universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

Heuristische Lösung gemischt-ganzzahliger Optimierungsprobleme

verfasst von

Bettina Hofbauer

angestrebter akademischer Grad

Magistra der Naturwissenschaften (Mag.rer.nat)

Wien, 2013

Studienkennzahl lt. Studienblatt:	A 405
Studienrichtung lt. Studienblatt:	Mathematik
Betreuer:	ao. Univ.-Prof. Dipl.-Ing. Dr. Hermann Schichl

Abstract

In dieser Arbeit beschäfitgen wir uns mit gemischt-ganzzahligen Optimierungsproblemen, die zur Klasse der NP-schweren Probleme gehören. Solche Probleme finden eine Vielzahl an Anwendungsmöglichkeiten in der Praxis, sind aber häufig nur sehr schwer zu lösen. Bei großen gemischt-ganzzahligen Optimierungsproblemen stellt sich oft bereits das Finden eines zulässigen Punktes als Herausforderung dar. Die Optimalität zu beweisen ist, mit den heutigen Algorithmen, teilweise gar nicht in vertretbarer Zeit möglich. Wir wollen uns daher auf heuristische Lösungsansätze konzentrieren, die nur zulässige Punkte erzeugen, jedoch lediglich beschränkte Aussagen über die Qualität der Punkte liefern. Dennoch werden wir auch kurz auf exakte Methoden eingehen, die Optimalität gewährleisten können. Unser Hauptaugenmerk werden wir auf die Feasibility Pump, einer Heuristik zur Findung zulässiger Punkte von gemischt-ganzzahligen Optimierungsproblemen, legen. Dieser von Fischetti et al. (2005) entwickelte Algorithmus, konnte sowohl mit Geschwindigkeit als auch der Qualität der gefundenen Punkte überzeugen.

Danksagung

An dieser Stelle möchte ich mich zuerst bei Herrn ao. Univ.-Prof. Dipl.-Ing. Dr. Hermann Schichl bedanken, der mich im Zuge dieser Arbeit, stets geduldig und mit seiner hervorragenden fachlichen Kompetenz, betreut hat. Des weiteren gilt mein Dank auch Herrn SPL ao. Univ.-Prof. Dr. Günther Hörmann. In den Anfangsphasen meines Studiums hatte er immer ein offenes Ohr für jegliche Fragen und hat stets versucht, seine Begeisterung für die Mathematik weiterzugeben. Auch meiner Familie möchte ich vielmals danken. Meinen Eltern, Hilde und Robert Hofbauer, die mich sowohl finanziell als auch mental immer unterstützt haben, meinen Brüdern Dipl. Ing. Michael Hofbauer, der mir mit seinem fachlichen Rat stets weitergeholfen hat, und Christoph Hofbauer. Darüber hinaus bedanke ich mich bei meinen Freunden für die mentale Unterstützung, im Speziellen bei meinem Freund Michael Grünner, der mir stets die Kraft gegeben hat, nach vorne zu sehen und mein Ziel nicht aus den Augen zu verlieren.

Inhaltsverzeichnis

1 Grundlagen

In vielen Bereichen der Wirtschaft, Logistik, Industrie, etc. ist die gemischt-ganzzahlige Optimierung in der heutigen Zeit nicht mehr wegzudenken. Im Folgenden möchte ich einen kurzen Überblick über dieses Thema sowie die Problematik und diverse Lösungsansätze präsentieren, wobei ich mich vorerst auf KALLRATH (2002) beziehe.

Oft ist es notwendig, bei Optimierungsproblemen für bestimmte Variablen Ganzzahligkeitsbedingungen einzuführen. Das ist einerseits dann der Fall, wenn im Zuge der Optimierung Entscheidungen gefällt werden müssen, wie z.B., ob ein bestimmtes Produkt hergestellt wird oder nicht, aber auch wenn es um Größen geht, bei denen ein nicht ganzzahliger Wert nicht sinnvoll wäre, wie die Anzahl von Maschinen oder Arbeitsschritten. Bei Entscheidungsproblemen arbeitet man mit sogenannten Binärvariablen, d.h. in unserem Beispiel bedeutet 1, dass das Produkt produziert wird und 0, dass es nicht produziert wird.

Notation Im Folgenden gelte für einen Vektor v, dass $v \geq 0$, genau dann wenn selbiges für jede Komponente von v gilt ($=$, $\leq$,... analog).

Bei der gemischt-ganzzahligen Optimierung hat man, wie der Name vermuten lässt, mit Optimierungsproblemen zu tun, die sowohl ganzzahlige als auch kontinuierliche Variablen beinhalten. Formal können wir solche Probleme folgendermaßen ausdrücken:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } h(x) = 0 \\ &\qquad g(x) \geq 0, \end{aligned} \tag{1.1}$$

wobei $f : X \times D \to \mathbb{R}$, $h : X \times D \to \mathbb{R}^k$, $g : X \times D \to \mathbb{Z}^l$, $x \in X \times D$ mit $X \subseteq \mathbb{R}^m$ und $D \subseteq \mathbb{Z}^n$. Dabei beschreibt f die Zielfunktion, h die Gleichungsnebenbedingungen und g die Ungleichungsnebenbedingungen. In unserem Fall besteht das Problem aus k Gleichungs- und l Ungleichungsnebenbedingungen. Weiters definieren wir die Indexmenge $I := \{i \mid x_i \in \mathbb{Z}\}$ und nennen x_i für $i \in I$ eine ganzzahlige Variable und x_j für $j \notin I$ eine kontinuierliche. Somit haben wir es im obigen Problem mit m kontinuierlichen und n ganzzahlige Variablen zu tun.

Erfüllt ein Vektor x alle Nebenbedingungen in (1.1), so nennen wir ihn zulässiger

Punkt. Weiters definieren wir die Menge aller zulässigen Punkte durch $S := \{x \in X \times D \mid g(x) \geq 0,\ h(x) = 0\}$ und bezeichnen sie als den zulässigen Bereich. Ziel der Optimierung ist es nun, eine optimale Lösung zu finden. Ein globales Minimum von (1.1) ist ein Vektor $\hat{x}$, der zulässig ist, und für den $f(\hat{x}) \leq f(x)$ für alle $x \in S$ gilt, falls ein solcher existiert. Gibt es ein $\epsilon > 0$, sodass $f(\hat{x}) \leq f(x)$ für alle $x \in S \cap B_\epsilon(\hat{x})$, so heißt $\hat{x}$ lokales Minimum. Bei einem Maximierungsproblem betrachtet man $-f$ und bemerkt, dass die Probleme $\max\{f(x) \mid x \in X \times D,\ g(x) \geq 0,\ h(x) = 0\}$ und $\min\{-f(x) \mid x \in X \times D,\ g(x) \geq 0,\ h(x) = 0\}$ äquivalent sind.

Wie wir im Weiteren sehen werden, ist es sinnvoll, Optimierungsprobleme der Form (1.1) in folgende Klassen zu unterteilen:

- Lineare Optimierung (LP): Solche bestehen nur aus kontinuierlichen Variablen, wobei alle Nebenbedingungen und die Zielfunktion linear sind.
- Nichtlineare Optimierung (NLP): In diesem Fall haben wir wieder nur kontinuierliche Variablen, aber mindestens eine Nebenbedingung oder die Zielfunktion ist bzw. sind nicht linear.
- Ganzzahlige Optimierung (IP): Dabei handelt es sich um Optimierungsprobleme, die nur ganzzahligen Variablen beinhalten.
- Gemischt-ganzzahlige Optimierung: Solche Optimierungsprobleme beinhalten sowohl ganzzahlige, als auch kontinuierliche Variablen.
  - Gemischt-ganzzahlige lineare Optimierung (MIP): alle Nebenbedingungen und die Zielfunktion sind linear.
  - Gemischt-ganzzahlige nichtlineare Optimierung (MINLP): mindestens eine Nebenbedingung oder die Zielfunktion ist bzw. sind nichtlinear.
- Globale Optimierung (GOP): Ein Optimierungsproblem gehört zu dieser Klasse, wenn nach einem globalen Optimum gesucht wird und nicht nur nach einem lokalen.

Nun gibt es für die obigen verschiedenen Klassen von Optimierungsproblemen auch verschiedene Verfahren zu deren Lösung, wobei diese teilweise ineinander übergehen. Hat man es z.B. mit einem MIP zu tun, benötigt man sowohl eine Methode zur Lösung von LPs, um die kontinuierlichen Variablen zu behandeln, als auch eine Methode zur Lösung von ganzzahligen Optimierungsproblemen.

1.1 Wichtige Resultate allgemeiner Optimierung

Wie in NEUMAIER & SCHICHL (2012) beschrieben wird, muss man gewisse Bedingungen an die Zielfunktion, die Nebenbedingungen und den zulässigen Bereich stellen, um Optimalität gewährleisten zu können. Meist wollen wir voraussetzen, dass es sich um stetig differenzierbare konvexe Funktionen handelt und der zulässige Bereich eine konvexe Menge bildet. Wobei man $S \subseteq \mathbb{R}^n$ konvex nennt, wenn für je zwei Punkte $x, y \in S$ auch die Menge $\overline{xy} := \{ty + (1-t)x \mid t \in [0,1]\}$ in S enthalten ist. Konvexität kann folgendermaßen für eine Funktion $f : S \to \mathbb{R}$ definiert werden:

- f wird konvex genannt, wenn $f(ty + (1-t)x) \leq tf(y) + (1-t)f(x)$ für alle $x, y \in S$ und $t \in [0,1]$ gilt.
- Ist f konvex und impliziert $f(ty+(1-t)x) = tf(y) + (1-t)f(x)$, dass $t \in \{0,1\}$ liegt, so heißt f strikt konvex.
- Man sagt f ist (strikt) konkav, wenn $-f$ (strikt) konvex ist.
- Gilt für alle $z \in \overline{xy}$ und $x, y \in S$: $f(z) \leq \max(f(x), f(y))$, so heißt f quasikonvex.
- f heißt quasikonkav, falls $-f$ quasikonvex ist
- Ist $f(z) < \max(f(x), f(y))$ für alle $z \in \overline{xy}$ und $x, y \in S$ erfüllt, so heißt f unimodal.

Obige Definitionen gelten auch für eine Funktion $F : \mathbb{R}^n \to \mathbb{R}^n$, falls jede Komponente die jeweiligen Bedingungen erfüllt.

Betrachten wir nun ein LP-Problem $\min\{f(x) = c^T x \mid Ax \geq b\}$ mit $x \in \mathbb{R}^n$ für eine Matrix $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$. In diesem Fall handelt es sich beim zulässigen Bereich $S = \{x \in \mathbb{R}^n \mid Ax \geq b\}$ um einen konvexen Polyeder. Ist S nun nichtleer und beschränkt, so kann man zeigen, dass eine globale Lösung des linearen Problems existiert, welche sich in einer Ecke dieses Polyeders befindet.

Für allgemeine kontinuierliche Optimierungsprobleme der Form

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } h(x) = 0, \end{aligned} \tag{1.2}$$

wobei $S \subseteq \mathbb{R}^n$, $f : S \to \mathbb{R}$, $h : S \to \mathbb{R}^m$, wollen wir nun die Lagrangeschen Multiplikatoren einführen, die durch folgendes Theorem gegeben sind. Dabei soll $g(x) = \nabla f(x)$

den Grandienten von f bezeichnen.

Theorem Sei S eine offene Teilmenge des $\mathbb{R}^n$ und f, h seien stetig differenzierbar. Darüber hinaus sei ein lokales Minimum $\hat{x}$ von obigem Optimierungsproblem (1.2) gegeben. Dann existieren $\kappa \geq 0$ und $\hat{y} \in \mathbb{R}^m$, sodass gilt:

$$\kappa g(\hat{x}) + h'(\hat{x})^T \hat{y} = 0,$$

wobei κ und $\hat{y}$ nicht beide gleich Null sind. Die Vektoren $\hat{y}$ wollen wir als Lagrangesche Multiplikatoren bezeichnen.

Gilt nun $\text{rk}(h'(\hat{x})) = m$, so ist $\kappa = 1$, und wir erhalten $g(\hat{x}) + h'(\hat{x})^T \hat{y} = 0$. Ist jedoch $\text{rk}(h'(\hat{x})) < m$, so kann $\kappa = 0$ sein, weil es $\hat{y} \neq 0$ gibt mit $h'(\hat{x})^T \hat{y} = 0$.

Mit Hilfe der Lagrangeschen Multiplikatoren wollen wir nun vorerst Optimierungsprobleme mit linearen Nebenbedingungen betrachten, also Probleme der Form:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } Ax \geq b, \end{aligned} \tag{1.3}$$

wobei $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$. Für Problem (1.3) erhalten wir somit folgende Optimalitätsbedingungen erster Ordnung:

Theorem Sei $f : S \to \mathbb{R}$ eine stetig differenzierbare Funktion, wobei $S = \{x \in \mathbb{R}^n \mid Ax \geq b\}$. Dann gilt:

1. Falls $\hat{x}$ eine lokale Lösung von (1.3) ist, so gibt es einen Vektor $\hat{y}$, sodass $g(\hat{x}) = A^T \hat{y}$ und $\inf(\hat{y}, A\hat{x} - b) = 0$ gelten.
2. Wenn f konvex ist, dann ist jede Lösung x, für die es ein $\hat{y}$ wie oben gibt, global.

Um zu hinreichenden Optimalitätsbedingungen für konvexe Probleme zu gelangen, beschäftigen wir uns nun mit dem sogenannten dualen Optimierungsproblem. Dafür betrachten wir vorerst folgendes Problem:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } h(x) \geq 0 \\ &\qquad x \in C, \end{aligned} \tag{1.4}$$

wobei C eine konvexe Menge und f, h konvexe und stetig differenzierbare Funktionen seien. In diesem Fall ist auch die zulässige Menge $S = \{x \in C \mid h(x) \geq 0\}$ konvex und damit jedes Minimum global. Bevor wir das zu (1.4) duale Problem konstruieren können, betrachten wir die Lagrange Funktion: $L(x, y) = f(x) + y^T h(x)$ und bemerken, dass folgendes Resultat gilt:

Proposition Falls für Problem (1.4) ein $x \in C$ und ein Vektor $y \in \mathbb{R}^m$ mit $y \geq 0$ existieren, sodass $g(x) + h'(x)^T y = 0$ gilt, so erhalten wir:

$$\min\{f(t) \mid t \in S\} \geq L(x, y).$$

Nun ist es an der Zeit, das zu (1.4) duale Problem folgendermaßen zu bilden:

$$\begin{aligned} &\max L(x, y) \\ &\text{s.t. } g(x) + h'(x)^T y = 0, \end{aligned} \tag{1.5}$$

für $x \in C$ und $y \geq 0$. Nach obiger Proposition gilt nun für eine Lösung $(\bar{x}, \bar{y})$ von (1.5) und eine Lösung $\hat{x}$ von (1.4), dass $L(\bar{x}, \bar{y}) \leq f(\hat{x})$, was uns zur Definition der Dualitätslücke führt. Diese ist gegeben durch

$$\Delta := f(\hat{x}) - L(\bar{x}, \bar{y}).$$

Mit Hilfe dieser Definitionen können wir nun die hinreichenden Optimalitätsbedingungen für konvexe Probleme formulieren:

Theorem Seien $f : C \to \mathbb{R}$ und $h : C \to \mathbb{R}^m$ konvexe und stetig differenzierbare Funktionen, wobei h' vollen Rang habe und $C \subseteq \mathbb{R}^n$ eine konvexe Menge sei. Existieren $\hat{x} \in C$ und $\hat{y} \in \mathbb{R}^m$, sodass die beiden Bedingungen $g(\hat{x}) + h'(\hat{x})^T \hat{y} = 0$ und $\inf(\hat{y}, -h(\hat{x})) = 0$ erfüllt sind, so ist $\hat{x}$ ein globales Minimum von (1.4) und $(\hat{x}, \hat{y})$ ein globales Maximum von (1.5). Die Dualitätslücke Δ ist in diesem Fall gleich Null.

1.2 Motivierende Beispiele

1.2.1 Das Rucksackproblem

Das Rucksackproblem, welches von Salkin & De Kluyver (1975) weitgehend untersucht wurde, stellt ein äußerst wichtiges Problem der gemischt-ganzzahligen Optimierung dar. Viele reale Probleme lassen sich auf das Rucksackproblem zurückführen bzw.

tritt es oft als Unterproblem von größeren auf. Der Name rührt von der gebräuchlichen Beschreibung des Problems. Auch wir wollen es anhand eines einfachen Beispiels mit Hilfe eines Rucksacks beschreiben. Wir betrachten also folgende Fragestellung: Es seien verschiedene Gegenstände, denen jeweils ein bestimmtes Gewicht und ein gewisser Wert zugeordet sind, gegeben. Ziel ist es nun, jene Gegenstände in einen Rucksack zu packen, die insgesamt den größten Wert ergeben, wobei der Rucksack einer Gewichtsbeschränkung unterliegt. Angenommen die Gewichtsbeschränkung liegt bei 70 kg und wir haben 4 Gegenstände mit folgenden Daten gegeben:

Gegenstand i	1	2	3	4
Wert W_i	15	100	40	90
Gewicht G_i	4	40	22	35

Für dieses konkrete Beispiel wollen wir annehmen, dass jeder Gegenstand nur einmal ausgewählt werden darf. Wir haben es somit mit folgendem binären Rucksackproblem zu tun:

$$\max f(x) = \sum_{i=1}^{4} W_i x_i$$
$$\text{s.t. } \sum_{i=1}^{4} G_i x_i \leq 70,$$

wobei x_i für $1 \leq i \leq 4$ nur die Werte 0 oder 1 annehmen kann. Nun bedeutet $x_i = 1$, dass der i-te Gegenstand gewählt wird und $x_i = 0$, dass er nicht gewählt wird. Bei diesem relativ überschaubaren Beispiel kann man die Lösung noch durch Ausprobieren aller Möglichkeiten per Hand lösen und erhält für $x = (1, 1, 1, 0)$ die optimale Lösung des Problems. In diesem Fall liegen wir mit 66 kg noch unter der Gewichtsbeschränkung, und der Wert der Zielfunktion $f(x)$ beträgt 155.

Im allgemeinen Rucksackproblem ist es generell erlaubt, die Gegenstände mehrmals auszuwählen. Womit wir zu folgender Formulierung gelangen:

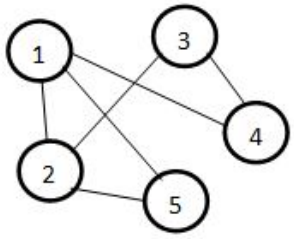
$$\max f(x) = \sum_{i=1}^{n} W_i x_i$$
$$\text{s.t. } \sum_{i=1}^{n} G_i x_i \leq c.$$

Dabei bezeichnet $x_i \in \{1, \ldots, n\}$ die gewählte Anzahl des Gegenstands i und c die Gewichtsgrenze.

