Gröbnerbasen und ganzzahliges Programmieren

Nils Przigoda

08. Mai 2008

FB3 - Universität Bremen

Seminararbeit im Rahmen der WE AlZAGK

Betreuer: Michael Hortmann

Inhaltsverzeichnis

1 Vorwort

Ziel meines Vortrags war es darzustellen, was das ganzzahlige Programmierproblem, engl.“Integer Programming” kurz “IP”, ist und warum man dieses auf das “Extended Integer Programming” erweitern muss, um dann mit Hilfe von Gröbnerbasen das IP zu lösen. Verdeutlicht werden soll dies an einigen 2 dimensionalen Polytopen und einer Realisierung der vorgestellten Lösungsalgorithmen in Java.

Die Vorlage dieses Vortrags ist “Chapter 7: Gröbner Bases and Integer Programming” (Günther M. Ziegler) des Buches “Some Tapas of Computer Algebra” von A.M. Cohen, H. Cuypers und H. Sterk erschienen im Springer Verlag.

Es sei auch noch gesagt, dass das ganzzahlige Programmieren ein wichtiges Thema innerhalb der Optimierung ist.

2 Polytope

Ein Polyeder P ist ein Schnitt von Halbebenen, der Art: $P := \{x \in \mathbb{R}^n | Ax \leq b\}$. Dabei seien $A \in \mathbb{R}^{m \times n}$ eine Matrix und $x \in \mathbb{R}^n$ und $b \in \mathbb{R}^m$ Vektoren. Wenn das Polyeder beschränkt ist, nennen wir Polytop.

Außerdem sei nun noch eine lineare Kostenfunction $cost : \mathbb{R}^n \to \mathbb{R},\ x \mapsto c^T x$ mit $c \in \mathbb{R}^n$ gegeben. Das lineare Optimierungsproblem ist nun, ein $x(\in P)$ zu finden, sodass $c^T x$ minimal ist. Dabei nennen wir die Punkte in P zulässige Punkte. Wenn P ein nichtleeres Polytop ist, so ist auch die Existenz einer optimalen Lösung x_0 garantiert. Wenn außerdem die Kostenfunktion *cost* in allgemeiner Lage ist, d.h. die Kostenfunktion *cost* für einen festen Wert keine parallele Halbebene zu einer der Seite des Polytops bildet, so ist die optimale Lösung sogar eindeutig. Das lineare Programmierproblem soll uns aber nicht weiter interessieren, da es mittlerweile mehrere Lösungverfahren gibt, wie zum Beispiel “CPLEX” und der “Simplex-Algorhitmus”. Auf diese wird im Weiteren auch nicht eingegangen.

Man betrachte nun ein Polytop $P := \{x \in \mathbb{R}^n |\ Ax = b,\ x \geq 0\}$, wobei A und b nur ganzzahlige Elemente haben, also $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$. Dies ist sinnvoll, weil bei

ganzzahligen Optimierungsproblemen keine rationalen oder reelen Zahlen auftauchen. Beim IP soll ein ganzzahliger Vektor, der $c^T x$ optimiert, gefunden werden. Die zulässigen Punkte sind dann $P \cap \mathbb{Z}^n$.

Aus der Polytop Theory wissen wir, dass wir ein Polytop wird nicht nur durch ein Gleichungssystem der Form $Ax = b$ bzw. $Ax \leq b$ sondern auch als konvexe Hülle seiner ganzzahligen Koordinaten darstellen können. In diesem Fall könnte man die konvexe Hülle mit

$$P_I := conv\{x \in \mathbb{Z}^n |\ Ax = b,\ x \geq 0\}$$

beschreiben. Die Menge P_I besteht aus endlich vielen Punkten. Allerdings sind die Gleichungen und/oder Ungleichungen zu der Menge P_I nicht bekannt. Wenn Sie bekannt wären könnte man das Problem mit Hilfe der Methoden der linearen Optimierung lösen. Man muss nun also eine andere Möglichkeit finden, dass IP zu lösen.

3 IP, EIP & Algorithmen

Im Folgenden seien $A \in \mathbb{N}^{m \times n}$ fest und $b \in \mathbb{N}^m$ beliebig. Weiter sei $c \neq \mathbf{0}$ unser Vektor für die Kostenfunktion *cost*. Unser Polyeder kann nun als

$$P_I(b) := conv\{x \in \mathbb{R}^n | Ax = b\}$$

geschrieben werden.

