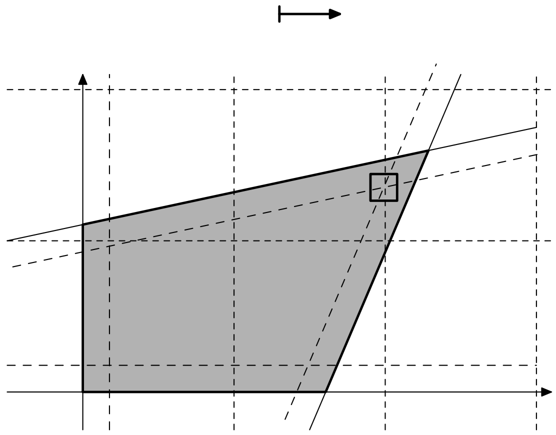
LURUPA

Rigorose Fehlerschranken für Lineare Programme



Diplomarbeit von

Christian Keil

Betreuer

Dr. Christian Jansson

an der

Technischen Universität Hamburg–Harburg,
AB Technische Informatik III

10. Februar 2004

Inhaltsverzeichnis

Kapitel 1

Einführung

Optimierung ist eine Disziplin, die für sehr viele Bereichen von Bedeutung ist. Aufgaben sollen, sobald einmal gelöst, so schnell, billig, effizient wie möglich bearbeitet werden. Alle diese Anforderungen lassen sich als Optimierungsprobleme auffassen, wobei die äußeren Umstände zusätzliche Bedingungen an die Lösung stellen. So lassen sich manche Lösungen auf Grund der Naturgesetze nicht realisieren, andere scheitern an der Endlichkeit verfügbarer Ressourcen.

Die Lineare Optimierung tritt als Spezialfall der Optimierung immer dann auf, wenn die Problemstellung und die Bedingungen lineare Zusammenhänge darstellen oder sich näherungsweise durch lineare Zusammenhänge darstellen lassen. Doch auch global optimale Punkte nichtlinearer Probleme lassen sich mit ihrer Hilfe finden, indem iterativ lineare Relaxationen der Probleme gelöst werden.

Gerade in den Ingenieurwissenschaften sind die Dimensionen der auftretenden Probleme so groß, dass eine Lösung nur auf dem Computer realisierbar ist. Es existieren verschiedene Programmpakete, die die Lösung solcher linearer und verwandter Optimierungsprobleme versprechen. Neben der unterschiedlichen Qualität der einzelnen Pakete stellt sich jedoch die Frage, inwieweit diese Art von Problemen überhaupt dazu geeignet ist, auf dem Computer gelöst zu werden.

In ihrer Arbeit [15] haben Ordonez und Freund festgestellt, dass 72% der in der Netlib lp library [12] enthaltenen Linearen Optimierungsprobleme schlecht konditioniert sind. Bei diesen Problemen ist also schon die Modellierung der Gestalt, dass leichte Fehler während der Lösung auf dem Computer zu beliebigen Ergebnissen führen können. Nach Anwendung der preprocessing Heuristiken des Lösers CPLEX 7.1 [8] zeigten nur noch 19% der Probleme diese Eigenschaft. Dies zeigt, wie wichtig zum einen die korrekte Modellierung Linearer Optimierungsprobleme ist, zum anderen aber auch, dass bei einem Fünftel dieser praktischen Probleme fragwürdig ist, wie aussagekräftig eine berechnete Lösung ist.

In den Arbeiten von Jansson [9] und Neumaier [13] werden Methoden vorgestellt, die rigorose Fehlerschranken für die Lösung solcher Probleme liefern. Unter Verwendung dieser Verfahren lässt sich die Qualität einer berechneten Lösung beurteilen und so erst entscheiden, ob man Schlussfolgerungen aus ihr ziehen oder weiteren Aufwand in eine bessere Lösung investieren sollte. In dieser Arbeit wird eine Modifikation des Verfahrens von Jansson präsentiert. Eine Konvergenzanalyse zeigt, dass dieses Verfahren bei hinreichend genauer Rechnung für Probleme mit endlicher Kondition in jedem Fall rigorose Schranken liefert. Um die Qualität der Schranken zu beurteilen, wurde das Verfahren in C++ implementiert und auf die Probleme der Netlib lp library angewendet. Es zeigt sich, dass das Verfahren bis auf wenige Ausnahmen scharfe Schranken im Falle endlicher Kondition liefert.

Weiteres Ziel dieser Arbeit ist, mit der Implementierung von Lurupa (Linear UndeRestimator Using Perturbed Approximations) ein modulares Softwarepaket zu schaffen, welches sich mit geringem Aufwand an unterschiedliche Löser für lineare Optimierungsprobleme anpassen lässt, um diese um die Fähigkeit der rigorosen Schrankenberechnung zu erweitern.

Im zweiten Kapitel folgen einige Hinweise zur Notation. Das dritte Kapitel enthält zur Motivation einige eingehende Betrachtungen über verifiziertes Rechnen und stellt den Zusammenhang zwischen den Beobachtungen von Ordonez und Freund und den hier verwendeten Algorithmen her. Kapitel vier befasst sich mit der Implementierung von Lurupa. Im fünften Kapitel werden Messergebnisse mit den implementierten Algorithmen und der Netlib präsentiert. Den Schluss bilden ein Fazit samt Ausblick.

Kapitel 2

Notation

In den folgenden Betrachtungen steht

$$fl(x)$$

für die Abbildung von x auf die Fließkommazahlen des Computers. Die Fließkommaoperationen werden durch

$$\begin{aligned}
x \oplus y &:= fl(x+y),\\
x \ominus y &:= fl(x-y),\\
x \otimes y &:= fl(x \cdot y),\\
x \oslash y &:= fl(x/y),
\end{aligned}$$

repräsentiert.

Die Relationen $=, \neq, \leq, \geq, <, >$ sind bei Vektoren und Matrizen elementweise zu verstehen. e steht für den Vektor dessen Komponenten alle gleich 1 sind, in Skalarprodukten kann hier auch

$$1^T x$$

stehen. Diese Schreibweise wird auch in Ungleichungen wie

$$x \geq 0$$

verwendet, die die Positivität der Komponenten von x fordert. Die Zeilen und Spalten einer Matrix sind mit $A_{i:}$ und $A_{:j}$ bezeichnet. Bei Verwendung einer Indexmenge I steht x_I für den Vektor der Komponenten von x, deren Index in I liegt. $A_{I:}$ bzw. $A_{:J}$ bezeichnet die Teilmatrizen aus den entsprechenden Zeilen bzw. Spalten von A.

Intervalle, Intervallvektoren und Intervallmatrizen werden mit Fettbuchstaben bezeichnet und enthalten alle Werte zwischen einer unteren und oberen Schranke:

$$\mathbf{x} := [\underline{x}, \overline{x}] = \{x \mid \underline{x} \leq x \leq \overline{x}\},$$
$$\inf\{\mathbf{x}\} = \underline{x},$$
$$\sup\{\mathbf{x}\} = \overline{x}.$$

Sie bilden die Mengen $\mathbb{IR}, \mathbb{IR}^n, \mathbb{IR}^{m \times n}$. Relationen sind für Intervallgrößen erfüllt, wenn sie für alle Werte im Intervall erfüllt sind. So ist

$$\mathbf{a} \leq \mathbf{b}$$

genau dann, wenn

$$\sup\{\mathbf{a}\} \leq \inf\{\mathbf{b}\}$$

gilt. Intervalloperationen sind als Verallgemeinerung der reellen Operationen zu verstehen. Das Ergebnis ist das kleinste Intervall, das alle Ergebnisse der reellen Operation zwischen je zwei Elementen aus den beteiligten Intervallen enthält, d.h. mit $\circ \in \{+, -, \cdot, /\}$ gilt

$$\mathbf{a} \circ \mathbf{b} := \bigcap \{\mathbf{c} \in \mathbb{IR} : \ a \circ b \in \mathbf{c} \text{ für alle } a \in \mathbf{a},\ b \in \mathbf{b}\}.$$

Kapitel 3

Theorie

3.1 Fehler beim Rechnen

Bei der Umsetzung von Algorithmen auf dem Computer können verschiedene Effekte auftreten, die numerische Berechnungen verfälschen [4], [11]. Sie sorgen dafür, dass die Ergebnisse nur bedingt zuverlässig sind und als Grundlage für rigorose Betrachtungen ausscheiden. Ursache hierfür ist die endliche Fließkommadarstellung der Werte im Computer, die an verschiedenen Stellen zu Fehlern führen kann. Maßgeblich für die zu erwartenden Fehler sind zum einen die kleinste darstellbare positive Zahl η_A und ε_A, die kleinste Zahl, die zu 1 addiert ein von 1 verschiedenes Ergebnis liefert,

$$\varepsilon_A := \min\{x \mid 1 \oplus x \neq 1\}.$$

Die folgenden Beispiele gehen von einer dreistelligen Fließkommaarithmetik mit einem Exponentenbereich von -5 bis 5 aus. Die Darstellung ist normalisiert, so dass die erste Ziffer der Mantisse von 0 verschieden ist. Rundung ist kaufmännisch, 1–4 wird abgerundet, 5–9 aufgerundet. Hier wäre

$$\eta_A = 1.00 \cdot 10^{-5}$$
$$\varepsilon_A = 5.00 \cdot 10^{-3}$$

$fl(x)$ steht für die Darstellung von x im Computer,

$$fl(0.1234) = 1.23 \cdot 10^{-1}$$
$$fl(23.45) = 2.35 \cdot 10^{1}$$

3.1.1 IEEE

Um eine einheitliche Fehlerabschätzung auf verschiedenen Architekturen zu ermöglichen und die Portabilität von Algorithmen zu erhöhen, hat die IEEE

[14] mehrere Standards für Arithmetiken veröffentlicht. Für Fließkommadarstellung sind dies die Standards IEEE 754 und 854 [5], [18], [20]. Diese legen neben der Darstellung von Mantisse und Exponent einer Fließkommazahl in einfacher und doppelter Genauigkeit zum Beispiel Regeln für die Rundung von Operationen fest. So sind einfache Operationen, wie Addition, Subtraktion, Multiplikation und Quadratwurzel exakt, also auf die, dem exakten Ergebnis nächste Fließkommazahl zu runden. Hierbei ist *round to even* zu verwenden, wenn die zu rundende Zahl exakt zwischen zwei Fließkommazahlen liegt, ist so zu runden, dass die letzte Ziffer des Ergebnisses gerade ist. So wäre

$$fl(1.005) = 1.00 \cdot 10^0,$$
$$fl(1.015) = 1.02 \cdot 10^0.$$

Eine Begründung für diese Art der Rundung liefern Reiser und Knuth [17]. Bei komplizierteren Operationen, wie zum Beispiel den transzendenten Funktionen, besteht keine Forderung nach exakter Rundung.

3.1.2 Eingabefehler

Schon bei der Eingabe von Daten in den Computer treten Fehler auf. So lässt sich die Zahl $\frac{1}{10} = 0.\overline{000101}_2$ als nicht abbrechender Binärbruch nicht exakt auf dem Rechner darstellen. Es wird eine Approximation x gespeichert, deren relativer Fehler den Betrag

$$\left| \frac{fl(\frac{1}{10}) - \frac{1}{10}}{fl(\frac{1}{10})} \right| \leq \varepsilon_A$$

hat. Hierbei ist noch nicht berücksichtigt, dass die Eingabedaten selbst fehlerbehaftet sein können, wenn sie beispielsweise aus Messungen oder anderen Rechenverfahren resultieren.

3.1.3 Rundungsfehler

Auch die Ergebnisse von arithmetischen Operationen sind fehlerbehaftet. Sind die exakten Ergebnisse nicht auf dem Rechner darstellbar, weil die Genauigkeit nicht ausreicht, so lassen sich auch hier nur Approximationen speichern. So ist zum Beispiel

$$1.00 \cdot 10^0 \oplus 1.00 \cdot 10^{-3} = fl(1.001 \cdot 10^0) = 1.00 \cdot 10^0.$$

Es lässt sich nur die genäherte Lösung $1.00 \cdot 10^0$ speichern. Ähnliches gilt für Subtraktion, Multiplikation und Division. Auch hier hat der relative Fehler den maximalen Betrag ε_A. Für komplexere Operationen kann dieser, wie schon erwähnt, noch größer sein. Wiederum ist nicht berücksichtigt, dass die Operanden selbst nicht exakt sein müssen.

Durch diese Fehler werden unter Umständen fundamentale Gesetze der Mathematik verletzt. So verletzt

$$0 = (1.00 \cdot 10^0 + 1.00 \cdot 10^5) - 1.00 \cdot 10^5 \neq 1.00 \cdot 10^0 + (1.00 \cdot 10^5 - 1.00 \cdot 10^5) = 1$$

die Assoziativität der Addition.

Verlässt das Ergebnis den darstellbaren Zahlenbereich, treten weitere Probleme auf. Dividiert man die betragsmäßig kleinste darstellbare Zahl beispielsweise durch 4, so tritt ein Unterlauf auf. Das Ergebnis wird 0 mit einem absoluten Fehler in Größenordnung der Lösung selbst:

$$1.00 \cdot 10^{-5} \oslash 4 = fl(2.50 \cdot 10^{-6}) = 0.$$

Auf diese Weise kann die Kommutativität der Multiplikation verletzt werden:

$$1.00 \cdot 10^{-5} = (2 \otimes 1.00 \cdot 10^{-5}) \oslash 4 \neq 2 \otimes (1.00 \cdot 10^{-5} \oslash 4) = 0$$

Wird der darstellbare Zahlenbereich dagegen überschritten, kommt es zu einem Überlauf, ist das Ergebnis in der Regel unbrauchbar. Es lassen sich keine weiteren Resultate daraus ableiten.

3.1.4 Verifiziertes Rechnen

So verstärken sich die Eingabe- und Rundungsfehler im Verlauf eines Algorithmus weiter. Schätzt man die Rundungsfehler in jedem Schritt ab, lässt sich der Fehler des Endergebnisses abschätzen. Man spricht von einer *Vorwärtsanalyse.* Eine Möglichkeit, diese vom Computer durchführen zu lassen, ist die *Intervallarithmetik.* Hierbei wird durch geeignetes Verändern des verwendeten Rundungsmodus ein Intervall bestimmt, welches das exakte Ergebnis garantiert enthält. Im obigen Beispiel der Addition würde hier einmal auf- und einmal abgerundet mit dem Ergebnis

$$1.00 \cdot 10^0 + 1.00 \cdot 10^{-3} \in [1.00 \cdot 10^0, 1.01 \cdot 10^0].$$

Durch die Eigenschaft der Verifiziertheit eignet sich die Intervallarithmetik als Grundlage für rigorose Berechnungen.

In den restlichen Betrachtungen dieser Arbeit wird davon ausgegangen, dass die Arithmetik der zugrundeliegenden Architektur nach IEEE 754 arbeitet. Somit gilt insbesondere

$$x \oplus y = (x + y)(1 + \varepsilon) \tag{3.1}$$
$$x \ominus y = (x - y)(1 + \varepsilon) \tag{3.2}$$

mit $|\varepsilon| \leq \varepsilon_A$.

3.2 Lineares Programm

Ein Lineares Programm (*LP*) ist kein Programm im informatischen Sinne. Programm ist hier als Planung zu verstehen [7]. Es ist ein Lineares Optimierungsproblem. Zur Theorie vergleiche [3], [6], [4]. Ziel ist die Minimierung der linearen *Zielfunktion*, die aus dem Skalarprodukt der *Variablen* x und der *Kosten* c besteht

$$\min c^T x \qquad \text{mit } c, x \in \mathbb{R}^n, \tag{3.3}$$

unter Einhaltung der linearen *Nebenbedingungen*

$$Bx = b \qquad \text{mit } B \in \mathbb{R}^{p \times n},\ b \geq 0 \in \mathbb{R}^p. \tag{3.4}$$

Dabei sind die *einfachen Schranken* für die Variablen x einzuhalten:

$$x \geq 0. \tag{3.5}$$

Der *Zulässigkeitsbereich* umfasst alle Punkte, die die Nebenbedingungen und einfachen Schranken erfüllen:

$$F := \{x \in \mathbb{R}^n \mid Bx = b, x \geq 0\}. \tag{3.6}$$

Ein Punkt $x \in F$ heißt *zulässiger Punkt*. Dies ist die *Standardform* eines LPs, weitere Formen enthalten Ungleichungen mit oberen und unteren Schranken als Nebenbedingungen, andere obere und untere einfache Schranken. Die allgemeinste Form stellt sich so dar:

$$\begin{aligned} &\min c^T x \\ \text{unter } &\underline{a} \leq Ax \leq \overline{a} \\ &Bx = b \\ &\underline{x} \leq x \leq \overline{x} \\ &\text{mit } A \in \mathbb{R}^{m \times n},\ \underline{a}, \overline{a} \in \mathbb{R}^m,\ \underline{x}, \overline{x} \in \mathbb{R}^n. \end{aligned} \tag{3.7}$$

Hier fehlt auch die Forderung nach der Positivität der rechten Seiten der Gleichungen. Variable mit unendlicher oberer und unterer Schranke heißen *freie Variable*, solche mit gleicher oberer und unterer Schranke heißen *fixiert*. Ungleichungsnebenbedingungen mit oberer und unterer Schranke $\underline{a}, \overline{a}$ heißen *ranges*. Maximieren der Zielfunktion lässt sich durch Multiplikation der Kosten mit -1, Minimierung und anschließende Multiplikation des minimalen Zielfunktionswertes mit -1 erreichen,

$$\max c^T x = -\min(-c)^T x$$

Alle Formen sind äquivalent, lassen sich ineinander überführen.

Jedem LP lässt sich ein anderes, sein *duales* LP, zuordnen. Wiederum für die Standardform hat es folgende Gestalt:

$$\begin{aligned} &\max b^T z \\ &\text{unter } B^T z + u = c \\ &\qquad u \geq 0 \\ &\qquad \text{mit } z \in \mathbb{R}^p, u \in \mathbb{R}^n. \end{aligned} \tag{3.8}$$

Wegen

$$b^T z = x^T B^T z \leq x^T (B^T z + u) = x^T c$$

sind die Zielfunktionswerte des dualen Problems stets kleiner als die des ursprünglichen, *primalen*. Die Nebenbedingungen des dualen Problems erinnern hierbei an die Kuhn–Tucker Bedingungen für die Optimalität eines Punktes des primalen Problems. Sie besagen, dass der Gradient der Zielfunktion im linearen Erzeugnis der Gradienten der *aktiven*, also mit Gleichheit erfüllten Nebenbedingungen liegen muss. Die Variablen des dualen Problems sind somit die Lagrange-Parameter. Ihre Vorzeichenbedingungen resultieren aus der unterschiedlichen Richtung der Gradienten der Nebenbedingungen. So wirkt eine $\geq$-Nebenbedingung, zum Beispiel die einfache untere Schranke für eine Variable, als untere Begrenzung in den Zulässigkeitsbereich hinein und damit in Richtung ihres Gradienten, der Lagrange-Parameter ist positiv. Eine $\leq$-Nebenbedingung wirkt ihrem Gradienten entgegen, womit der Lagrange-Parameter negativ wird. Der Lagrange-Parameter einer Gleichung ist dementsprechend nicht vorzeichenbeschränkt.