1.2.2 Das Problem des Handlungsreisenden

Wie in Hoffman & Padberg (2000) beschrieben, handelt es sich beim Problem des Handlungsreisenden (engl: Traveling Salesman Problem, TSP) um ein kombinatorisches Optimierungsproblem, welches durch die vielseitigen praktischen Anwendungsmöglichkeiten weitgehend untersucht wurde bzw. wird. Beschrieben wird das Problem folgendermaßen: Ein Handlungsreisender soll n Städte genau einmal besuchen, bevor er wieder nach Hause zurückkehrt. Die Kosten der Reise von Stadt i nach j sollen dabei c_{ij} betragen. Ziel ist es nun, die Städte in der kostengünstigsten Reihenfolge zu besuchen.

Mathematisch gesehen kann man das Problem mit Hilfe eines Graphen darstellen. Diesen Begriff wollen wir kurz definieren. Nach Diestel (2006) handelt es sich bei einem Graphen G um ein Paar (V, E) von Mengen mit $E \subseteq \binom{V}{2}$, wobei $\binom{V}{2}$ die Menge aller zweielementigen Teilmengen von V bezeichnet. Dabei werden Elemente aus V als Knoten und Elemente aus E als Kanten des Graphen bezeichnet. Man nennt einen Knoten $v \in V$ und eine Kante $e \in E$ inzident, wenn $v \in e$ gilt. Zur Veranschaulichung wollen wir ein konkretes Beispiel bildlich darstellen, in welchem die Menge der Knoten durch $V = \{1, 2, 3, 4, 5\}$ gegeben ist und die Menge der Kanten durch $E = \{\{1, 2\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{2, 5\}, \{3, 4\}\}$.



Es handelt sich hierbei um einen ungerichteten Graphen. Das bedeutet, wenn beispielsweise Knoten 1 und 2 durch eine Kante miteinander verbunden sind, so kann man sowohl von 1 nach 2 als auch von 2 nach 1 gelangen. Ersetzt man in der Grafik die Linien durch Pfeile, die angeben sollen in welche Richtung man sich bewegen darf, so erhält man einen gerichteten Graphen.

Beim Problem des Handlungsreisenden bilden nun die Städte die Knoten des Graphen, die durch Kanten aus E verbunden sind. Jeder Kante sind dabei die Kosten c_{ij} zugeordnet. Meist arbeitet man mit vollständigen Graphen, also mit jenen, bei denen jeder Knoten mit jedem anderen Knoten durch eine Kante verbunden ist. Wird jede

Stadt genau einmal durchlaufen, so spricht man von einer Tour oder einem Hamiltonkreis. Man unterscheidet zwischen folgenden beiden Problemen:

- asymmetrisches TSP: Die Kosten, um von Stadt A zu Stadt B zu kommen, unterscheiden sich von jenen Kosten, die für die umgekehrte Strecke anfallen würden. So ist, z.B. der Preis für ein Flugticket von Wien nach Paris oft verschieden zu einem Ticket von Paris nach Wien. In diesem Fall muss, zur Berechnung der besten Tour, auf das asymmetrische TSP zurückgegriffen werden. Hierfür definieren wir die binären Variablen x_{ij}, wobei $x_{ij} = 1$ gilt falls die Kante $i \to j$ auf dem Hamiltonkreis liegt. Ist das nicht der Fall, so setzen wir $x_{ij} = 0$. Für $K \neq \emptyset$ mit $K \subseteq \{1, \ldots, n\}$ erhalten wir somit folgende Formulierung für das asymmetrische TSP:

$$\begin{aligned} \min & \sum_{j=1}^{n} \sum_{i=1}^{n} c_{ij} x_{ij} \\ \text{s.t.} & \sum_{j=1}^{n} x_{ij} = 1, \ i = 1, \ldots, n \\ & \sum_{i=1}^{n} x_{ij} = 1, \ j = 1, \ldots, n \\ & \sum_{j \in K} \sum_{i \in K} x_{ij} \leq |K| - 1, \end{aligned}$$

  wobei $x_{ij} \in \{0, 1\}$ für alle i, j. Die Kosten c_{ij} dürfen sich hier von den Kosten c_{ji} unterscheiden.

- symmetrisches TSP: Der Unterschied zum asymmetrischen TSP liegt darin, dass hier $c_{ij} = c_{ji}$ vorausgesetzt wird. Es ist also nicht von Belang, welche Richtung wir wählen. Sei nun E die Menge aller Kanten des ungerichteten Graphen. Wir definieren die binären Variablen x_j für alle $j \in E$ und c_j die Kosten, die bei dieser Kante entstehen. Die Menge aller ungerichteten Kanten bezüglich des Knotens j wollen wir mit $J(j)$ bezeichnen. Weiters soll $E(K)$ die Teilmenge aller ungerichteten Kanten, die die Knoten in einer Teilmenge K, die wie oben definiert sei, miteinander verbinden. Somit erhalten wir im symmetrischen Fall folgende Formulierung:

$$\begin{aligned}
\min \frac{1}{2} \sum_{j=1}^{n} \sum_{k \in J(j)} & c_k x_k \\
\text{s.t.} \sum_{k \in J(j)} & x_k = 2, \ \forall j = 1, \ldots, n \\
\sum_{j \in E(k)} & x_j \leq |K| - 1, \ j = 1, \ldots, n \\
\sum_{i \in K} \sum & j \in K x_{ij} \leq |K| - 1,
\end{aligned}$$

wobei $x_j \in \{0, 1\}$ für alle $j \in E$.

Es ist leicht zu sehen, dass es sich beim symmetrischen TSP um einen Spezialfall des asymmetrischen TSP handelt. Zur Berechnung dieses Spezialfalls existieren nun Methoden, die oft einfacher bzw. effizienter funktionieren, weshalb diese Unterscheidung auch getroffen wurde.

1.3 Problematik ganzzahliger Probleme

Wie Li & Sun (2006) in ihrem Buch beschreiben, ist es im Vergleich zu kontinuierlichen Optimierungsproblemen weit schwieriger, ganzzahlige Probleme zu untersuchen. Das intensive Studium der reelen Zahlen, welche zur Entwicklung nützlicher analytischer Werkzeuge führten, vereinfachen die Arbeit mit kontinuierlichen Variablen erheblich. Einige wichtige Aussagen für konvexe Probleme haben wir bereits kennengelernt. Für ganzzahlige Probleme gelten diese jedoch fast nie, da wir es in diesem Fall im Allgemeinen nicht mit konvexen Mengen zu tun haben.

Um auf die Problematik, die mit der Ganzzahligkeit einhergeht, näher einzugehen, wollen wir nun Entscheidungsprobleme betrachten, also all jene Probleme die mit "Ja" oder "Nein" beantwortet werden können. Dabei sei beachtet, dass jedes Optimierungsproblem in ein Entscheidungsproblem umgewandelt werden kann. Es ist üblich, Entscheidungsprobleme aufgrund der Laufzeit ihrer Lösungsalgorithmen in verschiedene Komplexitätsklassen einzuteilen (siehe auch Walukiewicz (1991)):

- P: In diese Klasse fallen alle Probleme, die in polynomialer Zeit gelöst werden können, also mit einem Aufwand von $O(n^k)$. Dabei ist n die Eingabelänge des Problems, die der Problemgröße entspricht, und $k \geq 1$ eine konstante natürliche Zahl. Man kann zeigen, dass lineare Optimierungsprobleme zu dieser Klasse gehören.

- NP: Die Menge der NP-Probleme beinhaltet all jene Entscheidungsprobleme, zu denen ein nichtdeterministischer Algorithmus existiert, der in maximal $O(n^k)$ Rechenschritten eine bereits existierende Lösung überprüfen kann. Dabei ist nicht gesagt, dass eine Lösung in polynomialer Zeit gefunden werden kann. Dafür ist der Aufwand meist erheblich höher.

- NP-schwer: Ein Entscheidungsprobleme L fällt in diese Klasse, falls alle anderen Probleme, die in NP liegen, mit einem Aufwand von maximal $O(n^k)$, auf L zurückgeführt werden können.

- NP-vollständig: Diese Klasse beinhaltet all jene Probleme, die in NP liegen und zusätzlich zu den NP-schweren Problemen gehören. Es handelt sich dabei um die schwersten Probleme, die in NP liegen. Karp (1972) gelang es zu zeigen, dass unter anderen das oben kennengelernte Rucksackproblem zu dieser Klasse von Problemen gehört.

Es ist leicht ersichtlich, dass $P \subseteq NP$ gilt. Bis heute konnte noch nicht bewiesen werden, ob sogar Gleichheit gilt. Sollte man einen Lösungsalgorithmus finden, der in polynomialer Zeit ein NP-vollständiges Problem löst, dann könnte man mit vergleichbarem Aufwand alle anderen NP-Probleme lösen und hätte somit $P = NP$ bewiesen. Die meisten ganzzahligen Optimierungsprobleme fallen in die Klasse der NP-schweren Probleme, deren Lösung sich oft sehr schwierig darstellt. Dennoch gibt es gerade bei der gemischt-ganzzahligen linearen Optimierung bereits einige relativ effiziente Lösungsmethoden, die zumindest gute zulässige Punkte liefern, wenn auch nicht immer die optimalen. Im nichtlinearen Fall arbeitet man bis heute meist mit heuristischen Methoden, die nur zulässige Punkte erzeugen, jedoch keinerlei Aussage über deren Optimalität liefern.

2 MIP

In diesem Abschnitt betrachten wir Probleme der Form:

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } Ax = b \\ &\quad x \geq 0, \end{aligned} \tag{2.1}$$

bzw.

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } Ax \geq b, \end{aligned} \tag{2.2}$$

wobei $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ und $x \in \mathbb{R}^n$ mit $x_j \in \mathbb{Z}$ für alle $j \in I = \{i \mid x_i \in \mathbb{Z}\}$.

Hat man es mit kontinuierlichen Problem zu tun, sind diese beiden Schreibweisen äquivalent, da alle linearen Ungleichungsnebenbedingungen mit Hilfe von Schlupfvariablen in Gleichungsnebenbedingungen umgewandelt werden können. Bei ganzzahligen Problemen werden, durch die Einführung der Schlupfvariablen, einige der ganzzahligen Variablen zu kontinuierlichen, sofern A und b nicht gänzlich aus ganzzahligen Werten bestehen. Berücksichtigt man diese Tatsache kann man dennoch auch für diese Probleme beide Schreibweisen verwenden. Zur Erklärung verschiedener Theorien ist oft eine der Formulierungen nützlicher als die andere, weshalb auch wir beide Schreibweisen verwenden werden.

Es gibt nun 2 verschiedene Ansätze, solche Optimierungsprobleme zu lösen:

1. Exakte Methoden, wie:
   - Entscheidungsbaumverfahren
   - Schnittebenenverfahren
2. Heuristische Methoden, wie:
   - Simulated Annealing
   - Tabusuche

Beide Lösungsansätze haben ihre Vor- und Nachteile. Heuristische Methoden erzeugen - wenn überhaupt - nur zulässige Punkte, es kann jedoch nicht festgestellt werden, ob die Lösung optimal ist. Dieses Problem tritt bei den exakten Methoden nicht auf,

jedoch ist hierbei der Aufwand oft zu hoch, um sie effizient auf große Probleme anwenden zu können. Im Folgenden wollen wir einige Methoden zur Lösung von MIPs näher betrachten.

2.1 Exakte Methoden zur Lösung von MIPs

Wir wollen die exakten Methoden zur Lösung von MIPs wieder in verschiedene Klassen unterteilen:

- Vollständige Enumeration
- Begrenzte oder implizite Enumeration:
  - Branch & Bound
  - Schnittebenenverfahren
  - Branch & Price

Hierbei ist die vollständige Enumeration nur begrenzt nützlich, weil bei dieser, wenn man es mit rein ganzzahligen Problemen zu tun hat, einfach alle Möglichkeiten durchprobiert werden und durch Vergleich die optimale Lösung bestimmt wird. Im gemischtganzzahligen Fall müssen zusätzlich die kontinuierlichen Variablen betrachtet werden. Das Ausprobieren aller Möglichkeiten ist dann überhaupt nicht möglich, da es durch die kontinuierlichen Variablen auch unendlich viele Möglichkeiten gibt. Deshalb könnte man vorerst die kontinuierlichen Variablen betrachten, diese dann fixieren und mit der vollständigen Enumeration der ganzzahligen Variablen fortfahren. Mit dieser Methode erreicht man die Lösung eines Optimierungsproblems nur mit exponentiellem Aufwand. Es ist unschwer zu erkennen, dass dieser Aufwand wirklich nur für sehr kleine Probleme vertretbar bleibt. Daher wollen wir diesen Ansatz nicht weiter betrachten und uns an dieser Stelle dem Branch & Bound-Verfahren zuwenden, welches zur impliziten Enumeration gehört.

2.1.1 Branch & Bound-Verfahren

Das Branch & Bound-Verfahren wurde erstmals von Land & Doig (1960) vorgestellt und gilt als eines der gebräuchlichsten Verfahren um IPs bzw. MIPs zu lösen. Zur Beschreibung dieser Methode betrachten wir ein Optimierungsproblem der Form (2.1). Dabei wird vorerst von den Ganzzahligkeitsbedingungen abgesehen und das so

erhaltene lineare Optimierungsproblem, die sogenannte LP-Relaxation, gebildet:

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } Ax = b \qquad (P^1) \\ &\quad x \geq 0, \end{aligned}$$

wobei $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ und $x \in \mathbb{R}^n$.

Meist wird das erhaltene Problem mit Hilfe des Simplex-Verfahrens, einer Methode zur Lösung linearer Optimierungsprobleme, gelöst. Man erhält somit die optimale Lösung der LP-Relaxation. Bei einem Minimierungsproblem ist das jener Vektor $\hat{x}^1 \in \mathbb{R}^n$, für den die Zielfunktion den kleinsten Wert annimmt. Diesen Wert wollen wir mit $\hat{f}^1$ bezeichnen. Sind nun schon alle Ganzzahligkeitsbedingungen erfüllt, haben wir die optimale Lösung des ursprünglichen Problems bereits gefunden und sind somit fertig.

Es ist aber sehr wahrscheinlich, dass die erhaltenen Variablen nicht ganzzahlig sind, also außerhalb des zulässigen Bereichs liegen. Wir wissen jedoch, dass das MIP kein besseres Ergebnis liefern kann als $\hat{f}^1$ und können somit $\underline{f} := \hat{f}^1$ als untere Schranke festlegen. Außerdem setzen wir als obere Schranke $\overline{f} := \infty$. Sollte nun die j-te Koordinate x_j von x eine ganze Zahl sein, der erhaltene Wert bei der LP-Relaxation $\bar{x}_j$ jedoch nicht, wird die LP-Relaxation in zwei Unterprobleme unterteilt. Dies wollen wir tun, indem wir folgende Restriktionen hinzufügen: $x_j \leq \lfloor \bar{x}_j \rfloor$ bzw. $x_j \geq \lfloor \bar{x}_j \rfloor + 1$, wobei $\lfloor \bar{x}_j \rfloor \in \mathbb{Z}$ und $\lfloor \bar{x}_j \rfloor < \bar{x}_j$ gelte. Wir betrachten also nun folgende Unterprobleme:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } Ax = b \qquad (P^2) \\ &\quad x \geq 0 \\ &\quad x_j \leq \lfloor \bar{x}_j \rfloor, \end{aligned}$$

und

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } Ax = b \qquad (P^3) \\ &\quad x \geq 0 \\ &\quad x_j \geq \lfloor \bar{x}_j \rfloor + 1, \end{aligned}$$

Dieser Verzweigungsprozess liefert uns die ersten Knoten unseres Entscheidungsbaumes. Die beiden Unterprobleme werden nun durch ein duales Simplexverfahren gelöst. Tritt nun einer der 3 folgenden Fälle ein, wird der Entscheidungsbaum am Knoten P^k nicht weitergeführt:

1. Der Punkt $\hat{x}^k$ von P^k erfüllt bereits alle geforderten Ganzzahligkeitsbedingungen des ursprünglichen Problems.
2. Es gibt keine zulässige Punkte für P^k.
3. P^k hat einen Lösung $\hat{x}^k$, der jedoch nicht die Ganzzahligkeitsbedingungen des MIP erfüllt und $\hat{f}^k$, der Zielfunktionswert von P^k, ist größer als der Wert der Zielfunktion eines womöglich bereits gefundenen zulässigen Punktes des MIP.

Im ersten Fall wird der Punkt $\hat{x}^k$ gespeichert und mit anderen zulässigen Punkten des MIP, die entweder bereits gefunden wurden oder im weiteren Verlauf des Verfahrens auftreten, verglichen. Immer wenn wir auf einen zulässigen Punkt des MIP stoßen, setzen wir den zugehörigen Zielfunktionswert als obere Schranke fest und bezeichnen diesen mit $\overline{f}$. Ist am Ende $\hat{f}^k \geq \overline{f}$, dann haben wir mit $\hat{x}^k$ die optimale Lösung des MIP gefunden.

Tritt für einen Knoten keiner der obigen Fälle ein, muss dieser wieder, wie oben beschrieben, in zwei Unterprobleme aufgeteilt werden. Ein Knoten der unsere Schranke $\underline{f}$ unterschreitet, kann dabei verworfen werden. Ansonsten fährt man wie oben erklärt so lange fort, bis eine optimale Lösung des MIP gefunden wurde bzw. festgestellt wurde, dass dafür kein zulässiger Punkt existiert.

Im ungünstigsten Fall benötigt auch dieser Algorithmus einen exponentiellen Zeitaufwand, um zu einer Lösung des Optimierungsproblems zu gelangen. Meist kann jedoch, durch das Verwerfen von diversen Knoten, ein großer Teil der Rechenschritte eingespart werden.

2.1.2 Schnittebenenverfahren

Das Schnittebenenverfahren, erstmals vorgestellt von Gomory (1958), funktioniert ähnlich wie das Branch & Bound-Verfahren. Auch hier wird zuerst die LP-Relaxation des MIP betrachtet und dann das Problem durch zusätzliche Restriktionen eingeschränkt. Letzteres geschieht nun jedoch durch Hinzufügen von linearen Ungleichungen. Wir betrachten wieder die Lösung der LP-Relaxation. Erfüllt diese Lösung nicht die

Ganzzahligkeitsbedingungen wird daraufhin eine lineare Ungleichung aufgestellt, die von allen zulässigen Punkten des MIP erfüllt wird, nicht jedoch von der Lösung der LP-Relaxation. Das führt dazu, dass ein Teil des zulässigen Bereichs der LP-Relaxation abgeschnitten wird, was oft eine etwas bessere Annäherung an die optimale Lösung ermöglicht. Vor allem im späteren Verlauf des Algorithmus, gestaltet sich die Suche nach solchen linearen Ungleichungen jedoch oft relativ schwierig, und es kann meist nicht bewiesen werden, ob überhaupt noch eine Ungleichung mit obigen Eigenschaften existiert. Daher wird heutzutage oft eine Kombination aus Branch & Bound und Schnittebenenverfahren, das sogenannte Branch & Cut-Verfahren verwendet, welches z.B. von Padberg & Rinaldi (1987) erfolgreich auf das Problem des Handlungsreisenden angewendet wurde.