Die Familie der linearen Optimierungsprobleme mit festem A und c können wir als $LP_{A,c}$ schreiben. Die Lösung in Abhängigkeit für ein b können wir dann als

$$LP(b) := min\{c^T x |\ x \in \mathbb{N},\ Ax = b,\ x \geq \mathbf{0}\}$$

schreiben. Die Menge der zulässigen Lösungen soll beschränkt sein, diese Einschränkung ist bereits für $c \geq 0$ erfüllt. Daraus folgt sofort, dass die Lösung eines ganzzahligen Optimierungsproblemen

$$IP(b) := min\{c^T x |\ x \in \mathbb{N},\ Ax = b,\ x \in \mathbb{N}\}$$

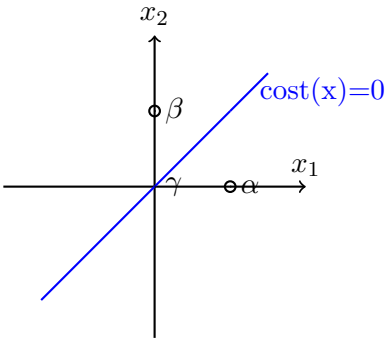
ebenfalls von b abhängt. Die Familie der ganzzahligen Optimierungsprobleme wird als $IP_{A,c}$ notiert.

Im Folgenden benötigen wir nun eine allgemeinere Funktion als *cost*. Wir wählen dafür eine beliebige Monomordung $\prec$, z.B. die lexikographische Ordnung, aus und definieren für $x, y \in \mathbb{Z}^n$ und $c \in \mathbb{N}^n$:

$$x \prec_c y \Leftrightarrow \begin{cases} c^T x < c^T y, & \text{oder} \\ c^T x = c^T y & \text{und } x \prec y \end{cases}$$

Beispiel. Seien $c = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$, $x = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $y = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ und $z = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, dann folgt:

$$y \prec_c x \prec_c z \Leftarrow \begin{cases} c^T x = 0 \\ c^T y = -1 \\ c^T z = -1 \end{cases}$$



Dieses Beispiel zeigt, dass wenn $c \geq 0$ nicht erfüllt ist, wir durch die Kosten keine Beschränkung auf $\mathbb{N}^2$ haben.

Definition 3.1. Ein Punkt $a \in \mathbb{N}^n$ heiße nicht-optimal, wenn einen Punkt $b \in \mathbb{N}^n$ mit der $Aa = Abt$ und $b \prec_c a$ existiert.

Bemerkung. Es sind alle alle Punkte $a \in \mathbb{N}^n$ nicht-optimal, die nicht gleichzeitig Element die Lösung in $IP(Aa)$ sind.

Mit den obigen Überlegen können wir auch sagen, dass $IP(b)$ zulässig ist, wenn $IP(b) \neq \emptyset$ gilt. Die Lösung bezüglich unserer Monomordnung $\prec_c$ ist dann auch eindeutig. Wenn $IP(b)$ jedoch die leere Menge ist, so ist $IP(b)$ nicht zulässig.

Als Hilfe benötigen wir gleich das Gordon-Dickson-Lemma:

Lemma 3.2. ***(Gordon-Dickson-Lemma, [AlgebraSkript])*** *Es sei $A \subset \mathbb{Z}^n_{\leq 0}$. Dann gilt:*

Für jedes Monomideal $I = \langle x^\alpha | \alpha \in A \rangle \subset \mathbb{K}[x_1, \ldots, x_n]$ existieren $\alpha_1, \ldots, \alpha_t \in A$, so dass $I = \langle x^{\alpha_1}, \ldots, x^{\alpha_t} \rangle$. Insbesondere ist also I endlich erzeugt.

Lemma 3.3. ***([Thomas])*** *Es gibt eine eindeutige, endliche und minimale Menge von Vektoren $m_1, \ldots, m_t \in \mathbb{N}^n$, sodass die Menge aller nicht-optimalen Punkte S mit der Ordnung $\prec_c$ eine Teilmenge $\mathbb{N}^n$ mit der Form*

$$S = \bigcup_{i=1}^{t} (a_i + \mathbb{N}^n)$$

ist. Die Menge $m_1, \ldots, m_t \in \mathbb{N}^n$ heißt Menge der minimalen, nicht-optimalen Punkt und wird als $M := \{m_1, \ldots, M_t\}$ notiert.

