Zweidimensionale Projektionen von linearen Programmen

Diplomarbeit
von
Stefan Fischer

eingereicht
beim Fachbereich Mathematik
der Technischen Universität Berlin

Februar 1998

Erstgutachter: Prof. G. Ziegler

Zweitgutachter: Prof. M. Grötschel

Inhaltsverzeichnis

1 Einleitung

Ziel dieser Arbeit ist es, zu zeigen wie die Polyeder von linearen Programmen auf eine zweidimensionale Ebene projiziert und dann anschließend graphisch darstellt werden können.

Für die Berechnung der Projektionen der linearen Programme wird der Schatteneckenalgorithmus (3.2) implementiert. Er ist eine Variante des Simplex-Verfahrens (3.1).

Im Unterschied zu linearen Programmen $\max c^T x, x \in P, P$ Polyeder, betrachtet der Schatteneckenalgorithmus noch eine weitere Zielfunktion $\overline{c}$. Sind c und $\overline{c}$ linear unabhängig, so spannt $\text{span}(c,\overline{c})$ eine Ebene auf. Auf diese Ebene projiziert man nun orthogonal das zum linearen Programm gehörende Polyeder P.

Ecken von P, die ebenfalls Ecken der Projektion sind, werden Schattenecken genannt.

Der Schatteneckenalgorithmus arbeitet wie folgt. Angenommen die beiden Zielfunktionen besitzen optimale Lösungen auf P und das lineare Programm sei nicht degeneriert. Dann startet der Schatteneckenalgorithmus bei einer Schattenecke, die optimal bzgl. $\overline{c}$ ist und wandert über Schattenecken zur optimalen Ecke bzgl. c.

Da die Schattenecken die charakteristischen Punkte der Projektionen sind, muß man alle Schattenecken berechnen, um die Projektionen darstellen zu können. Mit Hilfe des Schatteneckenalgorithmus kann man alle Schattenecken, sprich alle Ecken der Projektion, berechnen.

Trägt man die berechneten Schattenecken nun in ein Koordinatensystem mit c und $\overline{c}$ als Koordinatenachsen ein und verbindet diese Punkte nacheinander, so erhält man eine graphische Darstellung der Pojektion des linearen Programms.

In Kapitel 2 werden die theoretischen Grundlagen vorgestellt. Es wird erklärt, welche Gestalt ein Polyeder P besitzt und welche interessanten Teilmengen von P existieren.

Anschließend befassen wir uns mit dem Bezug von Polyedern zu linearen Programmen, bevor wir schließlich die Projektion von Polyedern und linearen Programmen definieren.

In Kapitel 3 wird dann die Funktionsweise des Schatteneckenalgorithmus ausführlich behandelt, bevor in Kapitel 4 darauf eingegangen wird, wie man die Projektion der linearen Programme mit Hilfe des Schatteneckenalgorithmus genau berechnen kann.

In Kapitel 5 wird anhand eines Beispielproblems die Funktionsweise des Algorithmus veranschaulicht. Hier soll gezeigt werden, inwieweit unterschiedliche Zielfunktionen Einfluß auf die Projektion nehmen.

Kapitel 6 beschäftigt sich dann mit der Implementierung des Algorithmus. Hier werden die wichtigsten Datenstrukturen und Hilfsprogramme vorgestellt.

Zum Abschluß der Arbeit werden in Kapitel 7 praktische Probleme ausgewertet. Es werden die graphischen Darstellungen der Projektionen der Probleme aufgezeigt. Hierbei sieht man, daß die Struktur der praktischen Probleme sehr verschieden sein kann.

Es gibt sowohl einfach strukturierte Probelme mit nur wenigen Schattenecken als auch Probleme mit vielen Schattenecken.

Grob kann man die Projektionen in vier Kategorien einteilen, in spitze, runde, rechteckige und unbeschränkte Projektionen.

Leider standen mir keine Probleme mit zwei Zielfunktionen zur Verfügung, so daß die zweite Zielfunktion auf verschiedene Weisen konstruiert wurde. Dabei sieht man, daß das Aussehen der Projektion in direktem Zusammenhang mit der Wahl der zweiten Zielfunktion steht.

2 Grundlagen

In diesem Kapitel werden die zu lösenden Probleme vorgestellt und notwendige Definitionen getroffen. Es dient zur Einführung in die zu behandelnde Problematik und zur Erläuterung des theoretischen Hintergrunds.

Das Verständnis über Gleichungs- und Ungleichungssysteme und den $\mathbb{R}^n$ wird vorausgesetzt.

Begonnen wird mit der Einführung des Polyeders. Anschließend werden die Seitenflächen von Polyedern, speziell die Ecken, definiert. Danach wird die Projektion von Polyedern eingeführt. Zum Abschluß des Kapitels werden dann der Begriff des linearen Programms und der Bezug zu den Polyedern, speziell den Ecken der Polyeder, erläutert.

2.1 Polyeder

Eine wichtige Teilmenge des $\mathbb{R}^n$ sind die Polyeder. Deshalb beschäftigen wir uns zuerst mit ihnen. Bevor wir den Begriff des Polyeders einführen definieren wir den Halbraum.

Definition Halbraum: Eine Teilmenge $H \subseteq \mathbb{R}^n$ heißt *Halbraum*, falls es ein $a \in \mathbb{R}^n \backslash \{0\}$ und ein $\alpha \in \mathbb{R}$ gibt, mit

$$H = \{x \in \mathbb{R}^n \mid a^T x \leq \alpha\}.$$

Ein Polyeder wird nun definiert durch die folgende Menge.

Definition Polyeder: Eine Teilmenge $P \subseteq \mathbb{R}^n$ heißt *Polyeder*, falls es ein $m \in \mathbb{N}$, eine Matrix $A \in \mathbb{R}^{m \times n}$ und einen Vektor $b \in \mathbb{R}^m$ gibt, mit

$$P = P(A, b) = \{x \in \mathbb{R}^n \mid Ax \leq b\}.$$

Wie man leicht erkennt, ist ein Polyeder P der Schnitt von endlich vielen Halbräumen, falls für das Polyeder P gilt $P \neq \mathbb{R}^n$.

Die leere Menge und der gesamte $\mathbb{R}^n$ sind ebenfalls Polyeder. Wenn das Polyeder beschränkt ist, d.h. wenn $P \subseteq \{x \in \mathbb{R}^n \mid \|x\| \leq S\}$, für ein $S > 0$, gilt, dann nennt man es *Polytop*.

Ein spezielles Polyeder ist das *Gleichheitspolyeder*. Es ist definiert durch

$$P^{=}(A,b) = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\},$$

mit $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$.

Polyeder können auch durch Gleichungen und Ungleichungen charakterisiet werden, so ist die Lösungsmenge von

$$\begin{aligned} Ax + By &= a \\ Cx + Dy &\leq b \\ x &\geq 0, \end{aligned}$$

mit A, B, C, D, a, b, x, y dimensionsverträglich, ein Polyeder, denn die Lösungsmenge ist identisch zu der Lösungsmenge von

$$\begin{pmatrix} A & B \\ -A & -B \\ C & D \\ -I & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} a \\ -a \\ b \\ 0 \end{pmatrix}.$$

Für die später betrachtete Problematik ist es wichtig zu wissen, wie man Ungleichungssysteme in Gleichungssysteme und nicht vorzeichenbeschränkte Variablen in vorzeichenbeschränkte Variablen transformieren kann.

Erweitert man ein Ungleichungssystem

1. $Ax \leq b$ durch das Einfügen sogenannter *Schlupfvariablen* $y \geq 0$ zu einem System der Form

2. $Ax + Iy = b, y \geq 0$, I Einheitsmatrix, so erhält man folgende Beziehung

Wenn x (1.) erfüllt , dann erfüllt $\binom{x}{y}$ (2.) mit $y = b - Ax$.

Wenn $\binom{x}{y}$ (2.) erfüllt, dann erfüllt x (1.).

Somit kann man aus jedem Ungleichungssystem ein Gleichungssytem erstellen. Zu beachten ist hierbei, daß das Ungleichungssystem und das Gleichungssystem hier zwei verschiedene Polyeder beschreiben.

Nun soll noch gezeigt werden, wie nicht vorzeichenbeschränkte Variablen durch vorzeichenbeschränkte Variablen dargestellt werden können.

Man ersetzt die nicht vorzeichenbeschränkte Variable x durch zwei vorzeichenbeschränkte Variablen $x^+, x^- \geq 0$ indem man x durch $x = x^+ - x^-$ darstellt.

Mit diesen beiden Transformationen kann jedes Polyeder $P(A,b)$ in ein Polyeder $P^{=}(\overline{A},\overline{b})$ transformiert werden, wobei das neue Polyeder nicht in demselben Vektorraum liegt wie das alte.

Eine weitere Darstellungsmöglichkeit für Polyeder soll noch angegeben werden.

Definition: $\sum_{i=1}^{k} \alpha_i x_i$ heißt *Konvexkombination*, wenn $\alpha \geq 0$ und $\alpha^T \mathbf{1} = 1$ gilt. Die Summe nennt man eine *konische Kombination*, wenn $\alpha \geq 0$ gilt.

Sei $A \subseteq \mathbb{R}^n$ eine nichtleere Teilmenge. Dann heißt $\text{conv}(A)$ konvexe Hülle von A und $\text{cone}(A)$ konische Hülle von A, d.h. die Menge aller Vektoren, die sich aus einer endlichen Anzahl von Vektoren aus A als konvexe bzw. konische Kombination darstellen lassen.

Mit den eingeführten Definitionen kann man ein Polyeder auch schreiben durch

Satz: Eine Teilmenge $P \subseteq \mathbb{R}^n$ ist genau dann ein Polyeder, wenn endliche Mengen $V, E \subseteq \mathbb{R}^n$ existieren, mit

$$P(A,b) = \text{conv}(V) + \text{cone}(E).$$

Die Darstellung des Polyeders durch $P(A,b)$ nennt man auch *äußere Beschreibung*, während man die Darstellung durch $\text{conv}(V) + \text{cone}(E)$ *innere Beschreibung* nennt.

2.2 Seitenflächen von Polyedern

Nachdem wir die Polyeder eingeführt haben, interessieren uns bestimmte Teilmengen der Polyeder, die sogenannten Seitenflächen.

Für die Definition einer Seitenflche muß noch erklärt werden, was man unter einer gültigen Ungleichung versteht.

Definition gültige Ungleichung: Sei $P = P(A,b)$ ein Polyeder. Eine Ungleichung $a^T x \leq \alpha$ heißt *gültig* bezüglich P, falls P eine Teilmenge des zugehörigen Halbraums der Ungleichung ist, also $P \subseteq \{x \in \mathbb{R}^n | a^T x \leq \alpha\}$ gilt.

Die Seitenflächen der Polyeder sind nun definiert durch

Definition Seitenflächen: Sei $P \subseteq \mathbb{R}^n$ ein Polyeder. Eine Menge $F \subseteq P$ heißt *Seitenfläche* von P, wenn es eine gültige Ungleichung $a^T x \leq \alpha$ bezüglich P gibt mit

$$F = P \cap \{x \in \mathbb{R}^n \mid a^T x = \alpha\}.$$

Gilt $F = \{x\}$, so nennt man die Seitenfläche eine *Ecke*. Eine eindimensionale Seitenfläche nennt man eine *Kante*. Gilt $F = x + \text{ cone } (\{z\})$, so nennt man die Seitenfläche *Extremalstrahl*, bei $F = x + \text{ lin } (\{z\})$ nennt man sie *Extremallinie*.

Eine Seitenfläche F heißt *echt*, wenn $F \neq P$ gilt, und sie heißt *nichttrivial*, wenn $\emptyset \neq F \neq P$ gilt. Außerdem ist jede Seitenfläche eines Polyeders wiederum ein Polyeder.

Die Anzahl der Seitenflächen eines Polyeders $P = P(A, b)$, mit $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, ist nach oben beschränkt durch $2^m + 1$.

Sind zwei Ecken a und b des Polyeders durch eine Kante miteinander verbunden, so sagt man entweder a und b sind *adjazent* zueinander oder man sagt a und b sind *benachbart*.

Nicht alle Polyeder müssen Ecken als Seitenflächen haben. Polyeder, die Ecken als Seitenflächen besitzen, werden *spitze Polyeder* genannt. $P^{=}(A, b)$ ist z.B. ein spitzes Polyeder.

Ein Polyeder P ist genau dann spitz, wenn auch jede nichtleere Seitenfläche des Polyeders spitz ist.

2.3 Lineare Programme

Die Probleme, die betrachtet werden, sind Optimierungsprobleme über Polyedern mit linearer Zielfunktion. Sie werden auch lineare Programme genannt.

Definition lineares Programm: Ein Problem der Form

$$\begin{aligned} \max \quad & c^T x && (1) \\ s.t. \quad & Ax \leq b, && (2) \\ & c, x \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, \end{aligned}$$

nennt man *lineares Programm*.

Wie man sieht, besteht ein lineares Programm aus zwei Teilen. Es besitzt eine lineare Zielfunktion und eine Menge, über der die Zielfunktion optimiert

werden soll. Diese Menge ist ein Polyeder. Lineare Programme optimieren also lineare Zielfunktionen über Polyedern.

Definition zulässiger Punkt: Ein Punkt $x_0 \in \mathbb{R}^n$ eines linearen Programms heißt *zulässig*, falls alle Ungleichungen (2) für x_0 erfüllt sind.

Ziel ist es also, einen bezüglich (2) zulässigen Punkt $\overline{x}$ zu finden mit

$$c^T\overline{x} = \max\{c^T x \mid x \text{ zulässig bezüglich (2) }\}$$

oder festzustellen, daß kein solcher Punkt $\overline{x}$ existiert, d.h. festzustellen, daß (1) über (2) unbeschränkt ist oder keine zulässige Lösung existiert.

Wichtig für die Vorgehensweise beim Lösen linearer Programme ist der folgende Satz.

Satz: Sei $P \subseteq \mathbb{R}^n$ ein spitzes Polyeder und habe das lineare Programm $\max c^T x$, $x \in P$ eine zulässige Optimallösung $\overline{x}$, dann hat das lineare Programm auch eine optimale Lösung x_0, mit x_0 ist eine Ecke von P.

In Kapitel 3 wird gezeigt, daß über spitzen Polyedern optimiert wird, die Polyeder also eine optimale Ecklösung besitzen, falls das Problem nicht unbeschränkt oder leer ist.

Ziel wird es dann sein, möglichst schnell diese optimale Ecke zu bestimmen. Wie dies realisiert wird, wird in Kapitel 3 beschrieben.

2.4 Projektion von linearen Programmen

Die linearen Programme sollen schließlich noch auf eine zweidimensionale Ebene projiziert werden. Bevor wir die Projektion von linearen Programmen definieren, beschäftigen wir uns kurz mit der Projektion von Polyedern. Unter der Projektion von Polyedern versteht man folgendes.

Definition Projektion von Polyedern: Gegeben sei ein Polyeder $P \subseteq \mathbb{R}^n$, eine Menge $E \subseteq \mathbb{R}^n$ und ein Vektor $c \in \mathbb{R}^n \backslash \{0\}$. Die Menge

$$\{x \in \mathbb{R}^n \mid x \in E, \exists\, \lambda \in \mathbb{R} \text{ mit } x + \lambda c \in P\}$$

heißt *Projektion* von P entlang c auf E.

Anschaulich kann die Projektion des Polyeders P auf E folgendermaßen beschrieben werden.

Man läßt zu c paralleles Licht aus dem Unendlichen auf E scheinen. Alle Punkte auf E durch die ein Strahl geht, der auch durch P geht, werden markiert. Die daraus resultierende Menge der markierten Punkte ist dann die Projektion von P auf E.

Während hier die Menge E eine beliebige Teilmenge des $\mathbb{R}^n$ ist, betrachten wir bei der Projektion von linearen Programmen nur Projektionen auf zweidimensionale Ebenen. Unter der Projektion von linearen Programmen versteht man

Definition Projektion von linearen Programmen: Sei $P \subseteq \mathbb{R}^n$ ein Polyeder und $\max c^T x$, $x \in P$ ein linares Programm. Sei $\overline{c} \in \mathbb{R}^n$ ein Vektor mit $\overline{c} \neq \lambda c$, f.a. $\lambda \in \mathbb{R}$. Außerdem gelte $c, \overline{c} \neq 0$. Setze $E := \text{span}(c, \overline{c})$ und sei $n_0 \in \mathbb{R}^n$ ein Normalenvektor auf E.

Dann heißt die Menge

$$\{x \in \mathbb{R}^n \mid x \in E, \exists\, \lambda \in \mathbb{R} \text{ mit } x + \lambda n_0 \in P\}$$

Projektion des linearen Programms entlang n_0 auf E.

Wie man leicht sieht unterscheidet sich die Projektion von linearen Programmen zu der Projektion von Polyedern nur darin, daß E zweidimensional ist und durch die Zielfunktion schon zum Teil vorgegeben ist, und daß die Pojektionsrichtung orthogonal zu E ist.

Wie die Projektion berechnet und dargestellt wird, wird in Kapitel 4 eingeführt.

3 Lösungsverfahren für lineare Programme

In diesem Kapitel wird der Schatteneckenalgorithmus vorgestellt, mit dessen Hilfe die linearen Programme gelöst und schließlich die Projektionen der linearen Programme berechnet werden sollen.

Bevor der Schatteneckenalgorithmus jedoch erklärt wird, soll erst einmal allgemein die Grundversion des Simplex-Verfahrens erläutert werden, denn der Schatteneckenalgorithmus ist eine Variante des Simplex-Verfahrens.