Die Lösbarkeit eines LPs hängt von der Beschaffenheit des Zulässigkeitsbereiches ab. Ist dieser in der Optimierungsrichtung, der Richtung des negativen Gradienten $-c$, unbeschränkt, so ist auch das LP selbst *unbeschränkt* (Abbildung 3.1). Die gestrichelten Linien in der Abbildung zeigen die Richtungen der Ungleichungen an. Ist F dagegen gleich der leeren Menge, so ist das Problem *unzulässig* (Abbildung 3.2). Ansonsten ist F als Schnitt der kon-

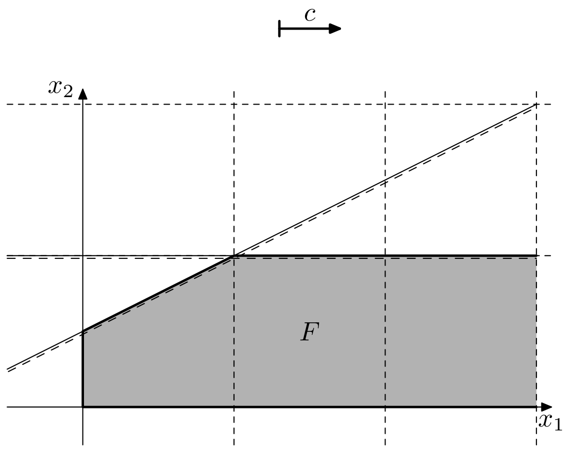


Abbildung 3.1: Unbeschränkter Zulässigkeitsbereich

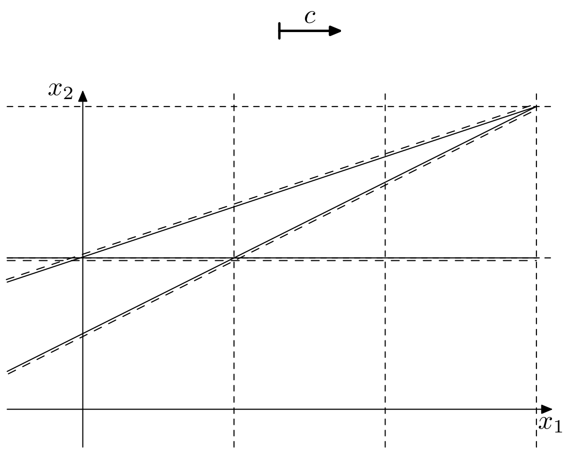


Abbildung 3.2: Leerer Zulässigkeitsbereich

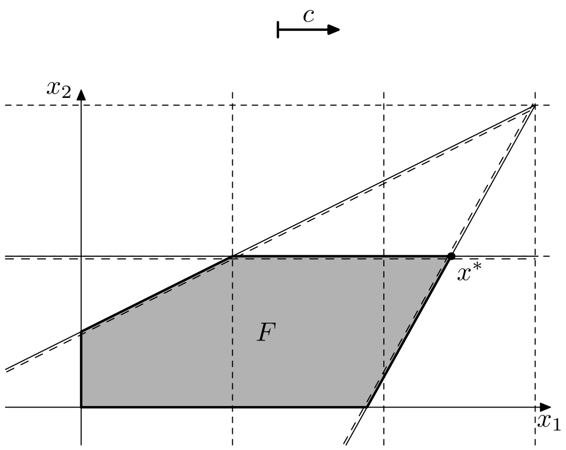


Abbildung 3.3: Beschränkter Zulässigkeitsbereich

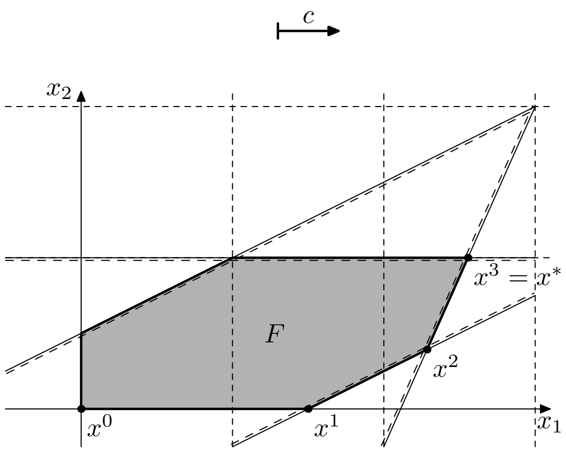


Abbildung 3.4: Abfolge der Ecken

vexen Lösungsmengen linearer Gleichungen und Ungleichungen selbst konvex (Abbildung 3.3). Die Zielfunktion muss ihren optimalen Wert somit (auch) in einer *Ecke* von F annehmen. Eine Ecke lässt sich mathematisch darüber charakterisieren, dass n linear unabhängige Nebenbedingungen, die einfachen Schranken eingeschlossen, mit Gleichheit erfüllt sind. Sind es mehr als n, die notwendigerweise nicht mehr linear unabhängig sein können, so spricht man von einer *entarteten* Ecke.

Auf dieser Beobachtung fußt der älteste Algorithmus zur Lösung von LPs, der *Simplexalgorithmus*. Dieser untersucht von einer Ecke des Zulässigkeitsbereichs startend die Ecken von F, immer in Richtung kleinerer Zielfunktionswerte (Abbildung 3.4). Ist er an einer Ecke angelangt, deren Nachbarn höhere Zielfunktionswerte haben, so folgt aus der Konvexität von F die Optimalität der gefundenen Ecke. Wird während der Ausführung in Richtung fallender Zielfunktionswerte keine Ecke gefunden, so ist das Problem unbeschränkt. Die Frage nach der Unzulässigkeit lässt sich zusammen mit der Bestimmung einer Startecke durch *2–Phasen Methoden* beantworten. Hierbei wird in der ersten Phase ein modifiziertes LP gelöst, welches für die Standardform so aussieht:

$$\begin{aligned} &\min 1^T s \\ &\text{unter } Bx + s = b \\ &\qquad x \geq 0, s \geq 0 \\ &\qquad \text{mit } s \in \mathbb{R}^p. \end{aligned} \tag{3.9}$$

Dieses hat einen nicht leeren Zulässigkeitsbereich, $x = 0, s = b$ ist eine Ecke. Des weiteren ist die Zielfunktion wegen $s \geq 0$ nach unten beschränkt. Somit gibt es einen optimalen Punkt. Ist der minimale Zielfunktionswert 0, so liefert die optimale Ecke einen zulässigen Punkt für das Originalproblem. Dieser muss eine Ecke des Originalproblems sein, da für $s = 0$ der Zulässigkeitsbereich mit dem des originalen LPs übereinstimmt. Ansonsten ist das Originalproblem unzulässig. Da die Anzahl der Ecken exponentiell mit der Dimension des LPs wächst, besitzt der Simplexalgorithmus im schlimmsten Fall exponentielle Laufzeit. Im Mittel ist er jedoch für Probleme kleiner und mittlerer Größenordnung kaum langsamer als *Interior Point* Methoden, welche auf dem Weg zur optimalen Ecke durch das Innere des zulässigen Bereiches laufen. Für diese lässt sich eine polynomielle Laufzeit beweisen.

Um die Lösung eines LPs weiter zu beschleunigen, verwenden die meisten Löser *Preprocessing* oder *Presolving*. Hierbei handelt es sich um verschiedene Heuristiken zur Reduzierung der Komplexität des Problems, vergleiche [6]. So werden zum Beispiel Gleichungsnebenbedingungen der Form

$$x_i = b_i$$

in eine Fixierung der Variable x_i auf den Wert b_i und anschließende Löschung der Nebenbedingung und der Variable überführt. Es werden minimal und maximal mögliche Werte für die Ungleichungsnebenbedingungen mit ihren rechten Seiten verglichen und bei Unvereinbarkeit schon die Unzulässigkeit des Problems festgestellt,

$$\begin{aligned} \min\{A_{i:}x \mid \underline{x} \leq x \leq \overline{x}\} > \overline{a} &\Rightarrow \text{unzulässig} \\ \max\{A_{i:}x \mid \underline{x} \leq x \leq \overline{x}\} < \underline{a} &\Rightarrow \text{unzulässig.} \end{aligned}$$

Für die weiteren Betrachtungen in dieser Arbeit wird folgende Form des

LPs verwendet:

$$\begin{aligned} \min\, & c^T x \\ \text{unter } & Ax \leq a \\ & Bx = b \\ & \underline{x} \leq x \leq \overline{x} \end{aligned} \tag{3.10}$$

Die Positivität von b wird nicht gefordert. Das duale Problem hierzu ist:

$$\begin{aligned} \max\, & a^T y + b^T z + \underline{x}^T u + \overline{x}^T v \\ \text{unter } & A^T y + B^T z + u + v = c \\ & y \leq 0, u \geq 0, v \leq 0 \\ \text{mit } & y \in \mathbb{R}^m, v \in \mathbb{R}^n. \end{aligned} \tag{3.11}$$

Diese Darstellung wird der Standardform vorgezogen, da sie beispielsweise freie Variable direkt repräsentieren kann. Wie sich zeigen lässt, verhindert die Transformation freier Variabler in die Differenz zweier positiver Variabler

$$x =: x^+ - x^-, \quad x^+, x^- \geq 0$$

die Berechnung der unteren Schranke, da sie zu inkorrekt gestellten Problemen führt.

3.3 Spärliche Matrizen

Im Bereich der Linearen Programmierung tauchen häufig dünn besetzte Matrizen auf. Da viele Variable nicht an allen Nebenbedingungen beteiligt sind, die zugrunde liegende Matrix aber Koeffizienten für alle Nebenbedingungen und Variablen enthalten muss, sind viele Elemente hier gleich 0. All diese Nulleinträge abzuspeichern ist bei den hier zu erwartenden Dimensionen aus speicherökonomischer Sicht nicht sinnvoll.

Um diesem Problem zu begegnen, bietet sich die spärliche Speicherung von Matrizen an. Hier wird versucht, durch zusätzliche Informationen, die man über die Matrix und ihre Einträge mitführt, die Speicherung von Nulleinträgen so weit wie möglich zu reduzieren. Dabei sollte man sowohl spezielle Strukturen der Matrix, als auch das Verhältnis von Null- zu Nicht-Nulleinträgen berücksichtigen.

Bei einer Diagonalmatrix beispielsweise reicht es vollkommen aus, die Diagonalelemente zu speichern. Somit reduziert sich der Platzbedarf bei quadratischen Matrizen von n^2 auf n. Sind auch von den Diagonalelementen nur wenige besetzt, so kann es sich lohnen, nur diese Elemente zusammen mit ihrer Position auf der Diagonale zu speichern, um weiteren Platz einzusparen. So lässt sich eine Matrix, wie

$$D = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

mit einem Speicherbedarf von 25 Integern, als Vektor der Diagonalelemente

$$D_1 = (1, 0, 2, 0, 0)$$

mit 5 Integern, oder als Vektor von Paaren der Diagonalelemente und ihrer Position auf der Diagonale speichern:

$$D_2 = ((1, 1), (2, 3)).$$

Dies reduziert den Speicherbedarf auf 4 Integer. Auf ähnliche Weise lassen sich andere Strukturen wie Dreiecksstruktur, Symmetrie usw. ausnutzen.

Eine verbreitete Möglichkeit der Speicherung, wenn nichts weiter über die Struktur der Matrix bekannt ist, ist ein bei lp_solve verwendeter spaltenorientierter Ansatz. Hier werden spaltenweise alle Einträge mit ihren Zeilenindizes gespeichert. Zusätzlich wird gespeichert, wo in dieser Abfolge die jeweiligen Spalten anfangen. Aus der Matrix

$$A = \begin{pmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 4 & 0 & 0 & 2 \\ 0 & 5 & 0 & 0 \end{pmatrix}$$

wird dann der Vektor der Elemente mit ihren Zeilenindizes

$$A_1 = ((4, 3), (2, 1), (5, 4), (1, 2), (2, 3))$$

und der Vektor der Spaltenanfänge

$$A_{1,\text{cols}} = (1, 2, 4, 4).$$

Da die dritte Spalte keine Elemente enthält, sind Anfang der dritten und vierten Spalte identisch. Der Speicherbedarf sinkt von 16 Integern auf 14 Integer.

An diesen Beispielen sieht man, wie stark sich der Speicherbedarf reduzieren kann. Allerdings steigt er bei entsprechend voll besetzten Matrizen auch an. Fügt man im letzten Beispiel einen Eintrag hinzu, so sind der Speicherbedarf der Matrix und ihrer spärlichen Darstellung identisch. Bei zwei weiteren Einträgen benötigt die spärliche Speicherung schon 18 Integer und damit 2 mehr als die nicht spärliche. So muss von Anwendung zu Anwendung entschieden werden, ob die zu erwartende Klasse von Matrizen von einer spärlichen Darstellung profitiert oder nicht.

Trotz des zusätzlichen Verwaltungsaufwandes können auch Rechenoperationen auf den Matrizen von der spärlichen Speicherung profitieren. Durch die Speicherung ist schon bekannt, welche Elemente Nullen sind. So reduziert sich die Addition zweier Diagonalmatrizen darauf, die Diagonalelemente zu addieren. Bei Speicherung aller Elemente müssten auch die Nichtdiagonalelemente verarbeitet werden, nur um zu erkennen, dass hier in der Addition Nullen entstehen. Die spärliche Addition nutzt aus, dass die Summe von Diagonalmatrizen wieder Diagonalgestalt hat, so dass diese Operationen entfallen. Das Verhältnis der Operationen ist, wie das der Speicherplatzanforderungen, n zu n^2. Ähnliches gilt für die Multiplikation von Matrizen, Multiplikation von Matrizen und Vektoren, usw. Auch hier finden sich natürlich Fälle, in denen die spärliche Ausführung von Operationen mehr Zeit benötigt als die nicht spärliche, da die Verwaltungsstrukturen für die resultierende Matrix erzeugt werden müssen.

3.4 Lurupa vs. Ordonez und Freund

In ihrer Arbeit [15] beschäftigen sich Ordonez und Freund mit Konditionszahlen von Linearen Programmen als Spezialfall der Konvexen Optimierung. Um eine Vorstellung von den praktisch vorkommenden Konditionszahlen zu bekommen, wird die Netlib lp library [12] als Testmenge verwendet. Neben den Erkenntnissen über die Konditionszahlen vor und nach Preprocessing mit CPLEX stellt sich heraus, dass die Kondition für verschiedene Beobachtungen verantwortlich ist. So hängt beispielsweise die benötigte Anzahl von Iterationen eines Interior Point Lösers linear von der Kondition nach Preprocessing ab.

Zur Berechnung der Konditionszahl wird das LP zunächst in eine andere Darstellung, das „ground-set model format“ übertragen,

$$\begin{aligned} z_*(d) = \min \quad & c^T x \\ \text{unter} \quad & Ax - b \in C_Y \\ & x \in P. \end{aligned} \tag{3.12}$$

Bei C_Y handelt es sich um einen Kegel, P ist eine abgeschlossene konvexe Menge. Für ein LP der Form (3.10) gilt

$$C_Y := \mathbb{R}^m_- \times \{0\}^p, \tag{3.13}$$

$$P := \{x \mid \underline{x} \leq x \leq \overline{x}\}, \tag{3.14}$$

wobei sich die rechte Seite der Ungleichungen a im Kegel $\mathbb{R}^m_-$ wiederfindet und $\underline{x}, \overline{x}$ unendliche Komponenten haben dürfen. Der Zulässigkeitsbereich ist somit

$$F_d = \{x \mid Ax - b \in C_Y, x \in P\}. \tag{3.15}$$

Parameter des Problems ist das Tripel $d = (A, b, c)$. Auf dieser Grundlage ist die Norm eines Problems definiert als

$$\|d\| := \max\{\|A\|_{\infty,1}, \|b\|_1, \|c\|_1\}, \tag{3.16}$$

wobei

$$\begin{aligned} \|a\|_1 &= \sum_i |a_i|, \\ \|a\|_\infty &= \max\{|a_i|\}, \\ \|A\|_{\infty,1} &= \max\{\|Ax\|_1 \mid \|x\|_\infty = 1\} \end{aligned}$$

gilt.

Der Abstand zur primalen Unzulässigkeit ist definiert als

$$\rho_P(d) := \inf\{\|\Delta d\| \mid F_{d+\Delta d} = \emptyset\}, \tag{3.17}$$

das Infimum der Normen der Störungen, die zu einem leeren Zulässigkeitsbereich führen. Analog hierzu ist die Definition des Abstands zur dualen Unzulässigkeit

$$\rho_D(d) := \inf\{\|\Delta d\| \mid F^D_{d+\Delta d} = \emptyset\}, \tag{3.18}$$

wobei F^D für den Zulässigkeitsbereich des dualen Problems steht. Diese beiden Größen werden ihrerseits mit Hilfe eines LPs bestimmt.

Die Konditionszahl des Problems folgt schließlich als Norm des Problems dividiert durch den minimalen Abstand zur Unzulässigkeit,

$$C(d) := \frac{\|d\|}{\min\{\rho_P(d), \rho_D(d)\}}. \tag{3.19}$$

Zur Bestimmung der Norm des Problems selbst wird $\|A\|_{\infty,1}$ von oben und unten eingeschlossen.

Sei nf die Menge der Indizes der nicht fixierten Variable, also derer mit unterschiedlicher oberer und unterer Schranke. Den Zusammenhang zwischen $\rho_P(d)$ und dem hier entwickelten Verfahren zeigt

Theorem 1 *Für ein Lineares Programm sind folgende Aussagen äquivalent*

$$\rho_P(d) > 0 \tag{3.20}$$

$$\exists x \in F_d : Ax < a,\ Bx = b,\ \underline{x}_{nf} < x_{nf} < \overline{x}_{nf} \textit{ und } \texttt{rang}\, B_{:,nf} = p \tag{3.21}$$

$$\texttt{relint}\, F_d \neq \emptyset \textit{ und } \texttt{rang}\, B_{:,nf} = p. \tag{3.22}$$

Beweis. Gilt (3.21) nicht, so lässt sich immer eine beliebig kleine Störung finden, die für einen leeren Zulässigkeitsbereich sorgt; (3.20) gilt nicht.

Gilt für eine Ungleichung $A_{i:}x \leq a_i$, dass sie für alle zulässigen Punkte mit Gleichheit erfüllt ist, so wird der Zulässigkeitsbereich leer, wenn man a_i um ein ε kleiner macht.

Gibt es eine Gleichung $B_{i:}x = b_i$, die für kein $x \in F_d$ erfüllt ist, so ist der Zulässigkeitsbereich schon leer, $\rho_P(d) = 0$.

Existiert eine nicht fixierte Variable x_i, die obdA gleich ihrer oberen Schranke sein muss, so existieren linear unabhängige Nebenbedingungen, die die Variable an diese Schranke zwingen und somit für alle zulässigen Punkte mit Gleichheit erfüllt sind. Sind hierbei Ungleichungen beteiligt, so erhält man einen leeren Zulässigkeitsbereich, wenn man, wie im ersten Fall, deren rechte Seiten um ein ε kleiner macht. Sorgen nur Gleichungen für diesen Zustand, so muss der i-te Einheitsvektor senkrecht auf dem Nullraum des, aus diesen Gleichungen gebildeten Gleichungssystems stehen. Er muss im linearen Erzeugnis ihrer Gradienten liegen. In einem Gleichungssystem aus diesen Gleichungen lässt sich mit einer LR-Zerlegung eine Gleichung der Form $x_i = \overline{x_i}$ erzeugen. Der Zulässigkeitsbereich wird leer, indem man hier die rechte Seite um ein ε größer macht und aus der Zerlegung eine gestörte rechte Seite für die Gleichungen herleitet, die $x_i > \overline{x_i}$ fordert.

