Strategien zum Lösen parametrischer linearer Gleichungssysteme

Universit¨at Karlsruhe (TH)
Fakult¨at f¨ur Informatik
Institut f¨ur Algorithmen und Kognitive Systeme

Strategien zum Lösen parametrischer linearer Gleichungssysteme

Studienarbeit
von
Manuel Kauers

betreut von
Prof. Dr. es-sciences physiques Jacques Calmet
Dr. Clemens Ballarin

Sommersemester 2001

Inhaltsverzeichnis

1 Einf¨uhrung

Das L¨osen *linearer Gleichungssysteme* ist ein im Zusammenhang mit unterschiedlichsten Fragestellungen der verschiedensten Fachgebiete auftretendes Problem. Ebensoweit verbreitet wie das Problem ist auch der Gauß-Algorithmus (aus [Gau01]) als dessen Standardl¨osung.

Viele Computeralgebrasysteme stellen zum L¨osen linearer Gleichungssysteme neben dem reinen Gauß-Algorithmus gleich eine ganze Bibliothek von Anpassungen zur Verf¨ugung, mit denen Systeme von spezieller Gestalt besonders effizient bearbeitet werden k¨onnen.

In dieser Arbeit soll es darum gehen, den Gauß-Algorithmus auf die bisher kaum untersuchte Situation *parametrisierter* linearer Gleichungssysteme anzupassen. Solche Systeme, in denen die Koeffizienten des Systems selbst ver¨anderlich sind, treten z. B. bei der Behandlung von Differentialgleichungen in der Computeralgebra oder bei der Verifikation von algebraischen Petrinetzen (vgl. [Jen96, WR98]) auf. Als weitere Anwendungen sind Eigenwertberechnungen oder Probleme aus der Analytischen Geometrie denkbar.

Vorarbeit auf dem Gebiet der parametrisierten linearen Gleichungssysteme wurde bereits von W. J. Sit geleistet, der in [Sit92] einen Algorithmus zur L¨osung parametrischer linearer Gleichungssysteme vorstellt. Die Grundlage von Sits Algorithmus ist die Cramersche Regel, erg¨anzt um eine effiziente Organisation des Parameterraums, so daß die L¨osung eines Gleichungssystems in einer m¨oglichst geringen Anzahl von „F¨allen“ dargestellt werden kann. Sit zeigt in einer *worst case* Betrachung, daß bei einer (naiven) Verwendung des Gauß-Algorithmus mit einer exponentiell gr¨oßeren Anzahl von F¨allen zu rechnen ist als sein Algorithmus liefert.

Sits Überlegungen sind eher theoretischer Natur. F¨ur ihn steht das Problem im Vordergrund, die Anzahl der F¨alle zu minimieren und jeden Fall m¨oglichst einfach darzustellen. F¨ur die Laufzeit seines Algorithmus gibt er obere Schranken an, w¨ahrend die Laufzeit (d. h. Effizienz) der Implementierung eine eher untergeordnete Rolle spielt.

Diese Gewichtung f¨allt bei uns genau umgekehrt aus. Wenn wir versuchen, die Anzahl der F¨alle zu reduzieren, dann geschieht dies vor allem zum Zweck der Zeiteinsparung. Ebenfalls im Gegensatz zu Sit interessieren wir uns nicht f¨ur das Verhalten unseres Algorithmus im *worst case*, sondern versuchen durch Einsatz von Heuristiken den Algorithmus f¨ur den *average case* und vor allem f¨ur „Matrizen aus dem Leben“ zu optimieren. Eigenschaften der Eingabe wie d¨unne Besetzung (symbolisch oder ¨uberhaupt) oder Regelm¨aßigkeiten sollen dabei ausgenutzt werden.

Außerdem soll die M¨oglichkeit bestehen, ein parametrisiertes Gleichungssystem mit Bedingungen an die Parameter zu versehen, so daß nur diejenigen L¨osungen berechnet

werden, die diesen Bedingungen genügen. Dies soll es möglich machen, den Lösungsalgorithmus in einem größeren computer-algebraischen Zusammenhang zu verwenden, indem Wissen über bestehende Einschränkungen an die Parameter bei der Lösungsfindung verwendet wird.

Um die Probleme anzudeuten, die bei der Übertragung des Gauß-Algorithmus auf den parametrisierten Fall auftreten, sollen zunächst einige typische Beispiele betrachtet werden. Anschließend wird das Problem auf eine formale Grundlage gestellt, und es werden die verwendeten Bezeichnungsweisen definiert. Das Einführungskapitel schließt mit einem Überblick über den Inhalt der folgenden Kapitel ab.

1.1 Einführende Beispiele

Bsp. 1 Man betrachte das durch die folgende Matrix definierte, homogene System[1]:

$$A(t) := \begin{pmatrix} 1 & -2 & 3 \\ 2 & t & 6 \\ -1 & 3 & t-3 \end{pmatrix}$$

Das System soll mit Hilfe des Gauß-Algorithmus auf *Treppennormalform*[2] gebracht werden.

$$\begin{pmatrix} 1 & -2 & 3 \\ 2 & t & 6 \\ -1 & 3 & t-3 \end{pmatrix} \overset{\cdot(-2)+}{\underset{+}{\rightsquigarrow}} \rightsquigarrow \begin{pmatrix} 1 & -2 & 3 \\ 0 & t+4 & 0 \\ 0 & 1 & t \end{pmatrix} \overset{:(t+4)}{\underset{\cdot(-1)+}{\rightsquigarrow}} \rightsquigarrow \begin{pmatrix} 1 & -2 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \overset{:t}{\rightsquigarrow}.$$

Da im Verlauf des Algorithmus durch $t+4$ und $-t$ geteilt wird, bleibt zu untersuchen, wie sich das System für Belegungen verhält, in denen diese Ausdrücke verschwinden. Es ergeben sich

$$A(-4) \rightsquigarrow \begin{pmatrix} 1 & -2 & 3 \\ 0 & 1 & t \\ 0 & 0 & 0 \end{pmatrix}, \qquad A(0) \rightsquigarrow \begin{pmatrix} 1 & -2 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

und damit Lösungen von völlig anderer Gestalt. Solche zusätzlichen Lösungen werden als *Spezialfälle* bezeichnet.

[1] In dieser Arbeit werden nur homogene Gleichungssysteme betrachtet, und diese werden in der üblichen Weise mit Matrizen identifiziert.

[2] Wir begnügen uns hier mit der Treppennormalform als „*Lösung*“ eines Systems, da es unproblematisch ist, aus dieser Darstellung die „echten“ Lösungen (je nach Fragestellung z. B. Lösungsraum, Rang oder Determinante) zu gewinnen.

Das Beispiel zeigt, daß die Gefahr unnötiger Fallunterscheidungen besteht. Wählt man im ersten Schritt nach einer Vertauschung der ersten und dritten Zeile z. B. $t-3$ als Pivot-Element, so hätte man den zusätzlichen Fall $t=3$ zu untersuchen, der aber keine echt neue Lösung bringt (*scheinbarer Spezialfall*). Durch eine geeignete Strategie zur Wahl des Pivots ist also die Zahl der Spezialfälle gering zu halten.

Das zweite Beispiel zeigt aber, daß selbst bei „optimaler" Pivotwahl noch scheinbare Spezialfälle auftreten können.

Bsp. 2 Es sei $A\colon \mathbb{R} \to \mathbb{R}^{3\times 3}$ definiert durch

$$A(t) := \begin{pmatrix} 1 & 1 & 1 \\ t & 2t & 2 \\ t+1 & 0 & 2t \end{pmatrix}.$$

Will man Spezialfälle vermeiden, so muß man im ersten Schritt a_{11} als Pivot wählen. Man erhält dann

$$A(t) = \begin{pmatrix} 1 & 1 & 1 \\ t & 2t & 2 \\ t+1 & 0 & 2t \end{pmatrix} \begin{matrix} \\ \leftarrow -t \\ \leftarrow -(t+1) \end{matrix} \rightsquigarrow \begin{pmatrix} 1 & 1 & 1 \\ 0 & t & 2-t \\ 0 & -t-1 & t-1 \end{pmatrix}.$$

Was immer man nun als Pivot auswählt, führt zu einem Spezialfall. Tatsächlich kann man aber wegen

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & t & 2-t \\ 0 & -t-1 & t-1 \end{pmatrix} \leftarrow + \quad \mid \cdot(-1)$$

$$\rightsquigarrow \begin{pmatrix} 1 & 1 & 1 \\ 0 & t & 2-t \\ 0 & 1 & -1 \end{pmatrix} \begin{matrix} \\ \leftarrow -t \\ \leftarrow + \end{matrix} \quad \mid :2 \quad \rightsquigarrow \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

auf eine Fallunterscheidung ganz verzichten.

Will man im Gauß-Algorithmus eine Vertauschung von Spalten zulassen und wählt im ersten Schritt eine andere Konstante als Pivot aus, so steht man im zweiten Schritt vor ähnlichen Problemen.

Neben einer geschickten Pivotwahl kann man die Zahl der Spezialfälle manchmal offenbar durch Einfügen geeigneter Zwischenoperationen weiter reduzieren.

Eine weitere Möglichkeit zur Vermeidung scheinbarer Spezialfälle ist die Verwendung von Wissen über den gerade untersuchten Fall. Befindet man sich z. B. gerade im Fall $t^2-1=0$, und als potentielle Pivot-Elemente stehen $t-3$, t^2 und $t-1$ zur Auswahl, dann kann man $t-3$ und t^2 wie Konstanten behandeln, denn aus $t^2-1=0$ folgt

$$t=1 \quad \vee \quad t=-1.$$

Allgemeiner gesprochen kann man das Wissen über den aktuellen Fall zur *Vereinfachung* von Matrixeintr̈agen verwenden. So kann man im Fall $t^2-1=0$ zwar keine Aussage über t^2+2t+1 machen, man kann diesen Ausdruck aber immerhin zu $2t+2$ vereinfachen.

Zusammenfassend gibt es drei Ansätze zur Vermeidung unnötiger Spezialfälle:

- Geeignete Auswahl von Pivot-Elementen
- Einfügen zusätzlicher Zeilenoperationen
- Verwendung von Wissen über den aktuellen Fall

1.2 Definition des Problems

Nach der eher informellen Beschreibung des Problems im vorangegangenen Abschnitt soll das Problem nun formal definiert und häufig verwendete Begriffe eingeführt werden. Einige mathematische Grundlagen, die vorausgesetzt werden, sind im Anhang ab Seite 55 zusammengestellt.

Es sei k stets ein algebraisch abgeschlossener Körper. Man denke bei Beispielen etwa an $\overline{\mathbb{Q}}$ oder $\mathbb{C}$.

Def. 1 Eine *Variablenbelegung* (oder *Belegung* oder *Spezialisierung* oder *Substitution*) ist eine Abbildung

$$\sigma\colon \{x_1,\ldots,x_r\} \to k.$$

Ist σ eine Belegung und $p \in k[x_1,\ldots,x_r]$ (vgl. Def. 10), dann sei

$$\sigma(p) := p(\sigma(x_1),\ldots,\sigma(x_r)) \in k.$$

Ist allgemeiner $t = f(t_1,\ldots,t_n)$ ein *Ausdruck* mit den Unterausdrücken $t_1,\ldots,t_n$ (z. B. eine Matrix), dann sei

$$\sigma(t) := f(\sigma(t_1),\ldots,\sigma(t_n)).$$

Die Menge aller Substitutionen $\sigma\colon \{x_1,\ldots,x_r\} \to k$ wird mit Σ bezeichnet.

Wir verwenden also den Begriff der „‘Variablenbelegung“’ zugleich als Substitution im Sinne der Logik[3] und als Einsetzungshomomorphismus für Polynome. Formal fallen beide Begriffe zusammen, wenn man die Definition der Anwendung einer Substitution auf einen Ausdruck so erweitert, daß nach Ersetzung der Variablen der entstehende Ausdruck mit Termersetzungsregeln wie $5^2 \to 25$ weitestmöglich vereinfacht wird.

Das zu lösende *Problem* lautet dann, für eine gegebene Matrix $A \in k(x_1,\ldots,x_r)^{n\times m}$ $(m,n,r \in \mathbb{N})$ und für jede Spezialisierung $\sigma\colon \{x_1,\ldots,x_r\} \to k$ das lineare Gleichungssystem

$$\sigma(A)\cdot x = 0$$

zu lösen. Hierbei sind $x_1,\ldots,x_r$ die *Parameter* des Systems $Ax = 0$.

[3] ..., wobei wir der Einfachheit halber Variablen nicht durch Ausdrücke, sondern nur durch Körperelemente (d. h. Konstanten im Sinne der Logik) ersetzen wollen.

Def. 2

1. Ist $p \in k[x_1, \ldots, x_r]$, so heißen die Aussagen $p = 0$ und $p \neq 0$ *Nebenbedingung* (oder *Randbedingung* oder *Bedingung*).

2. Eine endliche Menge $F := \{b_1, \ldots, b_n\}$ von Nebenbedingungen heißt *Fall*. Es sei Σ_F die Menge aller Spezialisierungen σ, f¨ur die

$$\sigma(b_1) \wedge \sigma(b_2) \wedge \cdots \wedge \sigma(b_n)$$

   wahr ist.

Def. 3 Sei $F = \{b_1, b_2, \ldots, b_n\}$ ein Fall und $p \in k[x_1, \ldots, x_r]$.

1. p heißt *konstant* bez¨uglich F (oder *relative Konstante*), wenn gilt

$$\forall\, \sigma \in \Sigma_F : \sigma(p) \in k \setminus \{0\}.$$

   Schreibweise: $F \models (p \neq 0)$.

   $p \neq 0$ heißt *echt konstant* (oder *absolute Konstante*), wenn p keine Variablen enth¨alt (d. h. $p \in k \setminus \{0\}$).

2. p heißt *Null* bez¨uglich F (oder *relative Null*), wenn gilt

$$\forall\, \sigma \in \Sigma_F : \sigma(p) = 0.$$

   Schreibweise: $F \models (p = 0)$.

   0 heißt *echte Null* (oder *absolute Null*).

3. p heißt *symbolisch*, wenn p nicht echt konstant ist.

4. F heißt *inkonsistent* (oder *widersprüchlich*), wenn $\Sigma_F = \varnothing$ gilt.

5. F heißt *konsistent*, wenn F nicht inkonsistent ist.

Bsp. 3 Im Fall $\{x - 1 = 0\}$ ist $x + 3$ eine (relative) Konstante, $x^2 - 5$ ist symbolisch, $(x-1)^2 \cdot (x+9)$ ist eine (relative) Null. Das Element 0 ist stets die einzige absolute Null, eine absolute Konstante ist z. B. 42.

Der Fall $\{x^2 - 1 = 0, x - 1 \neq 0, x + 1 \neq 0\}$ ist inkonsistent.

Allgemein gilt: Ein Fall F ist genau dann inkonsistent, wenn die Konjunktion der ihn definierenden Bedingungen unerf¨ullbar ist. Dies ist offensichtlich gleichbedeutend mit $F \models (1 = 0)$.

Def. 4 Sei $A = ((a_{ij})) \in k(x_1,\ldots,x_r)^{n\times m}$.

1. A heißt in *Treppennormalform*, wenn

$$A = \begin{pmatrix} 0 & \cdots & 0 & p_1 & p_2 & \cdots & p_k \end{pmatrix} \qquad (p_1,\ldots,p_k \in k(x_1,\ldots,x_r))$$

oder

$$A = \begin{pmatrix} 0 & \cdots & 0 & p_1 & p_2 & \cdots & p_k \\ \vdots & & \vdots & 0 & & & \\ \vdots & & \vdots & \vdots & & A' & \\ 0 & \cdots & 0 & 0 & & & \end{pmatrix} \qquad (p_1,\ldots,p_k \in k(x_1,\ldots,x_r))$$

mit A' in Treppennormalform. Ist $A = ((a_{ij}))$ in Treppennormalform, dann heißt ein Element $a_{ij} \neq 0$ mit $a_{kj} = 0$ $(k < i)$ und $a_{ik} = 0$ $(k > j)$ *Stufenelement* von A.

2. Ist F ein Fall, dann heißt $A' \in k(x_1,\ldots,x_r)^{n\times m}$ *Treppennormalform* von A bezüglich F, wenn A' Treppennormalform im Sinne von 1. ist, A durch elementare Zeilenumformungen nach A' überführbar ist und jedes Stufenelement von A' konstant bezüglich F ist.

3. Seien Σ die Menge aller Spezialisierungen und F_λ $(\lambda \in \Lambda)$ Fälle mit

$$\Sigma = \dot{\bigcup_{\lambda\in\Lambda}} \Sigma_{F_\lambda}.$$

Weiter sei A_λ eine Treppennormalform von A bezüglich F_λ. Dann heißt die Menge

$$L := \{(A_\lambda, F_\lambda) : \lambda \in \Lambda\}$$

eine *Lösung* (oder *Lösungsmenge*) des Systems A.

1.3 Überblick

Die Grundlage unseres Lösungsverfahrens ist der Gauß-Algorithmus, der entlang der am Ende von Abschnitt 1.1 aufgeführten Punkte erweitert werden soll. Wann immer es sich nicht vermeiden läßt, ein symbolisches Element p als Pivot auszuwählen, wird der Rest der Elimination doppelt ausgeführt, nämlich für den (Unter-)Fall $p = 0$ und den (Unter-)Fall $p \neq 0$.

Kapitel 2 beschreibt das Lösungsverfahren sowie die Heuristiken, die zur Vermeidung von Verzweigungen vorgeschlagen werden. Abschnitt 2.1 zeigt, daß es genügt, Matrizen über $k[x_1,\ldots,x_r]$ statt über $k(x_1,\ldots,x_r)$ zu betrachten. Es wird daher im folgenden davon ausgegangen, daß alle Nenner in einem Vorverarbeitungsschritt durch Multiplikation jeder Zeile mit dem Produkt aller in ihr vorkommenden Nenner durchmultipliziert wird, und nur noch der Unterfall betrachtet wird, in dem die beseitigten Nenner als konstant angenommen werden. In 2.2 und 2.3 werden Verfahren eingeführt, die zur Pivot-Wahl (Abschnitt 2.4) verwendet werden.

Die Behandlung der Randbedingungen erfolgt durch ein separates Modul, das vom Rest des L¨osungsalgorithmus unabh¨angig ist. Dieses Modul ist Thema von Kapitel 3. Das Modul stellt als Schnittstelle die Methoden isConstant, isZero, addConstant, addZero und simplifyPolynom mit der naheliegenden Funktionalit¨at zur Verf¨ugung. Kapitel 3 besch¨aftigt sich mit den Algorithmen, die diesen Methoden zugrunde liegen. Da es sich hier um den laufzeitkritischsten Teil in der gesamten Architektur handelt, werden zwei Ans¨atze zur Leistungssteigerung vorgestellt.

In Kapitel 4 wird die Implementierung des Systems im Computeralgebrasystem MuPAD beschrieben und ein Überblick ¨uber die Leistungsf¨ahigkeit gegeben. Die Arbeit schließt mit einer Zusammenfassung ab.

2 Der Lösungsalgorithmus

Unser L¨osungsverfahren ist eine Erweiterung des Algorithmus von Gauß und ist in Algorithmus 1 dargestellt.

Die in Zeile 2 verwendete Funktion simplifyMatrix ersetze jeden Eintrag a_{ij} der Matrix A durch simplifyPolynom(a_{ij}), wobei simplifyPolynom zusammen mit isConstant (Zeile 7) und isZero (Zeile 10) die Schnittstelle zur Verwaltungseinheit des aktuellen Falls C bildet. Diese Methoden entscheiden $C \models (p \neq 0)$ bzw. $C \models (p = 0)$ im Sinne von Definition 3. Die Realisierung jener Methoden ist Thema des n¨achsten Kapitels (Seite 33).

Primitive Methoden werden nicht eigens eingef¨uhrt, ihre Semantik wird auf Seite 69 spezifiziert. Im ¨ubrigen gehen wir davon aus, daß die verwendete Pseudocode-Sprache keiner Erkl¨arung bedarf.

```
1  function elimination(A,C,row,col)
2    A := simplifyMatrix(C,A);   // page 33
3    while row < rows(A) and col < cols(A) do
4      (i,j) := choosePivot(A);   // page 25
5      A := swapRow(A,i,row);
6      A := swapColumn(A,j,col);
7      if isConstant(C,A[row,col]) then   // valid pivot element
8        A := eliminationStep(A,row,col);   // page 19
9        row := row+1; col := col+1;
10     else if isZero(C,A[row,col]) then   // only zeros in this column
11       col := col+1;
12     else   // branch on = 0 and ≠ 0
13       return elimination(A,C∪{A[i,j]=0},row,col) ∪
14              elimination(A,C∪{A[i,j]≠0},row,col);
15   return {(A,C)};
```

Algorithmus 1 Grundger¨ust des L¨osungsalgorithmus. Initial seien $C = \varnothing$ und $row = col = 1$.

2.1 Gauß-Elimination in euklidischen Ringen

Der Gauß-Algorithmus verlangt in seiner urspr¨unglichen Form nach Matrizen ¨uber K¨orpern, da nach Auswahl eines Pivot-Elements a die Zeile mit diesem Element durch p zu teilen ist.

Es ist naheliegend, die Division der Pivotzeile durch a einzusparen und stattdessen die Zeile, die eliminiert wird, zunächst mit a zu multiplizieren:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{pmatrix} \begin{array}{l} \\ \mid \cdot a \\ \mid \cdot a \\ \mid \cdot a \end{array} \begin{array}{l} \\ \leftarrow \\ \leftarrow \\ \leftarrow \end{array} \begin{array}{l} -d \\ + \\ + \\ + \end{array} \begin{array}{l} \\ -g \\ \\ \end{array} \begin{array}{l} \\ \\ -j \\ \end{array} \rightsquigarrow \begin{pmatrix} a & b & c \\ 0 & ae-bd & af-cd \\ 0 & ah-bg & ai-cg \\ 0 & ak-bj & al-cj \end{pmatrix}. \tag{1}$$

Dieses Verfahren wird als *divisionsfreier* Gauß-Algorithmus bezeichnet. Es hat den entscheidenden Nachteil, daß die Eintr¨age der Matrix exponentiell in der Gr¨oße der Matrix wachsen (vgl. [GCL92], S. 392).