3.1 Simplex-Verfahren

Die Grundversion des Simplex-Verfahrens löst Probleme der Form

$$\begin{aligned} \max \quad & c^T x && (3)\\ s.t. \quad & Ax = b, x \geq 0 \text{ mit} && (4)\\ & A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, b \geq 0 \text{ und} \\ & c, x \in \mathbb{R}^n, \text{ mit } m < n. \end{aligned}$$

Wie man leicht sieht, kann das lineare Programm (1) durch das Einfügen von Schlupfvariablen $s_1, ..., s_m$ und durch Aufteilung der Variablen x in vorzeichenbeschränkte Variablen x^+ und x^- mit $x = x^+ - x^-$ in die gewünschte Form

$$\begin{aligned} \max \quad & c^T(x^+ - x^-) \\ s.t. \quad & A(x^+ - x^-) + Is = b, x^+, x^-, s \geq 0 \text{ mit} \\ & I \in \mathbb{R}^{m \times m} \text{ Einheitsmatrix,} \end{aligned}$$

gebracht werden.

Falls einige Komponenten von b noch negativ sind, werden die entsprechenden Gleichungen mit -1 multipliziert.

Durch diese Transformation wurde erreicht, daß zu jeder Optimallösung von (3) eine entsprechende Optimallösung von (1) mit gleichem Zielfunktionswert existiert und umgekehrt. Wenn eines der Probleme unbeschränkt oder unlösbar ist, so ist es auch das andere.

In [2] wird folgender Satz bewiesen

> **Satz:** Ist das Polyeder P spitz und hat das lineare Programm $\max c^T x, x \in P$ eine Optimallösung, so hat dieses lineare Programm auch eine Optimallösung, die eine Ecke ist. Man spricht dann von einer *optimalen Ecklösung.*

Da nichtleere Polyeder der Form $P^{=}(A,b)$ spitze Polyeder sind, hat jedes hier zu lösende Problem, das eine Optimallösung besitzt, ebenfalls eine optimale Ecklösung.

Diese Eigenschaft spielt eine entscheidende Rolle. Eigentlich müßte man jetzt alle Ecken des Polyeders und die zugehörigen Zielfunktionswerte berechnen und eine Ecke mit der besten Lösung auswählen. Dann wäre das Problem optimal gelöst.

Da ein Polyeder im allgemeinen aber exponentiell viele Ecken besitzen kann, ist dies keine elegante Variante. Wie das Problem geschickter gelöst werden kann, soll nun erläutert werden.

Um die Ecken programmtechnisch besser behandeln zu können, wird der Begriff der Basis eingeführt und einige Definitionen getroffen.

Definitionen: Gegeben seien $Ax = b$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m, b \geq 0$, $rang(A) = m$.

1. $\{1, ..., m\}$ bezeichnet die *Zeilenindexmenge*, $\{1, ..., n\}$ die *Spaltenindexmenge* von A.

2. B und N bezeichnen *Spaltenindexvektoren.* Falls die Reihenfolge der Elemente in B und N unerheblich ist, werden B und N auch einfach als Mengen aufgefaßt. B und N haben folgende Eigenschaft: $B \cap N = \emptyset$ und $B \cup N = \{1, ..., n\}$.

3. Anstatt $A_{.B}$ und $A_{.N}$ wird abkürzend A_B und A_N geschrieben. Ist A_B regulär, so heißt A_B Basismatrix und A_N Nichtbasismatrix von A. Ein Vektor x mit $x_B = A_B^{-1}b$ und $x_N = 0$ heißt *Basislösung* von A.

4. x mit $x_B = A_B^{-1}b$ und $x_N = 0$ heißt *zulässig*, falls $x_B \geq 0$ gilt, B heißt dann *zulässige Basis* und A_B *zulässige Basismatrix.*

5. Im weiteren bezeichne n die Anzahl der Variablen und m die Anzahl der Gleichungen bzw. Ungleichungen des gegebenen linearen Programms.

Die wichtige Beziehung zwischen Ecken und Basis beinhaltet der folgende Satz.

Satz: Seien $P = P^{=}(A,b) \subseteq \mathbb{R}^n$ ein Polyeder mit $rang(A) = m < n$ und $x \in P$. Dann ist x genau dann eine Ecke von P, wenn x eine zulässige Basislösung ist.

Jetzt kann man das Problem, alle Ecken zu enumerieren, durch Enumeration der Basen lösen. Hier treten allerdings dieselben rechnerischen Probleme auf wie zuvor.

Ziel ist es nun, ein Verfahren zu entwickeln, daß bei einer zulässigen Basislösung beginnt und dann eine neue zulässige Basislösung bestimmt, deren Zielfunktionswert besser ist.

Mit dem besseren Zielfunktionswert der neuen Basislösung will man erreichen, daß der Algorithmus nicht über alle Ecken des zu dem linearen Programm gehörenden Polyeders zur optimalen Ecke laufen muß sondern möglichst schnell die optimale Ecke erreicht, falls denn eine existiert.

Es gibt aber trotzdem Beispiele, bei dem der Algorithmus alle Ecken des Polyeders besuchen muß.

Wie man von einer Basislösung zur nächsten gelangt, sprich einen *Basisaustausch* vornimmt, sagt folgender Satz.

Satz Basisaustausch: Gegeben seien $Ax = b$, $b \geq 0$, $rang(A) = m$, $B = (p_1, ..., p_m)$ und $N = (n_1, ..., n_{n-m})$ seien Spaltenindexvektoren von A, so daß A_B eine Basismatrix ist. Setze

$$\begin{aligned} \overline{A} &:= A_B^{-1} A_N (= (\overline{a}_{rs})_{\substack{1 \leq r \leq m \\ 1 \leq s \leq n-m}}) \text{ und} \\ \overline{b} &:= A_B^{-1} b. \end{aligned}$$

Ist $\overline{a}_{rs} \neq 0$, so ist $A_{B'}$, mit $B' = (p_1, ..., p_{r-1}, n_s, p_{r+1}, ..., p_m)$, eine Basismatrix von A.

Die neue Basislösung $\overline{x}$ zu $A_{B'}$ kann aus der alten Basislösung x zu A_B auf folgende Weise berechnet werden.

$$\begin{aligned} \overline{x}_{p_i} &= \overline{b}_i - \frac{\overline{a}_{is}}{\overline{a}_{rs}} \overline{b}_r, i = 1..m, i \neq r \\ \overline{x}_{p_r} &= \frac{1}{\overline{a}_{rs}} \overline{b}_r \\ \overline{x}_i &= 0 \text{ andernfalls.} \end{aligned}$$

Um nicht alle Basen enumerieren zu müssen, soll der Zielfunktionswert bei jedem Basisaustausch möglichst verbessert, auf keinen Fall aber verschlechtert werden. Dies wird sichergestellt, wenn man folgende Tatsache berücksichtigt.

Satz Basisverbesserung: Gegeben sei ein lineares Programm (3). Sei A_B eine zulässige Basismatrix, x die zugehörige Basislösung, $\overline{A} = A_B^{-1}A_N$, $\overline{b} = A_B^{-1}b$ und $c_{red}^T = c_N^T - c_B^T A_B^{-1} A_N$ seien die *reduzierten Kosten*. Sei $n_s \in N$ ein Index mit $c_{red}[s] > 0$.

Dann gilt

1. Gilt $\overline{A}_{.s} \leq 0$ dann ist das zu lösenden Problem unbeschränkt.
2. Gilt $\overline{A}_{.s} \not\leq 0$ dann setze

$$\lambda_0 = \min\left\{\frac{\overline{b}_i}{\overline{a}_{is}} \mid \overline{a}_{is} > 0, 1 \leq i \leq m\right\}.$$

   Jetzt ist $A_{B'}$, mit $B' = \{b_1, ..., b_{r-1}, n_s, b_{r+1}, ..., b_m\}$ und mit $r \in \{i \mid i \in \{1, ..., m\}, \frac{\overline{b}_i}{\overline{a}_{is}} = \lambda_0\}$, eine zulässige Basismatrix mit Basislösung $\overline{x}$ und $c^T\overline{x} \geq c^T x$.
3. Ist das Problem nichtdegeneriert, so wird unter denselben Voraussetzungen wie oben aus $c^T\overline{x} \geq c^T x$ ein echt größer $c^T\overline{x} > c^T x$.

Um zu wissen, ob die aktuelle Ecklösung optimal ist, muß noch das Kriterium vorgestellt werden, das nachweist, daß es keine Ecklösung gibt, die einen besseren Zielfunktionswert liefert.

Als Optimalitätskriterium dienen die reduzierten Kosten. Nachdem die reduzierten Kosten bereits erwähnt wurden, sollen sie noch etwas genauer definiert werden.

Definition reduzierte Kosten: Gegeben sei ein lineares Programm der Form (3) und A_B sei eine zulässige Basismatrix von A. Dann kann man den Zielfunktionswert $c^T x$ schreiben als

$$c^T x = c_B^T A_B^{-1} b + (c_N^T - c_B^T A_B^{-1} A_N) x_N.$$

Der Term $c_N^T - c_B^T A_B^{-1} A_N$ heißt *reduzierte Kosten.*

Beim Basisaustausch wählen wir eine Nichtbasisvariable aus, deren reduzierter Kostenkoeffizient positiv ist. Nimmt man diese Variable in die Basis, so wird der Zielfunktionswert erhöht, außer es wird ein degenerierter Pivotschritt durchgeführt, sprich eine Null gegen eine Null getauscht. Aus dieser Eigenschaft läßt sich ein Optimalitätskriterium ableiten.

Satz: Gegeben sei ein lineares Programm der Form (3). Sei A_B eine zulässige Basismatrix.

1. Gilt dann für die reduzierten Kosten $c_N^T - c_B^T A_B^{-1} A_N \leq 0$, so ist die zugehörige Basislösung optimal für das lineare Programm.
2. Ist die zugehörige Basislösung nicht degeneriert und optimal, dann sind die reduzierten Kosten kleiner gleich Null.

Das Problem hierbei ist, daß wir im degenerierten Fall eine Opimallösung berechnet haben können, ohne zu wissen, daß die Lösung bereits optimal ist. In diesem Fall müssen wir weitere Basiswechsel vollziehen, bis wir wissen, daß wir optimal sind.

Da eine Ecke durch mehrere Basen dargestellt werden kann, kann es im degenerierten Fall vorkommen, daß das Programm *kreiselt*, sprich nur Basen berechnet, die eine Ecke beschreiben. In diesem Fall endet die Grundversion des Simplex-Verfahrens nicht. Man kann aber durch bestimmte Auswahlverfahren beim Tausch der Nichtbasisvariable mit einer Basisvariable die Endlichkeit des Algorithmus erreichen.

Die Auswahl einer Nichtbasisvariable, die in die Basis gelangen soll, nennt man *Wahl der Pivotspalte* und die Auswahl der Basisvariable, die die Basis verlassen soll, *Wahl der Pivotzeile.*

Jetzt sollen einige Möglichkeiten aufgezeigt werden, wie man die Pivotspalte bzw. die Pivotzeile auswählen und dadurch die Endlichkeit des Simplex-Verfahrens erreichen kann.

3.1.1 Wahl der Pivotspalte

1. *Kleinster Index Regel:* Wähle Pivotspalte s mit

$$s = \min\{i \mid n_i \text{ Nichtbasisvariable und } c_{red}[i] > 0\}$$

2. *Kleinster Variablenindex Regel:* Wähle Pivotspalte s mit

$$n_s = \min\{n_i \text{ Nichtbasisvariable} \mid c_{red}[i] > 0\}$$

3.1.2 Wahl der Pivotzeile

Sei s die ausgewählte Pivotspalte und
$R = \{r \mid r \in \{1, .., m\} \text{ und } \frac{\overline{b}_r}{\overline{a}_{rs}} = \min\{\frac{\overline{b}_i}{\overline{a}_{is}} \mid \overline{a}_{is} > 0\}\}$

1. *Kleinster Variablenindex Regel:* Wähle Pivotzeile $r \in R$ mit

$$p_r = \min\{p_i \text{ Basisvariable} \mid i \in R\}$$

2. *Lexikographische Zeilenauswahlregel:* Wähle Pivotzeile $r \in R$ mit

$$\tfrac{1}{\overline{a}_{rs}}(A_B^{-1})_{r\cdot} = \min_{lex}\{\tfrac{1}{\overline{a}_{is}}(A_B^{-1})_{i\cdot} \mid \overline{a}_{is} > 0\}$$

Definition lexikographisch positv: Wenn die erste von Null verschiedene Komponente eines Vektors $x \in \mathbb{R}^n$ positiv ist, so heißt der Vektor x lexikographisch positiv, wir schreiben dann $x \succ 0$. Sei $y \in \mathbb{R}^n$, dann gilt $x \succ y$, wenn $x - y \succ 0$ und $x \succeq y$, wenn $x - y \succ 0$ oder $x = y$. Mit $\min_{lex}$ bezeichnen wir das lexikographische Minimum einer gegebenen Menge von Vektoren.

Endlichkeit des Simplex-Verfahrens kann man nun erreichen, indem man die zweite Pivotspaltenauswahlregel mit der ersten Pivotzeilenauswahlregel kombiniert oder bei beliebiger Wahl der Pivotspalte die lexikographische Zeilenauswahlregel anwendet.

Jetzt wissen wir, wie man von einer Basislösung zur nächsten gelangt und dabei den Zielfunktionswert verbessert. Außerdem haben wir Kriterien dafür, wann wir optimal sind und wann wir wissen, daß das Problem unbeschränkt ist.

Nun kann die Grundversion des Simplex-Verfahrens angegeben werden. Das Simplex-Verfahren arbeitet wie folgt.

- Gestartet wird bei einer zulässigen Basisösung x_0. Existiert keine solche Basislösung x_0, so ist das Problem unlösbar und das Programm stoppt.
- Existiert dagegen eine Startlösung x_0, so liefert der Algorithmus eine Sequenz $x_0, ..., x_s$ mit den Eigenschaften
  1. $c^T x_0 \leq c^T x_1 \leq ... \leq c^T x_s$,
  2. $x_0, ..., x_s$ sind zulässige Basislösungen,
  3. x_{i-1}, x_i sind benachbart für $i = 1, ..., s$, d.h. es existiert eine Kante des Polyeders, die x_{i-1}, x_i verbindet und
  4. x_s ist entweder eine Optimallösung oder man weiß daß $c^T x$ auf (4) unbeschränkt ist.

Der Algorithmus wandert also von einer Ecke x_i zu einer benachbarten Ecke x_{i+1} mit der Eigenschaft $c^T x_i \leq c^T x_{i+1}$. Falls mehrere solcher benachbarter Ecken mit dieser Eigenschaft existieren, muß entschieden werden, welche Ecke ausgewählt werden soll. Diese Auswahl charakterisiert die verschiedenen Simplex-Varianten und macht sich in der Wahl der Pivotspalte und der Pivotzeile bemerkbar.

Man kann z.B. unter den zu x_i benachbarten Ecken die Ecke mit dem größten Zielfunktionswert auswählen.

Bei solch einem Basisaustausch müssen die oben eingeführten Regeln, wie man einen Basisaustausch mit Verbesserung des Zielfunktionswertes durchführt, eingehalten werden. Unbeschränktheit und Optimalität werden, wie oben erklärt, nachgewiesen.

3.1.3 Phase I

Es muß noch erklärt werden, wie die zulässige Startecke gefunden wird, denn im allgemeinen ist dies nicht so einfach. Um eine zulässige Basislösung zu berechnen, muß ein weiteres Problem gelöst werden. Das Lösen dieses Problems wird auch mit Phase I bezeichnet.

In der Phase I wird festgestellt, ob das zu lösende Problem eventuell keinen zulässigen Punkt besitzt, genau einen zulässigen Punkt besitzt oder optimiert werden muß.

Sei (3) das zu lösende lineare Programm. Wir konstruieren uns aus diesem Problem ein weiteres, das *Startproblem.*

Es hat die Form

$$\begin{array}{ll} \max & \mathbf{1}^T Ax \\ s.t. & Ax + Iy = b, x, y \geq 0 \text{ mit} \\ & A \in \mathbb{R}^{m \times n}, I \in \mathbb{R}^{m \times m} \textit{Einheitsmatrix}, \\ & b \in \mathbb{R}^m \text{ und } c, x \in \mathbb{R}^n, \text{ mit } m < n. \end{array} \tag{5}$$

Hierbei ist $\mathbf{1}$ der 1-Vektor, d.h. alle Komponenten des Vektors haben den Wert Eins.

Die Variablen y werden *künstliche* Variablen genannt. Da $Iy \geq 0$ gilt, ist die Zielfunktion durch $\mathbf{1}^T b$ nach oben beschränkt, denn es gilt $Ax \leq b$, also $\mathbf{1}^T Ax \leq \mathbf{1}^T b$.

Um für unser ursprüngliches Problem zulässig zu sein, muß $Ax = b$ gelten, also ein zulässiger Punkt x_0 des Startproblems gefunden werden mit $\mathbf{1}^T Ax_0 = \mathbf{1}^T b$. Wenn wir also einen zulässigen Punkt des Startproblems finden, der dieses

optimal löst, haben wir eine zulässige Lösung unseres eigentlichen Problems gefunden.

Bei dem Startproblem ist es dagegen einfach eine zulässige Startlösung zu finden. Man nimmt die Variablen y in die Basis und die Variablen x in die Nichtbasis und startet das Simplex-Verfahren.