Ist

$$\texttt{rang}\, B_{:,nf} \neq p$$

bilden die Spalten der Matrix keine Basis des $\mathbb{R}^p$. Es gibt also mindestens einen Vektor des $\mathbb{R}^p$ der linear unabhängig von diesen Spalten von B ist. Stört man b in diese Richtung gibt es entweder keine Lösung mehr für $Bx = b$, oder die Lösung würde eine Änderung des Wertes einer fixierten Variable erfordern; der Zulässigkeitsbereich wird leer.

Gilt (3.21), so lässt sich keine solche Störung finden, (3.20) gilt.

Ist $Ax < a$, so gibt es keine beliebig kleine Störung in a, die dafür sorgt, dass $Ax \leq a$ nicht mehr erfüllt ist. Auch Störungen an A sind wegen

$$\|(A + \Delta A)x\| \leq \|Ax\| + \|\Delta Ax\| \leq \|Ax\| + \|\Delta A\|\|x\|$$

in ihrer Wirkung auf das Ergebnis von Ax beschränkt. Sie können $Ax \leq a$ nicht verletzen, keinen leeren Zulässigkeitsbereich erzeugen.

Bei vollem Rang von $B_{:,nf}$ ist die Änderung der Lösung von $Bx = b$ linear von der Größe der Störung in B und b abhängig. Diese Änderung wird komplett von den nicht fixierten Variablen ausgeglichen, die für beliebig kleine Störungen ihre einfachen Schranken wegen $\underline{x}_{nf} < x < \overline{x}_{nf}$ nicht verletzen. Auch die Ungleichungen bleiben wegen $Ax < a$ für beliebig kleine Störungen erfüllt.

Das relative Innere $\texttt{relint}\, F$ ist die Menge aller Punkte, für die eine ganze Umgebung geschnitten mit der affinen Hülle von F in F liegt. Die affine Hülle von F ist gerade die Menge aller Punkte, die $Bx = b$ erfüllen. Somit ist (3.22) äquivalent dazu, dass es einen Punkt gibt, der die Ungleichungen mit echt ungleich erfüllt und $\texttt{rang}\, B_{:,nf} = p$, also (3.21) gilt. ∎

Analog lässt sich der Zusammenhang zwischen $\rho_D(d)$ und dem hier entwickelten Verfahren zeigen.

3.5 Schranken

Die in Lurupa implementierten rigorosen Fehlerschranken bei der Lösung von LPs beruhen auf dem von Jansson entwickelten Verfahren [9]. Dort wird ein LP durch das Parametertupel

$$P := (A, B, a, b, c) \in \mathbb{R}^{(m+p+1)n+m+p}$$

mit den einfachen Schranken

$$\underline{x} < \overline{x}$$

beschrieben. Hierbei können unendliche Schranken auftreten, d.h.

$$\begin{aligned} \underline{x}_i &= -\infty \\ \overline{x}_j &= \infty \end{aligned} \qquad i, j \in \{1, \ldots, n\}$$

ist erlaubt. Die Indizes der freien Variablen bilden die Indexmenge

$$J^{\infty} := \{j \mid \underline{x}_j = -\infty, \overline{x}_j = +\infty\}.$$

Ihr Komplement ist

$$J^{r} := \{1, \ldots, n\} \setminus J^{\infty}.$$

Um Intervallprobleme oder Eingabefehler zu modellieren, wird diese Darstellung auf Familien von Problemen erweitert, wobei die Parameter jeweils aus einem Parameterintervall stammen,

$$\mathbf{P} := (\mathbf{A}, \mathbf{B}, \mathbf{a}, \mathbf{b}, \mathbf{c}).$$

Die Grundidee der Verfahren ist, durch Perturbationen der Probleme ein Intervall zu finden, welches verifiziert innerhalb des zulässigen Bereiches liegt. Die Zielfunktionswerte dieses Intervalls liefern dann die Schranken basierend auf folgenden Theoremen.

Theorem 2 *Sei* $\mathbf{P} := (\mathbf{A}, \mathbf{B}, \mathbf{a}, \mathbf{b}, \mathbf{c})$ *eine Familie von Problemen mit einfachen Schranken* $\underline{x} < \overline{x}$. *Existiert ein Intervallvektor* $\mathbf{x} \in \mathbb{IR}^n$ *mit*

$$\begin{aligned} \mathbf{A}\mathbf{x} &\leq \mathbf{a}, \\ \underline{x} \leq \mathbf{x} &\leq \overline{x} \end{aligned}$$

und

$$\forall B \in \mathbf{B}, b \in \mathbf{b}\ \exists x \in \mathbf{x} : Bx = b,$$

dann enthält $\mathbf{x}$ *für jedes Problem aus der Familie eine zulässige Lösung, und*

$$\overline{f^*} := \sup\{\mathbf{c}^T\mathbf{x}\}$$

ist eine obere Schranke für die optimalen Zielfunktionswerte. Wenn die Zielfunktionen aller Probleme von unten beschränkt sind, gilt weiterhin, dass alle Probleme aus der Familie eine optimale Lösung haben.

Theorem 3 *Sei* $\mathbf{P} := (\mathbf{A}, \mathbf{B}, \mathbf{a}, \mathbf{b}, \mathbf{c})$ *eine Familie von LP-Problemen mit den einfachen Schranken* $\underline{x} < \overline{x}$. *Existieren Intervallvektoren* $\mathbf{y} \in \mathbb{IR}^m$ *und* $\mathbf{z} \in \mathbb{IR}^p$ *mit den Eigenschaften*

1. $$\mathbf{y} \leq 0, \tag{3.23}$$

2. $$\begin{aligned} &\forall A \in \mathbf{A},\ B \in \mathbf{B},\ c \in \mathbf{c}\ \exists y \in \mathbf{y},\ z \in \mathbf{z} : \\ &\quad (A_{:j})^T y + (B_{:j})^T z = c_j \text{ für } j \in J^\infty, \end{aligned} \tag{3.24}$$

3. *und die Intervalle*

$$\mathbf{d}_j := \mathbf{c}_j - (\mathbf{A}_{:j})^T\mathbf{y} - (\mathbf{B}_{:j})^T\mathbf{z} \text{ für } j \in J^r \tag{3.25}$$

erfüllen

$$\mathbf{d}_j \leq 0, \quad \text{wenn } \underline{x}_j = -\infty \text{ und } j \in J^r, \tag{3.26}$$

$$\mathbf{d}_j \geq 0, \quad \text{wenn } \overline{x}_j = +\infty \text{ und } j \in J^r, \tag{3.27}$$

so gilt die Ungleichung

$$\inf_{P \in \mathbf{P}} f^*(P) \geq \underline{f^*} := \min\{\mathbf{a}^T\mathbf{y} + \mathbf{b}^T\mathbf{z} + \sum_{\substack{j \in J^r \\ \underline{\mathbf{d}_j} > 0}} \underline{x}_j \mathbf{d}_j^+ + \sum_{\substack{j \in J^r \\ \underline{\mathbf{d}_j} < 0}} \overline{x}_j \mathbf{d}_j^-\}; \tag{3.28}$$

$\underline{f^*}$ ist eine untere Schranke für den global optimalen Zielfunktionswert. Wenn zusätzlich

1. *alle Eingabedaten reell sind (d.h. $P = \mathbf{P}$),*
2. *P eine optimale Lösung (y^*, z^*, u^*, v^*) besitzt,*
3. *$\mathbf{y} := y^*$, $\mathbf{z} := z^*$ und*
4. *die Größen in (3.25) und (3.28) exakt berechnet werden,*

so sind die Bedingungen (3.23), (3.24), (3.26) und (3.27) erfüllt und der optimale Wert $f^*(P) = \underline{f^*}$; die Schranke ist scharf für reelle Eingaben und exakte Berechnung.

Die Beweise finden sich in der Arbeit von Jansson.

Um die Fixierung von Variablen zu ermöglichen und eine Konvergenzanalyse durchzuführen, wurde folgende Modifikation des Verfahrens entwickelt. Es wird vorausgesetzt, dass der verwendete LP-Löser genau dann eine zulässige Lösung findet, wenn der Zulässigkeitsbereich nicht leer ist. Diese Lösung erfülle die Nebenbedingungen bis auf ein ϵ_{lp}. Zudem wird vorausgesetzt, dass der Intervalllöser zur Einschließung der Lösungsmenge eines Gleichungssystems $Bx = b$ Intervalle mit einem maximalen Radius von ϵ_{Int} liefert. e steht im folgenden für einen Vektor, dessen Komponenten alle gleich 1 sind.

3.5.1 Obere Schranke

Eine obere Schranke bestimmt sich mit folgendem Algorithmus.

1. *Seien $\epsilon_a, \epsilon_s > 0$ beliebig.*
2. *Löse $P(\epsilon) := (A, B, a - \epsilon_a e, b, c)$ mit komponentenweise definierten*

$$\underline{x}_i(\epsilon) := \begin{cases} \underline{x}_i & \text{, wenn } \underline{x}_i = -\infty \\ \underline{x}_i + \epsilon_s & \text{sonst,} \end{cases}$$

und

$$\overline{x}_i(\epsilon) := \begin{cases} \overline{x}_i & \text{, wenn } \overline{x}_i = +\infty \\ \overline{x}_i - \epsilon_s & \text{sonst.} \end{cases}$$

Liefert der LP-Löser keine approximative Lösung $\tilde{x}$, wiederhole Schritt 2 mit verringerten Parametern ϵ_a, ϵ_s.

3. *Erzwinge $\underline{x}(\epsilon) \leq \tilde{x} \leq \overline{x}(\epsilon)$ durch*

$$\tilde{x}_i = \begin{cases} \underline{x}_i(\epsilon) & \text{, wenn } \tilde{x}_i < \underline{x}_i(\epsilon) \\ \overline{x}_i(\epsilon) & \text{, wenn } \tilde{x}_i > \overline{x}_i(\epsilon) \\ \tilde{x}_i & \text{sonst.} \end{cases}$$

4. *Enthält das Problem keine Gleichungen, prüfe $A\tilde{x} \leq a$, $\underline{x} \leq \tilde{x} \leq \overline{x}$. Ist dies erfüllt, gib $c^T\tilde{x}$ als obere Schranke zurück, sonst erhöhe ϵ_a, ϵ_s und gehe zu Schritt 2.*

5. *Bestimme eine Einschließung $\mathbf{x}$ der Lösungsmenge von $B\tilde{\tilde{x}} = b$. Prüfe, ob diese $\sup\{A\mathbf{x}\} \leq a$, $\underline{x} \leq \mathbf{x} \leq \overline{x}$ erfüllt. Ist dies der Fall, so gib $\sup\{c^T\mathbf{x}\}$ als obere Schranke zurück, sonst erhöhe ϵ_a, ϵ_s und gehe zu Schritt 2.*

Die Konvergenz des Verfahrens liefert

Theorem 4 *Gilt für ein LP-Problem*

$$\rho_P > 0,$$

so findet der obige Algorithmus bei hinreichend großen Rechengenauigkeiten in einem Durchgang eine obere Schranke.

Beweis. Die Forderung $\rho_P > 0$ ist nach Theorem 1 äquivalent zu der Aussage

$$\exists x \in F_d : Ax < a,\ Bx = b,\ \underline{x}_{nf} < x_{nf} < \overline{x}_{nf} \text{ und } \texttt{rang}\, B_{:,nf} = p \tag{3.29}$$

Der Beweis erfolgt in 2 Schritten. Das perturbierte Problem in Schritt 2. hat wegen (3.29) eine Lösung. Es gibt eine obere Schranke δ für ϵ_a, ϵ_s, unterhalb derer ein $x \in F$ existiert, welches $Ax \leq a - \epsilon_a e$, $Bx = b$, $\underline{x}(\epsilon) \leq x \leq \overline{x}(\epsilon)$ erfüllt; es gilt $F(\epsilon) \neq \emptyset$.

Erfüllen die Rechengenauigkeiten diese Schranke, so findet der LP-Löser eine approximative Lösung $\tilde{x}$ mit den Eigenschaften

$$A\tilde{x} \leq a - \epsilon_a e + \epsilon_{lp} e + (a - \epsilon_a e)\varepsilon_A \tag{3.30}$$
$$B\tilde{x} = b \pm \epsilon_{lp} e \tag{3.31}$$
$$\tilde{x}_i \geq \underline{x}_i + \epsilon_s - \epsilon_{lp} - |\underline{x}_i + \epsilon_s|\varepsilon_A \qquad \text{für } \underline{x}_i \neq -\infty \tag{3.32}$$
$$\tilde{x}_i \leq \overline{x}_i - \epsilon_s + \epsilon_{lp} + |\overline{x}_i - \epsilon_s|\varepsilon_A \qquad \text{für } \overline{x}_i \neq +\infty. \tag{3.33}$$

Hiermit gilt in Schritt 3.

$$B\tilde{x} = b \pm \big(\epsilon_{lp} e + (\epsilon_{lp} + \Delta\underline{\overline{x}}_A)|B|e\big) \tag{3.34}$$
$$A\tilde{x} \leq a - \epsilon_a e + \epsilon_{lp} e + (a - \epsilon_a e)\varepsilon_A + (\epsilon_{lp} + \Delta\underline{\overline{x}}_A)|A|e \tag{3.35}$$

mit

$$\Delta\underline{\overline{x}}_A := \varepsilon_A \max_{\underline{x}_i, \overline{x}_i \neq \infty} \{|\underline{x}_i + \epsilon_s|,\, |\overline{x}_i - \epsilon_s|\} \tag{3.36}$$

Enthält das Problem keine Gleichungen, so erfolgt an dieser Stelle die Überprüfung, ob $A\tilde{x} \leq a,\ \underline{x} \leq \tilde{x} \leq \overline{x}$ gilt. Dies ist erfüllt, solange

$$\epsilon_a \geq \frac{\epsilon_{lp} + \varepsilon_A \max a + (\epsilon_{lp} + \Delta\underline{\overline{x}}_A)\max\{|A|e\}}{1 + \varepsilon_A} \tag{3.37}$$
$$\epsilon_s \geq 0 \tag{3.38}$$

gilt. Durch genügend hohe Rechengenauigkeiten in den verwendeten Komponenten geht der Bruch gegen 0, so dass die obere Schranke δ für ϵ_a, ϵ_s gewahrt werden kann; der Algorithmus findet einen verifizierten inneren Punkt, eine obere Schranke für f^* ist $c^T\tilde{x}$.

Sind Gleichungen im Modell enthalten, so folgt aus (3.29), dass es eine reguläre $p \times p$ Teilmatrix B_I von B gibt. Es existiert ein $\hat{x}$ mit $B\hat{x} = b$, das sich von $\tilde{x}$ nur in den p I-Komponenten unterscheidet und

$$\begin{gathered} \|\hat{x} - \tilde{x}\|_\infty \leq \Delta\tilde{x} := \kappa(B_I) \cdot \frac{\Delta b}{\|b\|_\infty} \cdot \|\hat{x}\|_\infty \\ \text{mit} \quad \Delta b := \epsilon_{lp} + (\epsilon_{lp} + \Delta\underline{\overline{x}}_A)\max\{|B|e\} \end{gathered} \tag{3.39}$$

erfüllt. Der Intervalllöser liefert nun $\mathbf{x} : \mathtt{rad}(\mathbf{x}) \leq \epsilon_{Int},\ \hat{x} \in \mathbf{x}$. Somit gilt

$$\max\{\|\sup\{\mathbf{x}\} - \tilde{x}\|_\infty, \|\inf\{\mathbf{x}\} - \tilde{x}\|_\infty\} \leq 2\epsilon_{Int} + \Delta\tilde{x}$$

Damit erfüllt $\mathbf{x}$ aber

$$\underline{x} + (\underline{\epsilon}_s - 2\epsilon_{Int} - \Delta\tilde{x})e \leq \mathbf{x} \leq \overline{x} - (\overline{\epsilon}_s + 2\epsilon_{Int} + \Delta\tilde{x})e \tag{3.40}$$

und

$$\sup\{A\mathbf{x}\} \leq a - \epsilon_a e + \epsilon_{lp} e + (a - \epsilon_a e)\varepsilon_A + \\ (\epsilon_{lp} + \Delta\underline{\overline{x}}_A + 2\epsilon_{Int} + \Delta\tilde{x})|A|e. \tag{3.41}$$

Die nun erfolgende Prüfung auf Zulässigkeit $\sup\{A\mathbf{x}\} \leq a$, $\underline{x} \leq \mathbf{x} \leq \overline{x}$ ist also erfolgreich, wenn

$$\epsilon_a \geq \frac{\epsilon_{lp} + \varepsilon_A \max a + (\epsilon_{lp} + \Delta\underline{\overline{x}}_A + 2\epsilon_{Int} + \Delta\tilde{x}) \max\{|A|e\}}{1 + \varepsilon_A} \tag{3.42}$$

$$\epsilon_s \geq 2\epsilon_{Int} + \Delta\tilde{x} \tag{3.43}$$

gilt. Auch hier gehen die Terme der rechten Seiten für genügend hohe Rechengenauigkeiten gegen 0, so dass die obere Schranke δ für ϵ_a, ϵ_s abermals gewahrt werden kann. Der Algorithmus findet $\mathbf{x}$, welches den verifiziert zulässigen Punkt $\hat{x}$ enthält, und liefert $\sup\{c^T\mathbf{x}\} \geq c^T\hat{x}$ als obere Schranke für f^*.

Umgekehrt, ist (3.29) falsch, so gibt es keine $\epsilon_a, \epsilon_s > 0$, welche ein Problem $P(\epsilon)$ mit nicht leerem Zulässigkeitsbereich $F(\epsilon)$ erzeugen. Der LP-Löser liefert keine approximative Lösung und der Algorithmus somit keine Schranke.

■

3.5.2 Untere Schranke

Eine untere Schranke folgt mit folgendem Algorithmus

1. *Sei $\epsilon_c > 0$ beliebig.*

2. *Löse $P(\epsilon) := (A, B, a, b, c(\epsilon))$ mit den einfachen Schranken $\underline{x}, \overline{x}$. Hierbei ist*

$$c_j(\epsilon) := \begin{cases} c_j + \epsilon_c & \text{, wenn } j \in J^r \text{ und } \underline{x}_j = -\infty \\ c_j - \epsilon_c & \text{, wenn } j \in J^r \text{ und } \overline{x}_j = +\infty \\ c_j & \text{sonst.} \end{cases}$$

   Liefert der LP-Löser keine approximative Lösung $(\tilde{y}, \tilde{z}, \tilde{u}, \tilde{v})$, wiederhole Schritt 2 mit verringertem Parameter ϵ_c.

