Gitterbasenreduktion mit Random Sampling und heuristischen Erweiterungen

An der Universität Bayreuth
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
vorgelegte Abhandlung

von

Heiko Vogel

aus Kulmbach

1. Gutachter: Prof. Dr. Alfred Wassermann
2. Gutachter: Prof. Dr. Michael Dettweiler

Tag der Einreichung: 27. 01. 2011
Tag des Kolloquiums: 20. 07. 2011

Ehrenwörtliche Erklärung

Ich versichere, dass ich die vorliegende Dissertation ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und den benutzten Quellen wörtlich oder inhaltlich entnommene Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Bayreuth, den 27. 01. 2011 Heiko Vogel

Abstract

This dissertation describes the mathematical subject concerned with lattice basis reduction and analyzes several algorithms which are involved in solving lattice based problems.

Stacked on the scientific findings of the diploma thesis "Gitterbasenreduktion mit Random Sampling" several modifications of the famous LLL-algorithm and the generalized BKZ-reduction are introduced: C. Schnorr's method for extending the Lovász step with deep insertions and a modified basis replacement routine for the enumeration step, which was first described by H. Hörner. Further, two different cutting techniques are explained, which allow to decrease the size of the enumeration tree. These techniques originate from A. Wassermann and P. Nguyen.

Schnorr's Random Sampling strategy is modified in order to deal with lattices, which have a bad GSA behaviour and an approach from Buchmann and Ludwig is implemented where the GSA behaviour gets totally irrelevant.

Finally a heuristical evaluation concept for lattice vectors is developed and implemented in form of a modified sieve algorithm, which originates from Ajtaj, Kumar, Sivakumar (AKS) and has been extensively examined by T. Vidick and P. Nguyen afterwards.

Regarding the quality of the achieved lattice reductions for hard market split problems in dimensions ≈ 120 these new methods produce excellent results in short time (about 5 hours on a machine with 3 GHz). Even for problem dimensions > 500 the findings are still quite satisfying, though the computation amount (> 7 days) is not negligible anymore.

In comparison with the commercial program CPLEX, which uses completely different methods for solving integer linear programs, the results are remarkably very good and motivate further investigations in the field of lattice reduction.

Zusammenfassung

Diese Dissertation beschäftigt sich mit dem mathematischen Teilgebiet der Gitterbasenreduktion. Aufbauend aus den Erkenntnissen der Diplomarbeit „Gitterbasenreduktion mit Random Sampling“ werden verschiedene Modifikationen am ursprünglichen LLL- bzw. BKZ-Verfahren vorgenommen: Es wird der von C. Schnorr entwickelte Ansatz, den LLL-Austauschschritt um Tiefeneinfügungen zu erweitern, aufgegriffen und eine alternative Methode zum Basisaustausch für das BKZ-Verfahren vorgestellt. Ferner werden zwei unterschiedliche Verfahren von A. Wassermann und P. Nguyen zum Abschneiden von Enumerationsbäumen beschrieben.

Die Random Sampling Strategie von Schnorr wurde überarbeitet, um ein schlechtes GSA-Verhalten des Gitters zu berücksichtigen und eine neuartige Strategie von Buchmann und Ludwig wurde implementiert, bei der das GSA-Verhalten vollkommen irrelevant ist.

Schließlich wird ein grundlegendes, heuristisches Bewertungskonzept für Gittervektoren entwickelt, das im Rahmen eines von T. Vidick und P. Nguyen beschriebenen Siebverfahrens, Anwendung findet.

Mit Hinblick auf die Qualität der erreichten Gitter-Reduktion für schwierige Market-Split Probleme in Dimensionen ≈ 120, liefern diese neuen Methoden hervorragende Ergebnisse in äußerst kurzer Zeit (ca. 5 Stunden auf einem 3 GHz Rechner). Auch für Problemdimensionen > 500 sind die Resultate durchaus noch zufriedenstellend - allerdings ist hierbei der Rechenaufwand (> 7 Tage) nicht mehr zu vernachlässigen.

Im Vergleich mit dem kommerziellen Programm CPLEX, das einen völlig anderen Ansatz zur Lösung von ganzzahlig-linearen Gleichungssystemen verfolgt, konnten sogar sehr gute Ergebnisse erzielt werden.

Inhaltsverzeichnis

Einleitung

Aus algebraischer Sicht befasst sich Mathematik mit Objekten, denen eine spezielle Struktur zugrunde liegt und Operationen bzw. Verknüpfungen zwischen diesen Objekten, die diese strukturellen Besonderheiten berücksichtigen. Ein äußerst reizvolles Objekt, das in vielen Teilgebieten der Mathematik in Erscheinung tritt und in dieser Dissertation etwas genauer unter die Lupe genommen werden soll, trägt die Bezeichnung Gitter (engl: lattice).

Ein Gitter ist eine diskrete, additive Untergruppe des $\mathbb{R}^n$ und besitzt somit ein endliches, linear unabhängiges Erzeugendensystem (Basis). Da es zu jedem Gitter allerdings unendlich viele verschiedene solcher Basen gibt, ist man an jenen interessiert, die sich dadurch auszeichnen, dass die zugehörigen Basisvektoren kurz und nahezu orthogonal zueinander sind.

Gitterbasenreduktion beschäftigt sich mit Verfahren, solche besonderen Basen für ein vorgegebenes Gitter zu finden. Die Reduktionsmethoden operieren dabei auf dem Gitter und führen durch eine Folge unimodularer Transformationen Basen ineinander über, ohne dabei die Struktur des Gitters zu verletzen. Durch einen einfachen Ansatz ist es damit insbesondere möglich, Lösungen für ganzzahlig-lineare Gleichungssysteme zu finden.

Die zugrundeliegende Theorie wurzelt ursprünglich in der äquivalenten Reduktionstheorie quadratischer Formen, die von Gauss, Hermite, Korkine und Zolotarev ins Leben gerufen wurde. Minkowski entwickelte schließlich in seiner „Geometrie der Zahlen“ eine geometrische Betrachtungsweise der Gitter.

Zu Beginn der 80'er Jahre erlangte die Gitterbasenreduktion neue Impulse von A. K. Lenstra, H. W. Lenstra und L. Lovász, die ein Verfahren entwickelten, das in polynomieller Rechenzeit mithilfe eines Computers eine reduzierte Basis eines Gitters berechnet. Die von ihnen beschriebene Methode verallgemeinert den Gauss'schen Reduktionsbegriff, der nur für zweidimensionale Gitter definiert ist, auf Gitter von beliebigem Rang (LLL-Reduktion).

In den darauffolgenden Jahren wurden, insbesondere von C. P. Schnorr, eine Vielzahl weiterer Reduktionsbegriffe geschaffen, die schließlich in der Entwicklung des BKZ-Verfahrens münden, das die LLL-Reduktion mit dem von Hermite, Korkine und Zolotarev begründeten Reduktionsbegriff vereint.

Auf Schnorr geht auch die Idee zurück, mithilfe der Gram-Schmidt Koeffizienten einer Gitterbasis, kurze Gittervektoren zu erzeugen. Dies geschieht durch Anwendung eines zufallsbasierten Verfahrens, was ich in meiner Diplomarbeit „Gitterbasenreduktion mit Random Sampling“ **[V05]** bereits näher untersucht habe.

Ziel dieser Dissertation ist es, aufbauend aus den Erkenntnissen der Diplomarbeit, verschiedene Modifikationen am ursprünglichen LLL- bzw. BKZ-Reduktionsverfahren vorzunehmen, das Random Sampling zu überarbeiten und grundlegende heuristische Bewertungsmodelle für Gittervektoren zu entwickeln, die in einem Siebverfahren Anwendung finden.

Im ersten Kapitel dieser Arbeit, werden zunächst mathematische Notationen und Begriffe eingeführt, die für das Verständnis der Theorie notwendig sind. Im Anschluss daran werden verschiedene Spezialbegriffe definiert und eine kurze Einführung in die Gittertheorie gegeben, wobei auch der wichtige Bezug zu ganzzahlig-linearen Gleichungssystemen näher erläutert wird.

Innerhalb des zweiten Kapitels werden die beiden essentiellen Algorithmen LLL-Reduktion und BKZ-Verfahren rekapituliert. Es wird auf den Enumerationsprozess eingegangen und eine neue, alternative Methode zum Basisaustausch vorgestellt. Außerdem werden diverse Modifikationen, wie Tiefeneinfügungen oder das Abschneiden von Enumerationsbäumen beschrieben. Den neuartigen Extreme Pruning Techniken wird ebenfalls besonderes Augenmerk zugute kommen.

Das dritte Kapitel befasst sich mit der Thematik des Random Sampling. Es werden die Theorie dargelegt und zwei unterschiedliche Strategien vorgestellt, mit denen geeignete Parameter für das Verfahren ermittelt werden können. Dies wären die Strategie von Schnorr, die von mir überarbeitet wurde, um ein schlechtes GSA-Verhalten des Gitters zu berücksichtigen und eine neuartige Strategie von Buchmann und Ludwig, bei der das GSA-Verhalten vollkommen irrelevant ist.

Im folgenden Kapitel wird ein neues, grundlegendes, heuristisches Modell zur Bewertung von Gittervektoren entwickelt. Dieses spezielle Modell findet Verwendung in einem Algorithmus zur Verringerung der heuristischen Bewertung von Basisvektoren und in einem Siebverfahren zur Herausfilterung von Vektoren mit guten heuristischen Eigenschaften.

Das Programmpaket, das zur Umsetzung der beschriebenen Algorithmen entwickelt wurde, wird im fünften Kapitel dokumentiert. Es werden die Eingabeformate, die Kommandozeilenparameter und der Aufbau erläutert.

Das letzte Kapitel dieser Dissertation dient der praktischen Überprüfung der implementierten Methoden in Form von Laufzeitvergleichen zu unterschiedlichen Probleminstanzen mit verschiedenen Parametersätzen, die tabellarisch aufgeführt werden.

Im Fazit werden diese Ergebnisse dann ausgewertet und schriftlich fixiert.

Im Anhang wird noch auf die Handhabung unlösbarer FP-Probleme und die Konstruktion linearer Codes eingegangen. Abschließend werden die Lösungsvektoren aufgelistet, die während der Benchmarks gefunden wurden.

KAPITEL 1

Theoretische Grundlagen

Im ersten Kapitel werden mathematische Grundlagen aus der Analysis und der Linearen Algebra rekapituliert und Notationen eingeführt, die im weiteren Verlauf dieser Arbeit verwendet werden. Damit dieses Kapitel nicht unverhältnismäßig lang wird, muss auf die Definition von Begriffen wie: Funktion, Monotonie, Menge, Gruppe, Vektorraum oder (lineare) Abbildung verzichtet werden. Hierzu sei auf entsprechende Literatur verwiesen [**F97, J98, K95**].

1.1. Symbole / Notation

$\mathbb{N}$ ist die Menge der natürlichen Zahl, $\mathbb{N}_0$ die Menge der natürlichen Zahlen einschließlich der 0. Die Menge der ganzen, bzw. rationalen Zahlen wird mit $\mathbb{Z}$, bzw. $\mathbb{Q}$ bezeichnet. $\mathbb{R}$ steht für die Menge der reellen Zahlen. Die Menge der nicht-negativen rellen Zahlen wird durch $\mathbb{R}_+$ und die Menge der negativen reellen Zahlen durch $\mathbb{R}_-$ gekennzeichnet. Analog geschieht dies für $\mathbb{Z}$ und $\mathbb{Q}$.

Die Menge der geordneten n-Tupel von natürlichen, ganzen, rationalen, bzw. reellen Zahlen entsprechen den Symbolen $\mathbb{N}^n$, $\mathbb{Z}^n$, $\mathbb{Q}^n$, bzw. $\mathbb{R}^n$. Im Falle von $\mathbb{Q}^n$, bzw. $\mathbb{R}^n$ spricht man auch von rationalen, bzw. reellen (n-dimensionalen) Vektorräumen, deren Elemente dann Vektoren genannt werden. Da $\mathbb{N}^n$ und $\mathbb{Z}^n$ Teilmengen von $\mathbb{Q}^n$, bzw. $\mathbb{R}^n$ sind, werden deren Elemente im folgenden auch Vektoren genannt.

Der Raum der reellen $m \times n$-Matrizen wird mit $\mathbb{R}^{m\times n}$ bezeichnet, wobei m die Anzahl der Zeilen und n die Anzahl der Spalten wiedergibt. Die Menge der invertierbaren, reellen $n \times n$-Matrizen erhält das Symbol $GL_n(\mathbb{R})$.

Die Schreibweise $x \in \mathbb{R}^n$ soll bedeuten, dass x ein Spaltenvektor, also

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \text{ mit Einträgen } x_j \in \mathbb{R},\ j = 1, 2, ...n \text{ ist.}$$

Ein Zeilenvektor wird durch $x^T = (x_1, x_2, ..., x_n)$ gekennzeichnet und entspricht damit einem transponierten Spaltenvektor.

Die Schreibweise $A \in \mathbb{R}^{m\times n}$ soll bedeuten, dass A eine Matrix, also

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} \text{ mit Einträgen } a_{i,j} \in \mathbb{R} \text{ ist.}$$

$\langle ., . \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ bezeichnet ein beliebiges Skalarprodukt des $\mathbb{R}^n$. Falls nicht anders vermerkt, wird jedoch das (euklidische) Standardskalarprodukt $\langle x, y \rangle := x^T y = \sum_{i=1}^n x_i \cdot y_i$ verwendet. Die Länge eines Vektors x lässt sich damit ausdrücken als $||x||_2 := \sqrt{\langle x, x \rangle}$.

Im eindimensionalen Fall, also wenn es sich bei x um einen Skalar handelt, symbolisiert $|x|$ die Länge von x, $\lfloor x \rfloor$ bzw. $\lceil x \rceil$ die ganze Zahl, die man erhält, wenn man x ab- bzw. aufrundet, und $\lfloor x \rceil$ die nächste ganze Zahl zu x.

Damit wäre die Notation vorerst geklärt, und es werden einige wichtige, grundlegende Definitionen und Sätze vermerkt.

1.2. Definitionen und Sätze

LEMMA 1. *Für die Funktion* $f : \mathbb{R} \to \mathbb{R}$, $t \mapsto t^p - pt$ *gilt:*

$$f(t) \leq 1 - p \text{ für alle } t \geq 0 \text{ und } 0 < p < 1$$

BEWEIS. Es ist $f'(t) = pt^{p-1} - p = p(t^{p-1} - 1)$. Nach Voraussetzung ist $p - 1 < 0$. Daher gilt $f'(t) > 0$ für $0 < t < 1$ und die Funktion ist streng monoton wachsend. Für $t > 1$ ist $f'(t) < 0$ und f ist streng monoton fallend. Sie nimmt für $t = 1$ ihr Maximum an, mit $f(1) = 1 - p$. □

PROPOSITION 2. *Für* $a, b \in \mathbb{R}$ *mit* $a, b > 0$ *und* $p, q \in \mathbb{R}$ *mit* $0 < p < 1$ *und* $p + q = 1$ *gilt:*

$$a^p b^q \leq pa + qb$$

BEWEIS. Nach dem vorherigem Lemma ist $t^p - pt \leq 1 - p$ für alle $t \geq 0$ und $0 < p < 1$. Es sei nun $t = ab^{-1}$ und $q = 1 - p$. Dann schreibt sich die Ungleichung als $a^p b^{-p} - p\frac{a}{b} \leq q$ und Multiplikation mit b ergibt:

$$a^p \cdot b^{1-p} - pa \leq qb$$

□

DEFINITION 3. Es sei $x \in \mathbb{R}^n$ und $p \in \mathbb{R}$ mit $p \geq 1$.

Dann definiert der Ausdruck

$$||x||_p := \left(\sum_{j=1}^n |x_j|^p \right)^{\frac{1}{p}}$$

die *p-Norm* von x.

Im Falle $p = \infty$ wird die Norm *auch als Maximums- bzw. Supremumsnorm* bezeichnet und es gilt:

$$||x||_\infty = \max\{|x_1|, |x_2|, ..., |x_n|\}$$

THEOREM 4. *(Hölder'sche Ungleichung):*

Es seien $x, y \in \mathbb{R}^n$ *und* $p, q \in \mathbb{N}$ *mit* $\frac{1}{p} + \frac{1}{q} = 1$. *Dann gilt:*

$$|\langle x, y \rangle| \leq \sum_{j=1}^n |x_j y_j| \leq ||x||_p \cdot ||x||_q$$

BEWEIS. Der erste Teil der Ungleichung ist klar (Dreiecksungleichung). Für den zweiten Teil ist zunächst festzustellen, dass für $a, b \in \mathbb{R}$ nach der vorherigen Proposition gilt: $ab = (a^p)^{\frac{1}{p}} (b^q)^{\frac{1}{q}} \leq \frac{a^p}{p} + \frac{b^q}{q}$. Damit erhält man für x, y:

$$\sum_{j=1}^{n} \frac{|x_j y_j|}{||x||_p ||y||_p} = \sum_{j=1}^{n} \frac{|x_j|}{||x||_p} \cdot \sum_{j=1}^{n} \frac{|y_j|}{||y||_p} \leq \sum_{j=1}^{n} \frac{1}{p} \frac{|x_j|^p}{||x||_p^p} + \sum_{j=1}^{n} \frac{1}{q} \frac{|y_j|^q}{||y||_q^q}$$

$$= \frac{1}{p} \cdot \frac{1}{||x||_p^p} \cdot \sum_{j=1}^{n} |x_j|^p + \frac{1}{q} \cdot \frac{1}{||y||_q^q} \cdot \sum_{j=1}^{n} |y_j|^q = \frac{1}{p} + \frac{1}{q} = 1.$$

□

THEOREM 5. *(Ungleichung von Minkowski):*

Für $x, y \in \mathbb{R}^n$ und $p \in \mathbb{R}$ mit $1 \leq p \leq \infty$ gilt die Ungleichung:

$$||x + y||_p \leq ||x||_p + ||y||_p$$

BEWEIS. Die Fälle $p = 1$ und $p = \infty$ sind trivial und folgen direkt aus der gewöhnlichen Dreiecksungleichung. Für $1 < p < \infty$ ergibt sich zunächst:

$$\begin{aligned} |x_j + y_j|^p = |x_j + y_j| \cdot |x_j + y_j|^{p-1} &\leq (|x_j| + |y_j|) \cdot |x_j + y_j|^{p-1} \\ &= |x_j| \cdot |x_j + y_j|^{p-1} + |y_j| \cdot |x_j + y_j|^{p-1} \end{aligned}$$

Und nach dem Aufsummieren:

$$\sum_{j=1}^{n} |x_j + y_j|^p \leq \sum_{j=1}^{n} |x_j| \cdot |x_j + y_j|^{p-1} + \sum_{j=1}^{n} |y_j| \cdot |x_j + y_j|^{p-1}$$

Mit $q := \frac{p}{p-1}$ erhält man $\frac{1}{p} + \frac{1}{q} = 1$. Zweimalige Anwendung der Hölder'schen Ungleichung auf der rechten Seite liefert:

$$\sum_{j=1}^{n} |x_j| \cdot |x_j + y_j|^{p-1} \leq \left(\sum_{j=1}^{n} |x_j|^p \right)^{\frac{1}{p}} \cdot \left(\sum_{j=1}^{n} \left(|x_j + y_j|^{p-1} \right)^q \right)^{\frac{1}{q}}$$

$$\text{und } \sum_{j=1}^{n} |y_j| \cdot |x_j + y_j|^{p-1} \leq \left(\sum_{j=1}^{n} |y_j|^p \right)^{\frac{1}{p}} \cdot \left(\sum_{j=1}^{n} \left(|x_j + y_j|^{p-1} \right)^q \right)^{\frac{1}{q}}.$$

Unter Ausnutzung der Identität $(p-1) \cdot q = p$ erhält man insgesamt:

$$\sum_{j=1}^{n} |x_j + y_j|^p \leq \left(\left(\sum_{j=1}^{n} |x_j|^p \right)^{\frac{1}{p}} + \left(\sum_{j=1}^{n} |y_j|^p \right)^{\frac{1}{p}} \right) \cdot \left(\sum_{j=1}^{n} |x_j + y_j|^{(p-1) \cdot q} \right)^{\frac{1}{q}}.$$

Dividiert man diese Ungleichung durch $\left(\sum_{j=1}^{n} |x_j + y_j|^{(p-1) \cdot q} \right)^{\frac{1}{q}}$ ergibt sich:

$$\left(\sum_{j=1}^{n} |x_j + y_j|^p \right)^{1 - \frac{1}{q}} \leq \left(\sum_{j=1}^{n} |x_j|^p \right)^{\frac{1}{p}} + \left(\sum_{j=1}^{n} |y_j|^p \right)^{\frac{1}{p}}$$

und folglich:

$$\left(\sum_{j=1}^{n}|x_j+y_j|^p\right)^{\frac{1}{p}} \leq \left(\sum_{j=1}^{n}|x_j|^p\right)^{\frac{1}{p}} + \left(\sum_{j=1}^{n}|y_j|^p\right)^{\frac{1}{p}}$$

□

THEOREM 6. *(Äquivalenz der Standardnormen)*

Für alle $x \in \mathbb{R}^n$ gilt die Ungleichungskette:

$$\frac{1}{n}|x_j| \leq \frac{1}{n}||x||_1 \leq ||x||_\infty \leq ||x||_p \leq \sqrt[p]{n}||x||_\infty \leq \sqrt[p]{n}||x||_1$$

BEWEIS. Zunächst gilt: $|x_j| \leq \sum_{j=1}^{n}|x_j| = ||x||_1$.

Desweiteren ist $|x_j| \leq \max_{j=1,2,...,n}|x_j|$, also $\frac{1}{n}\sum_{j=1}^{n}|x_j| \leq ||x||_\infty$.

Aus $|x_j|^p \leq \left(\sum_{j=1}^{n}|x_j|^p\right)$ folgt: $|x_j| \leq \left(\sum_{j=1}^{n}|x_j|^p\right)^{\frac{1}{p}} = ||x||_p$ und somit:

$$\max_{j=1,2,...,n}|x_j| \leq ||x||_p \Longleftrightarrow ||x||_\infty \leq ||x||_p.$$

Die vierte Ungleichung folgt aus dem Umstand:

$$||x||_p^p = \sum_{j=1}^{n}|x_j|^p \leq n \cdot \max\{|x_1|^p, |x_2|^p, ..., |x_n|^p\} = n \cdot ||x||_\infty^p$$

und die letzte Ungleichung aus:

$$||x||_\infty = \max_{j=1,2,...,n}|x_j| \leq \sum_{j=1}^{n}|x_j| = ||x||_1$$

□

DEFINITION 7. (Lineare Unabhängigkeit)

Seien $x_1, x_2, ..., x_m \in \mathbb{R}^n$. Die Vektoren $x_1, x_2, ..., x_m$ heißen *linear unabhängig*, falls gilt: Sind $\lambda_1, \lambda_2, ..., \lambda_m \in \mathbb{R}$ und ist $\sum_{i=1}^{m}\lambda_i x_i = 0$, so folgt $\lambda_1 = \lambda_2 = ... = \lambda_m = 0$.

Falls dies nicht gilt, spricht man von *Linearer Abhängigkeit*.

DEFINITION 8. (Erzeugnis, Spann, Lineare Hülle)

Seien $x_1, x_2, ..., x_m \in \mathbb{R}^n$. Die Menge

$$\text{span}(x_1, x_2, ..., x_m) := \sum_{i=1}^{m}\mathbb{R}x_i = \left\{\sum_{i=1}^{m}\lambda_i x_i \,|\, \lambda_i \in \mathbb{R},\, i = 1, 2, ..., m\right\}$$

nennt man das *Erzeugnis*, der *Spann* oder die *Lineare Hülle* von $x_1, x_2, ..., x_m$.

BEMERKUNG 9. Die $x_1, x_2, ..., x_m$ nennt man *Erzeugendensystem* des Spanns. Falls das Erzeugendensystem linear unabhängig ist, spricht man auch von einer *Basis*.