2.1.3 Branch & Price

Ein weiteres Verfahren zur Lösung von MIPs ist das sogenannte Branch & Price-Verfahren, welches von Barnhart et al. (1998) beschreiben wird. Diese Methode erweist sich besonders bei Problemen mit einer großen Anzahl an Variablen als nützlich. Das Branch & Price Verfahren funktioniert ähnlich wie die Branch & Cut Methode. Auch hier wird auf das Branch & Bound-Prinzip zurückgegriffen. Während jedoch beim Branch & Cut-Verfahren durch das Hinzufügen von zusätzlichen Nebenbedingungen mit der Erzeugung neuer Zeilen gearbeitet wird, beschäftigt sich die Branch & Price-Methode mit der Generierung von zusätzlichen Spalten, die in den Knoten des Branch & Bound-Baumes zur Anwendung kommen, also mit einer Pricing-Strategie. Zur Erklärung des Algorithmus der Spaltenerzeugung (engl.: column generation) betrachten wir folgendes Optimierungsproblem:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } Ax \leq b, \end{aligned} \tag{2.3}$$

wobei $x \in B$ eine ganze Zahl sein soll. Die Zielfunktion $f(x)$ muss dabei nicht zwingend linear sein.

Wir betrachten nun $B^* = \{x \in B : x \in \mathbb{Z}^n\}$ und bemerken, dass es sich hierbei um eine endliche Menge handelt, was den Grundgedanken von Spaltenerzeugung darstellt. Ist nämlich B beschränkt so besteht B^* aus endlich vielen Punkten, die wir mit $y_1, \ldots, y_p$ bezeichnen wollen. Speziell für binäre Variablen x enthält B^* gerade die Extremwerte der konvexen Hülle $\text{conv}(B^*)$. Wie in Nemhauser & Wolsey (1988) nachgelesen werden kann, ist $\text{conv}(B^*)$ selbst für unbeschränktes B ein Polyeder, welches durch

endlich viele Elemente erzeugt wird. Somit kann auch in diesem Fall Spaltenerzeugung angewendet werden. Dennoch wollen wir uns im Folgenden nur noch mit beschränkten Mengen B beschäftigen.

Sei nun $B^* = \{y_1, \ldots, y_p\}$. Dann kann, bezüglich der Konvexitätsbedingung:

$$\sum_{k=1}^{p} \lambda_k = 1,$$

wobei $\lambda_k \in \{0, 1\}$ für $k = 1, \ldots, p$, jeder Punkt $y \in B^*$ folgendermaßen geschrieben werden:

$$y = \sum_{k=1}^{p} y_k \lambda_k.$$

Wir erhalten somit die Spaltenerzeugungsform des ursprünglichen Problems (2.3):

$$\begin{aligned} \min & \sum_{k=1}^{p} -c_k \lambda_k, \\ \text{s.t.} & \sum_{k=1}^{p} a_k \lambda_k \leq b \\ & \sum_{k=1}^{p} \lambda_k = 1, \end{aligned} \tag{2.4}$$

wobei $-c_k = f(y_k)$, $a_k = Ay_k$ und $\lambda_k \in \{0, 1\}$ für $k = 1, \ldots, p$.

Obwohl die Zielfunktion in Problem (2.3), wie bereits erwähnt, nicht unbedingt linear sein muss, haben wir es nun wegen $y = y_k$ (da $y = \sum y_k \lambda_k$, $\sum \lambda_k = 1$ und λ_k binär sind) sehr wohl mit einem linearen Problem zu tun. Dies ist jedoch nur für beschränktes B möglich. Im unbeschränkten Fall müsste man andere Wege finden, um zur Linearität zu gelangen bzw. eine lineare Zielfunktion voraussetzen, wobei wir uns nicht weiter damit beschäftigen werden.

Wir nehmen nun an, es existiere eine Zerlegung von B, also $B = \bigcup_{1 \leq j \leq n} B_j$. Dann gilt $B_j^* = \{y_1^j, \ldots, y_{p_j}^j\}$, und wir erhalten die Spaltenerzeugungsform bezüglich (2.3)

mit individuellen Konvexitätsbedingungen für jedes B_j:

$$\begin{aligned} \min & \sum_{j=1}^{n} \sum_{k=1}^{p_j} -c_k^j \lambda_k^j, \\ \text{s.t.} & \sum_{j=1}^{n} \sum_{k=1}^{p_j} a_k^j \lambda_k^j \leq b \\ & \sum_{k=1}^{p_j} \lambda_k^j = 1, \end{aligned} \tag{2.5}$$

wobei $f(y_k^j) = -c_k^j$, $Ay_k^j = a_k^j$ und $\lambda_k^j \in \{0,1\}$ mit $j = 1, \ldots, n$ und $k = 1, \ldots, p_j$. Dieses Problem wollen wir als das Master Problem (MP) bezeichnen, welches wiederum für ganzzahliges $\lambda_k \geq 0$ für $k = 1, \ldots, p$ auf folgende vereinfachte Form gebracht werden kann:

$$\begin{aligned} \min & \sum_{k=1}^{p} -c_k \lambda_k, \\ \text{s.t.} & \sum_{k=1}^{p} a_k \lambda_k \leq b \\ & \sum_{k=1}^{p} \lambda_k = n, \end{aligned} \tag{2.6}$$

wenn die Teilmengen B_j mit $\bar{B} = \{y_1, \ldots, y_n\}$ für $j = 1, \ldots, n$ übereinstimmen. Die neuen Konvexitätsbedingungen $\sum_{k=1}^{p} \lambda_k = n$ erhält man durch Setzen von $\lambda_k = \sum_j \lambda_k^j$.

Noch ist nicht klar, warum dieses Verfahren besonders für sehr große Probleme nützlich ist. Der springende Punkt ist, dass man vorerst nur einen Teil des Master Problems betrachtet, welches weitaus weniger Variablen enthält. Das so erhaltene Problem wollen wir als eingeschränktes Master Problem (engl.: restricted Master Problem, RMP) bezeichnen. Nun wird die LP-Relaxation des RMP gelöst, und die erhaltene optimale duale Lösung mit d bezeichnet. Diejenigen Variablen, die für die Gewinnung zulässiger Punkte von Bedeutung sind, jedoch nicht im RMP enthalten sind, werden dann durch Lösung der folgenden Pricing Probleme gewonnen:

$$\begin{aligned} \min & -dx \\ \text{s.t.} & \ x \in B^*, \end{aligned} \tag{2.7}$$

bzw.

$$\begin{aligned} &\min -dx \\ &\text{s.t. } x \in \text{conv}(B^*). \end{aligned} \tag{2.8}$$

Nach Lösung des RMPs wird überprüft, ob man bereits eine ganzzahlige Lösung gefunden hat. Ist das nicht der Fall, wird der Branch & Bound Algorithmus angewendet und somit neue Knoten des Entscheidungsbaumes generiert. In den so entstehenden Knoten wird nun wieder der Spaltenerzeugungs-Algorithmus angewendet. Auch beim Branch & Price-Verfahren wird mit oberen und unteren Schranken für den Zielfunktionswert gearbeitet, um Teile des Entscheidungsbaumes ausschließen zu können und damit effektiver zu einer Lösung zu gelangen. Als untere Schranke dient das Ergebnis der Lösung des Spaltenerzeugungs-Algorithmus, sollte diese nicht ganzzahlig sein. Liefert der Algorithmus jedoch eine ganzzahlige Lösung, wird der entsprechende Zielfunktionswert als obere Schranke festgelegt.

2.2 Heuristische Methoden zur Lösung von MIPs

Für viele Probleme ist die Anwendung eines exakten Lösungsverfahren nicht sinnvoll, da der Rechenaufwand zu groß wäre. Deshalb wollen wir uns nun heuristischen Lösungsmethoden zuwenden, die zwar die Optimalität nicht garantieren können, aber dennoch gute Lösungen mit weitaus weniger Aufwand liefern. Vorab bemerken wir jedoch, dass auch jede exakte Methode als heuristisches Verfahren verwendet werden kann, indem man die Suche vorzeitig abbricht, sobald man eine bzw. eine hinreichend gute Lösung gefunden hat.

2.2.1 Simulated Annealing

Simulated Annealing, auch Simulierte Abkühlung genannt, ist ein verbreitetes Verfahren zur heuristischen Lösung von gemischt-ganzzahligen Optimierungsproblemen. Der Name rührt daher, dass es physikalisch gesehen einem Abkühlungsprozess ähnelt. Wie in Aarts et al. (1997) beschrieben wird, baut Simulated Annealing auf dem Konzept von Threshold Algorithmen auf, die mit lokaler Suche arbeiten und durch stochastische Faktoren gesteuert werden. Abbildung 1 zeigt eine Möglichkeit, Threshold Algorithmen zu implementieren.

Wir betrachten nun ein Optimierungsproblem der Form (2.2) und die Menge S der zulässigen Punkte, wobei wir $f : S \to \mathbb{R}$ die Kostenfunktion nennen. Weiters definieren wir eine Funktion $N : S \to P(S)$, welche eine Menge $N(x) \subseteq S$ von Nachbarpunkten

```
procedure THRESHOLD_ALGORITHM;
begin
    INITIALIZE (i_start);
    i := i_start;
    k := 0;
    repeat
        GENERATE (j from N (i));
        if f (j) − f (i) < t_k then i := j;
        k := k + 1;
    until STOP;
end;
```

Abbildung 1: Implementation von Grenzalgorithmen (Aarts et al. (1997))

für jedes $x \in S$ liefert. Ausgehend von einem Startpunkt $x \in S$ werden nun ein Nachbarpunkt y aus der Umgebung $N(x)$ gewählt und die Kosten $f(x)$ und $f(y)$ miteinander verglichen. Mit Hilfe bestimmter Schwellen, die durch die Folge $(t_k)_{k\geq 0}$ gegeben sind, wird entschieden, ob der Algorithmus mit dem momentanen Punkt x oder dem Nachbarpunkt y weiterarbeitet. Behält man x bei, wird ein lokaler Nachbarpunkt gesucht, der von y verschieden ist, und man wählt wieder mit Hilfe der Veränderung der Kosten einen der beiden Punkte aus. Wird jedoch stattdessen y akzeptiert, führt man die lokale Suche in $N(y)$ fort und geht im Weiteren analog vor.

Setzt man nun $t_k = 0$ für alle $k \in \{0, 1, 2, \ldots\}$, so wird y nur dann akzeptiert, wenn $f(y) \leq f(x)$ gilt. In diesem Fall spricht man von iterativer Verbesserung. Hat man den Startwert glücklich gewählt, also in der Nähe eines globalen Optimums, wird man jenes mit dieser Methode auch schnell erreichen. Leider ist das jedoch oft sehr unwahrscheinlich, und der Algorithmus wird meist bei einem mehr oder weniger guten lokalen Minimum zum Stillstand geraten.

Um dieses Problem zu umgehen, verwendet Simulated Annealing einen wahrscheinlichkeitstheoretischen Ansatz. Es werden zufällige Schwellen $t_k = a \in [0, \infty[$ gewählt, die für alle $k \in \{0, 1, 2, \ldots\}$ einen Erwartungswert von $E(t_k) = c_k$ haben und einer Verteilungsfunktion F_{c_k} folgen. Dabei bezeichnet $c_k \in \mathbb{R}^+$ eine monoton fallende Folge mit $\lim_{k\to\infty} c_k = 0$. Es gilt dann $P_{c_k}(t_k \leq z) = F_{c_k}(z)$ für die Wahrscheinlichkeit, dass eine Schwelle höchstens z beträgt. Sei nun $P_{c_k}(y)$ die Wahrscheinlichkeit, y zu akzeptieren und damit den momentanen Punkt x zu verwerfen. In der Originalversion, die erstmals von Kirkpatrick et al. (1983) beschrieben wurde, wird folgendes Akzeptanzkriterium

verwendet:

$$P_{c_k}(y) = \begin{cases} 1 & f(y) \leq f(x) \\ \exp(\frac{f(x)-f(y)}{c_k}), & f(y) > f(x) \end{cases} .$$

Auch andere Wahlen von F_{c_k} sind möglich, jedoch ist darauf zu achten, dass Punkte mit großem Kostenanstieg mit geringerer Wahrscheinlichkeit gewählt werden, als Punkte mit kleinem Kostenanstieg. Dadurch, dass nicht jedes y verworfen wird, das größere Kosten als x liefert, ist es durch Verwendung von Simulated Annealing auch möglich, sich von einem lokalen Minimum wieder zu entfernen, um möglicherweise zu einem besseren zu gelangen. Diese Tatsache stellt den großen Vorteil gegenüber der iterativen Verbesserung dar.

2.2.2 Tabu Suche

Eine weitere Möglichkeit zur heuristischen Lösung von MIPs der Form (2.2) liefert die Tabu Suche, die in Glover (1989) vorgestellt wird. Wie beim Simulated Annealing wird auch hier mit einem Startpunkt x begonnen und daraufhin die Umgebung $N(x) \subseteq S$ auf weitere zulässige Punkte durchsucht, wobei S wieder die Menge der zulässigen Punkte bezeichnet. Die Elemente $n \in N(x)$ wollen wir als die möglichen Züge bezeichnen. Es handelt sich also um diejenigen Nachbarpunkte, die für das gegebene Optimierungsproblem zulässig sind, und daher im weiteren Verlauf des Algorithmus den Startpunkt x ersetzen dürfen.

Weiters definieren wir die Menge $T \subseteq N$ der Tabu-Züge (engl. tabu moves), die wir als Tabu-Liste bezeichnen wollen. Diese Menge wird im Lauf des Algorithmus erstellt und enthält diejenigen Züge die nicht gewählt werden dürfen. Sie dient grundsätzlich dazu, eine eventuelle Zyklusbildung zu vermeiden, wobei meist eine maximale Anzahl an Iterationen t festgelegt wird, nach der die Menge T wieder geleert wird und man von neuem beginnt sie wieder zu befüllen. Eine genauere Beschreibung, wie diese Menge aufgebaut wird, folgt etwas später. Zuerst möchten wir das Grundgerüst der Tabu Suche, das durch folgende 4 Schritte gebildet wird, betrachten:

Schritt 1 Wir setzen $x^* = x$ für einen Startpunkt $x \in S$, die Anzahl der Iterationen $k = 0$ und die Menge der Tabu-Züge $T = \emptyset$.

Schritt 2 Sollte $N(x) \backslash T = \emptyset$ gelten, fahren wir mit Schritt 4 fort. Ansonsten setzen wir $k = k + 1$ und wählen ein $n_k \in N(x) \backslash T$, wobei $n_k(x)$ den optimalen Wert

aller $f(n)$ für $n \in N(x)\backslash T$ bezeichnen soll. Auf diese Weise wird also nicht jeder Nachbarpunkt zugelassen sondern nur der bestmögliche.

Schritt 3 Nun setzen wir $x = n_k(x)$ und prüfen, ob $f(x) < f(x^*)$ für die Zielfunktionswerte des Optimierungsproblems gilt. Ist das der Fall, so setzen wir $x^* = x$. Es ist leicht zu sehen, dass x^* denjenigen bereits gefundenen zulässigen Punkt bezeichnet, der momentan am besten ist.

Schritt 4 Der Algorithmus endet hier, falls entweder $N(x)\backslash T = \emptyset$ gilt oder eine gewisse Anzahl an Iterationen t überschritten wird. Wie man die maximale Anzahl der Iterationen wählen soll, werden wir später genauer betrachten. Tritt keiner dieser Fälle ein, wird die Menge T aktualisiert und ab Schritt 2 fortgefahren.

Sowohl die richtige Wahl der maximalen Iterationen t, als auch der Aufbau der Tabu-Liste T sind essenziell für die Effizienz der Tabu Suche. In vielen Anwendungen wird T in jedem Schritt durch den berechneten optimalen Wert $n_k(x)$ erweitert, um zu verhindern, einen bereits betrachteten Punkt erneut zu besuchen. Ein anderer Ansatz, der denselben Effekt liefert, besteht darin, T mit den inversen Zügen zu befüllen. In diesem Fall setzen wir also $T = \{n^{-1} : n = n_h, h > k - t\}$ für den momentanen Iterationsschritt k. Es gilt dann $n^{-1}(n(x)) = x$, was wiederum die Rückkehr zu einem bereits besuchten Punkt verhindert. In der Praxis wird die Tabu-Liste T jedoch meist folgendermaßen gewählt: $T = \bigcup C_h : h > k - t$. Dabei bezeichnet C_h eine Sammlung von Zügen, die n_h^{-1} enthält aber auch andere Züge, die gewisse Tabu-Attribute erfüllen.

Je größer man nun die maximale Anzahl an Interationen t wählt, umso größer ist die Wahrscheinlichkeit, eine Zyklusbildung zu vermeiden, da die Liste der Tabu-Züge dementsprechend länger wird. Dennoch ist eine zu große Wahl von t oft nicht sinnvoll, da zu viele Restriktionen dieser Art die Qualität der zulässigen Punkte erheblich verschlechtern können. Die große Herausforderung ist es also, t nicht zu groß aber auch nicht zu klein zu wählen. Da dazu jedoch noch keine allgemein gültigen Regeln bestehen, wird t für das gerade zu lösende Problem meist adaptiv angepasst.

2.2.3 Feasibility Pump (FP)

Grundlagen

In der Arbeit Fischetti et al. (2005) wird ein Algorithmus zur heuristischen Lösung von MIPs, die sogenannte Feasibility Pump (FP), beschrieben. Diese soll auf eine Vielzahl von Testproblemen anwendbar sein, aber auch in Geschwindigkeit und Qualität

mit bereits existierenden Algorithmen mithalten können.

Wir betrachten das MIP der Form (2.2) und dessen LP-Relaxation:

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } Ax \geq b, \end{aligned} \tag{2.9}$$

wobei $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ und $x \in \mathbb{R}^n$.