3. *Erzwinge $\tilde{y} \leq 0$ durch*

$$\tilde{y}_j = \begin{cases} 0 & \text{, wenn } \tilde{y}_j > 0 \\ \tilde{y}_j & \text{sonst.} \end{cases}$$

4. Enthält das Problem keine freie Variable, so überprüfe, ob

$$\mathbf{d}_j := c_j - (A_{:j})^T\tilde{y} - (B_{:j})^T\tilde{z} \le 0 \qquad , \textit{für } \underline{x_j} = -\infty \tag{3.44}$$

$$\mathbf{d}_j := c_j - (A_{:j})^T\tilde{y} - (B_{:j})^T\tilde{z} \ge 0 \qquad , \textit{für } \overline{x_j} = +\infty \tag{3.45}$$

gilt. Ist dies der Fall, gib

$$\underline{f^*} := \min\{a^T\tilde{y} + b^T\tilde{z} + \sum_{\substack{j\in J^r\\ \overline{\mathbf{d}}_j>0}} \underline{x}_j\mathbf{d}_j^+ + \sum_{\substack{j\in J^r\\ \underline{\mathbf{d}}_j<0}} \overline{x}_j\mathbf{d}_j^-\}$$

als untere Schranke zurück, sonst erhöhe ϵ_c und gehe zu Schritt 2.

5. Bestimme Einschließungen $\mathbf{y},\mathbf{z}$ der Lösung von $(A_{:j})^T\tilde{y} + (B_{:j})^T\tilde{z} = c_j$ für $j \in J^\infty$. Prüfe, ob diese

$$\mathbf{y} \le 0 \tag{3.46}$$

$$\mathbf{d}_j := c_j - (A_{:j})^T\mathbf{y} - (B_{:j})^T\mathbf{z} \le 0 \quad , \textit{für } j \in J^r \textit{ und } \underline{x_j} = -\infty \tag{3.47}$$

$$\mathbf{d}_j := c_j - (A_{:j})^T\mathbf{y} - (B_{:j})^T\mathbf{z} \ge 0 \quad , \textit{für } j \in J^r \textit{ und } \overline{x_j} = +\infty \tag{3.48}$$

erfüllen. Ist dies der Fall, gib

$$\underline{f^*} := \min\{a^T\mathbf{y} + b^T\mathbf{z} + \sum_{\substack{j\in J^r\\ \overline{\mathbf{d}}_j>0}} \underline{x}_j\mathbf{d}_j^+ + \sum_{\substack{j\in J^r\\ \underline{\mathbf{d}}_j<0}} \overline{x}_j\mathbf{d}_j^-\}$$

als untere Schranke zurück, sonst erhöhe ϵ_c und gehe zu Schritt 2.

Die Konvergenz des Verfahrens folgt auf ähnliche Weise wie für die obere Schranke.

Kapitel 4

Implementierung

4.1 Software

Zur Implementierung von Lurupa kamen verschiedene Softwarepakete zum Einsatz.

4.1.1 PROFIL/BIAS

Die Implementierung rigoroser Fehlerschranken macht die Verwendung verifizierter Operationen unverzichtbar. Zu diesem Zwecke wurde die PROFIL/BIAS C++ Bibliothek [10] gewählt. BIAS stellt performante Intervalloperationen nach Art der BLAS [2] zur Verfügung. Diese werden von PROFIL mit Datentypen wie Intervallen, Intervallvektoren und Intervallmatrizen gekapselt, so dass sich eine verifizierte Rechnung von ihrem reellen Pendant nur in den Typen der Parameter unterscheidet.

4.1.2 lp_solve

Als unter der LGPL verfügbarer LP-Löser stellt lp_solve [1] sowohl Methoden zur Lösung von LPs als auch von Mixed-Integer Programmen zur Verfügung. Die Verwendung spärlicher Matrizen zur Speicherung des LPs ermöglicht auch die Lösung großer Probleme mit mehreren tausend Variablen. Um den Einfluss verschiedener LP-Löser auf die Schrankenberechnung zu untersuchen, wurden Messungen mit den Versionen 3.2 und 4.0 gemacht. Diese ähneln sich in den Datenstrukturen, allerdings bringt Version 4.0 einige Erweiterungen und Verbesserungen des Algorithmus mit sich.

4.1.3 Doxygen

Zur Dokumentation des Codes wurde Doxygen [19] eingesetzt. Es ermöglicht die Dokumentation direkt im Quelltext an der zu dokumentierenden Stelle. Es stehen vielfältige Konstrukte zur Strukturierung, wie Parameter von Funktionen und Rückgabewerte zur Verfügung. Die Ausgabe der Dokumentation erfolgt wahlweise zum Beispiel nach HTML und LaTeX. Hierbei werden die Teile der Dokumentation entsprechend der Interaktion innerhalb des Programms referenziert, es werden Kollaborations- und Vererbungsdiagramme erstellt.

4.2 Struktur

Für die Implementierung wurde eine modulare Struktur gewählt. Die Funktionalität wurde in vier Klassen aufgeteilt.

CLurupa Die zentrale Klasse von Lurupa. Sie enthält die Implementierung der Algorithmen. Nach der Erzeugung der Objekte stellt sie die Schnittstelle für den Benutzer zur Verfügung und delegiert die Aufrufe.

CLPModule Als Prototyp für LP-Module ist hier die Funktionalität definiert, die für jeden zu verwendenden LP-Löser implementiert werden muss. Im Einzelnen sind dies

- **setLp** liest ein LP aus der Datenstruktur des verwendeten LP-Lösers, transformiert es in die Darstellung von Lurupa und speichert es in einer Instanz der Klasse CLP.
- **solveAPertLp** übernimmt Deflationsparameter für die rechte Seite der Ungleichungen und die einfachen Schranken. Das LP wird entsprechend perturbiert und eine approximative Lösung bestimmt. Der primale Teil der Lösung wird aus der Datenstruktur des Lösers entnommen und in der CLP Instanz abgelegt.
- **solveCPertLp** übernimmt einen Deflationsparameter für den Kostenvektor des primalen Problems. Nach Perturbation und approximativer Lösung des LPs werden die Lagrange-Parameter der Lösung ausgelesen und in der CLP Instanz abgelegt.
- **restoreUpper** stellt nach Berechnung der oberen Schranke und den entsprechenden Perturbationen des Originalproblems dieses aus den Informationen in der CLP Instanz wieder her.

restoreLower stellt nach Berechnung der unteren Schranken und den entsprechenden Perturbationen des Originalproblems dieses aus den Informationen in der CLP Instanz wieder her.

CDebug stellt Debug und Reportfunktionen zur Verfügung. Hier wird ein Level für die Menge der Ausgaben verwaltet. Die Ausgabefunktionen können wahlweise einen timestamp mitliefern. Es können PROFIL Matrizen und Vektoren als CSV (Comma Seperated Value) Daten abgespeichert werden, um diese im Nachhinein analysieren zu können.

CLP Speichert ein LP in Lurupas eigenem Format. Hierbei werden PROFIL Intervalldatentypen verwendet, um die transparente Verwendung der Intervalloperationen in den Algorithmen zu ermöglichen. Zusätzlich werden Metadaten, wie die Aussage des LP-Lösers über die Lösbarkeit des letzten LPs, die Darstellung von ∞ in den Strukturen des verwendeten LP-Lösers, die Optimierungsrichtung (Minimieren oder Maximieren), die Anzahl freier Variabler und ihr Index im Variablenvektor gespeichert.

4.3 Implementierung eines LP-Moduls

Um einen nicht unterstützten LP-Löser zur approximativen Lösung der LPs zu verwenden, muss ein LP-Modul für diesen implementiert werden. Dieses hat die Aufgabe, die Schnittstelle zwischen Lurupa und dem LP-Löser zu definieren. Die Deklarationen hierzu sind bereits in **CLPModule.h** enthalten, alle globalen Definitionen, wie die Makros zur Darstellung der Rückgabewerte, sind in **globals.h** definiert. Die zu implementierende Klasse muss somit von `CLPModule` erben, und die dort als virtuell deklarierten Methoden sind zu überschreiben.

```
class CLPM_sample : public CLPModule
```

4.3.1 setLp

ist für die Übertragung des Modells in die Darstellungsweise von Lurupa zuständig. Der Aufruf von `CLurupa::setLp` wird an diese Methode weitergeleitet und um weitere Parameter ergänzt. Die Signatur sieht folgendermaßen aus

```
bool CLPModule::setLp(void * pmodelIn, void * pfeasibility, CLP & lp,
                      double intData, double & eta)
```

Das LP ist aus dem übergebenen Modell auszulesen und in `lp` zu speichern. Die Klasse `CLP` hat folgende Gestalt:

```
class CLP{
public:
  INTERVAL_MATRIX IA;         // The inequalities
  INTERVAL_MATRIX IB;         // The equalities
  INTERVAL_VECTOR ia;         // Right hand sides of the inequalities
  INTERVAL_VECTOR ib;         // Right hand sides of the equalities
  INTERVAL_VECTOR ic;         // The cost vector
  INTERVAL_VECTOR isimBounds; // Vector of the simple bounds
  INTERVAL_VECTOR solution;   // Solution of the last solved problem
                              //  (changes during algorithm as
                              //   perturbated problems are solved)
  INTERVAL_VECTOR dualsEqu;   // Duals of the equalities of the last
                              //  solved problem
                              //  (changes during algorithm as
                              //   perturbated problems are solved)
  INTERVAL_VECTOR dualsInequ; // Duals of the inequalities of the last
                              //  solved problem <BR>
                              //  (changes during algorithm as
                              //   perturbated problems are solved)
  REAL infinite;              // Used lp-solvers representation of
                              //  infinity (e.g. as bound value)
  int freeVariables;          // Number of free variables in model
  int * aFreeVariables;       // The indices of the free variables
  int nNonFixedVars;          // Number of non fixed variables in model
  short feasibility;          // Lp-solvers judgement of last solved
                              //  problems feasibility
                              //  (possible values defined in globals.h)
  bool maximise;              // True if objective function is to be
                              //  maximised false otherwise
  char * name;                // Name of the lp model
};
```

Zu beachten ist, dass das übertragene Modell der Normalform (3.10) entspricht. Zu maximierende Zielfunktionen werden durch Multiplikation der Kosten mit -1 in zu minimierende überführt. Die gewünschte Optimierungsrichtung wird durch den flag `lp.maximise` angegeben. $\geq$-Ungleichungen werden durch Multiplikation mit -1 in $\leq$ überführt. Bei der Transformation der dualen Lösung ist darauf zu achten, wie die Vorzeichenkonventionen im verwendeten LP-Löser sind, und diese entsprechend in Lurupas zu überführen. Zum Beispiel ergibt sich eine Multiplikation der Lagrange-Parameter mit -1 je nachdem, ob man zur Berechnung fordert, dass der Kostenvektor im linearen Erzeugnis der Gradienten der Nebenbedingungen liegen soll (3.11), oder dass die Gradienten und der Kostenvektor linear abhängig sein sollen, was zu folgender Nebenbedingung führt:

$$A^T y + B^T z + u + v + c = 0.$$

Durch die Referenz auf `eta` ist es möglich, diesen Algorithmus Parameter abhängig vom jeweils zu lösenden LP zu verändern.

`intData` ist dazu gedacht, die Intervallfähigkeiten von Lurupa zu testen. Hierzu können zum Beispiel alle Parameter des LPs durch Intervalle mit gegebenem Mittelpunkt und relativem Radius ersetzt werden:

$$\mathbf{X} = [X * (1 - \text{intData}), X * (1 + \text{intData})].$$

Die relative Störung ist hier von Vorteil, weil Nulleinträge in den Parametern erhalten bleiben. Diese bedeuten ja, dass beispielsweise eine Variable nicht an einer Nebenbedingung beteiligt ist. Diese Information würde durch Störung der 0 verloren gehen.

`name` ist lediglich für Report- und Debugfunktionen von Bedeutung.

Rückgabewert ist `false`, wenn die Transformation nicht durchgeführt werden konnte, ansonsten `true`.

4.3.2 solveAPertLp

ist für die Lösung eines perturbierten Problems zur Bestimmung der oberen Schranke zuständig. Die Signatur ist

```
bool CLPModule::solveAPertLp(VECTOR & defParmA, VECTOR & defParmXLow,
                             VECTOR & defParmXUpp, CLP & lp)
```

Die rechten Seiten der Ungleichungen und die einfachen Schranken des perturbierten Problems ergeben sich aus den originalen und den Deflationsparametern:

```
Mid(lp.ia) - defParmA
Inf(lp.isimBounds) + defParmXLow
Sup(lp.isimBounds) - defParmXUpp
```

Nach der Lösung des entstandenen LPs ist in `lp.feasibility` zu speichern, ob der Löser das Problem für lösbar hält. Mögliche Werte sind:

`LU_LP_FEASIBLE`	Der Löser hat eine approximative Lösung berechnet.
`LU_LP_UNBOUNDED`	Der Löser hält das Problem für unbeschränkt.
`LU_LP_INFEASIBLE`	Der Löser hält das Problem für unlösbar.
`LU_LP_FAILURE`	Beim Lösen trat ein Fehler auf, es wurde keine Lösung bestimmt.

Wurde eine approximative primale Lösung berechnet, so ist diese wieder in Lurupas Format zu überführen und in `lp.solution` zu speichern. Hält der Löser das Problem für unbeschränkt, so kann man, falls zum Beispiel aus einem Simplexalgorithmus ein zulässiger Punkt bekannt ist, diesen als approximative Lösung speichern und hieraus eine Schranke erhalten.

Rückgabewert ist `false`, wenn ein Fehler auftritt – in den implementierten Modulen `CLPM_lps32` und `CLPM_lps40` zum Beispiel, wenn der Löser einen unbekannten Fehlercode zurück liefert – ansonsten `true`.

4.3.3 solveCPertLp

ist für die Lösung eines perturbierten Problems zur Bestimmung der unteren Schranken zuständig. Die Signatur ist

```
bool CLPModule::solveCPertLp(VECTOR & defParmC, CLP & lp)
```

Hier ist nur die Perturbation des Kostenvektors nötig. Die neuen Kosten ergeben sich aus den originalen und dem Deflationsparameter als:

```
Mid(lp.ic) - defParmC
```

Dabei ist darauf zu achten, dass diese Kosten für ein minimierendes originales LP stehen, bei einem maximierenden müssen diese Kosten noch mit -1 multipliziert werden. Die Beurteilung der Lösbarkeit dieses LPs ist wiederum in `lp.feasibility` zu speichern. Die duale Lösung ist entsprechend der Vorzeichenkonvention des Lösers zu modifizieren und in `lp.dualsEqu` und `lp.dualsInequ` zu speichern. Ist das duale Problem unbeschränkt, so lässt sich auch hier eine Schranke bestimmen, indem ein dual zulässiger Punkt, z. B. aus einer 2–Phasen Methode, als duale Lösung gespeichert wird.

Rückgabewert ist `false`, wenn ein Fehler auftritt – in den implementierten Modulen `CLPM_lps32` und `CLPM_lps40` zum Beispiel, wenn der Löser einen unbekannten Fehlercode zurück liefert – ansonsten `true`.

4.3.4 restoreUpper

ist dafür zuständig, das originale Problem nach der Berechnung der oberen Schranke wieder herzustellen. Die Signatur ist

```
void CLPModule::restoreUpper(CLP & lp);
```

Hierzu sind die originalen LP Parameter aus `lp` zu entnehmen und die perturbierten rechten Seiten der Ungleichungen und einfachen Schranken zu ersetzen.

4.3.5 restoreLower

ist dafür zuständig, das originale Problem nach der Berechnung der unteren Schranke wieder herzustellen. Die Signatur ist

```
void CLPModule::restoreLower(CLP & lp);
```

Die perturbierten Kosten sind durch die originalen aus `lp` zu ersetzen.

4.3.6 CLPM_lps32

Bei der Implementierung der Schnittstelle zu lp_solve 3.2 waren einige Besonderheiten zu beachten. So speichert lp_solve zwar intern ein minimierendes Modell mit ausschließlich $\leq$-Ungleichungen und Gleichungen, die Methoden zum Auslesen dieses Modells liefern aber das Ursprungsproblem zurück. Da diese Methoden eine je nach Wunsch vorgenommene interne Skalierung rückgängig machen, die die Lösung mancher Probleme erst ermöglicht, hat man die Wahl, entweder diese Skalierung selbst rückgängig zu machen, oder das Originalproblem aus den Methoden zu transformieren. Hierbei stellte sich heraus, dass die Skalierung in lp_solve 3.2 noch nicht unter allen Umständen korrekt implementiert ist. Um möglichst unabhängig von Änderungen der internen Struktur späterer Versionen zu sein, wurde trotzdem gegen die Methoden programmiert.

So wird in `setLp` zuerst einmal ausgelesen, ob die Zielfunktion zu minimieren oder zu maximieren ist. Dementsprechend werden die Koeffizienten der Zielfunktion und von $\geq$-Ungleichungen mit -1 multipliziert. Des weiteren sind die Nebenbedingungen in lp_solves Strukturen nicht nach $\geq, \leq, =$ geordnet. Hier werden zwei arrays erzeugt, die die Zuordnung zwischen der Position der Gleichungen und Ungleichungen in lp_solves und Lurupas Datenstrukturen speichert. Unendliche einfache Schranken werden bei lp_solve durch einen Zahlenwert von $1{\cdot}10^{24}$ kodiert, so dass dieser Wert als Darstellung für ∞ in `lp.infinite` gespeichert wird. Da die von lp_solve 3.2 verwendete Darstellung freier Variabler durch zwei positive Variable eine untere Schrankenberechnung verhindert, muss diese rückgängig gemacht werden. Lp_solve speichert allerdings keine Informationen über die beiden Komponenten. Sobald in einem LP eine freie Variable entdeckt wird, erzeugt lp_solve eine zweite mit den negativen Koeffizienten, deren Name sich aus dem ursprünglichen mit angehängtem „_“ ergibt. Beide bekommen eine untere Schranke von 0 und eine obere von $1{\cdot}10^{24}$. Nun werden hier die Namen der Variable auf diesen terminierenden „_“ hin überprüft. Wird einer gefunden, so wird über den Namen nach dem Index der positiven Komponente gesucht und für diese eine unendliche untere Schranke gespeichert. Um bei den Perturbationen der Probleme entsprechend die Koeffizienten beider Komponenten zu stören, wird die Zuordnung in einem array gespeichert. Für die Berechnung der dualen Lösung verwendet lp_solve die Darstellung (3.11); der Kostenvektor ergibt sich als Linearkombination der Gradienten der Nebenbedingungen. Allerdings werden hier wieder Daten für das Originalproblem geliefert, so dass die Lagrange-Parameter zum einen zu $\geq$-Ungleichungen gehören können, zum anderen der Gradient der Zielfunktion durch Wechsel von Minimierung zu Maximierung seine Richtung ändert. Im Falle einer Minimierung müssen somit die Lagrange-Parameter von $\geq$-Nebenbedingungen mit -1 multipliziert werden, bei Maximierung diejenigen der $\leq$-Nebenbedingungen und der Gleichungen. Um die Berechnung von Schranken im Falle von Unbeschränktheit und Unlösbarkeit des primalen Problems zu ermöglichen, wird lp_solve vor dem Lesen von primaler und dualer Lösung dazu angewiesen, diese neu zu berechnen. lp_solve tut dies nicht von selbst, wenn das Problem anscheinend keine optimale Lösung besitzt. Hierzu mussten die verantwortlichen Methoden `construct_solution` und `calculate_duals` in die Schnittstelle von lp_solve aufgenommen werden, da diese normalerweise nur intern genutzt werden können.