THEOREM 10. *Sei $V \subseteq \mathbb{R}^n$ und $x_1, x_2, ..., x_m \in \mathbb{R}^n$ eine Basis von V. Ist $y_1, y_2, ..., y_k \in \mathbb{R}^n$ eine weitere Basis von V. Dann muss gelten: $m = k$*

BEWEIS. (siehe **[J98]**) □

Dieser Satz ermöglicht den Begriff der Dimension zu definieren.

DEFINITION 11. (Dimension)

Ist $V \subseteq \mathbb{R}^n$ und $x_1, x_2, ..., x_m \in \mathbb{R}^n$ eine Basis von V. Dann sei:

$$\dim V := m$$

die *Dimension* von V.

DEFINITION 12. (Orthogonales Komplement)

Ist $V \subseteq \mathbb{R}^n$ ein Vektorraum. Dann heißt:

$$V^\perp := \{w \in \mathbb{R}^n \mid \langle w, v\rangle = 0 \text{ für alle } v \in V\}$$

das *orthogonale Komplement* von V.

Oft ist man an einer ganz speziellen Basis eines Vektorraums interessiert, einem Orthogonalsystem.

DEFINITION 13. (Orthogonalsystem, orthogonale Projektion)

Sei $V \subseteq \mathbb{R}^n$ und $x_1, x_2, ..., x_m \in \mathbb{R}^n$ eine Basis von V. Es bezeichnet

$$\pi_i : \mathbb{R}^n \to \text{span}(x_1, x_2, ..., x_{i-1})^\perp$$

die *orthogonale Projektion*, d.h. es gilt für alle $x \in \mathbb{R}^n$:

$$\begin{array}{rcl} \pi_i(x) & \in & \text{span}(x_1, x_2, ..., x_{i-1})^\perp \\ x - \pi_i(x) & \in & \text{span}(x_1, x_2, ..., x_{i-1}) \end{array}$$

Es bezeichne $\hat{x}_i := \pi_i(x_i)$. Man berechnet $\hat{x}_1, \hat{x}_2, ..., \hat{x}_m$ durch das *Gram-Schmidt-Verfahren*:

$$\hat{x}_1 := x_1$$

$$\hat{x}_i := x_i - \sum_{j=1}^{i-1} \mu_{i,j}\hat{x}_j \text{ für } i = 1, 2, ..., m$$

Dabei sind die *Gram-Schmidt-Koeffizienten* μ_{ij} erklärt durch:

$$\mu_{i,j} := \frac{\langle x_i, \hat{x}_j\rangle}{\langle \hat{x}_j, \hat{x}_j\rangle} = \frac{\langle x_i, \hat{x}_j\rangle}{||\hat{x}_j||^2}$$

Insbesondere gilt $\mu_{i,i} = 1$ und für $j > i$ ist $\mu_{i,j} = 0$. Die Vektoren $\hat{x}_1, \hat{x}_2, ..., \hat{x}_m$ nennt man ein *Orthogonalsystem* von V. Das Orthogonalsystem erlaubt es die ursprünglichen Vektoren x_i und die orthogonale Projektion $\pi_i(x)$ in folgender Form zu schreiben:

$$x_i = \sum_{j=1}^{i} \mu_{i,j}\hat{x}_j$$

$$\pi_i(x) = \sum_{j=i}^{m} \frac{\langle x, \hat{x}_j\rangle}{\langle \hat{x}_j, \hat{x}_j\rangle}\hat{x}_j \tag{1.2.1}$$

Bemerkung 14. Das so konstruierte Orthogonalsystem ist ebenfalls eine Basis von V, die sich dadurch auszeichnet, dass paarweise verschiedene $\hat{x}_i$ aufeinander senkrecht stehen, d.h. $\langle \hat{x}_i, \hat{x}_j \rangle = 0$ für alle $i \neq j$. Unter anderem ist $\hat{x}_i, \hat{x}_{i+1}, ..., \hat{x}_m$ eine Basis des $(m - i + 1)$ - dimensionalen Vektorraums $\pi_i(V)$.

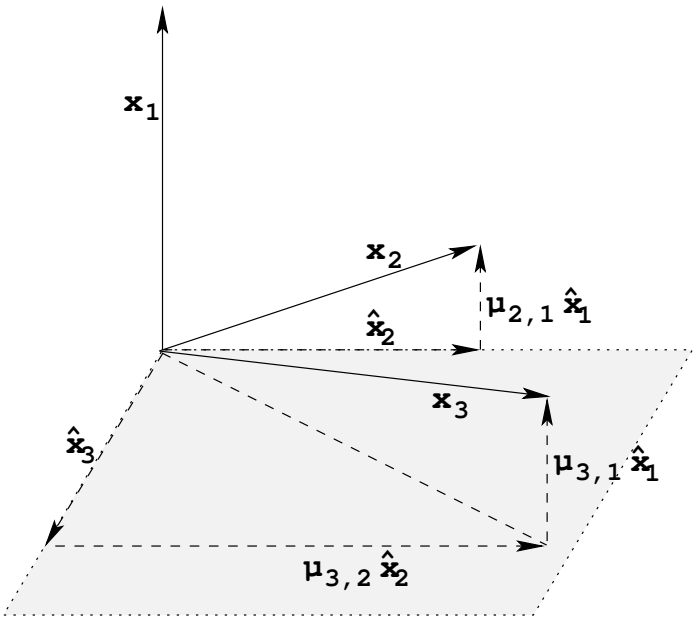


Abbildung 1.2.1. Basis x_1, x_2, x_3 und die zugehörige orthogonale Basis $\hat{x}_1, \hat{x}_2, \hat{x}_3$

Definition 15. (QR-Zerlegung)

Seien $x_1, x_2, ..., x_m \in \mathbb{R}^n$ linear unabhängige Vektoren und $\hat{x}_1, \hat{x}_2, ..., \hat{x}_m$ das zugehörige Orthogonalsystem. Die Darstellung der Basisvektoren im Orthogonalsystem lautet in Matrizenform:

$$X := [x_1 \ x_2 \ ... \ x_m] = [\hat{x}_1 \ \hat{x}_2 \ ... \ \hat{x}_m] \cdot [\mu_{i,j}]^T_{1 \leq i,j, \leq m}$$

Es sei nun $Q := [\frac{\hat{x}_1}{||\hat{x}_1||} \ \frac{\hat{x}_2}{||\hat{x}_2||} \ ... \ \frac{\hat{x}_m}{||\hat{x}_m||}] \in \mathbb{R}^{n \times m}$ und

$$R := \begin{bmatrix} ||\hat{x}_1|| & 0 & \dots & 0 \\ 0 & ||\hat{x}_2|| & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & ||\hat{x}_m|| \end{bmatrix} \cdot \begin{bmatrix} 1 & \mu_{2,1} & \mu_{3,1} & \dots & \mu_{m,1} \\ 0 & 1 & \mu_{3,2} & & \mu_{m,2} \\ \vdots & 0 & 1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \mu_{m,m-1} \\ 0 & \dots & \dots & 0 & 1 \end{bmatrix} \in \mathbb{R}^{m \times m}$$

Dann erhält man die eindeutige Zerlegung $X = Q \cdot R$, die auch *QR-Zerlegung* (von X) genannt wird. Insbesondere gilt:

$$\det(X^T X) = \det(R^T Q^T Q R) = \det(R^T R) = \prod_{i=1}^{m} ||\hat{x}_i||$$

BEMERKUNG 16. Die lineare Abbildung $T : \mathbb{R}^n \to \mathbb{R}^m$, $T(x) := Qx$ ist eine Isometrie, d.h. für alle $x, y \in \mathbb{R}^n$ gilt: $\langle x, y\rangle = \langle T(x), T(y)\rangle$.

Dieser kurze Ausflug in die Lineare Algebra endet mit der Methode, die das Kernstück sämtlicher gittertheoretischer Lösungsverfahren bildet.

ALGORITHMUS 17. (Gram-Schmidt Verfahren)

```
COMPUTE_GS([x_1 x_2 ... x_m], k)
{
  for (j = 1, 2, ..., k − 1) {
    G_{k,j} ⟵ ⟨x_k, x_j⟩

    t ⟵ 0

    for (i = 1, 2, ..., j − 1)
      t ⟵ t + μ_{j,i} · μ_{k,i}

    μ_{k,j} ⟵ (G_{k,j} − t)/c_j
  }

  t ⟵ 0

  for (j = 1, 2, ..., k − 1)
    t ⟵ t + μ²_{k,i}

  c_k ⟵ ⟨x_k, x_k⟩ − t

  return (μ, c)
}

COMPUTE_GS([x_1 x_2 ... x_m])
{
  for (i = 1, 2, ..., m)
    (μ, c) ⟵ COMPUTE_GS([x_1 x_2 ... x_m], i)

  return (μ, c)
}
```

BEMERKUNG 18. Obige Algorithmen berechnen die Gram-Schmidt Koeffizientenmatrix μ und den Vektor c, der die (quadrierten) euklidischen Längen der orthogonalen Basisvektoren beinhaltet.

1.3. Einführung in die Gittertheorie

Im Folgenden werden die wesentlichen Begriffe und Konzepte der Gittertheorie eingeführt und elementare Eigenschaften bewiesen.

DEFINITION 19. (Gitter)

Zu linear unabhängigen Vektoren $b_1, b_2, ..., b_m \in \mathbb{R}^n$ heißt die Menge

$$L(b_1, b_2, ..., b_m) := \sum_{i=1}^{m} \mathbb{Z} b_i = \left\{ \sum_{i=1}^{m} t_i b_i \,|\, t_i \in \mathbb{Z},\, i = 1, 2, ..., m \right\}$$

Gitter mit (geordneter) Basis $b_1, b_2, ..., b_m$ und *Rang* m. Die $n \times m$-Matrix

$$B := \Big[b_1 \; b_2 \; ... \; b_m \Big]$$

wird als *Basismatrix* bezeichnet. $L(B)$ nennt man das von den Spaltenvektoren der Matrix B erzeugte Gitter. span(L) bezeichnet die *Lineare Hülle* von $L(b_1, b_2, ..., b_m)$.

BEMERKUNG 20. Den Rang m nennt man auch die *Dimension des Gitters*. Weiterhin gilt $m \leq n$, da es in einem Raum der Dimension n, höchstens n linear unabhängige Vektoren geben kann. Falls der Rang m mit der Dimension n des Raumes übereinstimmt, spricht man auch von einem *volldimensionalem* oder *vollständigem* Gitter.

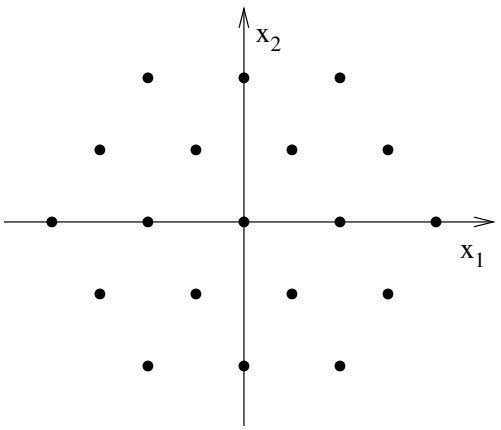


ABBILDUNG 1.3.1. Das 2-dimensionale, hexagonale Gitter
$\mathbb{Z}(1,0)^T + \mathbb{Z}(1/2, \sqrt{3}/2)^T$

DEFINITION 21. (Grundmasche)

Die *Grundmasche* eines Gitters $L \subseteq \mathbb{R}^n$ vom Rang m mit der Basis $b_1, b_2, ..., b_m$ ist das Parallelepiped

$$P(b_1, b_2, ..., b_m) := \left\{ \sum_{i=1}^{m} x_i b_i \,|\, x_i \in [0, 1) \text{ für alle } i = 1, ..., m \right\}$$

BEMERKUNG 22. Offenbar besitzt jeder Punkt $x \in$ span(L) eine eindeutige Zerlegung der Form $x = p + v$ mit $p \in P$ und $v \in L$. Es gilt: span$(L) = P + L$

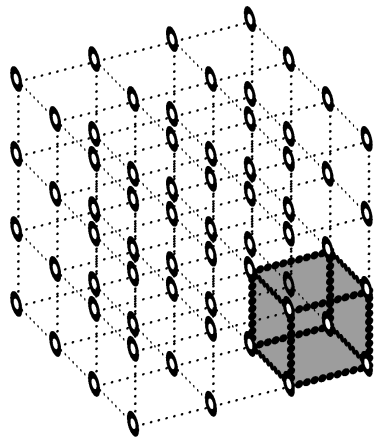


ABBILDUNG 1.3.2. Die Grundmasche des Gitters $\mathbb{Z}^3$

Ein typisches Gitter ist der ganzzahlige Lösungsraum eines linearen, reellen Gleichungssystems. Der größere reelle Lösungsraum ist ein linearer Raum. Damit bilden Gitter ein diskretes Analogon zu linearen Räumen und lassen sich auch wie folgt definieren:

DEFINITION 23. (Diskrete Menge)

Eine Menge $M \subseteq \mathbb{R}^n$ heißt *diskret*, wenn M keinen Häufungspunkt in $\mathbb{R}^n$ hat, also wenn für alle $m \in M$ und alle $\epsilon > 0$ gilt:

$$|\{x \in \mathbb{R}^n : ||x||_2 < \epsilon \cap M\}| < \infty$$

LEMMA 24. *Eine additive Gruppe $G \subseteq \mathbb{R}^n$ ist genau dann diskret, wenn 0 kein Häufungspunkt von G ist.*

BEWEIS. Es ist lediglich zu zeigen: Wenn $x \in \mathbb{R}^n$ ein Häufungspunkt von G ist, dann ist 0 ebenfalls ein Häufungspunkt von G. Sei also $(x_k)_{k\in\mathbb{N}}$ eine Folge paarweise verschiedener Vektoren von G, die gegen x konvergiert. Dann ist $y_k := x_k - x_{k+1} \in G$ (da G additive Gruppe ist) eine Folge mit $y_k \neq 0$ für alle $k \in \mathbb{N}$ und $\lim\limits_{k\longrightarrow\infty} y_k = 0$. Also ist 0 ein Häufungspunkt. □

THEOREM 25. *Sei $L \subseteq \mathbb{R}^n$. Dann gilt die folgende Äquivalenz:*

L ist ein Gitter $\Leftrightarrow$ L ist eine diskrete, additive Untergruppe des $\mathbb{R}^n$

BEWEIS.

„$\Longrightarrow$“: $b_1, b_2, ..., b_m$ sei die Gitterbasis von L. Die Abbildung

$$\phi : \mathbb{R}^m \longrightarrow \text{span } L,\ (t_1, t_2, ..., t_m) \mapsto \sum_{i=1}^{m} t_i b_i$$

ist ein Isomorphismus mit $\phi(\mathbb{Z}^m) = L$. Da ϕ^{-1} stetig auf span L ist, und $\mathbb{Z}^m$ diskret ist, muss auch L diskret sein.

„$\Longleftarrow$“: Es sei m die Maximalzahl von linear unabhängigen Vektoren in L.

Man beweist die Aussage durch Induktion über m:

$m = 1$: Sei $b \in L$ ein kürzester Vektor $\neq 0$ (solch ein Vektor existiert, da 0 kein Häufungspunkt ist). Dann gilt: $L(b) = L$. Denn gäbe es einen Vektor $b' \in L$ mit $b' \notin L(b)$. Dann müsste aufgrund von $m = 1$ gelten: $b' = \alpha \cdot b$, mit $\alpha \in \mathbb{R}\backslash\mathbb{Z}$. Dann wäre aber $b'' := b' - \lfloor\alpha\rfloor \cdot b = (\alpha - \lfloor\alpha\rfloor) \cdot b \in L$ und damit $||b''||_2 < ||b||_2$, im Widerspruch zur Wahl von b.

$m > 1$: Wähle zunächst $b_1 \in L\backslash\{0\}$, so dass $\frac{1}{k}b_1 \notin L$ für alle $k \geq 2$. Damit gilt: $L(b_1) = \text{span}(b_1) \cap L$.

Nun ist zu zeigen, dass $\pi_2(L)$ diskret ist: Hierzu sei die gegenteilige Behauptung angenommen, d.h. 0 ist ein Häufungspunkt von $\pi_2(L)$. Es sei $\big(\pi_2(v_k)\big)_{k\in\mathbb{N}}$ eine Folge von paarweise verschiedenen Vektoren aus $\pi_2(L)$, die gegen 0 konvergiert. Es gilt:

$$\pi_2(v_k) = v_k - \frac{\langle v_k, b_1\rangle}{\langle b_1, b_1\rangle} b_1.$$

Nun betrachtet man die Folge $w_k := v_k - \left\lfloor \frac{\langle v_k, b_1\rangle}{\langle b_1, b_1\rangle} \right\rfloor b_1$, die aus paarweise verschiedenen Vektoren von L besteht, da $\pi_2(v_k) = \pi_2(w_k)$ ist.

Für alle $k \in \mathbb{N}$ besitzt die Ungleichung $||w_k - \pi_2(v_k)||_2 \leq ||b_1||_2$ Gültigkeit, so dass die Folge $\big(w_k - \pi_2(v_k)\big)_{k\in\mathbb{N}}$ beschränkt ist. Nach dem Satz von Bolzano-Weiherstrass, besitzt sie damit eine konvergente Teilfolge $\big(w_{k_j} - \pi_2(v_{k_j})\big)_{j\in\mathbb{N}}$. Insbesondere muss dann aber die Folge $(w_{k_j})_{j\in\mathbb{N}}$ konvergieren, welches jedoch im Widerspruch zur Diskretheit von L steht.

Die Maximalzahl der linear unabhängigen Vektoren in $\pi_2(L)$ ist $m-1$. Folglich ist $\pi_2(L)$ (aufgrund der Induktionsvoraussetzung) ein Gitter vom Rang $m-1$. Es sei $\pi_2(b_2), \pi_2(b_3), ..., \pi_2(b_m)$ eine Basis von $\pi_2(L)$ mit $b_2, b_3, ..., b_m \in L$.

Es bleibt noch zu zeigen: $L \subseteq L(b_1, b_2, ..., b_m)$. Dazu sei $b \in L$. Wegen $\pi_2(b) \in \pi_2(L) = L\big(\pi_2(b_2), \pi_2(b_3), ..., \pi_2(b_m)\big)$ gibt es ein $\overline{b} \in L(b_2, b_3, ..., b_m)$ mit $\pi_2(b) = \pi_2(\overline{b})$. Es gilt $(b - \overline{b}) \in \text{span}(b_1)$. Nach Wahl von b_1 ist $L(b_1) = \text{span}(b_1) \cap L$ und somit $(b - \overline{b}) \in L(b_1)$. Es folgt $b \in L(b_1, b_2, ..., b_m)$.

□

Im Unterschied zu Vektorräumen, sind die Bilder von Gittern unter linearen Abbildungen im allgemeinen keine Gitter mehr. Ein Beispiel hierfür wäre das Bild des Gitters $\mathbb{Z}^2$ unter der linearen Abbildung

$$f : \mathbb{R}^2 \to \mathbb{R},\ (x_1, x_2) \mapsto x_1 + \sqrt{2}x_2.$$

Zwar ist $f(\mathbb{Z}^2)$ eine additive Untergruppe von $\mathbb{R}$, aber Diskretheit ist aufgrund der Irrationalität von $\sqrt{2}$ nicht mehr gegeben.

Es gibt jedoch Abbildungen, die die Gittereigenschaft erhalten, was der nächste Satz belegen wird.

THEOREM 26. *Es sei $L \subseteq \mathbb{R}^n$ ein Gitter der Dimension m und $b_1, b_2, ..., b_d \in L$ seien linear unabhängige Vektoren. Dann ist*

$$M_d := \pi_{(\mathbb{Z}b_1+\mathbb{Z}b_2+...+\mathbb{Z}b_d)^\perp}(L)$$

ein $(m-d)$-dimensionales Gitter.

BEWEIS. Es sei $b_0 := 0 \in \mathbb{R}^n$ und $\pi_{d+1} := \pi_{(\mathbb{Z}b_1+\mathbb{Z}b_2+...+\mathbb{Z}b_d)^\perp}$. Man beweist die Aussage durch Induktion über d:

$d = 0$: Da $M_0 = \pi_1(L) = L$, ist M_0 ein m-dimensionales Gitter.

$d > 0$: Da M_d das Bild der additiven Untergruppe $L \subseteq \mathbb{R}^n$ unter der linearen Abbildung π_{d+1} ist, ist M_d selbst eine additive Untergruppe des $\mathbb{R}^n$. Es bleibt zu zeigen, dass M_d diskret ist: Angenommen $0 \in \mathbb{R}^n$ ist ein Häufungspunkt von M_d. Es sei $\left(\pi_{d+1}(v_k)\right)_{k\in\mathbb{N}}$ eine Folge von paarweise verschiedenen Vektoren aus M_d, die gegen 0 konvergiert. Es gilt:

$$\pi_{d+1}(v_k) = \pi_d(v_k) - \frac{\langle v_k, \hat{b}_d\rangle}{\langle \hat{b}_d, \hat{b}_d\rangle}\hat{b}_d$$

Es sei nun

$$w_k := \pi_d(v_k) - \left\lfloor \frac{\langle v_k, \hat{b}_d\rangle}{\langle \hat{b}_d, \hat{b}_d\rangle} \right\rfloor \hat{b}_d$$

Die Folge $(w_k)_{k\in\mathbb{N}}$ besteht aus paarweise verschiedenen Vektoren von M_{d-1}, denn $\pi_{d+1}(v_k) = \pi_{d+1}(w_k)$.

Für alle $k \in \mathbb{N}$ besitzt die Ungleichung $||w_k - \pi_{d+1}(v_k)||_2 \leq ||\hat{b}_d||_2$ Gültigkeit, so dass die Folge $\left(w_k - \pi_{d+1}(v_k)\right)_{k\in\mathbb{N}}$ beschränkt ist. Nach dem Satz von Bolzano-Weiherstrass, besitzt sie damit eine konvergente Teilfolge $\left(w_{k_j} - \pi_{d+1}(v_{k_j})\right)_{j\in\mathbb{N}}$. Insbesondere muss dann aber die Folge $(w_{k_j})_{j\in\mathbb{N}}$ konvergieren, welches jedoch im Widerspruch zur Diskretheit von M_{d-1} steht.

Aus der Dimensionsformel für lineare Abbildungen folgt schließlich: $\dim M_d = m - d$.

□

DEFINITION 27. (Untergitter)

Seien $L_1, L_2 \subseteq \mathbb{R}^n$ Gitter. L_2 heißt *Untergitter* von L_1, falls $L_2 \subseteq L_1$. Das Untergitter L_2 hat vollen Rang, wenn beide Gitter den gleichen Rang besitzen.

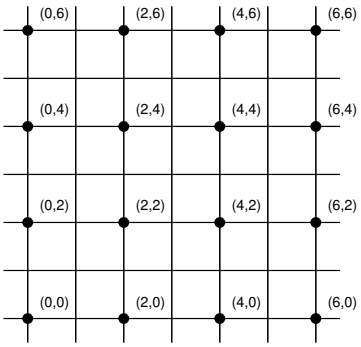


ABBILDUNG 1.3.3. Das Untergitter $L((2,0)^T,(0,2)^T)$ von $\mathbb{Z}^2$

BEMERKUNG 28. Ist B_1 eine Basismatrix von L_1 vom Rang m_1 und B_2 eine Basismatrix von L_2 vom Rang m_2, so existiert eine ganzzahlige $m_1 \times m_2$ Matrix T mit $B_2 = B_1T$. Denn da L_2 ein Untergitter von L_1 ist, lassen sich die Spaltenvektoren von B_2 durch ganzzahlige Linearkombinationen der Spaltenvektoren von B_1 ausdrücken.

Die nun folgende Definition und der nächste Satz stellen eine Beziehung zwischen verschiedenen Basismatrizen des gleichen Gitters her.