Verfolgt man das Verfahren jedoch einen Schritt weiter, so stellt man fest, daß sich die dann verbleibenden Zeilen s¨amtlich durch das ehemalige Pivot a teilen lassen:

$$\begin{pmatrix} a & b & c \\ 0 & ae-bd & af-cd \\ 0 & ah-bg & ai-cg \\ 0 & ak-bj & al-cj \end{pmatrix} \begin{array}{l} \\ \mid \cdot (ae-bd) \\ \mid \cdot (ae-bd) \end{array} \begin{array}{l} \\ \leftarrow \\ \leftarrow \end{array} \begin{array}{l} -(ah-bg) \\ + \\ + \end{array} \begin{array}{l} \\ \\ -(ak-bj) \end{array}$$

$$\rightsquigarrow \begin{pmatrix} a & b & c \\ 0 & ae-bd & af-cd \\ 0 & 0 & a^2ei-abdi-aceg-a^2fh+acdh+abfg \\ 0 & 0 & a^2el-abdl-acej-a^2fk+acdk+abfj \end{pmatrix}$$

Da a ein ehemaliges Pivot-Element ist, ist $a \neq 0$, so daß die Division erkl¨art ist. Allgemein gilt, daß nach dem i-ten Eliminationsschritt alle Elemente der verbleibenden Submatrix durch das Pivot des $(i-1)$-ten Schritts teilbar sind ($i > 1$).

Es l¨aßt sich zeigen, daß bei dieser sogenannten *Gauß-Bareiss-Elimination* die Eintr¨age nur noch linear in der Gr¨oße der Matrix anwachsen (vgl. [GCL92], S. 394).

F¨ur die weiteren Überlegungen ist von Interesse, welche Operationen auf der Matrix ausgef¨uhrt werden k¨onnen, ohne die Teilbarkeitseigenschaften zu zerst¨oren.

Satz 1 Die Teilbarkeitseigenschaft in der Gauß-Bareiss-Elimination bleibt erhalten, wenn zwischen zwei Eliminationsschritten beliebige elementare Zeilenumformungen durchgef¨uhrt werden.

Bew. Falls a eine Einheit ist, ist nichts zu zeigen, da dann jedes Element durch a teilbar ist. Sei also $a \notin R^\times$. Wir betrachten o. B. d. A. die Untermatrix

$$A := \begin{pmatrix} ae-bd & af-cd \\ ah-bg & ai-cg \\ ak-bj & al-cj \end{pmatrix}$$

aus (1) und rechnen nach.

1. Vertauschen von Zeilen ist aus trivialen Gr¨unden erlaubt.

2. Multiplikation einer Zeile mit $p \neq 0$

$$A \rightsquigarrow \begin{pmatrix} ae-bd & af-cd \\ ah\cdot p - bg\cdot p & ai\cdot p - cg\cdot p \\ ak-bj & al-cj \end{pmatrix} \begin{array}{l} \\ \\ \mid \cdot (ae-bd) \end{array} \begin{array}{l} \xrightarrow{-(ahp-bgp)} \\ \\ \leftarrow + \end{array}$$
$$\rightsquigarrow \begin{pmatrix} ae-bd & af-cd \\ 0 & a^2eip - abdip - acegp - a^2fhp + acdhp + abfgp \\ 0 & a^2el - abdl - acej - a^2fk + acdk + abfj \end{pmatrix}$$

3. Addition des p-fachen einer Zeile zu einer anderen

$$A = \begin{pmatrix} ae-bd & af-cd \\ ah-bg & ai-cg \\ ak-bj & al-cj \end{pmatrix} \begin{array}{l} \xrightarrow{p} \\ \leftarrow + \end{array}$$
$$\rightsquigarrow \begin{pmatrix} ae-bd & af-cd \\ ah-bg & ai-cg \\ a(\underbrace{k+ph}_{=:k'}) - b(\underbrace{j+pg}_{=:j'}) & a(\underbrace{l+pi}_{=:l'}) - c(\underbrace{j+pg}_{=j'}) \end{pmatrix} \begin{array}{l} \\ \\ \mid \cdot (ae-bd) \end{array} \begin{array}{l} \xrightarrow{-(ak'-bj')} \\ \\ \leftarrow + \end{array}$$
$$\rightsquigarrow \begin{pmatrix} ae-bd & af-cd \\ 0 & a^2ei - abdi - aceg - a^2fh + acdh + abfg \\ 0 & a^2el' - abdl' - acej' - a^2fk' + acdk' + abfj' \end{pmatrix}$$

Nach erfolgtem Eliminationsschritt (incl. Division durch das alte Pivot) beschränkt man die Betrachtung auf die verbleibende Submatrix, die (nach Variablenumbenennung) wieder von der gleichen Form wie A ist.

Induktiv erhält man die Behauptung. ∎

Das Verfahren von Gauß-Bareiss liefert einen ersten Algorithmus für eliminationStep:

```
1  procedure eliminationStep1(A, row, col)
2    // Let oldPivot be a global variable containing the last pivot element,
3    // and let initially oldPivot = 1.
4    for r = i+1 to rows(A) do
5      for c = j+1 to cols(A) do
6        A[r,c] := (A[i,j]·A[r,c] − A[i,c]·A[r,j]) / oldPivot;
7    oldPivot := A[i,j];
```

Algorithmus 2 Eliminationsschritt nach Gauß-Bareiss (vgl. [GCL92])

Das folgende Beispiel zeigt zwei Probleme, die bei Verwendung von Algorithmus 2 auftreten.

Bsp. 4 Nach der nach Algorithmus 2 durchgeführten Elimination

$$\begin{pmatrix} 2 & 1 & -1 \\ 0 & 3 & 1 \\ 3 & -2 & 0 \end{pmatrix} \begin{array}{l} \\ \cdot 2 \leftarrow +0 \\ \cdot 2 \leftarrow +(-3) \end{array} \rightsquigarrow \begin{pmatrix} 2 & 1 & -1 \\ 0 & 6 & 2 \\ 0 & -7 & 3 \end{pmatrix} \begin{array}{l} \\ \cdot 6 \leftarrow +7 \\ \\ \end{array} \begin{array}{l} \\ :2 \\ \\ \end{array}$$

$$\rightsquigarrow \begin{pmatrix} 2 & 1 & -1 \\ 0 & 6 & 2 \\ 0 & 0 & 16 \end{pmatrix}$$

ist weder Zeile 2 noch Zeile 3 optimal minimiert.

Im ersten Schritt wurde Zeile 2 durch die Elimination nur mit 2 multipliziert, um die Teilbarkeit im nächsten Schritt zu garantieren – für die Elimination selbst ist diese Multiplikation bedeutungslos.

Im zweiten Schritt entsteht zufällig eine höhere Teilbarkeit als von Bareiss vermutet. Eine derartige Suboptimalität war zu erwarten, wenn man bedenkt, daß der Divisor ja Teiler der gesamten Submatrix ist und Informationen über einzelne Zeilen nicht einfließen.

Neben diesen allgemeinen Problemen haben wir zusätzlich den Wunsch, auf der Matrix auch solche Vereinfachungsoperationen durchzuführen, die sich nicht als Folge elementarer Zeilenumformungen formulieren lassen, und die im Allgemeinen die Teilbarkeitseigenschaft von Bareiss zerstören (vgl. etwa Zeile 2 in Algorithmus 1).

Da wir ein großes Interesse daran haben, die Zeilen weitestmöglich zu reduzieren, weil hierdurch symbolische Einträge verschwinden könnten, erscheint es lohnend, an dieser Stelle eine höhere Komplexität in Kauf zu nehmen, um eine „optimal“ reduzierte Zeile zu erhalten. Statt durch das letzte Pivot wird daher in Zeile 6 durch den größten gemeinsamen Teiler jeder Zeile dividiert.

Allerdings ist das Teilen durch einen symbolischen Ausdruck nicht ohne weiteres möglich (er könnte verschwinden), weshalb man vom gcd zum *gccd* (greatest common constant divisor) übergeht, der wie folgt definiert ist.

```
1  function gccd(C, p_1, ..., p_n)
2    if p_1 = p_2 = ··· = p_n = 0 then
3      return 1;
4    g := gcd(p_1, ..., p_n);
5    [f_1^{e_1}, ..., f_k^{e_k}] := factor(g);   // let g = ∏_{i=1}^{k} f_i^{e_i}
6    p := 1;
7    for i = 1 to k do
8      if isConstant(C, f_i) then
9        p := p · f_i^{e_i};
10   return p;
```

Algorithmus 3 Berechnung des größten gemeinsamen Teilers von $p_1, \ldots, p_n$, der bezüglich C konstant ist

Zusammenfassend wird also folgendes endg¨ultiges Eliminationsschema verwendet:

```
1  function eliminationStep(C,A,i, j)
2    for r = i+1 to rows(A) do
3      if A[i, j] ≠ 0 then
4        for c = j+1 to cols(A) do
5          A[r,c] := A[i, j] · A[r,c] − A[i,c] · A[r, j];
6        g := gccd(C,A[r,i],...,A[r,cols(A)]);
7        if ∂g > 0 then    // avoid trivial divisions
8          for c = j+1 to cols(A) do
9            A[r,c] := A[r,c] / g;
```

Algorithmus 4 Elimination aller in Spalte j unter Zeile i stehenden Elemente von A

2.2 Das Markowitz-Kriterium

Steht man im Gauß-Algorithmus bei d¨unn besetzten Eingabematrizen vor der Entscheidung, welches Element man als Pivot ausw¨ahlen soll, so sollte man ber¨ucksichtigen, in wieweit sich eine Wahl auf die Dichte der Matrix auswirkt.

Überlegungen in diesem Zusammenhang f¨uhren zum Begriff des *Fill-ins*. Man betrachtet f¨ur jede Position der Submatrix die Ver¨anderung des Eintrags durch die Elimination in Abh¨angigkeit vom gew¨ahlten Pivot. Die Zahl der Positionen, an denen durch die Elimination eine Null „verschwindet“, wird als Fill-in bezeichnet.

Das aus der Numerik bekannte Kriterium von *Markowitz* f¨ur d¨unn besetzte Matrizen [DER86, Kap. 7] liefert die kleinste obere Schranke f¨ur den Fill-in.

Satz 2 Sei $A = ((a_{ij})) \in k^{n\times m}$,

$$r_i := |\{a_{ij} : a_{ij} \neq 0, j = 1,\ldots,m\}|$$
$$c_j := |\{a_{ij} : a_{ij} \neq 0, i = 1,\ldots,n\}|.$$

Dann ist $(r_i - 1)\cdot(c_j - 1)$ der maximal m¨ogliche Fill-in bei Wahl von q_j als Pivot-Element.

Bew. Der Fill-in kann nicht gr¨oßer sein: Das Pivot q_j ist nur zul¨assig im Fall $q_j \neq 0$. Daher verbleiben $(r_i - 1)$ zu eliminierende Zeilen. Alle diese Zeilen werden mit dem Pivot multipliziert, was noch keinen Fill-in bringt. Dann wird zu jeder dieser Zeilen ein Vielfaches der Pivot-Zeile addiert. Fill-in kann aber nur dort stattfinden, wo die Pivot-Zeile nichttriviale Eintr¨age hat. Neben dem Pivot sind dies genau $(c_j - 1)$ Stellen. Damit ist $(r_i - 1)\cdot(c_j - 1)$ eine obere Schranke f¨ur den Fill-in.

Wenn die Submatrix nur Nullen enth¨alt, findet an genau $(r_i - 1)\cdot(c_j - 1)$ Stellen ein Fill-in statt. ∎

Das Markowitz-Kriterium macht den nun naheliegenden Vorschlag, das Element $a_{ij} \neq 0$ als Pivot zu wählen, für das der potentielle Fill-in $(r_i - 1)(c_j - 1)$ minimal ist.

Bsp. 5 Ist $A = ((a_{ij}))$ von der Form

$$A := \begin{array}{c} \\ 2 \\ 3 \\ 1 \end{array} \begin{array}{c} \begin{array}{ccc} 3 & 1 & 2 \end{array} \\ \begin{pmatrix} * & 0 & * \\ * & * & * \\ * & 0 & 0 \end{pmatrix} \end{array},$$

so sind a_{31} und a_{22} die Elemente mit minimalem potentiellen Fill-in. Bei ihrer Wahl findet gar kein Fill-in statt.

Wählt man das laut Markowitz ungünstigste Element a_{21} als Pivot aus, so ist ein potentieller Fill-in von $4 = (3-1)(3-1)$ zu erwarten, nämlich an den Positionen $(1,2),(1,3),(3,2)$ und $(3,3)$. Da $a_{13} \neq 0$ ist, liegt der tatsächliche Fill-in mit 3 aber etwas günstiger als die Vorhersage des Kriteriums.[4]

Das Konzept des Fill-ins und das Markowitz-Kriterium sollen nun auf symbolische Matrizen übertragen werden. Die Unterscheidung zwischen Null- und Nicht-Null-Eintragen wird erweitert in eine Klassifizierung Null (absolut oder relativ), echt konstant, relativ konstant und symbolisch (vgl. Def. 3). Diese *Fill-in-Klassen* seien durch die Symbole 0, 1, x bzw. X repräsentiert.

Die Verknüpfungen $+$ und $\cdot$ seien auf den Fill-in-Klassen wie folgt definiert:

$\cdot$	0	1	x	X
0	0	0	0	0
1	0	1	x	X
x	0	x	x	X
X	0	X	X	X

$+$	0	1	x	X
0	0	1	x	X
1	1	1	X	X
x	x	X	X	X
X	X	X	X	X

Diese Definition ist stellenweise pessimistisch, z. B. gilt nicht notwendig $1+1=1$, denn durch gegenseitige Auslöschung könnte eine Null entstehen. Die Verknüpfungen sind gerade so definiert, daß $a \circ b$ die „schlimmste“ Klasse ist, in der die Verknüpfung eines Elements aus a mit einem Element aus b liegen kann.[5]

Die Frage ist nun, wie viele Elemente bei Wahl eines gewissen Pivots in Folge der Elimination ihre Klasse ändern. Elemente a der Submatrix gehen durch Elimination in Elemente $ab - cd$ über. Ein solcher Übergang wird als Fill-in der Klasse $*$ bezeichnet, wenn $ab - cd$ zur Klasse $*$ und a zu einer einfacheren Klasse gehört.

Satz 3 Sei $A = ((a_{ij})) \in k^{n \times m}$ eine Matrix. r_i^* sei die Anzahl der Elemente der Klasse $* \in \{0,1,x,X\}$ in Zeile i ($i = 1,\ldots,n$) und c_j^* die Zahl der Elemente der Klasse $*$ in Spalte j ($j = 1,\ldots,m$).

Es sei f_{ij}^* der „schlimmste“ anzunehmende Fill-in der Klasse $*$ ($* \in \{1,x,X\}$) bei Wahl des Pivots a_{ij}.

[4] Experimente mit zufälligen Matrizen zeigen, daß der dadurch entstehende Fehler vernachlässigbar ist ([DER86] und Kapitel 4).

[5] Man verifiziere dies z. B. für den nichttrivialen Fall $x + 1 = X$

1. Ist $a_{ij} = 1$, so ist

$$\begin{aligned} f_{ij}^1 &= (r_i^1 - 1) \cdot (c_j^1 - 1), \\ f_{ij}^x &= 0, \\ f_{ij}^X &= (m - r_i^0 - 1) \cdot (n - c_j^0 - 1) - (r_i^1 - 1) \cdot (c_j^1 - 1). \end{aligned}$$

2. Ist $a_{ij} = x$, so ist

$$\begin{aligned} f_{ij}^1 &= 0, \\ f_{ij}^x &= r_i^0 \cdot (n - c_j^0 - 1), \\ f_{ij}^X &= (m - r_i^0 - 1) \cdot (n - c_j^0 - 1). \end{aligned}$$

3. Ist $a_{ij} = X$, so ist

$$\begin{aligned} f_{ij}^1 &= f^x = 0, \\ f_{ij}^x &= (m-1) \cdot (n - c_j^0 - 1). \end{aligned}$$

Dabei sei *„schlimmer“* definiert durch $>$ mit $X > x > 1 > 0$.

Bew. Der Beweis erfolgt durch Nachrechnen und Abz¨ahlen der Stellen, an denen ein Fill-in der jeweiligen Klasse stattfindet. F¨ur die Klasse, der das Pivot angeh¨ort, ist von der Anzahl jeweils 1 abzuziehen (vgl. Bew. zu Satz 2)

1. Das Pivot ist konstant.

$$\begin{pmatrix} 1 & X & x & 1 & 0 \\ X & * & * & * & * \\ x & * & * & * & * \\ 1 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}$$

$-X$, $-x$, -1; $\cdot 1$, $+$

$$\rightsquigarrow \begin{pmatrix} 1 & X & x & 1 & 0 \\ 0 & * \cdot 1 - X \cdot X & * \cdot 1 - x \cdot X & * \cdot 1 - 1 \cdot X & * \cdot 1 \\ 0 & * \cdot 1 - X \cdot x & * \cdot 1 - x \cdot x & * \cdot 1 - 1 \cdot x & * \cdot 1 \\ 0 & * \cdot 1 - X \cdot 1 & * \cdot 1 - x \cdot 1 & * \cdot 1 - 1 \cdot 1 & * \cdot 1 \\ 0 & * & * & * & * \end{pmatrix}$$

Unter Zuhilfenahme der oben angegebenen Annahmen f¨ur $+$ und $\cdot$ erh¨alt man, daß der Kasten links oben genau die F¨alle mit potentiellem Fill-in X enth¨alt, w¨ahrend der mittlere Kasten die F¨alle mit Fill-in 1 und der Kasten rechts unten die F¨alle ohne Fill-in beinhaltet.

Die Anzahl der Elemente jedes Kastens entspricht genau der Behauptung im Satz. (Es findet kein x-Fill-in statt.)

2. Das Pivot ist relativ konstant

$$\begin{pmatrix} x & X & x & 1 & 0 \\ X & * & * & * & * \\ x & * & * & * & * \\ 1 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix} \quad \begin{matrix} & & & -X & -x & -1 \\ \cdot x & \leftarrow & + & & & \\ \cdot x & \leftarrow & & & + & \\ \cdot x & \leftarrow & & & & + \end{matrix}$$

$$\rightsquigarrow \begin{pmatrix} x & X & x & 1 & 0 \\ 0 & *\cdot x - X\cdot X & *\cdot x - x\cdot X & *\cdot x - 1\cdot X & *\cdot x \\ 0 & *\cdot x - X\cdot x & *\cdot x - x\cdot x & *\cdot x - 1\cdot x & *\cdot x \\ 0 & *\cdot x - X\cdot 1 & *\cdot x - x\cdot 1 & *\cdot x - 1\cdot 1 & *\cdot x \\ 0 & * & * & * & * \end{pmatrix}$$

Im Kasten links findet X-Fill-in, rechts Fill-in von x und unten gar kein Fill-in statt.

3. Das Pivot ist symbolisch.

$$\begin{pmatrix} X & X & x & 1 & 0 \\ X & * & * & * & * \\ x & * & * & * & * \\ 1 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix} \quad \begin{matrix} & & & -X & -x & -1 \\ \cdot X & \leftarrow & + & & & \\ \cdot X & \leftarrow & & & + & \\ \cdot X & \leftarrow & & & & + \end{matrix}$$

$$\rightsquigarrow \begin{pmatrix} X & X & x & 1 & 0 \\ 0 & *\cdot X - X\cdot X & *\cdot X - x\cdot X & *\cdot X - 1\cdot X & *\cdot X \\ 0 & *\cdot X - X\cdot x & *\cdot X - x\cdot x & *\cdot X - 1\cdot x & *\cdot X \\ 0 & *\cdot X - X\cdot 1 & *\cdot X - x\cdot 1 & *\cdot X - 1\cdot 1 & *\cdot X \\ 0 & * & * & * & * \end{pmatrix}$$

Der obere Kasten beinhaltet die potentiellen X-Fill-in-Situationen, und im Kasten unten findet wieder kein Fill-in statt.

Matrizen, für die die vorausgesagten Fill-ins tatsächlich stattfinden, erhält man durch Wahl geeigneter Belegungen für $*$. ■

Im ursprünglichen Verfahren wurde als Pivot das Element minimalen Fill-ins ausgewählt. Es bleibt nun zu überlegen, wie sich dies auf die Erweiterung mit den verschiedenen Fill-in-Klassen übertragen läßt.

An erster Stelle steht dabei wieder unser Interesse an Matrizen mit möglichst wenigen symbolischen Eintragen (d. h. wenig X-Fill-in), um so den Suchraum klein zu halten. Die Minimierung des x-Fill-ins hat den Sinn, die Terme klein zu halten, was sich insbesondere positiv auf die Cache-Trefferquote (vgl. Abschnitt 3.5) auswirkt. Die Minimierung des Fill-ins von Elementen der Klasse 1 hat die geringste Priorität.

Es ist daher naheliegend, die Fill-ins wie in Satz 3 zu ordnen, d. h. ein Element a_{ij} ist einem Element a_{kl} vorzuziehen, wenn

$$f_{ij}^X < f_{kl}^X \vee (f_{ij}^X = f_{kl}^X \wedge (f_{ij}^x < f_{kl}^x \vee (f_{ij}^x = f_{kl}^x \wedge f_{ij}^1 < f_{kl}^x)))$$

gilt. Falls der Fill-in in allen Klassen übereinstimmt, soll die Komplexität des Ausdrucks (z. B. gemessen im Grad oder in der Zahl der Monome[6]

Zusätzlich ist noch die Klasse des Pivots selbst zu berücksichtigen: Nur die Null-Matrix darf ein Pivot 0 zurückgeben, und konstante Werte (d. h. 1 und x) sind symbolischen Werten X stets vorzuziehen.