Zum Setzen der perturbierten rechten Seiten in `solveAPertLp` bietet lp_solve keine eigene Methode an. Daher werden diese Änderungen direkt in der Datenstruktur vorgenommen, die auch nur $\leq$-Ungleichungen kennt, Vorzeichenwechsel sind hier somit nicht vonnöten. Dafür ist zu berücksichtigen, ob das lp_solve Modell Skalierung verwendet. Die Perturbation der einfachen Schranken erfolgt mit den vorhandenen Methoden zum Setzen selbiger. Falls das Problem unbeschränkt zu sein scheint, erfolgt auch hier der Aufruf von lp_solves Routine zur Konstruktion einer Lösung. Diese wird gespeichert, so dass der Algorithmus versucht, aus ihr eine Schranke abzuleiten.

In `solveCPertLp` wird die bereitgestellte Methode zum Setzen der Koeffizienten von Zielfunktion und Nebenbedingungen verwendet. Beim Setzen der perturbierten Zielfunktion ist zu beachten, ob ursprünglich minimiert oder maximiert wird, da lp_solve bei Maximierung die übergebenen Werte mit -1 multipliziert, um sie in seine interne Darstellung, Minimierung der Zielfunktion, zu überführen. Hier wird, bei Unzulässigkeit des primalen Problems, die Methode zur Bestimmung der dualen Lösung aufgerufen, um mit dieser die Schrankenbestimmung zu versuchen.

Bei der Wiederherstellung des Originalproblems nach Abschluss der Algorithmen ist für die obere Schranke zu beachten, ob Skalierung verwendet wird, und für die untere Schranke, ob das Ursprungsproblem maximiert wird.

4.3.7 CLPM_lps40

Die Schnittstelle zu lp_solve 4.0 beinhaltet keine großen Unterschiede gegenüber der zu Version 3.2. Es lassen sich einige Daten mit Methoden auslesen oder speichern, für die dies mit Version 3.2 noch nicht funktionierte, so zum Beispiel die rechten Seiten der Nebenbedingungen. Hier muss man nun nicht mehr darauf achten, ob Skalierung verwendet wird oder nicht, dafür ist nun die Art der Nebenbedingung von Bedeutung. Handelt es sich im LP um eine $\geq$-Bedingung, so ist wieder eine Multiplikation mit -1 nötig. Auch hier wurde aufgrund der Weiterentwicklung gegen die Methoden programmiert.

Ebenfalls mussten `construct_solution` und `calculate_duals` von außerhalb von lp_solve aufrufbar gemacht werden. Größter Unterschied ist jedoch die direkte Unterstützung freier Variabler durch lp_solve 4.0. So entfällt hier der zusätzliche Aufwand, diese zu rekonstruieren und die Zuordnung der Komponenten zu speichern.

4.4 Anwendung

Nach der Instanziierung eines LP-Moduls wird dieses über `setLpModule` an Lurupa übergeben, und die weitere Kommunikation erfolgt nun über dieses Objekt.

```
CLurupa Lurupa(epsilon);
CLPM_sample sample;
Lurupa.setLpModule(&sample);
```

Als nächstes muss das LP mit dem LP-Löser gelöst werden. Ist dies geschehen, werden die Daten Lurupa übergeben. Zusätzlich wird das Urteil des Lösers über die Lösbarkeit des LPs benötigt, da dieses, zum Beispiel im Falle von lp_solve, nicht unbedingt in den Datenstrukturen auftaucht. Hier kann eine Unschärfe der Daten mit angegeben werden, die, im Falle von `CLPM_lps32` und `CLPM_lps40`, die LP Parameter als die Mittelpunktsparameter eines Intervallproblems mit einem relativen Radius von `intData` behandelt.

```
Lurupa.setLp(&model, &feasible, intData);
```

Dieser Aufruf wird an `sample` delegiert.

Nun kann mit

```
statusUp = Lurupa.upperBound(upBound);
int iterationsUp = Lurupa.getIteration();
statusLo = Lurupa.lowerBound(loBound);
int iterationsLo = Lurupa.getIteration();
```

die obere bzw. untere Schranke des optimalen Zielfunktionswertes berechnet werden, wobei die Schranke in `upBound` bzw. `loBound` gespeichert wird. Aus dem Rückgabewert der Funktion lässt sich ablesen, ob die Schrankenberechnung erfolgreich war, oder falls nicht, weshalb sie fehlgeschlagen ist. Mögliche Rückgabewerte sind:

`LU_B_OK`	Schrankenberechnung erfolgreich
`LU_B_ORGINF`	Keine Schranke - Originalproblem scheint unlösbar zu sein
`LU_B_NOENC`	Keine Schranke - Lösungsmenge von $Bx = b$ konnte nicht eingeschlossen werden[1]
`LU_B_WARN`	Keine Schranke - LP-Löser gab eine Warnung während des Lösungsprozesses
`LU_B_RANK`	Keine Schranke - Gleichungsmatrix B hat nicht maximalen Rang
`LU_B_ITER`	Keine Schranke - maximale Anzahl von Iterationen erreicht
`LU_B_FAILURE`	Keine Schranke - LP-Löser hat Lösung mit Fehlermeldung abgebrochen

`Lurupa.getIteration()` gibt die Anzahl der benötigten Iterationen für die letzte Schrankenberechnung zurück.

Kapitel 5

Numerische Resultate

Um die Qualität der Schranken und ihren Bezug zu den Konditionszahlen von Ordonez und Freund [15] zu prüfen, wurden die Probleme der Netlib [12] als Testmenge verwendet. Auch hier wurden die Probleme mit ranges, *boeing1, boeing2, forplan, nesm* und *seba* außer acht vorgelassen. Des weiteren die fünf Probleme, die zum Zeitpunkt der Arbeit von Ordonez und Freund nicht im MPS Format vorlagen, *qap8, qap12, qap15, stocfor3* und *truss*. Ferner die fünf Probleme, für die Ordonez und Freund innerhalb einer Woche keine Konditionszahl berechnen konnten, *dfl001, fit2p, maros-r7, pilot* und *pilot87*. Somit blieben 83 Probleme in der Testmenge. Es wurden Schranken mit lp_solve 3.2 und 4.0 als LP-Löser bestimmt. Als Testsystem kam ein Athlon 500 mit 256MB Hauptspeicher unter Red Hat Linux 7.3 zum Einsatz. Compiler war der gcc 2.96.

5.1 Netlib lp library

Als allgemein anerkannte Menge von Testproblemen für Untersuchungen hat sich die Netlib lp library etabliert. Hier werden LPs ,die in den Anwendungen aufgetreten sind, gesammelt und zur Verfügung gestellt. Die folgenden Auszüge aus der readme-Datei zeigen die Breite der Anwendungen.

STOCFOR1,2,3 are stochastic forestry problems from Gus Gassmann. To quote Gus, “All of them are seven-period descriptions of a forestry problem with a random occurrence of forest fires, and the size varies according to the number of realizations you use in each period.“ STOCFOR1 “is the deterministic version, STOCFOR2 has 2 realizations each in periods 2 to 7, and the monster STOCFOR3 has 4,4,4,2,2, and 2 realizations, respectively.“

BLEND, BOEING1, BOEING2 Concerning the problems he supplied, Nick Gould says that BLEND "is is a variant of the [oil refinery] problem in Murtagh's book (the coefficients are different) which I understand John Reid obtained from the people at NPL (Gill and Murray?); they were also the original sources for the SC problems"; BOEING1 and BOEING2 "have to do with flap settings on aircraft for economical operations"; PEROLD "is another Pilot model (Pilot1)"; and FINNIS "is from Mike Finnis at Harwell, a model for the selection of alternative fuel types."

LOTFI LOTFI, says Vahid Lotfi, "involves audit staff scheduling. This problem is semi real world and we have used it in a study, the results of which are to appear in Decision Sciences (Fall 1990). The detailed description of the problem is also in the paper. The problem is actually an MOLP with seven objectives, the first is maximization and the other six are minimization. The version that I am sending has the aggregated objective (i.e., z1-z2-z3-z4-z5-z6-z7)."

DFL001 DFL001, says Marc Meketon, "is a 'real-world' airline schedule planning (fleet assignment) problem. This LP was preprocessed by a modified version of the KORBX(r) System preprocessor. The problem reduced in size (rows, columns, non-zeros) significantly. The row and columns were randomly sorted and renamed, and a fixed adjustment to the objective function was eliminated. The name of the problem is derived from the initials of the person who created it."

D6CUBE Of D6CUBE, Robert Hughes says, "Mike Anderson and I are working on the problem of finding the minimum cardinality of triangulations of the 6-dimensional cube. The optimal objective value of the problem I sent you provides a lower bound for the cardinalities of all triangulations which contain a certain simplex of volume 8/6! and which contains the centroid of the 6-cube in its interior. The linear programming problem is not easily described."

MAROS, MAROS-R7, MODSZK1 Concerning the problems he submitted, Istvan Maros says that MAROS is an industrial production / allocation model about which "the customer does not want to reveal the exact meaning". MAROS-R7 is "an interesting real-life LP problem which appeared hard to some solvers." It "is an image restoration problem done via a goal programming approach. It is structured, namely, its first section is a band matrix with the dominating number of nonzeros, while the second section is also a band matrix with bandwidth equals 2 and coefficients +1, -1. The problem is a representative of a family of problems in which the number of rows and the bandwidth of the first section can vary. This one is a medium size problem from the family. MAROS-R7 became available in cooperation with Roni Levkovitz and Carison Tong." MODSZK1 is a "real-life problem" that is "very degenerate" and on which a dual simplex algorithm "may require up to 10 times" fewer iterations than a primal simplex algorithm. It "is a multi-sector economic planning model (a kind of an input/output model in economy)" and "is an old problem of mine and it is not easy to recall more."

GREENBEA, GRENNBEB A large refinery model; see the book "A Model-Management Framework for Mathematical Programming" by Kenneth H. Palmer et al. (John Wiley & Sons, New York, 1984).

GFRD-PNC, SIERRA R. Helgason, J. Kennington, and P. Wong, "An Application of Network Programming for National Forest Planning", Technical Report OR 81006, Dept. of Operations Research, Southern Methodist University.

NESM Gerald Brown, Naval Postgraduate School.

FIT1D, FIT1P, FIT2D, FIT2P Concerning these Bob Fourer says "The pairs FIT1P/FIT1D and FIT2P/FIT2D are primal and dual versions of the same two problems [except that we have negated the cost coefficients of the dual problems so all are minimization problems]. They originate from a model for fitting linear inequalities to data, by minimization of a sum of piecewise-linear penalties. The FIT1 problems are based on 627 data points and 2-3 pieces per primal pl penalty term. The FIT2 problems are based on 3000 data points (from a different sample altogether) and 4-5 pieces per pl term."

Um eine Vorstellung davon zu bekommen, welche Dimension die Probleme in der Netlib sind und wie häufig die unterschiedlichen Schrankentypen vorkommen, zeigt Tabelle 5.1 eine Übersicht. Die Probleme mit angehängtem „(R)" am Namen enthalten ranges. Es fehlen die Probleme *qap8, qap12, qap15*, die nicht als MPS Datei in der Netlib enthalten sind, sondern nur über einen Generator erzeugt werden können. Die Spalten enthalten folgende Werte:

n Anzahl der Variablen im LP
m + p Anzahl der Nebenbedingungen (Ungleichungen + Gleichungen)
UP einfache obere Schranken für Variable
LO einfache untere Schranken für Variable
FX fixierte Variable (obere und untere einfache Schranke gleich)
FR freie Variable (weder obere noch untere einfache Schranke)
PL positive Variable (nur untere einfache Schranke von 0)
UN Variable mit mindestens einer unbeschränkten Richtung

Tabelle 5.1: Dimension und Schrankentypen in der Netlib

			bound types					
Problemname	n	m + p	UP	LO	FX	FR	PL	UN
25FV47	1571	821						√
80BAU3B	9799	2262	√	√	√			√
ADLITTLE	97	56						√
AFIRO	32	27						√
AGG	163	488						√
AGG2	302	516						√
AGG3	302	516						√
BANDM	472	305						√
BEACONFD	262	173						√
BLEND	83	74						√
BNL1	1175	643						√
BNL2	3489	2324						√
BOEING1(R)	384	350	√	√				√
BOEING2(R)	143	166	√	√				√
BORE3D	315	233	√	√	√			√
BRANDY	249	220						√
CAPRI	353	271	√		√	√		√
CYCLE	2857	1903	√			√		√
CZPROB	3523	929			√			√
D2Q06C	5167	2171						√
D6CUBE	6184	415		√				√
DEGEN2	534	444						√
DEGEN3	1818	1503						√
DFL001	12230	6071	√					√
E226	282	223						√
ETAMACRO	688	400	√	√	√			√
FFFFF800	854	524						√
FINNIS	614	497	√	√	√			√
FIT1D	1026	24	√					
FIT1P	1677	627	√					√
FIT2D	10500	25	√					
FIT2P	13525	3000	√					√

Problemname	n	m + p	bound types UP	LO	FX	FR	PL	UN
FORPLAN(R)	421	161	$\surd$		$\surd$			$\surd$
GANGES	1681	1309	$\surd$	$\surd$				$\surd$
GFRD-PNC	1092	616	$\surd$	$\surd$				$\surd$
GREENBEA	5405	2392	$\surd$	$\surd$	$\surd$			$\surd$
GREENBEB	5405	2392	$\surd$	$\surd$	$\surd$	$\surd$		$\surd$
GROW15	645	300	$\surd$					$\surd$
GROW22	946	440	$\surd$					$\surd$
GROW7	301	140	$\surd$					$\surd$
ISRAEL	142	174						$\surd$
KB2	41	43	$\surd$					$\surd$
LOTFI	308	153						$\surd$
MAROS	1443	846		$\surd$	$\surd$			$\surd$
MAROS-R7	9408	3136						$\surd$
MODSZK1	1620	687				$\surd$		$\surd$
NESM(R)	2923	662	$\surd$	$\surd$	$\surd$			$\surd$
PEROLD	1376	625	$\surd$	$\surd$	$\surd$	$\surd$		$\surd$
PILOT	3652	1441	$\surd$	$\surd$	$\surd$			$\surd$
PILOT.JA	1988	940	$\surd$	$\surd$	$\surd$	$\surd$		$\surd$
PILOT.WE	2789	722	$\surd$	$\surd$	$\surd$	$\surd$		$\surd$
PILOT4	1000	410	$\surd$		$\surd$	$\surd$	$\surd$	$\surd$
PILOT87	4883	2030	$\surd$	$\surd$	$\surd$			$\surd$
PILOTNOV	2172	975	$\surd$		$\surd$			$\surd$
RECIPE	180	91	$\surd$	$\surd$	$\surd$			$\surd$
SC105	103	105						$\surd$
SC205	203	205						$\surd$
SC50A	48	50						$\surd$
SC50B	48	50						$\surd$
SCAGR25	500	471						$\surd$
SCAGR7	140	129						$\surd$
SCFXM1	457	330						$\surd$
SCFXM2	914	660						$\surd$
SCFXM3	1371	990						$\surd$
SCORPION	358	388						$\surd$
SCRS8	1169	490						$\surd$
SCSD1	760	77						$\surd$
SCSD6	1350	147						$\surd$
SCSD8	2750	397						$\surd$
SCTAP1	480	300						$\surd$
SCTAP2	1880	1090						$\surd$
SCTAP3	2480	1480						$\surd$
SEBA(R)	1028	515	$\surd$	$\surd$				$\surd$
SHARE1B	225	117						$\surd$
SHARE2B	79	96						$\surd$
SHELL	1775	536	$\surd$	$\surd$	$\surd$			$\surd$
SHIP04L	2118	402						$\surd$
SHIP04S	1458	402						$\surd$
SHIP08L	4283	778						$\surd$

			bound types					
Problemname	n	m + p	UP	LO	FX	FR	PL	UN
SHIP08S	2387	778						√
SHIP12L	5427	1151						√
SHIP12S	2763	1151						√
SIERRA	2036	1227	√					
STAIR	467	356	√		√	√		√
STANDATA	1075	359	√		√			√
STANDGUB	1184	361	√		√			√
STANDMPS	1075	467	√		√			√
STOCFOR1	111	117						√
STOCFOR2	2031	2157						√
STOCFOR3	15695	16675						√
TRUSS	8806	1000						√
TUFF	587	333	√	√	√	√		√
VTP.BASE	203	198	√	√	√	√		√
WOOD1P	2594	244						√
WOODW	8405	1098						√

Es zeigt sich die Spannweite in den Dimensionen von 32 (afiro) bis 15695 (stocfor3) Variable, von 24 (fit1d) bis 16675 (stocfor3) Nebenbedingungen. Die Bedeutung der Behandlung von Variablen mit mindestens einer unendlichen einfachen Schranke wird deutlich, denn von den 95 enthaltenen Problemen haben nur 3 endliche untere und obere Schranken für alle ihre Variablen.

5.2 lp_solve 3.2

Bei der Berechnung der Schranken mit lp_solve 3.2 wurde ε auf den in lp_solve voreingestellten Wert $1 \cdot 10^{-6}$ gesetzt. Bei der Lösung fällt als erstes auf, dass lp_solve bei Problemen während des Lösungsvorganges die Programmausführung abbricht. Somit ist keine weitere Fehlerbehandlung oder Interpretation möglich. Bei den LPs *25fv47, bandm, bnl1, bnl2, d2q06c, d6cube, ffff800, greenbea, greenbeb, israel, maros, perold, pilot4, pilotnov, pilot-ja, pilot-we, scfxm3, scsd1, scsd6, scsd8, stair, tuff* und *wood1p* konnte aufgrund numerischer Probleme von lp_solve während der Lösung des Originalproblems keine Schranke berechnet werden. Bei *cycle* wurde der Lösungsvorgang nach 10 Stunden ergebnislos abgebrochen. Mit Lösung des Originalproblems, aber nach 10 Stunden ergebnisloser Iteration, wurde die Schrankenberechnung bei *degen3, e226, etamacro* und *grow15* abgebrochen. Somit bleiben 52 Probleme als Testmenge.

Tabelle 5.2 zeigt die Messergebnisse für die untere Schranke, Tabelle 5.3 für die obere Schranke. Bei der Berechnung der unteren Schranke von *modszk1* brach lp_solve wegen numerischer Probleme ab. Die Berechnung der oberen

Schranke von *80bau3b* brach reproduzierbar aus nicht ersichtlichen Gründen ab. In den Tabellen folgt auf den Namen des Problems die Distanz zur Unzulässigkeit: bei der unteren Schranke zur dualen Unzulässigkeit, bei der oberen zur primalen. Hierbei ist die Distanz für das Originalproblem – vor Preprocessing – angegeben, da die Probleme auch in dieser Version an Lurupa übergeben werden. Die beiden nächsten Spalten geben die approximative Lösung von lp_solve und die hierzu benötigte Zeit in Sekunden an. Die letzten drei Spalten enthalten die berechnete Schranke, die benötigte Anzahl an Iterationen und das Verhältnis der Zeit für die Lösung zu der Zeit für die Schrankenberechnung. `ITER` bedeutet, dass die Schrankenberechnung wegen Überschreitung der Höchstzahl von Iterationen abgebrochen wurde. In diesem Fall wurde die Anzahl der Iterationen auf 31 gesetzt. `RANK` bedeutet, dass die Matrix des Gleichungssystems, dessen Lösung zur Schrankenberechnung einzuschließen ist, nicht vollen Rang hatte. Eine Einschließung und Schrankenberechnung war somit nicht möglich. 0 Iterationen bedeuten entweder, dass im Falle eines Rangdefizits keine Iterationen nötig waren, um dies festzustellen, oder dass schon aus der Näherungslösung des Originalproblems eine Schranke abgeleitet werden konnte; es waren keine Perturbationen nötig. Ein „–“ als Zeitverhältnis bedeutet, dass die benötigten Zeiten außerhalb des Messbereiches lagen und somit kein Verhältnis berechnet werden konnte.