Beweis. Zuerst wollen wir zeigen, dass $S \subseteq \mathbb{N}^n$ ist. Sei $\alpha \in S$, dann ist α ein nicht-optimaler Punkt für $IP(A\alpha)$. Es sei nun β o.B.d.A der optimale Punkt in $IP(A\alpha)$. Dann folgt für ein beliebiges $\gamma \in \mathbb{N}^n$:

1. $A(\alpha + \gamma) = A(\beta + \gamma)$, da $A\alpha = A\beta$
2. $(\alpha + \gamma),\ (\beta + \gamma) \in \mathbb{N}^n$, wenn $\alpha,\ \beta,\ \gamma \in \mathbb{N}^n$
3. $\beta \prec_c \alpha \Rightarrow (\beta + \gamma) \prec_c (\alpha + \gamma)$

1.-3. beinhalten, dass für alle nicht-optimalen Punkte $\alpha \in S$, auch $(\alpha + \gamma)$ ein nicht-optimaler Punkt ist. Also ist die Menge $\alpha + \mathbb{N}^n$ eine Teilmenge vom $\mathbb{N}^n$ aus nicht-optimalen Punkten. Aus dem Gordon-Dickson-Lemma folgt nun, dass eine Menge $a_1, \ldots, a_t \in S$ existiert, die eindeutig, endlich und minmal ist, sowie dass $S = \bigcup_{i=1}^{t} (m_i + \mathbb{N}^n)$ □

Definition 3.4. Eine Teilmenge $T \subset \mathbb{Z}^n$ heiße *Testmenge*, wenn

- $\forall\ t \in T :\ At = \mathbf{0}$

- $\forall\, t \in T:\ \mathbf{0} \prec_c t$ und
- für jeden nicht optimalen Punkt $x \in \mathbb{N}^n,\ \exists\, t \in T:\ x - t \geq \mathbf{0}$

Bemerkung. Aus der dritten Bedingung folgt auch die Bedingung, dass $T = \emptyset$ ist.

Wenn man eine solche Testmenge T hat, kann man sich leicht überlegen, dass für einen nicht-optimalen Punkt $x \in \mathbb{N}^n$ und ein Element $t \in T$ $Ax = A(x - t)$ gilt. Außerdem gilt $x - t \prec_c x$. Insgesamt heißt dies wir können jeden nicht-optimalen Punkt optimieren, wenn wir eine Testmenge T haben. Der folgende Algorithmus verdeutlich dies nocheinmal.

Algorithmus 3.5. *Der folgende Algorithmus kann die Familie von ganzzahligen Programmier-Problemen $IP_{A,c}$ lösen:*

Eingabe	*eine Testmenge T für $IP_{A,c}$ und mind. ein $x \in \mathbb{N}^n$: Ax zulässig für IP(Ax)*
Wiederhole	*finde ein $t \in T$, sodass $x - t \geq \mathbf{0}$*
	und setze $x = x - t$
Bis	*x optimal ist, d.h. kein $\nexists\, t \in T:\ x - t \geq \mathbf{0}$*

Eine Realisierung dieses Algorithmus in Java könnte so aussehen:

```
1  import algebra.Objects.MatrixInt.MatrixInt;
2  import algebra.Objects.VektorInt.VektorInt;
```

Der Befehl “import” importiert, die von mir geschriebenen Objekte “MatrixInt” und “VektorInt”, welche im Anhang dokumentiert zu finden sind.

```
11 public class Alg3_5 {

17         public static void main(String[] args) {
18                 VektorInt b = new VektorInt (new int[] {10,15});
19                 VektorInt c = new VektorInt (new int[] {1,3,14,17});
20                 MatrixInt A = new MatrixInt (new int[][] {{1,1,1,1},{0,1,2,3}});
21                 // Testausgaben b.printZ(); c.printZ(); A.print();
```

```
22
23                  VektorInt t1 = new VektorInt (new int[] {0,-1,2,-1});
24                  VektorInt t2 = new VektorInt (new int[] {1,-1,-1,1});
25                  VektorInt t3 = new VektorInt (new int[] {1,-2,1,0});
26
27                  Liste Tc = new Liste(b,c,A);
```