DEFINITION 29. (Unimodulare Matrix)

Eine ganzzahlige, quadratische Matrix U heißt *unimodular*, wenn gilt:

$$|\det(U)| = 1$$

THEOREM 30. *Sei $L := L(B) \subseteq \mathbb{R}^n$ ein Gitter vom Rang m. Eine $n \times m$-Matrix B' ist genau dann eine Basismatrix des Gitters L, wenn eine unimodulare $m \times m$-Matrix T mit $B' = BT$ existiert.*

BEWEIS. (siehe **[S02]**) □

BEMERKUNG 31. Eine direkte Folgerung aus diesem Satz ist, dass es für jedes Gitter unendlich viele verschiedene Basen gibt (wenn Rang $L > 1$).

Desweiteren ist es nun möglich die Gitterdeterminate zu definieren. Diese Größe ist von der gewählten Basis des Gitters unabhängig, d.h. sie ist eine Invariante.

DEFINITION 32. (Gitterdeterminante)

Die *Gitterdeterminante* eines Gitters $L \subseteq \mathbb{R}^n$ ist erklärt als

$$\det L := \sqrt{\det(B^T B)}$$

für eine Basismatrix B des Gitters.

Diese Größe ist deshalb invariant, weil (aufgrund von Satz 30) für eine weitere Basismatrix B' des Gitters L eine unimodulare Matrix T mit $B' = BT$ existiert und folglich ist:

$$\det(B'^T B') = \det\big((BT)^T(BT)\big) = \det(T)^2 \cdot \det(B^T B) = \det(B^T B)$$

BEMERKUNG 33. Die Matrix $G := B^T B$ heißt die *Gram-Matrix* zur Basismatrix B. Die Gram-Matrix G ist mindestens positiv semidefinit und sogar positiv definit, falls die Matrix B maximalen Rang hat. Man nennt $\det(G)$, also $(\det L)^2$ die *Diskriminante* des Gitters L.

KOROLLAR 34. Sei $L_1 \subseteq \mathbb{R}^n$ ein Gitter und L_2 ein Untergitter von L_1 mit vollem Rang. Weiter sei T die ganzzahlige, quadratische Matrix mit $B_2 = TB_1$ (wobei B_1 bzw. B_2 die Basismatrizen von L_1 bzw. L_2 seien). Dann ist:

$$\det L_2 = \det L_1 \cdot |\det(T)|$$

BEMERKUNG 35. Die ganze Zahl $|\det(T)|$ wird auch als *Index* des Untergitters L_2 in L_1 bezeichnet. Insbesondere ist L_2 eine Untergruppe der additiven Gruppe L_1 vom Index $[L_1 : L_2] = |\det(T)|$.

THEOREM 36. *Für jedes Gitter $L = L(b_1, b_2, ..., b_m) \subseteq \mathbb{R}^n$ gilt:*

$$det\ L = vol_m\big(P(b_1, b_2, ..., b_m)\big)$$

D.h. das m-dimensionale Volumen der Grundmasche stimmt mit dem Wert der Gitterdeterminante überein.

BEWEIS. (siehe **[S02]**) □

KOROLLAR 37. Es gilt: $\det L = \prod_{i=1}^{m} ||\hat{b}_i||_2$

BEWEIS. Die *QR-Zerlegung* von B liefert: $B = \hat{B} \cdot \mu^T$. Wir erhalten:

$$\det L = \det(B^T B) = \det(\mu \cdot \hat{B}^T \hat{B} \cdot \mu^T) = \det(\mu) \cdot \det(\hat{B}^T \hat{B}) \cdot \det(\mu^T)$$

Da die Dreiecksmatrix μ nur Einsen auf der Hauptdiagonale stehen hat, folgt: $\det L = \det(\hat{B}^T \hat{B})$. Und da die Vektoren $\hat{b_i}$ paarweise senkrecht aufeinander stehen, lässt sich schließen: $\det L = \prod_{i=1}^{m} ||\hat{b}_i||_2$. □

PROPOSITION 38. *(Ungleichung von Hadamard): Es sei $L \subseteq \mathbb{R}^n$ ein Gitter mit Basis $b_1, b_2, ..., b_m$. Dann gilt:*

$$|det\ L| \leq \prod_{i=1}^{m} ||b_i||_2$$

BEWEIS. Aufgrund der Orthogonalität der Vektoren $\hat{b}_j$ erhält man aus dem Gram-Schmidt'schen Orthogonalisierungsverfahren:

$$||b_i||_2^2 = ||\hat{b}_i||_2^2 + \sum_{j=1}^{i-1} \mu_{i,j}^2 ||\hat{b}_j||_2^2 \geq ||\hat{b}_i||_2^2.$$

Mit $\det L = \prod_{i=1}^{m} ||\hat{b}_i||_2$ folgt die gewünschte Ungleichung. □

DEFINITION 39. (Orthogonaler Defekt)

Sei $L \subseteq \mathbb{R}^n$ ein Gitter. Dann wird zu einer Basis $b_1, b_2, ..., b_m$ der Quotient

$$\nabla(L) := \frac{||b_1||_2 \cdot ||b_2||_2 \cdot ... \cdot ||b_m||_2}{\det L}$$

der *orthogonale Defekt* genannt.

BEMERKUNG 40. Mit der Ungleichung von Hadamard folgt, dass der orthogonale Defekt immer ≥ 1 ist. Er dient als Maß dafür, wie weit eine Basis davon entfernt ist, ein Orthogonalsystem zu sein. Der orthogonale Defekt ist genau dann 1, wenn die Basis orthogonal ist.

Als nächstes wird der Begriff des dualen Gitters definiert:

DEFINITION 41. (Duales Gitter)

Sei $L \subseteq \mathbb{R}^n$ ein Gitter. Das zu L *duale Gitter* ist definiert durch

$$L^\star := \{x \in \text{span}(L) \mid \langle x, y\rangle \in \mathbb{Z} \text{ für alle } y \in L\}$$

BEMERKUNG 42. Ein Gitter mit $L = L^\star$ nennt man auch *selbstdual* oder *unimodular*. Das duale Gitter wird manchmal auch als *reziprokes* oder *polares* Gitter bezeichnet. Ein triviales Beispiel für ein selbstduales Gitter ist $L = \mathbb{Z}^n$.

Zwischen dem Gitter L und dem dualen Gitter $L^\star$ existieren interessante Beziehungen, die im nächsten Satz dargelegt werden.

THEOREM 43. *Sei L ein Gitter mit zugehöriger Basismatrix $B \in \mathbb{R}^{n\times m}$ und der Gram-Matrix $G = B^T B$. Weiter sei $B = QR$ die QR-Zerlegung von B. Dann gelten:*

(1) $B^\star := Q(R^{-1})^T$ ist die Basismatrix des Gitters $L^\star$.
(2) G^{-1} ist die Gram-Matrix zur Basismatrix $B^\star$.
(3) L und $L^\star$ haben den gleichen Rang.
(4) $\det L^\star = \frac{1}{\det L}$
(5) $(L^\star)^\star = L$

BEWEIS. (siehe **[S02]**) □

BEMERKUNG 44. Die Basismatrix $B^\star$ nennt man die *zu B duale Basismatrix*. Im Fall volldimensionaler Gitter ist die zu B duale Basismatrix gegeben durch $B^\star := (B^{-1})^T$. Ein Gitter vollen Ranges ist genau dann selbstdual, wenn es eine orthogonale Basismatrix des Gitters gibt.

LEMMA 45. *Es gilt: $\langle b_i, b_k^\star\rangle = \delta_{i,k}$ für alle $1 \leq i, k \leq m$*

BEWEIS. Nach dem vorherigen Theorem erhält man:

$$B^T \cdot B^\star = R^T Q^T \cdot Q(R^{-1})^T = R^T \cdot (R^{-1})^T = I_m^T = I_m$$

□

THEOREM 46. *Wenn $B \in \mathbb{R}^{n\times m}$ die Basismatrix eines m-dimensionalen Gitters ist, $b_k^\star$ der zum Basisvektor b_k duale Basisvektor und $x \in \mathbb{R}^m$ ein beliebiger Koeffizientenvektor, dann gilt:*

$$x_k = \langle b_k^\star, Bx\rangle$$

BEWEIS. Nach dem Lemma ist:

$$\langle b_k^\star, b_i\rangle = \delta_{k,i} \text{ für alle } 1 \leq i, k \leq m$$

Folglich gilt für eine Linearkombination $b = \sum_{i=1}^{m} x_i b_i$ der Basisvektoren:

$$\langle b_k^\star, b\rangle = \langle b_k^\star, \sum_{i=1}^{m} x_i b_i\rangle = \sum_{i=1}^{m} x_i \cdot \langle b_k^\star, b_i\rangle = \sum_{i=1}^{m} x_i \cdot \delta_{k,i} = x_k$$

Also:

$$\langle b_k^\star, Bx\rangle = x_k$$

□

THEOREM 47. *Es gilt sogar eine etwas abgewandelte Variante:*

$$x_k = \langle b_k^\star, \pi_k(Bx)\rangle$$

BEWEIS. Es existiert ein $\lambda \in \mathbb{R}^m$ so, dass

$$Bx = \sum_{i=1}^{m} \lambda_i \hat{b}_i \text{ und } \pi_k(Bx) = \sum_{i=k}^{m} \lambda_i \hat{b}_i$$

Folglich lässt sich $Bx - \pi_k(Bx)$ schreiben als:

$$Bx - \pi_k(Bx) = \sum_{i=1}^{k-1} \lambda_i \hat{b}_i$$

Da $\langle b_k^\star, \hat{b}_i\rangle = 0$ für alle $1 \leq i \leq k-1$, folgt

$$\langle b_k^\star, Bx - \pi_k(Bx)\rangle = \langle b_k^\star, \sum_{i=1}^{k-1} \lambda_i \hat{b}_i\rangle = 0$$

Somit ist:

$$\begin{aligned} &\langle b_k^\star, Bx - \pi_k(Bx)\rangle = 0 \\ \Leftrightarrow\ &\langle b_k^\star, Bx\rangle = \langle b_k^\star, \pi_k(Bx)\rangle \\ \Leftrightarrow\ &x_k = \langle b_k^\star, \pi_k(Bx)\rangle \end{aligned}$$

□

THEOREM 48. *(Blichfeldt)*

Es sei $L \subseteq \mathbb{R}^n$ ein m-dimensionales Gitter und $M \subseteq span(L)$ eine Menge für die $vol_m(M)$ definiert ist. Ist $vol_m(M) > \det L$, dann existieren zwei verschiedene Punkte $m_1, m_2 \in M$ mit $m_1 - m_2 \in L$.

BEWEIS. Es bezeichne P die Grundmasche des Gitters und für einen beliebigen Gittervektor sei $M_v := M \cap (P + v)$. Da $(P + v) \cap (P + w) = \emptyset$ für $v, w \in L$ mit $v \neq w$ und L abzählbar ist, gilt:

$$\mathrm{vol}_m(M) = \sum_{v \in L} \mathrm{vol}_m(M_v)$$

Es sei nun $M_v' := M_v - v = (M - v) \cap P$.

Dann ist $M_v' \subseteq P$ und $\mathrm{vol}_m(M_v') = \mathrm{vol}_m(M_v)$. Des Weiteren sind die Mengen M_v' nicht paarweise disjunkt. Um dies einzusehen, wird von der gegenteiligen Behauptung ausgegangen, d.h.

$$\sum_{v \in L} \mathrm{vol}_m(M_v') = \mathrm{vol}_m(\dot{\cup} M_v') \leq \mathrm{vol}_m(P)$$

Weiterhin ergibt sich $\sum_{v\in L} \mathrm{vol}_m(M'_v) = \sum_{v\in L} \mathrm{vol}_m(M_v) = \mathrm{vol}_m(M)$. Also ist $\mathrm{vol}_m(M) \leq \mathrm{vol}_m(P) = \det L$. Dies steht allerdings im Widerspruch zur Voraussetzung, dass $\mathrm{vol}_m(M) > \det L$.

Man betrachte nun zwei unterschiedliche Vektoren $v, w \in L$ mit $M'_v \cap M'_w \neq \emptyset$. Für $m \in M'_v \cap M'_w$ gilt dann $m \in M'_v$ und $m \in M'_w$, d.h. es existieren $m_1 \in M_v, m_2 \in M_w$ mit $m = m_1 - v$ und $m = m_2 - w$. Da v ungleich w folgt $m_1 \neq m_2$ und es ergibt sich:

$$m_1 - m_2 = (m + v) - (m + w) = v - w \in L\backslash\{0\}$$

□

THEOREM 49. *(Minkowski)*

Es sei $L \subseteq \mathbb{R}^n$ ein m-dimensionales Gitter und $S \subseteq span(L)$ eine konvexe Menge, die symmetrisch zum Ursprung ist. Wenn $vol_m(S) > 2^m \cdot det\ L$, dann enthält S einen von Null verschiedenen Gitterpunkt $v \in S \cap L$.

BEWEIS. Man betrachte die Menge $S' := \{x \mid 2x \in S\}$.

Für das Volumen von S' gilt:

$$\mathrm{vol}_m(S') = 2^{-m} \cdot \mathrm{vol}_m(S) > \det L$$

Nach Theorem 48 existieren zwei Punkte $s_1, s_2 \in S'$ mit $s_1 - s_2 \in L\backslash\{0\}$. Damit sind $2s_1 \in S$ und $2s_2 \in S$. Es gilt sogar $-2s_2 \in S$, weil S symmetrisch zum Ursprung ist. Aufgrund der Konvexität von S folgt schließlich:

$$s_1 - s_2 = \frac{1}{2}(2s_1 + (-2s_2)) \in S \Rightarrow s_1 - s_2 \in S \cap L\backslash\{0\}$$

□

ASSUMPTION 50. *(Gauss'sche Heuristik)*

Es sei L ein m-dimensionales Gitter und M eine Menge für die $vol_m(M)$ definiert ist. Dann gilt mit großer Wahrscheinlichkeit für die Anzahl der Gitterpunkte in $L \cap M$:

$$|L \cap M| \approx \frac{vol_m(M)}{det\ L}$$

BEMERKUNG 51. Die Gauss'sche Heuristik läßt sich leider nicht allgemein beweisen. Es zeigt sich aber, dass sie insbesondere bei der Abschätzung des Aufwands von Enumerationsverfahren, erstaunlich nahe an den praktischen Ergebnissen liegt **[G10]**.

DEFINITION 52. (Sukzessive Minima)

Es sei L ein m-dimensionales Gitter. Dann sind die *sukzessiven Minima* für $i = 1, 2, ..., m$ definiert durch:

$$\lambda_i^{(p)} = \lambda_i^{(p)}(L) := \min \left\{ r > 0 \;\middle|\; \begin{array}{c} \text{Es gibt } i \text{ linear unabhängige} \\ \text{Vektoren } c_1, c_2, ..., c_i \in L \\ \text{mit } ||c_j||_p \leq r \text{ für } j = 1, 2, ..., i \end{array} \right\}$$

BEMERKUNG 53. Die $\lambda_i^{(p)}$ geben die Länge des kürzesten Gittervektors wieder, der linear unabhängig von den $i - 1$ Vektoren ist, die das vorherige Minimum bestimmen. Insbesondere entspricht $\lambda_1^{(p)}$ der p-Norm des kürzesten Gittervektors und es gilt: $\lambda_1^{(p)} \leq \lambda_2^{(p)} \leq ... \leq \lambda_m^{(p)}$.

THEOREM 54. *Es sei L ein m-dimensionales Gitter und $b_1, b_2, ..., b_m$ die zugehörige Basis. Dann gilt:*

$$\lambda_j^{(2)} \geq \min_{i=j,j+1,...,m} ||\hat{b}_i||_2 \text{ für } j = 1, 2, ..., m$$

BEWEIS. Nach Definition der sukzessiven Minima gibt es linear unabhängige Vektoren $c_1, c_2, ..., c_m \in L$ mit $\lambda_i^{(2)}(L) = ||c_i||_2$, $i = 1, 2, ..., m$:

$$c_k = \sum_{i=1}^{m} t_{i,k} b_i = \sum_{i=1}^{m} \hat{t}_{i,k} \hat{b}_i \text{ für } k = 1, 2, ..., m$$

Die Koeffizienten $t_{i,k}$ sind ganzzahlig und die $\hat{t}_{i,k}$ reell.

Es sei $\mu(k) := \max\{i \,|\, t_{i,k} \neq 0\}$. Wegen $b_i = \sum_{j=1}^{i} \mu_{i,j} \hat{b}_j$ und $\mu_{i,i} = 1$ ist $\hat{t}_{\mu(k),k} = t_{\mu(k),k}$ und aufgrund der linearen Unabhängigkeit der Vektoren $c_1, c_2, ..., c_j$ gibt es zu jedem j ein $k \leq j$ mit $\mu(k) \geq j$. Wäre dem nicht so, d.h. $\mu(k) < j$ für alle $k = 1, 2, ..., j$, dann wäre: $c_1, c_2, ..., c_j \in \text{span}(b_1, b_2, ..., b_{j-1}) \Rightarrow c_1, c_2, ..., c_j$ sind linear abhängig – ein Widerspruch.

Aus $k \leq j$ mit $\mu(k) \geq j$ lässt sich damit folgern:

$$\left(\lambda_j^{(2)}\right)^2 \geq \left(\lambda_k^{(2)}\right)^2 = ||c_k||_2^2 \geq \hat{t}_{\mu(k),k}^2 \cdot ||\hat{b}_{\mu(k)}||_2^2 \geq ||\hat{b}_{\mu(k)}||_2^2 \geq \min_{i=j,j+1,...,m} ||\hat{b}_i||_2$$

□

THEOREM 55. *Es sei L ein m-dimensionales Gitter. Dann gelten:*

(1) $\lambda_1^{(\infty)}(L) < (\det L)^{\frac{1}{m}}$
(2) $\lambda_1^{(2)}(L) < \sqrt{m} \cdot (\det L)^{\frac{1}{m}}$

BEWEIS. Es sei $S := B\left(0, (\det L)^{\frac{1}{m}}\right) \subseteq \text{span}(L)$ die m-dimensionale Kugel mit Radius $(\det L)^{\frac{1}{m}}$ bezüglich der Maximumsnorm.

Die Menge S ist konvex, symmetrisch zum Ursprung und es gilt:

$$\text{vol}_m(S) = \left(2 \cdot (\det L)^{\frac{1}{m}}\right)^m = 2^m \cdot (\det L)$$

Nach Theorem 49 existiert daher ein Gittervektor $v \in L \backslash \{0\}$ mit $v \in S \Rightarrow ||v||_\infty < (\det L)^{\frac{1}{m}}$. Also ist $\lambda_1^{(\infty)}(L) < (\det L)^{\frac{1}{m}}$. Die zweite Behauptung folgt aus Theorem 6, da für alle $v \in L$ gilt: $||v||_2 \leq \sqrt{m}||v||_\infty$. □

DEFINITION 56. (Hermite-Funktion)

Es bezeichne $\mathcal{L}_m$ die Menge aller m-dimensionalen Gitter. Die *Hermite-Funktion* $\gamma : \mathcal{L}_m \to \mathbb{R}_+$ ist definiert durch:

$$\gamma(L) := \frac{\lambda_1(L)^2}{(\det\, L)^{\frac{2}{m}}}$$

Das Supremum der Hermite-Funktion über $\mathcal{L}_m$ wird *Hermite-Konstante* genannt und ist definiert als:

$$\gamma_m := \sup\{\gamma(L) \,|\, L \in \mathcal{L}_m\}$$

THEOREM 57. *Es sei $L \subseteq \mathbb{R}^n$ ein m-dimensionales Gitter. Dann gelten:*

(1) $\prod_{i=1}^{m} \lambda_i(L) \leq \gamma_m^{\frac{m}{2}} \cdot \det L$
(2) $\det L \leq \prod_{i=1}^{m} \lambda_i(L)$

BEWEIS. (siehe **[S02]**) □

1.4. Ganzzahlige, lineare Gleichungssysteme

Bei all dieser Gittertheorie, stellt sich unweigerlich die Frage, worin denn nun der praktische Nutzen dieser Strukturen liegt. Um dies zu beantworten, bedarf es zunächst eines kleinen Exkurses in die Theorie der ganzzahligen, linearen Gleichungssysteme.

Es seien $h, k \in \mathbb{N}$ mit $h \leq k$, $A = (a_{ij}) \in \mathbb{Z}^{h \times k}$ mit $\text{rang}(A) = h$ und $d \in \mathbb{Z}^h$. Weiter seien $l, r \in \mathbb{Z}^k$ mit $l \leq r$, d.h. $l_j \leq r_j$ für $j = 1, 2, ..., k$. Man erhält damit das folgende ganzzahlige, lineare Entscheidungsproblem:

Existiert ein Vektor $x \in \mathbb{Z}^k$, so dass $A \cdot x = d$ und $l \leq x \leq r$?

Durch die Ersetzung $x := x - l$, $d := d - A \cdot l$ und $r := r - l$ resultiert die äquivalente Formulierung:

Existiert ein Vektor $x \in \mathbb{Z}^k$, so dass $A \cdot x = d$ und $0 \leq x \leq r$? (FP)

Einen geeigneten Lösungsvektor x zu bestimmen, ist ein nicht-triviales Unterfangen - es ist nämlich kein Algorithmus bekannt, der das Problem in polynomialer Rechenzeit löst - es ist $\mathcal{NP}$-vollständig **[G79]**. Wenn man auf die Einschränkung $0 \leq x \leq r$ verzichtet, lässt es sich allerdings in polynomialer Zeit lösen **[C96]**.

Die unschöne Situation der $\mathcal{NP}$-Vollständigkeit kann man durch eine gittertheoretische Betrachtungsweise etwas entschärfen:

Es sei $R := \text{kgV}(r_1, r_2, ..., r_k)$ und $c_i := \frac{R}{r_i}, i = 1, 2, ..., k$. Eine Basismatrix B sei nun folgendermaßen definiert:

$$B := \left[\begin{array}{c|cccc} -d_1 & a_{1,1} & a_{1,2} & \dots & a_{1,k} \\ -d_2 & a_{2,1} & a_{2,2} & \dots & a_{2,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -d_h & a_{h,1} & a_{h,2} & \dots & a_{h,k} \\ \hline -R & 2c_1 & 0 & \dots & 0 \\ -R & 0 & 2c_2 & \dots & 0 \\ \vdots & 0 & 0 & \ddots & 0 \\ -R & 0 & 0 & \dots & 2c_k \\ \hline R & 0 & 0 & \dots & 0 \end{array}\right] \in \mathbb{R}^{(h+k+1)\times(k+1)}$$

Es bezeichne L, das von B erzeugte Gitter, d.h. $L = L(B)$. Dann gilt:

THEOREM 58. *$x \in \mathbb{Z}^k$ löst (FP) genau dann, wenn die folgenden Bedingungen erfüllt sind:*

(1) $(\underbrace{0, 0, \dots, 0}_{h-mal}, 2c_1x_1 - R, 2c_2x_2 - R, ..., 2c_kx_k - R, R) \in L$
(2) $-R \leq 2c_ix_i - R \leq R$ $(i = 1, 2, \dots, k)$

Die erste Bedingung garantiert, dass x das Gleichungssystem $Ax = d$ löst, während die zweite Bedingung sicherstellt, das $0 \leq x \leq r$ erfüllt ist.