Tabelle 5.2: Untere Schranke mit lp_solve 3.2

Name	ρ_D	f^*	t_s	$\underline{f^*}$	$i_{\underline{f^*}}$	$\frac{t_{f^*}}{t_{\underline{f^*}}}$
80BAU3B	0	$9.8722e+05$	34.210	$9.8713e+05$	8	5.710
ADLITTLE	0.051651	$2.2549e+05$	0.010	$2.2549e+05$	1	1.000
AFIRO	1.000000	$-4.6475e+02$	0.000	$-4.6476e+02$	1	–
AGG2	0.771400	$-2.0239e+07$	0.050	$-2.0240e+07$	1	5.000
AGG3	0.771400	$1.0312e+07$	0.050	$1.0311e+07$	1	4.800
AGG	0.771400	$-3.5992e+07$	0.030	$-3.5993e+07$	5	6.333
BEACONFD	0	$3.3592e+04$	0.030	$3.3578e+04$	6	3.667
BLEND	0.040726	$-3.0812e+01$	0.010	$-3.0880e+01$	9	2.000
BORE3D	0.003539	$1.3731e+03$	0.070	ITER	31	7.714
BRANDY	0	$1.5185e+03$	0.240	ITER	31	1.583
CAPRI	0.095510	$2.6900e+03$	0.090	$1.2167e+03$	18	7.444
CZPROB	0.008807	$2.1852e+06$	3.840	$2.1852e+06$	1	1.539
FINNIS	0	$1.7279e+05$	0.280	ITER	31	3.250
FIT1D	∞	$-9.1464e+03$	0.750	$-9.1464e+03$	0	0.013
FIT1P	0.437500	$9.1464e+03$	4.580	$9.1462e+03$	5	0.666
FIT2D	∞	$-6.8464e+04$	140.640	$-6.8464e+04$	0	0.001
GANGES	1.000000	$-1.0959e+05$	2.410	$-1.0970e+09$	11	6.000
GFRD-PNC	0.347032	$6.9022e+06$	0.280	$6.9019e+06$	1	3.321
GROW22	0.968073	$-1.6083e+08$	4.200	$-1.6091e+08$	15	0.783
GROW7	0.968073	$-4.7788e+07$	0.100	$-4.7801e+07$	12	2.100
KB2	0.018802	$-1.7499e+03$	0.000	$-1.7842e+03$	13	–

Name	ρ_D	f^*	t_s	$\underline{f^*}$	$i_{\underline{f^*}}$	$\frac{t_{\underline{f^*}}}{t_{f^*}}$
LOTFI	0	$-2.5265e+01$	0.060	ITER	31	2.167
MODSZK1	0.108469	$3.2093e+02$	0.520			
RECIPE	0	$-2.6662e+02$	0.010	$-2.6662e+02$	0	1.000
SC105	0.133484	$-5.2202e+01$	0.010	$-1.1863e+06$	11	6.000
SC205	0.010023	$-5.2202e+01$	0.070	$-1.0363e+13$	14	4.714
SC50A	0.562500	$-6.4575e+01$	0.010	$-3.0777e+06$	12	2.000
SC50B	0.421875	$-7.0000e+01$	0.000	$-1.9049e+02$	9	–
SCAGR25	0.034646	$-1.4753e+07$	0.470	$-1.4754e+07$	1	0.766
SCAGR7	0.034646	$-2.3314e+06$	0.020	$-2.3315e+06$	1	1.000
SCFXM1	0	$1.8417e+04$	0.230	ITER	31	3.652
SCFXM2	0	$3.6660e+04$	1.110	ITER	31	3.378
SCORPION	0.949393	$1.8781e+03$	0.120	$1.8781e+03$	5	3.417
SCRS8	0	$9.0430e+02$	0.790	$9.0416e+02$	8	2.911
SCTAP1	1.000000	$1.4123e+03$	0.210	$1.4122e+03$	1	1.143
SCTAP2	1.000000	$1.7248e+03$	1.170	$1.7248e+03$	1	3.256
SCTAP3	1.000000	$1.4240e+03$	2.020	$1.4240e+03$	1	3.975
SHARE2B	0.287893	$-4.1573e+02$	0.020	$-4.1592e+02$	8	1.500
SHELL	1.777778	$1.2088e+09$	0.440	$1.2088e+09$	0	0.727
SHIP04L	13.146000	$1.7933e+06$	0.430	$1.7933e+06$	1	2.884
SHIP04S	13.146000	$1.7987e+06$	0.270	$1.7987e+06$	1	3.111
SHIP08L	21.210000	$1.9091e+06$	1.660	$1.9090e+06$	1	3.542
SHIP08S	21.210000	$1.9201e+06$	0.700	$1.9201e+06$	1	4.643
SHIP12L	7.434000	$1.4702e+06$	3.460	$1.4702e+06$	1	3.662
SHIP12S	7.434000	$1.4892e+06$	1.510	$1.4892e+06$	1	4.139
SIERRA	∞	$1.5394e+07$	0.640	$1.5394e+07$	0	1.672
STANDATA	1.000000	$1.2577e+03$	0.030	$1.2577e+03$	0	3.667
STANDGUB	0	$1.2577e+03$	0.030	$1.2577e+03$	0	5.000
STANDMPS	1.000000	$1.4060e+03$	0.310	$1.4059e+03$	8	6.000
STOCFOR1	0.011936	$-4.1132e+04$	0.000	$-4.1134e+04$	2	–
STOCFOR2	0.000064	$-3.9024e+04$	5.390	$-3.9032e+04$	15	7.824
VTP.BASE	0.500000	$1.2983e+05$	0.100	$1.2979e+05$	6	0.900

Tabelle 5.3: Obere Schranke mit lp_solve 3.2

Name	ρ_P	f^*	t_s	$\overline{f^*}$	$i_{\overline{f^*}}$	$\frac{t_{\overline{f^*}}}{t_{f^*}}$
80BAU3B	0	$9.8722e+05$	34.210			
ADLITTLE	0	$2.2549e+05$	0.010	$2.2550e+05$	4	1.000
AFIRO	0.397390	$-4.6475e+02$	0.000	$-4.6475e+02$	7	–
AGG2	0	$-2.0239e+07$	0.050	ITER	31	14.000
AGG3	0	$1.0312e+07$	0.050	ITER	31	14.000
AGG	0	$-3.5992e+07$	0.030	ITER	31	11.667
BEACONFD	0	$3.3592e+04$	0.030	ITER	31	52.667
BLEND	0.003541	$-3.0812e+01$	0.010	$-3.0803e+01$	9	7.000
BORE3D	0	$1.3731e+03$	0.070	RANK	0	3.714
BRANDY	0	$1.5185e+03$	0.240	RANK	0	0.083
CAPRI	0.000252	$2.6900e+03$	0.090	$2.6902e+03$	5	11.556

Name	ρ_P	f^*	t_s	$\overline{f^*}$	$i_{\overline{f^*}}$	$\frac{t_{\overline{f^*}}}{t_{f^*}}$
CZPROB	0	$2.1852e+06$	3.840	ITER	31	137.117
FINNIS	0	$1.7279e+05$	0.280	ITER	31	5.107
FIT1D	3.500000	$-9.1464e+03$	0.750	$-9.1463e+03$	1	0.040
FIT1P	1.271887	$9.1464e+03$	4.580	$9.1464e+03$	13	39.939
FIT2D	317.000000	$-6.8464e+04$	140.640	$-6.8464e+04$	1	0.002
GANGES	0	$-1.0959e+05$	2.410	ITER	31	533.394
GFRD-PNC	0	$6.9022e+06$	0.280	$6.9022e+06$	7	273.750
GROW22	0.572842	$-1.6083e+08$	4.200	$-1.6083e+08$	16	18.290
GROW7	0.572842	$-4.7788e+07$	0.100	$-4.7788e+07$	15	20.500
KB2	0.000201	$-1.7499e+03$	0.000	$-1.7499e+03$	1	–
LOTFI	0.000306	$-2.5265e+01$	0.060	$-2.5265e+01$	6	6.167
MODSZK1	0	$3.2093e+02$	0.520	RANK	0	16.333
RECIPE	0	$-2.6662e+02$	0.010	RANK	0	1.000
SC105	0	$-5.2202e+01$	0.010	$-5.2201e+01$	1	3.000
SC205	0	$-5.2202e+01$	0.070	$-5.2201e+01$	1	2.000
SC50A	0	$-6.4575e+01$	0.010	$-6.4574e+01$	1	0.000
SC50B	0	$-7.0000e+01$	0.000	$-6.9999e+01$	1	–
SCAGR25	0.021077	$-1.4753e+07$	0.470	$-1.4753e+07$	6	23.936
SCAGR7	0.022644	$-2.3314e+06$	0.020	$-2.3314e+06$	3	7.000
SCFXM1	0	$1.8417e+04$	0.230	ITER	31	19.261
SCFXM2	0	$3.6660e+04$	1.110	ITER	31	32.721
SCORPION	0	$1.8781e+03$	0.120	ITER	31	167.833
SCRS8	0	$9.0430e+02$	0.790	ITER	31	52.747
SCTAP1	0.032258	$1.4123e+03$	0.210	$1.4123e+03$	9	5.667
SCTAP2	0.586563	$1.7248e+03$	1.170	$3.0181e+03$	18	182.795
SCTAP3	0.381250	$1.4240e+03$	2.020	$2.9288e+03$	18	115.020
SHARE2B	0.001747	$-4.1573e+02$	0.020	$-4.1567e+02$	8	1.500
SHELL	0	$1.2088e+09$	0.440	RANK	0	25.682
SHIP04L	0	$1.7933e+06$	0.430	RANK	0	1.163
SHIP04S	0	$1.7987e+06$	0.270	RANK	0	1.259
SHIP08L	0	$1.9091e+06$	1.660	RANK	0	1.422
SHIP08S	0	$1.9201e+06$	0.700	RANK	0	1.786
SHIP12L	0	$1.4702e+06$	3.460	RANK	0	1.526
SHIP12S	0	$1.4892e+06$	1.510	RANK	0	1.715
SIERRA	0	$1.5394e+07$	0.640	RANK	0	14.484
STANDATA	0	$1.2577e+03$	0.030	ITER	31	96.000
STANDGUB	0	$1.2577e+03$	0.030	RANK	0	3.667
STANDMPS	0	$1.4060e+03$	0.310	ITER	31	34.710
STOCFOR1	0.001203	$-4.1132e+04$	0.000	$-4.1132e+04$	10	–
STOCFOR2	0.000437	$-3.9024e+04$	5.390	$-3.9024e+04$	15	238.043
VTP.BASE	0	$1.2983e+05$	0.100	ITER	31	2.700

Die Messungen zeigen eine sehr gute Übereinstimmung zwischen den Ergebnissen von Ordonez und Freund und den berechneten Schranken. Bis auf wenige Ausnahmen erhält man genau dann eine untere bzw. obere Schranke, wenn die duale bzw. primale Distanz zur Unzulässigkeit von 0 verschieden

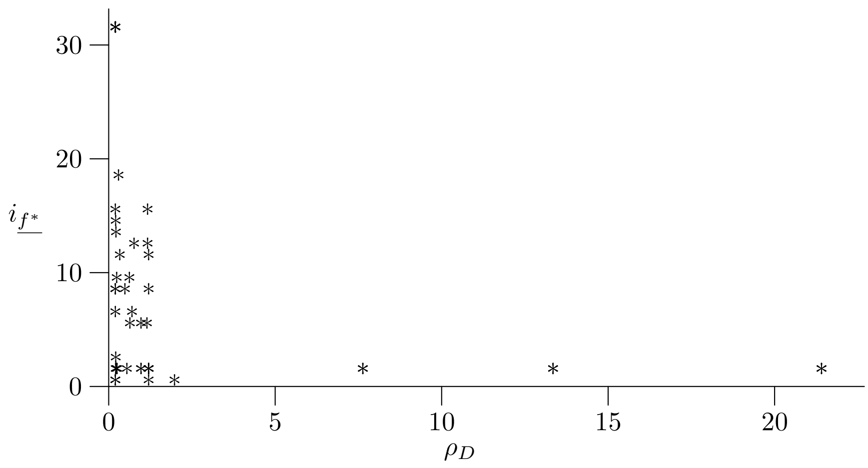


Abbildung 5.1: Zusammenhang zwischen dualem Abstand zur Unzulässigkeit und Anzahl benötigter Iterationen für die untere Schranke

ist.

Nur *bore3d, modszk1* liefern keine untere Schranke, obwohl der Abstand zur dualen Unzulässigkeit größer als 0 ist. Dagegen liefern *80bau3b, beaconfd, recipe, scrs8, standgub* eine untere bzw. *adlittle, gfrd-pnc, sc105, sc205, sc50a, sc50b* eine obere Schranke, obwohl der Abstand zur dualen bzw. primalen Unzulässigkeit 0 ist.

Abbildung 5.1 stellt die Anzahl der zur Schrankenbestimmung benötigten Iterationen dem Abstand zur dualen Unzulässigkeit gegenüber. Hierbei wurden die Probleme mit unendlichem Abstand *fit1d, fit2d, sierra* weggelassen, um den Zusammenhang nicht zu verzerren. Ebenfalls fehlt *modszk1*, da die Schrankenberechnung dort abbrach. Es fällt auf, dass im Bereich eines dualen Abstandes von 0 bis 1 ein ganzer Block von Problemen mit Iterationszahlen von 0 bis 18 liegt. 31 Iterationen stehen für die Probleme, deren Schrankenberechnung wegen Überschreitung der maximalen Iterationszahl abgebrochen wurde. Das Diagramm legt keinen funktionellen Zusammenhang zwischen den Größen nahe. Es sieht eher so aus, als ob ein zunehmender Abstand zur dualen Unzulässigkeit kleinere maximale Iterationszahlen bedingt. Da jedoch nur sieben Probleme einen dualen Abstand größer als 1 haben, scheint die Grundlage für derlei Aussagen nicht unbedingt gegeben.

Abbildung 5.2 stellt das Verhältnis von benötigter Zeit für die Berechnung der unteren Schranke zu der für die Lösung des Originalproblems gegen den dualen Abstand zur Unzulässigkeit dar. Auch hier wurden die Probleme *fit1d, fit2d, modszk1, sierra* außer acht gelassen. Zu erkennen ist wieder der Block von Problemen mit dualen Abständen zwischen 0 und 1 und Verhält-

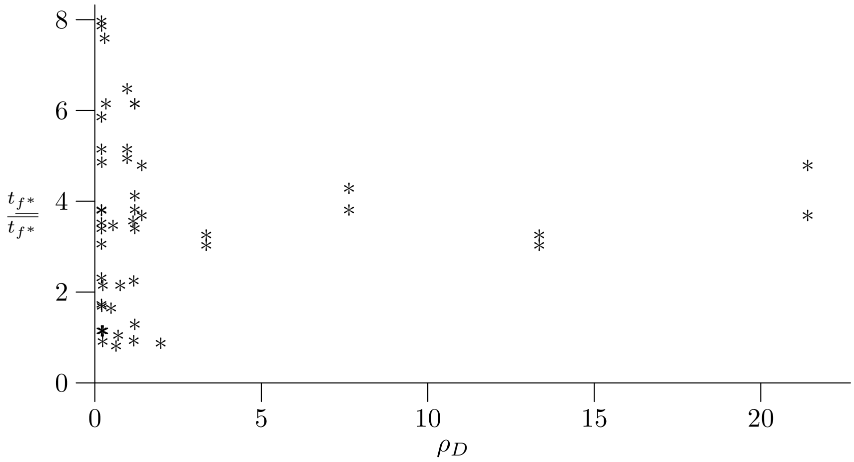


Abbildung 5.2: Zusammenhang zwischen dualem Abstand zur Unzulässigkeit und Verhältnis von benötigter Zeit zur unteren Schrankenberechnung zu benötigter Zeit zur Lösung des Originalproblems

nissen zwischen 0.666 und 7.824. Es drängt sich eher ein beschränkender als funktioneller Zusammenhang auf. Abermals ist die Datenbasis für Probleme mit einem dualen Abstand größer als 1 gering. Des weiteren war die Anzahl der benötigten Iterationen bei diesen Probleme einmal 0 und sechsmal 1, so dass hier nicht so sehr der Algorithmus zur Schrankenberechnung, sondern der Aufbau der benötigten Daten und Verwaltungsstrukturen samt Vorbereitungen für die Schrankenberechnung für die benötigte Zeit verantwortlich zeichnet.

Abbildung 5.3 enthält den Zusammenhang zwischen dem dualen Abstand zur Unzulässigkeit und der relativen Schärfe der unteren Schranke

$$\frac{|f^* - \underline{f^*}|}{|f^*|}.$$

Zusätzlich zu den Problemen *fit1d, fit2d, modszk1, sierra* wurden hier *recipe, shell, standata, standgub* weggelassen, da diese mit ihren extrem scharfen Schranken von $3.6 \cdot 10^{-16}$ und besser die Darstellung zu sehr verzerrt hätten. Wiederum zeigt sich der Block mit Schrankengüten von 0.001 und besser. Schlechtere Schranken treten für Probleme mit einem dualen Abstand größer als 1 nicht auf, nach wie vor bleibt jedoch zu beachten, dass in diesem Bereich hier sogar nur noch sechs Probleme liegen, da *shell* nicht mehr im Diagramm enthalten ist. Ein funktionaler Zusammenhang ist nicht zu erkennen.

