polynomial auf P_2 reduziert werden.

Wenn ein Problem P_1 polynomial auf ein Problem P_2 transformierbar ist, dann ist P_1 auch auf P_2 polynomial reduzierbar. Der für P_2 existierende Algorithmus muss dann nur einmal aufgerufen werden.
Die polynomiale Reduktion ist auch transitiv: $P_1 \prec P_2 \wedge P_2 \prec P_3 \Longrightarrow P_1 \prec P_3$

Bemerkung 3.7
Wenn es einen Algorithmus A_2 zur Lösung von P_2 mit einer Komplexität $f_{A_2}(d)$ gibt, dann erhält man einen Algorithmus zur Lösung von P_1, dessen Komplexität $f_{A_1}(d)$ sich folgendermaßen abschätzen lässt:

$$f_{A_1}(d) \leq p(d) + q(d) \cdot f_{A_2}(r(d))$$

Wird der Algorithmus A_2 zur Lösung von P_2 nicht sehr oft aufgerufen und ist der zusätzliche Aufwand zur Lösung von P_2 erträglich, das heißt die Komplexität $f_{A_2}(d)$ ist polynomial beschränkt, dann kann auf diesem Weg das Problem P_1 vernünftig gelöst werden. Dies impliziert, dass Problem P_1 effektiv lösbar ist, wenn Problem P_2 effektiv lösbar ist. Problem P_2 kann aber nicht effektiv lösbar sein, wenn Problem P_1 algorithmisch schwierig ist.

Das in (3.2) formulierte Zielfunktionsseparierungsproblem lässt sich polynomial auf das zugehörige Optimierungsproblem (3.1) reduzieren. Es genügt ein Aufruf eines Algorithmus zur Lösung des Optimierungsproblems, um jedes einzelne Zielfunktionsseparierungsproblem sofort zu entscheiden. Wird das Optimierungsproblem mit P und das zugehörige Zielfunktionsseparierungsproblem mit SP bezeichnet, dann gilt

$$SP \prec P\,.$$

Optimierungsprobleme sind folglich mindestens so schwierig wie die zugehörigen Zielfunktionsseparierungsprobleme.

Andererseits kann man das Optimierungsproblem gegebenenfalls (siehe Satz 3.1) durch eine Folge von Zielfunktionsseparierungsproblemen lösen. Hat man keine Kenntnisse über die annehmbaren Zielfunktionswerte eines Optimierungsproblems, dann ist das Optimierungsproblem zwar auf das Zielfunktionsseparierungsproblem reduzierbar, aber nicht sicher in polynomialer Zeit. Sind die zu separierenden Zielfunktionswerte durch eine Größe beschränkt, die polynomial von der Eingabelänge des Optimierungsproblems abhängt, dann ist das Optimierungsproblem auf sein zugehöriges Zielfunktionsseparierungsproblem polynomial reduzierbar.

Satz 3.6
Kann ein Problem P_1 polynomial auf ein Problem P_2 reduziert werden und gehört P_2 zur Klasse der polynomial lösbaren Probleme $\mathcal{P}$, dann gehört auch P_1 zur Klasse $\mathcal{P}$.

Ist das Optimierungsproblem auf das zugehörige Zielfunktionsseparierungsproblem *polynomial* reduzierbar und gibt es für das Zielfunktionsseparierungsproblem einen polynomialen Algorithmus, dann gehört das Optimierungsproblem auch zur Klasse der polynomial lösbaren Probleme $\mathcal{P}$.

Wichtig ist die Kontraposition von Satz 3.6. Gehört das Problem P_1 nicht zur Klasse der polynomial lösbaren Probleme $\mathcal{P}$, und ist Problem P_1 auf Problem P_2 polynomial reduzierbar, dann gehört auch P_2 nicht zu $\mathcal{P}$.

Definition 3.20
Ein Optimierungsproblem P heißt $\mathcal{NP}$-*schwer*, wenn das zugehörige Zielfunktionsseparierungsproblem $\mathcal{NP}$-vollständig ist.

Alle $\mathcal{NP}$-schweren Optimierungsproblem sind mindesten so schwierig wie die $\mathcal{NP}$-vollständigen Entscheidungsprobleme. Mit Definition 3.15 ist klar, dass alle $\mathcal{NP}$-*schweren* Optimierungsprobleme mindestens so schwierig wie alle $\mathcal{NP}$-*vollständigen* Entscheidungsprobleme sind. Da das zum Rundreiseproblem gehörige Zielfunktionsseparierungsproblem TSP-α (siehe Beispiel 3.4) $\mathcal{NP}$-*vollständig* ist, gehört das Rundreiseproblem zur Klasse der $\mathcal{NP}$-*schweren* Optimierungsprobleme.

Definition 3.21
Ein Optimierungsproblem P heißt $\mathcal{NP}$-*hart*, wenn jedes Entscheidungsproblem aus der Klasse $\mathcal{NP}$ polynomial auf das zugehörige Zielfunktionsseparierungsproblem transformiert werden kann.

Bei der Klasse $\mathcal{NP}$-hart wird nicht verlangt, dass das Zielfunktionsseparierungsproblem zu $\mathcal{NP}$ gehört. Dieses Entscheidungsproblem kann also schwieriger als alle zur Klassse $\mathcal{NP}$ gehörigen Entscheidungsprobleme sein.

Definition 3.22
Ein Optimierungsproblem P heißt $\mathcal{NP}$-*leicht*, wenn es ein Entscheidungsproblem $\hat{P} \in \mathcal{NP}$ gibt, auf das P polynomial reduziert werden kann.

Würde die Klasse $\mathcal{NP}$ nur effizient lösbare Entscheidungsprobleme beinhalten, dann lässt sich auch ein $\mathcal{NP}$-*leichtes* Optimierungsproblem einfach lösen. $\mathcal{NP}$-leichte Optimierungsprobleme sind also nicht wesentlich schwieriger als die Entscheidungsprobleme in $\mathcal{NP}$.
Wird ein Optimierungsproblem durch höchstens polynomial viele Aufrufe des zugehörigen Zielfunktionsseparierungsproblems gelöst und gehört das Zielfunktionsseparierungsproblem zur Klasse $\mathcal{NP}$, dann liegt ein $\mathcal{NP}$-leichtes Optimierungsproblem vor und es gilt

$$P \prec SP\,.$$

Definition 3.23
Ein Optimierungsproblem P heißt $\mathcal{NP}$-*äquivalent*, wenn es sowohl $\mathcal{NP}$-schwer als auch $\mathcal{NP}$-leicht ist.

Folgende (diskrete) Optimierungsprobleme sind NP-schwer:

- das gemischt ganzzahlige lineare Optimierungsproblem
- das rein ganzzahlige lineare Optimierungsproblem
- das lineare 0-1-Optimierungsproblem
- das Rucksackproblem
- das asymmetrische Rundreiseproblem
- das symmetrische Rundreiseproblem
- das Briefträgerproblem für gemischte Grafen
- fast alle Tourenplanungsprobleme
- fast alle Standortprobleme
- ...

Bemerkung 3.8
In Analogie zu Definition 3.18 soll ein Optimierungsproblem *streng* $\mathcal{NP}$-schwer genannt werden, wenn das zugehörige Zielfunktionsseparierungsproblem streng $\mathcal{NP}$-vollständig ist. Die oben angegebenen Rundreiseprobleme sind streng $\mathcal{NP}$-schwer, nicht aber das Rucksackproblem.

Ein wichtiger Aspekt der Komplexitätstheorie ist, dass man lernt, zwischen "einfachen" und "schwierigen" Problemen zu unterscheiden. Für schwierige Optimierungsprobleme in praktisch relevanten Größenordnungen muss man grundsätzlich andere Lösungswege beschreiten als bei Optimierungsproblemen, die zur Klasse $\mathcal{P}$ gehören.

Kapitel 4

Das Verzweigungsprinzip

Algorithmen, die stets eine optimale Lösung in endlich vielen Schritten gewährleisten, nennt man *exakte* Verfahren. Zur Lösung von schwierigen diskreten Optimierungsproblemen werden für diese Zwecke häufig Algorithmen eingesetzt, die nach dem Verzweigungsprinzip arbeiten, aber im Allgemeinen einen exponentiellen Rechenaufwand erfordern. Die auch unter dem Namen *branch and bound* bekannte Methode soll in diesem Abschnitt für kombinatorische Optimierungsaufgaben beschrieben werden. Hier war der zulässige Bereich nichtleer und endlich und damit stets die Existenz wenigstens einer Optimallösung gesichert. Die Methode kann in modifizierter Form auch auf allgemeine gemischt diskrete Optimierungsprobleme übertragen werden.

Eine ausgewogene Verknüpfung von Separations- und Relaxationstechniken für die zu lösende Aufgabe oder deren Einbettung bildet die Grundlage des zu formulierenden Verzweigungsprinzips. Sukzessive werden Teilmengen der zulässigen Menge S einer kombinatorischen Optimierungsaufgabe P untersucht. Teilmengen, in denen keine optimale Lösung von P liegt, sollen dabei rechtzeitig ausgesondert werden. Dies kann alles mit Hilfe eines Suchbaumes veranschaulicht werden. Dabei bezieht sich das Wort *branch* auf das Verzweigen des Suchbaumes. Hier werden neue Teilmengen von S mittels Separationstechniken erzeugt. Der Begriff *bound* deutet auf die Verwendung unterer Schranken für die Zielfunktionswerte bezüglich der erzeugten Teilmengen hin, mit deren Hilfe man unbrauchbare Teilmengen von S mittels Kenntnis einer oberen Schranke für den Optimalwert von P eliminieren möchte. Zur Ermittlung unterer Schranken werden im allgemeinen Relaxationen zu den Teilaufgaben gelöst. Eine obere Schranke erhält man durch Kenntnis einer zulässigen Lösung aus der Menge S. Diese beschafft man sich oft durch Anwendung eines Näherungsverfahrens. Der Prozess von Verzweigen und Beschränken wird solange fortgesetzt, bis über ein geeignetes Optimalitätskriterium die Optimalität eines Elementes aus der Menge S erkannt wird.

Die Methode *branch and bound* ist eine sehr flexible Lösungstechnik, die für viele Probleme der diskreten Optimierung angepasst erscheint. Mit der Beschränkung auf die Untersuchung nur gewisser Teilmengen von S kann man das gesamte Verfahren auch unter dem Oberbegriff der *impliziten Enumeration* einordnen. Erste Anwendungen dieser Methodik wurden in den Arbeiten von Land und Doig zur ganzzahligen linearen Optimierung 1960 und von Little, Murty, Swenny und Karel zur Lösung des Rundreiseproblems 1963 dargestellt.

4.1 Formulierung und Begründung des Verzweigungsprinzips

Zur kombinatorischen Optimierungsaufgabe P

$$f(x^*) = \min_{x \in S} f(x)$$

mit endlicher und nichtleerer Menge S als zulässigen Bereich werden Teilaufgaben $P_{\nu l}$ der Stufe ν der Gestalt

$$f(x^{\nu l}) = \min_{x \in S_{\nu l}} f(x)\,, \quad \nu = 1, 2, \ldots\,, \quad l = 1, ..., l_\nu \tag{4.1}$$

betrachtet. Die Teilmengen $S_{\nu l}$ dieser partiellen Optimierungsaufgaben sollen nichtleer und endlich sein, um formal die Existenz einer Optimallösungen $x^{\nu l}$ zu sichern.

Im Verlauf des noch zu beschreibenden Verzweigungsprozess werden Teilaufgaben ν-ter Stufe durch solche höherer Stufe ersetzt. Dies entspricht dann konkret der Zerlegung einer der Teilmengen $S_{\nu l}$. Aber nicht jede der in (4.1) angegebenen Teilaufgaben soll künftig betrachtet und ausgewertet werden, um eine vollständige Enumeration zu vermeiden.

Definition 4.1
Solche durch Aufteilung von Mengen $S_{\nu l}$ bearbeiteten Teilaufgaben $P_{\nu l}$ und auch solche, von denen bekannt ist, dass ihre zulässige Menge keine Optimallösung von P enthält, werden als *inaktiv* bezeichnet.

Alle bereits erzeugten Teilaufgaben, die noch nicht verzweigt oder noch nicht näher hinsichtlich des Enthaltenseins einer Optimallösung von P untersucht wurden, werden als *aktiv* angesehen.

Für die formulierten Teilaufgaben werden jetzt grundlegende Forderungen postuliert, die für den Prozess des Verzweigens und Beschränkens notwendig und nützlich sind.

(F1) Es existiert genau eine Aufgabe P_{11} erster Stufe. Deren zugehörige Menge S_{11} ist nichtleer und endlich.

(F2) Wird die Aufgabe $P_{\mu t}$ durch Teilaufgaben $P_{\nu l}$, $\nu > \mu$, $l = 1, ..., l_\nu$, $l_\nu \geq 2$, ersetzt, so sind alle Mengen $S_{l\nu}$ nichtleer und echte Teilmengen von $S_{\mu t}$.

(F3) Die Vereinigung der Mengen $S_{\nu l}$ aller jeweils aktiven Teilaufgaben $P_{\nu l}$ enthält wenigstens eine Optimallösung von P.

(F4) Für die Teilaufgaben $P_{\nu l}$ seien untere Schranken $b_{\nu l}$ für den zugehörigen optimalen Zielfunktionswert bekannt. $S_{\nu l} \subset S_{\mu t}$ impliziert dabei $b_{\nu l} \geq b_{\mu t}$.

(F5) Ist $S_{\nu l} = \{x^{\nu l}\}$ und $x^{\nu l} \in S$, dann gilt $b_{\nu l} = f(x^{\nu l})$.

Die einzelnen Forderungen sollen im Folgenden näher erläutert und einige spezielle Realisierungen angesprochen werden.

- Die durch Forderung (F1) charakterisierte Aufgabe P_{11} kann eine Einbettung der kombinatorischen Optimierungsaufgabe P sein. Diese Ersatzaufgabe ist zu Beginn des Verzweigungsprozess die einzige aktive Teilaufgabe. Mit Forderung (F3) enthält sie eine Optimallösung $x^* \in S^*$ von P. Damit gilt $S^* \cap S_{11} \neq \emptyset$ und auch $S \cap S_{11} \neq \emptyset$.

  Speziell kann $S_{11} = S$ gewählt werden.

- Forderung (F2) besagt, dass nicht alle Elemente der Menge $S_{\mu t}$ in einer der Teilmengen $S_{\nu l}$ vorkommen müssen. Dies ist eine schwache Form der Seperation. Forderung (F3) verhindert aber, dass man alle Optimallösungen von P eliminiert.

  Oft strebt man eine Partitition der Aufgabe $P_{\mu t}$ an. Dann gilt sowohl $S_{\mu t} = \bigcup_{l=1}^{l_\nu} S_{\nu l}$ als auch $S_{\nu l} \cap S_{\nu k} = \emptyset\,,\ l, k = 1, ..., l_\nu\,,\ l \neq k$.

  Die zu erzeugenden Teilaufgaben erfüllen die Relation $f(x^{\mu t}) \leq f(x^{\nu l})\,,\ l = 1, ..., l_\nu$, und bei einer Partition zusätzlich $f(x^{\mu t}) = \min_{l=1,...,l_\nu} f(x^{\nu l})$.

- Die Lösung der partiellen Optimierungsprobleme $P_{\nu l}$ ist im Allgemeinen genau so schwierig, wie die Lösung des Ausgangsproblems. Für die Beschränkung des Verzweigungsprozesses sollen deshalb untere Schranken $b_{\nu l} \leq \min_{x \in S_{\nu l}} f(x)$ zu diesen Optimierungsproblemen herangezogen werden. Ihre Kenntnis wird mit der Forderung (F4) vorausgesetzt.

  Bei Einschränkung des zulässigen Bereichs einer Teilaufgabe $P_{\mu t}$ gegenüber einer Teilaufgabe $P_{\nu l}$, welche die Mengeninklusion $S_{\nu l} \subset S_{\mu t}$ beschreibt und die durch eventuell aufeinanderfolgende Verzweigungsoperationen erfolgen kann, sollen sich die zugehörigen unteren Schranken nicht verschlechtern, wie es durch die Ungleichung $b_{\nu l} \geq b_{\mu t}$ beschrieben wird.

- Die Forderung (F5) besagt, dass bei Einelementigkeit einer Menge $S_{\nu l}$ und gleichzeitiger Zulässigkeit von $x^{\nu l}$ die untere Schranke $b_{\nu l}$ den wahren Funktionswert von $x^{\nu l}$ repräsentiert. Das Problem $P_{\nu l}$ kann wegen Forderung (F2) nicht mehr verzweigt werden, bleibt aber aktiv, da die zulässige Lösung $x^{\nu l}$ einem noch zu formulierenden Optimalitätskriterium zur Verfügung stehen soll.

  Nur im Fall $S_{11} \subseteq S$ gilt stets $x^{\nu l} \in S$. Teilaufgaben mit einelementigen Mengen $S_{\nu l}$ mit $x^{\nu l} \notin S$ gehören zu den inaktiven Problemen, da sie trivialerweise keine Optimallösung von P enthalten.

- In Verbindung mit der Kenntnis der in Forderung (F4) angegebenen unteren Schranken folgt aus der Forderung (F3) auch, dass es stets eine aktive Teilaufgabe $P_{\mu t}$ mit $b_{\mu t} \leq f(x^*)$ geben muss, wobei x^* eine Optimallösung von P ist.