Nun gibt es mehrere Möglichkeiten, wie das Simplex-Verfahren für das Startproblem enden kann. Sei $\overline{x} = \binom{x}{y}$ die optimale Ecke und $\overline{A} = (A, I)$.

1. Gilt $1^T Ax < 1^T b$, dann wissen wir, daß das ursprüngliche Problem keine zulässige Lösung besitzen kann, denn es gibt keinen Punkt x_0 für den $Ax_0 = b$ gilt.

2. Gilt $1^T Ax = 1^T b$ und sind keine künstlichen Variablen in der Basis und keine regulären Variablen in der Nichtbasis, dann ist m=n und das ursprüngliche Problem hat genau eine Lösung. Die Lösung des eigentlichen Problems ist dann x.

3. Gelten die Voraussetzungen wie unter 2., nur mit dem Unterschied, daß noch reguläre Variablen in der Nichtbasis sind. In diesem Fall haben wir eine zulässige Basis für das Ausgangsproblem gefunden und können mit ihr als Startbasis das eigentliche Optimierungsproblem lösen.

4. Gilt $1^T Ax = 1^T b$ und sind noch künstliche Variablen in der Basis, so müssen wir versuchen, die künstlichen Variablen aus der Basis zu entfernen.
   Da alle künstlichen Variablen in der Basis Null sind, können wir versuchen reguläre Nichtbasisvariable gegen sie in die Basis zu tauschen. Sei die r-te Basisvariable b_r eine künstliche Variable. Sie kann aus der Basis *hinauspivotisiert* werden, wenn es eine reguläre Nichtbasisvariable n_s gibt, mit $(\overline{A}_B^T \overline{A}_N)_{rs} \neq 0$. Der Satz über den Basisaustausch sagt uns, daß wir nach dem Basiswechsel wieder eine Basis erhalten. Da wir einen degenerierten Pivotschritt durchgeführen, sprich nur Nullen gegen Nullen tauschen, ist die neue Basis wieder zulässig, auch wenn wir eine Nichtbasisvariable mit $(\overline{A}_B^T \overline{A}_N)_{rs} < 0$ auswählen sollten. Außerdem sind wir wieder optimal, da sich der Zielfunktionswert nicht geändert hat.

5. Es gelten dieselben Voraussetzungen wie unter (4.). Kann eine künstliche Variable nicht aus der Basis hinauspivotisiert werden, so ist die entsprechende Zeile bei beliebiger Wahl für x_n stets erfüllt und kann gestrichen werden.

Gilt nach dem Streichen aller überflüssigen Ungleichungen, daß die Anzahl der Gleichungen gleich der Anzahl der regulären Variablen ist, so wissen wir, daß das ursprüngliche Problem nur einen zulässigen Punkt besitzt.
Im anderen Fall setzen wir alle regulären Variablen der Basis als Basis für das Ausgangsproblem und lösen mit ihr als Startbasis das eigentliche Optimierungsproblem.

3.2 Schatteneckenalgorithmus

Der Schatteneckenalgorithmus ist eine Variante des Simplex-Verfahrens. Er löst Probleme der Form

$$\begin{aligned} \max \quad & c^T x && (6)\\ s.t. \quad & Ax = b, x \geq 0 \text{ mit} && (7)\\ & A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, b \geq 0 \text{ und} \\ & c, x \in \mathbb{R}^n. \end{aligned}$$

Im Gegensatz zum Simplex-Verfahren benötigt er allerdings noch eine zweite Zielfunktion, sagen wir $\overline{c} \in \mathbb{R}^n$. Es muß darauf geachtet werden, daß $\overline{c}$ nicht lediglich ein Vielfaches der Zielfunktion c ist, d.h. $\overline{c} \neq \lambda c$, $\lambda \in \mathbb{R}$ gilt.

Der Algorithmus startet bei einer zulässigen Ecklösung x_0, falls denn eine existiert, die optimal bezüglich $\overline{c}$ ist, d.h.

$$\overline{c}^T x_0 = \max\{\overline{c}^T x \mid Ax = b, x \geq 0\}.$$

Falls $\overline{c}$ nicht anders vorgegeben wird und das Problem in der Form $\overline{A}x \leq \overline{b}$ gegeben ist (siehe 6.1.1), so kann man zu einer zulässigen Ecklösung x_0 ein $\overline{c}$ aus der Menge

$$M = \left\{ c' \in \mathbb{R}^n \mid c' = \sum_{\substack{i=1 \\ \overline{A}_{i.} x_0 = \overline{b}_i}}^{m} p_i \overline{A}_{i.}, \text{ mit } p_i \geq 0 \text{ und } c' \neq 0 \right\}$$

auswählen. Denn man sieht leicht, daß x_0 für alle $c_0 \in M$ eine Optimallösung ist.

Sei f_μ definiert durch

$$f_\mu = (1 - \mu)\overline{c} + \mu c.$$

Der Schatteneckenalgorithmus liefert nun eine Sequenz $x_0, ..., x_s$ und eine Einteilung $0 \leq \alpha_{-1} \leq \alpha_0 \leq ... \leq \alpha_s \leq 1$ des [0,1] Intervalls, so daß folgende Bedingungen erfüllt sind:

1. x_i geht aus x_{i-1} aus einem Pivotschritt hervor
2. x_i ist ein Maximum für f_μ für alle $\mu \in [\alpha_{i-1}, \alpha_i]$
3. $c^T x_0 \leq c^T x_1 \leq \ldots \leq c^T x_s$
4. x_s ist entweder eine Optimallösung oder man weiß, daß c auf (7) unbeschränkt ist.

Durch wachsende α_i erlangt die Funktion c immer mehr Gewicht, die Richtung der Zielfunktion wird also von $\overline{c}$ in Richtung von c gedreht.

Anschaulich arbeitet der Algorithmus wie folgt:

Sei $E = span(c, \overline{c})$ die Ebene, die durch die beiden Zielfunktionen c und $\overline{c}$ aufgespannt wird. Nun projiziert man das Polyeder $P = P^{=}(A, b)$ unter der zu E orthogonalen Projektion Γ auf E. Die Ecken $\overline{x}$ von P, deren Projektion $\Gamma(\overline{x})$ auch Ecken der Projektion $\Gamma(P)$ sind, werden *Schattenecken* genannt. Abbildung 1 zeigt die Schattenecken eines dreidimensionalen Polyeders bezüglich der Betrachtungsebene.

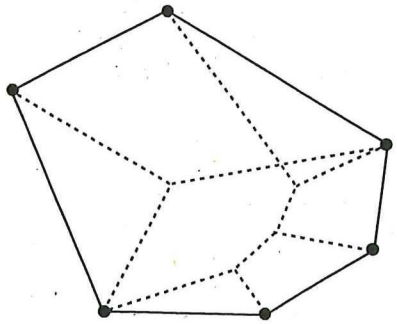

Abbildung 1: Schattenecken bezüglich der Betrachtungsebene

Ist das zu lösende Problem nichtdegeneriert, so besteht die Sequenz $x_0, \ldots, x_s$ nur aus Schattenecken. Und zwar wandert der Algorithmus über Schattenecken, die innerhalb des kleineren Winkels zwischen c und $\overline{c}$ liegen, in Richtung der Optimallösung.

Ist das Problem dagegen degeneriert, so dann es vorkommen, daß die Sequenz Ecken x_i enthält, die keine Schattenecken sind, jedoch die Eigenschaft $\Gamma(x_i) \in \partial\Gamma(P)$ besitzen. Dies bedeutet, daß die Projektion x_i keine Ecke der Projektion $\Gamma(P)$ ist. Die Projektionen dieser Ecken liegen dann auf dem Rand der Projektion des Polyeders.

Vergleicht man die Sequenzen, die der Schatteneckenalgorithmus und das Simplex-Verfahren liefern, miteinander, so stellt man fest, daß die Sequenz des Schatteneckenalgorithmus ein Simplex-Pfad ist.

Der Unterschied zum Simplex-Verfahren macht sich in der Auswahlregel der Pivotspalte bemerkbar. Die Pivotspalte wird beim Schatteneckenalgorithmus nach folgender Regel ausgewählt.

3.2.1 Wahl der Pivotspalte

Seien α_i und β_i die i-ten Einträge der reduzierten Kosten der Zielfunktionen c und $\overline{c}$. Wenn wir nach c optimieren, wählen wir eine Pivotspalte s mit der Eigenschaft

$$-\frac{\beta_s}{\alpha_s} = \min\left\{-\frac{\beta_i}{\alpha_i} \mid \alpha_i > 0\right\}$$

3.2.2 Wahl der Pivotzeile

Bei der Wahl der Pivotzeile geht man wie üblich vor. Zwei Varianten wurden in diesem Kapitel bereits vorgestellt. Hier kann man ein mögliches Kreiseln des Programms vermeiden, indem man eine lexikographische Zeilenauswahlregel implementiert [5].

3.2.3 Anwendung des Algorithmus

Eine Anwendung findet der Schatteneckenalgorithmus, wenn zwei mögliche Zielfunktionen gegeben sind, nach denen optimiert werden soll.

Hat man nun alle Schattenecken des Problems bezüglich der beiden Zielfunktionen berechnet, so hat man für jede Zielfunktion, die sich als konische Kombination der beiden gegebenen Zielfunktionen darstellen läßt, eine Optimallösung.

Kommt es jetzt also vor, daß man die Optimallösung zu einer Zielfunktion berechnen will, die eine konische Kombination der beiden Zielfunktionen ist, so muß man nur alle Zielfunktionswerte der berechneten Schattenecken berechnen und unter ihnen den optimalen Wert auswählen. Eine optimale Ecke befindet sich somit in der vom Schatteneckenalgorithmus bereits berechneten Sequenz $x_0, .., x_s$.

4 Berechnung der Projektionen

In diesem Kapitel soll erläutert werden, wie die Projektionen der linearen Programme (6) auf $span(c, \overline{c})$ mit Hilfe des Schattenalgorithmus errechnet und dargestellt werden.

Die graphische Darstellung erfolgt in einem Koordinatensystem. Auf der x-Achse werden die Funktionswerte der berechneten Ecken bezüglich der ersten Zielfunktion abgetragen und auf der y-Achse die Zielfunktionswerte der entsprechenden Ecken bezüglich der zweiten Zielfunktion.

Wie bereits geschildert wurde, startet der Schatteneckenalgorithmus bei einer Ecke, die optimal bezüglich $\overline{c}$ ist. Anschließend optimiert er das Problem nach einer weiteren Zielfunktion, c.

Die Ecken, die während der einzelnen Pivotschritte bei der Optimierung berechnet werden, sind entweder Schattenecken oder sind Ecken, deren Projektionen auf dem Rand der Projektion des linearen Programms liegen.

Außerdem wissen wir, daß der Schatteneckenalgorithmus bei der Optimierung nach c mit $\overline{c}$ als zweiter Zielfunktion zwischen dem kleineren Winkel zwischen den beiden Zielfunktionen c und $\overline{c}$ entlang läuft, sprich für jede Zielfunktion, die auf $span(c, \overline{c})$ zwischen dem kleineren Winkel zwischen c und $\overline{c}$ liegt, existiert in der vom Schatteneckenalgorithmus berechneten Sequenz von Schattenecken eine zugehörige Optimallösung.

Betrachtet man die Projektion eines linearen Programms, so stellt man fest, daß gerade die Schattenecken die charakteristischen Punkte der Projektion sind. Alle Ecken der Projektion sind nämlich Schattenecken. Man muß also alle Schattenecken des gegebenen Problems berechnen und in der richtigen Reihenfolge durch eine Strecke miteinander verbinden.

Wie die Berechnung aller Schattenecken und die graphische Darstellung der Projektion realisiert wird, soll nun anhand eines kleinen Beispiels erklärt werden.

Abbildung 2 zeigt uns die Darstellung der Projektion aller Schattenecken eines gegebenen Problems. Auf der x-Achse sind die Werte der einzelnen Ecken bezüglich der Zielfunkion c und auf der y-Achse die Werte der einzelnen Ecken bezüglich der Zielfunktion $\overline{c}$ eingetragen.

Der Algorithmus startet bei einer Lösung, die optimal bezüglich $\overline{c}$ ist und optimiert das Problem nach der anderen Zielfunktion c. Optimieren entspricht hier generell dem Maximieren.

Verbindet man die Projektion der Ecken, die während der Optimierung berechnet werden, in der Reihenfolge in der sie berechnet wurden, so ergibt sich

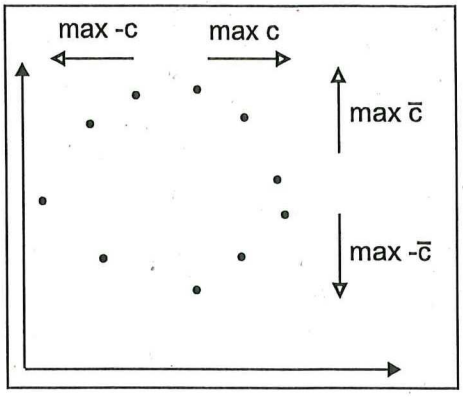

Abbildung 2: Schattenecken des Problems

die Situation, die in Abbildung 3 dargestellt ist. Denn der Schatteneckenalgorithmus berechnet, wie oben bereits erwähnt, alle Schattenecken, deren Projektion innerhalb des kleineren Winkels zwischen den beiden Zielfunkionen c und $\overline{c}$ auf $span(c, \overline{c})$ liegen.

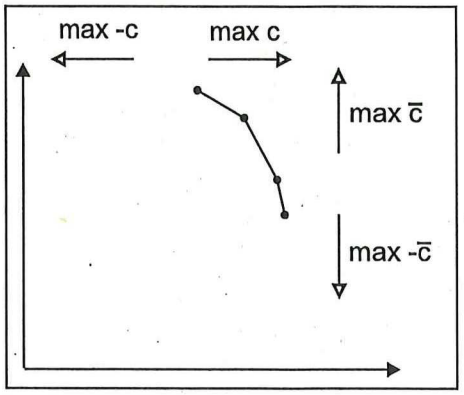

Abbildung 3: Projektion nach erster Optimierung

Da man jetzt optimal bezüglich c ist, kann man nach $-\overline{c}$ optimieren mit c als zweiter Zielfunktion. Nun verbindet man auch die Projektionen der während dieser Optimierung berechneten Ecken miteinander.

Hierbei muß noch angegeben werden, daß zwar nach $-\overline{c}$ optimiert wird, die berechneten Schattenecken jedoch bezüglich der beiden Zielfunkionen c und $\overline{c}$ in das Koordinatensystem eingetragen werden. Dies gilt auch für die noch durchzuführenden Phasen.

Abbildung 4 zeigt nun die Projektion nach zwei Optimierungen.

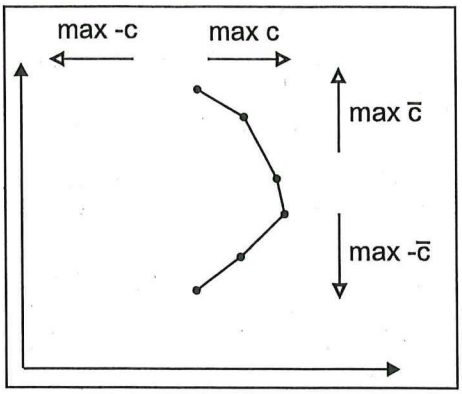

Abbildung 4: Projektion nach zweiter Optimierung

Nun ist klar, wie man weiter vorgeht. Zunächst optimiert man nach $-c$ mit $-\overline{c}$ als zweiter Zielfunktion. Abbildung 5 zeigt das Ergebnis.

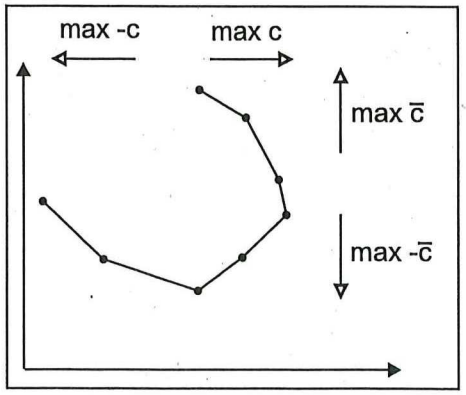

Abbildung 5: Projektion nach dritter Optimierung

Zuletzt muß noch nach $\overline{c}$ optimiert werden. Hier ist $-c$ die zweite Zielfunktion. Dadurch erhält man nun die vollständige Projektion des linearen Programms, dargestellt in Abbildung 6.

Nun ist klar, wie man die Projektion von linearen Programmen berechnet. Gestartet wird bei einer Ecke, die optimal bezüglich $\overline{c}$ ist und optimiert dann nacheinander nach c, $-\overline{c}$, $-c$ und $\overline{c}$ mit den entsprechenenden zweiten Zielfunktionen (s.o.). So hat man nämlich alle Winkel zwischen den Vektoren c und $\overline{c}$ auf $span(c, \overline{c})$ abgearbeitet und somit alle Schattenecken des gegebenen

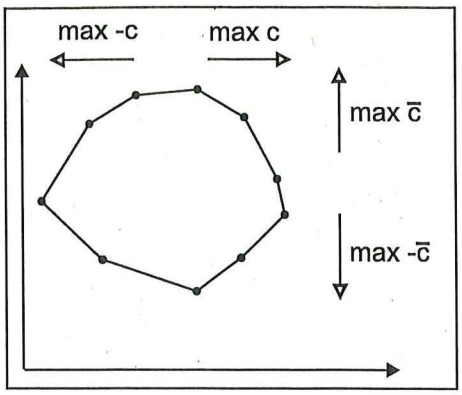

Abbildung 6: Projektion des linearen Programms

linearen Programms berechnet.