Zusammenfassend ergibt sich der folgende Auswahlalgorithmus, in dem die Bezeichnungsweisen aus Satz 3 verwendet werden:

```
1  function markowitz(C,A)
2    r := 1; c := 1;
3    for i = 1 to rows(A) do
4      for j = 1 to cols(A) do
5        if A[i,j] ≠ 0 then
6          if A[r,c] = 0 then
7            r := i; c := j;
8          else if A[r,c] = A[i,j] = X then
9            if f_ij^X < f_rc^X then
10             r := i; c := j;
11         else if f_ij^X < f_rc^X or (f_ij^X = f_rc^X and f_ij^x < f_rc^x) then
12           r := i; c := j;
13         else if f_ij^X = f_rc^X and f_ij^x = f_rc^x then
14           if f_ij^1 < f_rc^1 then
15             r := i; c := j;
16           else if f_ij^1 = f_rc^1 and isSimpler(A[i,j],A[r,c]) then
17             r := i; c := j;
18   return (r,c);
```

Algorithmus 5 Pivot-Wahl nach dem erweiterten Markowitz-Kriterium

2.3 Spaltenvereinfachung

Wie in Beispiel 2 auf Seite 9 gesehen, reicht eine gute Strategie zur Pivot-Wahl allein im Allgemeinen nicht aus, um die Zahl der Fälle zu minimieren. Die Architektur soll daher um ein Verfahren ergänzt werden, das für eine gegebene Spalte versucht, durch Linearkombination der Zeilen einen konstanten – oder zumindest einfacheren – Eintrag zu erhalten.

Das Verfahren besteht aus drei Schritten. Zunächst wird für jedes Element der Spalte geprüft, ob es sich durch ein anderes vereinfachen läßt. Gegebenenfalls wird eine Vereinfachung durchgeführt und die zugehörige Zeilenumformung notiert. Dies wird so oft wiederholt, bis entweder eine Konstante auftritt oder sich nichts mehr vereinfachen

[6] In unserer Implementierung ist ein Polynom p einfacher als q, wenn $\partial p \cdot |p| < \partial q \cdot |q|$ gilt. Bei Gleichheit dieses Produkts entscheidet $|\cdot|$. Herrscht auch hier Gleichheit, wird ∂ verglichen.

l¨aßt. Im zweiten Schritt wird das am weitesten vereinfachte Element der Spalte ausgew¨ahlt und die Menge der Zeilenoperationen bestimmt, die zum Erreichen dieses Elements n¨otig sind. Der dritte Schritt besteht aus der Ausf¨uhrung dieser Umformungen auf der gesamten Matrix.

Ein Polynom p l¨aßt sich mit Hilfe eines Polynoms q vereinfachen, falls $\mathrm{LT}(q) \mid \mathrm{LT}(p)$ gilt. In diesem Fall ist

$$p' := p - \frac{\mathrm{LM}(p)}{\mathrm{LM}(q)} \cdot q \tag{2}$$

das vereinfachte Polynom und die zugeh¨orige Zeilenoperation ist die Addition des $\mathrm{LM}(p)/\mathrm{LM}(q)$-fachen der Zeile von q zur Zeile von p. Wegen $\mathrm{LT}(p') \prec \mathrm{LT}(p)$ terminiert eine iterierte Anwendung dieses Vereinfachungsprinzips.

Der Algorithmus wird an einem Beispiel vorgef¨uhrt.

Bsp. 6 Gegeben sei die Matrix

$$A := \begin{pmatrix} z^4+1 & * & * & * \\ x^5+y^3+z & * & * & * \\ x^3 & * & * & * \\ y^4-y^3 & * & * & * \\ y^2+z & * & * & * \end{pmatrix}$$

Das Verfahren zur Spaltenvereinfachung soll auf die erste Spalte angewandt werden.

Im ersten Schritt werden die m¨oglichen Vereinfachungen bestimmt:

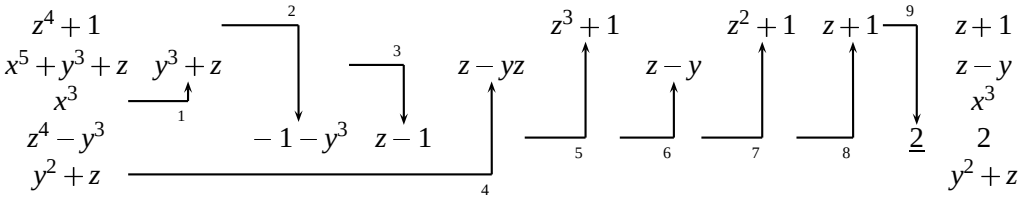

Abbildung 1 Erster Schritt der Spaltenvereinfachung: Aufbau des Vereinfachungsgraphen

L¨aßt sich ein Polynom auf verschiedene Arten vereinfachen, so w¨ahlt der Algorithmus eine m¨ogliche Art aus und ignoriert alle anderen Vereinfachungen.

Als einfachstes Element wird im zweiten Schritt das Element 2 ausgew¨ahlt. Es ist nun die minimale Folge von Operationen zu bestimmen, die 2 aus der Eingabespalte erzeugen. Dazu wird ausgehend vom ausgew¨ahlten Element rekursiv jede Operation markiert, die zur dessen Kombination beigetragen hat.

Im Beispiel sind dies die Operationen 1, 2, 3, 5, 7, 8 und 9. Die nicht markierten Umformungen 4 und 6 sind ¨uberfl¨ussig und werden gel¨oscht, um die Restmatrix nicht unn¨otig zu verkomplizieren.

Im dritten Schritt werden die ausgew¨ahlten Operationen auf der gesamten Matrix durchgef¨uhrt und $(4,1)$ als Pivot-Vorschlag zur¨uckgegeben.

Experimente zeigen, daß die Vereinfachung vor allem bei Matrizen mit einer Variablen erfolgreich ist. Das liegt daran, daß für zwei Polynome $p,q \in k[x]$ stets $\mathrm{LT}(p) \mid \mathrm{LT}(q)$ oder $\mathrm{LT}(q) \mid \mathrm{LT}(p)$ gilt, so daß eine Reduktion immer möglich ist. Für multivariate Polynome gilt dies nicht mehr. Sind z. B. $p,q \in k[x,y]$ mit $\mathrm{LT}(p) = x^2y$ und $\mathrm{LT}(q) = xy^2$, so ist keine Vereinfachung möglich. Man könnte zwar die Definition aus Gleichung (2) in

$$p' := \frac{\mathrm{lcm}(\mathrm{LT}(p),\mathrm{LT}(q))}{\mathrm{LM}(p)} \cdot p - \frac{\mathrm{lcm}(\mathrm{LT}(p),\mathrm{LT}(q))}{\mathrm{LM}(q)} \cdot q$$

erweitern, um auch solche Fälle abzudecken (vgl. die Definition *S*-Polynome im Zusammenhang mit Gröbnerbasen), aber um die zugehörige Zeilenumformung durchführen zu können, müßte gewährleistet sein, daß für den aktuellen Fall F gilt

$$F \models \left(\frac{\mathrm{lcm}(\mathrm{LT}(p),\mathrm{LT}(q))}{\mathrm{LM}(p)} \neq 0 \right).$$

Da die Wahrscheinlichkeit hierfür relativ gering ist, zeigt sich, daß sich der Einsatz der lang laufenden Beweisroutinen nicht lohnt.

Eine erfreuliche Eigenschaft des Algorithmus ist, daß in den meisten Fällen relativ schnell erkannt wird, wenn keine Vereinfachung möglich ist, so daß in diesem Fall kaum Zeit verloren wird. Andererseits haben die aufwendigen Datenstrukturen zur Folge, daß im Erfolgsfall ein erheblicher Teil der Laufzeit mit Spaltenvereinfachungen verbracht wird.

2.4 Pivot-Wahl

Die Strategien der letzten Abschnitte sollen nun zur Pivot-Wahl-Routine choosePivot zusammengesetzt werden. Dabei sind die Laufzeiten sowie die Erfolgsaussichten zu berücksichtigen.

An dieser Stelle hat man eine Reihe von Freiheitsgraden, mit denen man Einfluß auf das Laufzeitverhalten und die Gestalt der Lösung ausüben kann. Zum Beispiel wäre denkbar, statt dem Markowitzkriterium, das den Fill-in nur abschätzt, in einer erschöpfenden Tiefensuche festzustellen, welches Pivot tatsächlich die beste Wahl ist. Es zeigt sich aber, daß bereits bei einer Vorschau der Tiefe 1 die zusätzliche Laufzeit die Laufzeiteinsparung infolge der geringeren Zahl von Fällen übertrifft.

Wir verwenden die folgende siebenstufige Entscheidungsfindung, die wir als einen geeigneten Kompromiß zwischen Laufzeitminimierung und Reduzierung der Zahl der Fälle ansehen.

Sei A die Submatrix, für die ein Pivot zu bestimmen ist, und C der aktuelle Fall.

1. Falls $(r,c) := \mathrm{markowitz}(A)$ bezüglich C konstant ist, wird (r,c) zurückgegeben.
2. Andernfalls wird die Spalte c aus 1. nach einem echt konstanten Eintrag durchsucht. Falls ein solcher in Zeile r' existiert, wird (r',c) zurückgegeben.
3. Andernfalls wird versucht, die Spalte c mit dem Verfahren aus Abschnitt 2.3 zu

vereinfachen. Entsteht dabei ein bezüglich C konstanter Eintrag in Zeile r'', so wird die Position (r'', c) ausgewählt.[7]

4. Andernfalls wird die Spalte c nach Einträgen durchsucht, die bezüglich C konstant sind. Existiert ein solcher Eintrag in Zeile r''', so sei (r''', c) das ausgewählte Pivot.

5. Andernfalls gibt es nur symbolische Einträge in der Spalte c. Falls (r, c) aus 1. bezüglich C nicht verschwindet, wird dieser Wert als Pivot gewählt.

6. Andernfalls wird die Spalte nach irgendeinem Wert durchsucht, der in C nicht Null ist. Falls solche Einträge existieren, so wird die Position des einfachsten dieser Ausdrücke zurückgegeben.

7. Andernfalls handelt es sich um eine Nullspalte, und es kann irgendeine Position zurückgegeben werden, etwa $(1, c)$.

Wie für die Korrektheit von Algorithmus 1 erforderlich, ist das ausgewählte Element genau dann Null, wenn es sich um eine Nullspalte handelt. Darüberhinaus wird vermieden, symbolische Pivots zurückzugeben, da dies in Algorithmus 1 sofort eine Verzweigung nach sich ziehen würde. Um teure Aufrufe von isConstant zu vermeiden, wird im zweiten Schritt nur nach echten Konstanten gesucht. In der Implementierung wird aus dem gleichen Grund einer Suche nach einer relativen Konstanten stets eine Suche nach einer echten Konstanten vorausgeschickt.

[7] mögliche Variationen: Vereinfache probehalber jede Spalte und wähle die, für die die Kombination des konstanten Eintrags den wenigsten Fill-in in der Restmatrix bewirkt. Oder: Vereinfache Spalte c und wenn die Matrix verändert wurde, beginne wieder bei Schritt 1.

Beide Varianten führen zu einer geringfügig kleineren Fallunterscheidung, erhöhen aber doch die durchschnittliche Laufzeit.

3 Verwaltung von Nebenbedingungen

In diesem Kapitel geht es darum, eine Gleichheitstheorie zu entwickeln, mit deren Hilfe man für einen vorgelegten Fall F und ein Polynom p entscheiden kann, ob $F \models (p = 0)$ oder $F \models (p \neq 0)$ (oder keins von beiden) gilt. Unsere Theorie wird dabei weniger logische, als viel mehr algebraische Argumente benutzen.

Dazu bemerken wir zunächst, daß die Menge aller Polynome p mit $F \models (p = 0)$ ein Ideal bilden: Aus Definition 2 auf Seite 11 folgt nämlich unmittelbar

$$\forall\, p,q \in k[x_1,\ldots,x_r] : F \models (p=0) \wedge F \models (q=0) \Longrightarrow F \models (p+q=0)$$
$$\text{und} \qquad \forall\, p,q \in k[x_1,\ldots,x_r] : F \models (p=0) \Longrightarrow F \models (p\cdot q=0)$$

(vgl. Def 8 und Satz 8).

Hätte man ein Erzeugendensystem dieses Ideals, so könnte man mit dessen Hilfe entscheiden, ob p in diesem Ideal liegt und hätte so auch $F \models (p = 0)$ entschieden. Ist F von der Form $\{p = 0 : p \in P\}$ ($P \subseteq k[x_1,\ldots,x_r]$), so könnte man erwarten, daß P ein Erzeugendensystem für das Ideal $\{p : F \models (p = 0)\}$ ist. Dies ist aber nicht notwendigerweise wahr:

Bsp. 7 Im Fall $F := \{x^2 = 0\}$ gilt $F \models (x = 0)$, jedoch nicht $x \in \langle x^2 \rangle$.

Es wird sich aber zeigen (vgl. Satz. 8), daß dies in gewissem Sinne die einzige Ausnahme ist. Betrachtet man nämlich das Radikalideal $\mathrm{Rad}\langle P\rangle$ anstelle des von P erzeugten Ideals $\langle P\rangle$, so erhält man einen vollständigen und korrekten Kalkül. In obigem Beispiel ist dann $x \in \mathrm{Rad}\langle x^2\rangle = \langle x\rangle$, was zu $F \models (x = 0)$ nicht mehr im Widerspruch steht.

Zentrales Hilfsmittel des zu entwickelnden Kalküls werden Gröbnerbasen sein, weshalb zunächst die nötigen Konzepte und Resultate aus dieser Theorie bereitgestellt werden (vgl. auch die mathematischen Grundlagen im Anhang). In Abschnitt 3.2 wird dann darauf aufbauend der Kalkül $\vdash$ definiert und dessen Vollständigkeit und Korrektheit gezeigt. Die beiden letzten Abschnitte enthalten Vorschläge zur Effizienzsteigerung.

3.1 Mathematische Hilfsmittel: Gröbnerbasen

Wir folgen in unserer Darstellung im wesentlichen [BWK93] und [CLO92]. Aus Platzgründen verzichten wir darauf, die aufgeführten Sätzen zu beweisen und begnügen uns bei für unsere Zwecke wesentlichen Aussagen auf Plausibilitätserklärungen. Der an streng formalen Beweisen interessierte Leser sei auf [CLO92] und [BWK93] verwiesen.

Der Begriff der Gröbnerbasis wird in den beiden genannten Werken unterschiedlich (aber natürlich äquivalent) eingeführt. Wir zitieren die hier Definition aus [BWK93]

(Def. 5.18, gering modifiziert), die von einer Reduktionsrelation $\to$ auf $k[x_1,\ldots,x_r]$ ausgeht.

Def. 5 Seien $G \subseteq k[x_1,\ldots,x_r]$, $f, \tilde{f} \in k[x_1,\ldots,x_r] \setminus \{0\}$.

1. Sei $g \in G$. f läßt sich mit g zu $\tilde{f}$ reduzieren, Schreibweise $f \to_g^0 \tilde{f}$, falls ein Monom m von f mit $\mathrm{LM}(g) \mid m$ existiert, so daß
$$\tilde{f} = f - \frac{m}{\mathrm{LM}(g)} \cdot g.$$
$\to_g$ sei die reflexive, transitive Hülle von $\to_g^0$.

2. f läßt sich modulo G zu $\tilde{f}$ reduzieren, Schreibweise $f \to_G^0 \tilde{f}$, falls ein $g \in G$ existiert mit $f \to_g^0 \tilde{f}$. $\to_G$ sei die reflexive, transitive Hülle von $\to_G^0$.

3. Ist $\to_G$ *lokal konfluent*, d. h. für alle $f_1, f_2 \in k[x_1,\ldots,x_r]$ gilt
$$(\exists\, f : f \to_G^0 f_1 \wedge f \to_G^0 f_2) \Longrightarrow (\exists\, f : f_1 \to_G f \wedge f_2 \to_G f),$$
so heißt G *Gröbnerbasis* des Ideals $\langle G \rangle$.

Satz 4 Sei $G = \{g_1,\ldots,g_n\} \subseteq k[x_1,\ldots,x_r]$ eine Gröbnerbasis. Dann gilt

1. $\to_G$ ist konfluent, noethersch und definiert eindeutige Normalformen.
2. $\forall\, f \in k[x_1,\ldots,x_r] : f \in \langle G \rangle \iff f \to_G 0$
3. $\langle \mathrm{LT}(g_1),\ldots,\mathrm{LT}(g_n) \rangle = \langle \mathrm{LT}(g) : g \in \langle G \rangle \rangle$.

Bew. Proposition 5.38 in [BWK93, S. 207]. Daß $\to_G$ noethersch ist, folgt aus der Existenz des Algorithmus (s. u.). ∎

Die durch $\to_G$ definierte Normalform eines Polynoms p ist berechenbar. Der Algorithmus subtrahiert wiederholt Polynome $q \cdot g$ (mit $g \in G$ und q geeignet) von p, um nacheinander Monome aus p zu eliminieren. Dieser Schritt wird so oft wiederholt, bis keine weitere Vereinfachung mehr möglich ist (Algorithmus 6).

Die Korrektheit dieses Algorithmus folgt unmittelbar aus Satz 4.1. Durch eine leichte Modifikation läßt sich mit Algorithmus 6 zu $f \in k[x_1,\ldots,x_r]$ und einer Gröbnerbasis G auch die Linearkombination
$$f = \tilde{f} + \sum_{i=1}^{n} p_i \cdot g_i$$
mit $p_i \in k[x_1,\ldots,x_r], g_i \in G$ und der Normalform $\tilde{f}$ von f bestimmen. Dieser Art ist der entsprechende Algorithmus in [CLO92] formuliert.

Mit $f \to_G \tilde{f}$ meinen wir im folgenden $\tilde{f} = \mathrm{reduce}(f)$.

```
1  function reduce(f, {g_1, ..., g_n})
2    r := 0;
3    while f ≠ 0 do
4      i := 1;
5      break := false;
6      while i ≤ s and not break do
7        if LT(g_i) | LT(f) then
8          f := f − LM(f) / LM(g_i) · g_i;
9          break := true;
10       else
11         i := i + 1;
12     if not break then
13       r := r + LM(f);
14       f := f − LM(f);
15   return r;
```

Algorithmus 6 Algorithmus zur Berechnung einer Normalform von $f \in k[x_1,\ldots,x_r]$ modulo einer Gr¨obnerbasis $G = \{g_1,\ldots,g_n\}$

Satz 5 Jedes Ideal $\mathfrak{a} \trianglelefteq k[x_1,\ldots,x_r]$ hat eine Gr¨obnerbasis, und jedes beliebige Erzeugendensystem $\{a_1,\ldots,a_n\}$ von $\mathfrak{a}$ l¨aßt sich (algorithmisch) zu einer Gr¨obnerbasis $\{a_1,\ldots,a_n,g_1,\ldots,g_r\}$ von $\mathfrak{a}$ erweitern.

Bew. Die Behauptung des Satzes bezeugt der Buchberger-Algorithmus. F¨ur den Nachweis seiner Korrektheit verweisen wir wieder [CLO92, BWK93].

```
1  function sPoly(f, g)
2    lcm := lcm(LM(f), LM(g));
3    return lcm / LT(f) · f − lcm / LT(g) · g;
4  function gBasis({f_1, ..., f_n})
5    G := {f_1, ..., f_n};
6    repeat
7      G' := G;
8      for (p, q) in G' × G' with p ≠ q do
9        s := reduce(sPoly(p, q), G);
10       if s ≠ 0 then
11         G := G ∪ {s}
12   until G = G';
13   return G;
```

Algorithmus 7 Berechnung einer Gr¨obnerbasis nach dem Algorithmus von Buchberger ■

Ist $G \subseteq k[x_1,\ldots,x_r]$, und ist ist $(p,q) \in k[x_1,\ldots,x_r] \times k[x_1,\ldots,x_r]$ ein *kritisches Paar* bez¨uglich $\to_G$, dann ist das *S*-Polynom (sPoly) s von p und q gerade so definiert, daß (p,q) bez¨uglich $\to_{G\cup\{s\}}$ kein kritisches Paar mehr ist, aber noch $\langle G\rangle = \langle G\cup\{s\}\rangle$ gilt. So werden sukzessive alle kritischen Paare entfernt.

Mit den nun zur Verf¨ugung stehenden Mitteln l¨aßt sich ein Algorithmus angeben, der f¨ur beliebige Polynome $a_1,\ldots,a_n,p \in k[x_1,\ldots,x_r]$ entscheidet, ob $p \in \mathfrak{a} := \langle a_1,\ldots,a_n\rangle$ gilt (*Membership-Problem*).

Zun¨achst beschafft man sich mit Algorithmus 7 eine Gr¨obnerbasis G von $\mathfrak{a}$. Dann braucht man wegen Satz 4.2 nur noch zu pr¨ufen, ob $p \to_G 0$ gilt. Dazu kann man Algorithmus 6 verwenden:

---

1 **function** isMember$(p,\{a_1,\ldots,a_n\})$
2 $p' :=$ reduce$(p,$ gBasis$(\{a_1,\ldots,a_n\}))$;
3 **return** $(p' = 0)$;

Algorithmus 8 L¨osung des Membership-Problems $p \in \langle a_1,\ldots,a_n\rangle$

3.2 Gleichheit multivariater Polynome

Auf der Grundlage der Ergebnisse des vorherigen Abschnitts wird nun der Kalk¨ul $\vdash$ unserer Gleichungslogik definiert.

Als erstes werden die algebraisch unhandlichen Bedingungen der Form $p \neq 0$ auf Bedingungen der Form $q = 0$ f¨ur geeignete q reduziert.

Def. 6 Sei $p \in k[x_1,\ldots,x_r]$ und $y \notin \{x_1,\ldots,x_r\}$ eine neue Variable. Dann heißt das Polynom

$$\neg p := p\cdot y - 1 \in k[x_1,\ldots,x_r,y]$$

das *negierte Polynom* von p.

Die neue Variable y heißt *private Variable* oder *Slick-Variable*, die x_i im Gegensatz dazu *öffentliche Variablen*.

Satz 6 Sei $p \in k[x_1,\ldots,x_r]$. Dann gilt

1. $\{p = 0\}$ und $\{\neg p \neq 0\}$ beschreiben die gleichen F¨alle, d. h.
$$\Sigma_{\{p=0\}} = \bigcap_{y\in k} \{\sigma \in \Sigma : \sigma(p)\cdot y - 1 \neq 0\}.$$
2. $\{p \neq 0\}$ und $\{\neg p = 0\}$ beschreiben die gleichen F¨alle, d. h.
$$\Sigma_{\{p\neq 0\}} = \bigcup_{y\in k} \{\sigma \in \Sigma : \sigma(p)\cdot y - 1 = 0\}.$$

Bew.