Abbildungen 5.4, 5.5 und 5.6 enthalten die entsprechenden Zusammenhänge für die obere Schranke. Hier fehlen die Probleme *80bau3b*, da die

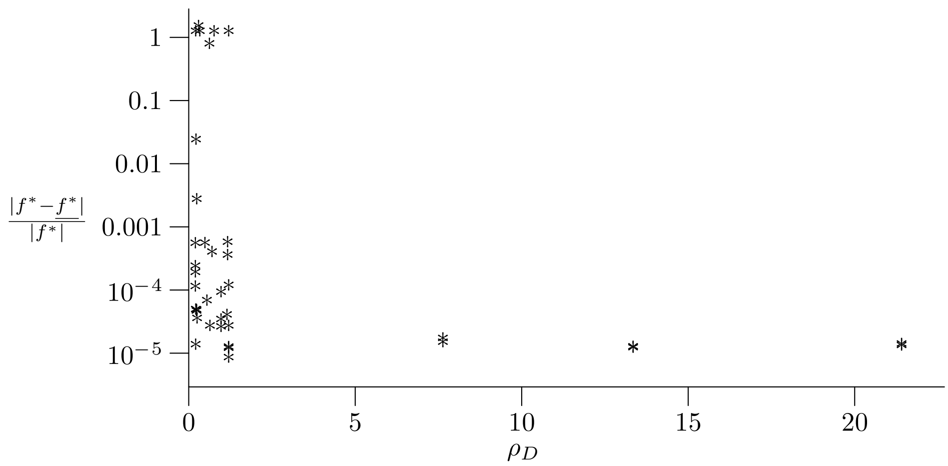


Abbildung 5.3: Zusammenhang zwischen dualem Abstand zur Unzulässigkeit und Güte der unteren Schranke

Schrankenberechnung abbrach, und *fit2d*, welches mit seinem primalen Abstand von 317 die Darstellung zu sehr verzerren würde. Bei der Darstellung des Zeitverhältnisses fehlen zusätzlich *afiro, kb2, sc50b, stocfor1*, da hier die Zeiten zur Lösung des Originalproblems unterhalb der Messgenauigkeit lagen. Die Darstellung der Schrankengüte enthält schließlich nur noch die 24 Probleme, die eine Schrankenberechnung ermöglichten, abzüglich *fit2d* wegen seines primalen Abstandes von 317.

Es zeigt sich wieder eine Blockbildung, wobei sich dieser in den primalen Abständen nahe der 0 ansiedelt. Die Probleme außerhalb dieses Blockes legen eine nach oben beschränkende Wirkung auch des primalen Abstandes nahe. Allerdings sind auch hier nur sieben Probleme mit einem primalen Abstand größer als 0.3 zu finden. Auffällig ist, dass zur Berechnung der oberen Schranke weit mehr Zeit benötigt wird als für die der unteren; die Zeitverhältnisse sind deutlich größer.

5.3 lp_solve 4.0

Bei der Berechnung der Schranken mit lp_solve 4.0 wurde ε auf den in lp_solve voreingestellten Wert $1 \cdot 10^{-9}$ gesetzt. Dies ist zwar genauer als bei lp_solve 3.2, aber da sich auch die Algorithmen unterscheiden, schien dies nicht unvernünftig. Es ergibt sich ein qualitativer Vergleich von zwei LP Lösern mit ihren Standardeinstellungen. Bei den LPs *bnl1, d2q06c, ffff800, greenbeb, grow22, maros, perold, pilot4, pilotnov, pilot-ja, pilot-we, scsd1, scsd6, scsd8, tuff* konnte aufgrund numerischer Probleme während der Lö-

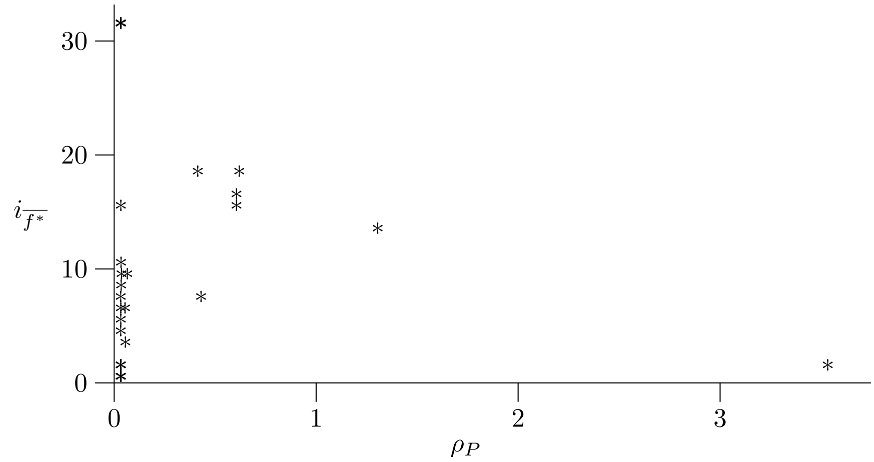


Abbildung 5.4: Zusammenhang zwischen primalem Abstand zur Unzulässigkeit und Anzahl benötigter Iterationen für die obere Schranke

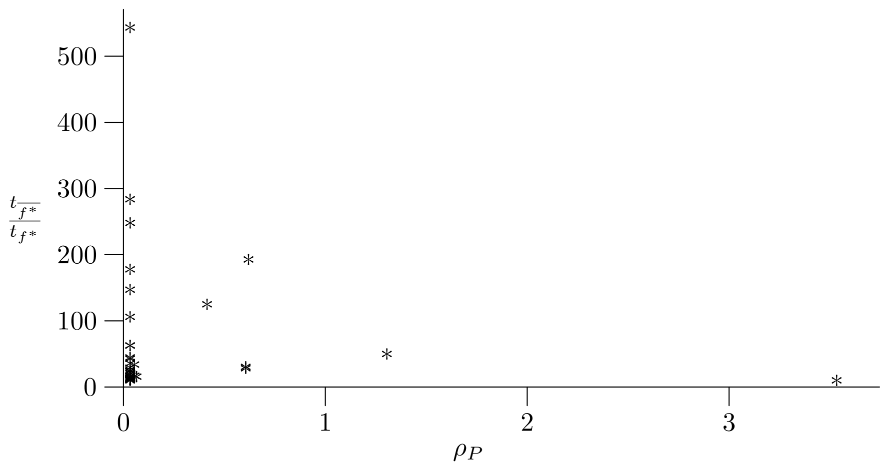


Abbildung 5.5: Zusammenhang zwischen primalem Abstand zur Unzulässigkeit und Verhältnis von benötigter Zeit zur oberen Schrankenberechnung zu benötigter Zeit zur Lösung des Originalproblems

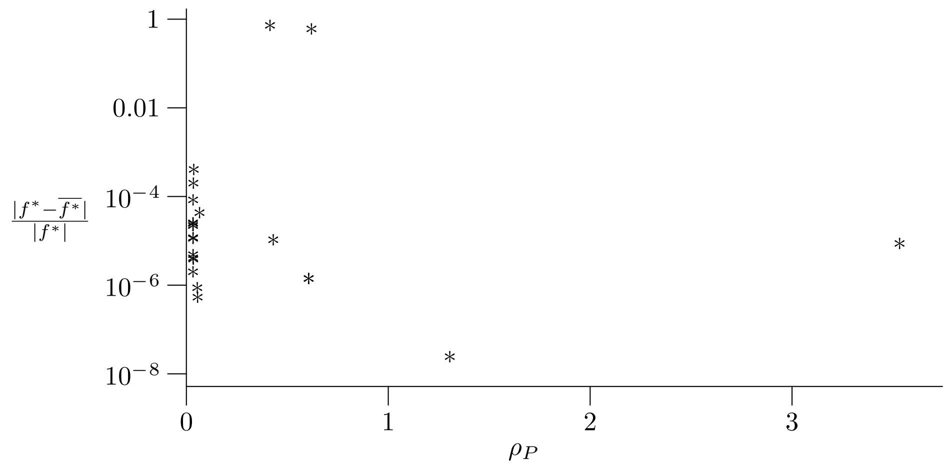


Abbildung 5.6: Zusammenhang zwischen primalem Abstand zur Unzulässigkeit und Güte der oberen Schranke

sung des Originalproblems mit lp_solve keine Schranke berechnet werden. Bei *25fv47*, *degen3* wurde der Lösungsvorgang nach 10 Stunden ergebnislos abgebrochen. Somit bleiben 66 Probleme als Testmenge.

Tabellen 5.4 und 5.5 zeigen die Messergebnisse für die untere und obere Schranke unter Verwendung von lp_solve 4.0. Auch hier brach die Berechnung der oberen Schranke von *80bau3b* reproduzierbar ab. Die Bedeutung der Tabellenspalten ist die gleiche wie bei den Tabellen für lp_solve 3.2.

Tabelle 5.4: Untere Schranke mit lp_solve 4.0

Name	ρ_D	f^*	t_s	$\underline{f^*}$	$i_{\underline{f^*}}$	$\frac{t_{\underline{f^*}}}{t_{f^*}}$
80BAU3B	0	$9.8722e+05$	55.420	$9.8722e+05$	1	1.378
ADLITTLE	0.051651	$2.2549e+05$	0.010	$2.2549e+05$	1	1.000
AFIRO	1.000000	$-4.6475e+02$	0.000	$-4.6475e+02$	1	–
AGG2	0.771400	$-2.0239e+07$	0.100	$-2.0239e+07$	1	2.900
AGG3	0.771400	$1.0312e+07$	0.090	$1.0312e+07$	1	3.222
AGG	0.771400	$-3.5992e+07$	0.070	$-3.5992e+07$	1	1.714
BANDM	0.000418	$-1.5863e+02$	1.080	$-1.5863e+02$	1	0.380
BEACONFD	0	$3.3592e+04$	0.040	$3.3592e+04$	1	2.250
BLEND	0.040726	$-3.0812e+01$	0.010	$-3.0812e+01$	1	2.000
BNL2	0	$1.8112e+03$	64.880	$1.8112e+03$	5	0.993
BORE3D	0.003539	$1.3731e+03$	0.150	$1.3731e+03$	1	1.133
BRANDY	0	$1.5185e+03$	0.380	ITER	31	2.000
CAPRI	0.095510	$2.6900e+03$	0.200	$2.6900e+03$	1	1.000
CYCLE	0	$-5.2264e+00$	83.270	$-5.2264e+00$	4	0.353
CZPROB	0.008807	$2.1852e+06$	5.940	$2.1852e+06$	1	1.212
D6CUBE	2.000000	$3.1549e+02$	46.560	$3.1549e+02$	1	0.152
DEGEN2	1.000000	$-1.4352e+03$	27.900	$-1.4352e+03$	1	0.038

Name	ρ_D	f^*	t_s	$\underline{f^*}$	$i_{\underline{f^*}}$	$\frac{t_{\underline{f^*}}}{t_{f^*}}$
E226	0	$-2.5865e+01$	0.330	$-1.8752e+01$	1	0.424
ETAMACRO	0.200000	$-7.5572e+02$	0.570	$-7.5572e+02$	5	2.614
FINNIS	0	$1.7279e+05$	0.470	ITER	31	2.915
FIT1D	∞	$-9.1464e+03$	1.000	$-9.1464e+03$	0	0.010
FIT1P	0.437500	$9.1464e+03$	15.470	$9.1464e+03$	1	0.194
FIT2D	∞	$-6.8464e+04$	143.330	$-6.8464e+04$	0	0.001
GANGES	1.000000	$-1.0959e+05$	5.060	$-1.0959e+05$	1	1.012
GFRD-PNC	0.347032	$6.9022e+06$	0.600	$6.9022e+06$	1	1.900
GREENBEA	0	INFEASIBLE	8.310	FAILURE	2	7.884
GROW15	0.968073	$-1.0687e+08$	2.690	$-1.0687e+08$	1	0.204
GROW7	0.968073	$-4.7788e+07$	0.300	$-4.7788e+07$	1	0.367
ISRAEL	0.166850	$-8.9664e+05$	0.190	$-8.9664e+05$	1	0.316
KB2	0.018802	$-1.7499e+03$	0.010	$-1.7499e+03$	1	1.000
LOTFI	0	$-2.5265e+01$	0.100	ITER	31	2.300
MODSZK1	0.108469	$3.2062e+02$	1.840	$3.2057e+02$	1	1.793
RECIPE	0	$-2.6662e+02$	0.010	$-2.6662e+02$	0	0.000
SC105	0.133484	$-5.2202e+01$	0.030	$-5.2202e+01$	1	1.333
SC205	0.010023	$-5.2202e+01$	0.260	$-5.2202e+01$	4	2.692
SC50A	0.562500	$-6.4575e+01$	0.000	$-6.4575e+01$	1	–
SC50B	0.421875	$-7.0000e+01$	0.010	$-7.0000e+01$	1	1.000
SCAGR25	0.034646	$-1.4753e+07$	0.900	$-1.4753e+07$	1	0.556
SCAGR7	0.034646	$-2.3314e+06$	0.040	$-2.3314e+06$	1	0.500
SCFXM1	0	$1.8417e+04$	0.430	ITER	31	2.488
SCFXM2	0	$3.6660e+04$	2.110	ITER	31	2.223
SCFXM3	0	$5.4901e+04$	6.200	ITER	31	1.784
SCORPION	0.949393	$1.8781e+03$	0.260	$1.8781e+03$	1	1.115
SCRS8	0	$9.0430e+02$	1.360	$9.0430e+02$	1	0.868
SCTAP1	1.000000	$1.4123e+03$	1.250	$1.4122e+03$	1	0.232
SCTAP2	1.000000	$1.7248e+03$	2.140	$1.7248e+03$	1	2.056
SCTAP3	1.000000	$1.4240e+03$	4.060	$1.4240e+03$	1	2.308
SHARE1B	0.000751	$-7.6589e+04$	0.150	$-7.6589e+04$	1	0.333
SHARE2B	0.287893	$-4.1573e+02$	0.030	$-4.1573e+02$	1	0.667
SHELL	1.777778	$1.2088e+09$	0.890	$1.2088e+09$	0	0.382
SHIP04L	13.146000	$1.7933e+06$	0.700	$1.7933e+06$	1	2.014
SHIP04S	13.146000	$1.7987e+06$	0.460	$1.7987e+06$	1	2.130
SHIP08L	21.210000	$1.9091e+06$	2.710	$1.9091e+06$	1	2.439
SHIP08S	21.210000	$1.9201e+06$	1.280	$1.9201e+06$	1	2.977
SHIP12L	7.434000	$1.4702e+06$	5.560	$1.4702e+06$	1	2.543
SHIP12S	7.434000	$1.4892e+06$	2.410	$1.4892e+06$	1	2.846
SIERRA	∞	$1.5394e+07$	1.260	$1.5394e+07$	0	0.810
STAIR	0	$-2.5127e+02$	2.370	$-2.5127e+02$	25	5.283
STANDATA	1.000000	$1.2577e+03$	0.070	$1.2577e+03$	0	1.571
STANDGUB	0	$1.2577e+03$	0.070	$1.2577e+03$	0	2.143
STANDMPS	1.000000	$1.4060e+03$	0.520	$1.4060e+03$	1	1.904
STOCFOR1	0.011936	$-4.1132e+04$	0.020	$-4.1132e+04$	1	1.000
STOCFOR2	0.000064	$-3.9024e+04$	9.010	$-3.9024e+04$	1	1.420
VTP.BASE	0.500000	$1.2983e+05$	0.120	$1.2983e+05$	1	0.417

Name	ρ_D	f^*	t_s	$\underline{f^*}$	$i_{\underline{f^*}}$	$\frac{t_{\underline{f^*}}}{t_{f^*}}$
WOOD1P	1.000000	$1.4429e+00$	3.950	$1.4429e+00$	1	4.038
WOODW	1.000000	$1.3045e+00$	14.310	$1.3045e+00$	1	3.080

Tabelle 5.5: Obere Schranke mit lp_solve 4.0

Name	ρ_P	f^*	t_s	$\overline{f^*}$	$i_{\overline{f^*}}$	$\frac{t_{\overline{f^*}}}{t_{f^*}}$
80BAU3B	0	$9.8722e+05$	55.420			
ADLITTLE	0	$2.2549e+05$	0.010	$2.2549e+05$	4	3.000
AFIRO	0.397390	$-4.6475e+02$	0.000	$-4.6475e+02$	6	–
AGG2	0	$-2.0239e+07$	0.100	ITER	31	12.900
AGG3	0	$1.0312e+07$	0.090	ITER	31	14.111
AGG	0	$-3.5992e+07$	0.070	ITER	31	8.571
BANDM	0	$-1.5863e+02$	1.080	ITER	31	21.954
BEACONFD	0	$3.3592e+04$	0.040	ITER	31	31.750
BLEND	0.003541	$-3.0812e+01$	0.010	$-3.0812e+01$	7	9.000
BNL2	0	$1.8112e+03$	64.880	ITER	31	24.489
BORE3D	0	$1.3731e+03$	0.150	RANK	0	1.733
BRANDY	0	$1.5185e+03$	0.380	RANK	0	0.053
CAPRI	0.000252	$2.6900e+03$	0.200	ITER	31	15.950
CYCLE	0	$-5.2264e+00$	83.270	RANK	0	0.136
CZPROB	0	$2.1852e+06$	5.940	ITER	31	74.296
D6CUBE	0	$3.1549e+02$	46.560	RANK	0	0.572
DEGEN2	0	$-1.4352e+03$	27.900	RANK	0	0.021
E226	0	$-2.5865e+01$	0.330	ITER	31	3.394
ETAMACRO	0	$-7.5572e+02$	0.570	RANK	0	1.825
FINNIS	0	$1.7279e+05$	0.470	ITER	31	4.468
FIT1D	3.500000	$-9.1464e+03$	1.000	$-9.1464e+03$	3	0.130
FIT1P	1.271887	$9.1464e+03$	15.470	$9.1464e+03$	14	14.051
FIT2D	317.000000	$-6.8464e+04$	143.330	$-6.8464e+04$	1	0.006
GANGES	0	$-1.0959e+05$	5.060	ITER	31	219.640
GFRD-PNC	0	$6.9022e+06$	0.600	$6.9022e+06$	6	118.433
GREENBEA	0	INFEASIBLE	8.310	RANK	0	59.848
GROW15	0.572842	$-1.0687e+08$	2.690	$-1.0687e+08$	14	10.472
GROW7	0.572842	$-4.7788e+07$	0.300	$-4.7788e+07$	11	8.633
ISRAEL	0.027248	$-8.9664e+05$	0.190	$-8.9664e+05$	1	0.263
KB2	0.000201	$-1.7499e+03$	0.010	$-1.7499e+03$	12	3.000
LOTFI	0.000306	$-2.5265e+01$	0.100	$-2.5265e+01$	5	3.300
MODSZK1	0	$3.2062e+02$	1.840	RANK	0	4.549
RECIPE	0	$-2.6662e+02$	0.010	RANK	0	1.000
SC105	0	$-5.2202e+01$	0.030	$-5.2202e+01$	1	1.333
SC205	0	$-5.2202e+01$	0.260	$-5.2202e+01$	1	1.077
SC50A	0	$-6.4575e+01$	0.000	$-6.4575e+01$	1	–
SC50B	0	$-7.0000e+01$	0.010	$-7.0000e+01$	1	1.000
SCAGR25	0.021077	$-1.4753e+07$	0.900	$-1.4753e+07$	6	13.522
SCAGR7	0.022644	$-2.3314e+06$	0.040	$-2.3314e+06$	3	4.000
SCFXM1	0	$1.8417e+04$	0.430	ITER	31	11.209

Name	ρ_P	f^*	t_s	$\overline{f^*}$	$i_{\overline{f^*}}$	$\frac{t_{\overline{f^*}}}{t_{f^*}}$
SCFXM2	0	$3.6660e+04$	2.110	ITER	31	18.664
SCFXM3	0	$5.4901e+04$	6.200	ITER	31	20.984
SCORPION	0	$1.8781e+03$	0.260	ITER	31	189.308
SCRS8	0	$9.0430e+02$	1.360	ITER	31	33.169
SCTAP1	0.032258	$1.4123e+03$	1.250	$1.4123e+03$	7	1.248
SCTAP2	0.586563	$1.7248e+03$	2.140	$2.3470e+03$	18	53.313
SCTAP3	0.381250	$1.4240e+03$	4.060	$2.1504e+03$	18	62.860
SHARE1B	0.000015	$-7.6589e+04$	0.150	$-7.6589e+04$	7	2.867
SHARE2B	0.001747	$-4.1573e+02$	0.030	$-4.1573e+02$	8	2.667
SHELL	0	$1.2088e+09$	0.890	RANK	0	12.753
SHIP04L	0	$1.7933e+06$	0.700	RANK	0	0.700
SHIP04S	0	$1.7987e+06$	0.460	RANK	0	0.739
SHIP08L	0	$1.9091e+06$	2.710	RANK	0	0.863
SHIP08S	0	$1.9201e+06$	1.280	RANK	0	1.039
SHIP12L	0	$1.4702e+06$	5.560	RANK	0	0.915
SHIP12S	0	$1.4892e+06$	2.410	RANK	0	1.025
SIERRA	0	$1.5394e+07$	1.260	RANK	0	7.333
STAIR	0.000580	$-2.5127e+02$	2.370	$-2.5127e+02$	1	0.958
STANDATA	0	$1.2577e+03$	0.070	ITER	31	56.571
STANDGUB	0	$1.2577e+03$	0.070	RANK	0	1.571
STANDMPS	0	$1.4060e+03$	0.520	ITER	31	26.712
STOCFOR1	0.001203	$-4.1132e+04$	0.020	$-4.1132e+04$	12	14.500
STOCFOR2	0.000437	$-3.9024e+04$	9.010	ITER	31	185.811
VTP.BASE	0	$1.2983e+05$	0.120	ITER	31	1.417
WOOD1P	0	$1.4429e+00$	3.950	ITER	31	18.289
WOODW	0	$1.3045e+00$	14.310	ITER	31	81.036

Es zeigt sich wieder die gute Übereinstimmung zwischen den Messungen und den Ergebnissen von Ordonez und Freund. Die LPs *capri, stocfor2* liefern keine obere Schranke, obwohl der primale Abstand zur Unzulässigkeit größer als 0 ist. Demgegenüber liefern *80bau3b, beaconfd, bnl2, cycle, e226, recipe, scrs8, stair, standgub* eine untere und *adlittle, gfrd-pnc, sc105, sc205, sc50a, sc50b* eine obere Schranke, obwohl der entsprechende Abstand zur Unzulässigkeit gleich 0 ist. Somit entsprechen 149 von 166 oder 89.76% der Schrankenberechnungen den Erwartungen.