BEWEIS. Es sei $x \in \mathbb{Z}^k$ eine Lösung von (FP). Es ist:

$$1 \cdot b_1 + x_1 b_2 + x_2 b_3 + ... + x_k b_{k+1} \in L$$

Also:

$$1 \cdot b_1 + x_1 b_2 + x_2 b_3 + ... + x_k b_{k+1}$$

$$= (-d + A \cdot x, 2c_1 \cdot x_1 - R, 2c_2 \cdot x_2 - R, ..., 2 \cdot c_k x_k - R, R)$$

$$= (\underbrace{0, 0, \ldots, 0}_{h-mal}, 2c_1 x_1 - R, 2c_2 x_2 - R, ..., 2c_k x_k - R, R) \in L$$

Des Weiteren gilt:

$$0 \leq x \leq r \Longleftrightarrow 0 \leq x_j \leq r_j \text{ für } j = 1, 2, ..., k$$

$$\Longleftrightarrow 0 \leq 2c_j x_j \leq 2c_j r_j, \text{ da } c_j \geq 0 \text{ für } j = 1, 2, ..., k$$

$$\Longleftrightarrow -R \leq 2c_j x_j - R \leq 2c_j r_j - R = 2R - R = R.$$

Zum Beweis der Rückrichtung: Es sei $y \in L$ ein Gittervektor mit

$$y_1, y_2, ..., y_h = 0, y_{h+k+1} = R$$

$$\text{und } -R \leq y_j \leq R \text{ für } j = h+1, h+2, ..., h+k$$

Da $y_{h+k+1} = R$ und b_1 der einzige Basisvektor ist, dessen letzte Komponente ungleich Null ist, folgt:

$$y = 1 \cdot b_1 + \sum_{j=1}^{k} \lambda_j b_{j+1}$$

Da $y_1, y_2, ..., y_h = 0$ folgt: $-d + A \cdot (\lambda_1, \lambda_2, ..., \lambda_k)^T = 0$. Damit erhält man:

$$1 \cdot b_1 + \sum_{j=1}^{k} \lambda_j b_j = (0, 0, ..., 0, -R + 2c_1\lambda_1, -R + 2c_2\lambda_2, ..., -R + 2c_k\lambda_k, R)^T$$

Schließlich liefert die Bedingung $-R \leq y_j \leq R$ für $j = h+1, h+2, ..., h+k$ die Ungleichung:

$$-R \leq -R + 2c_j\lambda_j \leq R \Longleftrightarrow 0 \leq \lambda_j \leq r_j \text{ für } j = 1, 2, ..., k$$

Damit entspricht der Vektor

$$(\lambda_1, \lambda_2, ..., \lambda_k)^T = \frac{1}{2}(\frac{y_{h+1}}{c_{h+1}} + R, \frac{y_{h+2}}{c_{h+2}} + R, ..., \frac{y_{h+k}}{c_{h+k}} + R)$$

einer Lösung von (FP). □

Durch einen eleganten Ansatz [**A98**], der auf Karen Aardal zurückgeht, ist es möglich die Dimension des Gitters sogar noch zu verringern. Zunächst multipliziert man die ersten h Zeilen der Basismatrix B mit einer genügend großen Konstante $N \in \mathbb{N}$:

$$B' := \left[\begin{array}{c|ccccc} -N \cdot d & & & N \cdot A & & \\ \hline -R & 2c_1 & 0 & 0 & 0 & 0 \\ -R & 0 & 2c_2 & 0 & \ddots & 0 \\ -R & 0 & 0 & 2c_3 & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ -R & 0 & 0 & 0 & 0 & 2c_k \\ \hline N \cdot R & 0 & 0 & 0 & \dots & 0 \end{array}\right] \tag{1.4.1}$$

Wenn N groß genug gewählt wurde und die Matrix A vollen Zeilenrang hat, dann erhält man, nach Anwendung des LLL-Algorithmus auf B', eine Gitterbasis, die aus $k-h+1$ Vektoren besteht, deren ersten k Komponenten allesamt gleich Null sind, und h Vektoren, die wenigstens einen Eintrag $\neq 0$ in den ersten h Komponenten besitzen.

$$\begin{array}{r} \left.\begin{array}{r} \\ h \text{ Zeilen} \\ \\ \\ \end{array}\right\{ \\ \left.\begin{array}{r} \\ k+1 \text{ Zeilen} \\ \\ \\ \end{array}\right\{ \end{array} \left(\begin{array}{ccc|ccc} 0 & \cdots & 0 & * & \cdots & * \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & * & \cdots & * \\ \hline & * & & & * & \\ & & & & & \\ & & & & & \end{array}\right) \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \text{Vielfache} \\ \text{von } N_1 \end{array} \tag{1.4.2}$$

$$\underbrace{\qquad}_{k-h+1 \text{ Spalten}} \quad \underbrace{\qquad}_{h \text{ Spalten}}$$

Letztere Vektoren werden aus der Gitterbasis entfernt und bei den verbleibenden $k-h+1$ Vektoren die ersten h Zeilen gestrichen, die nur Nullen enthalten. Das daraus resultierende Gitter $\subseteq \mathbb{R}^{k+1}$ wird mit L' bezeichnet.

Nun lässt sich folgern:

KOROLLAR 59. $x \in \mathbb{Z}^k$ löst (FP) genau dann, wenn die folgenden Bedingungen erfüllt sind:

(1) $(2c_1x_1 - R, 2c_2x_2 - R, ..., 2c_kx_k - R, R) \in L'$
(2) $-R \leq 2c_ix_i - R \leq R$ $(i = 1, 2, \ldots, k)$

BEWEIS. siehe [**A98, W02**] □

1.4.1. Gleichungssysteme mit Ungleichungen. Es soll nun der etwas allgemeinere Fall betrachtet werden, bei dem Ungleichungen mitbeteiligt sind, d.h. es sind die folgenden zusätzlichen Bedingungen zu erfüllen:

$$l_u \leq A_u x \leq r_u \text{ mit } A_u \in \mathbb{Z}^{u \times k} \text{ und } l_u, r_u \in \mathbb{Z}^u$$

Dieses System läßt sich umformen zu:

$$-(r_u - l_u) \leq \left[\ -(l_u + r_u) \mid 2A_u\ \right] \cdot \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_k \end{pmatrix} \leq (r_u - l_u)$$

$$x_0 = 1$$

Wenn also diese Ungleichungen in dem gittertheoretischen Ansatz ebenfalls mitberücksichtigt werden sollen, müssen die Vektoren der Basismatrix B folgendermaßen erweitert werden:

$$B'' = \left[\begin{array}{c|ccc} -(l_u + r_u) & 2 \cdot A_u & & \\ \hline b_1 & b_2 & \dots & b_{k+1} \end{array}\right] \tag{1.4.3}$$

Es bezeichne L'' das von B'' erzeugte Gitter, dann läßt sich jetzt schließen:

KOROLLAR 60. $x \in \mathbb{Z}^k$ löst (FP) und die zusätzlichen Ungleichungsnebenbedingungen genau dann, wenn die folgenden Bedingungen erfüllt sind:

(1) $((2A_u x - (l_u + r_u))^T, \underbrace{0, 0, \dots, 0}_{h-mal}, 2c_1x_1 - R, ..., 2c_kx_k - R, R) \in L''$
(2) $-(r_u - l_u) \leq 2A_u x - (l_u + r_u) \leq (r_u - l_u)$
(3) $-R \leq 2c_i x_i - R \leq R \ (i = 1, 2, \dots, k)$

BEMERKUNG 61. Falls die reduzierte Basismatrix B' entsprechend erweitert werden soll, muss zunächst sichergestellt werden, dass sie die folgende Form besitzt:

$$B' = \left[\begin{array}{c|ccc} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ \hline R & 0 & \dots & 0 \end{array}\right]$$

(Durch Multiplikation der letzten Zeile mit einer genügend großen Konstante und anschließender LLL-Reduktion, kann diese Form erzielt werden.)

Erweitert man nun die Basismatrix B' wie in 1.4.3, vereinfacht sich die 1. Bedingung des Korollar zu:

$$((2A_u x - (l_u + r_u))^T, 2c_1x_1 - R, ..., 2c_kx_k - R, R) \in L''.$$

Damit wären nun die relevanten Zusammenhänge zwischen ganzzahligen, linearen Gleichungssystemen und Gittern aufgezeigt. Die nächste Fragestellung wird lauten, wie man an Lösungen solcher Systeme gelangt.

In der Praxis zeigt sich, dass ein kürzester Gittervektor in den meisten Fällen die Bedingungen (2) und (3) erfüllt und folglich einer Lösung für das zugrundeliegende Gleichungssystem entspricht. Das Mittel zur Auffindung von Lösungen wird demnach die Suche nach kürzesten Gittervektoren sein.

KAPITEL 2

Gitterbasenreduktion

In diesem Kapitel werden die grundlegenden gittertheoretischen Probleme formuliert und Reduktionsbegriffe sowie die dazugehörigen Algorithmen vorgestellt, mit denen eine Gitterbasis derartig modifiziert werden kann, dass sie besser zur Lösung dieser Probleme geeignet ist.

DEFINITION 62. (Closest Vector Problem - CVP)

Es sei $L \subseteq \mathbb{R}^n$ ein m-dimensionales Gitter und $x \in \mathbb{R}^n$. Das Problem, einen Vektor $b \in L$, mit $||b-x||_p \leq ||b'-x||_p$ für alle $b' \in L$ zu finden, wird *Closest Vector Problem (CVP)* genannt.

Die homogene Version des CVP lautet:

DEFINITION 63. (Shortest Vector Problem - SVP)

Es sei $L \subseteq \mathbb{R}^n$ ein m-dimensionales Gitter. Das Problem, einen Vektor $b \in L$, mit $0 < ||b||_p \leq ||b'||_p$ für alle $b' \in L$ zu finden, wird *Shortest Vector Problem (SVP)* genannt.

BEMERKUNG 64. Ein Lösungsvektor b des SVP wird *kürzester Gittervektor* genannt. Jedes Gitter besitzt mindestens einen kürzesten Gittervektor und höchstens endlich viele.

Es gilt insbesondere: $\lambda_1^{(p)}(L) = ||b||_p$.

Eine weitere interessante Fragestellung behandelt die nächste Definition:

DEFINITION 65. (Shortest Base Problem - SBP)

Es sei $L \subseteq \mathbb{R}^n$ ein m-dimensionales Gitter. Das Problem, eine Basis von L zu bestimmen, so dass für die Basisvektoren $b_1, b_2, ..., b_m$ gilt:

$$\max_{1\leq i\leq m} ||b_i||_p = \min \left\{ \max_{1\leq i\leq m} ||b'_i||_p \;\middle|\; b'_1, b'_2, ..., b'_m \text{ ist eine Basis von } L \right\}$$

wird *Shortest Base Problem (SBP)* genannt.

BEMERKUNG 66. Ajtaj zeigte in **[Aj98]**, dass das SVP unter *randomisierter* Reduktion $\mathcal{NP}$-hart ist. Die $\mathcal{NP}$-Härte unter *deterministischen* Reduktionen ist weiterhin ein offenes Problem. Bereits länger bekannt ist die $\mathcal{NP}$-Härte des CVP und SBP. Genaueres dazu findet sich in **[M99, G99, C00]**.

Ziel der Gitterbasenreduktion ist es, zu einem gegebenen Gitter eine Gitterbasis zu finden, deren Vektoren kurz und möglichst orthogonal zueinander sind, um damit eventuell eine Lösung des SVP zu erhalten.

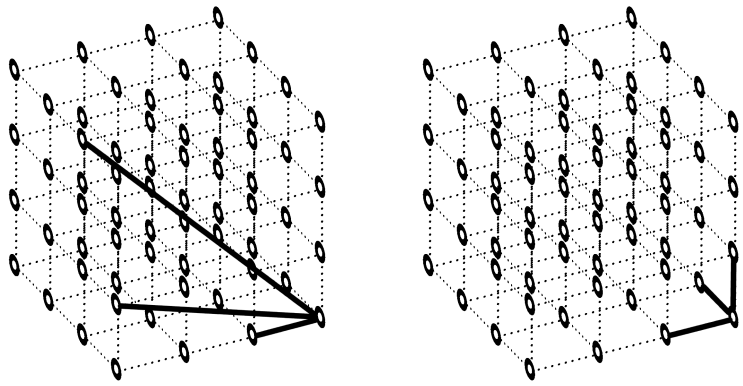


ABBILDUNG 2.0.1. Schlechte und gute Basis des Gitters $\mathbb{Z}^3$

Zur Lösung des CVP bzw. SBP ist bisher leider kein effizientes Verfahren bekannt. Allerdings gibt es Tricks, mit denen sich CVP Probleme in höherdimensionale SVP Probleme einbetten lassen, weshalb man in diesem Fall auch auf die klassische Gitterbasenreduktion zurückgreift [**M98**].

Die Methoden die im weiteren Verlauf dieser Arbeit vorgestellt werden, wurden allesamt mit Hinsicht auf die euklidische Norm entwickelt. Im folgenden bezeichnet daher die Schreibweise $||v||$ die euklidische Länge des Vektors v.

BEMERKUNG 67. Für die Suche nach kürzesten Vektoren in beliebigen p - Normen sei auf die Dissertation von H. Ritter verwiesen [**R97**].

2.1. LLL Reduktion

Die Vorgehensweise der LLL-Reduktion geht auf A. K. Lenstra, H. W. Lenstra und L. Lovász zurück [**L82**] und hat sich zur klassischen Methode der Gitterbasenreduktion entwickelt. Viele spätere Verfahren stellen im wesentlichen eine Erweiterung dieser Technik dar.

Im Unterschied zur globalen Natur der Hermite-Korkine-Zolotareff Reduktion [**K73, S87**], besitzt die LLL-Reduktion eher lokalen Charakter, aufgrund dessen es möglich wird, die Rechenzeit auf Polynomialzeit zu drücken. Der erste Vektor einer LLL-reduzierten Basis ist damit nicht mehr unbedingt ein kürzester Gittervektor, sondern lediglich eine Approximation dessen.

DEFINITION 68. (LLL-reduzierte Basis)

Es sei $L \subseteq \mathbb{R}^n$ ein m-dimensionales Gitter und $b_1, b_2, ..., b_m$ sei die zugehörige Basis. Man nennt die Basis *LLL-reduziert* (mit $\frac{1}{4} < \delta \leq 1$), wenn:

(1) $|\mu_{i,j}| \leq \frac{1}{2}$ für $1 \leq j < i \leq m$
(2) $\delta||\hat{b}_{i-1}||^2 \leq ||\hat{b}_i||^2 + \mu_{i,i-1}^2||\hat{b}_{i-1}||^2$ für $i = 2, 3, ..., m$

BEMERKUNG 69. Der Parameter δ kontrolliert die Güte der reduzierten Basis: Je kleiner δ, um so schwächer ist die Reduktion. Für $\delta < 1$ hat das Reduktionsverfahren polynomielle Laufzeit [**S02**].

Ist die erste Bedingung erfüllt, sagt man auch: Die Basis ist *längenreduziert*.

Die Bedingung (2), die oftmals auch als *Lovász-Bedingung* bezeichnet wird, lässt sich mithilfe der orthogonalen Projektion auch schreiben als:

$$\delta||\pi_{i-1}(b_{i-1})||^2 \leq ||\pi_{i-1}(b_i)||^2 \text{ für } i = 2, 3, ..., m$$

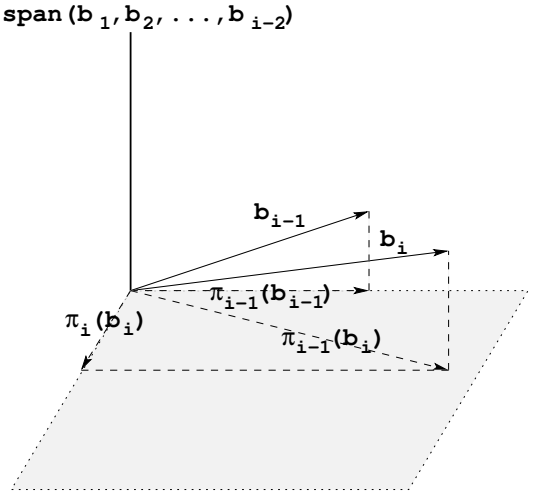


Abbildung 2.1.1. Veranschaulichung der Lovász-Bedingung

Es folgen nun einige Aussagen zu den Eigenschaften LLL-reduzierter Basen und den damit verbundenen Abschätzungen.

Theorem 70. *Es sei $L \subseteq \mathbb{R}^n$ ein m-dimensionales Gitter und $b_1, b_2, ..., b_m$ sei die zugehörige, mit dem Parameter δ LLL-reduzierte, Basis. Dann gelten mit $\alpha = \frac{1}{\delta - \frac{1}{4}}$:*

(1) $||\hat{b}_i||^2 \leq \alpha^{j-i}||\hat{b}_j||^2$ für $1 \leq i \leq j \leq m$
(2) $\alpha^{1-j} \leq \frac{||\hat{b}_j||_2^2}{\lambda_j^2}$ für $j = 1, 2, ..., m$
(3) $\frac{||b_j||^2}{\lambda_j^2} \leq \alpha^{m-1}$ für $j = 1, 2, ..., m$
(4) $||b_k||^2 \leq \alpha^{j-1}||\hat{b}_j||^2$ für $k \leq j$
(5) $||b_1||^2 \leq \alpha^{\frac{m-1}{2}}(\det L)^{\frac{2}{m}}$
(6) $\prod_{i=1}^m ||b_i||^2 \leq \alpha^{\binom{m}{2}}(\det L)^2$

Beweis. (1): Aus den beiden Eigenschaften für LLL-Reduziertheit folgt:

$$\delta||\hat{b}_i||^2 \leq ||\hat{b}_{i+1}||^2 + \mu_{i+1,i}^2||\hat{b}_i||^2 \leq ||\hat{b}_{i+1}||^2 + \frac{1}{4}||\hat{b}_i||^2$$

und somit:

$$(\delta - \frac{1}{4}) \cdot ||\hat{b}_i||^2 \leq ||\hat{b}_{i+1}||^2 \Leftrightarrow ||\hat{b}_i||^2 \leq \alpha||\hat{b}_{i+1}||^2$$

Für $j = i + k$ lässt sich hieraus nach k-maliger Anwendung folgern:

$$||\hat{b}_i||^2 \leq \alpha||\hat{b}_{i+1}||^2 \leq \alpha^2||\hat{b}_{i+2}||^2 \leq ... \leq \alpha^k||\hat{b}_{i+k}||^2 = \alpha^{j-i}||\hat{b}_j||^2$$

(2) + (4): Nach der Definition über sukzessive Minima gibt es ein k mit $1 \leq k \leq j$ und $\lambda_j \leq ||b_k||$. Die Anwendung von (1) liefert:

$$\begin{aligned}\lambda_j^2 &\leq ||b_k||^2 \leq ||\hat{b}_k||^2 + \frac{1}{4}\sum_{i=1}^{k-1}||\hat{b}_i||^2 \\ &\leq ||\hat{b}_j||^2(\alpha^{j-k} + \frac{1}{4}\sum_{i=1}^{k-1}\alpha^{j-i}) = ||\hat{b}_j||^2\alpha^{j-1}(\alpha^{1-k} + \frac{1}{4}\sum_{i=1}^{k-1}\alpha^{1-i})\end{aligned}$$

Es bleibt noch zu zeigen:

$$\alpha^{1-k} + \frac{1}{4}\sum_{i=1}^{k-1}\alpha^{1-i} \leq 1$$

Für $k = 1$ gilt die Ungleichung offenbar.

Für $k \geq 2$ gilt wegen $\alpha^{-1} = \delta - \frac{1}{4} \leq \frac{3}{4}$:

$$\alpha^{1-k} + \frac{1}{4}\underbrace{\sum_{i=1}^{k-1}\alpha^{1-i}}_{\text{geom. Reihe}} \leq (\frac{3}{4})^{k-1} + \frac{1}{4}\cdot\frac{1-(\frac{3}{4})^{k-1}}{1-\frac{3}{4}} = \frac{1}{4}\cdot\frac{1}{1-\frac{3}{4}} = 1$$

Aus der Ungleichungskette $\lambda_j^2 \leq ||b_k||^2 \leq ||\hat{b}_j||^2\alpha^{j-1}$ folgen (2) und (4).

(3): Nach Satz 54 gibt es ein $k \geq j$, so dass $\lambda_j \geq ||\hat{b}_k||$. Unter Verwendung von (1) und (4) ergibt sich:

$$\lambda_j^2 \geq ||\hat{b}_k||^2 \geq \alpha^{-k+j}||\hat{b}_j||^2 \geq \alpha^{-k+1}||b_j||^2 \geq \alpha^{-m+1}||b_j||^2$$

(5): Nach (4) gilt $||b_1||^2 \leq \alpha^{i-1}||\hat{b}_i||^2$, weshalb folgt

$$||b_1||^{2m} \leq \alpha^1 \cdot \alpha^2 \cdot ... \cdot \alpha^{m-1}\prod_{i=1}^{m}||\hat{b}_i||^2 = \alpha^{\binom{m}{2}}(\det\ L)^2$$

und somit

$$||b_1||^2 \leq \alpha^{\frac{m-1}{2}}(\det\ L)^{\frac{2}{m}}$$

(6): Nach (4) gilt $||b_i||^2 \leq \alpha^{i-1}||\hat{b}_i||^2$ und aus $\prod_{i=1}^{m}||\hat{b}_i||^2 = (\det\ L)^2$ folgt:

$$\prod_{i=1}^{m}||b_i||^2 \leq \prod_{i=1}^{m}\alpha^{i-1}||\hat{b}_i||^2 = \alpha^{\binom{m}{2}}(\det\ L)^2$$

□

Die Anzahl der Schritte, die man benötigt, um die Basis $b_1, b_2, ..., b_m$ eines m-dimensionalen Gitters $L \subseteq \mathbb{R}^n$ in eine (mit dem Parameter δ) LLL-reduzierte Basis zu überführen, beträgt:

$$\mathcal{O}(nm^3 \cdot \log_{\frac{1}{\delta}}\ M),$$

$$\text{wobei } M := \max_{i=1,2,...,m}||b_i||^2.$$

Für detailliertere Komplexitätsbetrachtungen siehe **[L83, S04]**.

Der Algorithmus zur Bestimmung einer LLL-reduzierten Basis lautet:

ALGORITHMUS 71. (LLL-Reduktion)

```
SIZE_REDUCE(B = [b_1 b_2 ... b_m], [μ_{i,j}]_{1≤i,j≤m}, k)
{
  for (j = k-1, k-2, ..., 1) {

    if (|μ_{k,j}| > 1/2) {
      b_k ⟵ b_k - ⌊μ_{k,j}⌉ b_j
      for (i = 1, 2, ..., k-1)
          μ_{k,i} ⟵ μ_{k,i} - ⌊μ_{k,j}⌉ μ_{j,i}
    }
  }

  return (B, μ)
}

LLL-REDUCE(B = [b_1 b_2 ... b_m], δ)
{
  k ⟵ 1

  while (k ≤ m) {

    (μ, c)⟵COMPUTE_GS(B, k)
    (B, μ)⟵SIZE_REDUCE(B, μ, k)

    if (k == 1) {
      k ⟵ k + 1
      continue
    }

    if (δ||b̂_{k-1}||^2 > ||b̂_k||^2 + μ^2_{k,k-1}||b̂_{k-1}||^2) {
      VERTAUSCHE(b_{k-1}, b_k)
      k ⟵ k - 1
    } else
      k ⟵ k + 1

  return(B)
}
```

2.2. Tiefeneinfügungen

Es wird nun eine Modifikation am LLL-Verfahren vorgenommen, die von Schnorr und Euchner beschrieben wurde. Anstatt des Lovász-Schritts werden *Tiefeneinfügungen (deep insertions)* durchgeführt. D.h. es werden nicht nur die Projektionen aufeinanderfolgender Basisvektoren b_{k-1} und b_k verglichen, sondern auch davor liegende Basisvektoren einbezogen.

Der LLL-Algorithmus befinde sich auf Stufe k:

Zunächst wird die Index-Variable i mit 1 initialisiert und überprüft, ob $\delta||\pi_i(b_i)||^2 \leq ||\pi_i(b_k)||^2$. Falls die Ungleichung nicht erfüllt ist, wird der Vektor b_k an die Stelle des Vektors b_i gebracht und die nachfolgenden Vektoren um eine Position nach hinten verschoben. Ist die Ungleichung erfüllt, wird i um eins inkrementiert. Solange $i < k$ ist, wird dieses Vorgehen iteriert. Im Falle $i = k$ wird der *Deep-Insertion* Schritt beendet.

Bemerkung 72. Die Laufzeit des derartig modifizierten LLL-Algorithmus ist schlimmstenfalls nur noch super-polynomial. Wenn eine Tiefeneinfügung aber nur dann durchgeführt wird, wenn $i \leq \tau$ oder $k - i \leq \tau$, wobei $\tau \in \mathbb{N}$ eine fest gewählte Konstante ist, dann bleibt die Laufzeit polynomial [**E94**].