Wie die Projektion algorithmisch realisiert wird, soll nun grob beschrieben werden.

Um die Projektion zu errechnen sind, wie es auch aus dem oben beschriebenen Beispiel hervorgeht, vier Phasen notwendig.

Wir starten bei einer Ecke, die optimal bezüglich $\overline{c}$ ist. Die zweite Zielfunktion sei wieder c. Sei L die Liste der Ecken, die während der Iterationen berechnet werden. Zu Beginn beinhalte die Liste L die Ecke x_0, wobei x_0 die Ecke ist, die optimal bezüglich $\overline{c}$ ist.

Phase A : Optimiere mittels Schatteneckenalgorithmus nach c mit $\overline{c}$ als zweiter Zielfunktion. Füge bei jeder Iteration die aktuelle Ecke ans Ende der Liste L an (das Ergebnis von *Phase A* entspricht der Abbildung 3).

Wenn eine Optimallösung existiert und erreicht ist, dann starte *Phase B*. Wenn dagegen die Unbeschränktheit des Problems bezüglich c festgestellt wird, dann beende das Programm mit 'c unbeschränkt'.

Phase B : Optimiere mittels Schatteneckenalgorithmus nach $-\overline{c}$ mit c als zweiter Zielfunktion. Füge bei jeder Iteration die aktuelle Ecke ans Ende der Liste L an (das Ergebnis von *Phase B* entspricht der Abbildung 4).

Wenn eine Optimallösung existiert und erreicht ist, dann starte *Phase C*. Wenn dagegen die Unbeschränktheit des Problems bezüglich $-\overline{c}$ festgestellt wird, dann beende das Programm mit '$-\overline{c}$ unbeschränkt'.

Phase C : Optimiere mittels Schatteneckenalgorithmus nach $-c$ mit $-\overline{c}$ als zweiter Zielfunktion. Füge bei jeder Iteration die aktuelle Ecke ans Ende der

Liste L an (das Ergebnis von *Phase C* entspricht der Abbildung 5).

Wenn eine Optimallösung existiert und erreicht ist, dann starte *Phase D*. Wenn dagegen die Unbeschränktheit des Problems bezüglich $-c$ festgestellt wird, dann beende das Programm mit '$-c$ unbeschränkt'.

Phase D : Optimiere mittels Schatteneckenalgorithmus nach $\overline{c}$ mit $-c$ als zweiter Zielfunktion. Füge bei jeder Iteration die aktuelle Ecke ans Ende der Liste L an (das Ergebnis von *Phase D* entspricht der Abbildung 6). Wenn Optimallösung erreicht ist, dann Stop.

Aus Kapitel 3 wissen wir, daß die Elemente von L entweder Schattenecken sind oder aber, daß deren Projektionen auf dem Rand der Projektion des linearen Programms liegen.

Berechnen wir nun nacheinander für alle $x \in L$ die zugehörigen Funktionswerte $c^T x$ und $\overline{c}^T x$, tragen den Punkt $(c^T x, \overline{c}^T x)$ in das Koordinatensystem ein und verbinden die Punkte nacheinander, dann ist die erhaltene Figur die Projektion des Polyeders auf $span(c, \overline{c})$, dargestellt im Koordinatensystem.

Jetzt muß noch gezeigt werden, daß die so berechnete Figur wirklich die gewünschte Projektion ist. Dazu betrachten wir den folgenden Satz.

Satz: Gegeben sei ein Optimierungsproblem der Form $\max_{x \in P} \overline{c}^T x$, $P = P^{=}(A, b)$, eine zweite Zielfunktion c und ein $x_0 \in P$ mit $\overline{c}^T x_0 = \max_{x \in P} \overline{c}^T x$.

Startet man nun bei x_0 und führt *Phase A* bis *Phase D* durch, und wird bei keiner der Phasen Unbeschränktheit festgestellt und sei $L = \{x_0, ..., x_n\}$ die Liste der während der Optimierungen berechneten Ecken.

Man erhält dann die Projektion, indem man

1. $(c^T x_i, \overline{c}^T x_i)$ für $i = 0, ..., n$ in das Koordinatensystem einträgt,
2. $(c^T x_i, \overline{c}^T x_i)$ mit $(c^T x_{i+1}, \overline{c}^T x_{i+1})$ für $i = 0, ..., n-1$ durch eine Strecke verbindet und
3. $(c^T x_0, \overline{c}^T x_0)$ mit $(c^T x_n, \overline{c}^T x_n)$ durch eine Strecke verbindet, falls $(c^T x_0, \overline{c}^T x_0) \neq (c^T x_n, \overline{c}^T x_n)$ gilt.

Beweis: Aus Kapitel 3 wissen wir, daß der Schatteneckenalgorithmus bei der Optimierung alle Schattenecken zwischen dem kleineren Winkel der beiden Zielfunktionen berechnet.

Betrachtet man die vier Phasen, so stellt man fest, daß die jeweils kleineren Winkel zwischen $(c, \overline{c})$, $(-\overline{c}, c)$, $(-c, -\overline{c})$ und $(\overline{c}, -c)$ auf $span(c, \overline{c})$ bearbeitet werden. Da diese vier Winkel aber gerade die vier Winkel zwischen den beiden Vektoren c und $\overline{c}$ auf $span(c, \overline{c})$ sind, denn $\overline{c}$ ist kein Vielfachs von c, werden alle Schattenecken des Polyeders bezüglich der beiden Zielfunktionen berechnet.

Alle berechneten Ecken, die keine Schattenecken sind, besitzen eine Projektion, die auf dem Rand der Projektion des linearen Programms liegt, so daß sie die Projektion nicht beeinflussen.

Verbindet man die entsprechenden Punkte der berechneten Ecken, wie oben angegeben, miteinander, so folgt die Behauptung.

Hierbei kann es vorkommen, daß die Projektion nicht geschlossen ist. Dieser Fall kann dann eintreten, wenn das Polyeder in eine Richtung der Zielfunktionen unbeschränkt ist oder wenn das Polyeder mehrere Optimallösungen bezüglich der ersten Zielfunktion besitzt.

Der Fall, daß man die Punkte $(c^T x_0, \overline{c}^T x_0)$ und $(c^T x_n, \overline{c}^T x_n)$ noch miteinander verbinden muß kann dann eintreten, wenn das Problem mehrere Optimallösung bezüglich der ersten Zielfunktion besitzt. In diesem Fall kann man die beiden Punkte durch eine Strecke miteinander verbinden, da die Optimalität bezüglich der ersten Zielfunktion erreicht ist, also keine Schattenecke zwischen ihnen liegen kann. □

Wenn dagegen in einer der Phasen Unbeschränktheit festgestellt wurde, dann beschreibt die erhaltene Projektion lediglich den Teil zwischen dem Startwert x_0 und der letzten Ecke, bevor Unbeschränktheit festgestellt wurde.

In diesem Fall kann man von der Startlösung x_0 die Berechnung der Projektion in die andere Richtung starten, um sich auch von der anderen Seite der Unbeschränktheit zu nähern. Dies wird dadurch erreicht, indem man

- in *Phase A* und *Phase B* c durch $-c$ und
- in *Phase C* und *Phase D* $-c$ durch c ersetzt.

Was kann man jetzt eigentlich anhand der Projektion des Polyeders über das Problem aussagen? Zum einen kann man anhand der Struktur der Projektion etwas über die Komplexität des Problems ablesen.

Hat man auf der anderen Seite ein Problem gegeben, bei dem sich die Zielfunktion öfter ändert und sich als Linearkombination der beiden Zielfunktionen, mit denen die Projektion berechnet wurde, darstellen läßt, dann kann man die neue Optimallösung aus der Projektion berechnen. Die Frage, wie

sich aus der Projektion des Problems eine Optimallösung für ein beliebige Zielfunktion c_0 der Form $c_0 = \alpha\, c + \beta\, \overline{c}$ berechnen läßt, ist leicht beantwortet. Sei (x_0, y_0) eine Ecke der Projektion. Nun berechnet man aus der Ecke den Wert für die neue Zielfunktion durch $\alpha\, x_0 + \beta\, y_0$. Berechnet man aus allen Ecken der Projektion den neuen Zielfunktionswert auf diese Weise und wählt unter ihnen den optimalen Wert aus, so erhält man die Optimallösung für die neue Zielfunktion.

Der Rechenaufwand, aus den Ecken der Projektion eine optimale Lösung für die neue Zielfunktion zu errechnen, ist weitaus geringer als eine neue Optimierung des Problems mit der neuen Zielfunktion durchzuführen.

5 Beispiel Würfel

In diesem Kapitel soll anhand des Würfels veranschaulicht werden, wie die Wahl der Zielfunktionen Einfluß auf die Projektion und die Anzahl der Pivotschritte nimmt. Als Beispiel dient ein Optimierungsproblem über einem Würfel im $\mathbb{R}^3$. Das hier betrachtete lineare Programm hat die Form

$$\begin{aligned} \max \quad & \sum_{i=0}^{2} c_i x_i \\ s.t. \quad & 1 \leq x_0 \leq 3 \\ & 1 \leq x_1 \leq 3 \\ & 1 \leq x_2 \leq 3. \end{aligned} \tag{8}$$

Die Zielfunktion wird in den einzelnen Fällen separat angegeben. Die zweite Zielfunktion wird wieder mit $\overline{c}$ bezeichnet.

Betrachten wir nun die folgenden drei Beispiele

1. Seien $c = x_0 + x_1$ und $\overline{c} = x_0$ die beiden Zielfunktionen.
   Der Algorithmus errechnet nun bei der Berechnung der Projektion nacheinander die Ecken $(3,3,3), (1,3,3), (1,1,3), (3,1,3)$ und wieder $(3,3,3)$. Wie man leicht erkennt, sind dies alles Schattenecken. Abbildung 7 zeigt den Weg, den der Algorithmus zurücklegt und Abbildung 10 die Projektion auf $span(c, \overline{c})$. Dabei repräsentiert die x-Achse die Zielfunktion c und die y-Achse $\overline{c}$

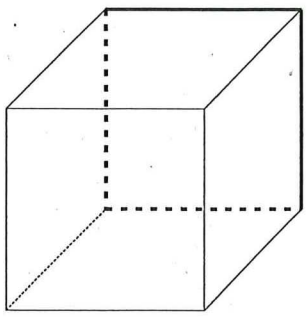

Abbildung 7: Schattenecken bezüglich des ersten Beispiels

2. Seien $c = x_0 + x_1$ und $\overline{c} = x_1 + x_2$ die beiden Zielfunktionen.
   Jetzt ändern sich die Pivotschritte und die Ecken $(3,3,3), (3,3,1)$,

$(3,1,1), (1,1,1), (1,1,3), (1,3,3), (3,3,3)$ werden errechnet. Auch hier erkennt man, daß die errechneten Ecken Schattenecken sind. Der Weg des Algoritmus wird in Abbildung 8, die Projektion auf $span(c, \overline{c})$ in Abbildung 11 verdeutlicht. Dabei stellen die beiden Koordinatenachsen wieder die beiden Zielfunktionen dar.

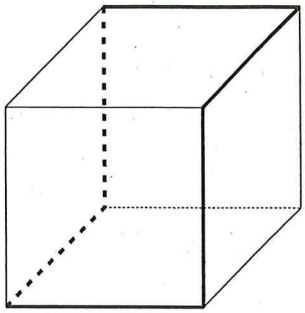

Abbildung 8: Schattenecken bezüglich des zweiten Beispiels

3. Seien $c = -x_0 + x_1 + x_2$ und $\overline{c} = x_2$ die beiden Zielfunktionen.
   Hier tritt der Fall ein, daß der Algorithmus nicht nur Ecken berechnet, die Schattenecken sind. Er errechnet die Ecken $(1,3,3), (1,3,1), (1,1,1), (3,1,1), (3,1,3), (3,3,3), (1,3,3)$. Den zurückgelegten Weg und die Projektion stellen die Abbildungen 9 und 12 dar. Daß nicht nur Schattenecken berechnet wurden, erkennt man daran, daß in Abbildung 12 Punkte existieren, die keine Ecke der Projektion darstellen.

   Wiederum repräsentieren die beiden Koordinatenachsen die beiden Zielfunktionen.

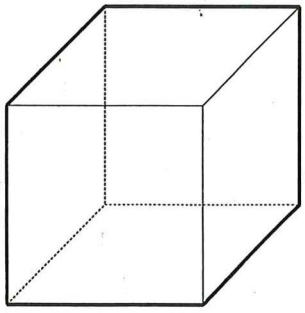

Abbildung 9: Schattenecken bezüglich des dritten Beispiels

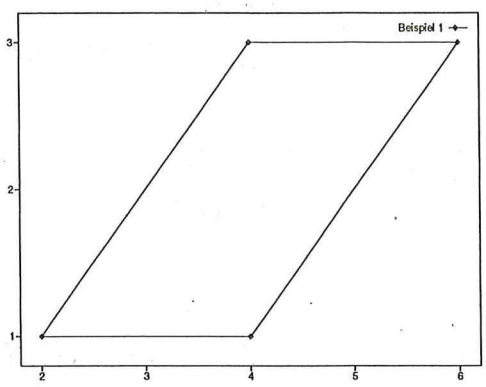

Abbildung 10: Projektion des ersten Beispiels

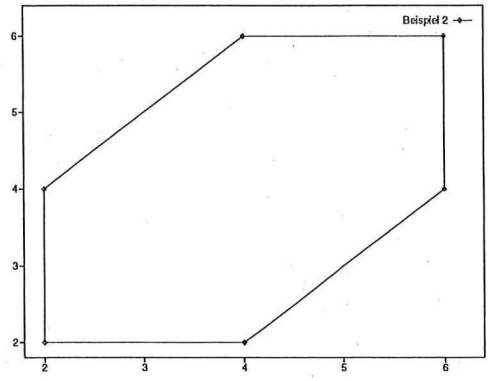

Abbildung 11: Projektion des zweiten Beispiels

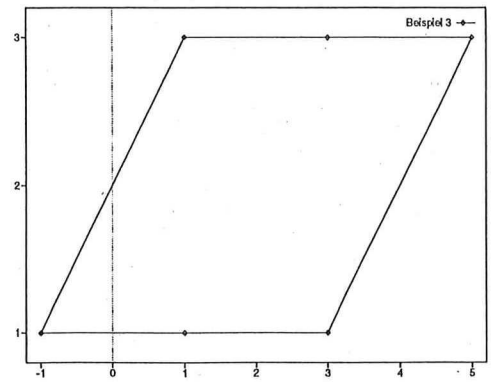

Abbildung 12: Projektion des dritten Beispiels

Die ersten beiden Beispiele sollen zeigen, daß die Wahl der zweiten Zielfunktion direkten Einfluß auf die Anzahl der Pivotschritte nehmen kann und die Projektionen erwartungsgemäß verschieden sind.

Der dritte Fall zeigt, daß es im degenerierten Fall vorkommen kann, daß bei der Berechnung der Projektion nicht nur Schattenecken berechnet werden.

Aber wie man deutlich sieht, liegen die Projektionen der Ecken, die keine Schattenecken sind, auf Kanten der Projektion (Abbildung 12) und beeinflussen die Projektion somit nicht.

6 Implementierung des Schatteneckenalgorithmus

In diesem Kapitel soll geschildert werden, wie der Algorithmus implementiert worden ist, welche Probleme bei der Berechnung der Projektionen auftraten und wie sie gelöst worden sind.

Es wurden zwei Versionen implementiert. Version A löst Probleme der Form (6), während Version B Probleme der Form

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax = b \\ & lb \leq x \leq ub \text{ mit} \\ & A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m \text{ und } lb, ub, c, x \in \mathbb{R}^n \end{aligned} \tag{9}$$

löst.

Beschäftigen wir uns zuerst mit der Version A.

6.1 Version A

Wie bereits erwähnt löst die *Version A* des Schatteneckenalgorithmus Probleme der Form (6). Da die Probleme aber in allgemeiner Form die Gestalt

$$\max \quad c^T x \tag{10}$$

$$s.t. \quad l \leq Ax \leq u \tag{11}$$

$$lb \leq x \leq ub \text{ mit} \tag{12}$$

$$A \in \mathbb{R}^{m \times n}, l, u \in \mathbb{R}^m \text{ und } lb, ub, c, x \in \mathbb{R}^n,$$

haben können, sprich sowohl obere als auch untere Schranken für die Ungleichungen als auch Beschränkungen nach oben und unten für die Variablen besitzen können, muß das Problem unter Umständen zuerst in die gewünschte Form transformiert werden.

Ist eine Ungleichung bzw. eine Variable nicht nach oben oder unten beschränkt, so sind die entsprechenden Einträge in u und l "∞" und "$-\infty$" und in ub und lb "∞" und "0".

6.1.1 Transformation des Problems

Bei der Transformation des Problems müssen sowohl die Ungleichungen als auch die oberen und unteren Schranken der Variablen umgeformt werden. Um das allgemein gegebene Problem in die benötigte Form zu bringen, geht man folgendermaßen vor.