1. „$\subseteq$“ Für $\sigma \in \Sigma_{\{p=0\}}$ gilt $\sigma(p) = 0$, damit

   $$\sigma(\neg p) = \sigma(p \cdot y - 1) = \sigma(p) \cdot y - 1 = -1$$

   für jede Belegung von y. Damit liegt σ in der rechten Seite.

   „$\supseteq$“ Sei σ eine Substitution mit $\sigma(p) \cdot y - 1 \neq 0$ ($y \in k$). Wäre auch $\sigma(p) \neq 0$, dann wäre $1/\sigma(p)$ eine zulässige Belegung für y, was wegen

   $$\sigma(p) \cdot y - 1 = \sigma(p) \cdot \frac{1}{\sigma(p)} - 1 = 0$$

   im Widerspruch zur Wahl von σ steht.

2. „$\subseteq$“ Sei $\sigma \in \Sigma_{\{p\neq 0\}}$. Dann ist $\sigma(p) \neq 0$ und durch Wahl von $y := 1/\sigma(p)$ folgt die Behauptung.

   „$\supseteq$“ Liegt σ in der rechten Seite, dann gibt es ein y mit $\sigma(p) \cdot y - 1 = 0$. Dann folgt unmittelbar $\sigma(p) \neq 0$ und damit die Behauptung. ∎

Man sieht leicht ein, daß sich Satz 6 und damit die Reduktion von $p \neq 0$ auf $\neg p = 0$ auf Fälle mit mehreren Bedingungen übertragen läßt, solange man nur darauf achtet, daß jede private Variable y nur einmal verwendet wird.

Formal erreicht man dies induktiv, indem man einen Fall F über $k[x_1, \ldots, x_r]$ zunächst in die Menge F_1 der Gleichungen und F_2 der Ungleichungen aufteilt und sukzessive Ungleichungen $p \neq 0$ aus F_2 löscht und $\neg p = 0$ in F_1 einfügt. Dabei erweitert man bei jeder solchen Verschiebung den Polynomring um eine neue private Variable.

Für den Rest dieses Abschnitts seien alle Ungleichungen in diesem Sinne beseitigt, F also eine Menge von Gleichungen $p = 0$ mit $p \in k[x_1, \ldots, x_r, y_{r+1}, y_{r+2}, \ldots]$. (Ist im folgenden von $k[x_1, \ldots, x_r]$ die Rede, so ist implizit stets $k[x_1, \ldots, x_r, y_{r+1}, y_{r+2}, \ldots]$ gemeint.) Bei jeder Verwendung des Operators $\neg$ werde eine neue Variable verwendet, die der Einfachheit halber stets mit y bezeichnet wird.

Def. 7 Es seien $P \subseteq k[x_1, \ldots, x_r]$ und $F := \{p = 0 : p \in P\}$. Weiter sei $p \in k[x_1, \ldots, x_r]$. Dann wird der Kalkül $\vdash$ definiert durch

$$\begin{aligned} F \vdash (p = 0) \quad &:\Longleftrightarrow \quad \text{reduce}(1, \text{gBasis}(P \cup \{\neg p\})) = 0 \\ F \vdash (p \neq 0) \quad &:\Longleftrightarrow \quad \text{reduce}(1, \text{gBasis}(P \cup \{p\})) = 0, \end{aligned}$$

wobei reduce und gBasis durch die Algorithmen 6 und 7 definiert sind.

Satz 7 Seien $P \subseteq k[x_1, \ldots, x_r]$ und $F := \{p = 0 : p \in P\}$. Weiter sei $p \in k[x_1, \ldots, x_r]$. Dann gilt

1. $F \vdash (p = 0) \iff 1 \in \langle P \cup \{\neg p\} \rangle$ und
2. $F \vdash (p \neq 0) \iff 1 \in \langle P \cup \{p\} \rangle$.

Bew. Die Aussagen folgen unmittelbar aus der Korrektheit von Algorithmus 8 (Ideal-Membership-Problem). ∎

Def. 8 Sei $P \subseteq k[x_1,\ldots,x_r]$ und $F := \{p = 0 : p \in P\}$.

1. Die Menge
$$V_F := \{(\sigma(x_1),\ldots,\sigma(x_r)) \in k^r : \sigma \in \Sigma_F\} \subseteq k^r$$
heißt von F *erzeugte Varietät*.

2. Die Menge
$$I_F := \{p \in k[x_1,\ldots,x_r] : F \models (p = 0)\} \subseteq k[x_1,\ldots,x_r]$$
heißt von F *erzeugtes Ideal*.

Satz 8 Sei $P \subseteq k[x_1,\ldots,x_r]$ und $F := \{p = 0 : p \in P\}$. Weiter sei V_F die von F erzeugte Variet¨at und I_F das von F erzeugte Ideal. Dann gilt

1. V_F ist wirklich eine Variet¨at, n¨amlich $V_F = \mathrm{Var}(P)$.
2. I_F ist wirklich ein Ideal, n¨amlich $I_F = \mathrm{Rad}\langle P\rangle$.

Bew.

1. Die Gleichheit $V_F = \mathrm{Var}(P)$ folgt unmittelbar aus der Definition von Σ_F. Mit $\mathrm{Var}(P)$ ist dann auch V_F eine Variet¨at.

2. I_F ist ein Ideal: Seien $p_i \in I_F$ und $q_i \in k[x_1,\ldots,x_r]$ $(i = 1,\ldots,n)$. F¨ur jedes $\sigma \in \Sigma_F$ gilt dann
$$\sigma(\underbrace{\sum_{i=1}^{n} p_i \cdot q_i}_{=:p}) = \sum_{i=1}^{n} \underbrace{\sigma(p_i)}_{=0} \cdot \sigma(q_i) = 0,$$
also $F \models (p = 0)$ und damit $p \in I_F$. Also ist I_F ein Ideal.

   Wegen
$$\begin{aligned} I_F &= \{p \in k[x_1,\ldots,x_r] : F \models (p = 0)\} \\ &= \{p \in k[x_1,\ldots,x_r] : \sigma(p) = 0 \ (\sigma \in \Sigma_F)\} \\ &= \{p \in k[x_1,\ldots,x_r] : p(a) = 0 \ (a \in V_F)\} \\ &= I(V_F) \overset{1.}{=} I(\mathrm{Var}(P)). \end{aligned}$$
   folgt aus dem Hilbertschen Nullstellensatz (Satz 13.5) die Identit¨at $I_F = \mathrm{Rad}\langle P\rangle$.

∎

Satz 9 Der Kalk¨ul $\vdash$ ist vollst¨andig und korrekt, d. h. es gilt

$$\begin{aligned} F \models (p = 0) \quad &\Longleftrightarrow \quad F \vdash (p = 0), \\ F \models (p \neq 0) \quad &\Longleftrightarrow \quad F \vdash (p \neq 0) \end{aligned}$$

f¨ur jede konsistente Gleichungsmenge F und jedes Polynom p.

Bew. Sei $F = \{p_i = 0 : i = 1, \ldots, m\}$ f¨ur eine Menge $P = \{p_1, \ldots, p_m\}$ von Polynomen. Dann zeigt

$$\begin{aligned} F \vdash (p = 0) &\overset{\text{Def. 7}}{\Longleftrightarrow} \operatorname{reduce}(1, \operatorname{gBasis}(P \cup \{\neq p\})) = 0 \\ &\overset{\text{Satz 7}}{\Longleftrightarrow} 1 \in \langle P \cup \{\neg p\} \rangle \qquad \overset{\text{Satz 13.6}}{\Longleftrightarrow} p \in \operatorname{Rad}\langle P \rangle \\ &\overset{\text{Satz 8}}{\Longleftrightarrow} p \in I_F \qquad \overset{\text{Def. 8}}{\Longleftrightarrow} F \models (p = 0), \\ F \vdash (p \neq 0) &\overset{\text{Def. 7}}{\Longleftrightarrow} \operatorname{reduce}(1, \operatorname{gBasis}(P \cup \{p\})) = 0 \\ &\overset{\text{Satz 7}}{\Longleftrightarrow} 1 \in \langle P \cup \{p\} \rangle \qquad \overset{\text{Satz 13.4}}{\Longleftrightarrow} \operatorname{Var}(P \cup \{p\}) = \varnothing \\ &\overset{\text{Satz 13.3}}{\Longleftrightarrow} \operatorname{Var}(P) \cap \operatorname{Var}(\{p\}) = \varnothing \overset{\text{Satz 8}}{\Longleftrightarrow} \forall\, x \in V_F : p(x) \neq 0 \\ &\overset{\text{Def. 8.1}}{\Longleftrightarrow} \forall\, x \in \Sigma_F : \sigma(p) \neq 0 \qquad \overset{\text{Def. 3}}{\Longleftrightarrow} F \models (p \neq 0) \end{aligned}$$

die Behauptung. ∎

3.3 Implementierung der Schnittstelle

Auf der Grundlage der Ergebnisse des vorangegangenen Abschnitts werden nun Algorithmen angegeben, die die Schnittstelle zum Nebenbedingungs-Modul implementieren.

Ein Fall $F = \{p = 0 : p \in C\}$ wird allein durch die Menge $C \subseteq k[x_1, \ldots, x_r]$ repr¨asentiert.

```
1  function simplifyPolynom(C, p)
2      return reduce(A[i, j], gBasis(C));
3  function addZero(C, p)
4      return C ∪ {p};
5  function addConstant(C, p)
6      return C ∪ {¬p};
7  function isZero(C, p)
8      return isMember(1, C ∪ {¬p});
9  function isConstant(C, p)
10     return isMember(1, C ∪ {p});
```

Algorithmus 9 Implementierung der Schnittstellenmethoden

3.4 Algebraische Verfeinerungen

In der ersten Version der Implementierung in MuPAD, in der die Algorithmen den hier abgedruckten Pseudo-Code-Versionen noch sehr nah waren, betrug der Anteil der Gröbnerbasenberechnung an der Gesamtlaufzeit über 98%. Durch verschiedene Optimierungen, von denen die beiden wichtigsten in diesem und dem folgenden Abschnitt beschrieben werden, konnte dieser Anteil auf einen Wert um 33% gedrückt werden (abhängig von der Gestalt der Matrix, vgl. Kapitel 4).

Intern werden die Fälle als Mengen von Polynomen repräsentiert, zu denen durch Hinzufügen jeweils eines weiteren Polynoms neue Fälle erzeugt werden können. Durch Ausnutzung dieser Eigenschaft läßt sich der Umfang der zu berechnenden Gröbnerbasen merklich reduzieren. Die Idee dabei ist, jedem Fall ein „Stück“ einer Gröbnerbasis zuzuordnen, so daß sich eine Gröbnerbasis als Vereinigung der Stücke aller Oberfälle schreiben läßt.

Mathematische Grundlage hierfür ist der folgende Satz:

Satz 10 Seien $\mathfrak{a} = \langle a_1, \ldots, a_n \rangle, \mathfrak{b} = \langle a_1, \ldots, a_n, b_1, \ldots, b_m \rangle \trianglelefteq k[x_1, \ldots, x_r]$, $\{a_1, \ldots, a_n\}$ sei eine Gröbnerbasis von $\mathfrak{a}$. Dann gibt es Polynome $c_1, \ldots, c_k \in k[x_1, \ldots, x_r]$, so daß $\{a_1, \ldots, a_n, c_1, \ldots, c_k\}$ eine Gröbnerbasis von $\mathfrak{b}$ ist.

Bew. Wir geben einen Algorithmus zur Berechnung der c_i an, dessen Korrektheit unmittelbar aus der Korrektheit des Buchbergeralgorithmus (Algorithmus 7) folgt.

```
1  function partialGBasis({a_1,...,a_n},{b_1,...,b_m})
2    A := {a_1,...,a_n};
3    C := ∅;
4    repeat
5      C' := C;
6      for (p,q) in (A ∪ C') × C' with p ≠ q do
7        s := reduce(sPoly(p,q), A ∪ C);
8        if s ≠ 0 then
9          C := C ∪ {s};
10   until C' = C;
11   return C;
```

Algorithmus 10 Berechnung einer partiellen Gröbnerbasis ■

Verbesserte Versionen des Buchbergeralgorithmus, wie sie in [BWK93, S. 222ff] und [GMN+91] beschrieben sind, und die in der Standardimplementierung von MuPAD verwendet werden, lassen sich entsprechend übertragen. Für unseren speziellen Fall läßt sich zudem ausnutzen, daß $m = 1$ in der Bezeichnungsweise des obigen Satzes ist.

Die Vorausberechnung und Verwendung der Gröbnerbasis-Stücke bringt Beschleunigungen des Gesamtalgorithmus um bis zu einem Faktor 40. Der Nachteil ist der erhöhte Speicherbedarf, der für die Speicherung der partiellen Basen nötig wird.

3.5 Technische Verfeinerungen

Besonders bei regelm¨aßig aufgebauten Matrizen ist zu beobachten, daß die Methoden isConstant und isZero mehrmals mit den gleichen Argumenten aufgerufen werden. Wenn z. B. im Fall C vom Markowitz-Kriterium (S. 19) ein relativ konstantes Element p als Pivot ausgew¨ahlt wird, dann wird mindestens dreimal isConstant(C,p) aufgerufen (in Algorithmus 5 zur Klassifizierung, in Schritt 1 der Pivot-Wahl (S. 25) und in Zeile 7 von Algorithmus 1).

Unn¨otiger Aufwand wird zus¨atzlich bei jeder Verzweigung betrieben: in Zeile 7 von Algorithmus 1 wird das Gr¨obnerbasis-St¨uck von $C \cup \{p\}$ und in Zeile 10 das St¨uck von $C \cup \{\neg p\}$ berechnet. Beide St¨ucke werden bei Ausf¨uhrung der Verzweigung erneut berechnet.

Es ist daher naheliegend, zur weiteren Beschleunigung einen *Ergebnis-Cache* einzuf¨uhren, in dem berechnete Gr¨obnerbasis-St¨ucke sowie abgeleitetes Wissen ¨uber Polynome abgelegt werden k¨onnen.

Problematisch ist dabei, daß Gr¨obnerbasen schnell sehr groß werden, so daß es nicht m¨oglich ist, s¨amtliche Ergebnisse im Speicher zu behalten. Der Cache ist daher ¨ahnlich einem Keller organisiert, auf dem der gerade betrachtete Fall mitsamt seinen Oberf¨allen liegt. Wird ein neuer Fall F in den Cache eingef¨ugt, so werden demnach solange die oben liegenden F¨alle gel¨oscht, bis der direkte Oberfall von F oben liegt oder der Cache leer ist.

F¨ur jeden Fall C im Cache werden f¨unf Listen angelegt:

1. Liste von Polynomen p mit $C \models (p = 0)$.
2. Liste von Polynomen p, f¨ur die $C \models (p = 0)$ falsch ist.
3. Liste von Polynomen p mit $C \models (p \neq 0)$.
4. Liste von Polynomen p, f¨ur die $C \models (p \neq 0)$ falsch ist.
5. Liste von Paaren (p,b), wobei b das Gr¨obnerbasisst¨uck des Falls $C \cup \{p = 0\}$ ist.

Alle Listen sind initial leer und werden von addZero , addConstant und dem Konstruktor gef¨ullt, sobald neue Ergebnisse vorliegen.

Um die Treffer-Rate des Cache zu erh¨ohen, werden die einfachen Aussagen des folgenden Satzes verwendet:

Satz 11 Seien F, G F¨alle mit $G \subseteq F$ und $p,q \in k[x_1,\ldots,x_r]$. Dann gilt:

1. $F \models (p \cdot q \neq 0) \iff F \models (p \neq 0) \wedge F \models (q \neq 0)$,
2. $F \models (p \cdot q = 0) \iff F \models (p = 0) \vee F \models (q = 0)$,
3. $G \models (p = 0) \Longrightarrow F \models (p = 0)$, $G \models (p \neq 0) \Longrightarrow F \models (p \neq 0)$,
4. $F \models (p = 0) \Longrightarrow F \not\models (p \neq 0)$, $F \models (p \neq 0) \Longrightarrow F \not\models (p = 0)$.

Eine optimierte Version der Methode isZero kann demnach wie folgt aussehen:

```
1  function isZero(C, p)
2    // 1. trivial checks
3    if p = 0 then
4      return true;   // p is strict zero.
5    else if ∂p = 0 then
6      return false;   // p is strict constant.
7    // 2. cache look-up
8    if C ∈ cache then
9      // Let i be the index of C in the cache list.
10     if p ∈ cacheList(cache[i], 2) then   // Is p known not to be zero?
11       return false;
12     for k = 1 to i do   // Is p known to be zero?
13       if ∃ q ∈ cacheList(cache[k], 1) : q | p then
14         return true;
15     // Is p known to be constant?
16     r := p;
17     repeat
18       r' := r;
19       for k = 1 to i do
20         if ∃ q ∈ cacheList(cache[k], 3) : q | r then
21           r := r / q;
22       if ∂r = 0 then   // r is strict constant.
23         return false;
24     until r' = r;
25   else
26     cacheAppend(C);
27   // 3. radical membership
28   C' := addConstant(C, p);
29   if isConsistent(C') then   // p is not zero.
30     addToCacheList(cache[k], 2, p);
31     return false;
32   else   // p is zero.
33     addToCacheList(cache[k], 1, p);
34     return true;
```

Algorithmus 11 Optimierte Version von isZero aus Algorithmus 9

Die Optimierung von isConstant geht analog. Die Cache-Behandlung der partiellen Basen geschieht in addZero und addConstant bzw. im Konstruktor.

Bei zufälligen Matrizen wird eine Trefferquote von bis zu 75% erreicht, was zu einer merklichen Beschleunigung führt.

F¨ur manche Anfragen, z. B. bei der Klassifizierung der Eintr¨age im erweiterten Markowitz-Kriterium, ist die Vollst¨andigkeit von $\vdash$ nicht unbedingt erforderlich. An diesen Stellen werden „schnelle“ Abfragen fastIsZero und fastIsConstant anstelle von isZero und isConstant eingesetzt, die Anfragen einzig durch Nachschlagen im Cache bearbeiten. Zum Beispiel entsteht fastIsZero aus Algorithmus 11, indem man die Zeilen 27 bis 34 durch

```
27  return unknown;
```

ersetzt, wobei man unknown in den meisten F¨allen als false interpretieren wird.

Zwar sinkt die Trefferquote durch die Verwendung der fast-Methoden auf etwa 30%, weil der Cache sich langsamer f¨ullt, doch die Einsparung an Basisberechnungen zahlt sich in den meisten F¨allen aus. Man hat hier also wieder die Wahl zwischen k¨urzerer Laufzeit und eleganteren L¨osungen.

4 Experimentell gewonnene Resultate

4.1 Implementierung

Um die Leistungsf¨ahigkeit des Verfahrens zu testen, wurde es in MuPAD implementiert. MuPAD ist ein Computeralgebra-System nach dem Vorbild von Maple, das an der Universit¨at Paderborn entwickelt wurde. Von Systemen wie AXIOM (vgl. Abschnitt 4.3) unterscheidet es sich zum einen durch eine wesentlich einfachere Handhabung und zum anderen durch eine geringere Effizienz.

Im vorliegenden Abschnitt soll ein kurzer Überblick ¨uber die Implementierung gegeben werden. F¨ur Details und Interna sei auf den ausf¨uhrlich dokumentierten Quellcode verwiesen. Zur Syntax von MuPAD siehe [OPRW98].

Ein Fall wird in einer Datenstruktur namens *Constraint Store* repr¨asentiert. Die privaten Variablen werden bei Erzeugung eines Stores ¨ubergeben und haben die Form `cs_var1`, `cs_var2`, ... Im ¨ubrigen entsprechen die Bezeichner der Schnittstellenfunktionen den Bezeichnungen, die in dieser Arbeit verwendet wurden.

Bsp. 8 $\{x^2-1=0, x-1\neq 0\} \models (x+1=0)$ kann wie folgt gezeigt werden:

```
>> varList := [x, cs_var.i $ i=1..5]:
>> C := ConstraintStore(varList);
                                {}
>> C := addZero(C, poly(x^2 - 1, varList)):
>> C := addConstant(C, poly(x - 1, varList));
                                    2
                     {x - 1 <> 0, x  - 1 = 0}
>> isZero(C, poly(x + 1, varList));
                                TRUE
```

Hinzuf¨ugen von $x+1\neq 0$ zu `C` f¨uhrt folglich zu einem inkonsistenen Store:

```
>> C := addConstant(C, poly(x + 1, varList));
                                          2
              {x - 1 <> 0, x + 1 <> 0, x  - 1 = 0}
>> isConsistent(C);
                               FALSE
```

Um die L¨osungsmenge eines durch eine Matrix A gegebenen Systems zu bestimmen, gibt es die Funktion `elim`. Die L¨osung wird als Liste von Paaren $[A', C']$ zur¨uckgegeben, wie in Definition 4 spezifiziert.

Man kann der Funktion `elim` optional einen ConstraintStore `C` ¨ubergeben, es wird dann nur der Fall `C` betrachtet.