Algorithmus 73. (LLL-Reduktion mit Tiefeneinfügungen)

```
DEEP_LLL-REDUCE(B = [b_1 b_2 ... b_m], τ, δ)
{
  k ⟵ 1
  while (k ≤ m) {

    (μ, c) ⟵ COMPUTE_GS(B, k)
    (B, μ) ⟵ SIZE_REDUCE(B, μ, k)

    if (k == 1) {
      k ⟵ k + 1
      continue;
    }

    t ⟵ ||b_k||^2
    i ⟵ 1

    while (i < k) {
      if (δ||b̂_i|| ≤ t || i > τ || k − i > τ) {
        t ⟵ t − μ^2_{k,i}
        i ⟵ i + 1
      } else {
        B ⟵ [b_1 b_2 ... b_{i−1} b_k b_i b_{i+1} ... b_{k−1} b_{k+1} ... b_m]
        k ⟵ i − 1
        break
      }
    }

    k ⟵ k + 1
  }

  return(B)
}
```

2.3. Block Korkine-Zolotareff Reduktion

In diesem Abschnitt wird eine Weiterentwicklung des LLL-Algorithmus vorgestellt, die (auf Kosten der Laufzeit) besser reduzierte Basen liefert. Das Verfahren wurde von C. P. Schnorr entwickelt und kann als das Bindeglied zwischen dem exponentiellen HKZ-Verfahren **[S87]** und der polynomiellen LLL-Methode angesehen werden.

Definition 74. (BKZ-reduzierte Basis)

Es sei $L \subseteq \mathbb{R}^n$ ein m-dimensionales Gitter und $b_1, b_2, ..., b_m$ sei die zugehörige Basis. Man nennt die Basis *BKZ-reduziert zur Blockweite* $2 \leq \beta \leq m$ oder *β-reduziert*, wenn:

(1) $|\mu_{i,j}| \leq \frac{1}{2}$ für $1 \leq j < i \leq m$
(2) $\pi_j(b_j), \pi_j(b_{j+1}), ..., \pi_j(b_{j+\beta-1})$ ist HKZ-reduzierte Basis für $j = 1, 2, ..., m - \beta + 1$

Bemerkung 75. Jede $(\beta+1)$-reduzierte Basis ist auch β-reduziert, und eine m-reduzierte Basis ist HKZ-reduziert. Insbesondere gilt der folgende Satz:

Theorem 76. *Die 2-reduzierten Basen entsprechen genau den LLL-reduzierten Basen mit $\delta = 1$.*

Beweis. Es sei $b_1, b_2, ..., b_m$ eine 2-reduzierte Basis.

Dann gilt für $j = 1, 2, ..., m - 1$:

$$\lambda_1\Big(L\big(\pi_j(b_j), \pi_j(b_{j+1})\big)\Big)^2 = ||\hat{b}_j||^2 \leq \pi_j(b_{j+1})^2 = ||\hat{b}_{j+1}||^2 + \mu_{j+1,j}||\hat{b}_j||^2$$

Mit $\delta = 1$ ist dies grade die zweite Eigenschaft einer LLL-reduzierten Basis:

$$\delta||\hat{b}_j||^2 \leq ||\hat{b}_{j+1}||^2 + \mu_{j+1,j}^2||\hat{b}_j||^2$$

Da jede $\beta-$reduzierte Basis zudem längenreduziert ist, erfüllen $b_1, b_2, ..., b_m$ die LLL-Eigenschaften.

Umgekehrt ist zu zeigen, dass jede mit $\delta = 1$ LLL-reduzierte Basis $b_1, b_2, ..., b_m$ auch 2-reduziert ist. Es muss also für $j = 1, 2, ..., m - 1$ gezeigt werden, dass jeder Vektor $(\neq 0)$ in $L\big(\pi_j(b_j), \pi_j(b_{j+1})\big)$ nicht kürzer ist als $\hat{b}_j = \pi_j(b_j)$. Wegen

$$||u \cdot \pi_j(b_j) + v \cdot \pi_j(b_{j+1})||^2 = (u + v \cdot \mu_{j+1,j})^2||\hat{b}_j||^2 + v^2||\hat{b}_{j+1}||^2$$

ist zu zeigen, dass für alle $(u, v) \in \mathbb{Z}^2 \backslash \{(0, 0)\}$ gilt:

$$(u + v \cdot \mu_{j+1,j})^2||\hat{b}_j||^2 + v^2||\hat{b}_{j+1}||^2 \geq ||\hat{b}_j||^2 \tag{2.3.1}$$

Die Ungleichung (2.3.1) gilt für $v = 0$. Da die Basis mit $\delta = 1$ LLL-reduziert ist, gilt nach dem zweiten LLL-Kriterium:

$$||\hat{b}_j||^2 \leq ||\hat{b}_{j+1}||^2 + \mu_{j+1,j}^2||\hat{b}_j||^2 \tag{2.3.2}$$

und man erhält damit den Fall $v = 1$ von (2.3.1). Da die Basis längenreduziert ist, gilt $\mu_{j+1,j}^2 \leq \frac{1}{4}$ und nach (2.3.2) folgt $\frac{3}{4}||\hat{b}_j||^2 \leq ||\hat{b}_{j+1}||^2$. Damit erhält man nun den Fall $|v| \geq 2$ von (2.3.1):

$$(u + v \cdot \mu_{j+1,j})^2||\hat{b}_j||^2 + v^2||\hat{b}_{j+1}||^2 \geq v^2||\hat{b}_{j+1}||^2 \geq 4||\hat{b}_{j+1}||^2 \geq 3||\hat{b}_j||^2$$

□

Zu BKZ-reduzierten Basen lassen sich in Abhängigkeit von der Blockweite β folgende Aussagen machen. Die Beweise hierzu finden sich in **[S02]**.

THEOREM 77. *Für jede β-reduzierte Basis $b_1, b_2, ..., b_m$ eines Gitters $L \subseteq \mathbb{R}^n$ gelten für $j = 1, 2, ..., m$:*

(1) $\frac{||\hat{b}_j||^2}{\lambda_j(L)} \leq \gamma_\beta^{2 \cdot \frac{m-j}{\beta-1}}$

(2) $\frac{||b_j||^2}{\lambda_j(L)} \leq \gamma_\beta^{2 \cdot \frac{m-j}{\beta-1}} \cdot \frac{j+3}{4}$

(3) $\frac{\lambda_j(L)}{||b_j||^2} \leq \gamma_\beta^{2 \cdot \frac{j-1}{\beta-1}} \cdot \frac{j+3}{4}$

(4) $||b_1|| \leq \gamma_\beta^{\frac{m-1}{\beta-1}} \cdot \lambda_1(L)$

(5) $||b_1|| \leq \gamma_\beta^{\frac{m-1}{\beta-1}} \cdot \max\{\hat{b}_{m-\beta+2}, \hat{b}_{m-\beta+3}, ..., \hat{b}_m\}$

Zur Formulierung eines Reduktionsalgorithmus empfiehlt es sich (ähnlich wie bei der LLL-Reduktion) mit folgender abgeschwächter Definition einer BKZ-reduzierten Basis zu arbeiten:

DEFINITION 78. ((β, δ)-reduzierte Basis)

Es sei $L \subseteq \mathbb{R}^n$ ein m-dimensionales Gitter und $b_1, b_2, ..., b_m$ sei die zugehörige Basis. Man nennt die Basis *BKZ-reduziert zur Blockweite* $2 \leq \beta \leq m$ *mit* $\frac{1}{4} < \delta \leq 1$ oder (β, δ)-*reduziert*, wenn:

(1) $|\mu_{i,j}| \leq \frac{1}{2}$ für $1 \leq j < i \leq m$

(2) $\delta||\hat{b}_j||^2 \leq \lambda_1\Big(L\big(\pi_j(b_j), \pi_j(b_{j+1}), ..., \pi_j(b_{j+\beta-1})\big)\Big)^2$
für $j = 1, 2, ..., m - \beta + 1$

BEMERKUNG 79. Zu (β, δ)-reduzierten Basen lassen sich ähnliche Abschätzungen wie in Satz 77 machen. Näheres hierzu findet sich in **[R97]**.

Der essentielle Bestandteil eines Verfahrens zur BKZ-Reduktion ist die Enumerationsmethode. Wenn $c_s := ||\hat{b}_s||_2^2$ und $k := \min(j + \beta - 1, m)$, dann zählt der folgende Enumerationsalgorithmus alle Vektoren $(u_j, u_{j+1}, ..., u_k) \in \mathbb{Z}^{k-j+1} \backslash \{0\}$ auf, für die gilt:

$$c_j(u_j, u_{j+1}, ..., u_k) := ||\pi_j(\sum_{i=j}^{k} u_i b_i)||^2 = \sum_{s=j}^{k}(\sum_{i=s}^{k} u_i \mu_{i,s}) \cdot c_s < ||\hat{b}_j||_2^2 \ (*)$$

Zurückgegeben wird ein Vektor $\bar{b} = \sum_{i=j}^{k} \bar{u}_i b_i$, wobei $\bar{u}$ ein Vektor aus der Aufzählung ist, für den (*) minimal wird.

ALGORITHMUS 80. (ENUM)

```
ENUM(B = [b_1 b_2 ... b_m], μ, c, j, k)
{
  c̄_j ⟵ c_j,  y_j ⟵ △_j ⟵ 0,  ũ_j ⟵ u_j ⟵ 1,  δ_j ⟵ 1
  s ⟵ t ⟵ j

  for (i = j+1, j+2, ..., k+1) {
    c̃_i ⟵ y_i ⟵ △_i ⟵ ũ_i ⟵ u_i ⟵ 0
    δ_i ⟵ 1
  }

  while (t ≤ k) {
    c̃_t ⟵ c̃_{t+1} + (y_t + ũ_t)^2 c_t

    if (c̃_t < c̄_j) {
      if (t > j) {

        t ⟵ t − 1
        y_t ⟵ Σ_{i=t+1}^{s} ũ_i μ_{i,t}
        ũ_t ⟵ v_t ⟵ ⌊−y_t⌉
        △_t ⟵ 0

        if (ũ_t > −y_t)
          δ_t ⟵ −1
        else
          δ_t ⟵ 1

      } else {
        c̄_j ⟵ c̃_j
        for (i = j, j+1, ..., k)
          u_i ⟵ ũ_i
      }

    } else {
      t ⟵ t + 1
      s ⟵ max(s, t)

      if (t < s)
        △_t ⟵ −△_t
      if (△_t δ_t ≥ 0)
        △_t ⟵ △_t + δ_t

      ũ_t ⟵ v_t + △_t
    }
  }

  b̄ ⟵ Σ_{i=j}^{k} u_i b_i

  return (b̄)
}
```

Bemerkung 81. Der Backtracking-Algorithmus enumeriert mit Tiefensuche beginnend bei $t = k$ alle ganzzahligen Vektoren $(\tilde{u}_t, \tilde{u}_{t+1}, ..., \tilde{u}_k)$ mit $\tilde{c}_t := c_t(\tilde{u}_t, \tilde{u}_{t+1}, ..., \tilde{u}_k) < \bar{c}_j$ (*). Hierbei entspricht $\bar{c}_j$ dem bisher gefundenen Minimalwert der Funktion c_j. Sobald ein Vektor gefunden wird, der Bedingung (*) erfüllt, wird t um 1 erniedrigt und das Spiel beginnt von vorne. Dieses Vorgehen wird solange iteriert bis $t = j$ ist. Ist dies der Fall, wird der Koeffizientenvektor $(\tilde{u}_t, \tilde{u}_{t+1}, ..., \tilde{u}_k)$ als aktuelle Minimalstelle in $(u_j, u_{j+1}, ..., u_k)$ gespeichert und die Variable $\bar{c}_j$ wird auf den neuesten Stand gebracht: $\bar{c}_j = c_j(u_j, u_{j+1}, ..., u_k)$.

Dies wird nun solange fortgesetzt, bis der Algorithmus bei $t = k+1$ angelangt ist - dann wurden alle Vektoren getestet, die unterhalb der Schranke $\bar{c}_j$ liegen und die Enumeration kann abgebrochen werden.

Die Gesamtanzahl N der Knoten, die im Verlauf der Enumeration von $L' := \pi_j(L(b_j, b_{j+1}, ..., b_k))$, benötigt werden, lässt sich wie folgt abschätzen:

Wenn $\mathrm{vol}(B_{k-t+1}(R))$ das Volumen der $(k-t+1)$-dimensionalen, euklidischen Kugel mit Radius $R := ||\hat{b}_j||_2^2$ bezeichnet, so kann die Anzahl der Knoten N_t auf der Höhe von t mit der Gauss'schen Heuristik abgeschätzt werden:

$$N_t = \frac{\mathrm{vol}(B_{k-t+1}(R))}{\mathrm{vol}(\pi_t(L))} = \frac{R^t \cdot \frac{\pi^{\frac{k-t+1}{2}}}{\Gamma(\frac{k-t+1}{2}+1)}}{\prod_{i=t}^{k} ||\hat{b}_i||_2} \tag{2.3.3}$$

Mit der Stirling'schen Formel für die Gammafunktion kann das Volumen der Kugel abgeschätzt werden durch:

$$\mathrm{vol}(B_{k-t+1}(R)) = \frac{R^{k-t+1} \cdot \pi^{\frac{k-t+1}{2}}}{\Gamma(\frac{k-t+1}{2}+1)}$$

$$\Rightarrow \mathrm{vol}(B_{k-t+1}(R)) \approx \frac{R^{k-t+1} \cdot \pi^{\frac{k-t+1}{2}}}{\left(\sqrt{2\pi} \cdot \left(\frac{k-t+1}{2}\right)^{\frac{k-t}{2}} \cdot e^{-(\frac{k-t+1}{2})}\right)}$$

Damit vereinfacht sich (2.3.3) zu:

$$N_t \approx \frac{R^{k-t+1} \cdot \pi^{\frac{k-t}{2}} \cdot e^{\frac{k-t+1}{2}}}{\left(\sqrt{2} \cdot \left(\frac{k-t+1}{2}\right)^{\frac{k-t}{2}}\right) \cdot \prod_{i=t}^{k} ||\hat{b}_i||_2}$$

Für die Gesamtanzahl der Knoten $N = \sum_{t=j}^{k} N_t$ erhält man demgemäß:

$$N \approx \sum_{t=j}^{k} \frac{R^{k-t+1} \cdot \pi^{\frac{k-t}{2}} \cdot e^{\frac{k-t+1}{2}}}{\left(\sqrt{2} \cdot \left(\frac{k-t+1}{2}\right)^{\frac{k-t}{2}}\right) \cdot \prod_{i=t}^{k} ||\hat{b}_i||_2}$$

Der vollständige Algorithmus zur BKZ-Reduktion lautet:

ALGORITHMUS 82. (BKZ-Reduktion)

```
BKZ-REDUCE(B = [b_1 b_2 ... b_m], β, δ)
{
  z ⟵ 0,  j ⟵ 0
  B ⟵ LLL-REDUCE(B, δ)

  while (z < m − 1) {
    j ⟵ j + 1
    k ⟵min(j + β − 1, m)

    if (j = m) {
      j ⟵ 1
      k ⟵ β
    }

    b̄ ⟵ENUM(B, c, j, k)

    h ⟵min(k + 1, m)

    if (δ||b̂_j||^2 > c_j(ū)) {
      BASIS(B, ū, j, k)
      z ⟵ 0
    } else
      z ⟵ z + 1

    [b_1 b_2 ... b_h] ⟵LLL-REDUCE(b_1, b_2, ..., b_h, δ)
  }

  return (B)
}
```

Ein kurze Erklärung zur Vorgehensweise des Algorithmus: Die Variable j durchläuft zyklisch die Zahlen $1, 2, ..., m-1$. Die Variable z zählt die Anzahl der Positionen j, welche die Ungleichung

$$\delta||\hat{b}_j||^2 \leq \lambda_1\Big(L\big(\pi_j(b_j), \pi_j(b_{j+1}), ..., \pi_j(b_k)\big)\Big)^2 \tag{2.3.4}$$

erfüllen. Falls diese Ungleichung für ein j nicht erfüllt ist, so wird der Vektor $\bar{b}$ in die Basis eingefügt, der LLL-Algorithmus aufgerufen und $z = 0$ gesetzt. Der Fall $j = m$ kann übersprungen werden, da (2.3.4) dann trivialerweise erfüllt ist.

Am Ende des Algorithmus ist (aufgrund von $z = m - 1$) die Ungleichung (2.3.4) für alle $j = 1, 2, ..., m$ erfüllt und die resultierende Basis ist längenreduziert.

Es bleibt zu untersuchen, wie genau der Vektor $\bar{b}$ in die Basis eingefügt wird:

Zunächst werden die Vektoren $b_j, b_{j+1}, ..., b_k$ so transformiert, dass für ein $k' \leq k$ gilt:

$$\bar{b} = \sum_{i=j}^{k'} u_i b_i \text{ mit } |u_{k'}| = 1$$

$$\iff b_{k'} = \frac{1}{u_{k'}} \cdot \bar{b} - \frac{1}{u_{k'}} \cdot \sum_{i=j}^{k'-1} u_i b_i$$

Dies zeigt, dass $b_{k'}$ durch $\bar{b}, b_j, b_{j+1}, ..., b_{k'-1}$ darstellbar ist. Also ist nicht nur $b_j, b_{j+1}, ..., b_k$ eine Basis, sondern auch $\bar{b}, b_j, ..., b_{k'-1}, b_{k'+1}, ..., b_k$, weshalb man $\bar{b}$ statt $b_{k'}$ verwenden kann.

ALGORITHMUS 83. (BASIS)

```
BASIS(B = [b_1 b_2 ... b_m], u, j, k)
{
  b̄ ⟵ Σ_{i=j}^{k} u_i b_i

  while (u_k = 0)
    k ⟵ k − 1

  i ⟵ k − 1

  while (|u_k| > 1) {

    while (u_i = 0)
      i ⟵ i − 1

    q ⟵ ⌈u_k / u_i⌋

    t ⟵ u_i
    u_i ⟵ u_k − q · u_i
    u_k ⟵ t

    t ⟵ b_k
    b_k ⟵ q · b_k − b_i
    b_i ⟵ t
  }

  for (i = k, k − 1, ..., j + 1)
    b_i ⟵ b_{i−1}
  b_j ⟵ b̄

  return(B)
}
```

Der Algorithmus ist wohldefiniert, denn es gilt stets:

$$\begin{aligned}\bar{b} &= \sum_{l=j}^{k} u_l b_l = u_i b_i + u_k b_k + \sum_{l=j, l\neq i}^{k-1} u_l b_l \\ &= (u_k - q \cdot u_i) \cdot b_k + u_i(q \cdot b_k + b_i) + \sum_{l=j, l\neq i}^{k-1} u_l b_l\end{aligned}$$

Aus $|u_k| > 1$ folgt: Es existiert ein i mit $j \leq i \leq k-1$ und $u_i \neq 0$, denn $\mathrm{ggT}(u_j, u_{j+1}, ..., u_k) = 1$. Wäre $\mathrm{ggT}(u_j, u_{j+1}, ..., u_k) = \lambda > 1$ und damit $(u_j, u_{j+1}, ..., u_k) = \lambda \cdot (u'_j, u'_{j+1}, ..., u'_k)$ mit $u'_i \in \mathbb{Z}$, dann würde gelten: $||\pi_t(\sum_{i=j}^{k} u'_i b_i)||^2 = \lambda^{-2} \cdot ||\pi_t(\sum_{i=j}^{k} u_i b_i)||^2$ für $t = j, j+1, ..., k$. Also würde auch $(u'_j, u'_{j+1}, ..., u'_k)$ aufgezählt, was aber der Minimalität von $||\pi_j(\bar{b})||^2$ widerspricht.

Bemerkung 84. Der Algorithmus zur Basiserweiterung beruht auf Überlegungen von H. Hörner [**H94**]. Es existieren aber auch andere Varianten, um den enumerierten Vektor in die Gitterbasis einzufügen, beispielsweise durch Anwendung des LLL-Algorithmus.

Bislang konnte eine polynomielle Rechenzeit des BKZ-Algorithmus nicht nachgewiesen werden. In der Praxis verhält er sich jedoch (für moderate Blocklängen β) recht gut [**E94**].

2.4. Abschneiden von Enumerationsbäumen

In der Praxis zeigt sich, dass der zeitaufwändigste Teil der Enumerationsabschnitt ist. Leider kann darauf nicht verzichtet werden, wenn man genügend kurze Gittervektoren finden will. Es gibt allerdings Möglichkeiten zum Abschneiden des Enumerationsbaums [**W02, W06**].