Transformation der Ungleichungen
Eine Ungleichung $l_i \leq (Ax)_{i\cdot} \leq u_i$ des Ungleichungssystems (11) wird transformiert in

$$\begin{array}{rcrll} (Ax)_{i\cdot} & \leq & u_i, & \text{falls} & l_i \neq u_i < \infty \\ -(Ax)_{i\cdot} & \leq & -l_i, & \text{falls} & -\infty < l_i \neq u_i \\ (Ax)_{i\cdot} & = & l_i, & \text{falls} & -\infty < l_i = u_i < \infty \end{array} \tag{13}$$

Hierbei ist es natürlich möglich, daß eine Ungleichung in zwei neue Ungleichungen transformiert wird. Gilt für eine Ungleichung $-\infty < l_i < u_i < \infty$, so wird diese Ungleichung in

$$\begin{array}{rcl} (Ax)_{i\cdot} & \leq & u_i \\ -(Ax)_{i\cdot} & \leq & -l_i \end{array}$$

umgeformt. Außerdem wird vorausgesetzt, daß $l \leq u$ gilt.

Transformation der Beschränkungen
Aus den unteren und oberen Schranken der Variablen (12) werden ebenfalls Ungleichungen erstellt.

Dürfen die Variablen auch negative Werte annehmen, so muß das bei der Transformation ebenfalls berücksichtigt werden.

Die Transformation von $lb_i \leq x_i \leq ub_i$ erfolgt nach folgenden Regeln

$$\begin{array}{rcrll} x_i & \leq & ub_i, & \text{falls} & 0 < ub_i < \infty \\ -x_i & \leq & -lb_i, & \text{falls} & 0 < lb_i < \infty \\ x_i^+ - x_i^- & \leq & ub_i, & \text{falls} & -\infty < ub_i \leq 0 \\ -x_i^+ + x_i^- & \leq & -lb_i, & \text{falls} & -\infty < lb_i < 0 \end{array} \tag{14}$$

Wird die Variable x_i durch $x_i^+ - x_i^-$ ersetzt, so muß x_i in allen Ungleichungen und in der Zielfunktion durch $x_i^+ - x_i^-$ ersetzt werden. Auch hier wird $lb \leq ub$ vorausgesetzt.

Durch das Hinzufügen von Schlupfvariablen und durch das positiv machen der neuen rechten Seite erhalten wir nun die gewünschte Form, so daß wir das Problem jetzt lösen können.

Wie bereits erwähnt, arbeitet der Algorithmus in Pivotschritten, in denen er von einer Ecke zu der nächsten gelangt, bzw. ein Basiselement gegen ein Nichtbasiselement tauscht, also einen Basisaustausch durchführt.

Ein solcher Pivotschritt des Algorithmus setzt sich aus mehreren Funktionen zusammen, die im folgenden beschrieben werden sollen.

6.1.2 Wahl der Pivotspalte

Wenn wir von einer Ecke zur nächsten gelangen wollen, muß ein Basisaustausch durchgeführt werden, d.h. ein Element, das nicht in der Basis ist, kommt in die Basis und ein Basiselement verläßt dafür die Basis.

Bei der Wahl der Pivotspalte wird eine Variable, die nicht in der Basis ist, ausgewählt, um in die Basis zu gelangen.

Sei c die Zielfunktion nach der optimiert werden soll und $\overline{c}$ die zweite Zielfunktion. c_{red} und $\overline{c}_{red}$ seien die aktuellen reduzierten Kosten, B sei die aktuelle Basis und N die aktuelle Nichtbasis.

Wähle nun eine Pivotspalte s mit der Eigenschaft

$$-\frac{\overline{c}_{red}[s]}{c_{red}[s]} = \min\left\{-\frac{\overline{c}_{red}[i]}{c_{red}[i]} \mid c_{red}[i] > 0 \text{ und } i \in N\right\}.$$

Wenn kein $c_{red}[i] > 0$ mit $i \in N$ existiert, dann ist die aktuelle Ecklösung optimal und das Programm kann beendet werden.

6.1.3 Wahl der Pivotzeile

Nachdem die neue Basisvariable berechnet worden ist, muß noch die Variable ausgewählt werden, die die Basis verläßt. Sei $ps := A_B^{-1} A_{\cdot n_s}$ und $\overline{b} := A_B^{-1} b$.

Wähle eine Pivotzeile z mit der Eigenschaft

$$\frac{\overline{b}[z]}{ps[z]} = \min\left\{\frac{\overline{b}[i]}{ps[i]} \mid ps[i] > 0, 1 \leq i \leq m\right\} \text{ und } ps[z] > 0.$$

Existiert kein $ps[i] > 0$ mit $1 \leq i \leq m$, so ist das Problem unbeschränkt und das Programm kann beendet werden.

Jetzt kann man den Basisaustausch durchführen, indem man die z-te Basisvariable aus der Basis entfernt und der Nichtbasis hinzufügt und die s-te Nichtbasisvariable die entsprechende Stelle in der Basis einnimmt.

6.1.4 Update

Nach einem Pivotschritt müssen die wichtigsten Variablen aktualisiert werden, so müssen die Indexfelder der Basis- und Nichtbasisvariablen aktualisiert werden. Dies ist jedoch kein weiteres Problem.

Ein Problem dagegen ist das Lösen der drei Gleichungssysteme

1. $A_B\pi = c_B$ für die Berechnung der reduzierten Kosten.
2. $A_B ps = A_{.n_s}$ für die Aktualisierung der Pivotspalte.
3. $A_B\bar{b} = b$ für die Aktualisierung der rechten Seite.

Hierzu stand ein Programm innerhalb von SoPlex von R.Wunderling zur Verfügung.

6.1.5 Terminierung

Das Programm endet mit einer Optimallösung, wenn keine Pivotspalte ausgewählt werden kann. Wenn keine Pivotzeile ausgewählt werden kann, so endet das Programm mit der Aussage, daß das Problem unbeschränkt ist.

6.1.6 Die Startbasis

In Kapitel 3 wurde erklärt, daß es im allgemeinen nicht so einfach ist, eine zulässige Startbasis zu finden. Deshalb wird zunächst ein Hilfsproblem (5) gelöst. Dabei wurde für jede Gleichung eine künstliche Variable eingefügt. Dadurch wächst die Dimension des Problems natürlich stark an.

Um möglichst wenige künstliche Variablen einfügen zu müssen, geht man folgendermaßen vor.

Habe das Problem inzwischen die Form

$$\begin{aligned} \max \quad & c^T x && (15)\\ s.t. \quad & Ax \leq a && (16)\\ & Bx = b. && (17) \end{aligned}$$

Für alle Gleichungen (17) müssen künstliche Variablen eingefügt werden. Da bei den Ungleichungen (16) sowieso Schlupfvariablen eingefügt werden müssen, kann man diese eventuell gleich in die Startbasis aufnehemen.

Dies ist genau dann möglich, wenn die rechte Seite der entsprechenden Zeile nichtnegativ ist. Am Ende der Phase I werden dann auch hier nur die künstlichen Variablen entfernt.

6.2 Version B

Wie bei der *Version A* sollen auch hier die wichtigsten Funktionen beschrieben werden. Bevor wir die einzelnen Funktionen jedoch erläutern, müssen noch einige wichtige Definitionen getroffen werden, da für diese Version der Begriff der Basis ein anderer ist.

Definitionen: Seien B und N die Spaltenindexvektoren wie sie in (3.1) definiert wurden und sei x_i, mit $lb_i \leq x_i \leq ub_i$ eine Variable des linearen Programms (9).

1. Gilt $lb_i = -\infty$ und $ub_i = \infty$, so heißt x_i *freie* Variable.
2. Gilt $lb_i = ub_i$, so heißt x_i *fixierte* Variable.
3. Die Menge N der Nichtbasisvariablen ist unterteilt in die Mengen N_f, N_l, N_u, N_k, wobei
   N_f die Menge der freien Variablen,
   N_l die Menge der Nichtbasisvariablen x_i mit $x_i = lb_i$,
   N_u die Menge der Nichtbasisvariablen x_i mit $x_i = ub_i$ und
   N_k die Menge der fixierten Variablen darstellt.
4. Sei A_B eine Basismatrix, dann ist eine Basislösung x definiert durch

$$\begin{aligned} x_{N_f} &= 0 \\ x_{N_l} &= lb_{N_l} \\ x_{N_u} &= ub_{N_u} \\ x_{N_k} &= lb_{N_k} \\ x_B &= A_B^{-1}(b - A_N x_N) \end{aligned} \tag{18}$$

5. Eine Basislösung x heißt zulässig, wenn $lb_B \leq x_B \leq ub_B$ gilt.

An der Definition der Basislösung erkennt man, daß im Gegensatz zur bisherigen Basislösung, hier Nichtbasisvariablen nicht notwendigerweise Null sein müssen. Dieser Unterschied macht sich natürlich bei der Wahl der Pivotspalte und der Pivotzeile bemerkbar.

Beginnen wollen wir nun wieder mit der Transformation des allgemeinen Problems (10).

6.2.1 Transformation des Problems

Da *Version B* Probleme der Form (9) löst, müssen nur die Ungleichungen des gegebenen Problems (11) transformiert werden. Dies wird ebenso realisiert wie es in (13) beschrieben ist.

Da die Variablen nicht in Ungleichungen transformiert werden müssen, wird die Dimension des Polyeders nicht unnötig vergrößert. Außerdem spart man dadurch im Vergleich zu (14) bis zu $2n$ Gleichungen.

6.2.2 Wahl der Pivotspalte

Bei der Wahl der Pivotspalte müssen wir bei den Nichtbasisvariablen unterscheiden, ob sie Elemente von N_f, N_l, N_u oder N_k sind.

Fixierte Variablen werden bei der Wahl der Pivotspalte gar nicht berücksichtigt. Für die restlichen Nichtbasisvariablen aus N_f, N_l und N_u wird folgendes geprüft.

Seien c_{red} und $\overline{c}_{red}$ die reduzierten Kosten. Optimalität haben wir dann erreicht, wenn

$$\begin{aligned} c_{red}[i] &= 0 \text{ für alle } i \in N_f, \\ c_{red}[i] &\leq 0 \text{ für alle } i \in N_l \quad \text{und} \\ c_{red}[i] &\geq 0 \text{ für alle } i \in N_u \end{aligned}$$

gilt.

Angenommen wir sind noch nicht optimal. Dann berechnen wir das Minimum aus $\min_f, \min_l$ und $\min_u$, wobei diese aus folgenden Mengen berechnet werden

$$\begin{aligned} \min_f &:= \min\left\{-\frac{\overline{c}_{red}[i]}{c_{red}[i]} \mid i \in N_f \text{ und } c_{red}[i] \neq 0\right\} \\ \min_l &:= \min\left\{-\frac{\overline{c}_{red}[i]}{c_{red}[i]} \mid i \in N_l \text{ und } c_{red}[i] > 0\right\} \\ \min_u &:= \min\left\{-\frac{\overline{c}_{red}[i]}{c_{red}[i]} \mid i \in N_u \text{ und } c_{red}[i] < 0\right\}. \end{aligned} \tag{19}$$

Nun wählen wir eine Pivotspalte s, so daß gilt

$$-\frac{\overline{c}_{red}[s]}{c_{red}[s]} = \min\{\min_f, \min_l, \min_u\}.$$

6.2.3 Wahl der Pivotzeile

Nachdem wir die Pivotspalte ausgewählt haben, müssen wir eine Pivotzeile wählen.

Im Gegensatz zur *Version A* muß nach einem Pivotschritt nicht unbedingt eine Nichtbasisvariable in die Basis gelangen.

Es kann auch vorkommen, daß der Wert einer Nichtbasisvariablen von einer ihrer Schranken auf die andere gesetzt wird, der Wert der Basisvariablen dementsprechend geändert wird und die Nichtbasisvariable trotzdem nicht in die Basis gelangt.

Seien s die gewählte Pivotspalte, c_{red} die aktuellen reduzierten Kosten und $ps = A_B^{-1} A_{.n_s}$ die entsprechende aktuelle Spalte, dann berechnet man die Pivotzeile folgendermaßen.

1. Sei $c_{red}[s] > 0$, dann berechne für alle $i = 1..m$

$$\psi_i = \begin{cases} \infty & \text{falls } ps[i] = 0 \\ \frac{x_{B_i} - l_{B_i}}{ps[i]} & \text{falls } ps[i] > 0 \\ \frac{x_{B_i} - u_{B_i}}{ps[i]} & \text{falls } ps[i] < 0 \end{cases}$$

2. Sei $c_{red}[s] < 0$, dann berechne für alle $i = 1..m$

$$\psi_i = \begin{cases} \infty & \text{falls } ps[i] = 0 \\ \frac{u_{B_i} - x_{B_i}}{ps[i]} & \text{falls } ps[i] > 0 \\ \frac{l_{B_i} - x_{B_i}}{ps[i]} & \text{falls } ps[i] < 0 \end{cases}$$

Berechne nun das Minimum

$$\Psi = \min \left\{ \min_{i=1..m} \psi_i, u_{n_s} - l_{n_s} \right\}.$$

Nun unterscheiden wir zwei Fälle

1. Angenommen es gilt $\Psi = \infty$. In diesem Fall können wir das Programmm beenden, denn dann wissen wir, daß das gegebene Problem unbeschränkt ist.

2. Gilt dagegen $\Psi < \infty$, so setzen wir

$$\begin{aligned} x_B &:= x_B - \Psi ps, \text{ falls } c_{red}[s] > 0 \text{ und} \\ x_B &:= x_B + \Psi ps, \text{ falls } c_{red}[s] < 0. \end{aligned}$$

Nun muß noch geklärt werden, ob die Nichtbasisvariable n_s nur von einer ihrer Schranken auf die andere gesetzt wird oder ob sie in die Basis gelangt.

Hierzu betrachten wir Ψ noch etwas genauer. Wir unterscheiden wieder zwei Fälle

(a) Gilt $\Psi = u_{n_s} - l_{n_s}$, dann setze den Wert der Nichtbasisvariablen von u_{n_s} auf l_{n_s} bzw. von l_{n_s} auf u_{n_s}, je nachdem welchen Wert die Nichtbasisvariable zu dieser Zeit besitzt. Anschließend aktualisiere die beiden Mengen N_u und N_l.

(b) Gilt $\Psi < u_{n_s} - l_{n_s}$ und sei $z \in \{1, .., m\}$ mit $\psi_z = \Psi$. Dann verläßt das z-te Basiselement die Basis und wird durch n_s in der Basis ersetzt. Zuvor erhält die Variable, die in die Basis gelangt den Wert

$$x_{n_s} = \begin{cases} l_{n_s} + \Psi & \text{falls } n_s \in N_l \\ u_{n_s} - \Psi & \text{falls } n_s \in N_u \\ \Psi & \text{falls } n_s \in N_f \text{ und } c_{red}[s] > 0 \\ -\Psi & \text{falls } n_s \in N_f \text{ und } c_{red}[s] < 0. \end{cases}$$

Dann gelangt die Variable an die z-te Stelle der Basis. Anschließend müssen noch die Mengen N_u, N_l, N_f und N_k aktualisiert werden.

6.2.4 Die Startbasis

Genauso wie bei *Version A* kann man auch hier durch das Einfügen von möglichst vielen Schlupfvariablen die Dimension des Problems geringer halten. Habe das Problem die Form

$$\begin{aligned} \max \quad & c^T x \\ s.t. \quad & Ax \leq a \\ & Bx = b \qquad (20) \\ & l \leq x \leq u \qquad (21) \end{aligned}$$

Für die Gleichungen (20) werden auch hier künstliche Variable eingefügt. Wir wollen nun die Ungleichungen betrachten.

Zunächst weisen wir allen Variablen die Werte

$$x_k := l_k \text{ , wenn } |l_k| \leq |u_k| \text{ und } l_k > -\infty$$
$$x_k := u_k, \text{ wenn } |l_k| > |u_k| \text{ und } u_k < \infty$$
$$x_k := 0 \text{ , wenn } l_k = -\infty \text{ und } u_k = \infty$$

zu. Anschließend werden die Variablen den entsprechenden Nichtbasismengen zugeordnet.

Der Wert der Basisvariablen wird durch (18) berechnet. Deshalb betrachtet man den Vektor

$$v = b - A_N x_N.$$

Die Schlupfvariable, die der i-ten Ungleichung hinzugefügt wird, kann genau dann in die Startbasis aufgenommen werden, wenn $v_i \geq 0$ gilt.

Der Startwert der eingefügten Schlupfvariablen ist dann v_i. Ansonsten wird die Schlupfvariable mit Wert Null hinzugefügt. Schlupfvariablen dürfen generell nur nichtnegative Werte annehmen.

6.2.5 Phase I

In (3.1.3) wird erläutert, wie man eine zulässige Startecke findet. Ein wichtiger Schritt ist das "Hinauspivotisieren" der künstlichen Variablen aus der Basis. Sind alle Nichtbasisvariablen Null, dann tauscht man Nullen gegen Nullen, verändert also die aktuelle Lösung nicht.

Hier muß nun noch gezeigt werden, daß sich auch bei der *Version B* nach einem solchen Tausch der Wert der Lösung sich verändert hat und die neue Basislösung wieder eine zulässige Basislösung ist.

Angenommen die r-te Basisvariable ist eine künstliche Variable und es gibt eine reguläre Nichtbasisvariable n_s, mit $(\overline{A}_B^{-1}\overline{A}_N)_{rs} \neq 0$.

Mit $\Psi = 0$ gilt dann die Basisvariablen werden aktualisiert durch

$$x_B := x_B \pm \Psi ps = x_B.$$

Der Wert der aktuellen Basisvariablen hat sich somit nicht geändert.