Die Beschränkung der Verzweigungsstruktur hängt ganz wesentlich von der Güte der in Forderung (F4) genannten unteren Schranken ab. Bei speziellen kombinatorischen Optimierungsaufgaben findet man oft aus der impliziten Beschreibung der zulässigen Menge S Ansatzpunkte, wie man derartige Schranken gewinnen kann.
An dieser Stelle soll die *Relaxationstechnik* genutzt werden. Man verzichtet also auf das exakte Lösen der partiellen Optimierungsaufgabe $P_{\nu l}$, sondern verwendet zur Bestimmung der unteren Schranke $b_{\nu l}$ eine *Relaxation* $\tilde{P}_{\nu l}$ der Gestalt

$$f(\tilde{x}^{l\nu}) = \min_{x \in \tilde{S}_{\nu l}} f(x)\,, \quad \nu = 1, 2, ...\,, \quad l = 1, ..., l_\nu\,, \tag{4.2}$$

mit $S_{\nu l} \subseteq \tilde{S}_{\nu l}$. Die Mengen $\tilde{S}_{\nu l}$ sind dabei so zu wählen, dass das Optimierungsproblem $\tilde{P}_{\nu l}$ leicht lösbar ist und die damit berechneten unteren Schranken $b_{\nu l} = f(\tilde{x}^{l\nu})$ möglichst gut sind. Bei jeder der zu konstruierenden Teilaufgaben $\tilde{P}_{\nu l}$ sollte die gleiche Relaxationstechnik angewendet werden, um die in Forderung (F4) erwähnte Monotonieeigenschaft der unteren Schranken bei Einschränkung der zulässigen Bereiche zu garantieren. Dies ist gewährleistet, wenn aus $S_{\nu l} \subset S_{\mu t}$ die Beziehung $\tilde{S}_{\nu l} \subseteq \tilde{S}_{\mu t}$ folgt.

Bemerkung 4.1
Erhält man beim Lösen der in (4.2) beschriebenen Relaxation $\tilde{P}_{\nu l}$ zur Bestimmung einer unteren Schranke $b_{\nu l}$ eine Optimallösung $\tilde{x}^{\nu l}$, die gleichzeitig zur Menge S gehört, dann gilt $b_{\nu l} = f(\tilde{x}^{l\nu}) = \min_{x \in S_{\nu l} \cap S} f(x)$. Die Teilaufgabe $P_{\nu l}$ enthält also höchstens noch solche zulässigen Lösungen aus der Menge S, deren Zielfunktionswerte nicht unter dem der zulässigen Lösung $\tilde{x}^{l\nu}$ liegen. Damit muss die Menge $P_{\nu l}$ nicht mehr verzweigt werden. Deshalb wird die Menge $S_{\nu l}$ formal zu einer Einermenge $S_{\nu l} = \{\tilde{x}^{\nu l}\}$ reduziert. Sie bleibt aktiv, da sie eine zulässige Lösung enthält, die noch mit einem Optimalitätskriterium getestet werden soll, steht aber einer Verzweigung nicht mehr zur Verfügung.

Mit Hilfe der unteren Schranken sollen die Teilaufgaben *ausgelotet* werden. Man möchte erkennen, ob eine Teilaufgabe $P_{\nu l}$ keine Optimallösung von P enthalten kann, um sie dann inaktiv zu setzen. Dazu ist es nötig, die unteren Schranken $b_{\nu l}$ der einzelnen Teilaufgaben mit einer oberen Schranke für den optimalen Zielfunktionswert der kombinatorischen Optimierungsaufgabe P zu vergleichen, da man den Optimalwert f^* nicht kennt. Eine derartige obere Schranke sei mit m_0 bezeichnet. Dann gilt $f(x^*) \leq m_0$, wenn x^* eine Optimallösung von P ist. Kennt man eine zulässige Lösung $\tilde{x} \in S$ als Referenzlösung, dann wählt man $m_0 = f(\tilde{x})$. Im Verlauf des Verzweigungsprozesses gewinnt man weitere zulässige Lösungen, wenn Teilaufgaben mit Einermengen entstehen oder der in Bemerkung 4.1 geschilderte Fall eintritt. Damit kann gegebenenfalls der Wert m_0 aktualisiert werden.

Generell wird eine aktive Teilaufgabe $P_{\nu l}$ inaktiv gesetzt, wenn die Ungleichung $b_{\nu l} > m_0$ gilt. Dann ist die Beziehung $f(x) > m_0$ für alle $x \in S_{\nu l}$, also auch für alle darin enthaltenen Lösungen aus S garantiert. Die Teilaufgabe $P_{\nu l}$ enthält damit keine Optimallösung von P und gilt damit als ausgelotet. Kennt man noch keine zulässige Lösung aus S und ist $f(x^*) < m_0$ gesichert, das heißt, man hat nur eine grobe obere Schranke m_0 bestimmt, dann kann auch für $b_{\nu l} \geq m_0$ die Teilaufgabe $P_{\nu l}$ inaktiv gesetzt werden.

Zur Formulierung eines Optimalitätskriteriums werden folgende Bezeichnungen benötigt:

$$q = \max\{\, \nu \,|\, P_{\nu l} \text{ ist aktiv } \} \qquad \Gamma_q = \{\, (\nu, l) \,|\, P_{\nu l} \text{ ist aktiv } \}\,.$$

Die Menge Γ_q enthält alle Indexpaare (ν, l), für die die Teilaufgaben $P_{\nu l}$ unmittelbar nach Erzeugung der Aufgaben P_{qr}, $r = 1, ..., r_q$, aktiv sind.

Satz 4.1
Gilt $b_{\lambda k} = \min_{(\nu,l) \in \Gamma_q} b_{\nu l}$ und ist ein $\tilde{x} \in S$ mit $f(\tilde{x}) = b_{\lambda k}$ bekannt, dann ist $\tilde{x}$ eine Optimallösung der kombinatorischen Optimierungsaufgabe P.

Die Chance, ein $\tilde{x} \in S$ als Optimallösung von P zu identifizieren, ist um so höher, je schärfer die bestimmten Schranken $b_{\nu l}$ sind.

Ein allgemeiner Algorithmus zur Lösung einer kombinatorischen Optimierungsaufgabe P besteht aus einer alternierenden Abfolge von Verzweigungsschritten, in denen neue Teilaufgaben generiert werden und Beschränkungsschritten, in denen entschieden wird, ob eine Teilaufgabe in den inaktiven Zustand versetzt werden kann. Das Optimalitätskriterium wird nach jeder Änderung der Menge der aktiven Teilaufgaben und bei Kenntnis neu gewonnener zulässiger Lösungen aus der Menge S erneut überprüft.

Struktur eines Algorithmus nach dem Verzweigungsprinzip

1. Ersetze die kombinatorische Optimierungsaufgabe P durch eine Ersatzaufgabe P_{11}. Dabei soll S_{11} endlich sein und mindestens eine Optimallösung x^* von P enthalten. Es sind eine untere Schranke b_{11} mit $b_{11} \leq \min_{x \in S_{11}} f(x)$ und eine obere Schranke m_0 mit $f(x^*) \leq m_0$ zu bestimmen.

2. Für die jeweils neuen Teilaufgaben $P_{\nu l}$ sind unter Beachtung von (F4) und (F5) untere Schranken $b_{\nu l}$ mit $b_{\nu l} \leq \min_{x \in S_{\nu l}} f(x)$ zu ermitteln. Werden in diesem Prozess zulässige Lösungen $\tilde{x}^{\nu l} \in S$ gefunden und gilt $f(\tilde{x}^{\nu l}) < m_0$, dann ist die obere Schranke m_0 durch $m_0 := f(\tilde{x}^{\nu l})$ zu aktualisieren.

3. Teilaufgaben $P_{\nu l}$ mit $b_{\nu l} > m_0$ werden inaktiv gesetzt. Die Menge der aktiven Teilaufgaben Γ_q ist entsprechend zu aktualisieren.

4. Erfüllt eine zulässige Lösung $\tilde{x} \in S$ die Bedingung $f(\tilde{x}) = \min_{(\nu,l) \in \Gamma_q} b_{\nu l}$, dann ist $\tilde{x}$ eine Optimallösung von P.

5. Eine der aktiven und noch verzweigbaren Teilaufgaben, etwa $P_{\mu t}$, wird in neue Teilaufgaben $P_{\nu l}$, $\nu > \mu$, $l = 1, ..., l_\nu$, $l_\nu \geq 2$, aufgespalten. Dabei sind die Forderungen (F2) und (F3) zu beachten.

   Das Verfahren ist beginnend mit Schritt 2 zu wiederholen.

Die Endlichkeit eines nach dem Verzweigungsprinzip konstruierten Algorithmus lässt sich leicht nachweisen. Die Analyse einer Teilaufgabe endet entweder mit der Erzeugung neuer Teilaufgaben, bei denen sich die Anzahl der Elemente verringert, oder mit der Beschränkung des Verzweigungsprozess an dieser Stelle durch Aussonderung. Da im Ausgangsproblem eine zulässige Menge mit nur endlich vielen Elementen vorlag, entstehen nach endlich vielen Schritten Teilaufgaben, deren zulässige Mengen sämtlich einelementig sind. Mit Forderung (F3) und (F5) ist das Optimalitätskriterium spätestens jetzt erfüllt.

Bemerkung 4.2
Das angegebene Lösungsprinzip wird oft in der folgenden modifizierten Variante verwendet. Die jeweils beste Zwischenlösung aus der Menge S wird nach Aktualisierung der oberen Schranke m_0 separat in einem Speicher Z^* abgelegt. Alle Probleme $P_{\nu l}$ mit einelementigen Mengen $S_{\nu l} = \{x^{\nu l}\}$ werden generell inaktiv gesetzt. Das geschieht durch den erweiterten Test $b_{\nu l} \geq m_0$. Das Lösungsverfahren bricht ab, wenn es keine aktiven Teilaufgaben mehr gibt. Der Speicher Z^* enthält dann eine Optimallösung von P.

Die Vorgehensweise eines Algorithmus zum Verzweigungsprinzip kann man durch einen Suchbaum veranschaulichen. Die Knoten des Baumes entsprechen den Teilaufgaben $P_{\nu l}$.

Die von einem Knoten ausgehenden Pfeile geben an, in welche Teilaufgaben die dem Knoten zugehörige Teilaufgabe aufgesplittet wird. Die unteren Schranken $b_{\nu l}$ werden als Information an die betreffenden Knoten geschrieben. Ein solcher Baum wird schrittweise aufgebaut. Das Problem P_{11} stellt die Wurzel des Baumes dar. Die Menge der jeweils aktiven Ersatzaufgaben wird durch die Endknoten des bisher erzeugten Baumes repräsentiert, für die noch nicht feststeht, ob die zugehörige Menge $S_{\nu l}$ keine Optimallösung von P enthält. Die restlichen Endknoten gehören zu den inaktiven Teilaufgaben. Die Abbildung 4.1 zeigt ein Beispiel eines Verzweigungsbaumes mit der erreichten Tiefe $q = 4$.

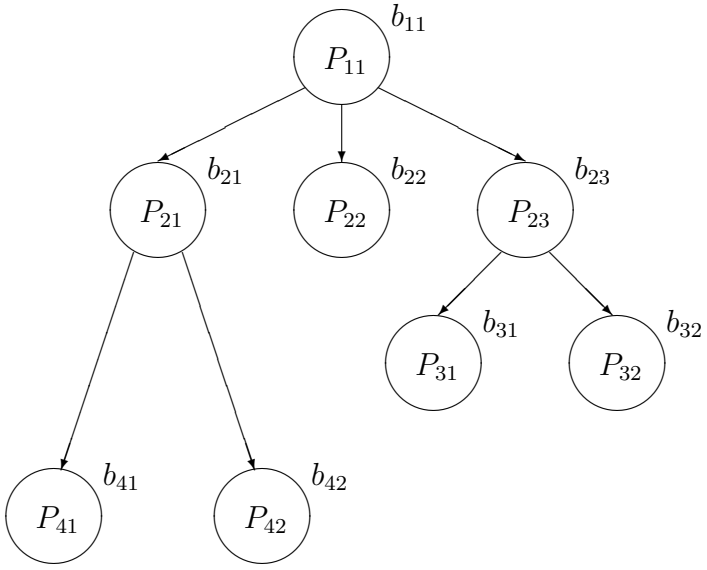


Abbildung 4.1: Beispiel eines Verzweigungsbaumes

Für Algorithmen zur Lösung konkreter kombinatorischer Optimierungsprobleme müssen grundsätzlich Regeln und Verfahren für die folgenden Operationen angegeben werden:

(a) Vorschrift zur Bildung der Ersatzaufgabe P_{11} und Verfahren zur Bestimmung einer oberen Schranke für den gesuchten Optimalwert von P.

(b) Vorschrift zur Auswahl einer zu verzweigenden Teilaufgabe $P_{\mu t}$.

(c) Vorschrift zur Konstruktion der die Teilaufgabe $P_{\mu t}$ ersetzenden Teilaufgaben $P_{\nu l}$.

(d) Verfahren zur Gewinnung unterer Schranken für die Teilaufgaben $P_{\nu l}$.

(e) Verfahren zur Feststellung, ob eine Teilaufgabe $P_{\nu l}$ eine Optimallösung von P enthält.

Mit einer geschickten Wahl der Ersatzaufgabe kann der Aufwand des gesamten Algorithmus reduziert werden. Allerdings muss man die Struktur der diskreten Menge S gut kennen, um eine einfach handhabbare endliche Menge S_{11} zu finden. Zur Erzeugung einer guten oberen Schranke werden vielfach Näherungsverfahren zur Erzeugung einer zulässigen Lösung der zu bearbeitenden kombinatorischen Optimierungsaufgabe vorgeschaltet. Diese sollen grundsätzlich in polynomialer Zeit arbeiten.

Die zu den Punkten (b) und (c) anzugebenden Vorschriften sind die Verzweigungsregeln (*branch*). Durch (b) wird festgelegt, wo jeweils verzweigt wird und (c) gibt die Art des Verzweigungsvorganges an. Bei der *Last In - First Out - Regel* (LIFO) wird die jeweils zuletzt erzeugte Teilaufgabe als nächstes weiterverarbeitet. Verzweigt man stets die aktive, noch verzweigbare Teilaufgabe mit der kleinsten unteren Schranke, dann spricht man von der *Minimum Lower Bound - Regel* (MLB). Die Art der Zerlegung einer Teilaufgabe hängt wesentlich von der Struktur der vorliegenden kombinatorischen Optinierungsaufgabe ab.

Durch die Vorschriften (d) und (e) soll der geschilderte Verzweigungsprozess beschränkt (*bound*) werden. Will man die benötigten unteren Schranken über eine Relaxation der zu untersuchenden Teilaufgabe finden, dann muss man Vorschriften zu ihrer Bildung angeben.

4.2 Anwendung des Verzweigungsprinzips

Mit Hilfe des im letzten Abschnitt formulierten Verzweigungsprinzips soll jetzt am Beispiel einer binären linearen Optimierungsaufgabe die Entwicklung eines exakten Algorithmus demonstriert werden. Ausgangspunkt sei konkret die folgende Aufgabe:

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &\rightarrow \min \\ \sum_{j=1}^{n} a_{ij} x_j &\geq b_i\,, \quad i = 1, ..., m \\ x_j &\in \{0;1\}\,, \quad j = 1, ..., n \end{aligned} \tag{4.3}$$

Dabei wird zur Vereinfachung der Diskussionen angenommen, dass die Daten der Aufgabe den Bedingungen $c_j > 0\,,\ j = 1, ..., n$, $b_i > 0\,,\ i = 1, ..., m$, $a_{ij} \geq 0\,,\ i = 1, ..., m\,,\ j = 1, ..., n$, $\sum_{i=1}^{m} a_{ij} > 0\,,\ j = 1, ..., n$, und $\sum_{j=1}^{n} a_{ij} \geq b_i\,,\ i = 1, ..., m$, genügen.
Dann stellt der Vektor $\tilde{x}$ mit $\tilde{x}_j = 1\,,\ j = 1, ..., n$, eine zulässige Lösung dar. Damit hat man eine obere Schranke $m_0 = \sum_{j=1}^{n} c_j$ generiert, die allerdings den maximalen Zielfunktionswert über dem zulässigen Bereich darstellt und demzufolge nicht besonders gut sein muss. Der gesuchte Optimalwert ist positiv, da der Nullvektor unzulässig ist und alle Zielfunktionskoeffizienten positiv sind.

Mit Hilfe der Grundmenge $G = \{\, x \in \mathbb{R}^n \mid \sum_{j=1}^{n} a_{ij} x_j \geq b_i\,,\ i = 1, ..., m \,\}$ kann der zulässige Bereich der Aufgabe (4.3) durch die nichtleere Menge $S = G \cap \{0, 1\}^n$ beschrieben werden. Diese diskrete Menge enthält höchstens $2^n - 1$ Elemente. Die zu minimierende Funktion f hat konkret die Gestalt $f(x) = c^\top x$.

In einem Verzweigungsprozess befindet sich jede binäre Variable in genau einem von drei Zuständen. Deshalb werden für die zu formulierenden Teilaufgaben $P_{\nu l}$ folgende Mengen vereinbart:

$R^0_{\nu l}$: Indizes von auf den Wert 0 fixierten Variablen

$R^1_{\nu l}$: Indizes von auf den Wert 1 fixierten Variablen

$R^{-1}_{\nu l}$: Indizes von freien Variablen

Sie bilden eine Partition der Indexmenge $N = \{1, ..., n\}$.

Die Mengen $S_{\nu l}$ für die Teilaufgaben $P_{\nu l}$ werden durch die Vorschrift

$$S_{\nu l} = \left\{ x \in G \,|\, x_j = 0,\, j \in R^0_{\nu l}\,,\;\; x_j = 1,\, j \in R^1_{\nu l}\,,\;\; x_j \in \{0;1\},\, j \in R^{-1}_{\nu l} \right\}$$

festgelegt.

Für die Ersatzaufgabe P_{11} sollen alle Variable noch frei wählbar sein. Damit setzt man für den Beginn des Verzweigungsprozess $R^0_{11} = R^1_{11} = \emptyset$ und $R^{-1}_{11} = N$. Es gilt $S_{11} = S$ und die Aufgabe P_{11} ist mit der Ausgangsaufgabe (4.3) identisch.