Nachdem Beginn der Main-Methode werden zunächst einmal die Vektoren b und c sowie die Matrix A für unser $IP_{A,c}$ initialisiert. Danach wird noch die Testmenge initialisiert. Sie besteht aus den drei Vektoren $t1, t2$ und $t3$. Mit "Liste Tc" wird eine Liste erstellt, in der im weiteren Verlauf die zulässigen Punkte sortiert gespeichert werden. Ich habe hier eine Liste gewählt, damit man den Weg der einzelnen Optimierungsschritte nochmal nachvollziehen kann. Wie dies auch in Abbildung 3.2 zu sehen ist.

```
29                  Tc.rlInsSorted(new VektorInt (new int[] {5,0,0,5}));
30                  Tc.rlPrint();
31
32                  int oldLen = 0, newLen = 0;
33                  do {
34                          oldLen = Gc.rlLen();
35                          Object tmp = Gc.rlHead();
36                          Tc.rlInsSorted(((VektorInt) tmp).Sub(t1));
37                          Tc.rlInsSorted(((VektorInt) tmp).Sub(t2));
38                          Tc.rlInsSorted(((VektorInt) tmp).Sub(t3));
39                          newLen = Gc.rlLen();
40                  } while (oldLen != newLen);
41                  Tc.rlPrint();
42
43          }
44  }
```

Es wird zunächst ein zulässiger Vektor der Liste hinzugefügt, danach folgt eine Ausgabe. Bevor dann die "int-Variablen" oldLen für die alte Länge der Liste und newLen für die Liste der Länge nach einem Durchgang mit dem Wert null initialisiert werden. Der Algorithmus in Pseudo-Code ist jetzt lediglich die *do - while*-Schleife. Hier wird zunächst oldLen der Länge vor einem Durchgang zugewiesen. Danach wird das erste Elemente der Liste genommen und die drei Vektoren der Testmenge subtrahiert und dann wieder der Liste sortiert hinzugefügt. Falls der Vektor in einem Eintrag eine negative Komponente oder nicht zulässig ist, wird dies in der Funktion "rlInsSorted" selbst abgefangen.

Nachdem Hinzufügen wird "newLen" nun die aktuelle Länge der Liste zugewiesen. Wenn nun *oldLen = newLen*, wird die Schleife abgebrochen, da das erste Element der Liste nicht weiter optimiert werden konnte. Anderfalls wurde im letzten Schritt eine Optimierung vorgenommen und es wird nun versucht den neuen Wert noch weiter zu optimieren.

Satz 3.6. *Die eindeutig bestimmte minimale Testmenge T für die Familie $IP_{A,c}$ ist gegeben durch*

$$T := \left\{ m_i - m_i^* \;\middle|\; \begin{array}{c} m_i \in M, \quad \textit{die Menge der minimal, nichtpoptimalen Punkte }, \} \\ m_i^* \textit{ optimal für } IP(Am_i) \textit{ mit } Am_i = Am_i^* \end{array} \right\}$$

Beweis. Man zeige zuerst, dass T eine Testmenge ist und danach, dass T minimal ist.

Testmenge:

- $\forall\ t \in T:\ At = \mathbf{0}$ gilt, weil m_i und m_i^* so gewählt waren, dass $Am_i = Am_i^*$, also auch $Am_i - Am_i^* = A(m_i - m_i^*) = \mathbf{0}$.
- $\forall\ t \in T:\ \mathbf{0} \prec_c t$ gilt, weil $\mathbf{0} \prec_c m_i^* \prec_c m_i^*$ und somit ist auch $\mathbf{0} \prec_c (m_i - m_i^*)$.
- für jeden nicht optimalen Punkt $x \in \mathbb{N}^n$, $\exists\ t \in T:\ x - t \geq \mathbf{0}$ gilt, weil für ein beliebiges x als Produkt von $m_i \in A$ und $v \in \mathbb{N}^n$ schreiben können. Dann kann man $m_i - m_i^* = t_i \in T$ als Optimierungsvektor für $x = v + m_i$ auffassen, weil $A(x - t_i) = A((v + m_i) - (m_i - m_i^*)) = A(v + m_i)$ und da $m_i^* \prec_c m_i$ ist, ist auch $m_i^* + v \prec_c m_i + v = x$.