Wir definieren $P := \{x \in \mathbb{R}^n \mid Ax \geq b\}$, den zulässigen Bereich der LP-Relaxation, $S := \{x \in D \subseteq \mathbb{R}^n \mid Ax \geq b,\ x_j \in \mathbb{Z},\ \forall j \in I\}$, den zulässigen Bereich des MIP, und die Menge $T := \{x \in D \subseteq \mathbb{R}^n \mid x_j \in \mathbb{Z},\ \forall j \in I\}$, die wir die Menge aller ganzzahligen Punkte nennen wollen. Dabei bezeichnet $I = \{i \mid x_i \in \mathbb{Z}\}$ die Indexmenge der Variablen, die der Ganzzahligkeitsbedingung unterliegen. Es ist leicht zu sehen, dass $S \subset P$ und $S \subset T$ gelten. Nun wollen wir mit $\tilde{x}$ den gerundeten Wert von x bezeichnen, wobei wir $\tilde{x}_j := [x_j]$ für alle $j \in I$ und $\tilde{x}_j := x_j$ für alle $j \notin I$ setzen. Dabei ist $[x_j]$ jener Wert der durch Auf- bzw. Abrunden zur nächsten ganzen Zahl entsteht. Weiters definieren wir die L_1-Norm, die den Abstand zwischen einem Punkt $x \in P$ und einem Punkt $\tilde{x} \in T$ angibt, wie folgt:

$$\Delta(x, \tilde{x}) = \sum_{j \in I} |x_j - \tilde{x}_j|.$$

Nehmen wir nun an, dass die Variablen des MIP für alle $j \in I$ durch $l_j \leq x_j \leq u_j$ beschränkt sind, können wir zuvor definierten Abstand auch folgendermaßen schreiben:

$$\Delta(x, \tilde{x}) = \sum_{j \in I : \tilde{x}_j = l_j} (x_j - l_j) + \sum_{j \in I : \tilde{x}_j = u_j} (u_j - x_j) + \sum_{j \in I : l_j < \tilde{x}_j < u_j} ({x_j}^+ + {x_j}^-),$$

wobei x^+ und x^- den positiven bzw. negativen Teil des Vektors x angeben. Zusätzlich müssen folgende Nebenbedingungen eingeführt werden: $x_j = \tilde{x}_j + {x_j}^+ + {x_j}^-$ mit ${x_j}^+ \geq 0$ und ${x_j}^- \geq 0$ für alle $j \in I$ mit $l_j < \tilde{x}_j < u_j$.

Unser Ziel ist es nun, den Abstand $\Delta(x, \tilde{x})$ so weit wie möglich zu reduzieren und so zu einem zulässigen Punkt des MIP zu gelangen. Wir betrachten also vorerst folgendes Optimierungsproblem:

$$\begin{aligned} &\min \Delta(x, \tilde{x}) \\ &\text{s.t. } Ax \geq b. \end{aligned} \tag{2.10}$$

Nun beginnen wir bei einem Punkt $x_1^* \in P$, den wir durch Lösung der LP-Relaxation erhalten, und konstruieren, wie zuvor erklärt, den gerundeten Wert $\tilde{x}_1 \in T$. Weiters fixieren wir den ganzzahligen Teil von $\tilde{x}_1$ und lösen das Problem (2.10) mit Hilfe einer LP-Methode. So erhalten wir $x_2^* \in P$. Ist nun $\Delta(x_2^*, \tilde{x}_1) = 0$, dann ist x_2^* bereits ein zulässiger Punkt des MIP und wir sind fertig. Ist das nicht der Fall, setzen wir $\tilde{x}_2$ als den gerundeten Wert von $x_2^* \in P$, fixieren ihn wieder und lösen damit erneut Problem (2.10). Wir fahren so lange fort bis $\Delta(x_k^*, \tilde{x}_k) = 0$ für ein $k \in \mathbb{N}$ erfüllt ist.

Durch die FP-Iterationen werden also zwei Folgen von Punkten x^* und $\tilde{x}$ erzeugt, die bestenfalls gegen eine MIP-Lösung konvergieren. Dabei kann man $\Delta(x_k^*, \tilde{x}_k)$ als Abstand von $\tilde{x}_k$ zum Polyeder P deuten. Nun erfüllt $\tilde{x}_k$ die Ganzzahligkeitsbedingungen des MIP, aber nicht unbedingt die Nebenbedingungen der LP-Relaxation, während x_k^* die Nebenbedingungen der LP-Relaxation erfüllt, aber nicht zwingend die Ganzzahligkeitsbedingungen. Bei jedem FP-Iterationsschritt, auch Pumping Cycle genannt, wird nun versucht den Abstand $\Delta(x_k^*, \tilde{x}_k)$ weiter zu reduzieren bis schlussendlich $\Delta(x_k^*, \tilde{x}_k) = 0$ erfüllt ist. Haben wir das erreicht, ist es uns gelungen, die Ganzzahligkeit von $\tilde{x}$ sozusagen in x^* zu pumpen. Diese Sichtweise ist auch verantwortlich für den Namen des Verfahrens.

Implementation für 0-1 MIPs

Im Weiteren wollen wir uns mit der Implementation des FP-Verfahrens beschäftigen, wobei wir uns dabei auf MIPs, deren ganzzahlige Variablen binär sind, die sogenannten 0-1 MIPs, beschränken. Wir betrachten also folgenden Spezialfall von (2.2):

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } Ax \geq b, \end{aligned} \tag{2.11}$$

wobei $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ und $x \in \mathbb{R}^n$ mit $x_j \in \{0, 1\}$ für alle $j \in I$ und erhalten damit folgende vereinfachte Form des Abstandes zwischen den Punkten x und $\tilde{x}$:

$$\Delta(x, \tilde{x}) = \sum_{j \in I : \tilde{x}_j = 0} x_j + \sum_{j \in I : \tilde{x}_j = 1} (1 - x_j),$$

da wir nun $0 \leq x_j \leq 1$ für alle $j \in I$ als Variablenschranken erhalten.

Abbildung 1 zeigt eine Möglichkeit, wie man den FP-Algorithmus implementieren könnte. Im Folgenden wollen wir erläutern, was in den einzelnen Schritten der Implementation genau passiert:

```
The Feasibility Pump (basic version):
  1. initialize nIT := 0 and x* := argmin{c^T x : Ax ≥ b};
  2. if x* is integer, return(x*);
  3. let x̃ := [x*] (= rounding of x*);
  4. while (time < TL) do
  5.    let nIT := nIT + 1 and compute x* := argmin{Δ(x, x̃) : Ax ≥ b};
  6.    if x* is integer, return(x*);
  7.    if ∃ j ∈ I : [x*_j] ≠ x̃_j then
  8.       x̃ := [x*]
        else
  9.       flip the TT=rand(T/2,3T/2) entries x̃_j (j ∈ I) with highest |x*_j − x̃_j|
 10.    endif
 11. enddo
```

Abbildung 2: Basic FP implementation for 0-1 MIPs (Fischetti et al. (2005))

Zeile 1 Hier bezeichnet `nIT` die Anzahl der Iterationen, die zu Beginn auf 0 gesetzt wird. Außerdem wird der Punkt $x^* \in P$ durch Lösung der LP-Relaxation gewonnen.

Zeile 2 Sollte x^* bereits ganzzahlig sein, endet der Algorithmus bereits hier und liefert uns die optimale Lösung des MIPs. Ist das nicht der Fall, wird mit Zeile 3 fortgefahren.

Zeile 3 In diesem Schritt wird der gerundete Wert $\tilde{x}$ von x^*, wie bereits beschrieben, berechnet.

Zeile 4 Hier beginnt unsere Schleife, die den Pumping Cycle darstellt. Dabei bezeichnet `TL` die maximale Berechnungszeit, die wir vorab festlegen müssen.

Zeile 5 Nun wird x^* durch Lösen des Problems (2.10) aktualisiert.

Zeile 6 Wieder wird überprüft, ob x^* ganzzahlig ist. Ist das der Fall, haben wir nach 6 Schritten einen zulässigen Punkt des MIPs gefunden und der Algorithmus stoppt an dieser Stelle. Ist x^* jedoch nicht ganzzahlig, wird mit Zeile 7 fortgefahren.

Zeile 7 Hier wird überprüft, ob $[x^*] = \tilde{x}$ gilt. Ist mindestens eine Komponente verschieden, wird mit Zeile 8 fortgefahren. Gilt jedoch $[x_j^*] = \tilde{x}_j$ für alle $j \in I$, gehen wir zu Zeile 9 über.

Zeile 8 Wir setzen $\tilde{x} := [x^*]$.

Zeile 9 Wenn alle Einträge von $[x^*]$ und $\tilde{x}$ übereinstimmen, würde die Weiterführung des Algorithmus zu nichts führen, da sich die Werte der beiden Punkte in den einzelnen Iterationsschritten nicht mehr verändern. Somit würde man, wenn man davon absieht, dass wir eine maximale Berechnungszeit festgelegt haben, in einer Endlosschleife landen. Deshalb wollen wir in diesem Schritt einige Koordinaten von $\tilde{x}$ verändern. Dafür wählen wir eine zufällige Anzahl $\texttt{TT} \in \{\frac{1}{2}T, \ldots, \frac{3}{2}T\}$ der Einträge von $\tilde{x}$ aus, für die $|x_j^* - \tilde{x_j}|$ für $j \in I$ die größten Werte annehmen, und ändern deren Wert von 1 auf 0 bzw. von 0 auf 1. `T` gibt dabei die durchschnittliche Anzahl der Variablen an, die wir vertauschen wollen, wobei diese Anzahl im vorhinein festgelegt werden muss.

Zeile 11 Wir sind am Ende der While-Schleife angelangt und springen wieder zu Schritt 4 zurück, wo wir den nächsten Pumping Cycle mit dem aktualisierten $\tilde{x}$ durchlaufen.

Der Algorithmus endet, sobald ein zulässiger Punkt x^* für das MIP gefunden bzw. die maximale Berechnungszeit überschritten wurde.

Trotz der Einbindung des Schrittes, in dem Komponenten der Variablen vertauscht werden, kommt es bei dieser Implementation des FP-Verfahrens häufig zu Zyklusbildung, d.h., es werden immer wieder die gleichen Folgen von Punkten betrachtet. Um dieses Problem zu umgehen, wurde der Algorithmus ein wenig aktualisiert. Es sollen nun die Punkte der letzten Iterationen verglichen werden, um eine Zyklusbildung festzustellen. Sobald eine solche vermutet wird, wird Schritt 7-10 einfach übersprungen und stattdessen ein zufälliger Wert $\rho \in [-0.3, 0.7]$ für alle $j \in I$ generiert. Sollte nun $|x_j^* - \tilde{x}_j| + \max\{\rho j, 0\} > 0.5$ gelten, wird $\tilde{x}_j$, wie gewohnt, vertauscht.

Um die Qualität der FP zu ermitteln wurde sie auf insgesamt 83 0-1 MIP Testprobleme (44 0-1 MIPs aus ACHTERBERG et al. und 39 0-1 schwere MIPs) angewendet und mit der geläufigen Software ILOG-Cplex 8.1 (IBM), die mit einer Branch & Bound Methode arbeitet, verglichen. Dabei wurden beide Algorithmen nach Finden des ersten zulässigen Punktes abgebrochen. Folgende Tabelle zeigt die erhaltenen Ergebnisse des Vergleichs der beiden Algorithmen, wobei wir untersuchen, ob ein zulässiger Punkt gefunden wurde (*gef.*) oder nicht (*nicht gef.*), wie oft die jeweilige Methode schneller einen zulässigen Punkt erreichte (*schneller*) und wie oft der gefundene Punkt besser war als derjenige des konkurrierenden Algorithmus (*besser*):

Name	gef.	nicht gef.	schneller	besser
FP	80	3	31	46
ILOG-Cplex 8.1	64	19	26	33

Wie man sieht, stellt die FP also eine äußerst effiziente Methode zur heuristischen Lösung von MIPs dar. Der Algorithmus gibt jedoch keinerlei Aussage über die tatsächliche Qualität des erhaltenen Punktes, da das Verfahren bereits nach dem ersten gefundenen zulässigen Punkt abbricht.

Um die Geschwindigkeit der FP-Methode noch etwas zu verbessern, wurden andere Varianten der FP vorgeschlagen und auf einen Teil der Testprobleme angewendet:

1. FP1: Hier wird die LP-Relaxation in Schritt 1 durch ein Primal-Dual-Verfahren nur näherungsweise gelöst, wobei die erhaltenen Werte verwendet werden, um eine untere Schranke für den optimalen Wert zu erzeugen. Bis Schritt 5 wird nun fortgefahren wie bisher. Daraufhin wird mit einem Primal-Simplex-Verfahren wieder nur eine Annäherung an die Lösung das Problems $\min\{\Delta(x, \tilde{x}) : Ax \geq b\}$ gesucht. Bei dieser Methode kann es jedoch passieren, dass zwar ein Punkt gefunden wird, aber nicht sichergestellt werden kann, dass dieser auch zulässig ist. In diesem Fall wird Schritt 6 übersprungen und ansonsten analog fortgefahren.

2. FP2: In dieser FP-Variante wird im 1. Schritt überhaupt kein Verfahren zur Lösung von LP-Problemen angewendet. Man erhält $\tilde{x}$ lediglich durch Runden eines beliebigen $x^* \in [0, 1]^n$. Ansonsten wird analog zur FP1-Variante fortgefahren.

Wie bereits erwähnt, kann die FP keinerlei Aussage über die Qualität der erhaltenen Lösung machen, wenn bereits nach dem ersten zulässigen Punkt des MIPs abgebrochen wird. Dasselbe Problem haben wir bei den Varianten FP1 und FP2, wobei hier die Qualität der gefundenen Punkte meist noch wesentlich schlechter ausfällt als beim gewöhnlichen FP-Verfahren, da nicht mit optimalen Lösungen der LP-Relaxation gearbeitet wird, sondern nur mit Approximationen. Um die Qualität der Punkte zu verbessern, scheint es einleuchtend, nicht nur eine, sondern mehrere zulässige Punkte zu generieren, wobei das Problem nach jedem gefundenen zulässigen Punkt durch die Einführung einer oberen Schranke, die wir mit $\overline{f}$ bezeichnen wollen, eingeschränkt wird. Wir modifizieren also die FP-Methode folgendermaßen:

- Nach Schritt 1 setzen wir $\overline{f} = +\infty$ und $\underline{f}^1 = c^T x^*$ als obere bzw. untere Schranke des Zielfunktionswertes des MIP fest.

- Bei Schritt 4 setzen wir `nIT-nIT0` $<$ `IL` als zusätzliche Bedingung fest. Dabei bezeichnet `nIT0` die Anzahl der FP-Iterationen, bis der erste zulässige MIP-Punkt gefunden wurde und `IL` die festgelegte Anzahl an Iterationen, die ab diesem Zeitpunkt maximal ausgeführt werden sollen.

- Wird in Schritt 5 ein $x^* \in S$ gefunden, setzen wir $\underline{f}^2 = c^T x^*$ und aktualisieren die obere Schranke folgendermaßen: $\overline{f} = \alpha \underline{f}^1 + (1-\alpha)\underline{f}^2$ für ein $\alpha \in (0,1)$.

Der restliche Verlauf des Algorithmus bleibt unverändert. Analog können wir diese Abänderungen des Algorithmus auch in die Variante FP1 einführen. Bei FP2 verwendet man als obere Schranke $\underline{f}^2 - \beta|\underline{f}^2|$ für $\underline{f}^2 \neq 0$, wobei keine untere Schranke $\underline{f}^1$ definiert wird.

Mit dieser Verfeinerung der Verfahren wurden FP1 und FP2 wieder auf einen Teil der Testprobleme angewendet. Tatsächlich konnten beide Verfahren gerade bei Problemen, bei denen die FP relativ lange Berechnungszeit benötigte, deutlich schneller ein Ergebnis liefern. Um jedoch auf einen qualitativ gleichwertigen Punkt zu kommen, ist es notwendig, die Algorithmen nicht bereits beim ersten gefundenen zulässigen Punkt abzubrechen, sondern wie oben beschrieben weiter laufen zu lassen. In den Tests stellte sich heraus, dass sich FP1 noch ein wenig effektiver als FP2 darstellt. Während FP1 für jedes Problem einen zulässigen Punkt innerhalb der maximalen Berechnungszeit fand, gelang das FP2 nur in 23 von 26 Fällen. Auch die Qualität der Punkte ist meist deutlich besser, wenn man mit der FP1-Variante arbeitet. Allgemein kann man sagen, dass die FP qualitativ am besten abgeschnitten hat. Hat man es jedoch mit Problemen zu tun, die einer langen Berechnungszeit bedürfen, ist es oft besser mit FP1 zu arbeiten. Man gelangt damit schnell zu einem ersten zulässigen Punkt, dessen Qualität man gegebenenfalls durch oben beschriebenes Verfahren noch verbessern kann.

Implementation für allgemeine MIPs

Um die von Fischetti et al. (2005) entwickelten Feasibility Pump auf allgemeine MIPs der Form (2.2) anwenden zu können, haben Bertacco et al. (2007) den Algorithmus etwas abgeändert. Abbildung 2 zeigt eine mögliche Implementation des modifizierten FP. Dabei muss, im Gegensatz zur Implementation für 0-1 MIPs, folgende Abstandsfunktion verwendet werden:

$$\Delta(x, \tilde{x}) = \sum_{j \in I : \tilde{x}_j = l_j} (x_j - l_j) + \sum_{j \in I : \tilde{x}_j = u_j} (u_j - x_j) + \sum_{j \in I : l_j < \tilde{x}_j < u_j} ({x_j}^+ + {x_j}^-),$$

Im Grunde werden dieselben Schritte durchlaufen wie bei der FP-Implementation

für 0-1 MIPs. Da wir es jedoch nicht mehr mit binären Variablen zu tun haben, ändert sich die Vorgehensweise der Vertauschung der Variablen ein wenig. Wie im FP für 0-1 MIPs wird geprüft, ob der gerundete Wert des aktualisierten x^* gleich $\tilde{x}$ ist. Ist das der Fall, wird $\delta_j = |x_j^* - \tilde{x_j}|$ für alle $j \in I$ berechnet. In Schritt 12 wird wieder eine zufällige Anzahl TT der Variablen, die die größten Werte δ_j annehmen, abgeändert. Der Wert kann aber nicht einfach wie bei binären Variablen von 0 auf 1 und umgekehrt vertauscht werden. Vielmehr ändern wir $\tilde{x}_j$ zu $\tilde{x}_j + 1$ falls $x_j^* > \tilde{x}_j$ bzw. $\tilde{x}_j - 1$ falls $x_j^* < \tilde{x}_j$.

```
The Feasibility Pump for general MIPs (basic scheme):
 1. initialize x* := argmin{c^T x : Ax ≥ b}
 2. x̃ := [x*] (:= rounding of x*)
 3. nIter := 0
 4. while (Δ(x*, x̃) > 0 and nIter < maxIter) do
 5.   nIter := nIter+1
 6.   x* := argmin{Δ(x, x̃) : Ãx ≥ b̃}
 7.   if Δ(x*, x̃) > 0 then
 8.     if [x*_j] ≠ x̃_j for at least one j ∈ I then
 9.       update x̃ := [x*]
10.     else
11.       for each j ∈ I define the score σ_j := |x*_j − x̃_j|
12.       move the TT=rand(T/2, 3T/2) components x̃_j with largest σ_j
13.       if cycling is detected, perform a restart operation
14.     endif
15.   endif
16. enddo
```

Abbildung 3: Basic FP implementation for general MIPs (Bertacco et al. (2007))

Schritt 13 dient der Ermittlung einer eventuellen Zyklusbildung. Ist nun das aktuelle $\tilde{x}$ bereits zu einem früheren Zeitpunkt ermittelt worden oder beträgt die Reduktion des Abstandes $\Delta(x^*, \tilde{x})$ über eine festgelegte Anzahl KK an Iterationen weniger als 10%, dann wird durch zufällige Störung einiger der Eintragungen von $\tilde{x}$ ein Neustart des Programms herbeigeführt.