Die Abbildungen 5.7, 5.8 und 5.9 zeigen die Zusammenhänge zwischen dem Abstand zur dualen Unzulässigkeit und der Anzahl benötigter Iterationen, dem Zeitverhältnis zwischen Lösungszeit und Schrankenberechnungszeit und der relativen Schärfe der unteren Schranke.

In diesen Abbildungen fehlen die Probleme *fit1d, fit2d, sierra*, die einen unendlichen Abstand zur dualen Unzulässigkeit haben. Bei den Zeitverhältnissen fehlen zusätzlich *afiro, sc50a*, bei denen keine Zeitverhältnisse zu ermitteln waren. Die Darstellung der Schrankengüte enthält schließlich die Probleme mit erfolgreicher Schrankenberechnung abzüglich *fit1d, fit2d, si-*

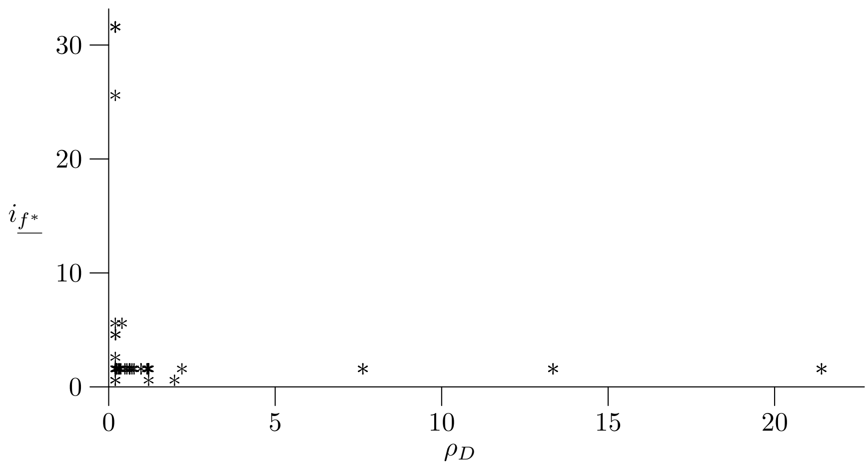


Abbildung 5.7: Zusammenhang zwischen dualem Abstand zur Unzulässigkeit und Anzahl benötigter Iterationen für die untere Schranke

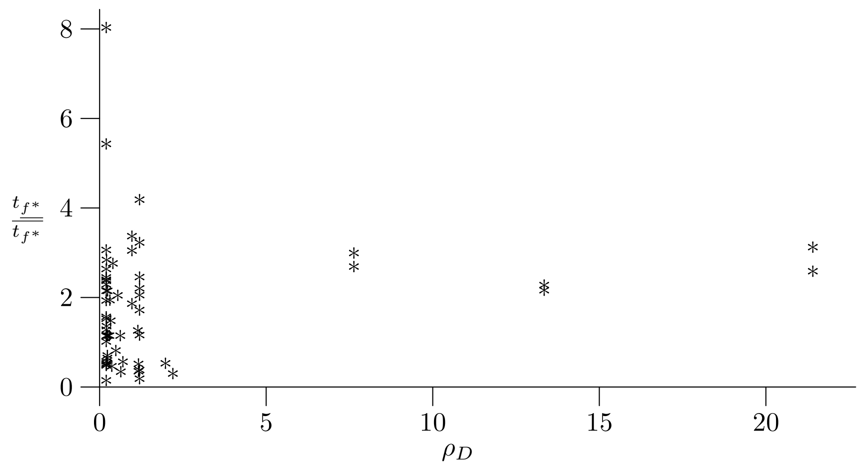


Abbildung 5.8: Zusammenhang zwischen dualem Abstand zur Unzulässigkeit und Verhältnis von benötigter Zeit zur unteren Schrankenberechnung zu benötigter Zeit zur Lösung des Originalproblems

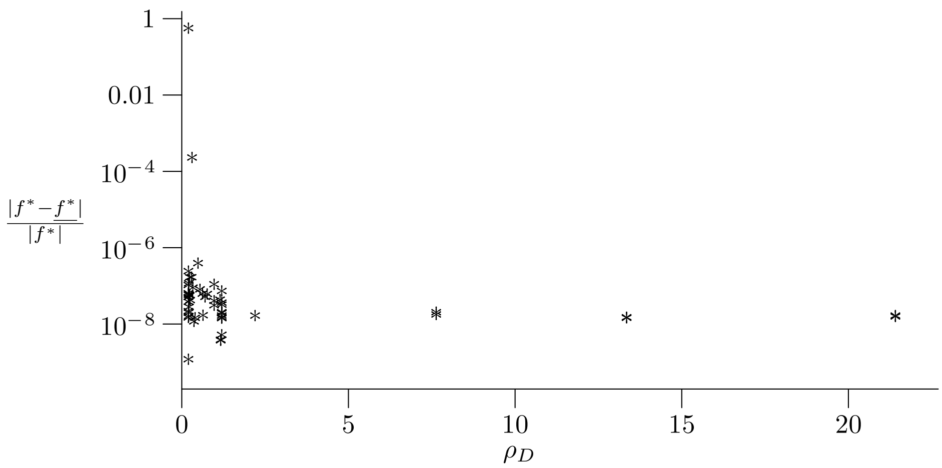


Abbildung 5.9: Zusammenhang zwischen dualem Abstand zur Unzulässigkeit und Güte der unteren Schranke

erra und *recipe, shell, standata, standgub*, da diese mit ihren extrem scharfen Schranken von $1 \cdot 10^{-16}$ und besser die Darstellung verzerrt hätten. Die generellen Beobachtungen bei den Messungen von lp_solve 3.2 zeigen sich auch hier: kein funktioneller Zusammenhang aber möglicher beschränkender Charakter. Auffällig ist, dass 47 Probleme mit einer Iteration für die untere Schranke auskommen. Der Block der Schrankengüten liegt diesmal unterhalb von $1 \cdot 10^{-6}$; dies dürfte unter anderem an der höheren Genauigkeit des Lösers liegen, die dafür sorgt, dass das verifizierte Intervall näher am globalen Optimum liegt.

Die Abbildungen 5.10, 5.11 und 5.12 zeigen die entsprechenden Zusammenhänge für die obere Schranke.

Hier fehlt in allen Abbildungen *fit2d* mit seinem großen primalen Abstand von 317 und *80bau3b*, da die Schrankenberechnung hier abbrach. Bei den Zeitverhältnissen fehlen zusätzlich *afiro, sc50a*, für die kein Zeitverhältnis zu ermitteln war. Die Schrankengüte ist schließlich für die 24 Probleme mit oberer Schranke, abzüglich *fit2d*, aufgetragen. Die Beobachtungen von lp_solve 3.2 passen auch hier: beschränkender Charakter des primalen Abstandes, größere Zeitverhältnisse als für die untere Schranke. Die Schrankengüten haben sich verbessert und liegen im Block diesmal unterhalb von $1 \cdot 10^{-6}$. Wie für die untere Schranke dürfte hier die höhere Genauigkeit mitverantwortlich sein.

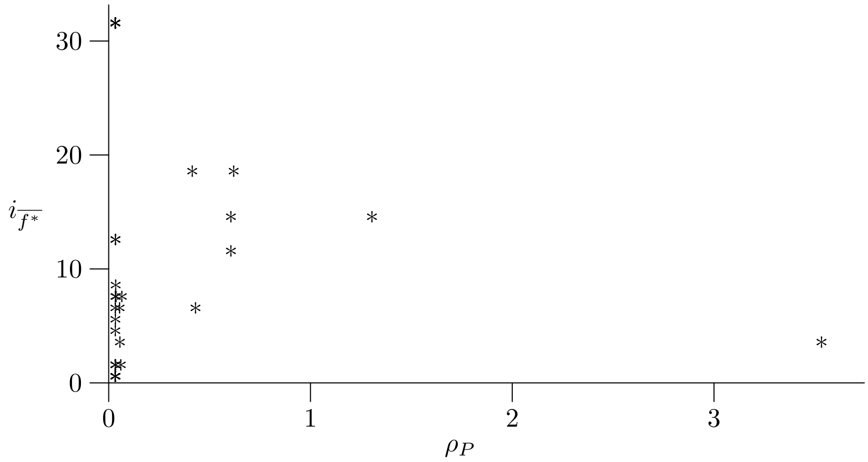


Abbildung 5.10: Zusammenhang zwischen primalem Abstand zur Unzulässigkeit und Anzahl benötigter Iterationen für die obere Schranke

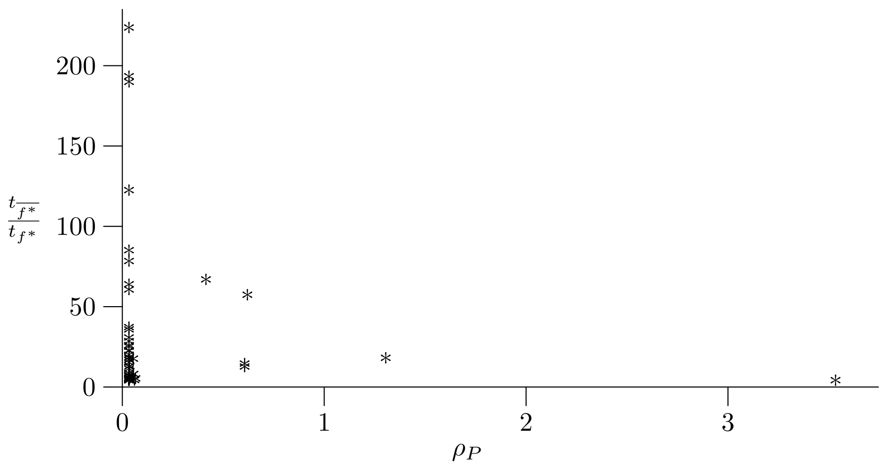


Abbildung 5.11: Zusammenhang zwischen primalem Abstand zur Unzulässigkeit und Verhältnis von benötigter Zeit zur oberen Schrankenberechnung zu benötigter Zeit zur Lösung des Originalproblems

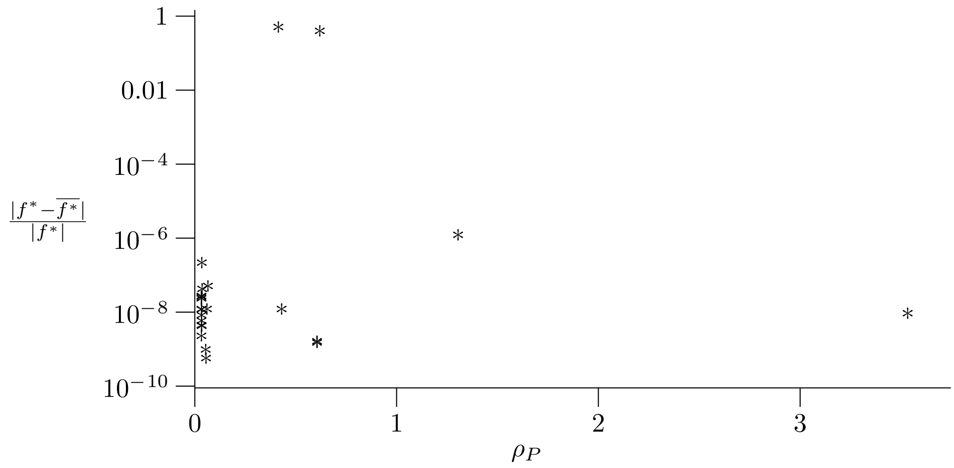


Abbildung 5.12: Zusammenhang zwischen primalem Abstand zur Unzulässigkeit und Güte der oberen Schranke

Kapitel 6

Ausblick

Mit der modularen Implementierung Lurupas wurde ein Paket geschaffen, welches mit wenig Aufwand an beliebige LP Löser angepasst werden kann. Dieser Ansatz hat bei Verwendung von lp_solve in den Versionen 3.2 und 4.0 funktioniert und sich bewährt. Hierbei zeigte sich, das der verwendete LP Löser zusammen mit seinen Parametern sehr wohl einen Einfluss auf die Schranken hat. Ist dieser Einfluss im vorliegenden Fall aufgrund der Ähnlichkeit der Algorithmen hauptsächlich auf die unterschiedliche Genauigkeit zurückzuführen, so erklärt dies alleine wohl noch nicht, warum einige LPs nur mit Version 3.2, andere mit Version 4.0 eine Schrankenberechnung erlauben. Interessant wäre in dieser Hinsicht, weitere Löser zu testen. Vor allem CPLEX wäre, idealerweise in Version 7.1, interessant, um die Probleme nach preprocessing mit ihren entsprechenden Abständen zur Unzulässigkeit zu vergleichen.

Es zeigen sich große Übereinstimmungen zwischen den Beobachtungen von Ordonez und Freund und den Messungen. 87.5% bzw. 89.76% der Messungen entsprechen den Abständen zur primalen und dualen Unzulässigkeit. Die vier erfolglosen Schrankenberechnungen bei primalen bzw. dualen Abständen größer als 0 scheiterten einmal an einem Abbruch durch lp_solve 3.2 und dreimal an der Iterationsschranke. So könnte es sein, dass eine höhere Schranke oder optimierte Algorithmusparameter hier noch eine rigorose Schranke liefern. Die 26 Schranken, die für Probleme mit einem Abstand von 0 zur Unzulässigkeit berechnet wurden, scheinen aufgrund der Theorie widersprüchlich. In den Tabellen von Ordonez und Freund taucht hier 0.000000 als Abstand zur Unzulässigkeit auf, so dass nicht klar ist, ob damit eine numerische oder eine mathematische 0 gemeint ist. Sollte sie numerisch sein, so wäre der Widerspruch aufgelöst. Ansonsten ergeben sich die Abstände zur Unzulässigkeit ihrerseits aus Optimierungsproblemen. Diese wurden nicht verifiziert gelöst, so dass das Ergebnis falsch sein kann. Gerade vor dem

Hintergrund ihrer Beobachtungen wäre auch ein Hinweis auf die Kondition dieser Optimierungsprobleme interessant gewesen. Sollten sie schlecht konditioniert sein, so entzögen sich die Beobachtungen quasi selbst die Grundlage.

Ich möchte mich bei Dr. Christian Jansson für seine Unterstützung bei der Anfertigung der vorliegenden Arbeit bedanken.

Hiermit versichere ich, dass ich die vorliegende Arbeit alleine und ausschließlich unter Verwendung der angegebenen Hilfsmittel angefertigt habe.

Literaturverzeichnis

[1] Michel Berkelaar, Peter Notebaert, and Kjell Eikland. lp_solve. World Wide Web. http://groups.yahoo.com/group/lp_solve.

[2] Basis linear algebra subprograms. World Wide Web. http://www.netlib.org/blas.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*, chapter 4.3 Linear Optimization. Cambridge University Press, 2004.

[4] I. N. Bronstein and K. A. Semendjajew. *Teubner Taschenbuch der Mathematik*, volume 1. Teubner, 1996.

[5] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991.

[6] H.J. Greenberg. *Mathematical Programming Glossary.* World Wide Web, http://www.cudenver.edu/ hgreenbe/glossary/, 1996–2004.

[7] John W. Gregory, Robert Fourer, and the Optimization Technology Center. Linear programming faq. World Wide Web. http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html.

[8] ILOG. Cplex. http://www.ilog.com/products/cplex.

[9] C. Jansson. Rigorous lower and upper bounds in linear programming. *to appear in SIAM Journal on Optimization*, 2004.

[10] Olaf Knüppel. Profil/bias v 2.0. Technical report, Technische Universität Hamburg-Harburg, February 1999.

[11] Donald Ervin Knuth. *The art of computer programming*, chapter 4 - Arithmetic. Addison–Wesley, 3rd edition, 1998.

[12] Netlib linear programming library. World Wide Web. http://www.netlib.org/lp.

[13] A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer programming. *Math. Programming A*. DOI:10.1007/s10107-003-0433-3.

[14] Institute of Electrical and Electronics Engineers. http://www.ieee.org.

[15] F. Ordonez and R. M. Freund. Computational experience and the explanatory value of condition numbers for linear optimization. Technical report, Cambridge, MA, 2002.

[16] John F. Reiser and Donald E. Knuth. Erratum: “Evading the Drift in Floating-Point Addition”. *Information Processing Letters*, 3(5):164–164, May 1975. See [17].

[17] John F. Reiser and Donald E. Knuth. Evading the drift in floating-point addition. *Information Processing Letters*, 3(3):84–87, January 1975. See erratum [16].

[18] Christian Rossi. Ieee 754. World Wide Web. http://cch.loria.fr/documentation/IEEE754.

[19] Dimitri van Heesch. Doxygen. World Wide Web. http://www.doxygen.org.

[20] Christopher Vickery. Ieee-754 references. World Wide Web. http://babbage.cs.qc.edu/courses/cs341/IEEE-754references.html.