Die Teilaufgaben $P_{\nu l}$ sollen vorab etwas näher untersucht werden. Auf Grund der an die Daten gestellten Voraussetzungen kann die Lösbarkeit sehr einfach entschieden werden. Die Menge $S_{\nu l}$ ist genau dann nichtleer, wenn die Ungleichungen

$$\sum_{j \in R^{-1}_{\nu l}} a_{ij} + \sum_{j \in R^1_{\nu l}} a_{ij} \geq b_i\,,\;\; i = 1, ..., m$$

erfüllt sind. Gilt zusätzlich $R^{-1}_{l\nu} = \emptyset$, dann ist $S_{\nu l}$ eine Einermenge. Sie enthält den vollständig fixierten Vektor $x^{\nu l} \in S$ und es gilt $f(x^{\nu l}) = \sum_{j \in R^1_{l\nu}} c_j$.

Es sei $S_{\nu l}$ eine nichtleere Menge, bei der noch nicht alle Variablen fixiert wurden. Damit wird $R^{-1}_{l\nu} \neq \emptyset$ vorausgesetzt. Um zu entscheiden, ob die Menge $S_{\nu l}$ einelementig sein kann, werden die noch frei wählbaren Variablen einem *Fixierungstest* unterworfen.

Gibt es für den Index $j_1 \in R^{-1}_{\nu l}$ ein $k \in \{1, ..., m\}$ mit

$$\sum_{j \in R^{-1}_{\nu l} \setminus \{j_1\}} a_{kj} \;+\; \sum_{j \in R^1_{\nu l}} a_{kj} < b_k\,,$$

dann kann die Variable x_{j_1} auf den Wert Eins fixiert werden. Das entspricht den Operationen $R^{-1}_{\nu l} := R^{-1}_{\nu l} \setminus \{j_1\}$ und $R^1_{\nu l} := R^1_{\nu l} \cup \{j_1\}$.

Gelingt es auf diese Weise alle noch frei wählbaren Variable zu fixieren, dann enthält $S_{\nu l}$ nur eine zulässige Lösung und muss künftig keiner Verzweigung unterworfen werden. Anderenfalls kann man bei Bedarf tatsächlich noch nichtleere Teilmengen von $S_{\nu l}$ bilden.

Als nächstes sollen untere Schranken $b_{\nu l}$ berechnet werden. Eine sehr einfache Möglichkeit resultiert aus der Voraussetzung, dass alle Zielfunktionskoeffizienten positiv sind. Deshalb kann $b_{\nu l} = \sum_{j \in R^1_{l\nu}} c_j$ als untere Schranke gewählt werden. Für eine einelementige Menge wäre dies auch der korrekte Zielfunktionswert.

Bessere Schranken erhält man bei Verwendung einer *LP-Relaxation* zur Teilaufgabe $P_{\nu l}$. Bei Teilaufgaben, deren zugehörigen Mengen $S_{\nu l}$ nicht nur aus einem Element besteht, ist $R^{-1}_{\nu l} \neq \emptyset$ gesichert. Die Menge $S_{\nu l}$ wird dann durch die Obermenge

$$\tilde{S}_{\nu l} = \left\{ x \in G \,|\, x_j = 0,\, j \in R^0_{\nu l}\,,\;\; x_j = 1,\, j \in R^1_{\nu l}\,,\;\; 0 \leq x_j \leq 1,\, j \in R^{-1}_{\nu l} \right\}$$

ersetzt. Die freien Variablen können jetzt auch gebrochene Werte zwischen Null und Eins annehmen. Anschließend wird die in (4.2) beschriebene Relaxation $\tilde{P}_{\nu l}$ gelöst. Konkret liegt hier eine lineare Optimierungsaufgabe mit zum Teil fixierten Variablen vor. Da die betrachtete Menge $\tilde{S}_{\nu l}$ nichtleer und kompakt ist, besitzt die Aufgabe $\tilde{P}_{\nu l}$ stets eine Optimallösung $\tilde{x}^{\nu l}$. Die benötigte Schranke $b_{\nu l}$ wird dann durch $b_{\nu l} = \sum_{j=1}^{n} c_j \tilde{x}^{\nu l}_j$ festgelegt.

Ist $\tilde{x}^{\nu l}$ ein 0-1-Vektor, das heißt es wurde eine zulässige Lösung der Aufgabe (4.3) erzeugt, dann muss das Problem $P_{\nu l}$ nicht mehr verzweigt werden. Im Falle $b_{\nu l} < m_0$ hat man die bisher beste zulässige Lösung gefunden. Sie wird durch die Anweisung $\tilde{x} := \tilde{x}^{\nu l}$ gespeichert und der zugehörige Zielfunktionswert durch $m_0 := b_{\nu l}$ übertragen.
Mit der aktuell besten Lösung $\tilde{x}$ wird über die Menge aller aktiven Teilprobleme stets der Optimalitätstest "Gilt $f(\tilde{x}) = \min_{(\nu,l)\in\Gamma_q} b_{\nu l}$?" ausgeführt.

Im Laufe des Verzweigungsprozess sei bereits ein Teil des Verzweigungsbaumes entstanden. Jetzt ist festzulegen, welche aktive und noch verzweigbare Teilaufgabe zur weiteren Bearbeitung ausgewählt werden soll. Unter all diesen Teilaufgaben soll diejenige Aufgabe $P_{\mu t}$ mit der kleinsten unteren Schranke $b_{\mu t}$ gewählt werden. Damit wird im Verlauf des Verzweigungsprozess die kleinste aller unteren Schranken stets angehoben, um auf ein schnelles Ansprechen des Optimalitätskriteriums zu hoffen.
Da $P_{\mu t}$ ein noch verzweigbares Problem ist, war die über die LP-Relaxation erzeugte Lösung $\tilde{x}^{\mu t}$ keine zulässige Lösung aus S. Damit existiert wenigstens ein $j_0 \in R^{-1}_{\mu t}$ mit $\tilde{x}^{\mu t}_{j_0} \in (0,1)$. Gibt es mehrere solcher Indizes, dann wähle man einen Index j_0 aus, für den der Abstand $|\frac{1}{2} - \tilde{x}^{\mu t}_{j_0}|$ minimal ausfällt.
Die Teilaufgabe $P_{\mu t}$ wird in genau zwei Teilaufgaben $P_{\nu 1}$ und $P_{\nu 2}$, $\nu > \mu$, zerlegt. Die neuen Teilmengen entstehen durch die folgenden Vorschriften:

$$S_{\nu 1} = \{x \in S_{\mu t} \,|\, x_{j_0} = 0\} \qquad\qquad S_{\nu 2} = \{x \in S_{\mu t} \,|\, x_{j_0} = 1\}$$

Entsprechend müssen die Indexmengen generiert werden:

$$\begin{array}{lllll} P_{\nu 1}: & R^0_{\nu 1} := R^0_{\mu t} \cup \{j_0\}\,, & R^1_{\nu 1} := R^1_{\mu t}\,, & R^{-1}_{\nu 1} := R^{-1}_{\mu t} \setminus \{j_0\} \\ P_{\nu 2}: & R^0_{\nu 2} := R^0_{\mu t}\,, & R^1_{\nu 2} := R^1_{\mu t} \cup \{j_0\}\,, & R^{-1}_{\nu 2} := R^{-1}_{\mu t} \setminus \{j_0\} \end{array}$$

Die Teilaufgabe $P_{\nu 1}$ wird anschließend sofort dem oben beschriebenen Fixierungstest unterworfen, da er wegen der Festlegung von $x_{j_0} = 0$ zu neuen Fixierungen führen kann. Für die Teilaufgabe $P_{\nu 2}$ muss der Fixierungstest nicht angewendet werden, da er gegenüber der bereits bearbeiteten Teilaufgabe $P_{\mu t}$ zu keiner neuen Fixierung führen kann.

Beispiel 4.1
Gegeben sei die binäre lineare Optimierungsaufgabe

$$\begin{array}{rcrcrcrcrcrcl} z & = & 8x_1 & + & 5x_2 & + & 8x_3 & + & 3x_4 & + & 6x_5 & \to & \min \\ & & 3x_1 & + & 4x_2 & + & 2x_3 & + & 4x_4 & + & 3x_5 & \geq & 10 \\ & & 2x_1 & + & x_2 & + & 5x_3 & + & 2x_4 & + & 4x_5 & \geq & 8 \\ & & & & & & & & & & x_j & \in & \{0,1\}\,, \quad j = 1,...,5 \end{array}$$

Mit der zulässigen Lösung $\tilde{x}_j = 1\,,\ j = 1,...,5$ erhält man die obere Schranke $m_0 = 30$.
Aufgabe P_{11}: Setze $R^0_{11} = R^1_{11} = \emptyset$ und $R^{-1}_{11} = \{1,2,3,4,5\}$. Die Diskretheitsbedingungen $x_j \in \{0,1\}\,,\ j = 1,...,5$, werden durch $0 \leq x_j \leq 1\,,\ j = 1,...,5$, ersetzt und die zugehörige lineare Optimierungsaufgabe gelöst. Als Optimallösung erhält man $x_1 = 0$, $x_2 = 0.61$, $x_3 = 0.28$, $x_4 = 1$, $x_5 = 1$, $z = 14.28$. Da die vorliegende Aufgabe die einzig aktive Aufgabe mit der unteren Schranke $b_{11} = 14.28$ ist, wird sie sofort in die beiden Teilaufgaben P_{21} und P_{22} verzweigt. Dazu wird die Variable x_2 ausgewählt.

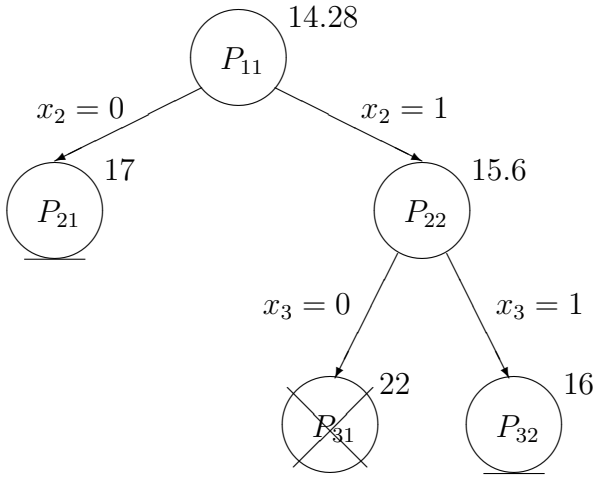
Aufgabe P_{21}: Mit der Verzweigung $x_2 = 0$ erhält man die Mengen $R^0_{21} = \{2\}$, $R^1_{21} = \emptyset$ und $R^{-1}_{21} = \{1, 3, 4, 5\}$. Der anzuwendende Fixierungstest setzt über die erste Restriktion die Variablen x_1, x_4 und x_5 auf den Wert Eins. Damit ändern sich die zwei Indexmengen $R^1_{21} = \{1, 4, 5\}$ und $R^{-1}_{21} = \{3\}$. Als Optimallösung der entsprechenden Relaxation erhält man $x_1 = 1$, $x_2 = 0$, $x_3 = 0$, $x_4 = 1$, $x_5 = 1$, $z = 17$. Damit liegt eine zulässige Lösung vor. P_{21} muss nicht mehr verzweigt werden, da für alle weiteren zulässigen Lösungen aus der Menge S_{21} nur $z \geq 17$ gelten kann. Da $b_{21} = 17 < 30 = m_0$ gilt, hat man die bisher beste zulässige Lösung gefunden. Sie wird auf den Vektor $\tilde{x} := (1\,0\,0\,1\,1)^\top$ gespeichert und die obere Schranke $m_0 := 17$ aktualisiert.

Aufgabe P_{22}: Mit der Verzweigung $x_2 = 1$ lauten die Indexmengen $R^0_{22} = \emptyset$, $R^1_{22} = \{2\}$ und $R^{-1}_{22} = \{1, 3, 4, 5\}$. Die Relaxation liefert die Optimallösung $x_1 = 0$, $x_2 = 1$, $x_3 = 0.2$, $x_4 = 1$, $x_5 = 1$, $z = 15.6$. Sie ist nicht zulässig bezüglich der Menge S_{22}. Die untere Schranke $b_{22} = 15.6$ liegt unter der oberen Schranke $m_0 = 17$. Da P_{22} das einzige im Moment verzweigbare Problem ist, werden die Variable x_3 ausgewählt und die Teilaufgaben P_{31} und P_{32} erzeugt.

Aufgabe P_{31}: Die Indexmengen lauten $R^0_{31} = \{3\}$, $R^1_{31} = \{2\}$ und $R^{-1}_{31} = \{1, 4, 5\}$. Mit Hilfe des Fixierungstest werden durch die zweite Restriktion der Reihe nach die Variablen x_1, x_4 und x_5 auf den Wert Eins gesetzt. Damit ändern sich die beiden Indexmengen $R^1_{31} = \{1, 2, 4, 5\}$ und $R^{-1}_{31} = \emptyset$. Somit stellt S_{31} eine einelementige Menge dar. Der Zielfunktionswert der fixierten Lösung beträgt $z = 22$. Wegen $b_{31} = 22 > 17 = m_0$ wird P_{31} in den Zusand inaktiv versetzt.

Aufgabe P_{32}: Nach der Festlegung $x_3 = 1$ erhält man die Mengen $R^0_{32} = \emptyset$, $R^1_{31} = \{2, 3\}$ und $R^{-1}_{32} = \{1, 4, 5\}$ Die Relaxation liefert die Optimallösung $x_1 = 0$, $x_2 = 1$, $x_3 = 1$, $x_4 = 1$, $x_5 = 0$, $z = 16$. Damit liegt eine zulässige Lösung vor. Wegen $b_{32} = 16 < 17 = m_0$ ist sie die bisher beste zulässige Lösung. Sie wird auf den Vektor $\tilde{x} := (0\,1\,1\,1\,0)^\top$ gespeichert und die obere Schranke $m_0 := 16$ aktualisiert.

Jetzt spricht das Optimalitätskriterium an. Zur kleinsten unteren Schranken $b_{32} = 16$ aller aktiven Teilaugaben gibt es eine zulässige Lösung $\tilde{x}$ mit $f(\tilde{x}) = 16$. Damit hat man eine Optimallösung $x^*_1 = 0$, $x^*_2 = 1$, $x^*_3 = 1$, $x^*_4 = 1$, $x^*_5 = 0$, $z_{\min} = 16$ gefunden.



Kapitel 5

Das Schnittprinzip

Auf dem Schnittprinzip beruhende Algorithmen wurden schon sehr früh zur Lösung von ganzzahligen linearen Optimierungsaufgaben eingesetzt. Erste Anwendungen findet man bereits 1958 bei Gomory. Numerische Schwierigkeiten haben die Anwendungen des Schnittprinzips etwas eingeschränkt. Derzeit sorgen aber ausgefeilte Schnittmethoden für spezielle diskrete Optimierungsaufgaben wieder für eine Belebung.

Eine allgemeine Formulierung und Begründung des Schnittprinzips erfolgte 1971 durch Emeličev. In diesem Abschnitt wird das allgemeine Schnittprinzip und eine einfache Realisierung für kombinatorische Optimierungsaufgaben vorgestellt. Eine ausführliche Darstellung findet man zum Beispiel in [Scho76]. Die Übertragung des Schnittprinzips auf die allgemeine diskrete Optimierungsaufgabe ist mit geringfügigen Änderungen problemlos möglich. In der Literatur findet man auch Schnittebenenverfahren für konvexe nichtlineare Optimierungsaufgaben.

Ausgangspunkt des Schnittprinzips ist die Relaxation einer zu lösenden kombinatorischen Optimierungsaufgabe P. Die zulässige Menge S von P wird also durch eine sie umfassende Menge, die nicht diskret sein muss, ersetzt. Diese Obermenge wird bis zum Erkennen einer Optimallösung sukszessive durch Schnittbedingungen reduziert. Die Schnittbedingungen werden dabei jeweils über eine Optimallösung einer gerade gelösten reduzierten Aufgabe generiert.

5.1 Formulierung und Begründung des Schnittprinzips

Zur kombinatorischen Optimierungsaufgabe P

$$f(x^*) = \min_{x \in S} f(x)$$

mit nichtleerer und endlicher Menge S als zulässigem Bereich wird eine rekursiv zu definierende Folge von Aufgaben P_k der Gestalt

$$g(x^k) = \min_{x \in R_k} g(x)\,, \quad k = 1, 2, ... \tag{5.1}$$

betrachtet. Die Aufgaben P_k sollen generell so konstruiert sein, dass sie stets eine Optimallösung x^k besitzen.

Für ein festes $k \geq 1$ werde jetzt die Aufgabe P_k betrachtet. Es sei x^k eine Optimallösung der Aufgabe P_k. Zu diesem Zeitpunkt kenne man noch keine Optimallösung x^* der kombinatorischen Optimierungsaufgabe P.

Mit Hilfe dieser Optimallösung x^k wird eine *Schnittmenge* $R(x^k)$ mit noch zu formulierenden Eigenschaften erzeugt. Die Aufgabe P_{k+1} entsteht dann durch die Mengendifferenz

$$R_{k+1} = R_k \setminus R(x^k)\,.$$

Der zulässige Bereich R_k der Aufgabe P_k wird durch die Schnittmenge $R(x^k)$ weiter eingeschränkt. Die Lösbarkeit der so entstandenen neuen Aufgabe P_{k+1} hängt also wesentlich von der Beschaffenheit der Schnittmenge $R(x^k)$ ab.

Für die rekursiv formulierten Optimierungsaufgaben P_k werden jetzt die grundlegenden Forderungen angegeben, die für den Schnittprozess notwendig sind und auch zur Beschreibung eines Optimalitätskriteriums führen.