Binäre und allgemeine ganzzahlige Variablen

Wir haben nun einerseits eine Variante des FP zur Lösung von 0-1 MIPs, andererseits eine Version zur Lösung von allgemeinen MIPs vorgestellt. Oft ist es jedoch der Fall, dass wir es mit MIPs zu tun haben, die sowohl binäre als auch allgemeine ganzzahlige Variablen enthalten. Wir erhalten somit den folgenden Spezialfall von (2.2):

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } Ax \geq b, \end{aligned} \tag{2.12}$$

wobei $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ und $x \in \mathbb{R}^n$ mit $x_j \in \mathbb{Z}$ für alle $j \in G$ und $x_k \in \{0, 1\}$ für alle $k \in B$. Dabei bilden die Mengen G und B eine Partition der Menge I im bisherigen Sinn, d.h. $G \cup B = I$ und $G \cap B = \emptyset$.

Um Probleme dieser Form zu lösen, haben BERTACCO et al. (2007) folgende 3 Stufen eingeführt:

- Binäre Stufe: In dieser Stufe betrachtet man vorerst nur die binären Variablen. Dabei können wir also mit der vereinfachten Form des Abstandes:

$$\Delta(x, \tilde{x}) = \sum_{j \in I : \tilde{x}_j = 0} x_j + \sum_{j \in I : \tilde{x}_j = 1} (1 - x_j),$$

  arbeiten und das FP-Verfahren für 0-1 MIPs anwenden. Tritt eine der drei folgenden Möglichkeiten ein, wird mit Stufe 2 fortgefahren:

  - Es wurde ein Punkt x^*, der alle binären Bedingungen erfüllt (nicht jedoch unbedingt die restlichen Ganzzahligkeitsbedingungen), gefunden.
  - Der Abstand $\Delta(x^*, \tilde{x})$ blieb während der letzten `KK` Iterationen unverändert.
  - Die maximale Berechnungszeit wurde erreicht.

  Im Unterschied zur Variante für allgemeine MIPs wird der Neustart des Programs in diesem Schritt nur durchgeführt, wenn das aktuelle x^* tatsächlich gleich einem in vorhergegangen Iterationen erhaltenem Wert ist. Genauer wird in diesem Fall eine zufällige Anzahl an Variablen, die sich in den letzten Iterationen nicht verändert haben, mit Wahrscheinlichkeit $|x_j^* - [x_j^*]| + \rho$, wobei in der Implementation $\rho = 0.03$ gewählt wurde, vertauscht.

- Allgemeine ganzzahlige Stufe: Hier wird der Punkt, der in Stufe 1 den kleinsten Abstand lieferte, als Anfangswert übernommen. Nun werden auch die allgemeinen ganzzahligen Variablen betrachtet und mit der oben beschriebenen Methode zur Lösung allgemeiner MIPs fortgefahren. Wird bis Ablauf der maximalen Berechnungszeit kein zulässiger MIP-Punkt gefunden, gehen wir zur nächsten Stufe über.

- Enumerations-Stufe: Die Idee dieser Stufe ist, dass man mit einer universellen Enumerations-Methode zum Finden zulässiger Punkte für das MIP fortfährt, sollte das FP-Verfahren nicht in vertretbarer Zeit zu einem erfolgreichem Ergebnis gelangen. Hierfür bezeichnen wir mit x^B jenen Punkt x^*, der in Abbildung 3 in Schritt 6 berechnet wurde, dessen Abstand zu $\tilde{x}$ am geringsten ist. Wir setzen

$\tilde{x} := [x^B]$ und nennen ihn den fast zulässigen Punkt des MIP. Nun verwenden wir, wie erwähnt, eine universelle Methode zur Lösung von MIPs und wenden diese auf das Originalproblem an, wobei wir die Zielfunktion $c^T x$ durch die Abstandsfunktion $\Delta(x - \tilde{x})$ ersetzen.

Bertacco et al. (2007) haben den oben beschriebenen Algorithmus auf diverse Testprobleme angewendet und, vorerst nur die Geschwindigkeit der FP, mit den kommerziellen Software-Paketen Xpress Optimizer (Corporation) und ILOG Cplex verglichen. Um ein vergleichbares Ergebnis zu erlangen, wurden dabei jeweils dieselben Algorithmen zur Lösung der LP-Relaxation und des Problems mit der Abstandsfunktion verwendet. Das heißt im Vergleich mit Xpress Optimizer wurde eine Xpress-basierte FP-Variante implementiert bzw. mit ILOG Cplex eine ILOG Cplex-basierte FP-Variante. Diese Varianten der FP wollen wir mit FP-xpress bzw. FP-cplex nennen. Folgende Tabelle zeigt nun die Ergebnisse:

Name	**gef.**	**nicht gef.**	**schneller**
Xpress Optimizer	28	9	4
FP-xpress	32	5	29
ILOG Cplex	35	2	17
FP-cplex	36	1	18

Es stellte sich also heraus, dass sowohl Xpress Optimizer als auch ILOG Cplex im Schnitt mehr Zeit benötigen als die FP, um zu einem zulässigen Punkt zu gelangen. Außerdem wurde mit der FP für mehr Probleme überhaupt einen zulässigen Punkt gefunden, als mit den beiden anderen Algorithmen. Die Qualität der Lösung lässt jedoch oft zu Wünschen übrig. Deshalb wurden noch 3 weitere Varianten der FP vorgeschlagen:

1. Analog zu jenem Verfahren zur Verbesserung der Qualität der Lösung von 0-1 MIPs, wird hier mit den jeweiligen oberen und unteren Schranken gearbeitet.

2. In dieser Variante wird das FP-Verfahren nach Finden des ersten zulässigen Punktes des MIPs gestoppt. Mit dem erhaltene Wert wird in einem lokalen Verzweigungs-Verfahren zur Lösung von MIPs weitergearbeitet, um eventuell eine qualitativ besseren Punkt zu erhalten.

3. Wie bei Variante 2 wird das FP-Verfahren gestoppt, sobald ein zulässiger Punkt gefunden wurde. Daraufhin wird sowohl dieser Punkt, als auch die Lösung der

Relaxation dazu verwendet, um mit einem lokalen Suchverfahren, der sogenannten Relaxation Induced Neighborhood Search (RINS), weiterzuarbeiten. Diese von Danna et al. (2005) entwickelte Methode erwies sich, gegenüber lokalen Verzweigungs-Verfahren, als deutlich effizienter.

2.3 Implementation des FP in Matlab

Um die Feasibility Pump selbst zu testen, wurde sie für binäre MIPs in Matlab implementiert und daraufhin auf 137 Testprobleme aus MIPLIB 2010 (Achterberg et al.) angewendet.

2.3.1 Konvertierung der Testprobleme

MIPLIB (Mixed Integer Problem Library) wurde 1992 von Robert E. Bixby, E.A. Boyd und R.R. Indovina gegründet, um eine Sammlung realer IPs und MIPs der Öffentlichkeit zugänglich zu machen. Im Jahr 2003 und 2010 wurde diese Online-Bibliothek an Testproblemen von einer Gruppe von Forschern nochmals aktualisiert, wobei die letzte Version für unsere Tests verwendet wurde. Auf MIPLIB 2010 stehen nun folgende Klassen von Problemen zur Verfügung:

- BP: In diesem Fall, besteht das Problem ausschließlich aus binären Variablen.
- IP: Das sind Probleme, die nur aus ganzzahligen Variablen bestehen.
- MBP: Hier sind sowohl kontinuierliche, als auch binäre Variablen enthalten.
- MIP: Wir haben es hier mit allgemeinen gemischt-ganzzahligen Problemen zu tun.

Diese Probleme sind weiters in folgende 3 Schwierigkeitsklassen unterteilt:

1. Leicht: In diese Klasse fallen all jene Probleme, die mit kommerziellen Softwarepaketen in weniger als einer Stunde gelöst werden können.
2. Schwer: In diesem Fall, wurde zwar bereits eine optimale Lösung für das Problem gefunden, jedoch nicht innerhalb oben angegebener Zeit bzw. nicht mit jeder beliebigen Software.
3. Offen: Diese Schwierigkeitsklasse enthält jene Probleme, deren optimale Lösung noch nicht bekannt ist.

Für unsere Zwecke haben wir also 137 Probleme aus der Klasse MBP verwendet, wobei davon 88 in die Klasse der leichten Prbleme fallen, 22 zu den schweren Problemen gehören und 27 zu den offenen. Unsere erste Aufgabe bestand nun darin, die Testprobleme, welche im `mps`-Format vorlagen, ins Matlab-Format zu konvertieren. Dafür steht von BORCHERS ein public domain Code zur Verfügung, den wir zur Verbesserung der Geschwindigkeit noch ein wenig modifiziert haben. Durch Aufrufen der Funktion `readmps` kann man nun die `mps`-Dateien in Matlab-Dateien umwandeln, was wir vorab für die Testprobleme durch folgende Implementation getan haben:

```
[status,files_string_mps]=dos('dir /B problems\mps\*.mps');
data=textscan(files_string_mps,'%s','delimiter','\n');
file_mps=data{1};
for i=1:size(file_mps,1)
    clear problem
    disp(['Processing file: ' file_mps{i}])
    problem=readmps(['problems\mps\' file_mps{i}]);
    save(['problems\mat\' file_mps{i}(1:(end-4)) '.mat'],'problem');
end
```

Abbildung 4: convert mps-files to mat-files

Dabei verwenden wir die Matlab-Kommandos `dos` und `textscan`, um die Dateien vollautomatisch einzulesen und daraufhin unter demselben Namen als Matlab-file abzuspeichern.

2.3.2 Implementation des Algorithmus

Kommen wir nun zur Implementation der Feasibility Pump. Wir können nun einfach durch die Zuordnung `s=load(['problems\mat\' file{i}])` die Daten für die Optimierung speichern und gegebenenfalls aufrufen, da wir die `mps`-Dateien bereits umgewandelt haben. Damit dies wieder vollautomatisch geschieht, greifen wir auch hier auf die Kommandos `dos` und `textscan` zurück. Folgende Auflistung zeigt diejenigen eingelesenen Daten, die für uns relevant sind:

- `name`: Enthält den Namen des Optimierungsproblems.
- `objsesense`: Durch 'MINIMIZE' oder 'MIN' bzw. 'MAX' oder 'MAXIMIZE' wird angegeben, ob es sich um ein Minimierungs- oder Maximierungsproblem handelt.
- `A`: Hier erhalten wir die Matrix, deren Zeilen sowohl die Gleichungs- und Ungleichungsnebenbedingungen als auch die Zielfunktion enthalten.
- `rowtypes`: Dies ist der Vektor, der angibt, ob es sich bei den Zeilen mit jeweiligem Index um Gleichungs-, Ungleichungsnebenbedingungen oder die Zielfunktion handelt. Dabei bedeutet 'L' bzw. 'G', dass wir es mit Ungleichungsnebenbedingungen

zu tun haben (kleiner bzw. größer), bei 'E' handelt es sich um Gleichungsnebenbedingungen und bei 'N' um die Zielfunktion.

- `lbnds`: Hier erhalten wir den Vektor der unteren Schranken.
- `ubnds`: Dabei handelt es sich um den Vektor der oberen Schranken.
- `rhs`: Dieser Vektor beeinhaltet die rechte Seite der Nebenbedingungen bzw. der Zielfunktion.
- `bintflags`: Dies ist der Vektor, der angibt, welche Variablen binär sein sollen. Es gilt `bintflags(i)=1` wenn $x_i \in \{0, 1\}$ gelten soll.
- `intflags`: Gilt hier `intflags(i)=1`, dann unterliegt x_i einer Ganzzahligkeitsbedingung.

Möchte man nun z.B. den Namen des Optimierungsproblems aufrufen, kann man das einfach durch Eingabe von `s.problem.name` tun. Die anderen Daten erhält man analog.

Im nächsten Schritt bestimmen wir die durchschnittliche Anzahl `T` der Variablen die vertauscht werden sollen und die maximale Berechnungszeit `TL`. Hier haben wir `T`=20 und `TL`=1200 gewählt und die Anzahl der Iterationen `nIT` zu Beginn auf 0 gesetzt. Darüber hinaus haben wir eine Variable `no_sol` eingeführt, die uns Auskunft darüber geben soll, ob ein Punkt für ein gewisses Problem gefunden wurde oder nicht. Diese wird vorerst auch auf 0 gesetzt. Wird kein Punkt gefunden, unterscheiden wir zwischen folgenden 3 Gründen und die Variable `no_sol` wird dementsprechend abgeändert:

- `no_sol`=1: Das festgelegte Zeitlimit wurde überschritten, bevor ein zulässiger Punkt gefunden wurde.
- `no_sol`=2: Das Optimierungsproblem ist unzulässig.
- `no_sol`=3: Es gilt $\frac{3T}{2} > n$, wobei n die Anzahl der zu optimierenden Variablen ist. Das darf insofern nicht passieren, weil in Zeile 9 eine zufällige Anzahl `TT` an Variablen aus dem Intervall $[\frac{T}{2}, \frac{3T}{2}]$ ausgewählt werden, die vertauscht werden sollen. Das ist natürlich nicht möglich, wenn `TT` einen größeren Wert als n annimmt.

Nun ist es an der Zeit, die eingelesenen Daten so aufzubereiten, dass wir mit der Optimierung beginnen können. Für die Lösung der LP-Relaxation und die Optimierung der Abstandsfunktion haben wir den Solver Gurobi 5.0.2 (Gurobi Optimization

(2013)) gewählt, welcher direkt über Matlab verwendbar ist. Mit Hilfe dieses Solvers können wir nun Probleme folgender Gestalt lösen:

$$\begin{aligned} \min\ & c^T x \\ \text{s.t. } & Ax = b \\ & l \leq x \leq u, \end{aligned}$$

wobei wir eine Matlab-Struktur mit den folgenden Komponenten benötigen:

`model.obj`: Diese Komponente bezeichnet den Vektor c der Zielfunktion, den wir folgendermaßen konstruieren. Da wir die Input-Daten später verändern werden, um

```
% Objective function min(c^t x)
obj_ind=find(char(s.problem.rowtypes)=='N');
model.obj=full(s.problem.A(obj_ind,:));
model.obj_start=model.obj;
size_model.obj=size(model.obj);
```

die Abstandsfunktion zu optimieren, speichern wir die ursprüngliche Version unter `model.obj_start` ab. Analog gehen wir bei den weiteren Input-Daten vor.

`model.A & model.rhs`: Wir extrahieren die dünn besetzte Matrix A und die rechte Seite b der Nebenbedingungen wie folgt:

```
% Equality and inequality constraints (Ax<=b, Ax<=b, Ax=b)
model.A=s.problem.A;
model.A(obj_ind,:)=[];
model.A_start=model.A;
model.rhs=full(s.problem.rhs);
model.rhs_obj=model.rhs(obj_ind);
model.rhs(obj_ind)=[];
model.rhs_start=model.rhs;
```

`model.sense`: Diese Komponente gibt die Art der Nebenbedingungen an (kleiner, größer oder gleich):

```
% Create vector that determines the sense of equality and inequality
% constraints e(i)=-1 (less than), e(i)=0 (equal), e(i)=1 (greater then)
rhs1_ind=find(char(s.problem.rowtypes)=='E');
rhs2_ind=find(char(s.problem.rowtypes)=='G');
rhs3_ind=find(char(s.problem.rowtypes)=='L');
model.sense=char(zeros(m,1));
model.sense(rhs1_ind)='=';
model.sense(rhs2_ind)='>';
model.sense(rhs3_ind)='<';
model.sense(obj_ind)=[];
model.sense_start=model.sense;
```

`model.lb & model.ub:` Die Vektoren der unteren und oberen Schranken werden folgendermaßen zusammengestellt:

```
% Set boundarys for elements of solution vectros
model.lb=full(s.problem.lbnds);
model.lb_start=model.lb;
model.ub=full(s.problem.ubnds);
model.ub_start=model.ub;
```

Darüber hinaus sind gewisse Parameter zu setzen, die den Ablauf des Lösungsalgorithmus bestimmen. Wir wählen ein duales Simplexverfahren, welches sich gerade für große LP-Probleme bewährt hat, ohne zusätzliche Statusausgabe aus Geschwindigkeitsgründen:

```
% Set parameters for optimization
params.method=1;
params.outputflag=0;
```

Nun können wir mit der Lösung der Relaxation beginnen, indem wir Gurobi aufrufen:

```
% Solve relaxed problem
result=gurobi(model, params);
```

Im nächsten Schritt überprüfen wir, ob eine Lösung für die LP-Relaxation gefunden wurde. Ist das nicht der Fall, so ist das Problem unzulässig, und der Algorithmus endet an dieser Stelle. Wurde jedoch eine Lösung gefunden, so ist diese in `result.x` gespeichert. Unsere Aufgabe ist es nun festzustellen, ob bereits alle Ganzzahligkeitsbedingungen erfüllt sind. Dafür löschen wir zunächst die Eintragungen, die kontinuierlich sein dürfen, und überprüfen danach, ob der verbleibende Vektor ganzzahlig ist. Ist das der Fall, so konstruieren wir den Vektor `result.x_new`, der wieder alle Eintragungen, also auch die kontinuierlichen, enthält und berechnen mit Hilfe dessen den Zielfunktionswert:

```
% Is the result already an integer? -> If yes, we are done!
if result.x==round(result.x)
   result.x_new=zeros(n,1);
   result.x_new(cont_ind)=result.x_cont;
   result.x_new(int_ind)=result.x;
   objval=model.obj*result.x_new-model.rhs_obj;
```

Nachdem wir die erhaltenen Ergebnisse abgespeichert haben, sind wir auch in diesem Fall am Ende des Algorithmus angekommen. Meist wird jedoch die Lösung nicht sofort alle Ganzzahligkeitsbedingungen erfüllen und wir müssen fortfahren. An dieser Stelle betreten wir den ersten Pumping Cycle, in dem vorerst überprüft wird, ob die

maximale Berechnungszeit überschritten wurde, was wiederum zu einem Abbruch des Algorithmus führt. Sind wir noch innerhalb des Zeitlimits, wird `nIT=nIT+1` gesetzt und die Abstandsfunktion konstruiert. Zuerst setzen wir $\tilde{x}$ fest, welches wir durch Rundung von `result.x` erhalten. Um die Zielfunktion in die Gestalt $c^T x$ zu bringen, verwenden wir folgende vereinfachte Form für die Optimierung:

$$\Delta_{new}(x, \tilde{x}) = \sum_{j \in I: \tilde{x}_j = 0} x_j + \sum_{j \in I: \tilde{x}_j = 1} -x_j.$$