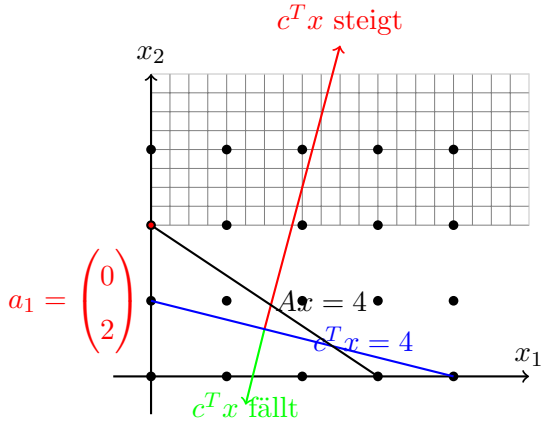
Dass die Menge T minimal ist, folgt weil eine beliebiges Paar von t_i und t_j für i$\neq j$ disjunkt sind, ansonsten wäre bereits M nicht minimal und m_i bzw. m_j kein minmaler nicht-optimaler Punkt. □

Bemerkung. Man kann sich nun auch leicht überlegen, dass für $n \leq 2$ $|T| = 1$ gelten muss. Man wähle eine beliebige 1×2-Matrix $A = \begin{pmatrix} a_{11} & a_{12} \end{pmatrix} \in \mathbb{N}^{1\times 2}$. Dann bildet das Gleichungssystem $Ax = 0$ für $x \in \mathbb{R}^2$ eine Gerade im $\mathbb{R}^2$ durch die Punkte $\alpha_1 = \begin{pmatrix} a_{12} \\ -a_{11} \end{pmatrix}$, $\alpha_2 = \begin{pmatrix} -a_{12} \\ a_{11} \end{pmatrix}$ und $\mathbf{0}$. Wenn wir nun einen Vektor $c = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \in \mathbb{N}^2$ haben, können wir ohne Beschränkung der Allgemeinheit annehmen, dass $\alpha_1 \prec_c \alpha_2$ sei. Dann erfüllt $\alpha_1 = \frac{1}{ggT(a_{11},a_{12})} \cdot \alpha_1$ die Bedingungen $A\alpha_1 = 0$, für $Ax = A(x-\alpha_1)$ ist $x-\alpha_1 \prec_c x$ und α_1 hat nur ganzzahlige Einträge. Damit ist α_1 Element der minimalen Testmenge T.

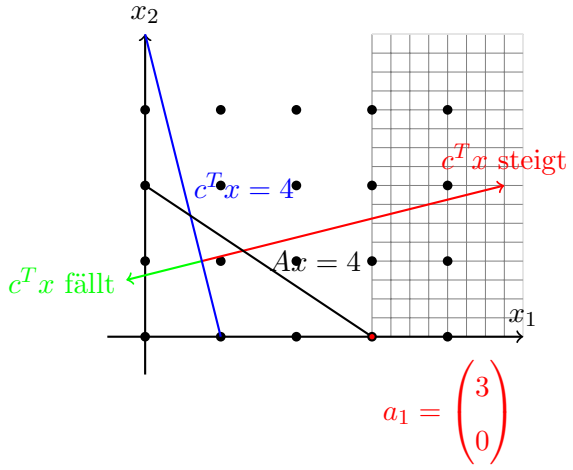
Da nun alle Lösungen von $Ax = b$ für $b \in \mathbb{N}^1$ und $x \in \mathbb{R}^2$ eine parallele Gerade zur Gerade von $Ax = 0$ bilden, kann ein minimales T außer α_1 kein weiteres zweites Element enthalten.

Man kann sich nun auch noch überlegen, welche Koordinaten der Punkt $m_1 \in M$, der Menge der minimalen nicht-optimalen Punkte, hat. Man überlege sich zunächst, dass dieser Punkt nur von A und c abhängt und für unsere Punkte α_1 und α_2 kann man die $\prec_c$-Frage anhand der Quotienten-Einträge von A und c bestimmen. Der Quotient von A sei $\delta_A = -\frac{a_{12}}{-a_{11}}$, für c sei er $\delta_c = -\frac{c_2}{c_1}$. Wenn $\delta_c < \delta_A$ gilt, so ist $_1 = \begin{pmatrix} 0 \\ \frac{a_{11}}{ggT(a_{11},a_{12})} \end{pmatrix}$, wenn $\delta_A < \delta_c$, so ist $_1 = \begin{pmatrix} \frac{a_{12}}{ggT(a_{11},a_{12})} \\ 0 \end{pmatrix}$. Falls $\delta_A = \delta_c$ gilt, so hängt Punkt von der gewählten Monomordnung $\prec$ ab.