(V1) Die Aufgabe P_1 ist eine Relaxation der Aufgabe P.

(V2) Für die Schnittmenge $R(x^k)$ zur Aufgabe P_k gilt $x^k \in R(x^k)$ und $R(x^k) \subset R_k\,.$

(V3) Wenn $x^k \notin S$ ist, dann soll $R(x^k) \cap S = \emptyset$ gelten.
Wenn $x^k \in S$ ist, dann soll $f(x^k) = \min\limits_{x \in R(x^k) \cap S} f(x)$ gelten.

Die erste Forderung beschreibt die Aufgabe, mit der der Schnittprozess starten soll. Die letzten beiden Forderungen stellen die eigentlichen Schnittbedingungen dar. Im Folgenden werden die einzelnen Forderungen erläutert und eine spezielle Realisierung behandelt.

- Forderung (V1) verlangt, dass die diskrete Menge S mittels $R_1 \supset S$ durch eine sie umfassende Menge R_1 ersetzt wird. Es wird nicht gefordert, dass die Menge R_1 diskret ist. Die ursprünglich zu minimierende Funktion f wird durch eine Minorantenfunktion g ersetzt. Damit gilt $g(x) \leq f(x)$ für $x \in S$.

  Eine spezielle Realisierung der Forderung (V1) liegt vor, wenn man als Funktion g speziell die Funktion f wählt.

- (V2) verlangt, dass die Schnittmenge $R(x^k)$ konkret eine erzeugte Optimallösung x^k der Aufgabe P_k enthält. Weiter soll die Schnittmenge $R(x^k)$ eine echte Teilmenge des zulässigen Bereichs R_k der Aufgabe P_k sein. Die Menge R_k wird also verkleinert, das heißt, von ihr wird etwas weggeschnitten, aber nicht alles. Damit gilt $R_{k+1} \neq \emptyset$.

  Generell ist bei Generierung der Schnittmenge $R(x^k)$ darauf zu achten, dass die zu bildende Differenzmenge $R_{k+1} = R_k \setminus R(x^k)$ die Existenz einer Optimallösung der neuen Optimierungsaufgabe P_{k+1} zulässt.

- Die Forderung (V3) regelt, was neben der Optimallösung x^k der Aufgabe P_k noch alles von P_k weggeschnitten werden kann. Generell sind beliebige Elemente aus der Menge $R_k \setminus S$ zugelassen. Aber es gibt Einschränkungen für das Wegschneiden von zulässigen Lösungen der Menge S.

Die erste Bedingung in (V3) verlangt, dass kein Element aus S zur Schnittmenge gehört, wenn die erzeugte Optimallösung x^k der Aufgabe P_k selbst nicht zu S gehört.

Sollte allerdings x^k zu Menge S gehören, dann gibt die zweite Bedingung in (V3) an, welche weiteren zulässigen Lösungen aus S zur Schnittmenge gehören dürfen. Erlaubt sind all diejenigen zulässigen Lösungen $x \in S$, deren Zielfunktionswert $f(x)$ den Wert $f(x^k)$ nicht unterschreitet. Damit weiß man, dass von den wegzuschneidenden zulässigen Lösungen nur die zulässige Lösung $x^k \in S$ als Kandidat für eine Optimallösung der Ausgangsaufgabe P zu notieren ist.

Mit Hilfe der schrittweise erzeugten Optimallösungen x^k zu den Aufgaben P_k, die zusätzlich auch zur Menge S gehören, soll jetzt ein Optimalitätskriterium angegeben werden. Dazu werden die Mengen

$$S_q = \{x^1, x^2, ..., x^q\} \cap S\,, \quad q = 1, 2, ...\ ,$$

gebildet. Zwei derartig aufeinanderfolgende Mengen unterscheiden sich um maximal ein Element.

Satz 5.1
Gilt $S_q \neq \emptyset$ und $f(x^t) = \min_{x \in S_q} f(x) \leq g(x^q)$, dann ist x^t eine Optimallösung der kombinatorischen Optimierungsaufgabe P.

Ist die Menge S_q nichtleer, dann stellt x^t die beste zulässige Lösung aus der Menge der erzeugten Optimallösungen der Aufgaben P_k, $k = 1, ...,$, dar, die auch zulässig waren. Andere im Verlauf des Schnittprozess entfernte zulässige Lösungen besitzen wegen (V3) keinen besseren Zielfunktionswert als $f(x^t)$. Die noch verbliebenen zulässigen Lösungen aus S in der Menge R_q haben wegen der Relaxationseigenschaft keinen besseren Zielfunktionswert als $g(x^q)$ und wegen der im Satz 5.1 geforderten Ungleichung auch keinen besseren Wert als $f(x^t)$.
Mit dem formulierten Optimalitätskriterium erkennt man die Optimalität der zulässigen Lösung x^t erst nach dem Lösen der Aufgabe P_q. Will man die Optimalität einer soeben berechneten Optimallösung der aktuellen Aufgabe, die zugleich zulässig für P ist, sofort erkennen, müssen die Voraussetzungen des Satzes 5.1 wesentlich verschärft werden.

Satz 5.2
Gilt $x^k \notin S$, $k = 1, ..., q-1$, $x^q \in S$ und $f(x^q) \leq g(x^q)$, dann ist x^q eine Optimallösung der kombinatorischen Optimierungsaufgabe P.

Unter den gegebenen Voraussetzungen gilt $S_k = \emptyset$, $k = 1, ..., q-1$, und $S_q = \{x^q\}$. Die Erfüllung der Ungleichung $f(x^q) \leq g(x^q)$ zeigt sofort die Optimalität der zulässigen Lösung $x^q \in S$ an.

Bemerkung 5.1
Für den wichtigen Spezialfall $g(x) = f(x)$, $x \in S$, sind die verschärften Voraussetzungen auf natürliche Weise erfüllt. Damit ist die erste zulässige Lösung unter den erzeugten Optimallösungen der sukzessive bearbeiteten Aufgaben P_k zugleich optimal für die Ausgangsaufgabe P. In diesem Fall wird von der *einfachen Realisierung* des Schnittprinzips gesprochen. Man benötigt dann auch nicht den zweiten Teil der in (V3) beschriebenen Voraussetzungen.

Mit den bisher beschriebenen Forderungen kann die Endlichkeit eines zu formulierenden Verfahrens noch nicht garantiert werden. Deshalb wird formal noch die folgende Voraussetzung postuliert:

(V4) Es gibt eine Aufgabe P_r, $r \geq 1$, mit $S_r \neq \emptyset$ und $\min\limits_{x \in S_r} f(x) \leq g(x^r)$.

Die Voraussetzung (V4) verlangt, dass eine Aufgabe P_r existiert, nach deren Lösung die Bedingungen für das im Satz 5.1 angegebene Optimalitätskriterium erfüllt sind. Das ist natürlich keine für die Konstruktion von Algorithmen aussagekräftige Forderung. Die Endlichkeit hängt wesentlich von der gewählten Obermenge R_1 und der Güte der erzeugten Schnittmengen ab.

Nach Formulierung der grundlegenden Voraussetzungen und der gegebenen Beschreibung eines allgemeinen Optimalitätskriteriums kann jetzt der Ablauf eines allgemeinen, auf dem Schnittprinzip basierenden Algorithmus zur Lösung einer kombinatorischen Optimierungsaufgabe P angegeben werden. Er besteht aus einer sequentiellen Folge von zu lösenden Optimierungsaufgaben, deren zulässige Bereiche durch Schnittbedingungen sich ständig verkleinern. Ab der ersten generierten zulässigen Lösung von P muss in jedem Schritt das Optimalitätskriterium geprüft werden.

Struktur eines Algorithmus nach dem Schnittprinzip

1. Wähle für die kombinatorische Optimierungsaufgabe P eine lösbare Relaxation P_1. Dazu muss eine Menge $R_1 \supseteq S$ und eine Minorantenfunktion g angegeben werden. Setze $q := 1$.

2. Für die Optimierungsaufgabe P_q ist eine Optimallösung x^q zu bestimmen.

3. Gilt $S_q = \{x^1, ..., x^q\} \cap S \neq \emptyset$ und $f(x^t) = \min_{x \in S_q} f(x) \leq g(x^q)$, dann ist x^t eine Optimallösung von P.

4. Bilde die Schnittmenge $R(x^q)$ so, dass die durch den reduzierten zulässigen Bereich $R_{q+1} = R_q \setminus R(x^q)$ neu entstehende Aufgabe P_{q+1} lösbar ist und die Forderungen (V2) und (V3) erfüllt werden.

   Setze $q := q + 1$ und gehe zu Schritt 2.

Bei der einfachen Realisierung des Schnittprinzips sind mit der speziellen Wahl von f als Minorantenfunktion g sukzessive Aufgaben P_k der Gestalt $f(x^k) = \min_{x \in R_k} f(x)$ zu lösen. Der dritte Schritt im Algorithmus kann durch den einfachen Test "$x^q \in S$?" ersetzt werden. Wird diese Frage mit ja beantwortet, dann ist x^q eine Optimallösung von P.
Die Festlegung der Menge R_1, welche die diskrete Menge S umfassen soll, hängt auch hier von der impliziten Beschreibung der diskreten Menge S ab. Man wird im Allgemeinen die schwierigen Teile dieser Beschreibung ignorieren, um eine leicht lösbare Relaxation zu gewinnen.
Die formale Beschreibung der Schnittmengen $R(x^k)$ lässt keine weiteren konkreten Aussagen zu. Deshalb hängt ihre Bildung von der konkreten Optimierungsaufgabe ab. Im nächsten Abschnitt soll genau dieser Aspekt berücksichtigt werden.

5.2 Anwendung des Schnittprinzips

Zur Demonstration des Schnittprinzips wird die rein ganzzahlige lineare Optimierungsaufgabe

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &\rightarrow \max \\ \sum_{j=1}^{n} a_{ij} x_j &= b_i\,, \quad i = 1, ..., m \\ x_j &\in \mathbb{N}_0\,, \quad j = 1, ..., n \end{aligned}$$

ausgewählt, bei der die Restriktionen in Gleichungsform vorliegen. Grundsätzlich seien alle Koeffizienten rational. Mit $\mathbb{N}_0$ wird die Menge der nichtnegativen ganzen Zahlen bezeichnet. Die formale Transformation auf eine zu minimierende Zielfunktion ist für die Anwendung des Schnittprinzips nicht zwingend notwendig.

Für die diskrete Menge $S = \{x \in \mathbb{R}^n \,|\, Ax = b\,,\ x_j \in \mathbb{N}_0\,,\ j = 1, ..., n\}$ kann man ohne weitere Voraussetzungen nicht zeigen, dass sie nichtleer und endlich ist. Damit liegt im Allgemeinen keine kombinatorische Optimierungsaufgabe vor. Die daraus resultierenden notwendigen Ergänzungen für das für kombinatorische Optimierungsaufgaben formulierte Schnittprinzip sind aber nur technischer Natur und werden hier mit angesprochen.

Zum Einsatz kommt die einfache Realisierung des Schnittprinzips, womit auch $f(x) = c^\top x$ als Zielfunktion für die Relaxation verwendet wird. Für die Erweiterung des zulässigen Bereichs S wird für jede Variable die Ganzzahligkeitsforderung durch die Nichtnegativitätsbedingung ersetzt. Damit erhält man $R_1 = \{x \in \mathbb{R}^n \,|\, Ax = b\,,\ x_j \geq 0\,,\ j = 1, ..., n\}$. Die Aufgabe P_1 entspricht dann der folgenden LP-Relaxation:

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &\rightarrow \max \\ \sum_{j=1}^{n} a_{ij} x_j &= b_i\,, \quad i = 1, ..., m \\ x_j &\geq 0\,, \quad j = 1, ..., n \end{aligned}$$

Diese LP-Relaxation wird mit Hilfe der Simplexmethode gelöst. Im Falle der Unlösbarkeit von P_1 gilt entweder $R_1 = \emptyset$ oder für $R_1 \neq \emptyset$ wächst die Zielfunktion über R_1 unbeschränkt. Im ersten Fall gilt sicher $S = \emptyset$. Für den zweiten Fall kann auch $S = \emptyset$ gelten. Da die Rationalität der Daten vorausgesetzt wurde, kann im Fall $S \neq \emptyset$ nur noch der Fall eintreten, dass die Zielfunktion über der dann nicht endlichen diskreten Menge S unbeschränkt wächst. In beiden Fällen ist damit auch P unlösbar. Im Falle der Lösbarkeit der LP-Relaxation P_1 liefert die Simplexmethode eine optimale Basislösung x^1. Gilt $x^1 \in S$, das heißt alle Komponenten von x^1 sind ganzzahlig, dann ist x^1 auch eine Optimallösung der rein ganzzahligen linearen Optimierungsaufgabe P. Damit ist nur der Fall $x^1 \notin S$ interessant.

Für die weiteren Betrachtungen sei x^1 eine optimale Basislösung von P_1, für die mindestens eine Komponente einen nichtganzzahligen Wert annimmt. Die zu dieser Basislösung x^1 gehörigen Nichtbasisvariable seien durch folgende Indexmenge erfasst:

$$\mathcal{N} = \{j \in \{1, ..., n\} \,|\, x_j \text{ ist Nichtbasisvariable in der Basislösung } x^1\}$$

Die s-te Gleichung des zu x^1 gehörigen Simplextableaus habe die Gestalt

$$x_{B_s} + \sum_{j \in \mathcal{N}} y_{sj} x_j = y_{s0} \tag{5.2}$$

Für die Basislösung x^1 gilt $x^1_j = 0\,,\ j \in \mathcal{N}$, und $x^1_{B_s} = y_{s0}\,$. Der Wert y_{s0} dieser ausgewählten Basisvariable x_{B_s} sei nicht ganzzahlig.
Die Koeffizienten der Gleichung (5.2) werden wie folgt zerlegt:

$$\begin{aligned} y_{s0} &= u_{s0} + v_{s0} \quad \text{mit} \quad u_{s0} = \lfloor y_{s0} \rfloor \\ y_{sj} &= u_{sj} + v_{sj} \quad \text{mit} \quad u_{sj} = \lfloor y_{sj} \rfloor\,, \quad j \in \mathcal{N} \end{aligned}$$

Dabei ist $\lfloor z \rfloor$ die größte ganze Zahl, die nicht größer als z ist. Folglich ist $z - \lfloor z \rfloor$ der nichtnegative Rest.
Für die vorliegende Zerlegung gilt konkret $0 < v_{s0} < 1$ und $0 \le v_{sj} < 1\,,\ j \in \mathcal{N}\,.$
Nach Einsetzen der zerlegten Größen in (5.3) erhält man nach einer weiteren Umformung die folgende Gleichung:

$$x_{B_s} + \sum_{j \in \mathcal{N}} u_{sj} x_j \ - u_{s0} = v_{s0} - \sum_{j \in \mathcal{N}} v_{sj} x_j \tag{5.3}$$

Für jede ganzzahlige Lösung x ist die linke Seite von (5.3) ganzzahlig. Folglich muß dann auch die rechte Seite von (5.3) ganzzahlig sein.
Für diese rechte Seite gilt aber wegen $x_j \ge 0\,,\ j \in \mathcal{N}$, $v_{sj} \ge 0\,,\ j \in \mathcal{N}$, und $v_{s0} < 1$ stets

$$v_{s0} - \sum_{j \in \mathcal{N}} v_{sj} x_j < 1\ .$$

Folglich kann bei Vorliegen einer zulässigen ganzzahligen Lösung x durch die rechte Seite von (5.3) keine positive ganze Zahl dargestellt werden. Unter diesem Aspekt kann man die Erfüllung der Restriktion

$$v_{s0} - \sum_{j \in \mathcal{N}} v_{sj} x_j \le 0$$

beziehungsweise der umgestellten linearen Ungleichung

$$\sum_{j \in \mathcal{N}} (-v_{sj})\, x_j \le -\, v_{s0} \tag{5.4}$$

fordern. Damit ist durch

$$R(x^1) = \{x \in R_1 \,|\, v_{s0} - \sum_{j \in \mathcal{N}} v_{sj} x_j > 0\}$$

eine Schnittmenge definiert, für welche die beiden grundlegenden Forderungen $x^1 \in R(x^1)$ und $R(x^1) \cap S = \emptyset$ erfüllt sind. Nur $R(x^1) \subset R_1$ kann nicht garantiert werden, da der Fall $S = \emptyset$ nicht ausgeschlossen ist. Bei der erzeugten Schnittmenge handelt es sich um den *Gomory-Schnitt* (siehe [Go58]), der als erste Anwendung des Schnittprinzips in der Optimierung gilt.

Die Konstruktion der nachfolgenden Aufgabe P_2 erfolgt formal durch die Bildung der Menge

$$R_2 = \{x \in R_1 \mid \sum_{j \in \mathcal{N}} (-v_{sj}) x_j \leq -v_{s0}\}\,.$$

Die Aufgabe P_2 entspricht somit der LP-Relaxation mit der zusätzlichen linearen Restriktion (5.4).

Nach Einführung einer Schlupfvariablen x_{n+1} zur Überführung der Ungleichung (5.4) in eine Gleichung , wobei wegen (5.3) an x_{n+1} indirekt auch die Ganzzahligkeitsforderung gestellt wird, kann zur Lösung von P_2 die aus dem optimalen Simplextableau zur Basislösung x^1 vorliegende kanonische Form wie folgt erweitert werden:

$$\begin{array}{lcll} x_{B_i} & + \sum\limits_{j \in \mathcal{N}} y_{ij} x_j & = & y_{i0} \quad , \; i = 1, ..., m \\ x_{n+1} & + \sum\limits_{j \in \mathcal{N}} (-v_{sj}) x_j & = & -v_{s0} \\ z & + \sum\limits_{j \in \mathcal{N}} y_{m+1,j} x_j & = & y_{m+1,0} \end{array} \tag{5.5}$$

Da sich die Werte der Optimalitätsindikatoren der Nichtbasisvariable $y_{m+1,j} \geq 0\,,\; j \in \mathcal{N}$, bei Einbindung der Schnittrestriktion nicht ändern, wird auch die erweiterte Basislösung mit den Basisvariablen $x_{B_i},\; i = 1, ..., m$, und x_{n+1} als dual zulässig erkannt. Sie ist aber wegen $x_{n+1} = -v_{s0} < 0$ nicht primal zulässig. Damit kann zur Lösung von P_2 der duale Simplexalgorithmus zum Einsatz kommen.