Dadurch wird nur die Konstante der Abstandsfunktion entfernt, die zwar den Zielfunktionswert ändert, aber nicht den optimalen Punkt. Durch Setzen von

$$c_j = \begin{cases} -2\tilde{x}_j + 1 & j \in I \\ 0 & j \notin I, \end{cases}$$

erhalten wir oben definierte Funktion in der gewünschten Form. Unter den ursprünglichen Nebenbedingungen können wir somit wieder den Gurobi Solver darauf anwenden. Es könnte nun passieren, dass es mit der so konstruierten Abstandsfunktion keinen Punkt gibt, der den ursprünglichen Nebenbedingungen genügt. In diesem Fall wollen wir im Gegensatz zur originalen FP schon jetzt einige Variablen vertauschen, wobei wir diesen Vorgang wiederholen, bis entweder das Zeitlimit überschritten wurde oder ein zulässiger Punkt gefunden wurde:

```
% if the problem is infeasible -> flip a random number
% of variables TT with highest |xstar_j-xtilde_j|
while strcmp(result.status,'OPTIMAL')==0
    if toc>=TL
        no_sol=1;
        objval=NaN;
        disp('No integer solution found');
    break;
    else
        result_cont=result.x(cont_ind);
        result.x(cont_ind)=[];
        result_int=result.x;
        TT=round(T*rand(1)+(T/2));
        [sorted,Indizes]=sort(abs(result.x-xtilde),'descend');
        xtilde(Indizes(1:TT))=1-xtilde(Indizes(1:TT));
        dist_xtilde=zeros(n,1);
        dist_xtilde(int_ind)=xtilde*(-2)+ones(size(xtilde));
        dist_xtilde(cont_ind)=0;
        model.obj=dist_xtilde;
        result=gurobi(model, params);
    end
end
```

Nach diesem Schritt wird erneut überprüft, ob nun alle Ganzzahligkeitsbedingungen

erfüllt sind. Wie oben wird also wieder der kontinuierliche Anteil des Vektors gelöscht. Sei nun x_{int} der erhaltene Vektor und $[x_{int}]$ sein gerundeter Wert. Diesmal geben wir uns auch damit zufrieden, wenn $|x_{int} - [x_{int}]| < \epsilon$ gilt, da kleine Abweichungen von der Ganzzahligkeit durch Rundungsfehler entstehen können. Dabei wollen wir $\epsilon = 10^{-7}$ setzen. Wichtig ist jedoch, dass wir im Folgenden mit dem echt ganzzahligen Wert weiterarbeiten. Haben wir also nun einen beinahe ganzzahligen Punkt gefunden, so setzen wir $x_{int} = [x_{int}]$, fixieren diesen im ursprünglichen Problem und optimieren nun nochmals die kontinuierlich Variablen, was wir wie folgt implementiert haben:

```
% Is the result already an integer? -> If yes, fix integer
% variables and optimize continuous variables
if norm(result.x-round(result.x),2)<1.e-7
    model.A=sparse(model.A_start);
    model.rhs=model.rhs_start-model.A(:,int_ind)*result_int;
    model.A(:,int_ind)=[];
    model.obj=model.obj_start;
    model.obj_int=model.obj(int_ind);
    model.obj(int_ind)=[];
    model.lb=model.lb_start;
    model.lb(int_ind)=[];
    model.ub=model.ub_start;
    model.ub(int_ind)=[];
    result=gurobi(model, params);
    result.x_new=zeros(n,1);
    result.x_new(cont_ind)=result.x;
    result.x_new(int_ind)=result_int;
    objval=model.obj_start*result.x_new-model.rhs_obj;
    nIT;
    toc
    break;
end
```

Haben wir jedoch wieder keinen Punkt gefunden, der alle Ganzzahligkeitsbedingungen erfüllt, wird vorerst überprüft ob $\tilde{x}$ mit dem gerundeten Wert von x_{int} übereinstimmt. Ist das der Fall, so werden, wie oben beschrieben, einige Variablen vertauscht. Ansonsten setzt man $\tilde{x} = [x_{int}]$:

```
% Check if the rounding of xstar changed
if xtilde==round(result.x)
    % Not changed -> flip a random number of variables TT with
    % highest |xstar_j-xtilde_j|
    % derive random TT between T/2 and 3T/2
    TT=round(T*rand(1)+(T/2));
    [sorted,Indizes]=sort(abs(result.x-xtilde),'descend');
    % Attention!!! TT must be smaller than n -> 3T/2 has to be
    % smaller than n.
    xtilde(Indizes(1:TT))=1-xtilde(Indizes(1:TT));
else
    xtilde=round(result.x);
end
```

An dieser Stelle kehrt man nun wieder an den Anfang des Pumping Cycles zurück, und die Prozedur startet erneut.

2.3.3 Auswertung der Ergebnisse

Wie bereits erwähnt haben wir die FP auf 137 Testprobleme aus MIPLIB 2010 angewendet. Darüber hinaus haben wir diese MIPs mit Hilfe von Gurobi heuristisch gelöst, um die beiden Verfahren einander gegenüberstellen zu können. Gurobi haben wir wieder über Matlab aufgerufen. Das Einlesen der Daten wurde dabei analog zur FP erledigt, wobei eine weitere Variable für die Matlab-Struktur benötigt wird:

`model.vtype` Wir bestimmen, welche Variablen der Ganzzahligkeit unterliegen und welche nicht. `'B'` bedeutet hier, dass es sich um eine binäre Variable handelt, und `'C'`, dass wir es mit einer kontinuierlichen zu tun haben. Um gleichwertige Bedingungen

```
% Determine integer and continous variables
 int_ind = s.problem.bintflags==1 | s.problem.intflags==1;
 cont_ind = find(~int_ind);
 int_ind = find(int_ind);
 model.vtype=char(zeros(n,1));
 model.vtype(int_ind)='B';
 model.vtype(cont_ind)='C';
```

zu schaffen, haben wir Gurobi nach dem Finden des ersten zulässigen Punktes abgebrochen. Außerdem haben wir auch hier ein Zeitlimit von 1200 Sekunden festgelegt. Wir wollen im Folgenden Geschwindigkeit und Qualität der erhaltenen Punkte beider Algorithmen miteinander vergleichen. Darüber hinaus wollen wir die Abweichung der Zielfunktionswerte von den optimalen Werten berechnen, sofern ein solcher bekannt ist.

Folgende Tabelle zeigt die Namen der verwendeten Testprobleme mit der zugehörige Anzahl der Variablen (n) und Nebenbedingungen (m), als auch die Anzahl der binären (B) und kontinuierlichen Variblen (C).

Name	n	m	B	C	Name	n	m	B	C
30-70-4.5-0.95-100	10976	12526	10975	1	neos-847302	737	609	729	8
aflow40b	2728	1442	1364	1364	neos-885086	4860	11574	2430	2430
atm20-100	6480	4380	2220	4260	neos-911880	888	83	840	48
b2c1s1	3872	3904	288	3584	neos-934278	23123	11495	19955	3168
beasleyC3	2500	1750	1250	1250	neos-942830	882	803	834	48
berlin-5-8-0	1083	1532	794	289	neos-948126	9551	7271	6965	2586
bg512142	792	1307	240	552	neos13	1827	20852	1815	12
biella1	7328	1203	6110	1218	neos15	792	552	160	632
bienst2	505	576	35	470	neos6	8786	1036	8340	446
binkar10-1	2298	1026	170	2128	net12	14115	14021	1603	12512
blp-ar98	16021	1128	15806	215	newdano	505	576	56	449
blp-ic97	9845	923	9753	92	nobel-eu-DBE	3771	879	1639	2132
core2536-691	15293	2539	15284	9	npmv07	220686	76342	1880	218806
core4872-1529	24656	4875	24645	11	ns1111636	360822	13895	13200	347622
csched007	1758	351	1457	301	ns1116954	12648	131991	7482	5166
csched008	1536	351	1284	252	ns1158817	1804022	68455	66022	1738000
csched010	1758	351	1457	301	ns1208400	2883	4289	2880	3
dano3mip	13873	3202	552	13321	ns1606230	4173	3503	3633	540
danoint	521	664	56	465	ns1644855	30200	40698	10000	20200
dg012142	2080	6310	640	1440	ns1702808	804	1474	666	138
dolom1	11612	1803	9720	1892	ns1830653	1629	2932	1458	171
g200x740i	1480	940	740	740	ns1856153	11998	35407	11956	42
glass4	322	396	302	20	ns1904248	38458	149437	38416	42
gmu-35-40	1205	424	1200	5	ns2081729	661	1190	600	61
gmu-35-50	1919	435	1914	5	ns2118727	167440	163354	159514	7926
gmut-75-50	68865	2565	68859	6	ns2122603	19300	24754	7588	11712
gmut-77-40	24338	2554	24332	6	ns2124243	156083	139280	16447	139636
k16x240	480	256	240	240	ns2137859	103361	206726	103041	320
leo1	6731	593	6730	1	ns930473	11328	23240	11176	152
leo2	11100	593	11099	1	nsr8k	38356	6284	32040	6316
liu	1156	2178	1089	67	p100x588b	1176	688	588	588
lotsize	2985	1920	1195	1790	p80x400b	800	480	400	400
map06	164547	328818	146	164401	pg	2700	125	100	2600
map10	164547	328818	146	164401	pg5-34	2600	225	100	2600
map14	164547	328818	146	164401	pigeon-10	490	931	400	90
map18	164547	328818	146	164401	pigeon-11	572	1123	473	99
markshare-5-0	45	5	40	5	pigeon-12	660	1333	552	108
n3705	10000	5150	5000	5000	pigeon-13	754	1561	637	117
n370a	10000	5150	5000	5000	pigeon-19	1444	3307	1273	171
neos-1112782	4140	2115	2070	2070	probportfolio	320	302	300	20
neos-1112787	3280	1680	1640	1640	qiu	840	1192	48	792
neos-1140050	40320	3795	38640	1680	r80x800	1600	880	800	800
neos-1171692	1638	4239	819	819	ran14x18.disj-8	504	447	252	252
neos-1171737	2340	4179	1170	1170	ran14x18	504	284	252	252
neos-1225589	1300	675	650	650	ran16x16	512	288	256	256
neos-1311124	1092	1643	546	546	rmatr100-p10	7359	7260	100	7259
neos-1396125	1161	1494	129	1032	rmatr100-p5	8784	8685	100	8684
neos-1426635	520	796	260	260	rmatr200-p10	35254	35055	200	35054
neos-1426662	832	1914	416	416	rmatr200-p20	29605	29406	200	29405
neos-1429212	416040	58726	54756	361284	rmatr200-p5	37816	37617	200	37616
neos-1436709	676	1417	338	338	rocII-4-11	9234	21738	9086	148
neos-1440460	468	989	234	234	rocII-7-11	16101	37215	15851	250
neos-1442119	728	1524	364	364	rocII-9-11	20679	47533	20361	318
neos-1442657	624	1310	312	312	satellites1-25	9013	5996	8509	504
neos-1601936	4446	3131	3906	540	satellites2-60-fs	35378	16516	34324	1054
neos-1605061	4111	3474	3570	541	satellites2-60	35378	20916	34324	1054
neos-1605075	4173	3467	3633	540	satellites3-40-fs	81681	35553	79961	1720
neos-476283	11915	10015	5588	6327	satellites3-40	81681	44804	79961	1720
neos-506422	2527	6811	63	2464	shipsched	13594	45554	10549	3045
neos-520729	91149	31178	30708	60441	siena1	13741	2220	11775	1966
neos-693347	1576	3192	1405	171	sing2	31630	28891	23377	8253
neos-738098	9093	25849	8946	147	swath	6805	884	6724	81
neos-799711	41998	59218	910	41088	uc-case3	37749	52003	11256	26493
neos-824661	45390	18804	15640	29750	uct-subprob	2256	1973	379	1877
neos-824695	23970	9576	8500	15470	unitcal-7	25755	48939	2856	22899
neos-826650	5912	2414	5792	120	usAbbrv.8.25-70	2312	3291	1681	631
neos-826694	16410	6904	16290	120	van	12481	27331	192	12289
neos-826812	15864	6844	10350	5514	zib54-UUE	5150	1809	81	5069
neos-826841	5516	2354	3488	2028					

Folgende Tabellen dienen als Grundlage unserer Auswertungen:

	FP Matlab			Gurobi		MIPLIB
Name	**Wert**	**nIT**	**Zeit**	**Wert**	**Zeit**	**Optimaler Wert**
30-70-4.5-0.95-100	448	2	2.328382	335	0.192842	3
aflow40b	7712	4	0.197207	2924	0.137829	1168
atm20-100	NA	10273	1200	NA	1200	2.46362e+006
b2c1s1	**48672.8**	7	1.401204	**67975.52**	0.291644	25687.9
beasleyC3	945	16	0.190117	937	0.078302	754
berlin-5-8-0	85	22	0.276481	79	0.026388	62
bg512142	1.20739e+008	1	0.088933	1.36527e+007	0.275749	NA
biella1	**1.34027e+007**	5	1.977749	**2.96025e+008**	0.237372	3.06501e+006
bienst2	64.3333	149	0.591903	64.3333	0.031781	54.6
binkar10-1	**8493.56**	12	0.147244	**11114.33**	0.017546	6742.2
blp-ar98	NA	10143	1200	6557.357	7.247088	6205.21
blp-ic97	7654.528	7	0.539322	5400.546	1.113043	4025.02
core2536-691	**713**	2	4.330045	**921**	0.898162	689
core4872-1529	**1635**	3	50.505411	**1983**	0.913598	NA
csched007	NA	126729	1200	572	79.254785	351
csched008	NA	137789	1200	189	14.615404	173
csched010	NA	110034	1200	505	59.797713	408
dano3mip	**756.625**	2	12.250694	**935.3333**	0.406780	NA
danoint	88	81	1.358748	83	0.351105	65.6667
dg012142	1.53407e+008	1	0.272561	7.85897e+007	3.074486	2.30087e+006
dolom1	**6.93279e+008**	5	5.602994	**1.75217e+009**	1.066513	NA
g200x740i	45263	12	0.084738	44069	0.026546	NA
glass4	**3.56670e+009**	1	0.011169	**3.69170e+009**	0.014242	1.20001e+009
gmu-35-40	-2.05522e+006	7	0.056651	-2.09367e+006	0.035736	-2.40673e+006
gmu-35-50	-2.20797e+006	31	0.282090	-2.23453e+006	0.071709	-2.60796e+006
gmut-75-50	-1.36281e+007	13	12.150390	-1.41156e+007	12.602499	NA
gmut-77-40	-1.34237e+007	15	2.890671	-1.41448e+007	3.228730	NA
k16x240	13590	2	0.012069	11359	0.007686	10674
leo1	**6.37095e+008**	4	0.594587	**7.55044e+008**	0.377726	4.04228e+008
leo2	1.44050e+009	3	0.879471	7.53083e+008	0.876126	4.04077e+008
liu	**6022**	1	0.029321	**6450**	0.033622	NA
lotsize	4.06539e+006	230	2.757931	4.00800e+006	3.874588	1.4802e+006
map06	-23	1	7.154907	-149	3.152942	-289
map10	-204	1	7.241151	-205	3.358256	-495
map14	**-423**	1	6.586298	**-422**	3.205956	-674
map18	**-628**	1	6.486691	**-540**	3.210292	-847
markshare-5-0	**617**	3	0.009946	**5335**	0.003700	1
n3705	1.78680+006	4	0.188778	1.34254e+006	0.219713	NA
n370a	2.01272+006	5	0.276480	1.35304e+006	0.225719	NA
neos-1112782	2.20024e+013	143	1.771252	2.20009e+013	0.061362	5.72103e+011
neos-1112787	1.95014e+013	5	0.069967	1.95014e+013	0.047667	5.65032e+011
neos-1140050	Infeasible			Infeasible		NA
neos-1171692	**-233**	1	0.092310	**-21**	0.123185	-273
neos-1171737	**-156**	1	0.112789	**-20**	0.253466	-195
neos-1225589	5.42432e+009	2	0.019469	5.00000e+009	0.017350	1.23107e+009
neos-1311124	**-103**	1	0.020146	**-63**	0.021290	NA
neos-1396125	NA	46652	1200	3000.053	5.496137	3000.05
neos-1426635	**-168**	1	0.013795	**-64**	0.013877	-176
neos-1426662	**-33**	1	0.028162	**-14**	0.030637	-44
neos-1429212	NA	553	1200	NA	1200	NA
neos-1436709	**-77**	1	0.021975	**-40**	0.022809	-128