Für die neue Basisvariable gilt

$$x_{n_s} = \begin{cases} l_{n_s} + \Psi = l_{n_s} & \text{falls } n_s \in N_l \\ u_{n_s} - \Psi = u_{n_s} & \text{falls } n_s \in N_u \\ \Psi = 0 & \text{falls } n_s \in N_f \text{ und } c_{red}[s] < 0 \\ -\Psi = 0 & \text{falls } n_s \in N_f \text{ und } c_{red}[s] > 0. \end{cases}$$

Somit hat sich auch der Wert der neuen Basisvariablen nicht verändert. Da alle Variablen denselben Wert haben wie zuvor und die neue Basismatrix wieder regulär ist, ist die neue Basislösung wieder zulässig.

Hier können die künstlichen Variablen also genauso aus der Basis "hinauspivotisiert" werden, wie es in der *Version A* geschieht.

6.3 Datenstrukturen

Im Wesentlichen beruht die Speicherung der Daten auf den Speicherstrukturen, die im Rahmen von SoPlex [7] entwickelt wurden. SoPlex ist ein Programm zum Lösen allgemeiner linearer Programme, das R.Wunderling im Rahmen seiner Promotion entwickelt hat. Neben der Speicherung der Daten wird noch seine Einleseroutine für lineare Programme im MPS oder LP Format verwendet.

Außerdem hat er eine Klasse entwickelt, die eine LU-Zerlegung von Matrizen durchführt und diese LU-Zerlegung mit Hilfe des Forest-Tomlin Verfahrans zwischen den einzelnen Pivotschritten aktualisiert.

Die Basis- und Nichtbasisvariablen werden in Indexfeldern gespeichert, wobei im Indexfeld für die *Version B* noch vermerkt wird, zu welcher der Mengen N_l, N_u, N_f und N_k die Nichtbasisvariablen gehören.

Die wichtigsten Funktionen sind

phaseI()	Löst das zum LP gehörende Startproblem
pivotspalte()	Berechnet die Pivotspalte bezüglich der ersten Zielfunktion
pivotspalte(u_{red},v_{red})	Berechnet die Pivotspalte bei der Optimierung von u unter der zweiten Zielfunktion v, wobei u_{red} und v_{red} deren reduzierte Kosten sind.
pivotzeile()	Berechnet die Pivotzeile
readFile(filename)	Liest das LP ein (von SoPlex)
update()	Aktualisiert die Indexfelder der Basis- und Nichtbasisvariablen und die LU-Zerlegung
transformLP()	Bringt ein allgemeines Problem auf die benötigte Form für *Version A* bzw. *Version B*.

6.4 Algorithmus

Der hier angegebene Algorithmus gilt für beide Versionen *Version A* und *Version B*. Der Unterschied macht sich lediglich in den Funktionen transformLP(), pivotspalte(), pivotspalte(u, v) und update() bemerkbar.

Der Algorithmus hat nun die folgende Gestalt

1. readFile()
2. transformLP()
3. phaseI(), wenn keine zulässige Startlösung existiert STOP, denn dann ist das Problem unlösbar.
4. Solange nicht optimal und nicht unbeschränkt bezüglich u
   (a) pivotspalte()
   (b) pivotzeile()
   (c) update()
5. Wenn unbeschränkt bezüglich u, STOP, denn das eigentliche Problem ist unbeschränkt.
6. Einlesen der zweiten Zielfunktion v
7. Solange nicht optimal und nicht unbeschränkt bezüglich v
   (a) pivotspalte(v_{red}, u_{red})
   (b) pivotzeile()
   (c) update()
8. Wenn unbeschränkt bezüglich v, dann gehe zu 14.
9. Solange nicht optimal und nicht unbeschränkt bezüglich $-u$
   (a) pivotspalte($-u_{red}$, v_{red})
   (b) pivotzeile()
   (c) update()
10. Wenn unbeschränkt bezüglich $-u$, dann gehe zu 14.
11. Solange nicht optimal und nicht unbeschränkt bezüglich $-v$

(a) pivotspalte($-v_{red}$, $-u_{red}$)

(b) pivotzeile()

(c) update()

12. Wenn unbeschränkt bezüglich $-v$, dann gehe zu 14.

13. Solange nicht optimal und nicht unbeschränkt bezüglich u

(a) pivotspalte(u_{red}, $-v_{red}$)

(b) pivotzeile()

(c) update()

14. Ausgabe der Punkte der Projektion in ein File. STOP

Ersetzt man in

Schritt 7	v_{red}	durch	$-v_{red}$
Schritt 9	v_{red}	durch	$-v_{red}$
Schritt 11	$-v_{red}$	durch	v_{red}
Schritt 13	$-v_{red}$	durch	v_{red}

so wandert der Algorithmus entlang der anderen Richtung der Projektion, wie der oben angegebene Algorithmus.

Der auf diese Weise veränderte Algorithmus wird dann gestartet, wenn in einem der Schritte 7, 9 oder 11 Unbeschränktheit festgestellt wird. Denn dann wird durch das Betrachten beider Richtungen die gesamte Projektion zwischen der Unbeschränktheit festgestellt.

6.5 Problematik

Das größte Problem, das bei der Implementierung auftrat, war das Auftreten von Rundungsfehlern. So traten Rundungsfehler bei der Berechnung der Inversen auf, so daß das Programm nicht korrekt terminieren konnte.

Eine erste Abhilfe hat die Einführung einer Variable *eps* geschafft. Jetzt werden zwei Werte a und b als gleich angesehen, wenn für die beiden Werte $|a - b| \leq eps$ gilt.

Dadurch lief der Algorithmus erst einmal stabiler. Es gab aber immer noch einige Praxisprobleme, die den Algorithmus zum Abbruch zwangen. Dies lag an dem Update der Basismatrix.

Nachdem die Klasse von SoPlex für eine LU-Zerlegung und deren Update eingebunden wurde, lief der Algorithmus stabil.

Der Schatteneckenalgorithmus eignet sich nicht zum schnellen Lösen linearer Programme, wenn man lediglich eine Zielfunktion betrachtet. Denn er benötigt im allgemeinen mehr Pivotschritte als andere Simplex-Verfahren und muß bei jedem Pivotschritt die reduzierten Kosten von zwei Zielfunktionen berechnen. Für die hier betrachtete Problematik ist er allerdings sehr nützlich.

7 Darstellung und Diskussion der Ergebnisse

In diesem Kapitel sollen die Projektionen einiger Probleme aus der Praxis aufgezeigt werden. Die Projektionen der verschiedenen Probleme sind sehr unterschiedlich. Diese sollen nun ein wenig beschrieben werden.

In den folgenden Abbildungen sind die Projektionen der einzelnen Probleme dargestellt. Dabei kann man an der x-Achse die Zielfunktionswerte der ersten Zielfunktion und an der y-Achse die Zielfunktionswerte der zweiten Zielfunktion ablesen.

Die Punkte sind die Projektionen der während der Berechnung der Projektion berechneten Ecken. Dabei sind alle Punkte der Projektionen, die eine Ecke darstellen, die Projektionen von Schattenecken. Alle übrigen Punkte sind die Projektionen von Ecken, die aufgrund der Degeneriertheit berechnet worden sind.

Leider standen mir keine Probleme zur Verfügung, die von vornherein zwei Zielfunktionen aufwiesen. So habe ich auf verschiedene Weisen eine zweite Zielfunktion konstruiert.

Zum einen wird als zweite Zielfunktion eine zufällige Zielfunktion erstellt, die allen Variablen einen zufälligen Zielfunktionskoeffizienten zwischen -1 und 1 zuweist. Weiterhin wird die zweite Zielfunktion auch durch eine einzelne Variable oder Ungleichung des Problems repräsentiert.

Dadurch daß die Zielfunktionskoeffizienten bei zufälligen zweiten Zielfunktion zwischen -1 und 1 liegen, kann es vorkommen, daß die Zielfunktionswerte der zweiten Zielfunktion dementsrechend gering gegenüber der ersten Zielfunktion sind.

Die Projektionen der hier betrachteten Probleme haben die verschiedensten Formen. Es gibt sowohl Probleme, deren Projektion spitz ist (7.1), fast rund ist (7.2), als auch eher rechteckige Projektionen (7.3), sowie unbeschränkte Projektionen (7.4).

Die Begriffe rund, spitz und rechteckig sind hier eher relativ zu sehen, denn die Werte auf den beiden Achsen, die die Zielfunktionen charakterisieren, sind nicht gleich.

Außerdem wird deutlich, daß es sowohl einfach strukturierte Probleme (Abbildung 13) gibt, so daß die Anzahl der Schattenecken gering ist, als auch Probleme mit sehr vielen Schattenecken (Abbildung 26).

Degeneriertheit kann man immer dann erkennen, wenn auf einer Strecke die Projektionen mehrerer Ecken zu sehen sind (Abbildung 33).

In Tabelle 1 wird angegeben, wie die einzelnen, hier aufgeführten Probleme

strukturiert sind, sprich aus wieviel Gleichungen, Ungleichungen und Variablen sich die einzelnen Probleme zusammensetzen.

Ranges gibt die Anzahl der Zeilen an, die sowohl eine obere als auch eine untere Schranke aufweisen.

Die Anzahl der berechneten Ecken gibt die Anzahl der verschiedenen Projektionen von Ecken der jeweiligen Projektion an. Bei der Projektion von unbeschränkten Problemen gibt diese Zahl die Anzahl der berechneten Projektionen von Ecken an, die errechnet wurden, bevor Unbeschränktheit festgestellt worden ist.

Um einen Vergleichswert für die berechneten Projektionen zu erhalten, wurden zufällige Probleme erstellt.

Da die zufälligen Probleme im allgemeinen aber voll besetzt sind, erfordern sie einen großen Speicherbedarf. Somit können sie nur als Vergleichswert für die kleineren Probleme herangezogen werden.

Ein einfach strukturiertes Problem ist das Problem *afiro* (Abbildung 13). Für die Projektion wurden lediglich 12 Ecken berechnet. Bildet man zufällige Probleme derselben Größe, sprich mit derselben Anzahl von Gleichungen, Ungleichungen und Variablen kommt man zu folgendem Ergebnis.

Im Durchschnitt wurden bei der Berechnung der Projektionen der zufälligen Probleme 54 Ecken berechnet. Die Projektionen der zufälligen Probleme sind gleichmäßiger als die Projektion des Problems *afiro*. Während die Projektion von *afiro* relativ spitz ist, sind die Projektionen der zufälligen Probleme eher rund. Abbildung 14 zeigt die Projektionen zweier dieser zufälligen Probleme.

Die 12 Ecken der Projektion von *afiro* bedeuten folgendes. Bildet man aus den beiden bereits gegebenen Zielfunktionen durch Linearkombination eine neue Zielfunktion, so befindet sich eine optimale Lösung unter den bereits berechneten Ecken, d.h. für jede Zielfunktion, die sich als Linearkombination der beiden gegebenen Zielfunktionen darstellen läßt, ist eine der 12 Ecken eine Optimallösung.

Die Anzahl der berechneten Ecken gibt also an, aus wievielen und vor allem aus welchen Ecken eine Optimallösung berechnet werden kann, wenn man nach einer Zielfunktion optimiert, die sich als Linearkombination der beiden Zielfunktionen darstellen läßt.

Das komplexeste, hier betrachtete Problembeispiel ist das Problem *fit1d* (Abbildung 26). Es besitzt 4188 verschiedene Ecken. Die Projektion dieses Problems ist am ehesten rund. Daran erkennt man, daß beim leichten Verändern der zu optimierenden Zielfunktion, wenn diese sich auf $\text{span}(c, \bar{c})$ befindet, sich

Probleme	Gleichungen	Ungleichungen	Rangres	Variable	zweite Zielfkt	ber. Ecken	Abb. Nr.
adlittle	15	41	0	97	zufällig	146	34
afiro	8	19	0	32	zufällig	13	13
agg	36	452	0	163	zufällig	185	21
agg2	60	456	0	302	zufällig	496	22
agg3	60	456	0	302	zufällig	510	23
beaconfd	140	33	0	262	0.Var	3	35
blend	43	31	0	83	0.Var	40	36
boeing1	9	253	89	384	zufällig	1238	24
boeing2	4	143	19	143	zufällig	233	25
bore3d	214	19	0	315	0.Var	24	37
brandy	166	54	0	249	0.Var	318	38
capri	142	129	0	353	0.Var	123	39
e226	33	190	0	282	9.Zeile	2607	31
etamacro	272	128	0	688	zufällig	986	32
fffff800	350	174	0	854	zufällig	370	17
finnis	47	450	0	614	1.Var	23	40
fit1d	1	23	0	1026	zufällig	4188	26
forplan	90	70	1	421	zufällig	347	27
grow7	140	0	0	301	zufällig	1288	28
israel	0	174	0	142	zufällig	418	41
kb2	16	27	0	41	zufällig	84	16
lotfi	95	58	0	308	0.Var	239	42
recipe	67	24	0	180	46.Var	47	33
sc105	45	60	0	103	zufällig	101	18
sc50a	20	30	0	48	zufällig	44	19
sc50b	20	30	0	48	zufällig	31	20
scagr7	84	45	0	140	0.Var	101	43
scagr25	300	171	0	500	zufällig	394	44
scfxm1	87	143	0	457	0.Var	102	45
share1b	89	28	0	225	zufällig	394	29
share2b	13	83	0	79	zufällig	137	30
stocfor1	63	54	0	111	1.Var	23	46
vtp.base	55	143	0	203	zufällig	47	47

Tabelle 1: Übersicht der betrachteten Probleme

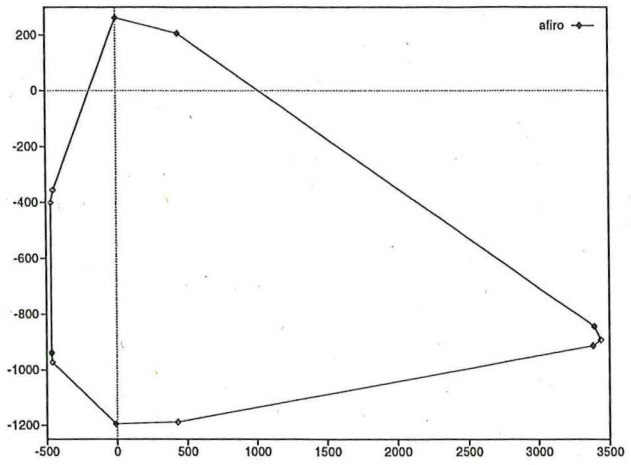

Abbildung 13: Problem afiro

die optimale Ecke ebenfalls verändert.

Vergleicht man das Problem *share2b* (Abbildung 30) mit zufälligen Problemen, so stellt man auch hier fest, daß die Anzahl der berechneten Ecken bei *share2b* (137) deutlich unter der der zufälligen Probleme (im Durchschnitt 258) liegt. Die Form der Projektion von *share2b* ähnelt aber im Gegensatz zum vorherigen Beispiel stark den Projektionen der zufälligen Probleme (Abbildung 15).

Vergleicht man die Probleme weiter, so sieht man, daß die Projektionen im allgemeinen ungleichmäßig sind, d.h. der Abstand der benachbarten Ecken der Projektionen zueinander ist nicht gleichmäßig. Zum einen gibt es eine Anhäufung von Ecken und zum anderen sind einige Ecken "isoliert".

Gute Beispiele dafür sind die Projektionen der Probleme *boeing2* (Abbildung 25), *etamacro* (Abbildung 32), *forplan* (Abbildung 27) und *share2b* (Abbildung 30).

Wählt man als zweite Zielfunktion nicht eine zufällige Zielfunktion sondern eine einzelne Variable aus, nach der man optimiert, so sehen die Projektionen gleich ganz anders aus.

Es fällt auf, daß die so erhaltenen Projektionen stark degeneriert sind, d.h. auf einer Strecke befinden sich die Projektionen etlicher Ecken.

Die Projektion des Problembeispiels *recipe* (Abbildung 33) zeigt dies deutlich. Hieraus läßt sich ersehen, daß diese Probleme etliche Lösungen besitzt, bezüglich derer die entsprechende Variable der zweiten Zielfunktion optimal ist.

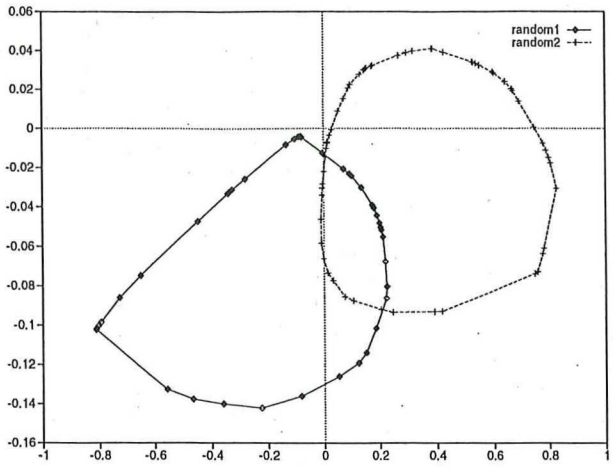

Abbildung 14: Zufällige Probleme für afiro

Aber auch bei anderen Zielfunktionen kann man solch eine Projektion beobachten. Ein Beispiel dafür ist die Projektion von *e*226 (Abbildung 31). Hier entspricht die zweite Zielfunktion der neunten Zeile der Nebenbedingungen des Problems.

Jetzt soll noch angegeben werden (Tabelle 2), welche der Probleme unbeschränkte Projektionen aufweisen, und für welche Zielfunktion bei der Maximierung Unbeschränktheit festgestellt worden ist.