Für $k \geq 2$ ergibt sich der folgende rekursive Lösungsablauf:

- Ist die Aufgabe P_k nicht lösbar, dannn ist deren zulässiger Bereich leer und es gilt automatisch auch $S = \emptyset\,.$
- Ist die Aufgabe P_k lösbar und die Optimallösung x^k von P_k ganzzahlig, dann ist x^k eine Optimallösung der rein-ganzzahligen linearen Optimierungsaufgabe P.
- Ist die Aufgabe P_k lösbar und die Optimallösung x^k von P_k nicht ganzzahlig, dann erfolgt die Bildung der Schnittmenge $R(x^k)$ und der Aufgabe P_{k+1} analog der vorliegenden Beschreibung für $k = 1$.

Unter Verwendung einer *lexikographischen* dualen Simplexmethode [Pie70] kann die Endlichkeit des skizzierten Verfahrens gezeigt werden. Leider ist das Verfahren gegen numerische Rundungsfehler anfällig.

Bemerkung 5.2
Die Aufgabe P_{q+1} wird formal durch $m + q$ Gleichungssrestriktionen beschrieben. Streicht man die im Verlauf des Verfahrens inaktiv werdenden Schnittrestriktionen, die dadurch gekennzeichnet sind, dass die zugehörigen Schlupfvariable x_{n+k} Basisvariable in der Optimallösung x^{q+1} sind, dann verbleiben für die Aufgabe P_{q+1} maximal $n+1$ Gleichungsrestriktionen (siehe [Pie70]). Damit kann ein starkes Anwachsen der Zahl der zu verarbeitenden Gleichungsrestriktionen verhindert werden.

Das Schnittebenenverfahren nach Gomory soll an einer kleinen aber markanten, in [Pie70] behandelten Optimierungsaufgabe demonstriert werden.

Beispiel 5.1
Gegeben ist die folgende rein ganzzahlige lineare Optimierungsaufgabe mit zwei Gleichungsrestriktionen und vier Variablen:

$$
\begin{array}{rcrcrcrcll}
z & = & & & x_2 & & & \to & \max & \\
 & & -2x_1 & + & 2x_2 & + & x_3 & = & 1 & \\
 & & 7x_1 & - & 2x_2 & & + \; x_4 & = & 14 & \\
 & & & & & & x_j & \geq & 0\,, & \textit{ganzzahlig}\,, \quad j = 1,2,3,4
\end{array}
$$

Fasst man die Variablen x_3 und x_4 als Schlupfvariable auf, so erkennt man, das die beiden Gleichungen aus den Ungleichungen $-2x_1 + 2x_2 \leq 1$ und $7x_1 - 2x_2 \leq 14$ entstanden sind. Der folgende Lösungsablauf kann also auch mit Hilfe einer graphischen Darstellung nachvollzogen werden.

Die Bestimmung einer Optimallösung für P_1 erfolgt mit Hilfe der primalen Simplexmethode, die in den nächsten drei Tabellen realisiert wird.

		x_1	x_2	x_3	x_4	
BV	c_B	0	1	0	0	$\bar{b}$
x_3	0	-2	$\boxed{2}$	1	0	1
x_4	0	7	-2	0	1	14
z		0	-1	0	0	0
x_2	1	-1	1	$\frac{1}{2}$	0	$\frac{1}{2}$
x_4	0	$\boxed{5}$	0	1	1	15
z		-1	0	$\frac{1}{2}$	0	$\frac{1}{2}$
x_2	1	0	1	$\frac{7}{10}$	$\frac{1}{5}$	$\frac{7}{2}$
x_1	0	1	0	$\frac{1}{5}$	$\frac{1}{5}$	3
z		0	0	$\frac{7}{10}$	$\frac{1}{5}$	$\frac{7}{2}$

Die letzte Tabelle beschreibt für P_1 die eindeutige Optimallösung mit den Werten $x_1 = 3$ und $x_2 = \frac{7}{2}$ für die Basisvariable. Die Lösung ist für die rein ganzzahlige lineare Optimierungsaufgabe nicht zulässig, da der Wert von x_2 nicht ganzzahlig ist.
Aus der zu x_2 gehörigen Zeile der letzten Tabelle kann sofort mittels (5.4) die zum Schnitt gehörige Restriktion

$$-\frac{7}{10}x_3 - \frac{1}{5}x_4 \leq -\frac{1}{2}$$

abgelesen werden. Für die graphische Veranschaulichung erhält man aus der gewonnenen Restriktion nach Elimination der Variablen x_3 und x_4 die Ungleichung $x_2 \leq 3$.

Die neue Aufgabe P_2 entsteht nun dadurch, dass die konstruierte Schnittrestriktion nach Einführen der Schlupfvariable x_5 der letzen Simplextabelle hinzugefügt wird. Da diese neue Basisvariable den negativen Wert $x_5 = -\frac{1}{2}$ annimmt, aber alle Optimalitätsindikatoren nichtnegativ sind, kommt jetzt die duale Simplexmethode zum Einsatz. Nach einer einzigen Transformation, bei der die Variable x_5 zur Nichtbasisvariable wird und damit die oben beschriebene Schnittrestriktion aktiv als Gleichung erfüllt wird, liegt bereits eine

Optimallösung für die Aufgabe P_2 vor. Für diese Aufgabe ist sie allerdings nicht die einzige. Die folgenden beiden Tabellen verdeutlichen den Verfahrensablauf.

		x_1	x_2	x_3	x_4	x_5	
BV	c_B	0	1	0	0	0	$\bar{b}$
x_2	1	0	1	$\frac{7}{10}$	$\frac{1}{5}$	0	$\frac{7}{2}$
x_1	0	1	0	$\frac{1}{5}$	$\frac{1}{5}$	0	3
x_5	0	0	0	$\boxed{-\frac{7}{10}}$	$-\frac{1}{5}$	1	$-\frac{1}{2}$
z		0	0	$\frac{7}{10}$	$\frac{1}{5}$	0	$\frac{7}{2}$
	δ_j			-1	-1		
x_2	1	0	1	0	0	1	3
x_1	0	1	0	0	$\frac{1}{7}$	$\frac{2}{7}$	$\frac{20}{7}$
x_3	0	0	0	1	$\frac{2}{7}$	$-\frac{10}{7}$	$\frac{5}{7}$
z		0	0	0	0	1	3

Die erzeugte Optimallösung mit den Werten $x_1 = \frac{20}{7}$, $x_2 = 3$ und $x_3 = \frac{5}{7}$ für die Basisvariable ist nicht zulässig für die Ausgangsaufgabe. Wählt man Variable x_1 zur Bildung der Schnittrestriktion aus, dann erhält man aus der zugehörigen Zeile der letzten Tabelle mittels (5.4) die Restriktion

$$-\frac{1}{7}x_4 - \frac{2}{7}x_5 \leq -\frac{6}{7}\,.$$

Für die graphische Veranschaulichung muss man die Variablen x_4 und x_5 eliminieren. Daraus resultiert dann die Ungleichung $x_1 \leq 2\,.$

Nach Einführen der Schlupfvariable x_6 kann die erzeugte Schnittrestriktion in das letzte Simplextableau integriert werden. Damit ist formal die neue Aufgabe P_3 entstanden. Da die neue Basisvariable den negativen Wert $x_6 = -\frac{6}{7}$ besitzt, kommt abermals die duale Simplexmethode zum Einsatz. Nach zwei Transformation, die in den folgenden drei Tabellen dargestellt sind, liegt eine Optimallösung für die Aufgabe P_3 vor.

		x_1	x_2	x_3	x_4	x_5	x_6	
BV	c_B	0	1	0	0	0	0	$\bar{b}$
x_2	1	0	1	0	0	1	0	3
x_1	0	1	0	0	$\frac{1}{7}$	$\frac{2}{7}$	0	$\frac{20}{7}$
x_3	0	0	0	1	$\frac{2}{7}$	$-\frac{10}{7}$	0	$\frac{5}{7}$
x_6	0	0	0	0	$\boxed{-\frac{1}{7}}$	$-\frac{2}{7}$	1	$-\frac{6}{7}$
z		0	0	0	0	1	0	3
	δ_j				0	$-\frac{7}{2}$		

		x_1	x_2	x_3	x_4	x_5	x_6	
BV	c_B	0	1	0	0	0	0	$\bar{b}$
x_2	1	0	1	0	0	1	0	3
x_1	0	1	0	0	0	0	1	2
x_3	0	0	0	1	0	-2	2	-1
x_4	0	0	0	0	1	2	-7	6
z		0	0	0	0	1	0	3
	δ_j					$-\frac{1}{2}$		
x_2	1	0	1	$\frac{1}{2}$	0	0	1	$\frac{5}{2}$
x_1	0	1	0	0	0	0	1	2
x_5	0	0	0	$-\frac{1}{2}$	0	1	1	$\frac{1}{2}$
x_4	0	0	0	1	1	0	-5	5
z		0	0	$\frac{1}{2}$	0	0	1	$\frac{5}{2}$

Zunächst soll bemerkt werden, dass die Schlupfvariable x_5 der zum ersten Schnitt gehörigen Restriktion abermals Basisvariable, jetzt mit positivem Wert, geworden ist. Damit erkennt man die Inaktivität der ersten Schnittrestriktion für die Aufgabe P_3. Deshalb wird die zu x_5 gehörige Zeile und Spalte der letzten Tabelle gelöscht, um die Zahl der zu verarbeitenden Restriktionen gering zu halten. Aus der jetzt vorliegenden Endtabelle kann auch die eindeutige Optimallösung für P_3 abgelesen werden.

		x_1	x_2	x_3	x_4	x_6	
BV	c_B	0	1	0	0	0	$\bar{b}$
x_2	1	0	1	$\frac{1}{2}$	0	1	$\frac{5}{2}$
x_1	0	1	0	0	0	1	2
x_4	0	0	0	1	1	-5	5
z		0	0	$\frac{1}{2}$	0	1	$\frac{5}{2}$

Die Optimallösung mit den Werten $x_1 = 2$, $x_2 = \frac{5}{2}$ und $x_4 = 5$ für die Basisvariable ist nicht zulässig für die Ausgangsaufgabe. Die einzige gebrochen Variable x_2 wird zur Bildung der Schnittrestriktion herangezogen. Man erhält aus der zugehörigen Zeile der letzten Tabelle aus (5.4) die Restriktion

$$-\frac{1}{2}x_3 \leq -\frac{1}{2}\,.$$

Für die graphische Veranschaulichung muss die Variable x_3 eliminiert werden. Damit ergibt sich die Ungleichung $-x_1 + x_2 \leq 0\,$.

Nach Einführung der Schlupfvariable x_7 wird die Schnittrestriktion in das letzte Simplextableau geschrieben. Die damit neue entstandene Aufgabe P_4 mit der wegen $x_7 = -\frac{1}{2}$ unzulässigen Basislösung wird mit Hilfe des dualen Simplexalgorithmus gelöst. Nach einer Transformation, die in den folgenden beiden Tabellen realisiert wird, liegt dann eine Optimallösung für die Aufgabe P_4 vor.

		x_1	x_2	x_3	x_4	x_6	x_7	
BV	c_B	0	1	0	0	0	0	$\bar{b}$
x_2	1	0	1	$\frac{1}{2}$	0	1	0	$\frac{5}{2}$
x_1	0	1	0	0	0	1	0	2
x_4	0	0	0	1	1	-5	0	5
x_7	0	0	0	$\boxed{-\frac{1}{2}}$	0	0	1	$-\frac{1}{2}$
z		0	0	$\frac{1}{2}$	0	1	0	$\frac{5}{2}$
	δ_j			-1				
x_2	1	0	1	0	0	1	1	2
x_1	0	1	0	0	0	1	0	2
x_4	0	0	0	0	1	-5	2	4
x_3	0	0	0	1	0	0	-2	1
z		0	0	0	0	1	1	2

Die abzulesende Optimallösung von P_4 ist offenbar in allen Komponenten ganzzahlig. Damit ist $x_1^* = 2$, $x_2^* = 2$ mit $z^* = 2$ auch eine Optimallösung der Ausgangsaufgabe.

Bemerkung 5.3
Aus (5.2) lässt sich eine naheliegende einfache Schnittbedingung ableiten. Um die nicht zu Menge S gehörige Lösung x^1 auszuschließen, aber alle anderen ganzzahligen Lösungen beizubehalten, muss man nur die Erfüllung der Ungleichung $\sum_{j\in\mathcal{N}} x_j \geq 1$ fordern. Sie definiert den sogenannten *Dantzig-Schnitt*. Bei Verwendung dieser Schnittbedingung kann die Endlichkeit eines entsprechend konstruierten Algorithmus *nicht* gezeigt werden.

Ausdrücklich sei darauf hingewiesen, dass die in diesem Abschnitt besprochenen Schnittbedingungen *nur* für rein ganzzahlige lineare Optimierungsaufgaben in Gleichungsform anwendbar sind. Für gemischt ganzzahlige lineare Optimierungsaufgaben in Gleichungsform, in die jede ganzzahlige lineare Optimierungsaufgabe transformierbar ist, haben Schnittrestriktion eine andere Gestalt.

Für eine gemischt ganzzahlige lineare Optimierungsaufgaben in Gleichungsform sei x^1 eine optimale Basislösung für die erste Teilaufgabe P_1, bei der an die Basisvariable x_{B_s} die Ganzzahligkeitsforderung gestellt wird, sie aber in der in (5.2) dargestellten Zeile s des optimalen Simplextabeaus den nichtganzzahligen Wert y_{s0} annimmt. Dann wird nur dieser Wert y_{s0} mittels $y_{s0} = u_{s0} + v_{s0}$, $u_{s0} = \lfloor y_{s0} \rfloor$ in zwei Zahlen zerlegt. Dafür muss zusätzlich die Menge $\mathcal{N}$ in die Mengen

$$\mathcal{N}^+ = \{j \in \mathcal{N} \,|\, y_{sj} \geq 0\}\,, \qquad \mathcal{N}^- = \{j \in \mathcal{N} \,|\, y_{sj} < 0\}$$

zerlegt werden. Eine mögliche Schnittbedingung hat dann die Gestalt

$$\sum_{j\in\mathcal{N}^+} y_{sj}x_j + \sum_{j\in\mathcal{N}^-} \frac{v_{s0}}{1-v_{s0}}\,(-y_{sj})\,x_j \geq v_{s0}\ . \tag{5.6}$$

Eine ausführliche Begründung kann zum Beispiel in [Bu72] nachlesen werden.

Kapitel 6

Näherungsverfahren

Wegen der exponentiell von der Problemgröße der Aufgabe abhängigen Rechenzeit sind schwierige Probleme der diskreten Optimierung ab einem gewissen Datenumfang für exakte Lösungsverfahren kaum zugänglich. Deshalb wird man sich häufig mit Näherungslösungen für diskrete Optimierungsprobleme zufrieden geben müssen, die aber wenigstens ein günstigeres Zeitverhalten aufweisen sollen.
Im Prinzip ist jede zulässige Lösung einer diskreten Optimierungsaufgabe auch eine Näherungslösung. Damit kann man einen Algorithmus A als *Näherungsverfahren* für ein diskretes Optimierungsproblem P bezeichnen, wenn er für jedes konkrete Beispiel nach endlich vielen Schritten entweder eine zulässige Lösung liefert oder die Aussage "Aufgabe unlösbar" tätigt. Der Hinweis auf ein günstigeres Zeitverhalten legt im Sinne der Komplexitätstheorie nahe, vorrangig polynomiale Näherungsverfahren einzusetzen.
Ausführliche Darstellungen zu Grundprinzipien heuristischer Lösungsverfahren, die sich in Eröffnungsverfahren und lokale Verbesserungsverfahren unterteilen lassen, letztere wiederum deterministisch oder stochastisch ablaufen können, findet man in zum Beispiel [BoGr93], [Iba87] oder [KoVy00].

6.1 Güte von Näherungsverfahren

Gegeben sei ein diskretes Optimierungsproblem P der Gestalt (1.1), welches für jedes konkrete Beispiel eine Optimallösung besitzen soll. Für den Spezialfall eines kombinatorischen Optimierungsproblems war dies automatisch erfüllt. Die dabei zu minimierende Zielfunktion kann auch durch eine zu maximierende Zielfunktion ersetzt werden.

Es sei A ein Algorithmus, der für jedes Beispiel ρ der diskreten Optimierungsaufgabe P eine zulässige Lösung $x(\rho)$ mit dem Zielfunktionswert $f(x(\rho))$ berechnet. Eine zugehörige Optimallösung sei mit $x^*(\rho)$ und der zugehörige optimale Zielfunktionswert mit $f(x^*(\rho))$ bezeichnet.

Definition 6.1
Ein Algorithmus A heißt *absoluter Näherungsalgorithmus* mit Genauigkeit h, falls für jedes Beispiel ρ von P die Ungleichung

$$| f(x^*(\rho)) - f(x(\rho)) | \leq h$$

gilt.

Das Konzept eines absoluten Näherungsalgorithmus bringt für diskrete Optimierungsaufgaben im Allgemeinen leider keinen Vorteil hinsichtlich des Zeitverhaltens des Näherungsalgorithmus. Dies lässt sich sehr schön am Beispiel des Rucksackproblems erläutern.