Bsp. 9 Das Beispiel 1 von Seite 8 soll automatisch gel¨ost werden.

```
≫ A := matrix(3, 3, [[1, -2, 3], [2, t, 6], [-1, 3, t - 3]]);
                      +-                -+
                      |   1, -2,   3     |
                      |                  |
                      |   2,  t,   6     |
                      |                  |
                      |  -1,  3, t - 3   |
                      +-                -+

≫ elim(A);
     -- -- +-                      -+                       --
     |  |  |  1, -2,      3        |                         |
     |  |  |                       |                         |
     |  |  |  0,  1,      t        |              2          |
     |  |  |                       |, {- 4 t - t  <> 0}      |,
     |  |  |                     2 |                         |
     |  |  |  0,  0, - 4 t - t     |                         |
     -- -- +-                      -+                       --
                                                      --
        -- +-           -+                      --    |
        |  |  1, -2, 3   |                        |    |
        |  |             |               2        |    |
        |  |  0,  1, t   |, {- 4 t - t  = 0}      |    |
        |  |             |                        |    |
        |  |  0,  0, 0   |                        |    |
        -- +-           -+                      -- --
```

Die L¨osung ist ¨aquivalent zu der in Beispiel 1 gefundenen. Man erh¨alt eine Teill¨osung, wenn man die Bedingung $\{t+4 \neq 0\}$ vorgibt:

```
≫ C := ConstraintStore([t, cs_var.i $ i=1..10]):
≫ C := addConstant(C, poly(t+4, [t, cs_var.i $ i=1..10]));
                          {t + 4 <> 0}

≫ elim(A, C);
           -- -- +-           -+                          --
           |  |  |  1, -2, 3   |                           |
           |  |  |             |                           |
           |  |  |  0,  1, 0   |, {t <> 0, t + 4 <> 0}      |,
           |  |  |             |                           |
           |  |  |  0,  0, t   |                           |
           -- -- +-           -+                          --
              -- +-           -+                       -- --
              |  |  1, -2, 3   |                        |  |
              |  |             |                        |  |
              |  |  0,  1, 0   |, {t = 0, t + 4 <> 0}   |  |
              |  |             |                        |  |
              |  |  0,  0, 0   |                        |  |
              -- +-           -+                       -- --
```

Man beachte, daß das zweite Treppennormalform von der ersten nicht subsumiert wird, da in Definition 4.2 verlangt wird, daß Strufenelemente relativ konstant sind. Dies

gew¨ahrleistet, daß man den Rang problemlos aus den Treppennormalformen ablesen kann.

Bei der Vorgabe von F¨allen ist darauf zu achten, daß alle Polynome im ¨ubergebenen Constraint Store ¨uber dem gleichen Ring definiert sind (d. h. die gleiche Variablenliste haben). Aus Effizienzgr¨unden wird n¨amlich darauf verzichtet, dies in den h¨aufig aufgerufenen Methoden zu ¨uberpr¨ufen und Polynome ggf. in einen gr¨oßeren Polynomring einzubetten.

Dies gilt insbesondere f¨ur die privaten Variablen. Die Funktion `elim` erwartet, daß ein Constraint Store, der zur L¨osung einer $n \times m$-Matrix ¨ubergeben wird, so viele freie Variablen enth¨alt, daß mindestens noch $n \cdot m + 1$ zus¨atzliche Bedingungen eingef¨ugt werden k¨onnen.

4.2 Experimente mit zufälligen Matrizen

Gerne h¨atten wir das Verhalten unseres L¨osungsverfahrens auf einer großen Menge von aus Anwendungen stammenden Matrizen getestet, doch es zeigte sich, daß derartige Matrizen entweder hochgradig trivial sind oder weit außerhalb unserer F¨ahigkeiten liegen. Es wurden daher zwei unabh¨angige experimentelle Ans¨atze verfolgt. Zum einen – und das ist Thema dieses Abschnitts – wurden statistische Untersuchungen auf einer Menge zuf¨allig erzeugter Matrizen durchgef¨uhrt. Im Gegensatz dazu wird dann in Abschnitt 4.4 das Verhalten des Algorithmus untersucht, wenn man ihn auf eine Matrix anwendet, wie man sie in der Computeralgebra beim L¨osen gew¨ohnlicher Differentialgleichungen antrifft.

Nach ersten Tests zeichnete sich ab, daß die Laufzeiten auch bei ¨ahnlich konfigurierten Matrizen stark variieren k¨onnen, vor allem aufgrund unterschiedlicher Anzahl von F¨allen. Es wurde daher f¨ur die weiteren Untersuchungen eine Menge C von 540 zuf¨allig erzeugten Matrizen (Corpus) fixiert. Der Corpus C wurde so gew¨ahlt, daß etwa die H¨alfte der in ihm enthaltenen Matrizen in einer angemessenen Zeit gel¨ost werden k¨onnen.

Zum Inhalt des Corpus. Es werden Matrizen ¨uber der Menge $\{-9,-8,\ldots,8,9\} \subseteq \mathbb{Z}$ betrachtet, die in Gr¨oße, Zahl der Nullen, Zahl der Polynome, Zahl der Variablen und Maximalzahl der Monome bzw. Maximalgrad der polynomiellen Eintr¨age variieren. Alle Matrizen im Corpus sind vom Format $n \times n$ mit $n = 4,5,6$, enthalten 2, $2n$ oder $n^2/2$ polynomielle Eintr¨age in bis zu drei Variablen, wobei die maximale Zahl der Monome und der Grad zwischen 1 und 5 variieren. Meist wurden von den nichtsymbolischen Eintr¨agen eine H¨alfte mit Konstanten und die andere H¨alfte mit Nullen belegt, manche Matrizen sind voll besetzt. Die genaue Zusammensetzung des Corpus wird im Anhang (S. 59) ausf¨uhrlicher beschrieben.

Alle Experimente wurden mit MuPAD 2.0 f¨ur Solaris auf einer Sparc Ultra 5 mit 192 MB durchgef¨uhrt. Bei einer Laufzeitbeschr¨ankung von 450 s und einer Speicherbeschr¨ankung von 20 MB (jeweils pro Matrix) k¨onnen 41.9% der Matrizen gel¨ost werden, bei 36.2% scheitert eine L¨osung am Speicherlimit und bei den restlichen 21.9% reicht die Zeit nicht.[8]

[8] Ohne Speicherbeschr¨ankung k¨onnen genau die gleichen Matrizen gel¨ost werden wie mit Be-

Abbildung 2 zeigt, wie häufig die Laufzeit für ein gelöstes Problem zwischen $2^{(i-1)}$ und 2^i ms liegt.[9]

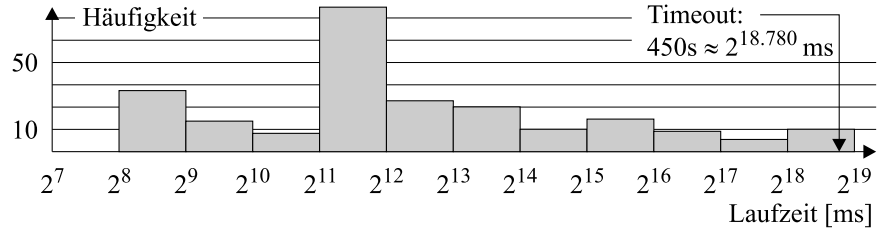

Abbildung 2 Logarithmisches „Histogramm" der Laufzeiten

Es zeigt sich, daß die meisten Matrizen sehr schnell fertig werden, während einige wenige wesentlich länger brauchen. Dies wird auch deutlich, wenn man das arithmetische Mittel der Laufzeiten (32 s) mit dem Median (4.3 s) vergleicht.

Die Laufzeit für ein bestimmtes Problem hängt dabei weniger von der äußeren Gestalt des Systems als von der Größe der Lösung (d. h. der Zahl der gefundenen Fälle) ab. Der Grund hierfür wird in Abbildung 3 deutlich: Da die Berechnung von Gröbnerbasen EXPSPACE-vollständig ist, während der Rest des Algorithmus „nur" polynomiell wächst, haben die Basisberechnungen einen entscheidenden Anteil an der Effizienz des gesamten Algorithmus. Die Größe der Fallunterscheidung hängt natürlich eng mit der Zahl der durchgeführten Basisberechnungen zusammen. Insbesondere werden bei Problemen, für die die Lösung in einem einzigen Fall präsentiert wird, überhaupt keine Gröbnerbasen berechnet.[10]

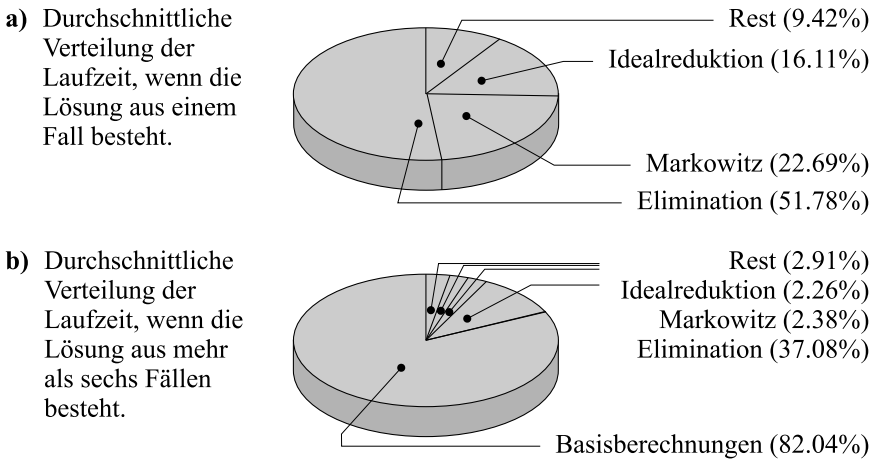

Abbildung 3 Verteilung der Laufzeit

schränkung. Jedes Problem, das zur Lösung mehr als 20 MB braucht, braucht also auch länger als 450 s.

[9] Alle Zahlen und Diagramme in diesem Abschnitt beziehen sich nur auf die Matrizen aus dem Corpus.

[10] Dies gilt nur, wenn – wie hier – keine Bedingungen vorgegeben werden.

Die bisherigen Beobachtungen legen also die ungew¨ohnliche Methode nahe, die Leistungsf¨ahigkeit des L¨osungsalgorithmus in Abh¨angigkeit von der L¨osung statt von der Eingabe zu betrachten.

Die folgende Abbildung stellt gewissermaßen einen Zusammenhang zwischen den Abbildungen 2 und 3 dar. Sie zeigt, wie h¨aufig welche Anzahl von F¨allen als L¨osung vorkommt.

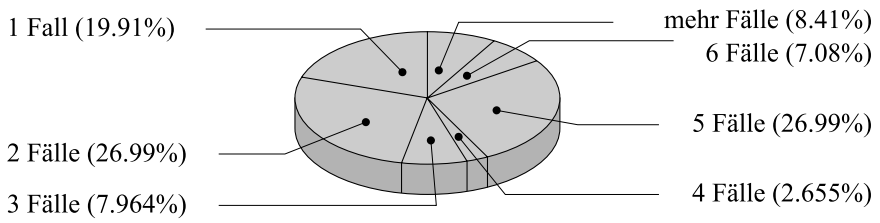

Abbildung 4 H¨aufi gkeit bestimmter Anzahlen von F¨allen

Interessanterweise gibt es recht wenige Probleme mit vier F¨allen. Vermutlich ist dies auf die spezielle Zusammensetzung des Corpus zur¨uckzuf¨uhren und nicht eine Eigenheit des Algorithmus. Problem 1 ist ein solches Exemplar.[11] Bei den angegebenen Resourcenbeschr¨ankungen ist die Zahl der F¨alle einer L¨osung maximal 10. Ein Beispiel f¨ur ein Problem, dessen L¨osung in 10 F¨allen pr¨asentiert wird, ist Problem 50.

Weitere Kenngr¨oßen, f¨ur die man sich interessieren k¨onnte, sind im Anhang ab Seite 59 tabellarisch zusammengestellt. Dabei werden diese Gr¨oßen sowohl in Abh¨angigkeit von der Gestalt der Matrix als auch in Abh¨angigkeit von der Zahl der F¨alle dargestellt.

Um den Einfluß der einzelnen Strategien auf das L¨osungsverhalten zu untersuchen, wurden die Matrizen des Corpus mit zwei Varianten des Algorithmus gel¨ost.

In der ersten Variante wurde bei der Pivot-Wahl auf die Verwendung des Markowitz-Kriteriums verzichtet. Die Zahl der Matrizen, die noch innerhalb der Resourcenbeschr¨ankung gel¨ost werden k¨onnen, sinkt in dieser Variante gegen¨uber dem Standardverfahren um ca. 30%. Der symbolische Fill-in steigt auf das Doppelte, der Fill-in an Konstanten gar auf das Dreifache. Die durchschnittliche Laufzeit steigt von 6216 ms auf 17070 ms, w¨ahrend die Zahl der gefundenen F¨alle nur geringf¨ugig steigt.[12]

Die zweite Variante verwendet das Markowitz-Kriterium wieder, verzichtet aber auf die Verwendung der Spaltenvereinfachung. Der Vergleich zeigt, daß die Spaltenvereinfachung f¨ur zuf¨allige Matrizen nur wenig bringt. Zusammenfassend l¨aßt sich sagen: Es k¨onnen etwa gleich viele Probleme gel¨ost werden, wobei die Zahl der gefundenen F¨alle sowie die durchschnittliche Laufzeit ohne Spaltenvereinfachung geringf¨ugig h¨oher sind. Diese Beobachtung ¨uberrascht wenig, wenn man sich erinnert, daß die

[11] Hier ist der Index des Problems im Corpus gemeint. Die Corpuselemente, auf die in der Arbeit bezug genommen wird, sind im Anhang ab Seite 65 abgedruckt. Der vollst¨andige Corpus ist in elektronischer Form (d. h. als MuPAD- oder AXIOM-Datei) beim Autor erh¨altlich.

[12] In diesem Vergleich wurden nur die Probleme betrachtet, die in beiden Varianten gel¨ost werden konnten.

Spaltenvereinfachung insgesamt nur selten aufgerufen wird (nur bei durchschnittlich 9.0% aller Pivotsuchen).

Die Meßwerte der beiden Variationen sind ebenfalls im Anhang zu finden.

Ein weiteres Experiment beschäftigte sich mit der Frage, welchen Vorteil es bringt, wenn man bei der Lösung von Problemen bereits Nebenbedingungen vorgibt.

Hier weiß man zunächst nicht richtig, was man eigentlich messen soll. Wenn man nämlich Fälle vorgibt, die aus zufällig erzeugten Polynomen zusammengesetzt sind, dann werden diese aller Wahrscheinlichkeit nach gar nichts mit dem Problem zu tun haben, und folglich auch keinen Einfluß auf das Lösungsverhalten haben.[13]

Da zufällig erzeugte Testfälle also ausscheiden, wurden in einem ersten Durchlauf die Fälle als Eingabe verwendet, die der Standardalgorithmus als Ausgabe liefert. Die Verwendung dieser Fälle sollte eine für den Algorithmus günstige Wahl sein, da der gleiche Weg durch den Parameterraum möglich ist, den der Algorithmus auch ohne Vorgabe gewählt hätte. Bei den meisten Problemen kann erwartungsgemäß eine erheblich kürzere Laufzeit beobachtet werden, wie Abbildung 5 zeigt. Hier wird die Laufzeit des Standardalgorithmus der Laufzeit bei Vorgabe der Fälle gegenübergestellt. Jeder Punkt (x,y) entspricht einer Matrix, die der Standardalgorithmus in Zeit x löst, und für die die Summe der Laufzeiten bei Vorgabe der Fälle y ist.

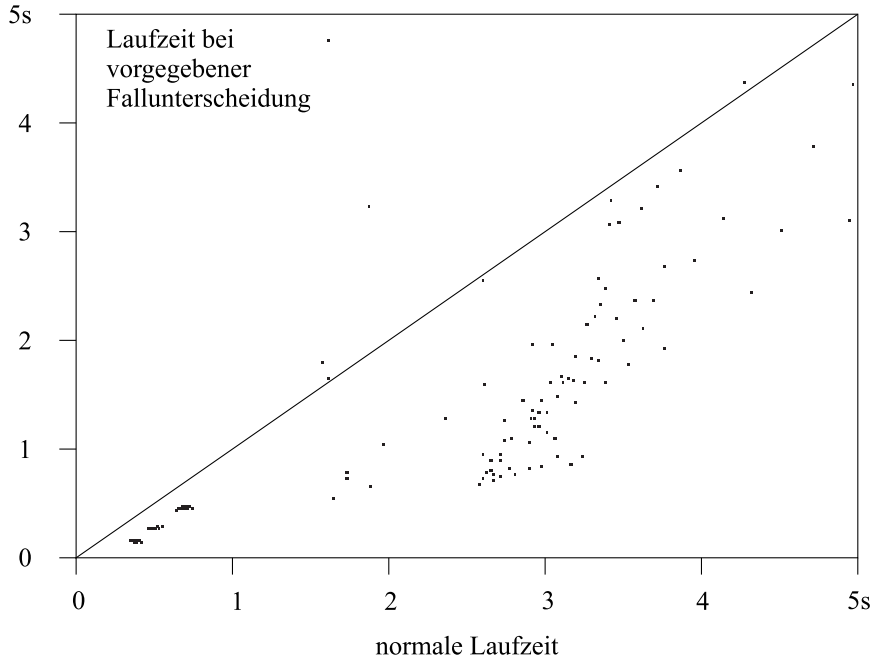

Abbildung 5 Gegenüberstellung der Laufzeiten des Gaußalgorithmus mit und ohne Vorgabe der Fallunterscheidung

[13] Genauer gesagt ist hier zu erwarten, daß sich die Laufzeit verschlechtert, da während der Lösungsfindung die gleiche Fallunterscheidung durchgeführt wird, aber durch die bereits vorhandenen Bedingungen die Größbnerbasisberechnungen aufwendiger werden.

Die Abbildung zeigt, daß in den meisten F¨allen erwartungsgem¨aß eine beachtliche Beschleunigung erzielt werden kann. Vergleicht man die durchschnittliche Laufzeit des Standardalgorithmus mit der Laufzeit bei Vorgabe eines einzigen Falls, so ist letztere um ca. 60% schneller.

Überraschenderweise gibt es aber auch Probleme, f¨ur die selbst die Vorgabe der eigenen F¨alle ung¨unstig ist. So kann Problem 32 vom Standardalgorithmus in 7380 ms gel¨ost werden, gibt man jedoch den vierten der f¨unf angegebenen F¨alle vor, so werden 17100 ms zu dessen Bearbeitung gebraucht, w¨ahrend der Aufwand bei Vorgabe eines der vier anderen F¨alle mit 1090, 750, 2740 und 740 ms im erwarteten Rahmen bleibt.

Obwohl man erwarten k¨onnte, daß bei Vorgabe eines Falls, der vom Algorithmus selbst in einer L¨osung angegeben wurde, der Parameterraum hinreichend eingeschr¨ankt ist, um die L¨osung des partiellen Problems in einem einzigen Fall anzugeben, ist dies nicht immer zu beobachten. Es kann n¨amlich Eintr¨age geben, die f¨ur den Standardalgorithmus zun¨achst symbolisch sind und erst bei einer sp¨ater erfolgenden Verzweigung als konstant angenommen werden. Wenn dieser Eintrag aber sofort relativ konstant ist, kann dies eine andere Pivotwahl zur Folge haben, so daß die weiteren Bedingungen im schlimmsten Fall unbrauchbar werden. Im Corpus wird aus diesem Grund bei sieben Problemen eine zus¨atzliche Verzweigung n¨otig, so daß die L¨osung in zwei F¨allen angegeben wird. Als Beispiel sei Problem 61 genannt. Kurioserweise ist dies aber *nicht* die Ursache f¨ur das Ph¨anomen, das oben am Problem 32 illustriert wurde. Tats¨achlich ist n¨amlich bei sechs der sieben Probleme die Summe der Laufzeiten bei Vorgabe der F¨alle trotz der zus¨atzlichen Verzweigung geringer als die entsprechende Laufzeit ohne vorgegebenen F¨allen.

Da die genannten Ph¨anomene als Einzelerscheinungen aufgefasst werden sollten, l¨aßt sich zusammenfassend feststellen, daß bei den meisten Problemen eine beachtliche Beschleunigung erzielt werden kann, wenn man die Betrachtung auf einen Teil des Parameterraums einschr¨ankt, der beim Standardalgorithmus als Teil der L¨osung erscheint. Das war aber zu erwarten.

Interessanter als diese *best case* Untersuchung ist daher das folgende Experiment. Um an brauchbare Fallunterscheidungen f¨ur die Probleme im Corpus zu gelangen, die sich noch dazu erheblich von den F¨allen unterscheiden, die Algorithmus 1 liefert, wurden die Matrizen des Corpus zun¨achst mit dem Algorithmus von Sit gel¨ost. Ein ausf¨uhrlicher Vergleich der beiden Algorithmen ist Inhalt des n¨achsten Abschnittes.

Die Fallunterscheidungen, die Sits Algorithmus liefert, unterscheiden sich enorm von jenen, die unser Algorithmus liefert (vgl. auch S. 51), h¨angen dabei aber nat¨urlich dennoch mit den entsprechenden Problemen zusammen. Was passiert also, wenn man diese F¨alle als Vorgaben f¨ur unseren Algorithmus verwendet?

Bei realistischen Zeit- und Speicherbeschr¨ankung l¨ost Sits Algorithmus 173 Corpuselemente.[14] Wenn man die F¨alle dieser L¨osungen als Eingabe f¨ur Algorithmus 1 verwendet, k¨onnen bei einer Zeitbeschr¨ankung von 5 min (pro Fall) noch 99 Probleme vollst¨andig bearbeitet werden (d. h. f¨ur jeden Fall, den Sits L¨osung f¨ur das entsprechende Problem angibt). Diese Probleme werden in der Standardkonfiguration mit ei-

[14] Genauer: Sit l¨ost 197 Probleme in weniger als f¨unf Minuten, aber bei 24 Problemen dauert die Erzeugung der ConstraintStore-Objekte in MuPAD l¨anger als f¨unf Minuten

ner durchschnittlichen Laufzeit von 2282 ms und durchschnittlich 2.17 F¨allen gel¨ost. Summiert man analog zu Abbildung 5 die Laufzeiten f¨ur die verschiedenen F¨alle eines Problems auf, so kommt man auf eine durchschnittliche Summe von 4890 ms. Es scheint also zun¨achst doch g¨unstiger zu sein, im Allgemeinen auf die Vorgabe von F¨allen zu verzichten und die interessierenden F¨alle aus der L¨osung des Standardalgorithmus auszuw¨ahlen.

Wenn man von den trivialen Problemen, f¨ur die die L¨osung in einem Fall darstellbar ist, absieht, ist Problem 1 das einzige Corpuselement, f¨ur das die Vorgabe der Sitschen F¨alle eine Beschleunigung bringt. Dies ist auch das einzige Problem, das keine zus¨atzlichen Verzweigungen durchf¨uhrt, obwohl Sit weniger F¨alle (n¨amlich 3) liefert als der Standardalgorithmus (n¨amlich 4).