Beispiel. Die folgenden Beispiele sollen die obigen Überlegungen verdeutlichen: Seien $A = \begin{pmatrix} 2 & 3 \end{pmatrix}$, $c = \begin{pmatrix} 1 \\ 4 \end{pmatrix}$, dann ist $\delta_c = -\frac{4}{1} = -4 < -\frac{3}{2} = \delta_A$. Also $a_1 = \begin{pmatrix} 0 \\ \frac{2}{ggT(2,3)} \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$



Als Nächstes seien nun $A = \begin{pmatrix} 2 & 3 \end{pmatrix}$, $c = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$, dann ist $\delta_A = -\frac{3}{2} < -\frac{1}{4} = \delta_c$. Also $a_1 = \begin{pmatrix} \frac{3}{ggT(2,3)} \\ 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$.



Die schraffierten Flächen sollen die Menge M der nicht-optimalen Punkte andeuten.

Man kann nun endlich eine erste Verbindung zwischen Gröbner-Basen und ganzzahligen Programmierproblemen mit Hilfe von Satz 3.5 herstellen. Denn die Menge T ent-

spricht der reduzierten GröbnerBasis des Binomial-Ideals

$$I_A := \langle X^{a^+} - X^{a^-} \,|Aa = \mathbf{0}, a \in \mathbb{Z}^n \rangle$$

mit der Monomordnung $\prec_c$.

Bemerkung. Mit a^+ ist der Teil des Vektors $a \in \mathbb{Z}^n$ gemeint, dessen Einträge positiv sind, die negativen Einträge setze man gleich 0. Analog ist $a^- := (-a)^+$, man setze alle positiven Einträge gleich 0 und nehme bei den negativen Einträge deren Beträge.

Beispiel.

$$a = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \; a^+ = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \; a^- = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Dass dies wenig hilfreich ist, sollte klar sein, da wir keine Basis dieses Ideals kennen und somit auch nicht keine reduzierte Gröbner-Basis zu berechnen können. Die Lösung dieses Problems ist eine Erweiterung des IP auf das "Extended Integer Programming Problem", kurz. EIP:

$$EIP(b) := min_{\prec_{(K\mathbf{1},c)}} \left\{ \begin{pmatrix} y \\ x \end{pmatrix} \in \mathbb{N}^{m+n} \,|E_m y + Ax = b \right\},$$

wobei $K \in \mathbb{N}$ eine sehr große Konstante und $\mathbf{1}$ den 1-Vektor mit m-Einträgen seien. Also $K\mathbf{1} = \begin{pmatrix} K \\ \vdots \\ K \end{pmatrix}$

Mit $EIP_{A,c}$ ist im Folgenden die Familie der "Extended" $IP_{A,c}$ gemeint. Offensichtlich ist $y = b$ und $x = 0$ eine Lösung des $EIP_{A,c}(b)$. Damit haben wir bereits eine Lösung gefunden, während wir für $IP_{A,c}(b)$ immernoch keine Lösung kennen. Wenn wir nun eine Testmenge für $EIP_{A,c}$ hätten, könnten wir $y = b,\ x = 0$ solange optimieren, bis wir die optimale Lösung haben. Da K eine sehr große Konstanten ist, wird der Algorithmus 3.5 nicht nur die optimale Lösung liefern, sondern auch noch zeigen, ob $IP_{A,c}(b)$ zulässig ist oder nicht. Wenn $IP_{A,c}(b)$ zulässig ist, so liefert der Algorithmus zum Beispiel $y = 0$

und $x = x_0$. Dann ist x_0 die optimale Lösung von $IP_{A,c}(b)$. Falls $y \neq 0$ war, so ist das Probleme auch nicht zulässig gewesen.