Beispiel 6.1
Gegeben sei das folgende im Abschnitt 2.2 eingeführte Rucksackproblem:

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &\rightarrow \max \\ \sum_{j=1}^{n} g_j\, x_j &\leq G \\ x_j &\in \{0;1\}\,, \quad j = 1, ..., n \end{aligned}$$

Alle Daten werden hier als positiv und ganzzahlig vorausgesetzt. Damit ist auch jeder Zielfunktionswert einer zulässigen Lösung automatisch ganzzahlig.

Ein Beispiel ρ ist durch die Eingabe der Zahlen n und G sowie der beiden Zeilenvektoren $c^\top$ und $g^\top$ gegeben. Formal wird der Datensatz mit $\rho = (n, c^\top, g^\top, G)$ beschrieben. Eine Optimallösung für dieses Beispiel sei mit $x^*(\rho)$ bezeichnet.

Die Bestimmung einer Näherungslösung mit einer vorgegebenen absoluten Genauigkeit $h \geq 1$ hinsichtlich der Abweichung vom maximalen Zielfunktionswert entspricht dann der Suche einer Lösung des folgenden Ungleichungssystems in binären Variablen:

$$\begin{aligned} \sum_{j=1}^{n} c_j x_j^* - \sum_{j=1}^{n} c_j x_j &\leq h \\ \sum_{j=1}^{n} g_j\, x_j &\leq G \\ x_j &\in \{0;1\}\,, \quad j = 1, ..., n \end{aligned}$$

Angenommen, es gibt einen polynomialen Algorithmus A zur Lösung dieses Problems. Dieser erzeugt dann für den Datensatz $\rho = (n, c^\top, g^\top, G)$ in Polynomialzeit eine zulässige Lösung $x(\rho)$ mit $c^\top x^*(\rho) - c^\top x(\rho) \leq h$.
Ein weiterer Datensatz $\tilde{\rho}$ unterscheide sich vom Datensatz ρ nur dadurch, dass alle Zielfunktionskoeffizienten mit dem Faktor $h + 1$ multipliziert werden. Dann besitzt die Aufgabe $\tilde{\rho} = (n, (h + 1)c^\top, g^\top, G)$ natürlich auch die Optimallösung $x^*(\rho)$. Der Algorithmus A erzeugt auch für die Aufgabe $\tilde{\rho}$ in Polynomialzeit eine zulässige Lösung, die mit $\tilde{x}(\rho)$ bezeichnet wird. Folglich gilt $(h+1)c^\top x^*(\rho) - (h+1)c^\top \tilde{x}(\rho) \leq h$. Daraus folgt aber sofort $c^\top x^*(\rho) - c^\top \tilde{x}(\rho) \leq \frac{h}{h+1} < 1$. Wegen der vorausgesetzten Ganzzahligkeit aller Daten muss dann schärfer $c^\top x^*(\rho) - c^\top \tilde{x}(\rho) = 0$ gelten, womit $\tilde{x}(\rho)$ für den Datensatz $\tilde{\rho}$ eine Optimallösung des Rucksackproblems ist. Damit würde der Algorithmus A in Polynomialzeit bei Verwendung der äquivalenten mit $h + 1$ multiplizierten Zielfunktion stets eine Optimallösung des Rucksackproblems unter der oben genannten Voraussetzungen erzeugen. Das Rucksackproblem gehört aber zur Klasse der $\mathcal{NP}$-schweren Optimierungsprobleme. Folglich sind polynomiale Algorithmen zur Bestimmung einer Optimallösung derzeit nicht bekannt.

Für die Betrachtung relativer Fehlerschranken wird vorausgesetzt, dass für jedes Beispiel ρ der zu betrachtenden lösbaren diskreten Optimierungsprobleme P stets $f(x(\rho)) > 0$ und damit auch $f(x^*(\rho)) > 0$ gilt.

Definition 6.2
Ein Algorithmus A heißt *relativer Näherungsalgorithmus* mit Genauigkeit h, falls für jedes Beispiel ρ von P die Ungleichung

$$1 \leq \frac{f(x(\rho))}{f(x^*(\rho))} \leq h \quad \text{für Minimierungsprobleme}$$

beziehungsweise

$$h \leq \frac{f(x(\rho))}{f(x^*(\rho))} \leq 1 \quad \text{für Maximierungsprobleme}$$

gilt.

Wählt man für diskrete Optimierungsprobleme mit zu minimierender Zielfunktion $h = 1+\varepsilon$ und für Maximierungsprobleme $h = 1-\varepsilon$ jeweils mit $0 < \varepsilon < 1$, dann erhält man in beiden Fällen die Charakterisierung von ε-optimalen Näherungsalgorithmen. Dies entspricht der nachfolgenden Definition.

Definition 6.3
Ein Algorithmus A heißt *ε-optimaler Näherungsalgorithmus* mit Genauigkeit ε, falls für jedes Beispiel ρ von P die Ungleichung

$$\frac{|\, f(x^*(\rho)) - f(x(\rho))\,|}{f(x^*(\rho))} \leq \varepsilon$$

gilt.

Für die nächste Definition wird die Abhängigkeit eines Näherungsalgorithmus A für das diskrete Optimierungsproblem P vom Parameter ε betrachtet.

Definition 6.4
Ein Algorithmus $A(\varepsilon)$ heißt *Näherungsschema*, wenn für jedes $\varepsilon \in (0;1)$ durch den Algorithmus $A(\varepsilon)$ ein ε-optimaler Algorithmus erzeugt wird.

Für ein Näherungsschema $A(\varepsilon)$ hat man zwei wesentliche Eingabegrößen. Dies ist zum einen der benötigte Speicherplatz $s(\rho)$ für ein Beispiel des zu betrachtenden diskreten Optimierungsproblems P und zum anderen die vorzugebende Genauigkeit ε.

Definition 6.5
Das Näherungsschema $A(\varepsilon)$ heißt *polynomial*, wenn das Zeitverhalten der zugehörigen ε-optimalen Algorithmen für jedes $\varepsilon \in (0;1)$ polynomial in Abhängigkeit der Eingabelänge $s(\rho)$ ist.

Für ein so definiertes polynomiales Näherungsschema $A(\varepsilon)$ kann die Rechenzeit durchaus exponentiell anwachsen, wenn man das Zeitverhalten in Abhängigkeit von der Genauigkeit, also der wachsenden Größe $\frac{1}{\varepsilon}$ betrachtet.

Definition 6.6
Ein polynomiales Näherungsschema $A(\varepsilon)$ heißt *vollständig polynomial*, wenn das Zeitverhalten in Abhängigkeit vom Aufwand ε polynomial in der Größe $\frac{1}{\varepsilon}$ ist.

Wenn für ein $\mathcal{NP}$-schweres diskretes Optimierungsproblem ein vollständig polynomiales Näherungsschema existiert, dann sind die zu konstruierenden ε-optimalen Algorithmen meist sehr anspruchsvoll und benötigen oft einen erhöhten Speicherplatzaufwand, um die Polynomialität in $\frac{1}{\varepsilon}$ zu garantieren.

6.2 Greedy-Algorithmen

Im Abschnitt 1 wurde in (1.2) eine kombinatorische Optimierungsaufgabe mit Hilfe eines Mengensystems formuliert. Eine vernünftige näherungsweise oder exakte Lösung eines solchen Problems erfordern spezielle Strukturen für ein derartiges Mengensystem.

Gegeben sei die Grundmenge $N = \{1, ..., n\}$. Mit $\mathcal{F}$ sei eine Familie von Teilmengen von N bezeichnet. Für ein solches Mengensystem gilt $\mathcal{F} \subseteq Pot(N)$.

Definition 6.7
Ein Mengensystem $\mathcal{F}_U \subseteq Pot(N)$ heißt *Unabhängigkeitssystem* auf der Grundmenge N, wenn die Bedingungen

- $\emptyset \in \mathcal{F}_U$
- $K \in \mathcal{F}_U \wedge L \subset K \Longrightarrow L \in \mathcal{F}_U$

erfüllt sind.

Die zweite Bedingung verlangt, das mit jeder Menge F aus dem Unabhängigkeitssystems $\mathcal{F}_U$ auch alle ihre Teilmengen zu diesem Unabhängigkeitssystem gehören. Man spricht deshalb auch von einem sogenannten *monotonen* Mengensystem.

Bemerkung 6.1
Die Mengen F aus einem Unabhängigkeitssystem $\mathcal{F}_U$ werden *unabhängige Mengen* genannt. Die übrigen Teilmengen $F \in Pot(N) \setminus \mathcal{F}_U$ bezeichnet man als abhängige Mengen.
Eine unabhängige Menge F heißt *maximal*, wenn $F \cup \{k\} \notin \mathcal{F}_U$ für alle $k \in N \setminus F$ gilt. Eine maximal unabhängige Menge ist also keine echte Teilmenge einer anderen Menge aus dem Unabhängigkeitssystem.

Gegeben seien ein Unabhängigkeitssystem $\mathcal{F}_U$ und Zahlen $c_j\,,\, j \in N$. Eine Teilmenge $F \subseteq N$ erhält üblicherweise die Bewertung $c(F) = \sum_{j \in F} c_j$.
Die Bestimmung einer Menge F^* aus dem Unabhängigkeitssystem $\mathcal{F}_U$, so dass $c(F^*)$ maximal ist, heißt *Maximierungsproblem über einem Unabhängigkeitssystem* und kann wie folgt notiert werden:

$$\begin{aligned} c(F) &\longrightarrow \max \\ F &\in \mathcal{F}_U \end{aligned} \tag{6.1}$$

Bei Maximierungsproblemen über Unabhängigkeitssystemen lassen sich Indizes j mit nichtpositiver Bewertung c_j eliminieren:
Einer beliebige Menge $F \in \mathcal{F}_U$, $F \neq \emptyset$, wird die Menge $\hat{F} = F \backslash \{j \in N \mid c_j \leq 0\}$ zugeordnet. Es gilt $\hat{F} \in \mathcal{F}_U$ und $c(\hat{F}) \geq c(F)$, womit $\hat{F}$ zulässig und bezüglich des Zielkriteriums nicht schlechter als F ist. Folglich kann eine derartige Menge F aus dem Optimierungsprozess ausgeschlossen werden.

Gilt $c_j > 0\,,\ j \in N$, dann liefert das Maximierungsproblemen über Unabhängigkeitssystemen stets eine maximale unabhängige Menge F^*.

Beispiel 6.2
Das Rucksackproblem

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &\rightarrow \max \\ \sum_{j=1}^{n} g_j\, x_j &\leq G \\ x_j &\in \{0;1\}\,, \quad j = 1, ..., n \end{aligned}$$

ist ein Maximierungsproblem über einem Unabhängigkeitssystem. Vorausgesetzt wird die Positvität aller Daten.

Die Grundmenge $N = \{1, 2, ..., n\}$ entspricht der Nummerierung der Gegenstände, die eingepackt werden können.
Eine zulässige Lösung x wird durch eine Menge F mittels der Zuordnung

$$x_j = 1 \Longleftrightarrow j \in F\,,\ j = 1, ..., n$$

beschrieben. Damit sind die Diskretheitsforderungen an x berücksichtigt.

Die Zulässigkeit hinsichtlich der Kapazitätsgrenze G wird durch

$$F \in \mathcal{F}_U \Longleftrightarrow g(F) = \sum_{j \in F} g_j \leq G$$

geregelt.

Das so beschriebene Mengensystem $\mathcal{F}_U$ erfüllt die beiden Bedingungen an ein Unabhängigkeitssystem. Offensichtlich gilt $\emptyset \in \mathcal{F}_U$, denn der Vektor $x = 0$ ist wegen $G > 0$ zulässig. Für eine Menge $F \in \mathcal{F}_U$ mit $F \neq \emptyset$ gilt $g(F) \leq G$. Wegen der Positivität der Gewichte g_j gilt für jede Teilmenge $L \subset F$ auch $g(L) = \sum_{j \in L} g_j < \sum_{j \in F} g_j \leq G$. Damit gilt $L \in \mathcal{F}_U$.

Der Zielfunktionswert einer zulässigen Lösung x des Rucksackproblems wird mit Hilfe der Mengenzuordnung durch

$$z(x) = \sum_{j=1}^{n} c_j\, x_j = \sum_{j \in F} c_j = c(F)$$

erfasst. Wegen der Positivität des Nutzenskoeffizienten c_j ist eine unabhängige Menge, die nicht maximal ist, keine Optimallösung des Rucksackproblems.

Unabhängigkeitssysteme sind recht allgemeine mathematische Objekte, hinter denen sich aber viele konkrete Anwendungen verbergen. Für die Formulierung tiefgreifenderer mathematischer Aussagen sind an die Unabhängigkeitssysteme allerdings noch weitere einschneidende Forderungen notwendig.

Definition 6.8
Ein Mengensystem $\mathcal{F}_M \subseteq Pot(N)$ heißt *Matroid* auf der Grundmenge N, wenn $\mathcal{F}_M$ ein Unabhängigkeitssystem ist und die Bedingung

$$K \in \mathcal{F}_M \wedge L \in \mathcal{F}_M \wedge |K| = |L| + 1 \quad \Longrightarrow \quad \exists p \in K \setminus L : L \cup \{p\} \in \mathcal{F}_M$$

erfüllt ist.

Die zusätzliche Forderung an ein Unabhängigkeitssystem verlangt, dass in einer Menge $K \in \mathcal{F}_M$, deren Mächtigkeit um eins größer als die Mächtigkeit einer Menge $L \in \mathcal{F}_M$ ist, stets ein nicht zu L gehörender Index p existieren muss, so dass die um p erweiterte Menge L auch zu $\mathcal{F}_M$ gehört.

Beispiel 6.3
Es sei $N = \{1, ..., n\}$ eine Grundmenge und $N_1, ..., N_k$ eine Partition dieser Grundmenge. Damit gilt $N = \bigcup_{i=1}^{k} N_i$ und $N_p \cap N_q = \emptyset$, $p, q = 1, ..., k, p \neq q$. Desweiteren sind nichtnegative ganze Zahlen $b_1, ..., b_k$ gegeben. Das Mengensystem

$$\mathcal{F}_{PM} = \{ F \subseteq N \mid |F \cap N_i| \leq b_i, \, i = 1, ..., k \}$$

ist ein Matroid und wird *Partitionsmatroid* genannt.
Durch die Bedingung $|F \cap N_i| \leq b_i$ ist garantiert, dass jede Menge $F \in \mathcal{F}_{PM}$ höchstens b_i Elemente aus der Menge N_i enhält.
Für $n = 5$, $N_1 = \{1, 2\}$, $N_2 = \{3, 4\}$, $N_3 = \{5, 6\}$, $b_1 = 1$, $b_2 = 0$ und $b_3 = 2$ erhält man das Matroid

$$\mathcal{F}_{PM} = \{\emptyset, \{1\}, \{2\}, \{5\}, \{6\}, \{1, 5\}, \{1, 6\}, \{2, 5\}, \{2, 6\}, \{5, 6\}, \{1, 5, 6\}, \{2, 5, 6\}\}.$$

Es gibt einen einfachen theoretischen Zusammenhang zur Darstellung eines Unabhängigkeitssystems mit Hilfe von Matroiden.

Satz 6.1
Ist $\mathcal{F}_U$ ein Unabhängigkeitssystem auf der Grundmenge N, dann gibt es endlich viele Matroide $\mathcal{F}_{M_1}$,...,$\mathcal{F}_{M_l}$, über der gleichen Grundmenge N, so dass

$$\mathcal{F}_U = \bigcap_{k=1}^{l} \mathcal{F}_{M_k}$$

gilt.

In konkreten Fällen kann die Zahl der für die Beschreibung eines Unabhängigkeitssystems notwendigen Matroide recht hoch sein.

Im Folgenden wird für das Maximierungsproblem (6.1) über dem Unabhängigkeitssystem $\mathcal{F}_U$ ein simples Näherungsverfahren beschrieben, das im Falle des Vorliegens eines Matroids sogar eine Optimallösung erzeugt. Es ist ein Verfahren, dass ausgehend von der leeren Menge in jedem Schritt nach der größtmöglichen Verbesserung des Zielfunktionswertes der bis dahin vorliegenden Teillösung trachtet. Deshalb wird dieses Verfahren als *greedy* oder *myophisch* (gierig, kurzsichtig) bezeichnet.

Greedy-Algorithmus für Maximierungsprobleme über Unabhängigkeitssysteme

1. Sortiere die Bewertungen in nichtaufsteigender Reihenfolge $c_1 \geq c_2 \geq ... \geq c_n$. Setze $F_G = \emptyset$ und $j = 1$.
2. Im Falle $c_j \leq 0$ endet der Algorithmus.
   Gilt $F_g \cup \{j\} \in \mathcal{F}_U$, dann setze $F_g := F_g \cup \{j\}$.
3. Für $j = n$ endet der Algorithmus.
   Anderenfalls setze $j := j + 1$ und gehe zu Schritt 2.

Der Greedy-Algorithmus durchläuft nach der Sortierung jedes Element $j \in N$ mit $c_j > 0$ genau einmal. Dabei wird der Index j entweder in die Greedy-Lösung aufgenommen oder für immer verworfen.

Satz 6.2
Es sei $\mathcal{F}_U$ ein System unabhängiger Mengen über der Grundmenge N, F_g die mit dem Greedy-Algorithmus erzeugte Näherungslösung und F^* eine Optimallösung des Maximierungsproblems (6.1). Dann ist $\mathcal{F}_U$ genau dann ein Matroid über der Grundmenge N, wenn $c(F_g) = c(F^*)$ für alle rellen Bewertungen c_j , $j = 1, ..., n$, gilt.

Für ein Maximierungsproblem über einem Matroid liefert die erzeugte Greedy-Lösung automatisch eine gesuchte Optimallösung.