Da die Stichprobe durch die n¨otige Zeitbeschr¨ankungen stark dezimiert wurde, wurde das Experiment mit einer gr¨oßeren Stichprobe bei niedrigerer Zeitschranke wiederholt. Aus einer Menge von ca. 13 000 Problemen, deren Struktur der des Corpus entspricht, konnten etwa 2 600 Probleme in der vorgegebenen Zeit von beiden Algorithmen bearbeitet werden. Abbildung 6 zeigt, daß die Verh¨altnisse doch nicht so schlecht sind wie zun¨achst vermutet. Ein Vergleich mit der Punktwolke aus Abbildung 5 zeigt, daß die Regionen mit der h¨ochsten Dichte einander entsprechen. Der markanteste Unterschied l¨aßt sich als „Verschmierung“ der rechten Wolke in Richtung der h¨oheren Laufzeit charakterisieren.[15]

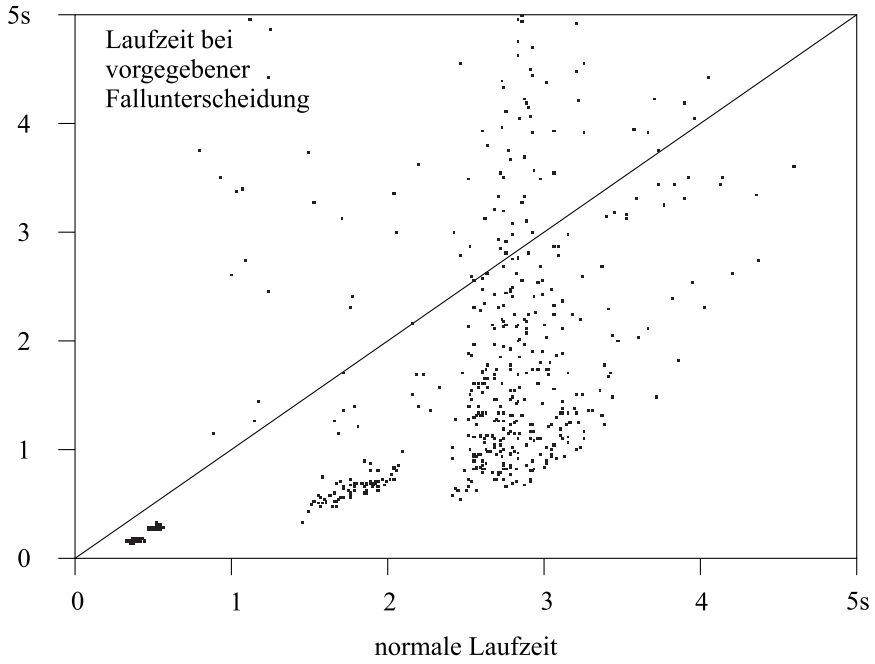

Abbildung 6 Gegen¨uberstellung der Laufzeiten des Gaußalgorithmus mit und ohne Vorgabe der Fallunterscheidung, die Sits Algorithmus als Ausgabe liefert

[15] Aus Platzgr¨unden wurde darauf verzichtet, auch das Analogon zu Abbildung 5 f¨ur die gr¨oßere Stichprobe abzudrucken, da dieses tats¨achlich die gleiche Verteilung wie Abbildung 5 zeigt.

4.3 Ein Vergleich mit Sits Algorithmus

Wie bereits in der Einführung zu Beginn der Arbeit angesprochen, wurde zur Lösung parametrischer linearer Gleichungssysteme bereits in [Sit92] ein Algorithmus angegeben, der auf der Cramerschen Regel beruht. Für $C \in k(x_1,\ldots,x_r)^{n\times m}, A \in k(x_1,\ldots,x_r)^m$ wird das System

$$C \cdot x = A$$

betrachtet. Seien C, x und A von der Gestalt

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$$

mit $C_{11} \in k[x_1,\ldots,x_r]^{c\times c}$ und $\det C_{11} \neq 0$. Dann ist

$$T = \begin{pmatrix} C_{11}^{-1} \cdot A_1 & -C_{11}^{-1} C_{12} \\ 0 & I \end{pmatrix}$$

eine Treppennormalform von C für den Fall

$$\begin{aligned} F_C := &\{\det C_1 1 \neq 0, \det(C_{21} \cdot C_{11}^{-1} \cdot A_1 - A_2) = 0\} \cup \\ &\{\det C' = 0 : C' \text{ ist } (c+1)\times(c+1)\text{-Untermatrix von } C\}. \end{aligned}$$

In einer von $c = \min\{m,n\}+1$ bis $c = 0$ laufenden Schleife bestimmt der Algorithmus alle Treppennormalformen dieser Gestalt, wobei C_{11} in jeder Iteration alle $c \times c$-Untermatrizen von C durchläuft, für die der Fall F_C konsistent ist.

Auf Details des Algorithmus und Verfeinerungen zur Verringerung von Fällen soll hier nicht eingegangen werden. Der Interessierte Leser sei auf [Sit92] verwiesen.

Sits Algorithmus wurde im Algebrasystem AXIOM implementiert, und es liegt nun nahe, die beiden (grundverschiedenen) Lösungsansätze miteinander zu vergleichen.

Ein ernsthafter Laufzeitvergleich scheidet aus zwei Gründen aus. Zum einen handelt es sich bei AXIOM und MuPAD um grundlegend verschiedene Architekturen, die insbesondere im Laufzeitverhalten stark voneinander abweichen. Zum anderen hatte Sit nach eigenen Angaben kein Interesse an einer *effizienten* Implementierung seines Algorithmus.

Statt der Laufzeit soll daher die Anzahl der Fälle pro Lösung verglichen werden. Nachdem Sit gezeigt hat, daß im *worst case* bei Verwendung des Gauß-Algorithmus eine exponentiell höhere Anzahl von Fällen zu erwarten ist als bei seinem Algorithmus (Theorem 9.1 [Sit92, S. 389]), eignet sich ein Vergleich der Lösungsmengen zur Bewertung der Verbesserung, die der Einsatz unserer Heuristiken gegenüber einer naiven Implementierung des Gauß-Algorithmus bewirkt.

Zum Vergleich wurden die 173 Corpuselemente betrachtet, die beide Algorithmen in angemessener Zeit lösen können. Das Ergebnis ist in Abbildung 7 dargestellt: Der Tabelleneintrag (i,j) (Zeile/Spalte) steht für die Anzahl der Matrizen, deren Lösung beim Gauß-Algorithmus i Fälle und bei Sits Algorithmus j Fälle enthält. (Der Übersichtlichkeit wegen wurde $\cdot$ anstelle von 0 geschrieben.)

Zahl der Fälle in Algorithmus 1 \ Zahl der Fälle bei Sit	1	2	3	4	5	6	7	8	9
1	45	·	·	·	·	·	·	·	·
2	·	49	8	1	·	·	·	·	·
3	·	7	7	2	·	·	·	·	·
4	·	·	2	2	·	2	·	·	·
5	·	·	3	23	8	1	·	·	·
6	·	·	·	1	4	1	1	·	·
7	·	·	·	·	1	1	2	·	·
8	·	·	·	·	·	·	·	1	·
9	·	·	·	·	·	·	·	·	1

Abbildung 7 Gegenüberstellung der Zahl der Fälle bei Sit und Algorithmus 1

Wie zu erwarten war, schneidet Sits Algorithmus in diesem Vergleich etwas besser ab. Allerdings unterscheidet die Zahl der Fälle in den beiden Algorithmen für kein Problem um mehr als zwei, was als Erfolg zu werten ist.

Ein Beispiel für eine Matrix, die Sits Algorithmus besser handhaben kann, ist Problem 44. Hier gibt Sits Algorithmus eine Lösung mit drei Fällen aus, während die Lösung unseres Algorithmus fünf Fälle umfaßt. Umgekehrt ist Problem 56 eine Matrix, mit der unser Algorithmus (4 Fälle) besser zurechtkommt als der von Sit (6 Fälle). Daß derartige Probleme überhaupt existieren, wurde so nicht erwartet.

4.4 Experimente mit „Matrizen aus dem Leben“

Die Matrix $M \in \mathbb{R}[x,y,z]^{8\times 6}$ sei definiert durch

$$M := \begin{pmatrix} t & 0 & 0 & 0 & z & -y \\ 0 & t & 0 & -z & 0 & x \\ 0 & 0 & t & y & -x & 0 \\ x & y & z & 0 & 0 & 0 \\ 0 & -z & y & t & 0 & 0 \\ z & 0 & -x & 0 & t & 0 \\ -y & x & 0 & 0 & 0 & t \\ 0 & 0 & 0 & x & y & z \end{pmatrix}$$

Für Belegungen $\sigma\colon \{x,y,z,t\} \to \mathbb{R}$ mit $\operatorname{Rang}\sigma(M) = 6$ ergeben sich hieraus die charakteristischen Lösungen der Maxwellschen Differentialgleichungen. Auf den Zusammenhang zwischen partiellen Differentialgleichungen und parametrisierten linearen Gleichungssystemen kann an dieser Stelle nicht näher eingegangen werden. Der interessierte Leser sei auf [Hör76, RR93, DL88] verwiesen. Für die Zwecke dieser Arbeit genügt zu wissen, daß man sich für die Spezialisierungen σ interessiert, für die das zu einer Differentialgleichung (bzw. zu einem Diffenrentialgleichungssystem) gehörende lineare Gleichungssystem maximalen Rang hat.

Algorithmus 1 benötigt zur Lösung des durch M gegebenen Systems etwas weniger als 10 s und liefert eine Lösung, die die folgenden zwölf Fälle unterscheidet:

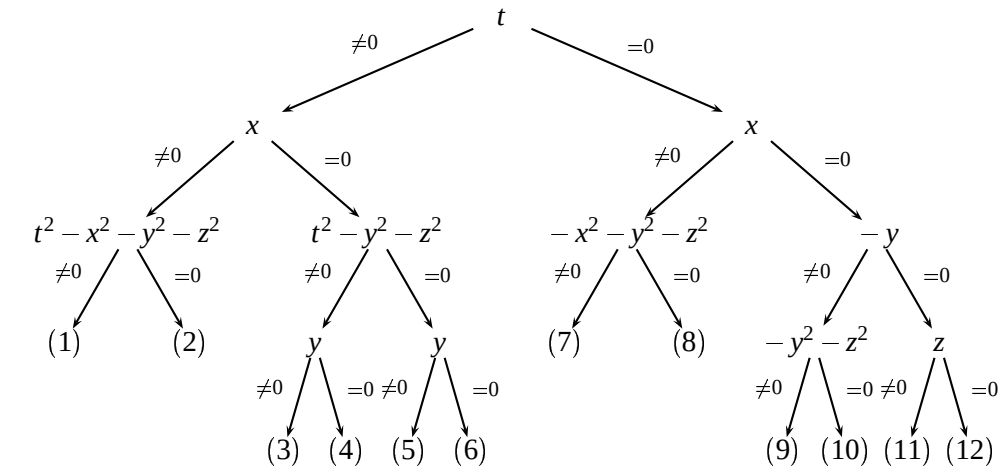

Abbildung 8 L¨osungsraum des Systems *M*

Die Berechnung der L¨osungsmenge entspricht einer Tiefensuche in diesem Baum. Betrachtet man die (hier nicht abgedruckten) Treppennormalformen, erh¨alt man den Rang 6 in den F¨allen 1, 3, 4, 7, 9 und 11, in den F¨allen 2, 5, 6, 8 und 10 erh¨alt man den Rang 4 und im Fall 12 die Nullmatrix. Eine genaue Inspektion von Abbildung 8 zeigt, daß *M* genau dann den Rang 6 hat, wenn

$$t^2 - x^2 - y^2 - z^2 \neq 0$$

ist. Genau dies wird von der Theorie vorhergesagt.

Man kann den Einfluß der in Kapitel 2 beschriebenen Verfeinerungen durch eine Gegen¨uberstellung mit einer naiven Implementierung erkennen, wie dies schon in Abschnitt 4.2 geschehen ist. Es zeigt sich dann, daß ohne die vorgestellten Strategien die L¨osung des Systems etwa sechsmal so viel Zeit in Anspruch nimmt und dabei die L¨osungsmenge fast doppelt so viele F¨alle umfaßt, die zudem in recht redundanter Form dargestellt werden.[16]

Entscheidenden Einfluß an der geringeren Laufzeit und der geringeren Zahl der F¨alle hat wie auch f¨ur zuf¨allig erstellte Matrizen das erweiterte Markowitz-Kriterium. Ohne seine Verwendung steigt die Zahl der F¨alle von 12 auf 19 und die Laufzeit um etwa 50%. Die Spaltenvereinfachung dagegen erzielt generell schlechtere Ergebnisse bei Matrizen mit vielen Variablen, da die Wahrscheinlichkeit f¨ur einen Match zweier Leitterme mit der Zahl der Variablen sinkt (vgl. S. 23 und S. 60). Immerhin hat die L¨osung bei Verwendung der Spaltenvereinfachung zwei F¨alle weniger als ohne und wird dabei sogar 5% schneller gefunden.

Die Idealreduktion zur Vereinfachung der Matrixeintr¨age nach einer Verzweigung hat haupts¨achlich Einfluß auf die Darstellung der F¨alle in der L¨osung, tr¨agt aber auch zu einer Beschleunigung (um ca. 35%) und zu einer Verringerung der Zahl der F¨alle (um zwei) bei.

[16] Man beachte, daß die einzelnen F¨alle in Abbildung 8 „optimal“ dargestellt sind, obwohl F¨alle etwa der Form $\{x \neq 0, x \cdot t = 0\}$ nicht selbst¨andig zu $\{x \neq 0, t = 0\}$ vereinfacht werden k¨onnen, da $x \cdot t \to t$ von der Idealvereinfachung nicht geleistet wird.

Trotz allem ist die gefundene L¨osung nicht optimal. Wie bereits festgestellt, l¨aßt Abbildung 8 bereits vermuten, daß neben einzelnen Variablen vor allem das Polynom

$$p := t^2 - x^2 - y^2 - z^2$$

offenbar eine entscheidende Rolle spielt. Alle nichttrivialen Polynome sind Spezialierungen von p. Unter Verwendung von

$$z = 0 \iff z^2 = 0 \qquad \text{und} \qquad z \neq 0 \iff z^2 \neq 0$$

kann man auch die Verzweigung nach z durch eine Verzweigung nach p ersetzen. Durch Umordnung erh¨alt man folgende Darstellung des Parameterraums, wobei das Dreieck stets f¨ur den entsprechenden Unterbau auf der linken Seite steht.

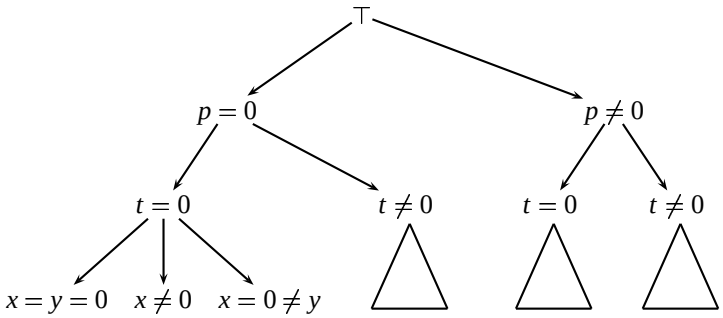

Abbildung 9 Optimierte Darstellung des Parameterraums von M

Die drei Unterf¨alle, die durch das linke Dreieck im rechten Unterbaum symbolisiert sind, lassen sich zu einem zusammenfassen:

Satz 12 In der Bezeichnungsweise dieses Abschnitts gilt: Im Fall $\{p \neq 0, t = 0\}$ hat M die Treppennormalform $\mathrm{diag}(p, p, p, p, p, p)$.[17]

Bew. Um zugleich $p = t^2 - x^2 - y^2 - z^2 \neq 0$ und $t = 0$ zu erf¨ullen, muß gelten

$$x \neq 0 \vee y \neq 0 \vee z \neq 0.$$

Da M im Fall $t = 0$ durch Umordnen der Zeilen und Weglassen von Nullzeilen auf die Form

$$\begin{pmatrix} 0 & -M' \\ M' & 0 \end{pmatrix}$$

gebracht werden kann, gen¨ugt es, die Untermatrix M' zu betrachten.

[17] Die Aussage dieses Satzes ist nat¨urlich trivial. Im Fall $p \neq 0$ (also sogar noch allgemeiner als $\{p \neq 0, t = 0\}$) hat M vollen Rang, also gilt $M \rightsquigarrow I$ und Multiplikation jeder Zeile mit p f¨uhrt zur verlangten Gestalt. Es kommt hier darauf an zu erkennen, warum der Algorithmus das nicht „sieht“.

1. Fall: $x \neq 0$. Hier gilt:

$$\begin{pmatrix} x & y & z \\ 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \overset{\cdot(-x),\ +z,\ +y}{\rightsquigarrow} \begin{pmatrix} p & 0 & 0 \\ 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix} \overset{-z/p,\ y/p,\ +}{\rightsquigarrow}$$

$$\rightsquigarrow \begin{pmatrix} p & 0 & 0 \\ 0 & -z & y \\ 0 & 0 & -x \\ 0 & x & 0 \end{pmatrix} \overset{+,\ \cdot(-p/x),\ \cdot p/x,\ z/p}{\rightsquigarrow}$$

$$\rightsquigarrow \begin{pmatrix} p & 0 & 0 \\ 0 & p & 0 \\ 0 & 0 & p \end{pmatrix}.$$

Die F¨alle $y \neq 0$ und $z \neq 0$ sind zu Fall 1 symmetrisch und gehen analog. ∎

Es bestehen also zwei Schwierigkeiten: Zum einen muß die erste Zeile vor der Elimination „verkompliziert“ werden, wozu die Kenntnis der herausragenden Rolle des Polynoms p erforderlich ist. Die andere Schwierigkeit besteht darin, eine Fallunterscheidung derart durchzuf¨uhren, daß die Treppennormalformen am Schluß ¨ubereinstimmen und die F¨alle zusammengefaßt werden k¨onnen.

Solch komplexe Überlegungen lassen sich nat¨urlich schwerlich mit heuristischen Ans¨atzen steuern. Hier scheint der Algorithmus von Sit (vgl. Abschnitt 4.3) besser abzuschneiden, denn in der Tat pr¨asentiert Sit die L¨osung von M in zehn F¨allen, die sich in folgendem Baum darstellen lassen:

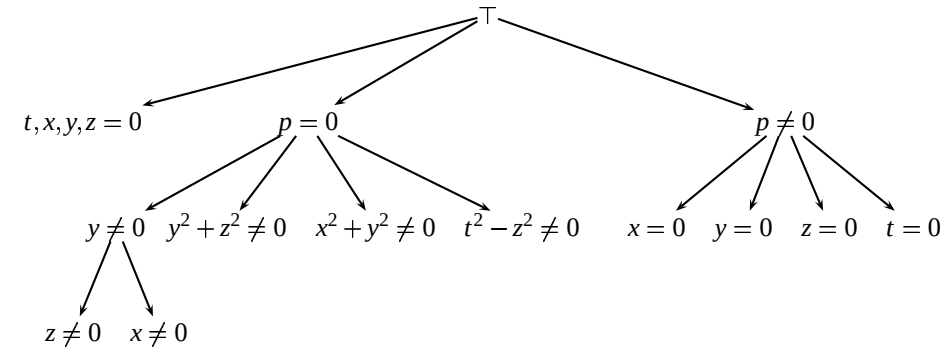

Abbildung 10 Parameterraum von M nach Sits Algorithmus

Es soll aber nicht verschwiegen werden, daß auch die von Sits Algorithmus angegebene L¨osung nicht optimal ist. So wird die Bedingung $p = 0$ in drei F¨allen dargestellt als

$$t^3 - tx^2 - ty^2 - tz^2 = 0 \wedge t^2x - x^3 - xy^2 - xz^2 = 0$$
$$\wedge\ t^2y - x^2y - y^3 - yz^2 = 0 \wedge t^2z - x^2z - y^2z - z^3 = 0,$$

also

$$p \cdot t = 0 \wedge p \cdot x = 0 \wedge p \cdot y = 0 \wedge p \cdot z = 0,$$

was äquivalent ist zu $p = 0 \vee (t = x = y = z = 0)$. Da $t = x = y = z = 0$ bereits in Fall 1 erschöpfend behandelt wurde, geht es also gerade um den Fall $p = 0$.

5 Ergebnisse

Es wurde gezeigt, daß sich der Gauß-Algorithmus auf parametrische lineare Gleichungssysteme ¨ubertragen l¨aßt. Durch eine geschickte Strategie zur Pivot-Wahl und mit Hilfe zus¨atzlicher Zwischenschritte zur Vereinfachung der Matrix l¨aßt sich der Suchraum entscheidend einschr¨anken, was sich in einer akzeptablen Laufzeit und in einer kurzen Darstellung der L¨osung (d. h. wenige F¨alle) ¨außert.

Zentrale Entwurfsentscheidung war die Trennung der eigentlichen Elimination von der Verwaltung des Parameterraums, so daß das Konzept der F¨alle in einem unabh¨angigen Modul untergebracht ist und prinzipiell auch f¨ur andere Probleme mit ¨ahnlichen Fallunterscheidungen geeignet ist.

Die Architektur wurde in MuPAD implementiert und auf zuf¨allig erzeugten Matrizen getestet. In einem statistischen Vergleich mit dem Algorithmus von Sit, f¨ur den Laufzeit und Zahl der gefundenen F¨alle asymptotisch exponentiell besser sind als beim „normalen“ Gauß-Algorithmus, zeigte sich, daß durch die Verwendung der vorgestellten Heuristiken in den meisten F¨allen eine beachtliche Leistungssteigerung erzielen l¨aßt.

Abh¨angig davon, ob man mehr Wert auf die Minimalit¨at der L¨osung oder eine kurze Laufzeit legt, wird man sich f¨ur den einen oder anderen Algorithmus entscheiden. Auch wurde beobachtet, daß eine L¨osung, die mit dem Gauß-Algorithmus gefunden wird, zwar mehr F¨alle umfaßt als die entsprechende L¨osung von Sits Algorithmus, daß die F¨alle selbst aber in den meisten F¨allen erheblich kompakter und weniger redundant dargestellt werden. Zus¨atzlich ist die Fallunterscheidung, die der Gauß-Algorithmus liefert, nach Konstruktion disjunkt, w¨ahrend Sits F¨alle sich ¨uberlappen k¨onnen und teilweise gar ineinander enthalten sind.