	FP Matlab			Gurobi		MIPLIB
Name	**Wert**	**nIT**	**Zeit**	**Wert**	**Zeit**	**Wert**
neos-1440460	**-157**	1	0.017177	**-56**	0.017398	-179.25
neos-1442119	**-107**	1	0.023927	**-56**	0.024682	-181
neos-1442657	**-106**	1	0.019916	**-48**	0.021705	-154.5
neos-1601936	NA	38805	1200	20382	34.822571	3
neos-1605061	NA	10497	1200	27	351.157294	12
neos-1605075	NA	25908	1200	84213	64.795348	9
neos-476283	631.7513	2	38.494269	411.8969	10.255979	406.363
neos-506422	740	1	0.090307	740	0.147575	0
neos-520729	**-1.37500e+006**	2	24.940855	**1.11865e+009**	1.347743	-1.38500e+006
neos-693347	NA	6315	1200	241	36.735692	234
neos-738098	NA	636	1200	-1099	9.076211	-1099
neos-799711	NA	2383	1200	-9.01794e+006	6.281922	-1.11702e+007
neos-824661	97	5	1.857253	87	0.591745	33
neos-824695	**71**	5	0.646034	**75**	0.290665	31
neos-826650	NA	26954	1200	32	4.253396	29
neos-826694	85	4	0.535809	61	0.309369	58
neos-826812	80.0114	3	0.537236	61.026	0.207594	58.011
neos-826841	NA	59929	1200	31.0086	1.056794	29.0082
neos-847302	8	22	0.522245	5	2.138197	4
neos-885086	**-145**	1	0.979601	**-11**	0.942227	-243
neos-911880	**346.91**	1	0.028711	**2612.76**	0.012522	54.76
neos-934278	**69238**	1	10.156664	**350113**	0.314054	260
neos-942830	**29**	11	0.248261	**32**	2.323871	16
neos-948126	**13743**	2	17.206507	**35697**	0.224252	2607
neos13	**-46.3368**	1	2.252018	**-28.03964**	0.898004	-95.4748
neos15	1.62488e+008	5	0.038061	143201.2	0.073343	80598.4
neos6	**122**	6	1.181903	**140**	3.864634	83
net12	337	14	1.723895	337	0.847520	214
newdano	**81**	3	0.025691	**91**	0.133302	65.6667
nobel-eu-DBE	**4.24036e+006**	3	0.176374	**7.51887e+006**	0.194298	608910
npmv07	**1.04810e+011**	1	4.591593	**1.04811e+011**	4.868273	1.04810e+011
ns1111636	248	7	92.985187	240	5.754739	NA
ns1116954	NA	1370	1200	NA	1200	0
ns1158817	Infeasible			Infeasible		Infeasible
ns1208400	NA	10732	1200	2	102.387223	2
ns1606230	NA	33030	1200	75233	19.430827	21
ns1644855	-1001	3	119.539625	-1519	10.043258	-1524.33
ns1702808	NA	167163	1200	Infeasible		Infeasible
ns1830653	NA	26504	1200	39622	33.647739	20622
ns1856153	NA	7189	1200	79.65941	0.587814	NA
ns1904248	NA	48	1200	81.36815	3.007127	NA
ns2081729	NA	141497	1200	10.3	0.056471	9
ns2118727	NA	247	1200	Infeasible		Infeasible
ns2122603	NA	3613	1200	6.43229e+007	9.425051	NA
ns2124243	127950	5	8.259067	89360	112.747615	NA
ns2137859	NA	104	1200	172463	1.801540	NA
ns930473	**989153**	10	8.889767	**1.00531e+006**	7.546329	NA
nsr8k	5.38505e+009	5	294.682819	4.95349e+009	1.732536	NA
p100x588b	66533	5	0.029340	58977	0.016236	47878
p80x400b	50618	5	0.023729	45896	0.013075	39667
pg	**-6282.032**	3	0.051300	**-5407.687**	0.042761	-8674.34
pg5-34	-13363.62	1	0.028483	-13976.7	0.244492	-14339.4

	FP Matlab			Gurobi		MIPLIB
Name	**Wert**	**nIT**	**Zeit**	**Wert**	**Zeit**	**Wert**
pigeon-10	NA	196422	1200	0	0.017328	-9000
pigeon-11	NA	165704	1200	0	0.022691	-10000
pigeon-12	NA	146020	1200	0	0.023883	-11000
pigeon-13	NA	126396	1200	0	0.027308	-12000
pigeon-19	NA	60060	1200	0	0.065198	NA
probportfolio	**21.7294**	2	0.03082550	**NA**	1200	16.7342
qiu	**936.875**	4	0.159756	**1577.109**	0.102521	-132.873
r80x800	9496	5	0.038760	5520	0.020258	5332
ran14x18.disj-8	5605	4	0.103655	5129	0.075901	3735
ran14x18	6704	2	0.012397	4265	0.014377	3712
ran16x16	8710	5	0.020336	4333	0.011040	3823
rmatr100-p10	**666**	1	0.366008	**763**	0.048782	423
rmatr100-p5	1516	2	0.798859	1322	0.060140	976
rmatr200-p10	**3017**	1	4.415995	**3333**	0.444810	2017
rmatr200-p20	1729	1	2.307047	1402	0.317731	837
rmatr200-p5	**6039**	3	11.302861	**6066**	0.517057	4521
rocII-4-11	NA	4054	1200	-3.59727	0.514901	-6.65564
rocII-7-11	NA	2208	1200	-3.60487	43.307046	NA
rocII-9-11	NA	1493	1200	NA	1200	NA
satellites1-25	NA	21750	1200	49	0.239839	-5
satellites2-60-fs	NA	1887	1200	48	0.566091	-19
satellites2-60	NA	569	1200	80	1.295923	-19
satellites3-40-fs	NA	157	1200	39	1.524412	-25
satellites3-40	NA	118	1200	80	3.593722	-25
shipsched	NA	3690	1200	181464	12.738654	NA
siena1	**1.08743e+008**	5	17.811629	**2.85981e+009**	1.588913	NA
sing2	**3.33680e+007**	5	4.035224	**3.27698e+008**	0.950908	**NA**
swath	1333.759	156	3.779948	1596.818	2.847867	467.407
uc-case3	NA	1706	1200	376317.5	1.318055	7204.92
uct-subprob	**473**	1	0.098820	**497**	0.052906	314
unitcal-7	NA	3064	1200	2.85674e+007	29.778671	1.96356e+007
usAbbrv.8.25-70	**169**	46	1.273438	**200**	0.050718	NA
van	**15.76663**	3	9.526011	**61.05925**	1.839264	NA
zib54-UUE	1.95034e+007	10	1.451001	1.88843e+007	0.602545	1.03340e+007

Mit Hilfe dessen haben wir unsere Auswertung gestartet, wobei wir die Ergebnisse wieder in einer Tabelle zusammengefasst haben.

Name	**gef.**	**nicht gef.**	**schneller**	**besser**	ϕ **Geschw.**	ϕ **Abw. Opt.**
FP Matlab	93	42	29	46	15.23761	42.82
Gurobi	128	5	62	41	9.03958	711.66

Die einzelnen Spalten wollen wir nun genauer erklären:

- **gef.**: Diese Werte entsprechen der Anzahl der Probleme, für die ein Punkt mit der jeweiligen Methode gefunden wurde.
- **nicht gef.**: Diese Spalte steht für die Anzahl der Probleme, die nicht im vorgegebenen Zeitlimit heuristisch gelöst werden konnten. Probleme, die sich als unzulässig herausstellten, wurden hierbei nicht berücksichtigt.

- **schneller**: Hier wird angegeben ob der jeweilige Algorithmus schneller zu einem zulässigen Punkt gelangt ist, wobei wir nur jene Probleme betrachtet haben, die von beiden Methoden heuristisch gelöst werden konnten.

- **besser**: Auch hier haben wir nur jene Probleme betrachtet, bei denen beide Algorithmen einen Punkt gefunden haben und sie daraufhin bezüglich deren Qualität verglichen.

- ϕ **Geschw.**: Diese Werte entsprechen der durchschnittlichen Geschwindigkeit der beiden Methoden.

- ϕ **Abw. Opt.**: In dieser Spalte ist jeweils die durchschnittliche Abweichung zum optimalen Wert gegeben. Dafür haben wir jeweils die Zielfunktionswerte mit den in MIPLIB angegebenen optimalen Werten ins Verhältnis gestellt, und davon sodann den Durchschnitt berechnet. Die größte Abweichung der jeweiligen Methoden haben wir aus der Berechnung ausgeschlossen, weil diese das Ergebnis erheblich verfälscht hätten.

Wir sehen also, dass Gurobi, sowohl durch Geschwindigkeit, als auch durch die Anzahl der gefundenen Punkte, überzeugt. Die Qualität der Punkte lässt jedoch oft zu Wünschen übrig. Im Vergleich dazu, liefert die FP erheblich bessere heuristische Lösungen, jedoch bei Weitem nicht so viele.

Wir wollen nun noch weitere Auswertungen starten. Durch Rundungsfehler kann es nämlich passieren, dass die Nebenbedingungen eines Optimierungsproblems geringfügig verletzt werden. Das ist teilweise nicht weiter schlimm, es gibt jedoch auch Probleme, wo die exakte Einhaltung der Nebenbedingungen essenziell ist. In diesem Fall, wären solche fast zulässigen Punkte unbrauchbar. Deshalb wollen wir nun überprüfen, ob bei unseren Ergebnissen auch alle gefundenen Punkte den Gleichungs-, Ungleichungs- und Schrankennebenbedingungen genügen. Dies wollen wir tun, indem wir die gefundenen Punkte in das Originalproblem einsetzen und gegebenenfalls die Größe der Übertretung angeben. Wieder haben wir dazu Matlab zu Hilfe genommen, wobei wir diejenigen Probleme, wo Übertretungen festgestellt wurden, für die jeweilige Methode, in folgenden Tabellen zusammengefasst haben. Dabei bezeichnet EC die Gleichungsnebenbedingungen, GC und LC die Ungleichungsnebenbedingungen, und lb und ub die Schrankenbedingungen. Ein Wert im jeweiligen Tabellenfeld gibt dabei die Größe der Verletzung an, und OK bedeutet, dass keine Verletzung vorliegt.

FP Matlab						
Name	**EC**		**GC**	**LC**	**lb**	**ub**
b2c1s1	-1.42e-014	1.42e-014	OK	OK	OK	OK
blp-ic97	-4.12e-013	OK	OK	OK	OK	OK
dano3mip	OK	OK	-1.14e-013	OK	OK	OK
leo1	OK	7.45e-008	OK	OK	OK	OK
leo2	-5.59e-008	OK	OK	OK	OK	OK
neos-476283	OK	OK	-7.11e-015	OK	OK	OK
neos-911880	OK	OK	OK	8.88e-016	OK	OK
neos15	-5.68e-014	1.14e-013	OK	5.68e-014	OK	OK
npmv07	-4.58e-005	2.38e-005	OK	OK	OK	OK
ran14x18.disj-8	OK	OK	-1.30e-009	OK	OK	OK
sing2	-4.55e-013	2.27e-013	OK	OK	OK	OK
swath	-9.95e-014	OK	OK	OK	OK	OK

Gurobi						
Name	**EC**		**GC**	**LC**	**lb**	**ub**
aflow40b	-3.49e-011	6.10e-011	OK	OK	OK	OK
b2c1s1	-1.45e-012	1.45e-012	OK	9.09e-013	OK	OK
beasleyC3	-2.16e-012	2.41e-011	OK	OK	OK	OK
berlin-5-8-0	OK	OK	-2.39e-008	OK	-2.39e-008	OK
bg512142	-2.91e-011	2.91e-011	-2.91e-011	2.91e-011	OK	OK
bienst2	-1.42e-014	OK	-1.42e-014	1.78e-015	OK	OK
binkar10-1	-4.00e-015	4.44e-015	OK	OK	OK	OK
blp-ar98	-1.41e-012	OK	OK	OK	OK	OK
blp-ic97	OK	9.95e-014	OK	OK	OK	OK
csched007	-4.97e-014	5.06e-014	OK	OK	OK	OK
csched008	-2.22e-014	2.22e-014	OK	OK	OK	OK
dano3mip	-1.42e-014	1.28e-013	-7.96e-013	1.42e-014	OK	OK
dg012142	-3.18e-012	3.18e-012	-9.55e-012	9.09e-013	OK	OK
g200x740i	-5.54e-011	2.18e-012	OK	OK	OK	OK
gmu-35-40	-4.55e-013	OK	OK	OK	OK	OK
gmu-35-50	-1.34e-011	OK	OK	OK	OK	OK
gmut-75-50	-2.18e-011	7.28e-012	OK	OK	OK	OK
gmut-77-40	-1.46e-011	7.28e-012	OK	OK	OK	OK
k16x240	-4.53e-010	5.02e-011	OK	OK	OK	OK
leo1	-5.96e-008	OK	OK	OK	OK	OK
leo2	OK	1.86e-008	OK	OK	OK	OK
lotsize	-1.09e-011	1.09e-011	OK	1.09e-011	OK	OK
n3705	-1.50e-011	1.42e-012	OK	1.42e-012	OK	OK
n370a	-1.86e-011	1.42e-012	OK	1.42e-012	OK	OK
neos-1171692	OK	OK	OK	3.73e-014	OK	OK
neos-1171737	OK	OK	OK	2.04e-014	OK	OK
neos-1311124	OK	OK	OK	1.07e-014	OK	OK
neos-1396125	-6.66e-016	1.33e-015	OK	OK	OK	OK
neos-1426635	OK	OK	OK	1.78e-014	OK	OK
neos-1426662	OK	OK	OK	1.78e-015	OK	OK
neos-1436709	OK	OK	OK	7.11e-015	OK	OK
neos-1440460	OK	OK	OK	1.42e-014	OK	OK
neos-1442119	OK	OK	OK	7.11e-015	OK	OK
neos-1442657	OK	OK	OK	7.11e-015	OK	OK
neos-476283	-1.99e-013	1.00e-012	OK	OK	-6.33e-010	OK
neos-506422	OK	OK	OK	2.84e-014	OK	OK
neos-520729	-1.09e-010	1.09e-010	OK	OK	-4.28e-008	OK

Gurobi						
Name	**EC**		**GC**	**LC**	**lb**	**ub**
neos-799711	-1.66e-008	9.21e-008	-1.75e-009	9.07e-008	-1.56e-007	OK
neos-885086	OK	OK	OK	3.20e-014	OK	OK
neos-911880	OK	OK	OK	2.66e-015	OK	OK
neos13	OK	OK	OK	5.55e-017	OK	OK
neos15	-8.53e-014	1.42e-014	OK	2.84e-014	OK	OK
net12	OK	OK	OK	1.82e-014	OK	OK
nobel-eu-DBE	-1.78e-014	1.78e-014	OK	1.86e-010	OK	OK
npmv07	-5.82e-011	2.18e-011	-1.14e-013	5.00e-006	-4.55e-013	-1.19e-009
ns1111636	-1.71e-011	5.37e-007	OK	OK	OK	OK
ns1208400	-4.35e-010	4.35e-010	-7.13e-010	5.56e-010	-8.55e-010	-8.87e-010
ns1644855	OK	OK	-3.55e-015	5.57e-011	OK	OK
ns1830653	OK	OK	-2.74e-013	OK	OK	OK
ns1904248	OK	OK	OK	8.43e-012	OK	OK
ns2081729	OK	OK	OK	1.78e-015	OK	OK
ns2122603	-7.11e-015	3.97e-014	-8.00e-008	1.12e-014	OK	OK
ns2124243	-4.44e-016	OK	OK	4.44e-016	OK	OK
ns2137859	OK	OK	OK	1.14e-013	OK	OK
ns930473	OK	OK	OK	2.33e-010	OK	OK
p100x588b	-2.91e-011	3.32e-011	OK	OK	OK	OK
p80x400b	-5.40e-011	2.15e-011	OK	OK	OK	OK
pg	OK	3.64e-012	OK	1.46e-011	OK	OK
qiu	OK	3.55e-015	OK	1.00e-006	OK	OK
r80x800	-3.73e-011	4.46e-011	OK	OK	OK	OK
ran14x18.disj-8	OK	OK	-1.30e-009	OK	OK	OK
ran14x18	-1.16e-011	8.53e-014	OK	8.53e-014	OK	OK
ran16x16	-1.03e-011	9.95e-014	OK	5.68e-014	OK	OK
rocII-4-11	OK	OK	-1.00e-0006	9.99e-016	OK	OK
rocII-7-11	OK	OK	-1.00e-006	1.03e-014	OK	OK
satellites1-25	OK	3.64e-012	-2.91e-011	7.28e-011	OK	OK
satellites2-60-fs	OK	3.64e-012	-5.82e-011	OK	OK	OK
satellites2-60	OK	3.64e-012	-5.82e-011	OK	OK	OK
satellites3-40-fs	OK	3.64e-012	-5.82e-011	OK	OK	OK
satellites3-40	OK	3.63e-012	-5.82e-011	OK	OK	OK
shipsched	-7.28e-012	2.73e-009	OK	6-69e-010	OK	OK
sing2	-2.273e-013	OK	-5.68e-014	OK	OK	OK
swath	-2.94e-013	OK	OK	OK	OK	OK
uc-case3	-2.27e-013	2.27e-013	-6.06e-014	5.68e-014	OK	OK
unitcal-7	-7.05e-012	1.66e-011	-4.26e-010	2.72e-010	OK	OK
usAbbrv.8.25-70	OK	OK	-1.57e-008	OK	-1.57e-008	OK

Überraschenderweise treten, unter Verwendung von Gurobi, also wesentlich häufiger Verletzungen der Nebenbedingungen auf, als bei der FP-Implementation in Matlab. Während die FP gerade in 12 Fällen eine geringfügige Übertretung der Nebenbedingungen liefert, tritt dieser Fall bedeutend häufiger mit Hilfe von Gurobi ein. So sind im Grunde unter den 128 durch Gurobi gefundenen Punkten nur 52 echt zulässige Punkte. Die FP hingegen liefert in weitaus mehr Fällen einen echt zulässigen Punkt. Aus dieser Sicht ist also die FP seiner Konkurrenz überlegen. Abschließend wollen wir letztere Ergebnisse in folgender Tabelle zusammenfassen:

Name	gefunden	echt zulässig	fast zulässig
FP Matlab	93	81	12
Gurobi	128	52	76

3 MINLP

Wie in Kallrath (2002) beschrieben wird, gehören MINLPs zu den schwersten Optimierungsproblemen der heutigen Zeit. Sie kombinieren sozusagen die Problematik der Ganzzahligkeit mit derjenigen der nichtliniearen Optimierung. Ein intensives Studium dieser Probleme würde das Ausmaß dieser Diplomarbeit sprengen. Dennoch wollen wir, der Vollständigkeit wegen, einen kurzen Überblick über das Thema geben.

Wir betrachten nun also Probleme der Form:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } g(x) \leq 0, \end{aligned} \tag{3.1}$$

wobei $f : X \times D \to \mathbb{R}$ nicht linear sei, $g : X \times D \to \mathbb{R}^k$, $x \in X \times D$ mit $X \subseteq \mathbb{R}^m$ und $D \subseteq \mathbb{Z}^n$. Dabei wollen wir sowohl die Zielfunktion f als auch die Nebenbedingungen g als konvex und stetig differenzierbar annehmen.

Trifft man diese Voraussetzungen nicht, so hat man es mit nicht-konvexen MINLPs zu tun. In diesem Fall kann oft nicht sichergestellt werden, dass eine Lösung der Relaxation global ist, was die Sache nocheinmal erschwert. In der globalen Optimierung wurden Techniken entwickelt, um dieses Problem zu lösen. Wir werden uns hier jedoch nicht weiter damit beschäftigen.

Im Folgenden wollen wir einige Verfahren zur Lösung von konvexen MINLPs vorstellen.