Die im folgenden angegebenen Probleme sind in vier Klassen eingeteilt worden. Zuerst werden die Projektionen der Probleme angegben, die eine spitze Projektion aufweisen, danach werden die runden Projektionen gezeigt, anschließend die rechteckigen, bevor schließlich die Projektionen der unbeschränkten Probleme aufgelistet werden.

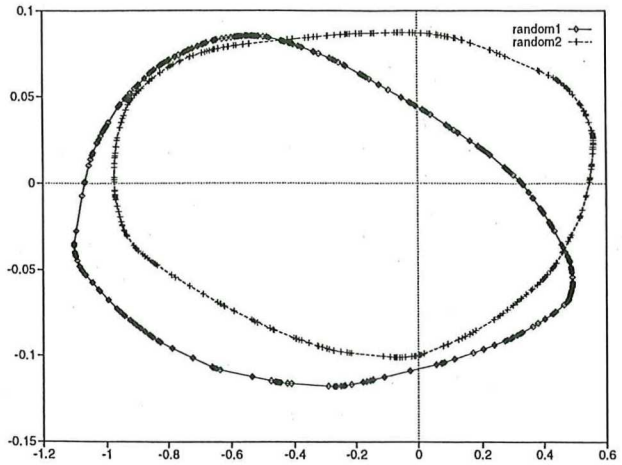

Abbildung 15: Zufällige Probleme für share2b

Probleme	unbeschränkte Zielfunktionen
adlittle	$-c$ und $\overline{c}$
beaconfd	$-c$
blend	$-c$
bore3d	$-c$
brandy	$-c$
capri	$-c$
finnis	$-c$
israel	$-c$
lotfi	$-c$ und $\overline{c}$
scagr7	$-c$
scagr25	$-c$ und $\overline{c}$
scfxm1	$-c$
stocfor1	$-c$
vtp.base	$-c$ und $-\overline{c}$