Der entscheidende Vorteil des parametrischen Gauß-Algorithmus ist jedoch eine zu¨atzliche Funktionalit¨at, die im Sitschen Ansatz nicht m¨oglich ist: Wenn man sich nur f¨ur einen gewissen Teil des Parameterraums interessiert, kann man durch Vorgabe dieses Teils den Suchraum f¨ur die L¨osung a priori extrem einschr¨anken. Bei derartigen partiellen Fragestellungen ist der Gauß-Algorithmus vorzuziehen, da sonst mit einem aufwendigen Verfahren aus der kompletten L¨osung die relevanten F¨alle ausgew¨ahlt werden m¨ußten, was (neben der unn¨otigen Berechnung der nicht interessierenden F¨alle) zus¨atzliche Zeit in Anspruch nimmt und eine kompakte Darstellung der F¨alle selbst nicht zu erwarten ist.

Zum Abschluß sollen einige Fragen aufgeworfen werden, die sich w¨ahrend der Erstellung der Studienarbeit gestellt haben, die aber nicht weiter verfolgt werden konnten:

- Wie l¨aßt sich der Algorithmus ¨ubertragen auf K¨orper, die nicht algebraisch abgeschlossen sind? Ist es z. B. m¨oglich, $\mathbb{R} \subseteq \mathbb{C}$ durch einen Fall zu charakterisieren, der Bedingungen wie $x^2 + 1 \neq 0$ enth¨alt, die in $\mathbb{R}$ tautologisch sind?

- Welche Verfahren sind denkbar, mit denen die gefundene L¨osung nachtr¨aglich vereinfacht werden kann, etwa durch Zusammenfassung von F¨allen? Wie kann man w¨ahrend der Elimination unterst¨utzen, daß F¨alle entstehen, die sich zusammenfassen lassen (vgl. Satz 12)?

- Die Vorgabe von Bedingungen kann zu einer bedeutenden Verlangsamung der Berechnung f¨uhren, wenn die vorgegebenen Bedingungen zur L¨osung des Problems nicht n¨utzlich sind und somit nur die Basisberechnungen verteuern. Kann man a priori feststellen, ob es sinnvoll ist, eine oder mehrere der vorgegebenen Bedingungen zu l¨oschen? K¨onnten hierzu Methoden aus dem Automatischen Beweisen wie Tautologiel¨oschung, L¨oschung purer Literale oder Subsumtionstests adaptiert werden?

- Welche M¨oglichkeiten gibt es, auch das Konstantenwissen zur Vereinfachung von Polynomen heranzuziehen? Ist $x = 0$, so ist $x + y = y$, aber wenn man $x \neq 0$ weiß, kann man damit weder $x + y$ noch $x \cdot y$ vereinfachen. Wir verwenden Konstantenwissen durch Ausnutzung der Äquivalenzen

  $$x \cdot y = 0 \iff y = 0 \quad \text{und} \quad x \cdot y \neq 0 \iff y \neq 0,$$

  wozu eine Faktorisierung n¨otig ist (vgl. S. 18). L¨aßt sich das Faktorisieren umgehen?

- Welche Probleme treten bei der Übertragung auf nicht-homogene parametrische lineare Gleichungssysteme auf, und welche der gefundenen Strategien lassen sich auf Systeme polynomieller Gleichungen ¨ubertragen?

- Lassen sich Heuristiken finden, mit denen Fallunterscheidungen gefunden werden, die „interessante“ Polynome enthalten? Dazu ist nat¨urlich zun¨achst zu kl¨aren, welche Polynome interessant sind. Zum Beispiel w¨urde man das Polynom p aus Abschnitt 4.4 als interessant bezeichen.

Mathematische Grundlagen

Dieses Kapitel ist eine weitgehend unzusammenh¨angende Auflistung an Definitionen und Aussagen, die an verschiedenen Stellen der Arbeit vorausgesetzt werden. Es soll haupts¨achlich darum gehen, die Sprech- und Schreibweisen festzulegen und fundamentale Resultate bereitszustellen.

Begriffe, die hier nicht aufgef¨uhrt sind, werden in ihrer allgemein ¨ublichen oder intuitiven Bedeutung verwendet. Insbesondere ist die Notation von Aussagen (wie ¨ublich) an die Syntax der Pr¨adikatenlogik angelehnt. Wo es sinnvoll erscheint, wird $A(x)$ $(x \in B)$ anstelle von $\forall\, x \in B : A(x)$ geschrieben.

Def. 9 Ein K¨orper k heißt *algebraisch abgeschlossen,* wenn jedes Polynom $p \in k[x]$ mit $\partial p > 0$ eine Nullstelle in k hat.

In dieser Arbeit ist k stets ein algebraisch abgeschlossener K¨orper.

Def. 10

1. Seien $r \in \mathbb{N}$ und x_i *Variablen* $(i = 1, \ldots, r)$. Weiter sei $e \in \mathbb{N}_0^r$. Dann heißt ein Ausdruck der Form

   $$t := \prod_{i=1}^{r} x_i^{e_i}$$

   ein *Term* in $x_1, \ldots, x_r$, und

   $$\partial t := \sum_{i=1}^{r} e_i$$

   heißt der *Grad* des Terms t.

2. Sei t ein Term und $a \in k$. Dann heißt der Ausdruck $m := a \cdot t$ *Monom* in $x_1, \ldots, x_r$, a heißt *Koeffizient* von m, und $\partial m := \partial t$ heißt *Grad* von m.

3. Sind $m_1, \ldots, m_j$ Monome mit paarweise verschiedenen Termen, so heißt ein Ausdruck der Form

   $$p := \sum_{i=1}^{j} m_i$$

   Polynom ¨uber $x_1, \ldots, x_r$, und

   $$\partial p := \max_{1 \le i \le j} \partial m_i$$

   heißt *Grad* von p. Mit $|p| := k$ sei die Anzahl der Monome in p bezeichnet.

4. Es sei $k[x_1,\dots,x_r]$ die Menge aller Polynome über $x_1,\dots,x_r$. Multiplikation, Addition, Teilbarkeit und – wenn möglich – Division seien auf $k[x_1,\dots,x_r]$ wie üblich definiert.

5. Sind $p,q \in k[x_1,\dots,x_r]$, $p \neq 0$, dann heißt ein Ausdruck der Form $r := p/q$ *rationale Funktion*. Es sei $k(x_1,\dots,x_r)$ der Körper aller rationalen Funktionen über $x_1,\dots,x_r$.

Def. 11

1. Eine Menge $\mathfrak{a} \subseteq k[x_1,\dots,x_r]$ heißt *Ideal*, wenn gilt

$$\forall\, a \in \mathfrak{a}\ \forall\, p \in k[x_1,\dots,x_r] : a \cdot p \in \mathfrak{a},$$
$$\forall\, a_1, a_2 \in \mathfrak{a} : a_1 + a_2 \in \mathfrak{a}.$$

   Schreibweise in diesem Fall: $\mathfrak{a} \trianglelefteq k[x_1,\dots,x_r]$.

2. Polynome $a_1,\dots,a_n \in k[x_1,\dots,x_r]$ heißen *Basis* oder *Erzeugendensystem* von $\mathfrak{a}$, wenn gilt

$$\forall\, a \in \mathfrak{a}\ \exists\, p_1,\dots,p_n \in k[x_1,\dots,x_r] : a = \sum_{i=1}^{n} p_i \cdot a_i.$$

   Schreibweise: $\mathfrak{a} = \langle a_1,\dots,a_n\rangle$.

3. Zu $\mathfrak{a} \trianglelefteq k[x_1,\dots,x_r]$ heißt

$$\operatorname{Rad}\mathfrak{a} := \{p \in k[x_1,\dots,x_r] : \exists\, m \in \mathbb{N} : p^m \in \mathfrak{a}\} \trianglelefteq k[x_1,\dots,x_r]$$

   das *Radikal* von $\mathfrak{a}$.

Def. 12

1. Seien $P \subseteq k[x_1,\dots,x_r]$. Dann heißt die Menge

$$\operatorname{Var}(P) := \{x \in k^r : p(x) = 0\ (p \in P)\}$$

   die *Varietät* von P.

2. Ist $V \subseteq k^r$ eine Varietät, so heißt

$$I(V) := \{p \in k[x_1,\dots,x_r] : p(x) = 0\ (x \in V)\} \trianglelefteq k[x_1,\dots,x_r]$$

   das von V erzeugte Ideal.

Satz 13 Es gelten:

1. (Hilberts Basissatz) Jedes Ideal $\mathfrak{a}$ hat ein endliches Erzeugendensystem.
2. Sind $\mathfrak{a}, \mathfrak{b} \trianglelefteq k[x_1, \ldots, x_r]$, dann ist auch $\mathfrak{a} \cap \mathfrak{b}$ ein Ideal, und es gilt $\operatorname{Rad}(\mathfrak{a} \cap \mathfrak{b}) = \operatorname{Rad}\mathfrak{a} \cap \operatorname{Rad}\mathfrak{b}$.
3. Für $P, Q \subseteq k[x_1, \ldots, x_r]$ gilt $\operatorname{Var}(P \cup Q) = \operatorname{Var}(P) \cap \operatorname{Var}(Q)$.
4. (Schwacher Nullstellensatz) Sei $\mathfrak{a} \trianglelefteq k[x_1, \ldots, x_r]$. Dann gilt
$$V(\mathfrak{a}) = \varnothing \iff 1 \in \mathfrak{a} \iff \mathfrak{a} = k[x_1, \ldots, x_r].$$
5. (Hilberts Nullstellensatz) Für jedes Ideal $\mathfrak{a} \trianglelefteq k[x_1, \ldots, x_r]$ gilt $I(V(\mathfrak{a})) = \operatorname{Rad}\mathfrak{a}$.
6. Ist $\mathfrak{a} = \langle a_1, \ldots, a_n \rangle \trianglelefteq k[x_1, \ldots, x_r]$ und $p \in k[x_1, \ldots, x_r]$, so gilt
$$p \in \operatorname{Rad}\mathfrak{a} \quad \iff \quad 1 \in \langle a_1, \ldots, a_n, 1 - yp \rangle \trianglelefteq k[x_1, \ldots, x_r, y].$$

Bew.

1. Theorem 4 in [CLO92, S. 75]
2. Poposition 9 und Proposition 16 in [CLO92, S. 185, 189]
3. Lemma 2 in [CLO92, S. 11]
4. Theorem 1 in [CLO92, S. 169]
5. Theorem 6 in [CLO92, S. 175]
6. Proposition 8 in [CLO92, S. 177] ∎

Def. 13 Auf der Menge T der Terme sei durch $\preccurlyeq$ eine Ordnung mit folgenden Eigenschaften gegeben (vgl. Def. 1 in [CLO92, S. 54]):

1. $\preccurlyeq$ ist eine Totalordnung.
2. Für alle Terme a, b, c gilt $a \preccurlyeq b \implies a + c \preccurlyeq b + c$.
3. Jede nicht-leere Teilmenge von T hat ein bezüglich $\preccurlyeq$ minimales Element.

$\preccurlyeq$ heißt *Termordnung*.

Ist $p \in k[x_1, \ldots, x_r]$, $p = a_1 t_1 + \cdots + a_n t_n$ $(a_i \in k, t_i \in T)$, und ist i_0 der Index des bezüglich $\preccurlyeq$ maximalen Terms in p, dann heißen

$$\operatorname{LT}(p) := t_{i_0}, \qquad \operatorname{LC}(p) := a_{i_0} \quad \text{und} \quad \operatorname{LM}(p) := a_{i_0} \cdot t_{i_0}$$

der *Leitterm*, *Leitkoeffizient* bzw. das *Leitmonom* von p (bezüglich $\preccurlyeq$).

Für zwei Polynome $p, q \in k[x_1, \ldots, x_r]$ gelte $p \preccurlyeq q :\iff \operatorname{LT}(p) \preccurlyeq \operatorname{LT}(q)$.

Meßwerte

Zusammensetzung des Corpus

Der Corpus enthält 540 Matrizen, die sich in Größe, Zahl der Variablen, Komplexität der Polynome und symbolischer und konstanter Dichte unterscheiden. Die folgende Tabelle zeigt, wie viele Matrizen welcher Gestalt im Corpus enthalten sind.

Ein Polynom heißt dabei von Komplexität *k*, wenn es bis zu *k* Monome enthält und einen maximalen Grad von *k* hat. Koeffizienten der Polynome sowie konstante Einträge wurden zufällig (gleichverteilt) aus $\{-9,-8,\ldots,-1,1,\ldots,9\}$ gezogen. Die Null tritt nicht als spezielle Konstante auf. Konstanten sind keine speziellen Polynome. Die Polynome wurden mit der MuPAD-Funktion `polylib::randpoly` erzeugt.

Größe	Variablen	Komplexität	symb. Einträge	Nullen	Anzahl
4	1	2	8	4	20
5	1	2	10	7	20
6	1	2	12	12	20
4	2	2	8	4	20
5	2	2	10	7	20
6	2	2	12	12	20
4	3	2	8	4	20
5	3	2	10	7	20
6	3	2	12	12	20
4	2	2	8	4	15
5	2	2	10	7	15
6	2	2	12	12	15
4	2	3	8	4	15
5	2	3	10	7	15
6	2	3	12	12	15
4	2	4	8	4	15
5	2	4	10	7	15
6	2	4	12	12	15
4	2	5	8	4	15
5	2	5	10	7	15
6	2	5	12	12	15
4	2	2	2	0	15
5	2	2	2	0	15
6	2	2	2	0	15
4	2	2	8	0	15
5	2	2	10	0	15
6	2	2	12	0	15
4	2	2	2	8	15
5	2	2	2	10	15
6	2	2	2	12	15
4	2	2	4	4	15
5	2	2	7	7	15
6	2	2	12	12	15

Lösungsverhalten der Standardkonfiguration

Die nächste Tabelle enthält interessante Kenngrößen zum Lösungsverhalten im Corpus. Die rechte Spalte enthält jeweils die Kenngröße für den gesamten Corpus, während in der Mitte der Tabelle die Matrizen mit gleicher Zahl von Variablen (links) bzw. gleicher symbolischer Komplexität (rechts) betrachtet werden. Hier ist zu beachten, daß aufgrund der Asymmetrie des Corpus nicht für jede Konfiguration die gleiche Anzahl von Problemen zur Verfügung steht, z. B. enthält der Corpus 360 Matrizen mit zwei Variablen, jedoch nur 60 Matrizen mit einer bzw. drei Variablen. (Ähnliches gilt für die Komplexität, vgl. vorherige Tabelle.)

Für die meisten Kenngrößen sind Minimum (min), Maximum (max), Durchschnitt (avg), Median (med) und Streuung/Standardabweichung (str) angegeben.

Variablen		1	2	3	*	*	*	*
Grad		*	*	*	2	3	4	*
Anzahl		60	156	10	214	9	3	226
Anteil		100.0	37.14	16.67	59.45	20.0	5.0	41.85
	min	2	1	5	1	2	2	1
	max	4	10	9	10	7	6	10
F	avg	2.317	3.827	6.4	3.505	4.0	4.667	3.54
	med	2	1	6	1	6	2	1
	str	0.4997	2.373	1.114	2.206	1.886	1.886	2.196
	min	1570	310	7850	310	2930	3560	310
	max	85210	395880	387380	395880	287200	78430	395880
LZ	avg	4486	37610	107100	29760	80920	37030	31890
	med	3230	680	132150	680	281560	3560	680
	str	10530	80820	130800	73760	112500	31080	75950
	min	0.9317	0.0	11.79	0.0	2.047	5.337	0.0
	max	95.42	98.88	98.57	98.88	97.68	93.28	98.88
GB	avg	8.732	41.95	74.72	33.71	46.51	61.06	34.58
	med	1.858	0.0	84.93	0.0	97.6	5.337	0.0
	str	14.74	39.19	25.37	37.4	42.24	39.56	37.85
	min	0.0	0.0	0.002581	0.0	0.0	0.0	0.0
	max	7.027	2.454	0.2799	7.027	0.2	0.1031	7.027
SV	avg	1.966	0.09974	0.06856	0.6251	0.032	0.04712	0.5938
	med	1.858	0.0	0.01513	0.0	0.007103	0.0	0.0
	str	1.509	0.2882	0.08026	1.183	0.06115	0.04255	1.159
	min	3.227	0.6618	0.5427	0.5427	1.119	3.914	0.5427
	max	88.39	89.94	29.48	89.94	87.37	79.78	89.94
EL	avg	72.6	37.29	10.08	45.72	44.09	31.22	45.46
	med	78.95	47.06	2.225	47.06	1.119	79.78	47.06
	str	16.67	26.76	10.82	28.97	37.94	34.42	29.5
	min	0.5985	0.1059	0.1007	0.1007	0.3169	0.5355	0.1007
	max	17.61	30.16	4.105	30.16	5.6	3.089	30.16
M	avg	7.719	8.741	1.527	8.479	2.453	1.758	8.15
	med	8.049	23.53	0.4389	23.53	0.4049	3.089	23.53
	str	2.595	9.417	1.439	8.172	1.827	1.045	8.082
	min	0.2699	0.09599	0.1826	0.09599	0.3303	1.224	0.09599
	max	10.69	18.75	87.16	87.16	9.146	3.371	87.16
IV	avg	3.544	6.324	11.19	5.979	2.853	1.967	5.801
	med	3.715	16.18	11.75	16.18	0.3303	3.371	16.18

	str	1.401	6.474	25.52	7.978	2.51	0.9933	7.817
	min	0.3873	0.1702	0.1833	0.1702	0.4074	1.007	0.1702
	max	15.92	14.59	6.437	15.92	7.8	8.427	15.92
R	avg	5.441	5.595	2.409	5.49	4.058	3.947	5.413
	med	5.573	13.24	0.6432	13.24	0.5363	8.427	13.24
	str	2.29	3.756	2.244	3.434	3.018	3.219	3.432
	min	50.0	58.33	66.67	50.0	71.43	70.59	50.0
	max	84.62	100.0	77.78	100.0	83.33	83.33	100.0
ME	avg	69.03	84.15	72.96	79.73	78.61	76.8	79.64
	med	75.0	100.0	77.78	100.0	72.22	83.33	100.0
	str	8.026	10.91	3.557	12.34	4.824	5.208	12.07
	min	0.0	0.0	0.0	0.0	0.0	50.0	0.0
	max	100.0	100.0	50.0	100.0	100.0	100.0	100.0
SVE	avg	91.67	54.48	7.5	62.26	61.11	66.67	62.27
	med	100.0	100.0	0.0	100.0	0.0	100.0	100.0
	str	20.75	47.73	16.0	46.08	45.81	23.57	45.85
	min	5.042	0.0	20.09	0.0	11.24	11.24	0.0
	max	20.64	25.44	25.45	25.45	21.88	24.66	25.45
CE	avg	9.266	13.51	23.45	12.6	15.97	19.25	12.82
	med	5.7	0.0	20.69	0.0	18.96	11.24	0.0
	str	3.635	9.142	1.752	8.47	4.066	5.782	8.368
	min	0	0	0	0	0	0	0
	max	6	3	2	6	1	0	6
f^0	avg	1.55	0.4487	0.8	0.7897	0.2222	0.0	0.7566
	med	2	0	1	0	0	0	0
	str	1.309	0.6915	0.6	1.031	0.4157	0.0	1.016
	min	1	0	0	0	0	0	0
	max	10	8	3	10	3	0	10
f^X	avg	4.2	1.404	1.4	2.224	1.0	0.0	2.146
	med	8	0	3	0	0	0	0
	str	2.462	1.522	1.114	2.21	1.054	0.0	2.188

In keinem Experiment konnte jemals eine Matrix vom Grad 5 gel¨ost werden, weshalb sich eine Spalte f¨ur diese Komplexit¨at er¨ubrigt.

Die Abk¨urzungen in der vorangegangenen Tabelle haben die folgende Bedeutung:

Anzahl	Zahl der Matrizen, die in der jeweiligen Konfi guration gel¨ost werden konnten
Anteil	Verh¨altnis der gel¨osten Probleme zur Anzahl der Probleme dieser Konfi guration
F	Zahl der F¨alle pro L¨osung
LZ	Gesamtlaufzeit
GB	Anteil der Zeit zur Berechnung von Gr¨obnerbasen an der Gesamtlaufzeit
SV	Anteil der Laufzeit von simplifyColumn an der Gesamtlaufzeit
EL	Anteil der Zeit zur Elimination an der Gesamtlaufzeit
M	Anteil der Laufzeit von markowitz an der Gesamtlaufzeit
IV	Anteil der Laufzeit von simplifyPolynom an der Gesamtlaufzeit
R	Anteil der restlichen Laufzeit an der Gesamtlaufzeit
ME	Anteil der Pivot-Wahlen, in denen das von Markowitz vorgeschlagene Element genommen wurde
SVE	Anteil der erfolgreichen Aufrufe von simplifyColumn
CE	Cache-Trefferquote
f^0	Fill-in von konstanten oder symbolischen Eintr¨agen

f^X | Fill-in von symbolischen Eintr¨agen

Zur Messung des Fill-ins einer Klasse $*$ (im Sinne von Abschnitt 2.2, S. 19) wurde die Elimination Schritt f¨ur Schritt verfolgt. Jedesmal, wenn ein Matrixeintrag durch einen anderen ersetzt wurde (d. h. w¨ahrend Elimination, Idealvereinfachung oder Spaltenvereinfachung), wurde der neue Eintrag mit dem alten verglichen. Lag der neue Eintrag in Klasse $*$ und der alte in einer einfacheren [schlimmeren] Klasse, dann wurde f^* um 1 erh¨oht [erniedrigt]. Die Eintr¨age, die w¨ahrend der Elimination einer Spalte nach Konstruktion zu 0 werden, werden dabei nicht ber¨ucksichtigt.

Die folgende Tabelle zeigt, wie sich die Kenngr¨oßen von oben in Abh¨angigkeit von der Zahl der gefundenen F¨alle verhalten. (Es werden die gleichen Abk¨urzungen verwendet.)