Zunächst einige theoretische Vorüberlegungen:

Theorem 85. *Es seien $b_1, b_2, ..., b_m \in \mathbb{R}^n$ die Basisvektoren des Gitters und $b \in span(b_1, b_2, ..., b_m)$, d.h. $b = \sum_{i=1}^{m} x_i b_i$ mit $x \in \mathbb{R}^m$.*

Dann ist:

$$\pi_k(b) = \pi_k\left(\sum_{i=k}^{m} x_i b_i\right) = \left(\sum_{i=k}^{m} x_i \mu_{i,k}\right) \cdot \hat{b}_k + \pi_{k+1}\left(\sum_{i=k+1}^{m} x_i b_i\right)$$

Beweis. Nach 1.2.1 gilt einerseits: $\pi_k(b) = \sum_{j=k}^{m} \frac{\langle b, \hat{b_j} \rangle}{\langle \hat{b}_j, \hat{b}_j \rangle} \hat{b}_j$

Andererseits ist aber:

$$\pi_k\left(\sum_{i=k}^{m} x_i b_i\right) = \sum_{i=k}^{m} x_i \cdot \pi_k(b_i) = \sum_{i=k}^{m} x_i \cdot \left(\sum_{j=k}^{i} \mu_{i,j} \hat{b}_j\right)$$

Umordnung der Summationsreihenfolge ergibt:

$$\sum_{i=k}^{m} x_i \cdot \left(\sum_{j=k}^{i} \mu_{i,j} \hat{b}_j\right) = \sum_{j=k}^{m} \hat{b}_j \cdot \left(\sum_{s=j}^{m} x_s \cdot \mu_{s,j}\right) = \sum_{j=k}^{m} \hat{b}_j \cdot \frac{\langle b, \hat{b}_j \rangle}{\langle \hat{b}_j, \hat{b}_j \rangle}$$

Also ist:

$$\pi_k(b) = \pi_k(\sum_{i=k}^{m} x_i b_i)$$

Der zweite Teil der Gleichung ergibt sich folgendermaßen:

$$\begin{aligned}\pi_k\left(\sum_{i=k}^{m} x_i b_i\right) &= \sum_{i=k}^{m} x_i \cdot \left(\sum_{j=k}^{i} \mu_{i,j}\hat{b}_j\right) = \sum_{i=k}^{m} x_i \cdot \left(\mu_{i,k}\hat{b}_k + \sum_{j=k+1}^{i} \mu_{i,j}\hat{b}_j\right) \\ &= \left(\sum_{i=k}^{m} x_i\mu_{i,k}\right) \cdot \hat{b}_k + \sum_{i=k}^{m} x_i\left(\sum_{j=k+1}^{i} \mu_{i,j}\hat{b}_j\right)\end{aligned}$$

Da die letzte Summe für $i = k$ verschwindet, ergibt sich unter Anwendung von $(*)$ letztendlich:

$$\begin{aligned}\pi_k\left(\sum_{i=k}^{m} x_i b_i\right) &= \left(\sum_{i=k}^{m} x_i\mu_{i,k}\right) \cdot \hat{b}_k + \sum_{i=k+1}^{m} x_i\left(\sum_{j=k+1}^{i} \mu_{i,j}\hat{b}_j\right) \\ &= \left(\sum_{i=k}^{m} x_i\mu_{i,k}\right) \cdot \hat{b}_k + \pi_{k+1}\left(\sum_{i=k+1}^{m} x_i b_i\right)\end{aligned}$$

□

LEMMA 86. *Für einen beliebigen Vektor* $b = \sum_{i=1}^{m} x_i b_i$ *gilt:*

$$\langle \pi_j(b), \pi_k(b)\rangle = ||\pi_j(b)||_2^2 \text{ für alle } 1 \leq k < j \leq m$$

BEWEIS.

$$\begin{aligned}\langle \pi_j(b), \pi_k(b)\rangle &= \left\langle \sum_{h=j}^{m} \frac{\langle b, \hat{b}_h\rangle}{\langle \hat{b}_h, \hat{b}_h\rangle}\hat{b}_h, \sum_{l=k}^{m} \frac{\langle b, \hat{b}_l\rangle}{\langle \hat{b}_l, \hat{b}_l\rangle}\hat{b}_l \right\rangle \\ &= \sum_{h=j}^{m}\sum_{l=k}^{m} \left\langle \frac{\langle b, \hat{b}_h\rangle}{\langle \hat{b}_h, \hat{b}_h\rangle}\hat{b}_h, \frac{\langle b, \hat{b}_l\rangle}{\langle \hat{b}_l, \hat{b}_l\rangle}\hat{b}_l \right\rangle \\ &= \sum_{h=j}^{m}\sum_{l=k}^{m} \frac{\langle b, \hat{b}_h\rangle}{\langle \hat{b}_h, \hat{b}_h\rangle} \cdot \frac{\langle b, \hat{b}_l\rangle}{\langle \hat{b}_l, \hat{b}_l\rangle} \cdot \langle \hat{b}_h, \hat{b}_l\rangle\end{aligned}$$

Für alle $l = k, k+1, ..., j-1$ ist $\langle \hat{b}_j, \hat{b}_l\rangle = 0$ und somit:

$$\begin{aligned}\langle \pi_j(b), \pi_k(b)\rangle &= \sum_{h=j}^{m}\sum_{l=j}^{m} \frac{\langle b, \hat{b}_h\rangle}{\langle \hat{b}_h, \hat{b}_h\rangle} \cdot \frac{\langle b, \hat{b}_l\rangle}{\langle \hat{b}_l, \hat{b}_l\rangle} \cdot \langle \hat{b}_h, \hat{b}_l\rangle \\ &= \left\langle \sum_{h=j}^{m} \frac{\langle b, \hat{b}_h\rangle}{\langle \hat{b}_h, \hat{b}_h\rangle}\hat{b}_h, \sum_{l=j}^{m} \frac{\langle b, \hat{b}_l\rangle}{\langle \hat{b}_l, \hat{b}_l\rangle}\hat{b}_l \right\rangle \\ &= \langle \pi_j(b), \pi_j(b)\rangle = ||\pi_j(b)||_2^2\end{aligned}$$

□

THEOREM 87. *Es seien $u_t, u_{t+1}, ..., u_m \in \mathbb{Z}$ festgewählt. Wenn $u_1, u_2, ..., u_{t-1} \in \mathbb{Z}$ existieren, so dass für $b = \sum_{i=1}^{m} u_i b_i$ die Ungleichung $||b||_p \leq F_p$ erfüllt ist, dann gilt für alle $y_t, y_{t+1}, ..., y_m \in \mathbb{R}$:*

$$\left| \sum_{i=t}^{m} y_i \cdot ||\pi_i(b)||_2^2 \right| \leq F_p \cdot || \sum_{i=t}^{m} y_i \cdot \pi_i(b)||_q$$

wobei $F_p > 0$, $1 \leq p, q \leq \infty$ und $\frac{1}{p} + \frac{1}{q} = 1$.

BEWEIS. Nach Lemma 86 gilt für alle $k < i$:

$$\langle \pi_k(b), \pi_i(b) \rangle = ||\pi_i(b)||_2^2$$

Wenn $||b||_p = ||\pi_1(b))||_p \leq F_p$ dann folgt mit der Hölder'schen Ungleichung:

$$\begin{aligned} \left| \sum_{i=t}^{m} y_i ||\pi_i(b)||_2^2 \right| &= \left| \sum_{i=t}^{m} y_i \langle \pi_1(b), \pi_i(b) \rangle \right| = \left| \langle \pi_1(b), \sum_{i=t}^{m} y_i \pi_i(b) \rangle \right| \\ &\leq ||\pi_1(b)||_p \cdot || \sum_{i=t}^{m} y_i \pi_i(b)||_q \leq F_p \cdot || \sum_{i=t}^{m} y_i \pi_i(b)||_q \end{aligned}$$

□

Diesen Satz kann man sich auf zwei Arten zunutze machen:

Es bezeichne $\omega^{(t)} := \pi_t(\sum_{i=t}^{m} u_i b_i)$. Dann besitzt $\omega^{(t)}$ unter anderem eine Darstellung der Form $\omega^{(t)} = x \cdot \hat{b}_t + \omega^{(t+1)}$ mit $x \in \mathbb{R}$. Für alle $r \in \mathbb{Z}$ mit $x \cdot r > 0$ sei $\tilde{\omega}^{(t)} := (x + r) \cdot \hat{b}_t + \omega^{(t+1)}$ und $\eta := \frac{x}{x+r}$, dann gilt:

$$\omega^{(t)} = \eta \cdot \tilde{\omega}^{(t)} + (1 - \eta) \cdot \omega^{(t+1)} \text{ und } 0 < \eta < 1 \tag{2.4.1}$$

Wenn nun $\tilde{\omega}^{(t)}$ zu einer Lösung erweitert werden kann, dann wählt man $(y_t, y_{t+1}, ..., y_m) = (\eta, 1 - \eta, 0, ..., 0)$ und erhält unter Anwendung des Satzes:

$$\eta ||\tilde{\omega}^{(t)}||_2^2 + (1 - \eta)||\omega^{(t+1)}||_2^2 \leq F_p \cdot ||\eta \cdot \tilde{\omega}^{(t)} + (1 - \eta) \cdot \omega^{(t+1)}||_q$$

Die Verwendung von (2.4.1) liefert:

$$\begin{aligned} ||\omega^{(t)}||_2^2 &\leq \eta^2 ||\tilde{\omega}^{(t)}||_2^2 + (1 - \eta)^2 ||\omega^{(t+1)}||_2^2 \\ &\leq \eta ||\tilde{\omega}^{(t)}||_2^2 + (1 - \eta)||\omega^{(t+1)}||_2^2 \\ &\leq F_p \cdot ||\eta \cdot \tilde{\omega}^{(t)} + (1 - \eta) \cdot \omega^{(t+1)}||_q \\ &= F_p \cdot ||\omega^{(t)}||_q \end{aligned}$$

D.h. wenn $\tilde{\omega}^{(t)}$ so erweitert werden kann, dass die Ungleichung $||\tilde{b}||_p \leq F_p$ erfüllt ist, dann erfüllt $\omega^{(t)}$ die Ungleichung $||\omega^{(t)}||_2^2 \leq F_p \cdot ||\omega^{(t)}||_q$.

Die zweite Art wie man sich den Satz zunutze machen kann, besteht darin den Vektor $(y_t, y_{t+1}, y_{t+2}, ..., y_m) = (1, -1, 0, ..., 0)$ zu wählen und damit zu

erhalten:

$$\begin{aligned} &\left|||\pi_t(b)||_2^2 - ||\pi_{t+1}(b)||_2^2\right| \leq F_p \cdot |||\pi_t(b) - \pi_{t+1}(b)||_q \\ \iff &\left|\left(\sum_{i=t}^{m} u_i\mu_{i,t}\right)^2 \cdot ||\hat{b}_t||_2^2\right| \leq F_p \cdot ||\left(\sum_{i=t}^{m} u_i\mu_{i,t}\right) \cdot \hat{b}_t||_q \end{aligned}$$

Die Äquivalenz folgt aus Theorem 85.

Eine letzte Umformung ergibt schließlich:

$$\left|\sum_{i=t}^{m} u_i\mu_{i,t}\right| \leq F_p \cdot \frac{||\hat{b}_t||_q}{||\hat{b}_t||_2^2} \tag{2.4.2}$$

Damit ist es möglich zwei Abschneidebedingungen zu formulieren:

Wenn $||\omega^{(t)}||_2^2 > F_p \cdot ||\omega^{(t)}||_q$ oder $|\sum_{i=t}^{m} u_i\mu_{i,t}| > F_p \cdot \frac{||\hat{b}_t||_q}{||\hat{b}_t||_2^2}$, dann gilt für alle $r \in \mathbb{Z}$ mit $x \cdot r > 0$, dass $\tilde{\omega}^{(t)} = (x + r) \cdot \hat{b}_t + \omega^{(t+1)}$ nicht zu $\tilde{b}$ erweitert werden kann, so dass die Ungleichung $||\tilde{b}||_p \leq F_p$ erfüllt ist, d.h. es braucht in Richtung von r nicht weitergesucht werden.

Eine weitere Abschneidebedingung liefert das nächste Lemma:

LEMMA 88. *Damit $||b||_p = ||\sum_{i=1}^{m} u_i b_i||_p \leq F_p$ gewährleistet ist, muss der Koeffizient u_t die folgende Ungleichung erfüllen:*

$$|u_t| \leq F_p \cdot ||b_t^\star||_q$$

BEWEIS. Nach Theorem 46 ist $u_t = \langle b_t^\star, b\rangle$.

Also gilt nach der Hölder'schen Ungleichung:

$$|u_t| = |\langle b, b_t^\star\rangle| \leq ||b||_p \cdot ||b_t^\star||_q$$

Da $||b||_p \leq F_p$ gelten soll, muss dann zwangsläufig gelten:

$$|u_t| \leq ||b||_p \cdot ||b_t^\star||_q \leq F_p \cdot ||b_t^\star||_q$$

□

Wenn $|u_t| > F_p \cdot ||b_t^\star||_q$, dann kann die Enumeration in Richtung des Vorzeichens von u_t abgebrochen werden, da die Ungleichung $||b||_p \leq F_p$ nicht mehr erfüllbar ist .

BEMERKUNG 89. Zwei weitere programmiertechnische Verbesserung wären:

(1) Im Fall $p = \infty$ kann die Enumeration eines Teilbaums abgebrochen werden, wenn ein $1 \leq j \leq n$ existiert, so dass die j-te Komponente von $\pi_t(b)$ ungleich 0 ist, aber die j-ten Komponenten der ersten $t-1$ Basivektoren gleich 0 sind. In diesem Fall kann sich nämlich die j-te Komponente von $\pi_t(b)$ für beliebige Selektionen $u_1, u_2, ..., u_{t-1}$ nicht mehr ändern.

(2) Der Enumerationsbaum ist symmetrisch ist, d.h. mit $v \in L$ liegt auch $-v$ in L. Folglich kann die Hälfte aller Vektoren bei der Aufzählung weggelassen werden, indem man den indexgrößten Nichtnullkoeffizienten $u_t > 0$ wählt.

Die Pruning Methode, welche die eben besprochenen Abschneidetechniken berücksichtigt, ist von folgender Gestalt:

ALGORITHMUS 90. (Pruning Methode)

```
PRUNE(b*_t, b̂_t, b̃_t, c̃_t, ũ_t, y_t, F_∞, F_1, F_2)
{
  if (c̃_t > F_2 · F_2)
    return true

  d ⟵ |ũ_t + y_t|

  if (d > F_2 · ||b̂_t||_2 / ||b̂_t||_2^2)
    return true

  if (c̃_t > F_∞ · ||b̃_t||_1)
    return true

  if (c̃_t > F_1 · ||b̃_t||_∞)
    return true

  if (d > F_∞ · ||b̂_t||_1 / ||b̂_t||_2^2)
    return true

  if (d > F_1 · ||b̂_t||_∞ / ||b̂_t||_2^2)
    return true

  if (|ũ_t| > F_p · ||b*_t||_q)
    return true

  return false
}
```

BEMERKUNG 91. Um unnötige Berechnungen von $b_t^\star, \hat{b}_t$, und $\tilde{b}_t$ zu vermeiden sind die Tests so angeordnet, dass zuerst die weniger aufwändigen Fälle geprüft werden.

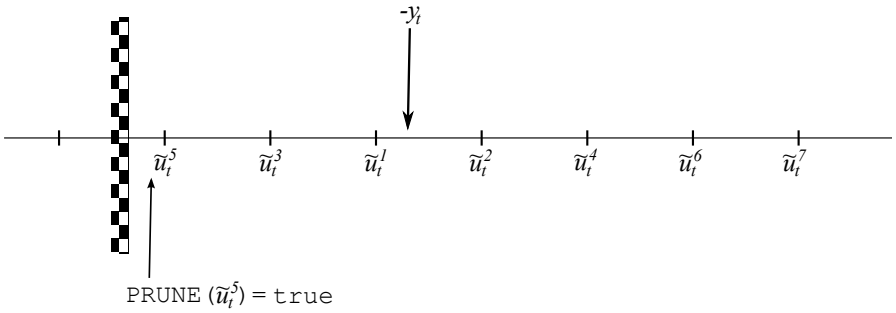
Die Funktion erwartet drei Parameter F_1, F_2, F_∞. Allerdings liefert der verwendete Enumerationsverfahren nur den Wert von F_2. Um gültige Werte für F_1 und F_∞ zu erhalten, setzt man $F_1 := \max\{||v||_1 : ||v||_2 = F_2\}$ und $F_\infty := \max\{||v||_\infty : ||v||_2 = F_2\}$.

Es folgt der modifizierte Enumerationsalgorithmus. Die eingefügten Stellen sind durch rote Schriftfarbe gekennzeichnet:

ALGORITHM. *(Modifizierte Enumeration)*

```
PRUNE_ENUM(...)
{
  c̄_j ⟵ c_j,  y_j ⟵ △_j ⟵ 0,  ũ_j ⟵ u_j ⟵ 1,  δ_j ⟵ 1
  s ⟵ t ⟵ j
  b̃_j ⟵ 0
  p_j ⟵ false

  for (i = j+1, j+2, ..., k+1) {
    c̃_i ⟵ y_i ⟵ △_i ⟵ ũ_i ⟵ u_i ⟵ 0
    δ_i ⟵ 1
    b̃_i ⟵ 0
    p_i ⟵ false
  }

  while (t ≤ k) {
    c̃_t ⟵ c̃_{t+1} + (y_t + ũ_t)^2 c_t
    b̃_t ⟵ b̃_{t+1} + (y_t + ũ_t) · b̂_t

    if (c̃_t < c̄_j) {
      F ⟵ √cbar

      if (PRUNE(b̂_t, b̃_t, c̃_t, ũ_t, y_t, F, F·n, F)) {
        if (p_t == false) {
          △_t ⟵ −△_t
          if (△_t δ_t ≥ 0)
            △_t ⟵ △_t + δ_t
          else
            δ_t ⟵ −δ_t
          ũ_t ⟵ v_t + △_t
          p_t ⟵ true

          continue
        }

        ...

      } else {
          t ⟵ t + 1
          s ⟵ max(s, t)

          if (t < s && p_t == false)
            △_t ⟵ −△_t

          ...
      }
    }

    ...
}
```

Bei genauerer Betrachtung des Enumerationsvorgangs ist zu bemerken, dass auf jeder Stufe t der Enumeration zunächst zwei Zahlen bestimmt werden: $y_t \longleftarrow \sum_{i=t+1}^{s} \tilde{u}_i \mu_{i,t}$ und $v_t \longleftarrow \lfloor -y_t \rceil$. Der Wert von v_t dient zur Initialisierung des Koeffizienten $\tilde{u}_t$ des zu enumerierenden Vektors $b = \sum_{t=j}^{k} \tilde{u}_t b_t$. Wenn der Algorithmus zu einem späteren Zeitpunkt wieder bei Level t ankommt (ausgehend vom Level $t-1$) erhält $\tilde{u}_t$ den zweitnächsten Wert zu $-y_t$, was im Falle von $v_t < -y_t$ der Wert $v_t + 1$ ist. Danach folgen jeweils alternierend die Belegungen: $v_t - 1$,$v_t + 2, v_t - 2, ...$



Die Grafik veranschaulicht zusätzlich den Fall, dass die Enumeration ab der 5. Belegung von $\tilde{u}_t$ abgeschnitten werden kann. In diesem Fall wird dann nur noch in eine Richtung weitergesucht.

2.5. Extreme Pruning

Während die eben vorgestellten Mechanismen den Enumerationsbaum nur leicht stutzen, soll nun eine Variante vorgestellt werden, die nicht so behutsam vorgeht, sondern den Enumerationsbaum wirklich radikal verkürzt.

Die Idee geht zurück auf Nicolas Gama, Phong Q. Nguyen und Oded Regev **[G10]**. Sie verwenden „Bounding Functions“, mit deren Hilfe die zentrale Enumerationsvorschrift entscheidend abgeändert wird, was eine deutliche Verkleinerung des zu durchsuchenden Bereichs zur Folge hat. Zwar verringert sich dabei die Wahrscheinlichkeit einen kürzesten Vektor zu finden, allerdings wird dieser Umstand durch eine signifikante Zeitersparnis beim Abzählvorgang relativiert.

Dieser sei im folgenden noch einmal kurz skizziert:

Beginnend bei $t = k, k-1, ..., j+1, j$ werden sukzessive alle Vektoren $(u_t, u_{t+1}, ..., u_k) \in \mathbb{Z}^{k-t+1} \backslash \{0\}$ herangezogen, für die gilt:

$$c_t(u_t, u_{t+1}, ..., u_k) := ||\pi_t\big(\sum_{i=t}^{k} u_i b_i\big)||^2 = \sum_{s=t}^{k} \big(\sum_{i=s}^{k} u_i \mu_{i,s}\big) \cdot c_s < \bar{c}_j$$

Hierbei wird $\bar{c}_j$ zu Beginn der Enumeration mit der quadrierten, euklidischen Länge des j-ten orthogonalen Basisvektors initialisiert und im Verlauf der Enumeration geupdatet.

Es werden nun drei Begrenzungsfunktionen vorgestellt, die anstelle von $\bar{c}_j$ verwendet werden.

Für l mit $j \leq l \leq k$ und $\beta := k - j + 1$ seien folgende Funktionen definiert:

DEFINITION 92. (Lineare Begrenzungsfunktion)

$$R_1(l) := \left(\frac{k-l+1}{\beta}\right) \cdot \bar{c}_j$$

Diese Funktion bewirkt für große l eine erhebliche Einschränkung des zu enumerierenden Bereichs. Je kleiner l gewählt wird, desto weniger wird abgeschnitten.

Im folgenden sei $\alpha \in \mathbb{R}$ eine Konstante mit $0 < \alpha < 1$.

DEFINITION 93. (Treppen-Begrenzungsfunktion)

$$R_2(l) := \begin{cases} \bar{c}_j, & \text{falls } l \leq \frac{j+k}{2} \\ \alpha \cdot \bar{c}_j, & \text{falls } l > \frac{j+k}{2} \end{cases}$$

Bei dieser Funktion regelt der konstante Faktor α wie nachhaltig der Baum verkürzt wird. Allerdings wird der Wert nur für große l zu Gewichtungszwecken eingesetzt. Für kleine l bleibt die Schranke davon unbeeinflusst. In der Praxis hat sich $\alpha = \frac{1}{2}$ als guter Wert erwiesen **[G10]**.

DEFINITION 94. (Stückweise-Lineare Begrenzungsfunktion)

$$R_3(l) := \begin{cases} \left(2\alpha - 1 + (1-\alpha) \cdot \frac{2\cdot(k-l)+1}{\beta}\right) \cdot \bar{c}_j, & \text{falls } l \leq \frac{j+k}{2} \\ \alpha \cdot \left(\frac{2\cdot(k-l)+1}{\beta}\right) \cdot \bar{c}_j, & \text{falls } l > \frac{j+k}{2} \end{cases}$$

Diese Funktion kombiniert die Wirkungsweisen der beiden zuvor vorgestellten Funktionen. Sie ist stückweise linear und insbesondere in $l = \frac{j+k}{2}$ stetig und für $\alpha = \frac{1}{2}$ ergibt sich wieder die Lineare Begrenzungsfunktion.

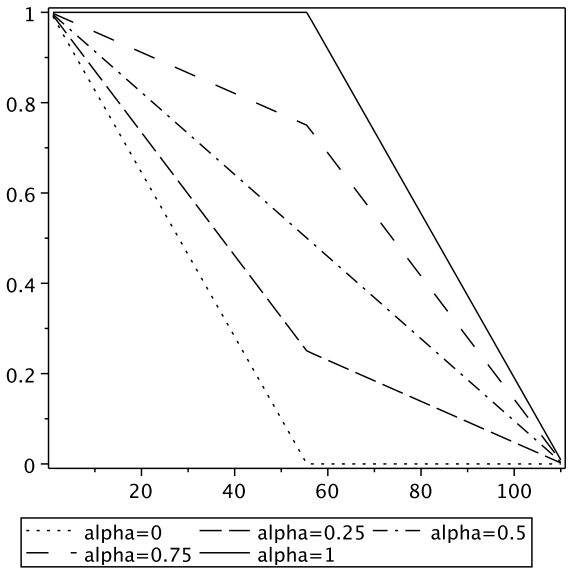


ABBILDUNG 2.5.1. $R_3(l)$ für $1 \leq l \leq 110$, $\alpha \in \{0, 0.25, 0.5, 1\}$

Die modifizierte Enumerationsvorschrift, die Gebrauch von den Begrenzungsfunktionen R_σ $(\sigma = 1, 2, 3)$ macht, lautet:

Suche alle Vektoren $(u_t, u_{t+1}, ..., u_k) \in \mathbb{Z}^{k-t+1} \backslash \{0\}$, für die gilt:

$$c_t(u_t, u_{t+1}, ..., u_k) < R_\sigma(t)$$

Die Gesamtanzahl N der Knoten, die im Verlauf eines derartig modifizierten Enumerationsverfahrens verwendet werden, lässt sich ähnlich wie in Kapitel 2.3 mit der Gauss'schen Heuristik abschätzen. Allerdings besteht diesmal der gravierende Unterschied darin, dass keine $(k - t + 1)$-dimensionale Kugel

$$B_{k-t+1} := \left\{ x \in \mathbb{R}^{k-t+1} \;\middle|\; \sum_{i=1}^{k-t+1} x_i^2 \leq \bar{c}_j \right\}$$

verwendet wird, sondern eine Schnittmenge von Zylindern:

$$C_{k-t+1} := \left\{ x \in \mathbb{R}^{k-t+1} \;\middle|\; \forall s \leq k - t + 1 \; : \; \sum_{i=1}^{s} x_i^2 \leq R_\sigma(s) \right\}$$

Das Volumen von C_{k-t+1} lässt sich durch das folgende Integral ausdrücken:

$$2^{k-t+1} \cdot \int_{x_1=0}^{\sqrt{R_\sigma(1)}} \int_{x_2=0}^{\sqrt{R_\sigma(2)-x_1^2}} ... \int_{x_{k-t+1}=0}^{\sqrt{R_\sigma(k-t+1)-\sum_{i=1}^{k-t} x_i^2}} dx_1 dx_2 ... dx_{k-t+1}$$

Dessen Wert lässt sich allerdings nicht ohne weiteres berechnen. Da aber $C_{k-t+1} \subset B_{k-t+1}$, kann man sich des folgenden, stochastischen Ansatzes bedienen:

$$\mathrm{vol}(C_{k-t+1}) = \mathrm{vol}(B_{k-t+1}) \cdot \Pr_{x \in B_{k-t+1}} \left(\forall s \leq k - t + 1 \; : \; \sum_{i=1}^{z} x_i^2 \leq R_\sigma(s) \right)$$

Der Wahrscheinlichkeitsausdruck lässt sich mittels Monte-Carlo Simulation abschätzen und im Zuge experimenteller Überprüfungen **[G10]** kann dann schließlich gezeigt werden, dass:

$$\left(\sum_{t=j}^{k} \left(\frac{\mathrm{vol}(B_{k-t+1}(R))}{\mathrm{vol}(\pi_t(L))} \right) \right) : \left(\sum_{t=j}^{k} \left(\frac{\mathrm{vol}(C_{k-t+1}(R))}{\mathrm{vol}(\pi_t(L))} \right) \right) \approx 2^{\frac{\beta}{2}}.$$

Im günstigsten Fall ist Extreme Pruning also bis zu $2^{\frac{\beta}{2}}$-mal schneller, als die konventionelle Enumerationsmethode.