Proposition 3.7. *([CoTra]) Das Ideal*

$$I_{E_m,A} := \langle Y^{a_1^+} X^{a_2^+} - Y^{a_1^-} X^{a_2^-} \mid (E_m, A)a_1a_2 = \mathbf{0} \rangle$$

wird von den Binomen

$$Y^{Ae_j} - X_j \qquad \text{für } 1 \leq j \leq n.$$

erzeugt. Die reduzierte Gröbner Basis von $I_{(E_m,A)}$ mit der Monomordnung $\prec_{(K1,c)}$ ist das minimale Testmenge $T_{(K1,c)}$ für die Familie $EIP_{A,c}$. Dabei können wir die Elemente aus $T_{(K1,c)}$ über die kanonische Bijektion

$$\begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \leftrightarrow Y^{a_1^+} X^{a_2^+} - Y^{a_1^-} X^{a_2^-}$$

bestimmen.

Wenn wir nun alles zusammenfassen, kommen wir auf folgenden Algorithmus:

Algorithmus 3.8. *Für*

$$IP(b) \qquad min_{\prec_c}\{x \in \mathbb{N}^n | Ax = b\}$$

mit $A \in \mathbb{N}^{m \times n}, b \in \mathbb{N}^m, c \in \mathbb{N}^n$ kann das dazugehörige EIP-Problem in 2 Schritten gelöst werden.

1. Teil	*Berechne Testmenge*
Eingabe	A, c
Berechne	*reduzierte Gröber Basis $T_{(K1,c)}$ für das Ideal $I := \langle Y^{Ae_j} - X_j \rangle$*
Ausgabe	*Testmenge $T_{(K1,c)}$*
2. Teil	*Reduzierung*
Eingabe	$T_{(K1,c)}, b$
Reduziere	*das Monom Y^b mit der Monomordung $\prec_{(K1,c)}$ und nehme $Y^{a_1} X^{a_2}$*
Ausgabe	*Wenn $a_1 \neq \mathbf{0}$ dann "nicht zulässig"*
	Wenn $a_1 = \mathbf{0}$ dann "$x_0 = a_2$ ist optimal"

Eine sorgfältig dokumentierte Implementierung dieses Algorithmus in Java ist als Anhang beigefügt. Wir wollen nun noch einige Beispiele betrachten.

Beispiel. **([Thomas])** Gegeben seien

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{pmatrix}, \; c = \begin{pmatrix} 1 \\ 3 \\ 14 \\ 17 \end{pmatrix} \quad \text{und } T = \{ \begin{pmatrix} 0 \\ -1 \\ 2 \\ -1 \end{pmatrix}, \; \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix}, \; \begin{pmatrix} 1 \\ -2 \\ 1 \\ 0 \end{pmatrix} \}.$$

Dieses Beispiel wurde bereits in Algorithmus 3.5 fest implementiert. Der optimale Punkt für $b = \begin{pmatrix} 10 \\ 15 \end{pmatrix}$ ist nach Thomas $\begin{pmatrix} 0 \\ 7 \\ 1 \\ 2 \end{pmatrix}$. Die Ausgaben der Implementierungen der Algorithmen 3.5 und 3.8 bestätigen dies:

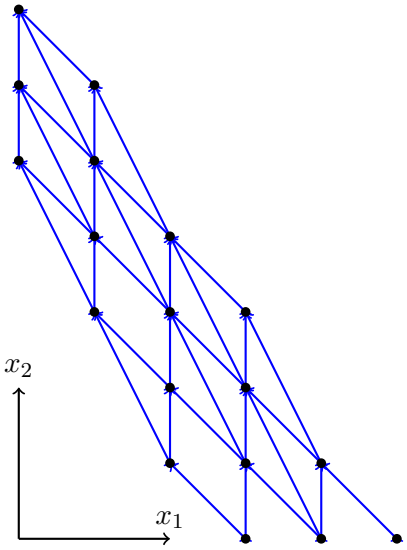
Ausgabe Alg. 3.5:

```
1  ...
2  Die Länge der folgdenen Liste beträgt: 8
3
4   0 7 1 2
5  termc: 69
6
7   1 6 0 3
8  termc: 70
9  ...
```

Ausgabe 3.8:

```
1  ...
2  Bitte geben Sie jetzt den 2-zeiligen Ergebnis-Vektor b ein:
3  10
4  Der Eingebene Wert war: 10
5  15
6  Der Eingebene Wert war: 15
7  Das Problem ist zulässig und die optimale Lösung lautet:
8
9   0 7 1 2
```

Thomas stellt die Menge, der zulässigen Punkte für $IP(\begin{pmatrix}10\\15\end{pmatrix})$, in der x_1 und x_2-



Ebene wie folgt dar: Abb 3.2

Er untersucht das Problem auf Basis eines gerichten Graphen, in dem das Optimum dem Knoten ohne Ausgehende Kante entspricht und auf geometrische Art.