Beispiel 6.4
Gegeben sei die Grundmenge $N = \{1; 2; 3; 4; 5\}$ mit Bewertung $c_1 > c_2 > c_3 > c_4 > 0 > c_5$.

- Mit $\mathcal{F}_U = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{3, 4\}, \{1, 3, 4\}\}$ ist ein Unabhängigkeitssystem, aber kein Matroid gegeben. Der Greedy-Algorithmus erzeugt die Näherungslösung $F_g = \{1, 2\}$. In Abhängigkeit von der konkreten Bewertung gilt für die Optimallösung F^* die Beziehung $F^* \in \{\{1, 2\}, \{1, 3, 4\}\}$. Für $c_2 = c_3 + c_4$ sind beide Mengen optimal. Im Fall $c_2 > c_3 + c_4$ stimmt die erzeugte Greedy-Lösung mit der einzigen Optimallösung übereinein. Auch wenn kein Matroid vorliegt, kann die Greedy-Lösung für bestimmte Bewertungen den besten Zielfunktionswert realisieren. Dies gilt aber nicht für alle möglichen Bewertungen, hier nicht für $c_2 < c_3 + c_4$.
- Für das Matroid $\mathcal{F}_M = \{F \,|\, F \subseteq N \wedge |F| \leq 3\}$ erzeugt der Greedy-Algorithmus die Optimallösung $F^* = F_g = \{1; 2; 3\}$.
- Das Mengensystem $\mathcal{F} = \{\emptyset, \{5\}, \{1, 2, 3, 4\}, \{1, 2, 3, 4, 5\}\}$ ist kein Unabhängigkeitssystem. Wendet man trotzdem den Greedy-Algortihmus an, so erhält man die Lösung $F_g = \emptyset$ mit $c(F_g) = 0$. Damit wird der Optimalwert $c(\{1, 2, 3, 4\}) = c_1 + c_2 + c_3 + c_4$ weit verfehlt.

Viele diskrete Optimierungsmodelle werden mit Hilfe von Variablen beschrieben. Bezüglich dieser Modelle lässt sich das Grundprinzip des Greedy-Algorithmus als spezielles Suchverfahren wie folgt abwandeln:

> Gegeben ist ein Teilproblem einer diskreten Optimierungsaufgabe in den Variablen $x_1, ..., x_n$, bei dem k Variable fixiert und die restlichen $n-k$ Variable noch frei wählbar sind. Man bestimmt diejenige freie Variable x_j, die lokal die Zielfunktion in einem Schritt optimal verbessert und legt dann ihren Wert unter Beachtung der einzuhaltenden Restriktionen fest. Anschließend betrachtet man das Teilproblem mit nur noch $n-k-1$ freien Variablen.

Als Beispiel eines so formulierten Greedy-Algorithmus für zu minimierende Zielfunktionen kann auch die Bestimmung eines ersten Transportplans für das klassische Transportproblem nach der Gesamtminimumregel aufgefasst werden.

6.3 Näherungsverfahren für das Rucksackproblem

Das bereits mehrfach angesprochene Rucksackproblem **RSP**

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &\rightarrow \max \\ \sum_{j=1}^{n} g_j\, x_j &\leq G \\ x_j &\in \{0;1\}, \quad j = 1, ..., n \end{aligned}$$

gehört trotz seiner einfachen Modellierung zur Klasse der schwer lösbaren Optimierungsprobleme. Mit Hilfe der dynamischen Optimierung kann ein pseudopolynomialer Algorithmus zu seiner exakten Lösung angegeben werden. An dieser Stelle sollen konkrete Näherungsverfahren beschrieben und bewertet werden. Ausführliche Darstellungen findet man in den beiden Monographien [MaTo90] und [KePfPi04].

Generell kann für Rucksackproblem $c_j > 0$, $g_j > 0$, $g_j \leq G$, $j = 1, ..., n$, und $\sum_{j=1}^{n} g_j > G$ vorausgesetzt werden. Oft nimmt man auch die Ganzzahligkeit alle benötigten Zahlen an.

Für die zu beschreibenden Näherungsverfahren wird es sich als als günstig erweisen, die Variablen und damit die zu betrachtenden Gegenstände so durchzunummerieren, dass

$$\frac{c_1}{g_1} \geq \frac{c_2}{g_2} \geq \ ... \ \geq \frac{c_n}{g_n} \tag{6.2}$$

gilt. Hier werden die Gegenstände intuitiv nach ihrer "Effektivität", also dem Verhältnis von Nutzen zu Aufwand (Gewicht) sortiert. Diese Sortierung wird auch die Basis für den zu beschreibenden Greedy-Algorithmus sein.

Generell hat man durch die vorzunehmende Sortierung in Abhängigkeit von der Dimension n grundsätzlich einen Mindestaufwand von $\mathcal{O}(n \log n)$ bei künftigen Aufwandsbetrachtungen zu berücksichtigen.

Einen nützlichen Einblick in die Struktur des Rucksackproblems liefert die zugehörige *stetige Relaxation*. Gleichzeitig wird auch die Bedeutung der vorgeschlagenen Quotientensortierung einsichtig. Die Forderungen $x_j \in \{0;1\}$, $j = 1, ..., n$, werden durch die Beschränkungen $0 \leq x_j \leq 1$, $j = 1, ..., n$, ersetzt. Man geht formal von einer beliebigen Teilbarkeit eines jeden Gegenstandes aus. Die Relaxation **LP** des Rucksackproblems wird durch die folgende lineare Optimierungsaufgabe beschrieben:

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &\rightarrow \max \\ \sum_{j=1}^{n} g_j\, x_j &\leq G \\ 0 \leq x_j &\leq 1\,, \quad j = 1, ..., n \end{aligned} \tag{6.3}$$

Liegt die in (6.2) vorgeschlagene Sortierung bereits vor, dann kann man sofort eine Optimallösung x^{LP} und den zugehörigen Optimalwert z^{LP} der stetigen Relaxation angeben. Dazu muss schrittweise jede Variable in Reihenfolge ihrer Sortierung so groß wie möglich gewählt werden. Dies entspricht der Übertragung der Idee des Greedy-Verfahrens auf die stetige Relaxation des Rucksackproblems.

Dazu ist nur der sogenannte *Split-Index* $s \in \{2, ..., n\}$ zu bestimmen, für den

$$\sum_{j=1}^{s-1} g_j \leq G \quad \wedge \quad \sum_{j=1}^{s} g_j > G \tag{6.4}$$

gilt. Er trennt die Variablenmengen, die ihren Wert für eine Optimallösung an der oberen beziehungsweise unteren Schranke annehmen werden.

Satz 6.3
Eine Optimallösung x^{LP} der stetigen Relaxation (6.3) des Rucksackproblems unter der Voraussetzung (6.2) ist durch die Vorschrift

$$\begin{aligned} x_j^{LP} &= 1\,, && j = 1, ..., s-1 \\ x_s^{LP} &= \frac{1}{g_s}\,(G - \sum_{j=1}^{s-1} g_j) \\ x_j^{LP} &= 0\,, && j = s+1, ..., n \end{aligned}$$

gegeben. Für den zugehörigen optimalen Zielfunktionswert gilt

$$z^{LP} = \sum_{j=1}^{s-1} c_j + \frac{c_s}{g_s}\,(G - \sum_{j=1}^{s-1} g_j)\ .$$

Unter Beachtung der notwendigen Sortierung beträgt der Aufwand zur Bestimmung des Split-Index s und der damit möglichen Darstellung einer Optimallösung $\mathcal{O}(n \log n)$. An dieser Stelle soll nur bemerkt werden, dass es zum Bestimmen des Split-Index s auch einen Algorithmus mit Aufwand $\mathcal{O}(n)$ gibt, der demzufolge ohne Sortierung auskommt. Hier sei auf die Beschreibung in [KePfPi04] verwiesen.

Es sei x^* eine Optimallösung des Rucksackproblems und z^* der zugehörige Optimalwert. Dann gilt auf Grund der vorgenommenen Relaxation stets $z^* \leq z^{LP}$. Die bestimmte Optimallösung x^{LP} der stetigen Relaxation LP besitzt höchstens eine Komponente, die nicht ganzzahlig ist. Für diese Komponente s gilt $0 \leq x_s^{LP} < 1$. Im Falle $x_s^{LP} = 0$ ist x^{LP} eine zulässige und damit auch optimale Lösung des Rucksackproblems.

Bemerkung 6.2
Gilt $0 < x_s^{LP} < 1$, dann erhält man aus der Optimallösung x^{LP} der stetigen Relaxation des Rucksackproblems auf einfache Weise eine Näherungslösung für das Rucksackproblem, indem man der Variablen x_s den Wert Null zuweist. Die so entstandene Näherungslösung x^{gs} mit den Variablenwerten $x_j^{gs} = 1\,,\ j = 1, ..., s-1,\ x_j^{gs} = 0\,,\ j = s, ..., n$, und dem Zielfunktionswert $z^{gs} = \sum_{j=1}^{s-1} c_j$ wird als *Greedy-Split-Lösung* bezeichnet.

Der eigentliche Greedy-Algorithmus versucht nur noch, ausgehend von der Greedy-Split-Lösung x^{gs}, bei der bereits die ersten $s-1$ Gegenstände eingepackt wurden und der Gegenstand s als erster die Restkapazität $G - \sum_{j=1}^{s-1} g_j$ überschreitet, weitere Gegenstände mit Nummer $j \in \{s+1, ..., n\}$ unter Beachtung der Restkapazität zu berücksichtigen.

Greedy-Algorithmus für das Rucksackproblem

1. Sortiere die Gegenstände nach ihrer Effektivität (6.2).
   Setze $x_1^g := x_2^g := ... := x_n^g := 0$, $z^g := 0$, $G_R := G$ und $j := 1$.
2. Gilt $g_j \leq G_R$, dann setze $x_j^g := 1,\ G_R := G_R - g_j$ und $z^g := z^g + c_j$
3. Für $j = n$ endet der Algorithmus.
   Anderenfalls setze $j := j + 1$ und gehe zu Schritt 2.

Da die anfängliche Sortierung für den Greedy-Algorithmus zwingend notwendig ist und die Entscheidung über die Wertzuweisung der Variablen in linearer Zeit abläuft, kann der entstehende Aufwand $\mathcal{O}(n \log n)$ nicht reduziert werden.

Der hier formulierte Greedy-Algorithmus liefert neben der *Greedy-Lösung* x^g mit Zielfunktionswert z^g auch die noch verbleibende, nicht mehr nutzbare Restkapazität G_R zur Information. Bei der Realisierung des Greedy-Algorithmus wird der Split-Index nicht explizit ermittelt. Er lässt sich aber leicht bestimmen. Es ist derjenige Index s, bei dem zum ersten Mal während des Verfahrens die zu testende Ungleichung "$g_j \leq G_R$?" nicht erfüllt ist. Damit gilt $s = \min\{j \in \{1, ..., n\} \,|\, x_j^g = 0\}$ und die Greedy-Split-Lösung x^{gs} ist damit verifiziert.

Beispiel 6.5
Gegeben sei ein Rucksackproblem mit 7 Gegenständen und einer Kapazität $G = 22$. Die einzelnen Bewertungen und Gewichte sind der folgenden Tabelle zu entnehmen:

j	1	2	3	4	5	6	7
c_j	11	12	14	10	17	3	4
g_j	4	5	7	8	15	3	5

Die vorgegebene Nummerierung entspricht bereits der benötigten Quotientensortierung.

Der Greedy-Algorithmus liefert die Lösung $x_1^g = x_2^g = x_3^g = 1$, $x_4^g = x_5^g = 0$, $x_6^g = 1$, $x_7^g = 0$ mit Zielfunktionswert $z^g = 40$. Die Restkapazität beträgt $G_R = 3$ Einheiten.
Mit dem Split-Index $s = 4$ erhält man die Greedy-Split-Lösung $x_1^{gs} = x_2^{gs} = x_3^{gs} = 1$, $x_4^{gs} = x_5^{gs} = x_6^{gs} = x_7^{gs} = 0$ mit Zielfunktionswert $z^{gs} = 37$ und einer Restkapazität von 6 Einheiten.
Wegen der relativ hohen Restkapazität ist der Abstand zum optimalen Zielfunktionswert der stetigen Relaxation recht hoch. Er ergibt sich aus der Lösung $x_1^{LP} = x_2^{LP} = x_3^{LP} = 1$, $x_4^{LP} = \frac{3}{4}$, $x_5^{LP} = x_6^{LP} = x_7^{LP} = 0$ und liefert den Zielfunktionswert $z^{LP} = 44\frac{1}{2}$.
Die Optimallösung des Rucksackproblems lautet $x_1^* = x_2^* = x_3^* = 1$, $x_4^* = x_5^* = x_6^* = 0$, $x_7^* = 1$. Der optimale Zielfunktionswert beträgt $z^* = 41$ bei einer Restkapazität von nur einer Einheit.
Damit ergibt sich bei diesem Beispiel ein relative Fehler von $\frac{z^g}{z^*} = 0,9756$, oder anders ausgedrückt $\frac{z^* - z^g}{z^*} = 0,0244$.
Wendet man für dieses Beispiel den Greedy-Algorithmus bei Sortierung nur nach den Zielfunktionskoeffizienten $c_5 \geq c_3 \geq c_2 \geq c_1 \geq c_4 \geq c_7 \geq c_6$ an, dann erhält man die wesentlich schlechtere Näherung $x_5 = x_3 = 1$, $x_2 = x_1 = x_4 = x_7 = x_6 = 0$ mit Zielfunktionswert $z = 31$. Dabei ist der Rucksack voll ausgelastet.

Die Zielfunktionswerte z^{gs} und z^g der angegebenen Näherungslösungen und die Optimalwerte z^* und z^{LP} des Rucksackproblems und seiner stetigen Relaxation erfüllen die folgenden Ungleichungen:

$$z^{gs} \leq z^g \leq z^* \leq z^{LP} < z^{gs} + c_s \leq z^g + c_s$$

Damit kann für den absoluten Fehler der Greedy-Lösung als auch der Greedy-Split-Lösung die Abschätzung

$$z^* - z^g \leq z^* - z^{gs} < c_s \tag{6.5}$$

angegeben werden.

Das folgende *pathologische Beispiel* zeigt, dass die Genauigkeit h im Sinne eines relativen Näherungsalgorithmus gegen Null gehen kann und damit kein ε-optimaler Algorithmus mit Genauigkeit $\varepsilon < 1$ vorliegt.
Die Kapazität eines Rucksackes betrage $G = M$, wobei $M > 2$ eine hinreichend große Zahl sei. Für zwei zu betrachtende Gegenstände gelte $c_1 = 2$, $g_1 = 1$ und $c_2 = g_2 = M$.
Der Greedy-Algorithmus liefert die Näherung $x_1^g = 1$, $x_2^g = 0$ mit Zielfunktionswert $z^g = 2$.
Die Optimallösung lautet $x_1^* = 0$, $x_2^* = 1$ mit Optimalwert $z^* = M$.
Für $M \to \infty$ erhält man $\dfrac{z^g}{z^*} = \dfrac{2}{M} \to 0$ und damit $\dfrac{z^* - z^g}{z^*} = 1 - \dfrac{z^g}{z^*} = 1 - \dfrac{2}{M} \to 1$.
Außerdem wird auch der absolute Fehler $z^* - z^g = M - 2$ beliebig groß.

Das pathologische Beispiel gibt aber auch einen entscheidenden Hinweis: Das Einpacken eines einzelnen Gegenstandes mit hohem Nutzen, aber gleichzeitig hohem Gewicht *kann* eine Optimallösung eines konkreten Rucksackproblems darstellen.
Das Berücksichtigen allein dieser Tatsache führt bei der nachfolgenden Erweiterung des Greedy-Algorithmus zu wesentlich günstigeren Aussagen.

Erweiterter Greedy-Algorithmus für das Rucksackproblem

1. Bestimme die Greedy-Lösung x^g und deren Zielfunktionswert z^g.
2. Bestimme eine Lösung $\tilde{x}$ mit $\tilde{x}_q = 1$, $\tilde{x}_j = 0\,,\ j = 1, ..., n\,,\ j \neq q$, und $c_q = \max\limits_{j=1,...,n} c_j$.
3. Gilt $c_q \leq z^g$, dann setze $x^{eg} := x^g$ und $z^{eg} := z^g$.
   Gilt $c_q > z^q$, dann setze $x^{eg} := \tilde{x}$ und $z^{eg} := c_q$.

Der Aufwand gegenüber dem Greedy-Algorithmus erhöht sich nicht wesentlich. Zur Bestimmung des Wertes $z^{eg} = \max\{z^g, c_q\}$ muss nur der größte Zielfunktionskoeffizient gesucht werden. Dies ist in linearer Zeit möglich.

Aus (6.5) und Schritt 3 des erweiterten Greedy-Algorithmus erhält man die Abschätzung

$$z^* < z^g + c_s \leq z^g + \max_{j=1,...,n} c_j \leq z^{eg} + z^{eg} = 2z^{eg}\,.$$

Damit ergibt sich für den schlechtesten Fall eine verbesserte untere Schranke $\frac{z^{eg}}{z^*} > \frac{1}{2}$ für den relativen Fehler gegenüber dem gewöhnlichen Greedy-Algorithmus.

Satz 6.4
Der erweiterte Greedy-Algorithmus für das Rucksackproblem ist ein ε-optimaler Algorithmus mit der Genauigkeit $\varepsilon = \frac{1}{2}$.

Der Aussage des Satzes ist so zu interpretieren, dass die angegebene Fehlerschranke $\varepsilon = \frac{1}{2}$ asymptotisch angenommen wird.