KAPITEL 3

Random Sampling

Um kurze Gittervektoren $b = \sum_{j=1}^{m} t_j b_j = \sum_{j=1}^{m} \mu_j \hat{b}_j$ (mit $t_j \in \mathbb{Z}, \mu_j \in \mathbb{R}$) zu erzeugen, verwenden die zuvor beschriebenen Reduktionsmethoden den folgenden Ansatz:

Es ist $||b||_2^2 = \sum_{j=1}^{m} \mu_j^2 ||\hat{b}_j||_2^2$. Um die Länge von b zu verkürzen, versucht man zum einen $|\mu_j|$ zu verkleinern (d.h. den Vektor b so zu verändern, dass $|\mu_j| \leq \frac{1}{2}$) und zum anderen die Längen der orthogonalen Basisvektoren $\hat{b}_j$ zu vermindern. Letzteres erreicht man dadurch, dass man den j-ten Basisvektor b_j durch einen Vektor $b \neq 0$ aus $L(b_j, b_{j+1}, ..., b_m)$ ersetzt, welcher den Ausdruck $||\pi_j(b)||_2^2 = \sum_{i=j}^{m} \mu_i^2 ||\hat{b}_i||_2^2$ über einer passenden Teilmenge $S_j \subseteq L(b_j, b_{j+1}, ..., b_m)$ minimiert. Die Standardverfahren zur Gitterreduktion unterscheiden sich eigentlich nur in der Wahl dieser Teilmenge S_j. LLL-Reduktion verwendet $S_j = L(b_j, b_{j+1})$, BKZ-Reduktion verwendet die Menge $S_j = L(b_j, b_{j+1}, ..., b_{j+\beta-1})$ und HKZ-Reduktion nutzt das gesamte Gitter $S_j = L(b_j, b_{j+1}, ..., b_m)$. Der Vektor $b \neq 0$ aus S_j, für den $||\pi_j(b)||_2^2$ minimal ist, wird durch Anwendung eines Enumerationsverfahrens gewonnen.

Random Sampling **[S03, V05]** verläuft etwas anders:

Es wird ein *zufälliger* Gittervektor $b \in L(b_1, b_2, ..., b_m)$ erzeugt, für den $||\pi_j(b)||_2 < ||\hat{b}_j||_2$ ist. Dieser Vektor wird in die Gitterbasis eingefügt. Anschließend wird die neue Gitterbasis $b_1, b_2, ..., b_{j-1}, b, b_j, b_{j+1}, ..., b_{m-1}$ BKZ-reduziert. Dieses Vorgehen wird nun solange iteriert, bis man einen genügend kurzen Gittervektor gefunden hat.

Die Idee, die hinter der Erzeugung des *zufälligen* Gittervektors steckt, wird zunächst für den Spezialfall $j = 1$ vorgestellt, später wird dann auf die Verallgemeinerung $j \geq 1$ eingegangen:

Man sucht nach kurzen Vektoren $b = \sum_{i=1}^{m} \mu_i \hat{b}_i \in L$ mit kleinen Koeffizienten $\mu_1, \mu_2, ..., \mu_k$ $(k \leq m)$. Damit wird $||b||_2^2 = \sum_{i=1}^{m} \mu_i^2 ||\hat{b}_i||_2^2$ klein. Wichtig ist dabei, dass die Vektoren $\hat{b}_1, \hat{b}_2, ..., \hat{b}_k$ länger sind als die $\hat{b}_i$ mit $i > k$, d.h. kleine Koeffizienten $\mu_1, \mu_2, ..., \mu_k$ haben einen größeren Einfluß auf die Länge von b als kleine μ_i mit $i > k$. Diese Idee wird nun unter der Annahme analysiert, dass die Längen $||\hat{b}_1||_2^2, ||\hat{b}_2||_2^2, ..., ||\hat{b}_m||_2^2$ nahe bei einer geometrischen Folge liegen.

3.1. Die Sampling-Methode

Es sei $1 \leq u < m$ konstant und $b_1, b_2, ..., b_m$ eine Basis des Gitters $L \subseteq \mathbb{R}^n$. Man möchte Gittervektoren $b = \sum_{i=1}^{m} t_i b_i = \sum_{i=1}^{m} \mu_i \hat{b}_i$ erzeugen, für die gilt:

$$|\mu_i| \leq \begin{cases} \frac{1}{2} \text{ für } i < m-u \\ 1 \text{ für } i \geq m-u \end{cases} \tag{3.1.1}$$

$$\text{und } \mu_m = 1 \tag{3.1.2}$$

Der folgende Sampling-Algorithmus (SA) generiert einen solchen Gittervektor in $\mathcal{O}(nm)$ Schritten. Die Wahl von $\mu_m = 1$ garantiert, dass

$$b_1, b_2, ..., b_{j-1}, b, b_j, b_{j+1}, ..., b_{m-1}$$

eine Basis des Gitters bleibt.

ALGORITHMUS 95. (Sampling Algorithmus)

```
SA([b_1 b_2 ... b_m], u)
{
  b ⟵ b_m
  for (j = 1, 2, ..., m − 1)
    μ_j ⟵ μ_{m,j}

  for (i = m − 1, m − 2, ..., 1) {
    Wähle ein μ ∈ ℤ, so daß

      |μ_i − μ| ≤ { 1/2 für i < m − u
                  { 1   für i ≥ m − u

    b ⟵ b − μ · b_i

    for (j = 1, 2, ..., i)
      μ_j ⟵ μ_j − μ · μ_{i,j}
  }

  return (b)
}
```

Bei genauerer Betrachtung des Auswahlkriteriums für μ innerhalb des Algorithmus ist ersichtlich, dass wenigstens 2^u verschiedene Gittervektoren der Form (3.1.1) existieren, denn für $i \geq m - u$ hat man für jedes μ mindestens zwei Wahlmöglichkeiten.

Außerdem ist ersichtlich, dass der Koeffizient μ_i $(m-i)$-mal verändert wird, was dazu führt, dass die μ_i nahezu gleichverteilt sind (insbesondere für kleine i), was entscheidend für das Verfahren sein wird.

Die beiden Annahmen, die man für den späteren Beweis zur Effizienz des Random Sampling benötigt, werden nun formuliert:

Randomness Assumption (RA). Die Koeffizienten μ_i des Vektors $b = \sum_{i=1}^{m} \mu_i \hat{b}_i$ den SA liefert, seien für $i < m-u$ bzw. $m-u \leq i < m$ gleichmäßig in $[-\frac{1}{2}, \frac{1}{2}]$ bzw. $[-1, 1]$ verteilt. Weiter seien die μ_i für verschiedene i und die Koeffizienten μ_i, μ_i' zweier von SA erzeugten Vektoren b, b' statistisch unabhängig.

Geometric Series Assumption (GSA). Es sei $||\hat{b}_i||_2^2/||b_1||_2^2 = q^{i-1}$ $(i = 1, 2, ..., m)$ eine geometrische Folge mit dem Quotienten q, $\frac{3}{4} \leq q < 1$.

Bemerkung 96. In der Praxis werden $||\hat{b}_i||_2^2/||b_1||_2^2$ den Wert q^{i-1} nur approximieren, ohne Gleichheit zu erreichen. Wichtigerweise werden die Schlussfolgerungen, die unter der Annahme GSA getätigt werden, auch gelten, wenn für zufällige (gleichverteilte) $\mu_i \in [-\frac{1}{2}, \frac{1}{2}]$ der Wert

$$\sum_{i=1}^{m} \mu_i^2 (||\hat{b}_i||_2^2/||b_1||_2^2 - q^{i-1})$$

genügend klein (z.B. kleiner als 0.01) ist.

BKZ-reduzierte ($\beta \geq 20$) Gitterbasen haben normalerweise ein recht gutes GSA-Verhalten. Insbesondere ergibt sich das folgende Bild, wenn man die Werte $\log_2(||b_1||_2^2/||\hat{b}_i||_2^2)$ für $i = 1, 2, ..., m$ in ein Koordinatensystem einzeichnet:

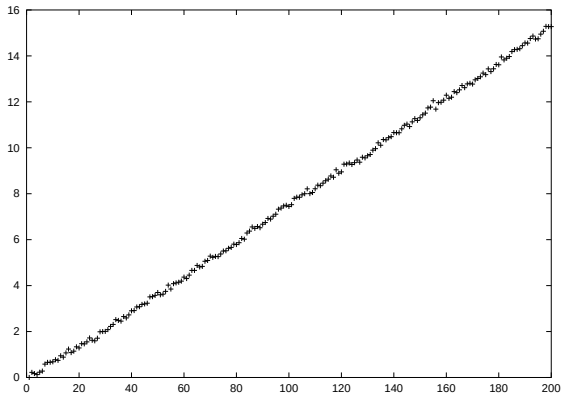


Abbildung 3.1.1. Die Werte $\log_2(||b_1||_2^2/||\hat{b}_i||_2^2)$ für $i = 1, 2, ..., 200$ einer BKZ-Basis mit gutem GSA-Verhalten

Die entscheidende Aufgabe wird nun also sein, q zu bestimmen und den Parameter u, der letztlich maßgebend für den Sampling Algorithmus ist, geeignet zu wählen. Um dies zu bewerkstelligen werden zwei Strategien näher beleuchtet. Zunächst folgen allerdings erst die beiden Algorithmen zur Erzeugung kurzer Gittervektoren.

ALGORITHMUS 97. (Erzeugung kurzer Gittervektoren)

```
SHORT(B = [b_1 b_2 ... b_m], u, k)
{
  for (i = 1, 2, ..., 2^u) {
    b ⟵ SA(B, u)
    if (||b||^2 < 0.99 · ||b_1||^2)
      return (b)
  }

  return (0)
}
```

Der nächste Algorithmus verallgemeinert SHORT und erzeugt für ein vorgegebenes $J \in \mathbb{N}$, Gittervektoren b mit $||\pi_j(b)||_2^2 < 0.99 \cdot ||\hat{b}_j||_2^2$ und $j \leq J$.

ALGORITHMUS 98. (Erzeugung kurzer Projektionen)

```
ESHORT(B = [b_1 b_2 ... b_m], u, J)
{
  Erzeuge via SA mindestens 2^u Gittervektoren
    b = Σ_{i=1}^m μ_i b̂_i .

  Bestimme das Paar (b, j), welches die Ungleichung
    Σ_{i=j}^m μ_i^2 ||b̂_i||^2 < 0.99 · ||b̂_j||^2
  mit kleinstmöglichem j ≤ J erfüllt.

  Falls ein solches Paar existiert:
    return (b, j)

  return (0, 0)
}
```

3.2. Strategie von Schnorr

Es seien $k, u \geq 1$ Konstanten, mit $k + u < m$. Man betrachte den Fall, dass die vom Algorithmus SA erzeugten Vektoren $b = \sum_{i=1}^m \mu_i \hat{b}_i$ folgendes erfüllen:

$$|\mu_i|^2 \leq \frac{1}{4} q^{k-i} \text{ für } i = 1, 2, ..., k \tag{3.2.1}$$

Unter der Annahme RA ist dies mit einer Wahrscheinlichkeit von $\prod_{i=1}^k q^{(k-i)/2} = q^{\binom{k}{2}/2}$ der Fall. Es soll nun die Wahrscheinlichkeit bestimmt werden, dass $||b||_2^2 < ||b_1||_2^2$, wenn RA, (3.1.1), (3.1.2) und (3.2.1) erfüllt sind.

LEMMA 99. *Für zufällige (gleichverteilte)* $\mu_i \in [-\frac{1}{2}, \frac{1}{2}]$ *erhält man den Erwartungswert* $E[\mu_i^2] = \frac{1}{12}$ *und für zufällige (gleichverteilte)* $\lambda_i \in [-1, 1]$ *den Erwartungswert* $E[\lambda_i^2] = \frac{1}{3}$.

Beweis.

$$E[\mu_i^2] = \frac{1}{\frac{1}{2} - (-\frac{1}{2})} \cdot \int_{-\frac{1}{2}}^{\frac{1}{2}} x^2 dx = \frac{1}{3}\left(\frac{1}{2}\right)^3 - \frac{1}{3}\left(-\frac{1}{2}\right)^3 = \frac{1}{12}$$

$$\text{und } E[\lambda_i^2] = \frac{1}{1-(-1)} \cdot \int_{-1}^{1} x^2 dx = \frac{1}{2} \cdot \left(\frac{1}{3}1^3 - \frac{1}{3}(-1)^3\right) = \frac{1}{3}$$

□

Korollar 100. Für zufällig (gleichverteilte) $\nu_i \in [-\frac{1}{2}q^{\frac{k-i}{2}}, \frac{1}{2}q^{\frac{k-i}{2}}]$ erhält man den Erwartungswert $E[\nu_i^2] = \frac{1}{12}q^{k-i}$.

Lemma 101. *Unter den Annahmen RA und GSA erfüllen die vom Algorithmus SA erzeugten Vektoren b mit einer Wahrscheinlichkeit* $\geq \frac{1}{2}q^{\binom{k}{2}/2}$:

$$\frac{||b||_2^2}{||b_1||_2^2} \leq \frac{1}{12}[kq^{k-1} + (q^k + 3q^{m-u-1})/(1-q)]$$

Beweis. Nach (3.2.1), dem Lemma 99 und dem Korollar erhält man:

$$E[\mu_i^2 \,|\, (3.2.1)] = \begin{cases} \frac{1}{12}q^{k-i} & \text{für } i = 1, 2, ..., k \\ 1/12 & \text{für } i = k+1, k+2, ..., m-u-1 \\ 1/3 & \text{für } i = m-u, m-u+1, ..., m-1 \end{cases}$$

Unter der Annahme GSA ergibt dies: $E\left[\frac{||b||_2^2}{||b_1||_2^2} \,\middle|\, (3.2.1)\right] =$

$$\begin{aligned}
&= \frac{1}{12}\left[\sum_{i=1}^{k} q^{k-i}\frac{||\hat{b}_i||_2^2}{||b_1||_2^2} + \sum_{i=k+1}^{m-u-1}\frac{||\hat{b}_i||_2^2}{||b_1||_2^2} + 4\sum_{i=m-u}^{m-1}\frac{||\hat{b}_i||_2^2}{||b_1||_2^2}\right] + \frac{||\hat{b}_m||_2^2}{||b_1||_2^2} \\
&= \frac{1}{12}\left[\sum_{i=1}^{k} q^{k-i+i-1} + \sum_{i=k+1}^{m-u-1} q^{i-1} + 4q^{m-u-1}\sum_{i=1}^{u} q^{i-1}\right] + q^{m-1} \\
&= \frac{1}{12}\left[kq^{k-1} + \left[(q^k - q^{m-u-1}) + 4q^{m-u-1}(1-q^u)\right]/(1-q)\right] + q^{m-1} \\
&= \frac{1}{12}\left[kq^{k-1} + (q^k + 3q^{m-u-1} - 4q^{m-1})/(1-q)\right] + q^{m-1}
\end{aligned}$$

Da $-\frac{4q^{m-1}}{12(1-q)} + q^{m-1} \leq 0$ (wegen $q \geq \frac{3}{4}$) folgt:

$$E\left[\frac{||b||_2^2}{||b_1||_2^2} \,\middle|\, (3.2.1)\right] \leq \frac{1}{12}[kq^{k-1} + (q^k + 3q^{m-u-1})/(1-q)]$$

Weil (3.2.1) mit einer Wahrscheinlichkeit von $q^{\binom{k}{2}/2}$ erfüllt ist und

$\Pr\left[\frac{||b||_2^2}{||b_1||_2^2} \leq E\left[\frac{||b||_2^2}{||b_1||_2^2}\right]\right] \geq \frac{1}{2}$, ergibt sich schließlich:

$$\Pr\left[\frac{||b||_2^2}{||b_1||_2^2} \leq \frac{1}{12}[kq^{k-1} + (q^k + 3q^{m-u-1})/(1-q)]\right] \geq \frac{1}{2}q^{\binom{k}{2}/2}$$

□

THEOREM 102. *Es sei $b_1, b_2, ..., b_m$ die Basis eines Gitters $L \subseteq \mathbb{R}^n$ und GSA sei mit $\frac{3}{4} \leq q < 1$ erfüllt. Wenn k und m genügend groß sind, dann findet SHORT in $\mathcal{O}(nm \cdot q^{-k^2/4})$ Schritten mit einer Wahrscheinlichkeit $> \frac{1}{2}$ einen Vektor $b \in L$ mit $||b||_2^2 < 0.99 \cdot ||b_1||_2^2$.*

BEWEIS. Die Ungleichung

$$\frac{1}{12}[kq^{k-1} + (q^k + 3q^{m-u-1})/(1-q)] \leq \frac{1}{12}[kq^{k-1} + (q^k + 3q^{3k})/(1-q)]$$

ist für alle k, m mit $m \geq 3k + u + 1$ erfüllt.

Da $\phi_q(k) := \frac{1}{12}[kq^{k-1} + (q^k + 3q^{3k})/(1-q)]$ für $k \to \infty$ gegen 0 geht (weil $0 < q < 1$ ist), existiert ein $K_q \in \mathbb{N}$ mit: $\phi(K_q) < 0.99$.

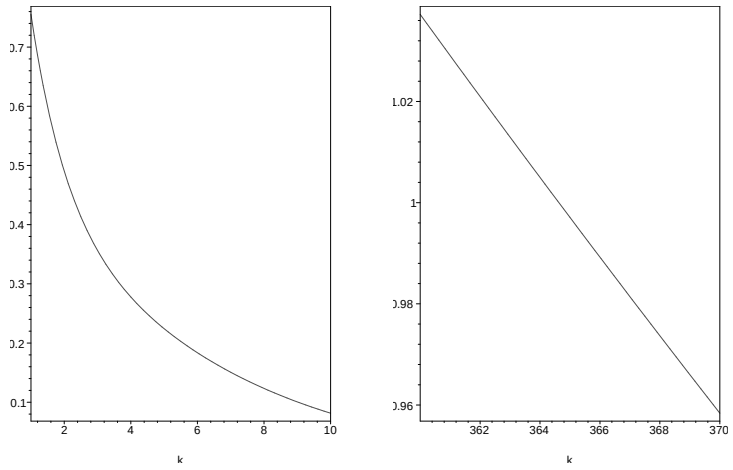


ABBILDUNG 3.2.1. Veranschaulichung von $\phi_{0.75}(k)$ und $\phi_{0.99}(k)$

Man hat damit gezeigt: Wenn k und m genügend groß sind, dann ist $\frac{1}{12}[kq^{k-1} + (q^k + 3q^{m-u-1})/(1-q)] < 0.99$.

Nach Lemma 101 folgt, für die von SA erzeugten Vektoren b:

$$\Pr\left[\frac{||b||_2^2}{||b_1||_2^2} < 0.99\right] \geq \frac{1}{2}q^{\binom{k}{2}/2}$$

Um zu garantieren, dass SHORT mit mindestens 50%-iger Wahrscheinlichkeit ein b findet, so dass $||b||_2^2 < 0.99 \cdot ||b_1||_2^2$ ist, müssen mindestens $2q^{-\binom{k}{2}/2}$ Vektoren erzeugt werden.

Dazu eine kurze Erklärung:

Ein Ereignis E habe die Wahrscheinlichkeit $\Pr[E] = p$ $(0 < p < 1)$. Die Wahrscheinlichkeit, dass in v (voneinander unabhängigen) Versuchen, das Ereignis E mindestens einmal eintritt, beträgt: $1 - (1-p)^v$. Bei p^{-1}-maliger Wiederholung ergibt sich damit die Wahrscheinlichkeit: $1 - (1-p)^{p^{-1}}$.

Da die Ungleichung $1 < e^{-p} + p$ für alle $p > 0$ erfüllt ist ($e^{-p} + p$ wächst streng monoton für $p > 0$ und $e^{-0} + 0 = 1$) folgt schließlich:

$$1 < e^{-p} + p \Leftrightarrow (1-p) < e^{-p} \Leftrightarrow (1-p)^{p^{-1}} < e^{-1}$$

$$\Leftrightarrow 1 - (1-p)^{p^{-1}} > 1 - e^{-1}$$

Weil $1 - e^{-1} > \frac{1}{2}$ muss SA wenigstens $\left(\frac{1}{2}q^{\binom{k}{2}/2}\right)^{-1}$ -mal aufgerufen werden. □

Aus Satz 102 kann eine Vorschrift zur Bestimmung von u hergeleitet werden, wenn der GSA-Quotient q bekannt ist:

(1) Definiere $u(k) := 1 + \left\lceil -\frac{1}{2} \cdot \binom{k}{2} \cdot \log_2(q) \right\rceil$.
(2) Wähle ein minimales k für das gilt:

$$3k + u(k) + 1 \leq m \text{ und } \phi_q(k) < 0.99$$

(3) Falls so ein k existiert, setzt man $u = u(k)$ und ist fertig.
(4) Falls nicht, wählt man ein k für das gilt:

$$3k + u(k) + 1 \leq m \text{ und } \phi_q(k) \text{ ist minimal.}$$

(5) Falls $\phi_q(k) < 2$, setzt man $u = u(k)$ und ist ebenfalls fertig.
(6) Falls $\phi_q(k) \geq 2$, setze $u = 2$.

Bemerkung 103. Die Aussagen von Satz 102 gelten eigentlich nur, wenn Schritt 3 erfüllt ist. Da es aber nicht immer möglich ist, ein solches k zu finden (wenn die Dimension m des Gitters zu gering ist), wird die Forderung $\phi_q(k) < 0.99$ in Schritt 4 zu $\phi_q(k) < 2$ relaxiert. Falls sich wiederum kein passendes k finden lässt, so setzt man letztendlich $u = 2$. Auf diese Weise erhält man ein u, dass die Laufzeit und Effizienz der Algorithmen nicht allzu stark verschlechtert.

Es stellt sich nun noch die Frage, wie der GSA-Quotient q bestimmt wird. Da GSA in der Regel nur näherungsweise erfüllt ist, setzt man zuerst $q_i := (||\hat{b}_i||^2/||b_1||^2)^{\frac{1}{i-1}}$ für $i = 2, 3, ..., m$ und definiert q schließlich als den Median dieser q_i.

Bemerkung 104. Ich habe mich für den Median und nicht den Mittelwert entschieden, da praktische Experimente gezeigt haben, dass insbesondere für kleine i öfters mal ein paar „Ausreißer“ unter den q_i dabei sind, die den Mittelwert zu stark beeinflussen würden. Ein weiterer Vorteil des Medians ist, dass gilt:

$$\Pr\left[\left(\frac{||\hat{b}_i||_2^2}{||b_1||_2^2}\right)^{\frac{1}{i-1}} \leq q\right] \geq \frac{1}{2} \Longrightarrow \Pr\left[\frac{||\hat{b}_i||_2^2}{||b_1||_2^2} \leq q^{i-1}\right] \geq \frac{1}{2}$$

D.h. alle Aussagen, die man unter GSA getroffen hat, gelten mit halbierter Wahrscheinlichkeit auch für den Median.

3.3. Buchmann / Ludwig Strategie

Der von Johannes Buchmann und Christoph Ludwig entwickelte Ansatz **[L05]**, verwendet einen Algorithmus BEST_BOUND, um für ein vorgegebenes u eine untere Grenze für die Wahrscheinlichkeit, dass die Prozedur SHORT (bzw. ESHORT) einen Vektor b liefert, der kürzer ist als $\sqrt{\delta}||b_1||_2$ (bzw. $||\pi_j(b)||_2 < \sqrt{\delta}||\hat{b}_j||_2$), zu berechnen. Dieser Algorithmus ist vollkommen unabhängig von der Geometric Series Assumption (GSA) und liefert ein verallgemeinertes Random Sampling Verfahren.