4 Programme

Algorithmus 3.5

Für den Algorithmus 3.5 wurden die fünf Dateien *Alg_3_5.java*, *Liste.java*, *VektorInt.java*, *MatrixInt.java* und *RList.java* erstellt. Die eigentliche Algorithmus ist dabei in der ersten Datei zu finden. Diese wiederum ruf die nächsten drei Dateien auf, damit die entsprechenden Objekt und deren Methoden und Funktionen genutzt werden können. Die *Liste.java* ruft zur Verwaltung der Listen Objekte die *RList.java* auf.

Die Datei *Alg_3_5.java* hat feste Werte vorgegeben, d.h. wenn man ein anderes Beispiel berechnen möchten muss man die Werte im Quellcode ändern. Dies ist zwar nicht sonderlich schön ist, eine geänderte Version ist aber auch im nächsten Programme-Code enthalten.

Die Dateien *VektorInt.java* und *MatrixInt.java* repräsentieren die Objekte Matrix und Vektor sowie alle benötigten Funktionen wie +, -, *, *bereit*.

Algorithmus 3.8

Vorweg sei gesagt, dass der Algorthmus in meiner Java-Version mit 3.9 bezeichnet ist. Da ich ich mich bei den Namen nachdem zugrunde liegenden Artikel von Günther Ziegler gerichtet habe.

Für den Algorithmus 3.8 wurden die 7 Dateien *Alg_3_5.java*, *Alg_3_9.java*, *VektorBigInt.java*, *MatrixBigInt.java*, *Polynom.java*, *Liste.java*,und *RList.java* erstellt. Des Weiteren wurden die Dateien *Rational.java* von Anton Mellit und *Input.java* von der AG Betriebssysteme, Uni-Bremen Jan Peleska, verwendet. Der eigentliche Algorithmus ist in den ersten beiden Datei zu finden. Diese wiederum benutzen die Objekten, Methoden und Funktionen der Dateien *VektorBigInt.java*, *MatrixBigInt.java*, *Polynom.java* und *Rational.java*. Ebenso wird die zur Verwaltung noch die *Liste.java* genutzt, welche wiederum zur internen Verwaltung der einzelnen Listen-Objekte die *RList.java* nutzt. Die Input.java wird innerhalb von *Alg_3_9.java* für das Parsen der Eingabe gebraucht.

Die Datei *Alg_3_9.java* hat einen festen Werte vorgegeben, wenn man kein eigenes Beispiel eingeben möchte. Wenn man ein eigenes Beispiel eingeben möchte, wird man zunächst gebeten eine Matrix A einzugeben und danach den c-Vektor für $\prec_c$. Dann wird die Gröbner-Basis berechnet mit Hilfe der Funktionen, welche größtenteils im Polynom-Objekt zu finden sind. Wenn die Gröbner-Basis berechnet ist, muss man noch den gewünschten Wert für b eingeben.

Dann wird der *Alg_3_5.java* als Objekt mit den eingebenen Daten initialisiert und die optimale Lösung berechnet.

Die Dateien *VektorBigInt.java* und *MatrixBigInt.java* repräsentieren die Objekte Matrix und Vektor sowie alle benötigten Funktionen wie $+$, $-$, $*$, $\backslash$ bereit.

Literatur

[CoTra] Conti, P. ;Traverso, C. (1991). *„Buchberger algorithm and integer programming “*. Lectures Notes in Computer Science 539, Springer Verlag, S. 130-139 in Proceedings AAECC-9 (New Orleans).

[AlgebraSkript] Feichtner-Kozlov, D. (2007/2008). *„Skript zur Vorlesung: Algebra “*. Version vom 28. M 2008, S. 139-140.

[Thomas] Thomas, R.R. (1995). *„A geometric Buchberger algorithm for integer programming “*. Math. Operations Research, Volumne 20, S. 864-884.