Tabelle 2: Übersicht der unbeschränkten Probleme

7.1 Spitze Projektionen

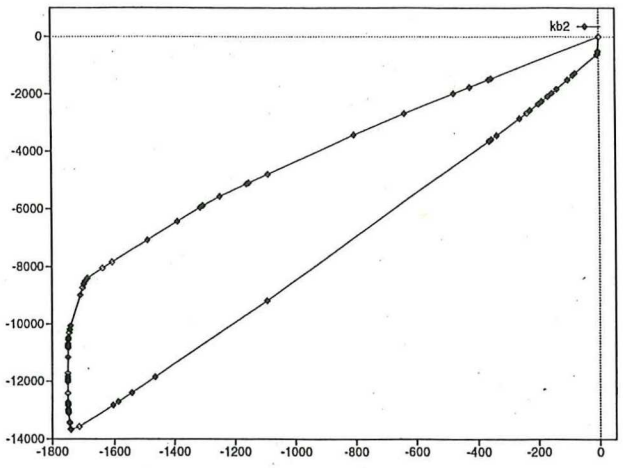

Abbildung 16: Problem kb2

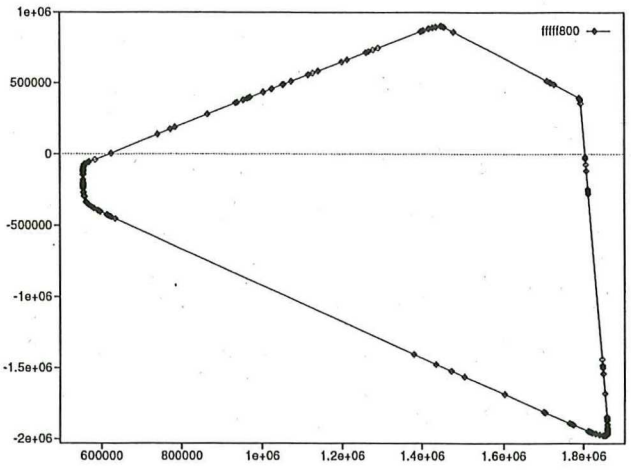

Abbildung 17: Problem fffff800

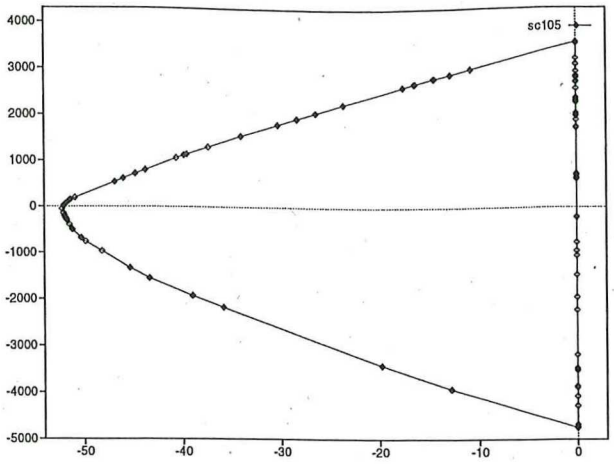

Abbildung 18: Problem sc105

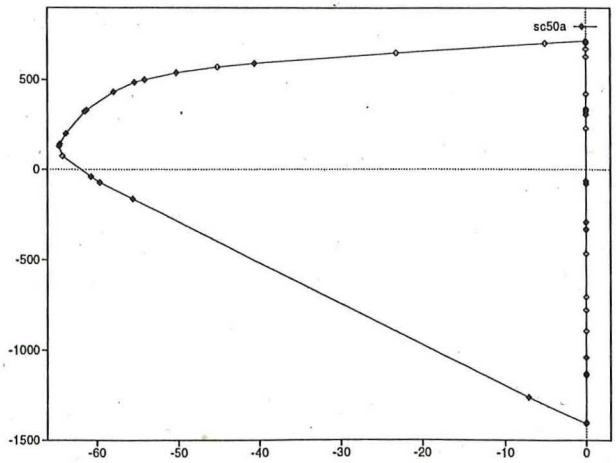

Abbildung 19: Problem sc50a

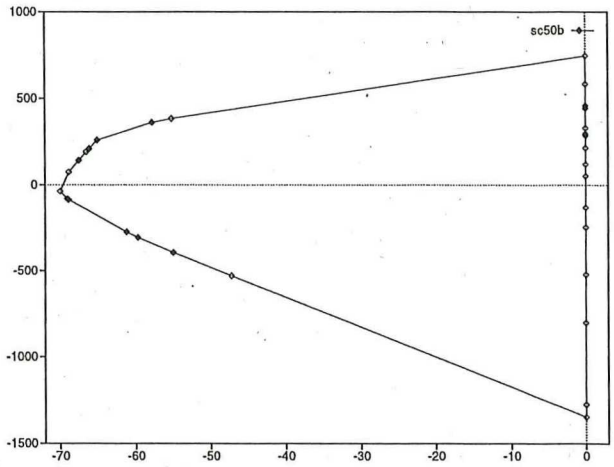

Abbildung 20: Problem sc50b

7.2 Runde Projektionen

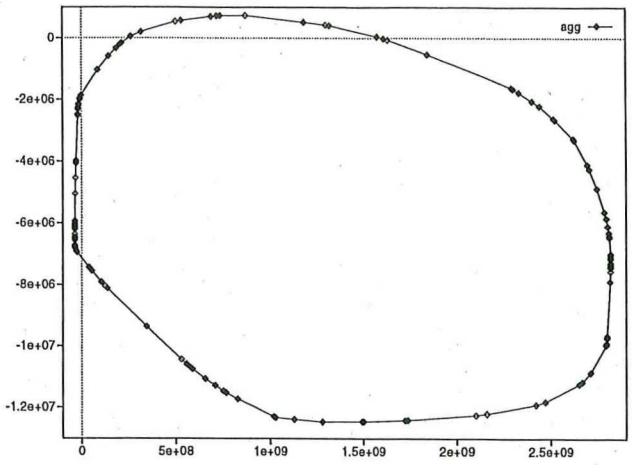

Abbildung 21: Problem agg

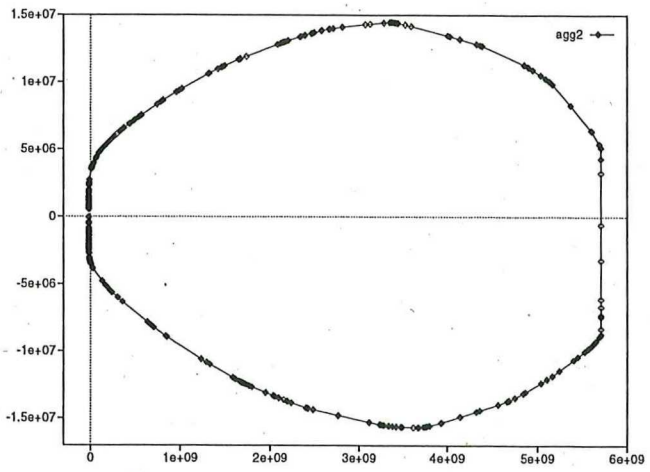

Abbildung 22: Problem agg2

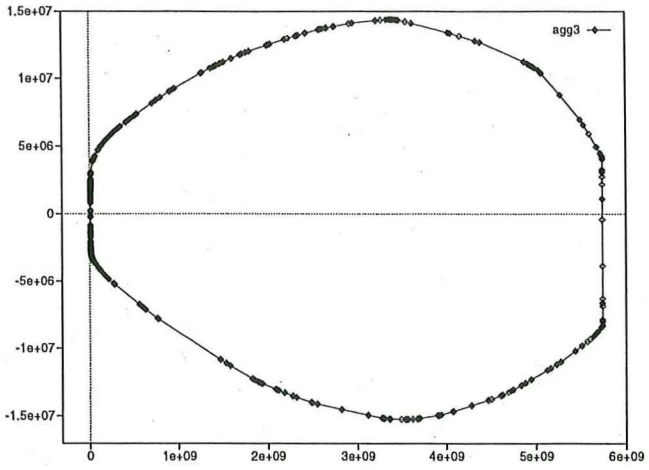

Abbildung 23: Problem agg3

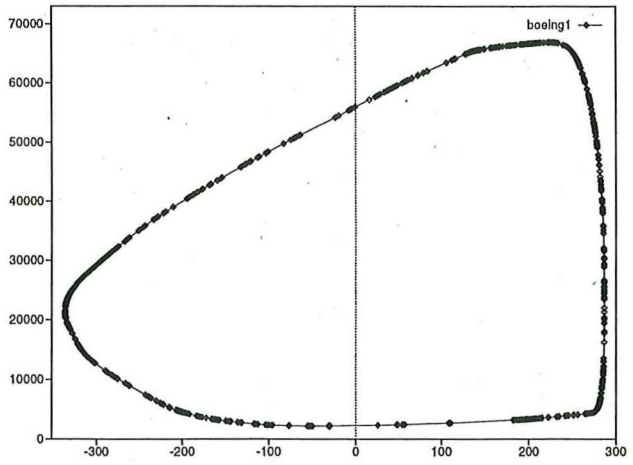

Abbildung 24: Problem boeing1

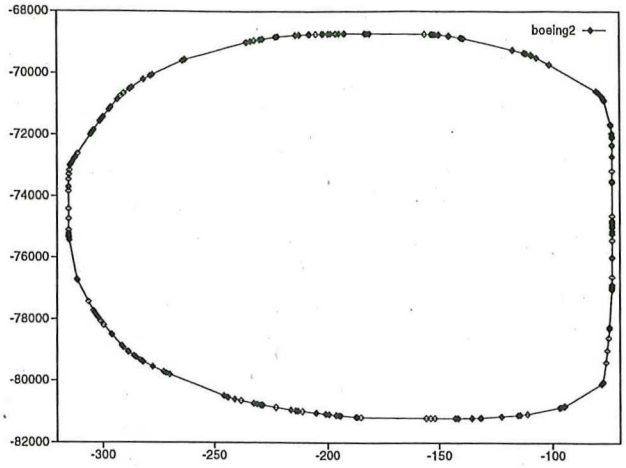

Abbildung 25: Problem boeing2

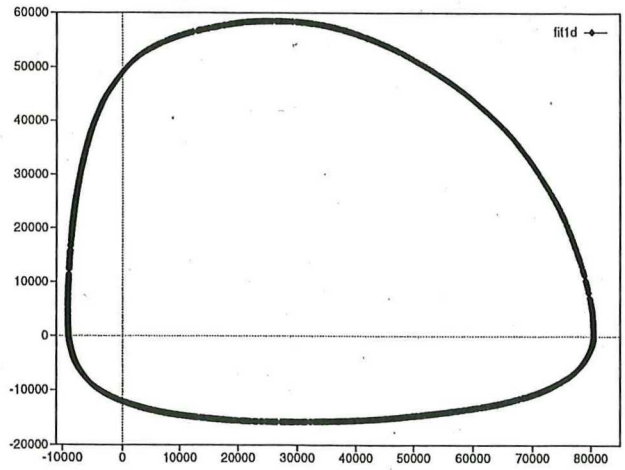

Abbildung 26: Problem fit1d

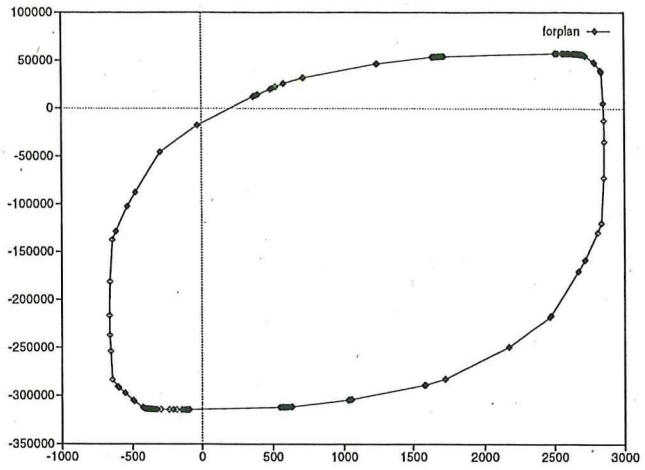

Abbildung 27: Problem forplan

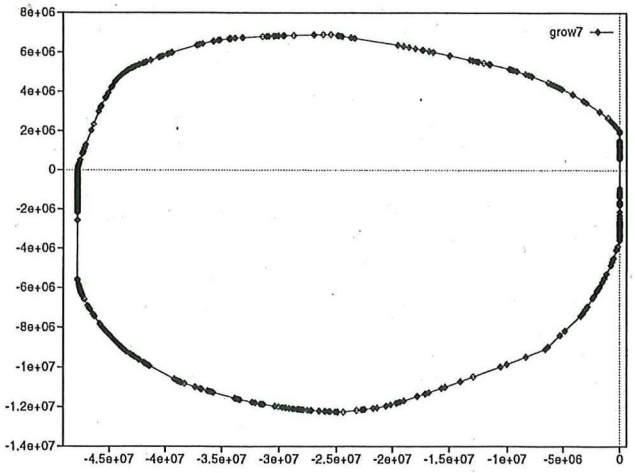

Abbildung 28: Problem grow7

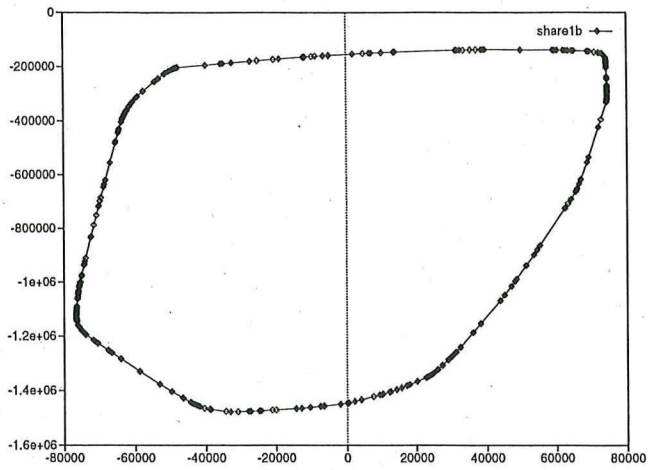

Abbildung 29: Problem share1b

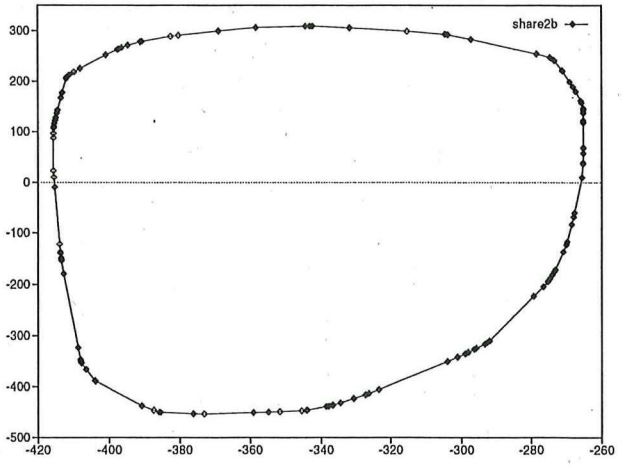

Abbildung 30: Problem share2b

7.3 Rechteckige Projektionen

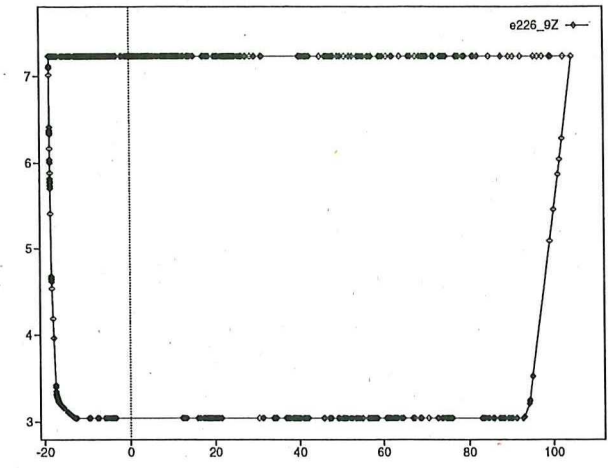

Abbildung 31: Problem e226

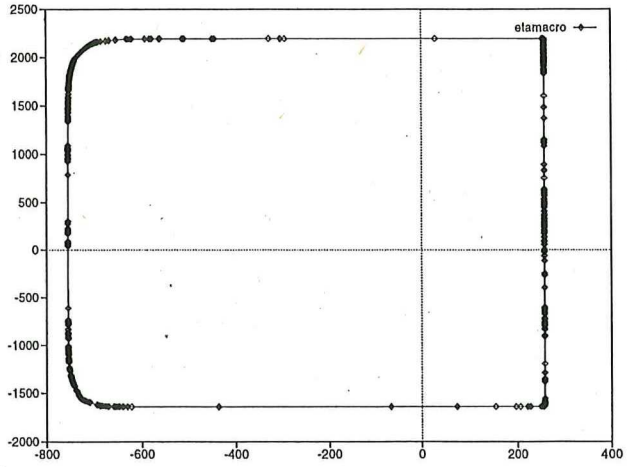

Abbildung 32: Problem etamacro

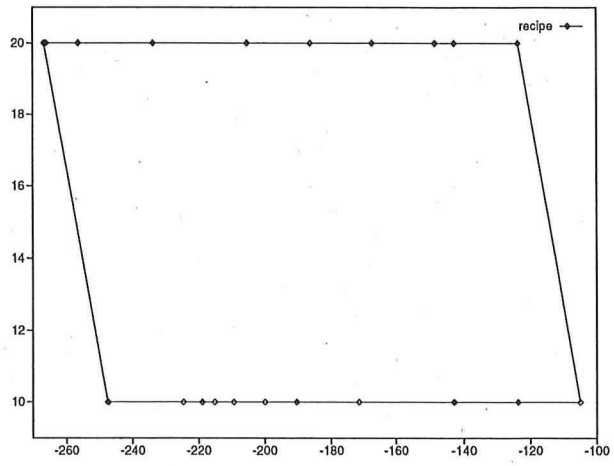

Abbildung 33: Problem recipe

7.4 Unbeschränkte Projektionen

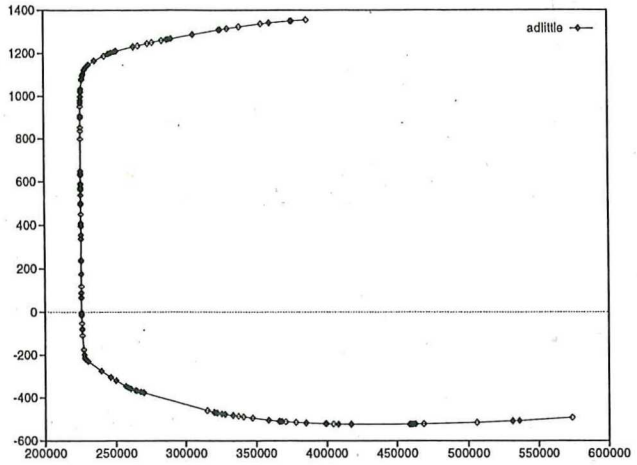

Abbildung 34: Problem adlittle

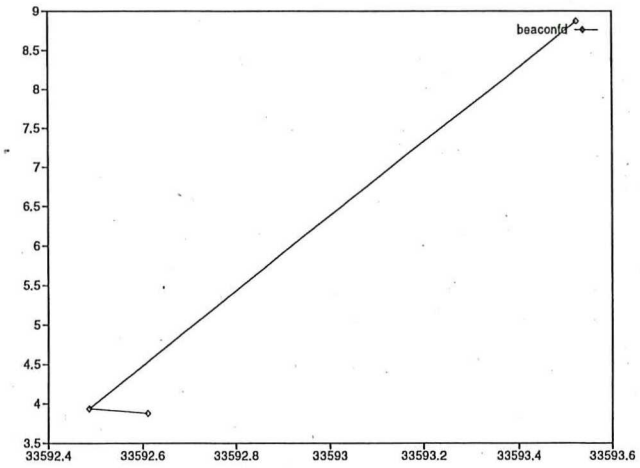

Abbildung 35: Problem beaconfd

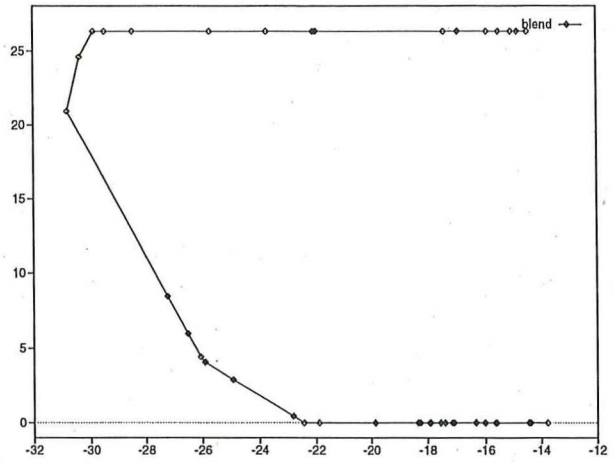

Abbildung 36: Problem blend

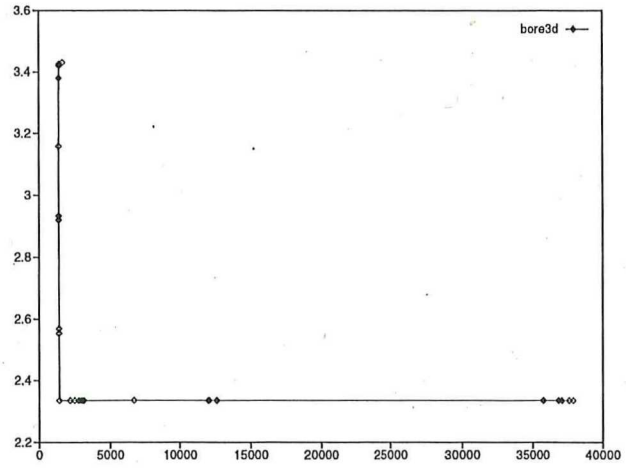

Abbildung 37: Problem bore3d

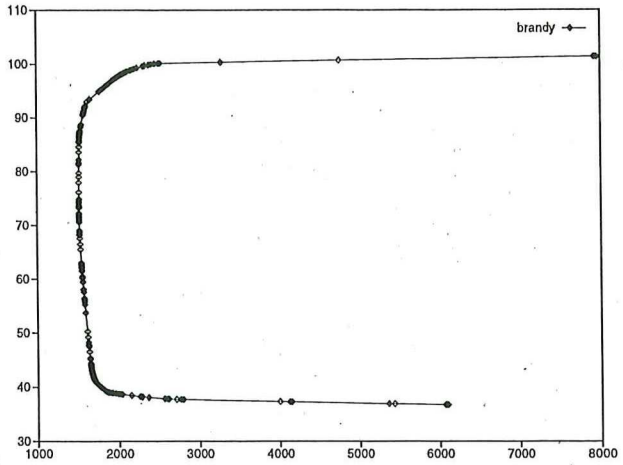

Abbildung 38: Problem brandy

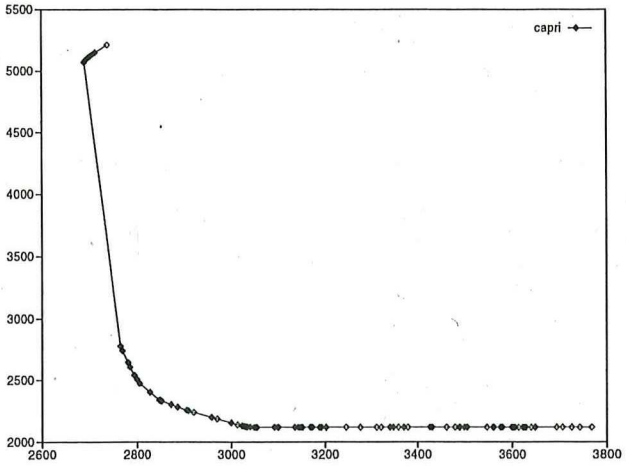

Abbildung 39: Problem capri

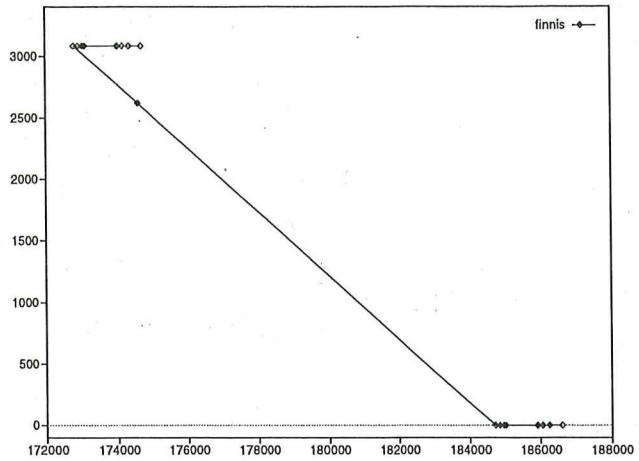

Abbildung 40: Problem finnis

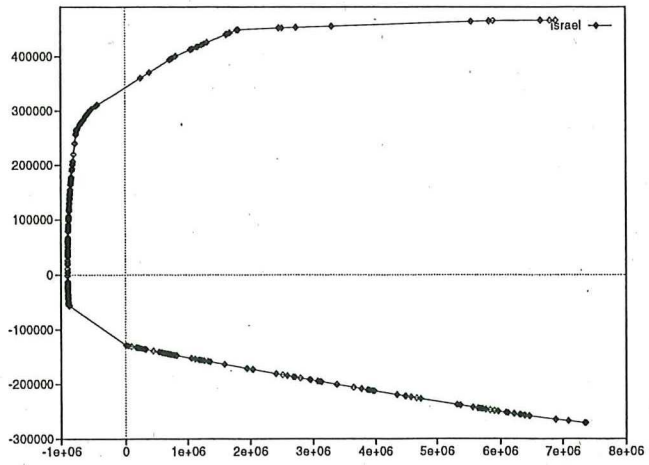

Abbildung 41: Problem israel

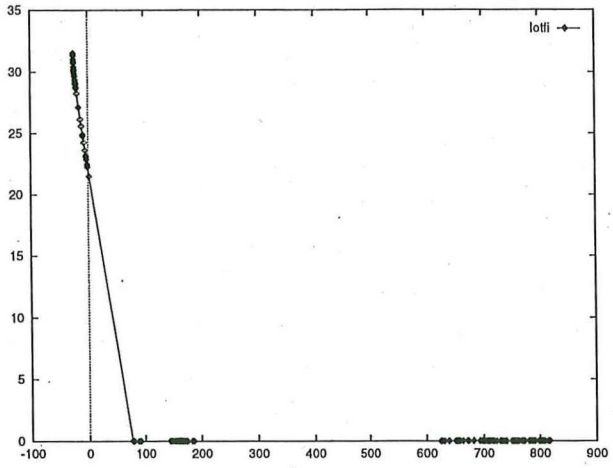

Abbildung 42: Problem lotfi

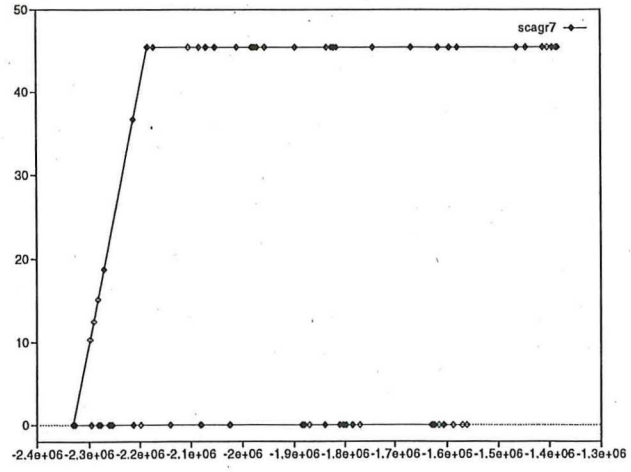

Abbildung 43: Problem scagr7

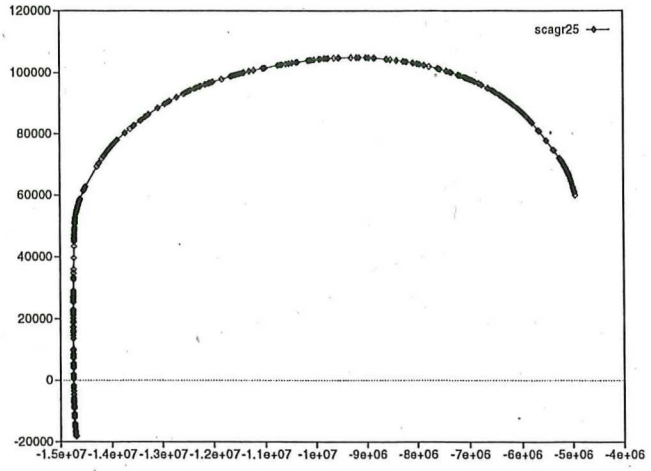

Abbildung 44: Problem scagr25.eps

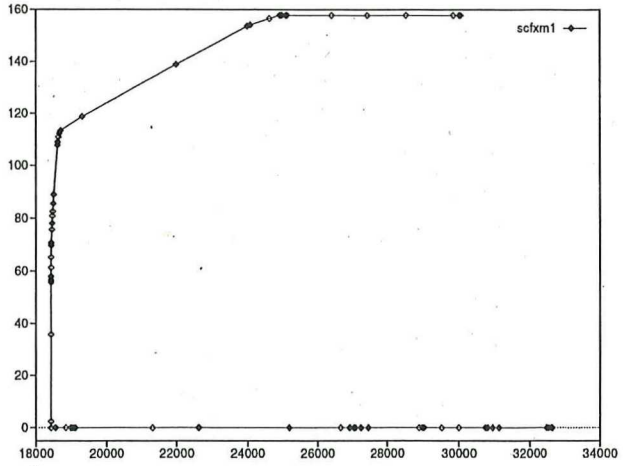

Abbildung 45: Problem scfxm1

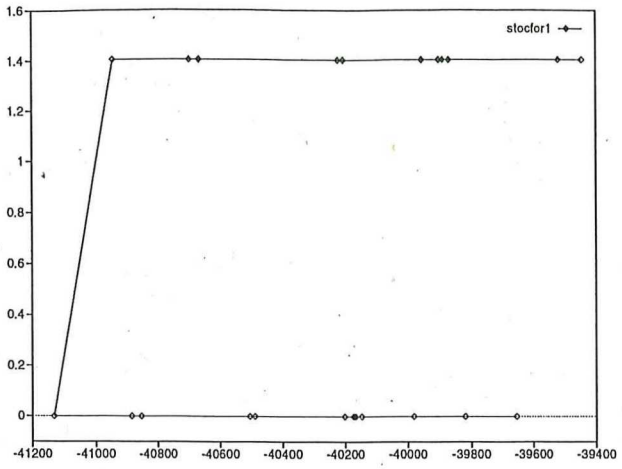

Abbildung 46: Problem stocfor1

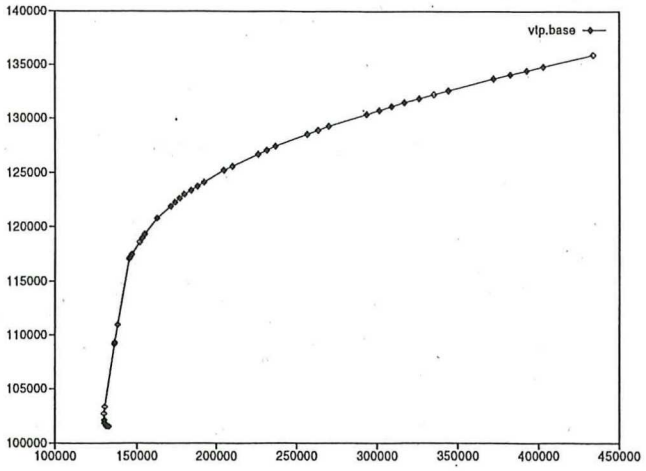

Abbildung 47: Problem vtp.base

Literatur

[1] Robert E. Bixby, Das Implementieren des Simplex-Verfahrens: Die Startbasis, Preprint SC 92-11 (April 1992) Konrad-Zuse-Zentrum für Informationstechnik Berlin

[2] M.Grötschel, Skript Lineare Optimierung, Universität Augsburg, 1985

[3] M.Grötschel, L.Lovász, A.Schrijver, Geometric Algorithm and Combinatorial Optimization, Springer Verlag Berlin, 1988

[4] Victor Klee and Peter Kleinschmidt, Geometry of the Gass-Saaty Parametric Cost LP Algorithm

[5] K.Murty, Linear Programming, Wiley, New York, 1983

[6] David F.Shanno, Computational Methods for Linear Programming, Rutcor Research Report RRR 19-93, September, 1993

[7] Roland Wunderling, Paralleler und Objektorientierter Simplex Algorithmus, Technical Report TR 96-6 (Dezember 1996), ZIB Berlin

Die selbständige u. eigenhändige Anfertigung versichert an Eidesstatt
10623 Berlin, den 26.2.98

Fischer

Eingereicht am 26.2.98 im Referat für Studienangelegenheiten

i. A. Hartz