	F¨alle	1	2	3	4	5	6	>6	*
	Anzahl	45	61	18	6	61	16	19	226
LZ	min	310	1630	1570	1590	2610	6820	4640	310
	max	710	3750	85210	4340	386980	395880	362390	395880
	avg	485.1	2956	8056	2595	49530	119200	100800	31890
	med	680	3230	4410	3650	47160	281560	61180	680
	str	141.7	475.4	18730	1059	83220	137700	109800	75950
GB	min	0.0	0.9317	8.955	14.51	17.24	11.79	36.42	0.0
	max	0.0	5.337	95.42	47.02	98.6	98.88	98.63	98.88
	avg	0.0	2.215	22.61	33.45	69.66	79.04	82.04	34.58
	med	0.0	1.858	19.95	19.45	90.82	97.6	86.39	0.0
	str	0.0	0.8502	18.91	12.93	23.08	22.69	17.27	37.85
SV	min	0.0	0.0	0.0	0.0	0.0	0.002581	0.00276	0.0
	max	0.0	7.027	3.829	1.622	0.616	0.4399	0.8621	7.027
	avg	0.0	1.611	1.355	0.4208	0.09051	0.08744	0.11	0.5938
	med	0.0	1.858	2.04	0.274	0.04241	0.007103	0.04904	0.0
	str	0.0	1.647	1.065	0.5828	0.1248	0.1093	0.1917	1.159
EL	min	44.45	73.26	3.227	12.58	0.5427	0.6618	0.7202	0.5427
	max	62.86	89.94	72.9	64.66	55.9	39.15	35.13	89.94
	avg	51.78	82.03	57.31	37.9	21.74	9.939	10.3	45.46
	med	47.06	78.95	56.92	64.66	5.916	1.119	5.672	47.06
	str	4.206	4.361	17.68	23.68	16.88	10.87	10.79	29.5
M	min	12.5	2.273	0.5985	4.378	0.1059	0.1007	0.2014	0.1007
	max	30.16	10.38	15.92	17.61	9.196	4.839	10.56	30.16
	avg	22.69	6.203	8.147	10.01	2.757	1.397	2.381	8.15
	med	23.53	8.049	9.977	5.48	0.9966	0.4049	2.386	23.53
	str	4.289	2.268	2.95	4.708	2.135	1.395	2.533	8.082
IV	min	12.5	1.701	0.2699	3.226	0.1021	0.09599	0.1932	0.09599
	max	18.75	4.192	8.28	11.32	6.13	87.16	9.146	87.16
	avg	16.11	2.961	4.297	7.05	2.047	7.371	2.26	5.801
	med	16.18	3.715	5.215	3.562	0.8482	0.3303	3.22	16.18
	str	1.405	0.5922	1.489	3.424	1.439	20.78	2.225	7.817
R	min	5.714	1.873	0.3873	6.452	0.1702	0.1794	0.2539	0.1702
	max	13.33	8.427	15.92	15.72	13.79	6.305	10.78	15.92
	avg	9.419	4.977	6.285	11.16	3.705	2.161	2.905	5.413
	med	13.24	5.573	5.896	6.575	1.378	0.5363	2.288	13.24
	str	2.155	1.225	2.829	3.76	2.878	2.072	2.767	3.432
	min	100.0	50.0	55.55	58.33	66.66	70.58	69.56	50.0

	max	100.0	85.71	83.33	84.61	81.25	77.77	82.75	100.0
ME	avg	100.0	71.39	73.67	72.18	78.42	73.44	75.05	79.64
	med	100.0	75.00	75.00	81.81	81.25	72.22	82.75	100.0
	str	0.0	9.439	7.551	9.202	2.887	2.892	4.755	12.07
	min	100.0	100.0	50.0	0.0	0.0	0.0	0.0	0.0
	max	100.0	100.0	100.0	100.0	100.0	100.0	50.0	100.0
SVE	avg	100.0	100.0	77.77	16.66	21.31	31.25	9.123	62.27
	med	100.0	100.0	50.0	0.0	100.00	0.0	0.0	100.0
	str	0.0	0.0	24.84	37.26	39.93	29.97	16.02	45.85
	min	0.0	5.042	7.74	18.92	14.74	18.92	14.71	0.0
	max	0.0	16.39	20.79	23.66	25.45	24.78	24.47	25.45
CE	avg	0.0	9.427	12.91	21.41	20.33	22.1	19.37	12.82
	med	0.0	5.7	8.779	23.66	14.79	18.96	15.08	0.0
	str	0.0	3.452	3.851	1.795	2.515	1.889	3.205	8.368

Die beiden folgenden Tabellen zeigen, welchen Einfluß die Heuristiken auf die verschiedenen Kenngr¨oßen haben.

Zun¨achst die Meßwerte f¨ur den Fall, daß man auf das Markowitz-Kriterium verzichtet. In dieser Konfiguration kann keine der Matrizen vom Grad 4 oder 5 gel¨ost werden, so daß die entsprechenden Spalten entfallen k¨onnen. Außerdem entfallen die Meßwerte f¨ur M und ME (sie sind allesamt 0). Ansonsten werden die gleichen Bezeichnungen verwendet wie bisher.

Variablen		1	2	3	*	*	*
Grad		*	*	*	2	3	*
Anzahl		60	91	3	152	2	154
Anteil		100.0	21.67	5.0	42.22	4.445	28.52
	min	2	1	6	1	4	1
	max	4	9	7	9	7	9
F	avg	2.417	3.198	6.333	2.921	5.5	2.955
	med	2	1	6	1	7	1
	str	0.5857	2.459	0.4714	2.005	1.5	2.02
	min	1580	240	11170	240	36500	240
	max	17600	358930	293600	358930	71630	358930
LZ	avg	3620	22880	109900	16580	54070	17070
	med	3440	480	25020	480	71630	480
	str	2299	58410	130000	51160	17570	51050
	min	0.6803	0.0	59.27	0.0	82.26	0.0
	max	79.49	98.94	96.92	98.94	91.34	98.94
GB	avg	9.912	31.87	79.3	23.42	86.8	24.24
	med	1.744	0.0	81.7	0.0	82.26	0.0
	str	14.21	38.46	15.47	33.06	4.543	33.63
	min	0.0	0.0	0.01362	0.0	0.0	0.0
	max	32.5	0.9926	0.1791	32.5	0.01396	32.5
SV	avg	5.092	0.0931	0.07755	2.067	0.00698	2.04
	med	5.814	0.0	0.03997	0.0	0.01396	0.0
	str	5.757	0.1846	0.07258	4.367	0.00698	4.345
	min	16.76	0.6659	0.654	0.654	2.625	0.654
	max	91.26	92.62	31.6	92.62	7.233	92.62
L	avg	76.63	49.25	15.58	59.98	4.929	59.26

	med	84.01	64.58	14.47	64.58	2.625	64.58
	str	15.16	27.94	12.66	27.18	2.304	27.71
	min	1.25	0.1033	1.719	0.1033	0.548	0.1033
	max	8.193	27.08	3.133	27.08	14.07	27.08
IV	avg	3.719	12.29	2.358	8.777	7.31	8.758
	med	3.488	27.08	1.719	27.08	14.07	27.08
	str	1.316	9.816	0.5856	8.721	6.762	8.7
	min	1.818	0.1798	0.1907	0.1798	0.8767	0.1798
	max	7.317	15.63	5.819	15.63	1.033	15.63
R	avg	4.646	6.496	2.696	5.764	0.9549	5.701
	med	4.942	8.333	2.078	8.333	1.033	8.333
	str	1.147	3.829	2.339	3.178	0.07819	3.204
	min	0.0	0.0	0.0	0.0	0.0	0.0
	max	100.0	100.0	33.33	100.0	0.0	100.0
SVE	avg	92.08	16.81	11.11	46.63	0.0	46.02
	med	100.0	0.0	0.0	0.0	0.0	0.0
	str	20.99	32.28	15.71	46.34	0.0	46.34
	min	12.5	0.0	25.0	0.0	30.77	0.0
	max	38.46	34.88	28.17	38.46	34.88	38.46
CE	avg	21.11	14.64	26.89	17.2	32.83	17.4
	med	15.0	0.0	28.17	0.0	34.88	0.0
	str	5.735	14.61	1.364	12.22	2.057	12.27
	min	0	0	1	0	0	0
	max	9	6	1	9	1	9
f^0	avg	3.833	0.978	1.0	2.112	0.5	2.09
	med	6	0	1	0	0	0
	str	2.05	1.43	0.0	2.193	0.5	2.187
	min	1	0	1	0	1	0
	max	17	15	3	17	4	17
f^X	avg	8.016	1.879	2.0	4.296	2.5	4.273
	med	16	0	2	0	4	0
	str	3.779	2.635	0.8165	4.336	1.5	4.316

Die letzte Tabelle zeigt schließlich, wie sich der Algorithmus ohne Spaltenvereinfachung verh¨alt.

Variablen	1	2	3	*	*	*	*
Grad	*	*	*	2	3	4	*
Anzahl	59	158	10	215	9	3	227
Anteil	98.33	37.62	16.67	59.72	20.0	5.0	42.03
min	2	1	5	1	2	2	1
max	5	10	9	10	7	6	10
F avg	3.61	3.899	6.4	3.921	4.0	4.667	3.934
med	3	1	6	1	6	2	1
str	0.6383	2.385	1.114	2.111	1.886	1.886	2.101
min	1630	300	7820	300	2920	3450	300
max	66090	387950	387300	387950	286750	74510	387950
LZ avg	9270	34270	107400	28660	8530	35970	30990
med	4270	640	128810	640	280750	3450	640
str	12480	71450	130800	65840	111600	29320	68800
min	1.158	0.0	11.68	0.0	2.054	5.217	0.0
max	93.69	98.66	98.56	98.66	97.65	93.12	98.66

GB	avg	40.79	42.4	76.2	43.06	47.07	61.19	43.47
	med	20.38	0.0	84.65	0.0	97.65	5.217	0.0
	str	22.91	38.86	24.72	35.09	42.71	39.71	35.56
	min	4.297	0.7221	0.4879	0.4879	1.111	4.039	0.4879
	max	87.26	89.8	25.19	89.8	87.67	79.71	89.8
L	avg	44.21	37.21	9.136	37.64	43.76	31.06	37.8
	med	60.42	46.88	2.244	46.88	1.111	79.71	46.88
	str	20.04	26.32	9.404	24.35	38.29	34.47	25.25
	min	0.8927	0.1725	0.09812	0.09812	0.3208	0.4832	0.09812
	max	17.18	29.69	3.548	29.69	5.422	4.058	29.69
M	avg	6.336	8.771	1.323	8.12	2.337	2.037	7.81
	med	8.665	25.0	0.4425	25.0	0.3776	4.058	25.0
	str	2.599	9.453	1.176	8.297	1.798	1.496	8.19
	min	0.4691	0.1331	0.1875	0.1331	0.3348	1.236	0.1331
	max	9.816	20.0	87.28	87.28	9.131	3.188	87.28
IV	avg	3.467	6.163	11.08	5.85	2.861	1.891	5.679
	med	3.747	17.19	12.04	17.19	0.3348	3.188	17.19
	str	1.308	6.333	25.61	7.904	2.535	0.9176	7.744
	min	0.6506	0.2288	0.1678	0.1678	0.4255	1.114	0.1678
	max	15.95	16.25	5.755	16.25	8.032	7.826	16.25
R	avg	5.193	5.457	2.258	5.321	3.964	3.815	5.247
	med	6.792	10.94	0.6288	10.94	0.5236	7.826	10.94
	str	2.21	3.762	2.048	3.433	3.023	2.893	3.425
	min	76.92	66.67	66.67	66.67	71.43	70.59	66.67
	max	93.33	100.0	77.78	100.0	83.33	83.33	100.0
ME	avg	89.35	84.42	72.96	85.59	78.61	76.8	85.2
	med	90.91	100.0	77.78	100.0	72.22	83.33	100.0
	str	3.414	10.45	3.557	9.589	4.824	5.208	9.548
	min	8.397	0	19.72	0	11.24	11.24	0
	max	18.56	24.55	25.75	25.75	21.93	23.72	25.75
CE	avg	12.47	13.4	22.72	13.4	15.73	18.73	13.57
	med	9.091	0	19.77	0	18.04	11.24	0
	str	2.774	8.941	1.935	7.976	3.966	5.395	7.863
	min	0	0	0	0	0	0	0
	max	6	3	2	6	1	0	6
f^0	avg	1.508	0.462	0.8	0.7814	0.2222	0.0	0.7489
	med	2	0	1	0	0	0	0
	str	1.281	0.6993	0.6	1.008	0.4157	0.0	0.9949
	min	1	0	0	0	0	0	0
	max	11	8	3	11	3	0	11
f^X	avg	4.187	1.399	1.4	2.2	1.0	0.0	2.123
	med	8	0	3	0	0	0	0
	str	2.613	1.518	1.114	2.244	1.054	0.0	2.22

Extreme Corpuselemente

Aus Platzgr¨unden wird darauf verzichtet, den ganzen Corpus abzudrucken. Exemplarisch werden hier nur jede Probleme aufgelistet, auf die im Text verwiesen wird.

Die Numerierung entspricht dem Index des Problems im Corpus.

$$\begin{pmatrix} 0 & 2x-8 & -14x^2 & 7x^2-5 \\ -1 & 5x+x^2 & -5 & -4 \\ 0 & -14x^2 & 0 & 0 \\ -5 & 4x^2 & 5x+4 & 3x-9 \end{pmatrix} \tag{1}$$

$$\begin{pmatrix} y^2-7 & 0 & 0 & 4 \\ 2x^2+9y^2 & 6xy+2x^2y^2 & 7x-2xy^2 & -9y+1 \\ 0 & 9x+5 & -5x-6y^2 & 0 \\ 6 & 5 & 2xy^2+7 & -7 \end{pmatrix} \tag{32}$$

$$\begin{pmatrix} -5 & 4y^2-7x^2y & -2 & -9x-9x^2 \\ 0 & -6x^2+3x^2y & 0 & 3 \\ -xy+7x^2 & -7xy-x^2y^2 & 8 & -3xy+8x^2 \\ 3x^2-xy^2 & 8x^2y^2 & 0 & 0 \end{pmatrix} \tag{44}$$

$$\begin{pmatrix} 0 & 15y & -7y & -7x+xy \\ 5y^2+7x^2y & 2 & 0 & -4x^2y^2 \\ 0 & -7 & 2x-7x^2 & -1 \\ 2x^2y^2+6 & 2 & 0 & -3y+7y^2 \end{pmatrix} \tag{50}$$

$$\begin{pmatrix} y^2+8xy^2 & -4x^2+y^2 & -5x+6x^2y^2 & 5x+5y^2 \\ -6 & 0 & 0 & -8x-7x^2 \\ -4 & -1 & -8xy^2 & 3y-4x^2y^2 \\ 6y^2+7x^2y^2 & -9 & 0 & 0 \end{pmatrix} \tag{56}$$

$$\begin{pmatrix} 9y+6xy & 8 & 9y^2 & 5 \\ y^2-x^2y & 5y-x^2 & 0 & -2 \\ -3x-3x^2y^2 & 0 & 3xy-x^2y & -9y-8y^2 \\ 0 & 7x-3xy^2 & 0 & -9 \end{pmatrix} \tag{61}$$

Literatur

[BWK93] T. Becker, V. Weispfenning, and H. Kredel. *Gröbner Bases*. Springer, 1993.

[CLO92] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer, 1992.

[DER86] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, 1986.

[DL88] R. Dautray and J.-L. Lions. *Mathematical Analysis and Numerical Methods for Science and Technology*. Springer, 1988.

[DST88] J. H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra*. Academic Press, 1988.

[Gau01] C. F. Gauß. Disquisitiones arithmeticae. Universit¨at G¨ottingen, 1801.

[GCL92] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Press, 1992.

[GMN+91] A. Giovini, T. Mora, G. Niesi, L Robbiano, and C. Traverso. "One sugar cube, please," or Selection strategies in the Buchberger algorithm. *Proc. International Sympositum on Symbolic and Algebraic Computation (ISSAC '91)*, page 49, 1991.

[H¨or76] L. H¨ormander. *Linear partial differential operators*. Springer, 1976.

[Jen96] K. Jensen. *Coloured Petri Nets*. Springer, 1996.

[Kap95] D. Kapur. An approach for solving systems of parametric equations. *Principles and practice of constraint programming: The Newport papers*, pages 217–243, 1995.

[OPRW98] W. Oevel, F. Postel, J. R¨uscher, and S. Wehmeier. *Das* MuPAD-*Tutorium*. SciFace Software, 1998.

[RR93] M. Renardy and R. Rogers. *An Introduction to Partial Differential Equations*. Springer, 1993.

[Sit92] W. J. Sit. An algorithm for solving parametric linear systems. *Journal of Symbolic Computation*, pages 353–394, 1992.

[vzGG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[WR98] G. Rozenberg W. Reisig. *Lectures on Petri Nets I: Basic Methods*. Springer, 1998.

Funktionen und Prozeduren

Die folgende Liste enthält die Deklarationen aller verwendeten Funktionen und Prozeduren.

elimination(A, C, row, col)	Bestimmt für den Fall C die Lösungsmenge des Systems $A = ((a_{ij}))_{i=row\,j=col}^{n\quad m}$ ↑S. 15
swapRow(A, i, j)	Gibt die Matrix A' zurück, die aus A durch Vertauschen der i-ten und j-ten Zeile entsteht ↑S. 15
swapColumn(A, i, j)	Gibt die Matrix A' zurück, die aus A durch Vertauschen der i-ten und j-ten Spalte entsteht ↑S. 15
rows(A)	Anzahl der Zeilen von A ↑S. 15
cols(A)	Anzahl der Spalten von A ↑S. 15
gccd$(C, p_1, \ldots, p_n)$	Bestimmt den größten gemeinsamen Teiler der Polynome $p_1, \ldots, p_n$, der bezüglich C konstant ist ↑S. 18
gcd$(p1, \ldots, p_n)$	Bestimmt den größten gemeinsamen Teiler der Polynome $p_1, \ldots, p_n$ ↑S. 18
factor(p)	Faktorisiert das Polynom p ↑S. 18
eliminationStep(C, A, i, j)	Eliminiert alle Elemente $A[k, j]$ mit $k > j$ ↑S. 19
markowitz(C, A)	Wählt ein günstiges Element in A aus, das als Pivot verwendet werden kann ↑S. 23
reduce$(f, \{g_1, \ldots, g_n\})$	Bestimmt die Normalform von f bezüglich der Gröbnerbasis $G := \{g_1, \ldots, g_n\}$. ↑S. 29
sPoly(f, g)	Bestimmt das S-Polynom von f und g ↑S. 29
gBasis$(\{f_1, \ldots, f_n\})$	Bestimmt eine Gröbnerbasis von $\{f_1, \ldots, f_n\}$ ↑S. 29
isMember$(p, \{a_1, \ldots, a_n\})$	Prüft, ob $p \in \langle a_1, \ldots, a_n \rangle$ gilt (Membership-Problem) ↑S. 30
simplifyPolynom(C, p)	Vereinfacht das Polynom p bezüglich C ↑S. 33
addZero(C, p)	Erzeugt den Fall $C \cup \{p = 0\}$ ↑S. 33
addConstant(C, p)	Erzeugt den Fall $C \cup \{p \neq 0\}$ ↑S. 33
isZero(C, p)	Prüft, ob $C \models (p = 0)$ gilt ↑S. 33
isConstant(C, p)	Prüft, ob $C \models (p \neq 0)$ gilt ↑S. 33

Symbolverzeichnis

$\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$	Menge der natürlichen, ganzen, rationalen, reellen bzw. komplexen Zahlen
$\mathbb{N}_0$	Menge der natürlichen Zahlen incl. 0
$\|A\|$	Mächtigkeit der Menge A
$A \dot\cup B$	Disjunkte Vereinigung der Mengen A und B
k	Ein beliebiger, aber fester algebraisch abgeschlossener Körper
$k[x_1, \ldots, x_r]$	Ring der Polynome über k in den Variablen $x_1, \ldots, x_r$
$k(x_1, \ldots, x_r)$	Körper der rationalen Funktionen über k in den Variablen $x_1, \ldots, x_r$
$R^{n \times m}$	Ring der Matrizen der Größe $n \times m$ über R
$A[i,j], a_{ij}$	Matrix-Element der Matrix A bzw. $((a_{ij}))$ an der Stelle (i,j)
$\mathrm{Rang} A$	Rang der Matrix A
$\det A$	Determinate der Matrix A
$\mathrm{diag}(d_1, \ldots, d_n)$	Diagonalmatrix mit den Elementen $d_1, \ldots, d_n$
Σ	Menge der Spezialisierungen $\{x_1, \ldots, x_r\} \to k$
$\|p\|$	Anzahl der Monome im Polynom p
∂p	Grad des Polynoms p
$\mathfrak{a} \trianglelefteq R$	$\mathfrak{a}$ ist ein Ideal in R
$\langle f_1, \ldots, f_n \rangle$	Von $f_1, \ldots, f_n$ erzeugtes Ideal
$\mathrm{Rad}\,\mathfrak{a}$	Radikal des Ideals $\mathfrak{a}$
$\mathrm{Var}(P)$	Von $P \subseteq k[x_1, \ldots, x_r]$ erzeugte Varietät
$I(V)$	Von der Varietät V erzeugtes Ideal
$\to_g^0$	Reduktion modulo $g \in k[x_1, \ldots, x_r]$
$\to_G^0$	Reduktion modulo $G \subseteq k[x_1, \ldots, x_r]$
$\to_g$	Reflexive, transitive Hülle von $\to_g^0$
$\to_G$	Reflexive, transitive Hülle von $\to_G^0$
$p \preccurlyeq q$	Term- und Polynomordnung
$\mathrm{LT}(p)$	Leitterm des Polynoms p
$\mathrm{LC}(p)$	Leitkoeffizient des Polynoms p
$\mathrm{LM}(p)$	Leitmonom des Polynoms p
$p = 0, p \neq 0$	Nebenbedingungen
Σ_F	Menge der Spezialisierungen eines Falls F
$\neg p$	Negiertes Polynom
V_F, I_F	Vom Fall F erzeugte Varietät / erzeugtes Ideal
$F \models b$	Im Fall F gilt die Bedingung b
$F \vdash b$	Aus der Definition von F ist b ableitbar

Index