3.1 Methoden zur Lösung von MINLP

3.1.1 Branch & Bound für MINLPs

Nach Gupta & Ravindran (1985) funktioniert das Branch & Bound-Verfahren für konvexe MINLPs nach demselben Prinzip wie jenes, das für lineare Probleme verwendet wird. Wieder wird zuerst die Relaxation des ursprünglichen Problems (3.1) gelöst, also:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } g(x) \leq 0, \end{aligned} \tag{3.2}$$

wobei $f : \mathbb{R}^{m \times n} \to \mathbb{R}$, $h : \mathbb{R}^{m \times n} \to \mathbb{R}^k$, $x \in \mathbb{R}^{m \times n}$. Nun müssen wir jedoch, statt des Simplexverfahrens, eine Methode zur Lösung von NLPs verwenden. Wie wir bereits im

ersten Kapitel festgestellt haben, erhalten wir dadurch im Fall von konvexen Problemen jedenfalls ein globales Minimum, falls ein solches existiert. Durch Hinzufügen der Restriktionen $x_j \leq \lfloor \bar{x}_j \rfloor$ bzw. $x_j \geq \lfloor \bar{x}_j \rfloor + 1$ erhalten wir nun wieder die Unterprobleme, die die Knoten des Entscheidungsbaums darstellen:

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } h(x) \leq 0 \\ &\qquad x_j \leq \lfloor \bar{x}_j \rfloor, \end{aligned} \tag{3.3}$$

und

$$\begin{aligned} &\min f(x) \\ &\text{s.t. } h(x) \leq 0 \\ &\qquad x_j \geq \lfloor \bar{x}_j \rfloor + 1, \end{aligned} \tag{3.4}$$

die wieder mit einer geeigneten NLP-Methode gelöst werden. Der restliche Verlauf des Algorithmus, ist völlig analog zum linearen Fall. Das gilt auch für die Schranken, so bilden die Zielfunktionswerte, die durch Lösungen der Relaxationen gewonnen werden die unteren und diejenigen Werte, die erhalten werden, wenn die Ganzzahligkeitsbedingungen erfüllt sind, die oberen Schranken.

3.1.2 Äußere Approximation

Nach Kallrath (2002) handelt es sich bei der Äußeren Approximation um ein Verfahren zur Lösung von beschränkten MINLPs, welches mit der abwechselnden Lösung kontinuierlicher nichtlinearer Teilprobleme und linearer MIP-Masterprobleme arbeitet. Der zulässige Bereich des ursprünglichen Problems wird hierbei als Durchschnitt einer Sammlung einfach aufgebauter Mengen, wie etwa Polyeder, aufgefasst. Für die Beschreibung des Verfahrens wollen wir Optimierungsprobleme folgender Form betrachten:

$$\begin{aligned} &\min c^T y + f(x) \\ &\text{s.t. } g(x) + By \leq 0, \end{aligned} \tag{3.5}$$

wobei $x \in X \subseteq \mathbb{R}^n$, $y \in U \subseteq \mathbb{R}^m$. Die Mengen X und U sollen dabei folgende Polyeder definieren: $X := \{x \in \mathbb{R}^n \mid A_1 x \leq a_1\}$ und $U := \{y \in \mathbb{Z}^m \mid A_2 y \leq a_2\}$. Wieder müssen wir Regularitätsbedingungen an die Funktionen stellen. Diese sollen auch hier wieder

stetig differenzierbar und konvex sein. Wie man an der Formulierung des Problems sehen kann, wird vorausgesetzt, dass die ganzzahligen Variablen nur in den linearen Termen des Problems enthalten sind.

Sei nun $\underline{f} := \min\{f(x) \mid x \in X\}$ und $\overline{f} := \max\{f(x) \mid x \in X\}$ und $\mu \in [\underline{f}, \overline{f}]$. Dann können wir das ursprüngliche Problem folgendermaßen umformulieren:

$$\begin{aligned} &\min c^T y + \mu \\ &\text{s.t. } f(x) - \mu \leq 0 \\ &\qquad g(x) + By \leq 0. \end{aligned} \tag{3.6}$$

Somit haben wir das MINLP in ein Problem mit linearer Zielfunktion umgewandelt. Den zulässigen Bereich dieses Problems wollen wir mit S_0 bezeichnen. Die Nebenbedingungen sollen nun unter anderem folgende Bedingung erfüllen:

$$\forall y \in U \cap V \; \exists x \in X : \; g(x) + By < 0,$$

wobei $V := \{y \mid g(x) + By \leq 0 \text{ für zumindest ein } x \in X\}$. Solche Bedingungen werden Slaterbedingungen genannt. Der zulässige Bereich S_0 lässt sich nun als Durchschnitt unendlich vieler Halbräume darstellen:

$$\begin{aligned} 0 \geq & f(x) - \mu \geq f(x^i) + \nabla f(x^i)^T (x - x^i) - \mu \\ 0 \geq & g(x) + By \geq g(x^i) + \nabla g(x^i)^T (x - x^i) + By, \end{aligned}$$

für alle $x^i \in X$, $x \in X$, $\mu \in [\underline{f}, \overline{f}]$ und $y \in U$. Diese Beobachtung führt uns nun zu folgender Darstellung von (3.6):

$$\begin{aligned} &\min c^T y + \mu \\ &\text{s.t. } f(x^i) + \nabla f(x^i)^T (x - x^i) - \mu \leq 0 \\ &\qquad g(x^i) + \nabla g(x^i)^T (x - x^i) + By \leq 0. \end{aligned} \tag{3.7}$$

für alle $x^i \in X$, $x \in X$, $\mu \in [\underline{f}, \overline{f}]$ und $y \in U$. Dadurch haben wir erreicht (3.6) auf ein lineares semi-infinites gemischt-ganzzahliges Problem zu transferieren. Dieses hat allerdings unendlich viele Nebenbedingungen, was Formulierung (3.7) im Moment noch unbrauchbar macht. Um dieses Problem zu umgehen, betrachten wir die Projektion von (3.5) auf y:

$$\min_{y^i \in U \cap V} \{\inf_{x \in X} \{c^T y^i + f(x) \mid g(x) + By^i\}\} =: \min_{y^i \in U \cap V} z(y^i).$$

Nach Duran & Grossmann (1986) ist nun $U \cap V$ eine diskrete und endliche Menge, womit nur noch endlich viele Punkte für die Optimierung von Belang sind. Sei nun ein $y^i \in U \cap V$ gegeben, so ist das durch die Projektion gegebene Infimum auch optimale Lösung von (3.5) bei fixiertem y^i. Darüber hinaus gilt:

$$\begin{aligned} z(y^i) &= c^T y^i + \min f(x) \\ \text{s.t. } & g(x) + By^i \leq 0, \end{aligned} \tag{3.8}$$

für alle $y^i \in U \cap V$ und $x \in X$. Sei nun

$$T := \{i \mid x^i \text{ ist gobales Minimum von } z(y^i),\ y^i \in U \cap V\},$$

so kann schließlich das Masterproblem formuliert werden:

$$\begin{aligned} z &= \min c^T y + \mu \\ \text{s.t. } & f(x^i) + \nabla f(x^i)^T (x - x^i) - \mu \leq 0 \\ & g(x^i) + \nabla g(x^i)^T (x - x^i) + By \leq 0, \end{aligned} \tag{3.9}$$

für alle $i \in T$, $x \in X$, $\mu \in [\underline{f}, \overline{f}]$ und $y \in U$. Um zu vermeiden, für alle y^i Problem (3.8) lösen zu müssen, wird nun mit passenden Relaxationen M^k des Masterproblems weitergearbeitet, die wie folgt definiert sind:

$$\begin{aligned} z^k &= \min c^T y + \mu \\ \text{s.t. } & (x, y) \in \Omega^k, \qquad (M^k) \end{aligned}$$

für $x \in X$, $\mu \in [\underline{f}, \overline{f}]$ und $y \in U$. Dabei ist Ω^k für $T^k \subseteq T$ mit $T^k := \{i \in T \mid i = 1, \ldots, k\}$ definiert durch:

$$\begin{aligned} \Omega^k := \{(x, y) \mid & f(x^i) + \nabla f(x^i)^T (x - x^i) - \mu \leq 0, \forall i \in T^k, \\ & g(x^i) + \nabla g(x^i)^T (x - x^i) + By \leq 0, \forall i \in T^k\}. \end{aligned}$$

Durch lösen des Problems M^k, das all jene Nebenbedingungen des Masterproblems, die durch $i \in T \backslash T^k$ gegeben sind, vernachlässigt, kann man nun den Zielfunktionswert bestimmen. Diesen wollen wir mit z^k bezeichnen. Die Folge $\underline{z} = (z^k)_k$ bildet dann eine monoton wachsende Folge an unteren Schranken. Des weiteren werden durch $\overline{z} := (z(y^i))_i$ die oberen Schranken gewonnen.

Nach all diesen Beobachtungen können wir nun den genauen Ablauf der Äußeren

Approximation beschreiben:

1. Zu Beginn wird $\Omega^0 := \mathbb{R}^n \times \mathbb{R}^m$, $\underline{z} = -\infty$, $\overline{z} = \infty$ und die Anzahl der Iterationen $i := 1$ gesetzt. Weiters wählen wir $y^1 \in U$ bzw. $y^1 \in U \cap V$.

2. Nun wird das nichtlineare Teilproblem (3.8) gelöst. Existiert ein globales Minimum für dieses Problem und gilt $z(y^i) \leq \overline{z}$, so wird $\overline{z} := z(y^i)$, $y^* := y^i$ und $x^* := x^i$ gesetzt. Bevor wir mit Schritt 3 fortfahren wird darüber hinaus Ω^i folgendermaßen aktualisiert:

$$\Omega^i := \Omega^i - 1 \cap C(x^i),$$

wobei

$$\begin{aligned} C(x^i) :=& \{(x,y) \mid f(x^i) + \nabla f(x^i)^T(x - x^i) - \mu \leq 0, \\ & g(x^i) + \nabla g(x^i)^T(x - x^i) + By \leq 0, \mu \in \mathbb{R}\}. \end{aligned}$$

Ist Problem (3.8) unzulässig, wird y^i durch Setzen von $\Omega^i := \Omega^i - 1 \cap C(x^i)$ abgeschnitten, um zu Schritt 3 überzugehen.

3. An dieser Stelle wird Problem M^i gelöst, wobei folgende zwei Fälle eintreten können:

   - Existiert für M^i kein zulässiger Punkt, so sind wir am Ende des Algorithmus angelangt. In diesem Fall entspricht (x^*, y^*) der optimalen Lösung von (3.5).
   - Gibt es dagegen einen zulässigen Punkt (x, y, μ) für M^i, so wird $\underline{z} = z_i$, $y^i := y$ und $i := i + 1$ gesetzt, um wieder zu Schritt 2 zurückzukehren.

3.1.3 Feasibility Pump für MINLPs

Wie in der Arbeit von Bonami et al. (2009) gezeigt wurde, kann die FP auch zur Findung zulässiger Punkte für MINLPs verwendet werden. Wir betrachten hier Probleme folgender Gestalt:

$$\begin{aligned} &\min f(x,y) \\ &\text{s.t. } g(x,y) \leq 0, \end{aligned} \tag{3.10}$$

für $x \in \mathbb{Z}^n$ und $y \in \mathbb{R}^m$. Dabei seien f und g stetig differenzierbare und konvexe Funktionen. Der Grundgedanke liegt wieder darin zwei Folgen von Punkten $(\overline{x}^0, \overline{y}^0), \ldots, (\overline{x}^k, \overline{y}^k)$

und $(\hat{x}^1, \hat{y}^1), \ldots, (\hat{x}^{k+1}, \hat{y}^{k+1})$ zu erzeugen, wobei die $(\overline{x}^i, \overline{y}^i)$ die Nebenbedingungen des ursprünglichen Problems erfüllen, jedoch nicht unbedingt die Ganzzahligkeitsbedingungen, während $(\hat{x}^i, \hat{y}^i)$ den Ganzzahligkeitsbedingungen genügt, aber nicht zwingend den Nebenbedingungen. Zur Generierung der Folge $(\hat{x}^1, \hat{y}^1), \ldots, (\hat{x}^{k+1}, \hat{y}^{k+1})$ wird mit äußerer Approximation gearbeitet, um die nichtlinearen Nebenbedingungen durch lineare zu approximieren. Haben wir eine Menge $\{(\overline{x}^k, \overline{y}^k) \mid k = 1, \ldots, i-1\}$ an zulässigen Punkten der Relaxation gegeben, so kann der zulässige Bereich des MINLP folgendermaßen geschrieben werden:

$$g(\overline{x}^k, \overline{y}^k) + J_g(\overline{x}^k, \overline{y}^k)((x, y) - (\overline{x}^k, \overline{y}^k)) \leq b, \ \forall k = 0, \ldots, i-1,$$

mit $x \in \mathbb{Z}^n$ und $y \in \mathbb{R}^m$.

Zu Beginn der FP für MINLP wird nun $(\overline{x}^0, \overline{y}^0)$ durch Berechnung der optimalen Lösung der Relaxation von (3.10) ermittelt. Es handelt sich nun um ein nichtlineares Problem, weshalb dafür auch eine NLP Methode verwendet werden muss. Für $i \geq 1$ erhält man dann den Punkt $(\hat{x}^i, \hat{y}^i)$ durch Lösung von:

$$\begin{aligned} &\min \|x - \overline{x}^{i-1}\|_1 \\ &\text{s.t. } g(\overline{x}^k, \overline{y}^k) + J_g(\overline{x}^k, \overline{y}^k)((x, y) - (\overline{x}^k, \overline{y}^k)) \leq b, \end{aligned} \tag{3.11}$$

für alle $k = 0, \ldots, i-1$ mit $x \in \mathbb{Z}^n$ und $y \in \mathbb{R}^m$. Daraufhin wird das nichtlineare Problem

$$\begin{aligned} &\min \|x - \overline{x}^i\|_2 \\ &\text{s.t. } g(x, y) \leq 0, \end{aligned} \tag{3.12}$$

mit $x \in \mathbb{R}^n$ und $y \in \mathbb{R}^m$, gelöst, um $(\overline{x}^i, \overline{y}^i)$ zu gewinnen. Problem (3.11) und (3.12) werden nun so lange abwechselnd gelöst, bis ein zulässiger Punkt für (3.10) gefunden wurde oder (3.11) unzulässig wird.

4 Fazit

Wir haben nun einige Methoden zur heuristischen Lösung von MIPs kennengelernt und die FP von Fischetti et al. (2005) in Matlab implementiert. Diese Verfahren erreichen immer mehr Bedeutung, da, im Speziellen bei sehr große Problemen, eine beweisbar optimale Lösung oft nicht mit vertretbarem Aufwand gefunden werden kann. Ist also schnell eine Entscheidung zu fällen, gibt man sich oft mit heuristischen Lösungen zufrieden, ohne Optimalität zu fordern.

Unsere Version der FP hat sich nun als durchaus effizienter Algorithmus herausgestellt. In der Qualität der gefundenen Punkte, ist er der professionellen Konkurrenz sogar voraus. Zum ersten ist die Abweichung zum optimalen Wert im Schnitt geringer, und zum zweiten treten in wesentlich weniger Fällen geringfügige Verletzungen der Nebenbedingungen auf. Wird also die genaue Einhaltung der Nebenbedingungen gefordert, ist die Verwendung unseres FP jedenfalls sinnvoller als die von Gurobi, da letztere Methode in 59,4 % keinen echt zulässigen Punkt gefunden hat. Selbiges passierte mit Hilfe der FP in nur 12,9 % der Fälle. Verbesserungswürdig bei unserer Implementation wären jedoch sowohl die Geschwindigkeit, als auch die Anzahl der gefundenen heuristischen Lösungen. Ersteres könnte man, wie in Fischetti et al. (2005) beschrieben, durch approximative Lösung der LP-Relaxationen erreichen. Dabei wird jedoch mitunter die Qualität der gefundenen Punkte beeinträchtigt. Letzteres dürfte sich verbessern lassen, indem man bei Feststellung einer Zyklusbildung einen zusätzlichen Störfaktor hinzufügt. Weiters könnte man den Algorithmus so aufbauen, dass er nicht nach dem ersten gefundenen Punkt stoppt, um qualitativ bessere Punkte zu erzeugen.

Literatur

Emile HL Aarts, Jan HM Korst, and Peter JM Van Laarhoven. Simulated annealing. *Local search in combinatorial optimization*, pp. 91–120, 1997.

T. Achterberg, T. Koch, and A. Martin. The mixed integer programming library: MIPLIB 2010. http://miplib.zib.de.

C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007.

P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119(2):331–352, 2009.

Brian Borchers. READMPS MATLAB / Octave Code to Read an MPS file. http://infohost.nmt.edu/~borchers/readmps.html.

Fair Isaac Corporation. FICO Xpress Optimization Suite. http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx.

E. Danna, E. Rothberg, and C.L. Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2005.

Reinhard Diestel. *Graphentheorie*. Springer DE, 2006.

Marco A Duran and Ignacio E Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical programming*, 36(3):307–339, 1986.

M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

Fred Glover. Tabu search-part I. *ORSA Journal on computing*, 1(3):190–206, 1989.

R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

Omprakash K Gupta and A Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546, 1985.

Inc. Gurobi Optimization. Gurobi optimization. http://www.gurobi.com/, 2013.

KL Hoffman and Manfred Padberg. Traveling salesman problem. *Encyclopedia of Operations Research and Management Science, 2nd edn. Kluwer, Boston*, pp. 849–853, 2000.

IBM. IBM ILOG CPLEX Optimization Studio. http://www-142.ibm.com/software/products/de/de/ibmilogcpleoptistud/.

J. Kallrath. *Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis: mit Fallstudien aus Chemie, Energiewirtschaft, Metallgewerbe, Produktion und Logistik*. Vieweg, 2002.

Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

Scott Kirkpatrick, MP Vecchi, et al. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.

A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pp. 497–520, 1960.

D. Li and X. Sun. *Nonlinear integer programming*. Springer Science + Business Media, LLC, 2006.

George L Nemhauser and Laurence A Wolsey. *Integer and combinatorial optimization*, vol. 18. Wiley New York, 1988.

Arnold Neumaier and Hermann Schichl. Optimization - theory and algorithms, buchentwurf, 2012.

M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987.

H.M. Salkin and C.A. De Kluyver. The knapsack problem: a survey. *Naval Research Logistics Quarterly*, 22(1):127–144, 1975.

S. Walukiewicz. *Integer programming*, vol. 46. PWN-Polish Scientific Publishers, 1991.

Lebenslauf

Persönliche Daten

Name:	Bettina Hofbauer
Geburtsort:	Wien
Nationalität:	Österreich

Ausbildung

1990 - 1994	Volksschule, Hoefftgasse, Wien
1994 - 1998	GRG 3, Hagenmüllergasse, Wien
1998 - 2003	HLTW 13, Bergheidengasse, Wien
2005 - lfd	Diplomstudium der Mathematik, Universität Wien
	Schwerpunkt: Angewandte Mathematik und Scientific Computing
07/09 -11/09	Auslandssemester, Massey University, Albany, Neuseeland

Berufserfahrung

07/03 - 05/06	Rezeptionistin, Österreichisches Verkehrsbüro AG, Wien
06/06 - 06/08	Rezeptionistin, V.A. Hotel GmbH, Courtyard by Marriott Wien Schönbrunn
07/08 - 08/08	Praktikum, UniCredit Bank Austria AT, Wien
	Abteilung: Risk Management und Risk Analysis
09/08 - 06/09	Rezeptionistin, V.A. Hotel GmbH, Courtyard by Marriott Wien Schönbrunn
12/09 - lfd	Rezeptionistin, V.A. Hotel GmbH, Courtyard by Marriott Wien Schönbrunn