Ein passendes Beispiel kleinster Dimension wird durch den Datensatz $n = 3$, $G = 2M$, $c_1 = 2$, $g_1 = 1$, $c_2 = g_2 = M$, $c_3 = g_3 = M$ realisiert, wobei $M > 2$ eine hinreichend große Zahl ist. Der erweiterte Greedy-Algorithmus wählt die durch den Greedy-Algorithmus bestimmte Näherungslösung $x_1^{eg} = x_2^{eg} = 1$, $x_3^{eg} = 0$ mit dem Zielfunktionswert $z^{eg} = M + 2$ aus. Die Optimallösung lautet $x_1^* = 0$, $x_2^* = x_3^* = 1$. Der zugehörige Optimalwert ist $z^* = 2M$. Für $M \to \infty$ kommt man der Fehlerschranke $\varepsilon = \frac{1}{2}$ von unten beliebig nahe.

Ausgangspunkt für die Konstruktion eines Näherungsschemas in Abhängigkeit von einer zu erreichenden Genauigkeit ε ist die Einbeziehung aller Teilmengen L mit maximal l Gegenständen aus der Menge $N = \{1, ..., n\}$ und die Auswertung der daraus mit geringem Aufwand erzeugbaren Näherungslösungen. Die Wahl der Zahl l hängt direkt von der Vorgabe der Fehlerschranke ε ab. Sinnvollerweise sollte $\varepsilon < \frac{1}{2}$ gewählt werden, da man für $\varepsilon = \frac{1}{2}$ mit dem erweiterte Greedy-Algorithmus bereits einen schnellen Näherungsalgorithmus kennt.

In der folgenden Konstruktion eines Näherungsschemas werden zuerst alle Teilmengen $L \subseteq N$ mit weniger als l Gegenständen betrachtet. Realisiert eine Teilmenge L eine zulässige Lösung des Rucksackproblems, dann wird sie gespeichert, falls sie den bisher besten Zielfunktionswert besitzt. Teilmengen L mit weniger als l Gegenständen werden nicht vervollständigt, auch wenn theoretisch noch Platz für weitere Gegenstände wäre.

Anschließend werden alle Teilmengen $L \subseteq N$ mit genau l Gegenständen betrachtet. Wenn durch L die Kapazität des Rucksackes noch nicht überschritten wurde, dann werden weitere noch nicht berücksichtigte Gegenstände aus einer Teilmenge der Menge $N \setminus L$ mit Hilfe des erweiterten Greedy-Algorithmus ausgewählt. Hat man auf diese Weise eine zulässige Lösung mit bisher bestem Zielfunktionswert erzeugt, so wird sie gespeichert.

Näherungsschema $A(\varepsilon)$ für das Rucksackproblem

1. Setze $l := \min\left\{\left\lceil\frac{1}{\varepsilon}\right\rceil - 2\,,\, n\right\}$, $x_j^A := 0\,,\ j \in N$, und $z^A := 0\ .$

2. Für alle Teilmengen $L \subseteq N$ mit $|L| < l$ wird folgende Anweisung ausgeführt:
   Gilt $\sum\limits_{j\in L} g_j \le G$ und $\sum\limits_{j\in L} c_j > z^A\,,$
   dann setze $x_j^A := 1\,, j \in L$, $x_j^A := 0\,, j \in N \setminus L$ und $z^A := \sum\limits_{j\in L} c_j\,.$

3. Für alle Teilmengen $L \subseteq N$ mit $|L| = l$ und $\sum\limits_{j\in L} g_j \le G$
   werden die folgenden Schritte realisiert:

   (a) Setze $G_L := G - \sum\limits_{j\in L} g_j$ und $N_L := \{j \in N \setminus L \,|\, c_j \le \min\limits_{k\in L} c_k \wedge g_j \le G_L\}\,.$
   Für $N_L = \emptyset$ wird $z^L := 0$ gesetzt und nur Schritt (c) ausgeführt.

   (b) Erzeuge mit dem erweiterten Greedy-Algorithmus für das Rucksackproblem mit der Gegenstandsmenge N_L und der Kapazität G_L eine Näherungslösung $x_j^L\,,\ j \in N_L$, und deren Zielfunktionswert $z^L\,.$

   (c) Gilt $z^L + \sum\limits_{j\in L} c_j > z^A\,,$
   dann setze $x_j^A := x_j^L\,,\ j \in N_L$, $x_j^A := 1\,,\ j \in L$, $x_j^A := 0\,,\ j \in N \setminus (L \cup N_L)$,
   und $z^A := z^L + \sum\limits_{j\in L} c_j\,.$

Die Verwendung der eingeschränkten Teilmenge N_L gegenüber der Menge $N \setminus L$ der theoretisch noch integrierbaren Gegenstände im Schritt 3(a) des Näherungsschemas verhindert im Falle der Verschiedenheit aller Zielfunktionskoeffizienten die sonst mögliche mehrfache Erzeugung einer gleichen Näherungslösung aus unterschiedlichen Mengen L. Gleichzeitig werden Gegenstände mit zu großem Gewicht hinsichtlich der Restkapazität G_L aussortiert.

Satz 6.5
Das Näherungsschema $A(\varepsilon)$ für das Rucksackproblem erzeugt für jedes $\varepsilon \in (0; \frac{1}{2})$ einen polynomialen ε-optimalen Algorithmus.

Bei einer cleveren Implementation des Näherungsschemas lässt sich der Aufwand auf die Größenordnung $\mathcal{O}(n^l)$ reduzieren ([KePfPi04]).

Das Näherungsschema $A(\varepsilon)$ ist kein *vollständig* polynomiales Näherungsschema. Der Aufwand wächst ins Unermessliche, wenn sowohl die Zahl n als auch die Zahl $\frac{1}{\varepsilon}$ wächst, das heißt wenn man für Rucksackprobleme mit sehr vielen Gegenständen immer genauere Näherungslösungen haben möchte. Dennoch gibt es vollständig polynomiale Näherungsschemata. Ihre Konstruktion ist sehr aufwendig. Auch hier sei auf [KePfPi04] verwiesen.

Einen noch vertretbaren Aufwand hinsichtlich der Rechenzeit für Rucksackprobleme mit sehr vielen Gegenständen erreicht man, wenn nur Teilmengen L mit höchstens zwei Elementen auswertet werden. Für $l = 2$ lässt sich mit den Näherungsschema $A(\varepsilon)$ ein ε-optimaler Algorithmus mit Genauigkeit $\varepsilon = \frac{1}{4}$ erreichen. Das Vorgehen des Näherungsschemas soll für diesen Fall an Hand eines kleinen Beispiels skizziert werden.

Beispiel 6.6
Gegeben sei ein Rucksackproblem mit 6 Gegenständen. Die Kapazitätsschranke betrage $G = 96$ Einheiten. Die einzelnen Bewertungen und Gewichte sind in der folgenden Tabelle bereits hinsichtlich der notwendigen Quotientensortierung angeordnet:

j	1	2	3	4	5	6
c_j	2	22	23	27	28	29
g_j	1	21	22	26	27	49

Die eindeutige Optimallösung lautet $x_1^* = 0$, $x_2^* = x_3^* = x_4^* = x_5^* = 1$, $x_6^* = 0$. Der maximale Nutzen beträgt $z^* = 100$ und der Rucksack ist voll gepackt.
Es soll jetzt eine Näherungslösung mit Fehlerschranke $\varepsilon = \frac{1}{4}$ mit Hilfe des beschriebenen Näherungsschemas bestimmt werden. Damit wird die Erzeugung einer Näherungslösung x^A mit Zielfunktionswert $z^A \geq 75$ garantiert. Die Anwendung des gewöhnlichen Greedy-Algorithmus würde die Näherungslösung $x_1^g = x_2^g = x_3^g = x_4^g = 1$, $x_5^g = x_6^g = 0$ liefern. Der Zielfunktionswert $z^g = 74$ unterschreitet die Schranke.
Die Vorgabe $\varepsilon = \frac{1}{4}$ liefert im Schritt 1 die Zahl $l = 2$. Im Schritt 2 werden deshalb lediglich zulässige Lösungen, bei denen nur ein Gegenstand eingepackt wird, hinsichtlich ihres Zielfunktionswertes verglichen. Da der letzte Gegenstand den höchsten Nutzwert besitzt, erhält man $x_1 = x_2 = x_3 = x_4 = x_5 = 0$, $x_6 = 1$ mit $z = 29$ am Ende von Schritt 2 als bisher beste Näherungslösung.
Der dritte Schritt des Näherungsschemas soll für einige wenige zweielementige Teilmengen L demonstriert werden.

- Die Menge $L = \{5, 6\}$ mit den zwei größten Nutzwerten erzeugt das Rucksackproblem mit Kapazität $G_L = 20$. Wegen zu hoher Gewichte reduziert sich die Gegenstandsmenge auf $N_L = \{1\}$. Die daraus resultierende Näherungslösung $x_1 = 1$, $x_2 = x_3 = x_4 = 0$, $x_5 = x_6 = 1$ mit $z = 59$ ist nicht besonders gut. Der Greedy-Algorithmus mit einer Sortierung nach den Zielfunktionskoeffizienten c_j würde auch genau diese Näherungslösung erzeugen.

- Die Menge $L = \{4, 6\}$ erzeugt das Rucksackproblem mit Kapazität $G_L = 21$ und Gegenstandsmenge $N_L = \{1, 2\}$. Der erweiterte Greedy-Algorithmus wählt nicht die Greedy-Lösung, sondern den einzelnen Gegenstand mit Nummer 2 aus. Die erzeugte Näherung $x_1 = 0$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$, $x_5 = 0$, $x_6 = 1$ besitzt den Zielfunktionswert $z = 78$. und würde bereits die Vorgabe erfüllen.

- Für $L = \{2, 3\}$ erhält man die Kapazität $G_L = 53$. Die mögliche Gegenstandsmenge $N \setminus L = \{1, 4, 5, 6\}$ wird wegen der Forderung $c_j \leq \min\{22, 23\}$ auf die Menge $N_L = \{1\}$ reduziert. Damit wird $x_1 = x_2 = x_3 = 1$, $x_4 = x_5 = x_6 = 0$ als Näherungslösung mit $z = 47$ erzeugt. Ohne die Einschränkung würde die Menge $N \setminus L$ die Näherungslösung $x_1 = x_2 = x_3 = x_4 = 1$, $x_5 = x_6 = 0$ mit $z = 74$ liefern. Diese Näherungslösung erhält man aber auch aus der Menge $L = \{3, 4\}$. Mit der vorgenommenen Einschränkung auf die Menge N_L wird hier ein mehrfaches Erzeugen dieser Näherungslösung vermieden und der Aufwand hinsichtlich der Größe der zu betrachtenden Teilprobleme gesenkt.
- Für $L = \{1, 6\}$ erhält man $G_L = 46$, aber $N_L = \emptyset$. Der erweiterte Greedy-Algorithmus wird nicht aufgerufen. Mit $x_1 = 1$, $x_2 = x_3 = x_4 = x_5 = 0$, $x_6 = 1$ entsteht eine unbrauchbare Näherungslösung mit $z = 31$.
- Die Menge $L = \{4, 5\}$ erzeugt das Rucksackproblem mit Kapazität $G_L = 43$ und Gegenstandsmenge $N_L = \{1, 2, 3\}$. Der erweiterte Greedy-Algorithmus wählt die Greedy-Lösung, bestehend aus den ersten beiden Gegenständen aus. Die erzeugte Näherungslösung $x_1^A = x_2^A = 1$, $x_3^A = 0$, $x_4^A = x_5^A = 1$, $x_6^A = 0$ liefert für das Näherungsschema mit $z^A = 79$ den größten Zielfunktionswert.

Das vorliegende Beispiel hat erkennbar eine problematische Struktur. Die nicht mehr nutzbare Restkapazität aller im Laufe des Verfahrens erzeugten akzeptierbaren Näherungslösungen ist recht hoch und die Genauigkeitsschranke $\varepsilon = \frac{1}{4}$ wird nur unwesentlich unterschritten. Erst mit $l = 3$ würde auch die Optimallösung als Näherungslösung erzeugt.

Literaturverzeichnis

[BoGr93] I.M. Bomze und W. Grossmann: Optimierung - Theorie und Algorithmen, BI-Wissenschaftsverlag, Mannheim Leipzig Wien Zürich, 1993

[Bu72] R.E. Burkard: Methoden der Ganzzahligen Optimierung, Springer Verlag, Wien, New York, 1972

[Co71] S.A. Cook: The complexity of theorem proving procedures, Proceedings of the 3rd Annual ACM Symposium on the theory of Computing, 1971, 151-158

[Ed65] J. Edmonds: Minimum partition of the matroid into independent subsets, Journal of Research of the National Bureau of Standards, B69(1965), 67-72

[GaJo79] M.R. Garey and D.S. Johnson: Computers and Intractability: A Guide to the Theory of *NP-Completeness*, Freeman, San Francisco, 1979

[Go58] R.E. Gomory: Outline of an Algorithm for Integer Solutions to Linear Programs, Bull. Amer. Math. Soc. 64, 275-278, 1958

[Iba87] T. Ibaraki: Enumerative Approaches to Combinatorial Optimization I. II., Annals of Operations Research 10,11, 1987

[Kar75] R.M. Karp: On the complexity of combinatorial problems, Networks, 5(1975), 45-68

[KePfPi04] H. Kellerer, U. Pferschy and D. Pisinger: Knapsack Problems, Springer Verlag, Berlin Heidelberg New York, 2004

[KoVy00] B. Korte and J. Vygen: Combinatorial Optimization, Theory and Algorithms, Springer Verlag, Berlin Heidelberg New York, 2000

[Lan02] H.W. Lang: Algorithmen in Java, R. Oldenbourg Verlag, 2002

[Lad75] R.E. Ladner: On the structure of polynomial time reducibility, Journal of the ACM, 22(1975), 155-171

[MaTo90] S. Martello, P. Toth: Knapsack Problems, John Wiley & Sons, New York Chichester Brisbane Toronto Singapore, 1990

[NeWo88] G.L. Nemhauser and L.A. Wolsey: Integer and Combinatorial Optimization, John Wiley & Sons, New York Chichester Brisbane Toronto Singapore, 1988

[PaSt82] C.H. Papadimitriou and K. Steiglitz: Combinatorial Optimization: Algorithms and Complexity,
Prentice-Hall, Englewood Cliffs, 1982

[Pie70] J. Piehler: Ganzzahlige lineare Optimierung,
Teubner Verlagsgesellschaft, 1970

[Scho76] M. Schoch: Das Erweiterungsprinzip und seine Anwendung zur Entwicklung von Algorithmen für die Lösung kombinatorischer Optimierungsaufgaben,
Deutscher Verlag der Wissenschaften, 1976

[Schr03] A. Schrijver: Combinatorial Optimization - Polyhedra and Efficiency,
Springer-Verlag, Berlin, 2003

[Tur36] A.M. Turing: On computable numbers, with an application to the Entscheidungsproblem,
Proceedings of the London Mathematical Society, 42(1936), 2, 230-265

Freiberger Graduierungsarbeiten:

- Werner Lyska:
  Kombinatorische Algorithmen zur Lösung linearer 0-1-Optimierungsaufgaben
  Dissertation, 1976

- Stephan Dempe:
  Polynomiale Näherungsalgorithmen für ein $\mathcal{NP}$-schwieriges lineares gemischt-ganzzahliges Optimierungsproblem spezieller Struktur
  Dissertation (Chemnitz), 1982

- Petra Müller:
  Untersuchung einer allgemeinen linearen gemischt-ganzzahligen 0-1-Optimierungsaufgabe mit einer Nebenbedingung
  Dissertation, 1986

- Elke Lehmann:
  Ein mathematisches Modell für die Planung prophylaktischer Instandhaltungsmaßnahmen an Energieanlagen unter Berücksichtigung von Resourcenbeschränkungen - Lösungsverfahren für ganzzahlige Optimierungsaufgaben spezieller Struktur
  Dissertation, 1989

- Jens Klopfer:
  Verfahren zur Lösung eines Tourenproblems der diskreten Schütgutoptimierung
  Diplomarbeit, Oktober 1995

- Mario Fritsch:
  Untersuchungen zur quadratischen 0-1-Optimierung mit Methoden des Simulated Annealing
  Diplomarbeit, April 1996

- Jörg Wengler:
  Heuristiken zur Lösung eines Tourenproblems der diskreten Schütgutoptimierung
  Diplomarbeit, Oktober 1996

- Kathrin Kempe:
  Untersuchungen zu diskontinuierlichen Transportproblemen
  Diplomarbeit, November 1997

- Steffen G. Meusel:
  Minimizing the Placement-Time on Printed Circuit Boards
  Dissertation, Februar 1998

- Andreas Herrmann:
  Zwei-kriterielle diskrete Optimierungsaufgaben
  Diplomarbeit, Mai 1999

- Frank Fuhren:
  Moderne Heuristiken für das Rundreiseproblem
  Diplomarbeit, Mai 2000

- Viktor Jakobi:
  Heuristische Verfahren für Rundreiseprobleme
  Bakkalaureusarbeit, Januar 2004

- Sylvia Chares:
  Untersuchungen zu einem Standardisierungsproblem
  Bakkalaureusarbeit, August 2004

- Anne Heftenberger:
  Diskontinuierliche Versorgungsprobleme mit Min-Max-Zielfunktionen
  Bakkalaureusarbeit, Dezember 2004

- Frank Martin:
  Diskrete Zwei-Ebenen-Optimierung
  Bakkalaureusarbeit, Januar 2005

- Susann Trommler:
  Auswertung von Näherungsverfahren für Rucksackprobleme
  Bakkalaureusarbeit, Januar 2007

- Quingfeng Meng:
  Näherungsverfahren zur Lösung des flexiblen für Rucksackproblems
  Diplomarbeit, Juni 2008

- Stefanie Melcher:
  Untersuchungen zu einem speziellen Standortauswahlproblem
  Bakkalaureusarbeit, November 2010