Ihre Idee basiert auf folgendem Wahrscheinlichkeitsmodell:

Es sei $(S_{q,k})$ ein zufälliger Vorgang, der parametrisiert ist durch $q \in [0,1]$ und $1 \leq k \leq m - u_{max} < m$. Die Wahrscheinlichkeit für das Eintreten von $(S_{q,k})$ wächst streng monoton in q. Es sei $0 \leq q_\delta \leq 1$ der maximale Parameter, für den $E\left[||b||^2 \,|\, (S_{q,k})\right] \leq \delta||b_1||^2$ noch erfüllt ist.

Dann gilt für die Erfolgswahrscheinlichkeit:

$$\Pr\left[||b||_2^2 \leq \delta||b_1||_2^2\right] \geq \Pr\left[||b||^2 \leq E\left[||b||^2 \,|\, (S_{q,k})\right] \,|\, (S_{q,k})\right] \cdot \Pr\left[(S_{q,k})\right]$$

Also:

$$\Pr\left[||b||_2^2 \leq \delta||b_1||_2^2\right] \geq \frac{1}{2}\Pr\left[(S_{q,k})\right]$$

BEST_BOUND berechnet $k_{best} := \max_{k=1,2,...,m-u_{max}} \left\{\left\lfloor \log_2(\frac{1}{2}\Pr\left[(S_{q,k})\right]\right\rfloor\right\}$.

Die Konsequenz ist: Wenn durch Random Sampling wenigstens $2^{-k_{best}}$ Gittervektoren erzeugt werden, dann liegt die Wahrscheinlichkeit einen genügend kurzen Gittervektor zu finden mindestens bei $\frac{1}{2}$.

Das Ereignis $(S_{q,k})$ wird nun etwas präzisiert:

Es sei $b = \sum_{i=1}^{m} \lambda_i \hat{b}_i$ ein Gittervektor und $\sigma : \{1,2,...,m\} \to \{1,2,...,m\}$ eine Permutation, so dass:

$$||\hat{b}_{\sigma(1)}||^2 \geq ||\hat{b}_{\sigma(2)}||^2 \geq ... \geq ||\hat{b}_{\sigma(m-u_{max-1})}||^2$$
$$\text{und } \sigma(i) = i \text{ für } i \geq m - u_{max} \tag{3.3.1}$$

Es seien $q \in [0,1]$ und $1 \leq k \leq m - u_{max}$. Dann bezeichnet $(S_{q,k})$ das Ereignis:

$$\lambda_{\sigma(i)}^2 \leq \frac{1}{4}q^{k-i} \text{ für } i = 1,2,...,k-1$$

Es folgt unter der Annahme RA (Randomness Assumption):

$$\Pr\left[(S_{q,k})\right] = \prod_{i=1}^{k-1} \Pr\left[|\lambda_{\sigma(i)}| \leq \frac{1}{2}q^{\frac{k-i}{2}}\right] = q^{\frac{k(k-1)}{4}}$$

und unter Annahme, dass $(S_{q,k})$ eingetreten ist, erhält man:

$$E(\hat{B},k,q) := E\left[||b||_2^2 \,|\, (S_{q,k})\right] = \sum_{i=1}^{m} E\left[\lambda_i^2 \,|\, (S_{q,k})\right] \cdot ||\hat{b}_i||_2^2$$

$$= \sum_{i=1}^{k-1} q^{k-i}\frac{||\hat{b}_{\sigma(i)}||_2^2}{12} + \sum_{i=k}^{m-u_{max}-1} \frac{||\hat{b}_{\sigma(i)}||_2^2}{12} + \sum_{i=m-u_{max}}^{m-1} \frac{||\hat{b}_i||_2^2}{3} + ||\hat{b}_m||_2^2$$

Die erwartete Länge $E(\hat{B}, k, q)$ ist eine Polynomfunktion bzgl. q mit lauter nicht-negativen Koeffizienten. Somit ist die Abbildung $f : [0,1] \to \mathbb{R}$, $q \mapsto E(\hat{B}, k, q) - \delta||b_1||^2$ eine stetige, streng monoton wachsende Funktion, die genau dann eine Nullstelle besitzt, wenn $f(0) \leq 0 \leq f(1)$. Die eindeutige Nullstelle q_δ kann mithilfe eines Standard-Algorithmus (z.B. Regula Falsi **[P92]**) berechnet werden.

Wenn $f(1) < 0$ dann ist der (bedingungslose) Erwartungswert $E\left[||b||_2^2\right]$ bereits klein genug und es folgt: $\Pr\left[||b||_2^2 \leq \delta||b_1||_2^2\right] \geq \frac{1}{2}$. Andererseits, also wenn $f(0) > 0$ dann liefert die beschriebene Vorgehensweise für den gewählten Parameter k keine positive untere Grenze für $\Pr\left[||b||_2^2 \leq \delta||b_1||_2^2\right]$.

Der Algorithmus BEST_BOUND bestimmt zu vorgegebenem u_{max} die Anzahl der Vektoren, die von SA generiert werden muss, damit die Wahrscheinlichkeit einen passenden Gittervektor zu finden $> \frac{1}{2}$ ist. Um einen geeigneten Wert für u_{max} zu erhalten, wird BEST_BOUND für alle möglichen Werte von $u = 1, 2, ..., m$ aufgerufen. u_{max} wird dann definiert als derjenige Index u, bei dem die Anzahl der zu generierenden Vektoren minimal wird.

Bemerkung 105. Letztendlich ist die Berechnung von $\Pr[(S_{q,k})]$ für alle $1 \leq k \leq m - u_{max}$ der aufwendigste Schritt, insbesondere wegen der Nullstellenberechnung.

Es folgen die entsprechenden Algorithmen:

Algorithmus 106. (Nullstellenberechnung)

```
REGULA_FALSI(f, a, b)
{
  x' ⟵ b

  while (1) {
    x ⟵ a − (f(a)·(b−a)) / (f(b)−f(a))
    if (|x − x'| < ε · |x|)
      return x

    x' ⟵ x
    if (f(a) · f(x) > 0)
      a ⟵ x
    else
      b ⟵ x
  }

  return x
}
```

ALGORITHMUS 107. (Buchmann / Ludwig - Algorithmus)

```
EXP_LENGTH(c, k, u, q)
{
  return ( 1/12 Σ_{i=1}^{k-1} q^{k-i} c_i + 1/12 Σ_{i=k}^{m-u-1} c_i + 1/3 Σ_{i=m-u}^{m-1} c_i + c_m)
}

LOG_SUCC_BOUND(c, k, u, δ)
{
  if (EXP_LENGTH(c, k, u, 1) ≤ δc_1)
    return −1
  else if (EXP_LENGTH(c, k, u, 0) ≥ δc_1)
    return −∞

  q_δ ⟵ REGULA_FALSI( EXP_LENGTH(c, k, u, q) − δc_1, 0, 1)

  return ⌊ k(k−1)/4 log_2(q_δ) − 1 ⌋
}

BEST_BOUND(B̂, u_max, δ)
{
  c ⟵ (||b̂_{σ(1)}||^2, ||b̂_{σ(2)}||^2, ..., ||b̂_{σ(m)}||^2)  /* σ wie in 3.3.1 */

  return max_{k=1,2,...,m−u_max} { LOG_SUCC_BOUND(c, k, u_max, δ) }
}
```

Dieses Teilkapitel endet mit dem eigentlichen Random Sampling-Algorithmus:

ALGORITHMUS 108. (Random Sampling)

```
RSR(B = [b_1 b_2 ... b_m], J)
{
  Bestimme q, k und u mittels einer der
  zuvor angegebenen Strategien.

  (b, j) ⟵ ESHORT(B, u, J)

  if (j ≠ 0)
    B ⟵ BKZ-REDUCE([b_1 b_2 ... b_{j−1} b b_j b_{j+1} ... b_{m−1}], 20, 0.99)

  return (B)
}
```

KAPITEL 4

Heuristische Erweiterungen

4.1. Bewertung von Gittervektoren

Die bisher vorgestellten Verfahren zu Gitterbasenreduktion wurden dahingehend entwickelt, dass sie einen kürzesten Vektor für das SVP finden. In vielen praktischen Anwendungen ist dieser SVP-Lösungsvektor dann auch tatsächlich eine geeignete Lösung für das ursprüngliche Gleichungssystem.

Es gibt nun leider aber Situationen, wo ein SVP-Vektor nicht unbedingt das zugrundeliegende Problem löst. Beispielweise weil ein kürzester Vektor bezüglich einer anderen, als der euklidischen Norm gesucht ist. In diesem Abschnitt wird eine Heuristik definiert, die diesen Umstand berücksichtigt und es ermöglicht einen Gittervektor hinsichtlich seiner Tauglichkeit als Lösungsvektor zu bewerten. Für diese heuristische Bewertung werden allerdings drei Vergleichs-Vektoren benötigt:

Es seien $l, u \in \mathbb{R}^n$ Vektoren mit unteren und oberen Grenzen für einen Lösungsvektor b, d.h. die folgende Ungleichungen müssen erfüllt sein:

$$l_j \leq b_j \leq u_j \, \forall \, j = 1, 2, \ldots n$$

Des Weiteren bezeichne $o \in \{-1, 0\}^n$ den Vektor, der komponentenweise angibt, ob 0 ein gültiger Wert für die j-te Komponente eines Lösungsvektors b ist.

$$\text{Das heißt: } o_j = \begin{cases} 0, & b_j = 0 \text{ ist erlaubt} \\ -1, & b_j = 0 \text{ ist verboten} \end{cases}$$

Man definiert nun eine Abweichungsfunktion:

Definition 109. (Abweichungsfunktion)

$$a_{l,u,o} : \mathbb{R} \to \mathbb{R},$$

$$x \mapsto \begin{cases} 0, & \text{falls } l \leq x \leq u \text{ und } x \neq 0 \\ 0, & \text{falls } l \leq x \leq u \text{ und } x = 0 \text{ und } o = 0 \\ 1, & \text{falls } l \leq x \leq u \text{ und } x = 0 \text{ und } o = -1 \\ x - u, & \text{falls } x > u \\ l - x, & \text{falls } x < l \end{cases}$$

und vier heuristische Bewertungsfunktionen:

Die Default-Heuristik wird noch keinen Gebrauch von der Abweichungsfunktion machen, sondern lediglich den (quadrierten) euklidischen Abstand des übergebenen Vektors y zum Mittelpunkt der Grenzen l und u berechnen.

DEFINITION 110. (Heuristische Bewertungsfunktion 0)

$$H_0 : \mathbb{R}^n \to \mathbb{R},$$
$$H_0(y) := ||\frac{1}{2} \cdot (u + l) - y||_2^2$$

Die nächsten drei Bewertungsprozeduren sind etwas feiner gestrickt:

DEFINITION 111. (Heuristische Bewertungsfunktion 1)

$$H_1 : \mathbb{R}^n \to \mathbb{R},$$
$$H_1(y) := \sum_{j=1}^{n} \min\big(1, a_{l_j,u_j,o_j}(y_j)\big)$$

sowie:

DEFINITION 112. (Heuristische Bewertungsfunktion 2)

$$H_2 : \mathbb{R}^n \to \mathbb{R},$$
$$H_2(y) := \max_{j=1,2,\ldots,n} a_{l_j,u_j,o_j}(y_j)$$

und schließlich:

DEFINITION 113. (Heuristische Bewertungsfunktion 3)

$$H_3 : \mathbb{R}^n \to \mathbb{R},$$
$$H_3(y) := \sum_{j=1}^{n} a_{l_j,u_j,o_j}(y_j)$$

BEMERKUNG 114. Während H_1 die Anzahl der Komponenten zählt, die den gewünschten Anforderungen nicht genügen, ermittelt H_2 die Bewertung aus der Vektorkomponente, die von den Grenzbedingungen am stärksten abweicht. H_3 hingegegen gewichtet jede Grenzverletzung mit der Größe der Abweichung und kombiniert damit H_1 und H_2.

Dieses Bewertungsmodell ist grundlegend und in dieser Form bisher noch nicht in der Literatur erwähnt worden. Es werden sogleich zwei Algorithmen vorgestellt, die davon Gebrauch machen werden.

4.2. Paarweise Reduktion

Zunächst soll ein einfaches Verfahren entwickelt werden, das es ermöglicht, die heuristische Bewertung der Basisvektoren eines Gitters zu verringern.

Es werden zwei beliebige Basisvektoren b' und b'' gewählt und die folgenden vier Kombinationen gebildet:

$$v_1 := -b' - b'',\ v_2 := -b' + b'',\ v_3 := b' - b'' \text{ und } v_4 := b' + b''$$

Sodann wird überprüft, ob $H(v_i) < H(b')$ bzw. $H(v_i) < H(b'')$ für ein $i = 1, 2, 3, 4$. Ist dies der Fall, wird b' bzw. b'' durch den Vektor v_i ersetzt. Dies wird solange fortgesetzt bis sich keine Verbesserung mehr erzielen lässt.

Der Reduktions-Algorithmus lässt sich demgemäß wie folgt notieren:

ALGORITHMUS 115. (Paarweise Reduktion)

```
PAIR_REDUCE (B = [b_1 b_2 ... b_m])
{
  bool retry

  while (retry) {

    retry ⟵ false

    for (h = 1, 2, ..., m) {
      for (i = 1, 2, ..., m) {

        if (i == h)
          continue;

        for (l = −1, 1) {
          b_0 = b_h + l · b_i

          e_0 = H(b_0)
          e_h = H(b_h)
          e_i = H(b_i)

          if (e_0 < e_h)
            i_o = h
          else if (e_0 < e_i)
            i_0 = i
          else {

            e_0 = H(−b_0)

            if (e_0 < e_h)
              i_0 = −h
            else if (e_0 < e_i)
              i_0 = −i
            else
              continue;
          }

          if (i_0 < 0) {
            i_0 = −i_0
            b_0 = −b_0
          }

          b_{i_0} = b_o

          retry ⟵ true
        }
      }
    }
  }
}
```

Es ist möglich diesen Ansatz noch weiter zu verallgemeinern, indem mehr als zwei Vektoren miteinander kombiniert werden. Dies hat allerdings einen exponentiellen Anstieg der Laufzeit zur Folge.

Man kann diese Laufzeiterhöhung durch ein etwas anders geartetes Verfahren kompensieren, wodurch sich jedoch der Speicherplatzbedarf vergrößert.

4.3. Heuristisches Siebverfahren

Im Jahre 2001 wurde der mathematischen Welt ein Algorithmus von Ajtaj, Kumar und Sivakumar präsentiert [**A01**], der für große Dimensionen (> 50) mit großer Wahrscheinlichkeit in kürzerer Laufzeit als das HKZ-Verfahren eine Lösung für das SVP findet. Allerdings wird hierfür exponentiell viel Speicher benötigt, im Vergleich zum polynomiellen Speicherplatzbedarf herkömmlicher Enumerationsalgorithmen.

Es soll nun mithilfe der zuvor definierten Heuristiken und einer Variante des gerade erwähnten AKS-Algorithmus eine neuartige Möglichkeit geschaffen werden, aus einer endlichen Menge von Gittervektoren geeignete Kandidaten für einen kürzesten Lösungsvektor herauszufiltern.

Hierzu wird ein Siebverfahren verwendet, das auf den Ideen von AKS aufbaut, von Phong Q. Nguyen und Thomas Vidick [**N07**] weiterentwickelt und von mir hinsichtlich der Verwendung von Heuristiken adaptiert wurde.

Der Kernpunkt des Verfahrens sind drei Mengen - die Ausgangsmenge S der zu siebenden Vektoren, die Zielmenge S' für die heuristisch gut bewerteten Vektoren und die Zentrierungsmenge C , welche die heuristisch schlecht bewerteten Vektoren beinhaltet.

Der Ablauf ist relativ einfach und beruht im Wesentlichen darauf, das sukzessive Vektoren aus der Ausgangsmenge S herausgenommen und die heuristische Bewertung ermittelt wird. Liegt die Bewertung eines Vektors v unterhalb eines vorgegebenen Schwellwertes $\gamma \cdot R$, landet er in der Menge S', andernfalls in C. Doch bevor er dorthin gelangt, wird zunächst noch überprüft, ob es vielleicht nicht eventuell einen Vektor $c \in C$ gibt, so dass $H(v-c) \leq \gamma \cdot R$ ist. In diesem Fall wird der Vektor v nämlich nicht zur Menge C hinzugefügt, sondern $v - c$ wandert in S'.

Sind alle Elemente der Ausgangsmenge überprüft, wird S mit der Zielmenge S' reinitialisiert und das Spiel beginnt von vorne. Da sich die Anzahl der Elemente in jedem Durchlauf verringert und auch der Schwellenwert stetig abnimmt, terminiert das Verfahren nach endlich vielen Schritten. Am Ende wird aus den verbliebenen Vektoren derjenige mit der besten heuristischen Bewertung ermittelt und als Siebresultat zurückgegeben.

Der vollständige Algorithmus lautet damit:

ALGORITHMUS 116. (Heuristisches Sieb)

```
SIEVE (S = {v_1, v_2, ..., v_s}, γ)
{
  while (1) {

    R ⟵ max {H(v) | v ∈ S}

    if (R == 0)
      break;

    C ⟵ ∅
    S' ⟵ ∅

    for v ∈ S {

      if (H(v) ≤ γ · R)
        S' ⟵ S' ∪ {v}
      else {

        if (∃c ∈ C : H(v − c) ≤ γ · R)
          S' ⟵ S' ∪ {v − c}
        else
          C ⟵ C ∪ {v}
      }
    }

    if (|S'| == 0)
      break;

    S ⟵ S'
  }

  min ⟵ R

  for v ∈ S {

    if (H(v) < min) {
      min = H(v)
      v' ⟵ v
    }
  }

  return v'
}
```

Aufbauend auf der Annahme, dass die Vektoren der Menge $S \cap C_n(\gamma, R)$ gleichverteilt in dem Torus $C_n(\gamma, R) = \{x \in \mathbb{R}^n \,|\, \gamma R \leq ||x||_2 \leq R\}$ mit

$0 < \gamma < 1$ liegen, kann gezeigt werden, dass das Verfahren in der Fassung von Nguyen und Vidick mit großer Wahrscheinlichkeit einen kürzesten Vektor liefert, wenn $|S| \approx (\frac{4}{3})^{\frac{n}{2}}$ ist **[N07]**.

Diese Anzahl wird schnell sehr groß, was die folgende Tabelle verdeutlicht:

n	$(4/3)^{\frac{n}{2}}$
2	1.77
10	17.76
25	1328.83
50	1765781
100	3117982410208
250	1716659463758359581381902648975 4

Bereits ab Problemdimensionen > 100 ist es äußerst unpraktikabel eine solche Menge von Vektoren zu generieren und auszusieben.

In der Praxis zeigt sich allerdings, dass es gar nicht nötig ist mit so großen Sieben zu arbeiten. In vielen Fällen reicht es aus, wenn nur die heuristisch gut bewerteten Vektoren, die während der Abarbeitung der anderen Algorithmen abfallen, in die zu siebende Menge hineinverfrachtet werden.

Mit dieser Vorgehensweise konnten in äußerst kurzer Zeit sehr gute Ergebnisse erzielt werden (vgl. Fazit).

KAPITEL 5

Das Programmpaket

Zur Realisation der im Rahmen dieser Dissertation dargelegten Methoden wurde ein UNIX-Programm und diverse, zusätzliche Tools geschaffen, die sich auf der beiliegenden CD-ROM befinden.

Zur Entwicklung wurde der gcc-Compiler **[G87]** verwendet und zur Implementierung von Varianten mit beliebig wählbarer Präzision wurde die NTL-Library von Victor Shoup **[S08]** eingesetzt.

5.1. Installation und Kurzbeschreibung

Zur Installation muß zunächst die CD-ROM gemountet werden:

```
# mount /dev/cdrom /mnt/
```

Danach kann das Archiv entpackt werden:

```
# tar zxvf /mnt/rasa.tar.gz
```

In dem erzeugten Verzeichnis sollten sich folgende Dateien befinden:

- gen_ntru, gen_prb
- code2prb, lat2prb, lp2prb, ntru2prb
- prb2lp, prb2lat
- check_map, check_map-rr
- check_ntru, check_ntru-rr
- check_rev, check_rev-rr
- rasa, rasa-rr

Bemerkung 117. Die Programme liegen in einer 64-Bit Fassung vor. Entsprechende kompilierte Versionen für die 32-Bit Architektur befinden sich im Unterverzeichnis „arch/x86“.

Für das Hauptprogramm *rasa*, sowie die Verifizierer *check_ map, check_ ntru* und *check_ rev* existiert neben der Version mit double-Präzision auch eine Variante mit beliebig wählbarer Genauigkeit. Letztere besitzt die Endung *-rr* und verfügt über den zusätzlichen Kommandozeilenparameter *prec*, der angibt mit wie vielen Präzisions-Bits gerechnet werden soll.

Bevor näher auf das Hauptprogramm und dessen Kommandozeilenparameter eingegangen wird, erfolgt zunächst eine Kurzbeschreibung der diversen Zusatzprogramme:

gen_ ntru erzeugt ein Public/Private-Key Schlüsselpaar für das Kryposystem NTRU, das in einer Datei mit der Endung *.ntru* gesichert wird. Diese

Datei kann anschließend mit dem Tool *ntru2prb* in das *prb*-Format konvertiert werden. Um direkt mit einer Datei im *prb*-Format zu starten, empfiehlt sich die Verwendung von *gen_prb*, mit dessen Hilfe sich Testinstanzen von Market-Split Problemen erzeugen lassen.

Für Codierungstheoretiker dürfte das Bash-Skript *scripts/get_code.sh* von Interesse sein - es ermöglicht direkten Zugriff auf eine Online-Datenbank, in der Informationen zu Linearen Codes hinterlegt sind. Das Skript speichert diese Information in Files, die mit der Endung *.code* versehen sind. Anschließend können diese Dateien durch einen Aufruf des Konverters *code2prb* ins *prb*-Format gebracht werden (siehe 5.3.3).

Damit die Performance von *rasa* mit kommerziellen Solvern wie etwa ILOG CPLEX verglichen werden kann, muss das CPLEX lp-Format ins prb-Format überführt werden. Dies wird von *lp2prb* bewerkstelligt. Die entgegengesetzte Konvertierung übernimmt *prb2lp*.

Die Problemdateien (Endung: *.prb*) können mit Hilfe des Konvertierungsprogramms *prb2lat* in äquivalente Gitterprobleme übersetzt werden, die nun vom Hauptprogramm *rasa* verarbeitet werden können.

Nach Aufruf des Hauptprogramms verbleibt eine Datei mit der Endung *.map_*PID** im Arbeitsverzeichnis. Darin befindet sich das aktuelle Gitter, die zugehörige Transformationsmatrix, der heuristisch kürzeste Gittervektor und der kürzeste Lösungsvektor mit Schlupf. Vollständige Lösungen werden am Ende der Datei aufgelistet.

Die Konsistenz der *map*-Files läßt sich durch einen Aufruf von *check_map* überprüfen. Hierbei werden auch gleich evtl. gefundene (vollständige) Lösungsvektoren in entsprechende Lösungen des ursprünglichen Problems zurückübersetzt und in einer Datei mit der Endung *.sol* gesichert. Der Gittervektor mit der besten heuristischen Bewertung landet in einer Datei mit der Endung *.heu*, während ein unvollständiger Lösungsvektor (mit Schlupf) in einer Datei mit der Endung *.slk* landet.

Bemerkung 118. Auf den Unterschied zwischen vollständiger FP und unvollständiger IP-Lösung wird im Anhang nochmals näher eingegangen.

Zur nachträglichen Überprüfung von Dateien mit der Endung *.sol*, *.heu* oder *.slk* ist *check_rev* gedacht. Damit werden die Eingabedaten zunächst wieder in Gittervektoren zurückübersetzt und nachgeprüft, ob diese auch wirklich im zugehörigen Gitter liegen. Falls die Lösungen aus Problemen entspringen, die im Rahmen des Kryposystems NTRU erzeugt wurden (gen_ntru), kann mit dem Programm *check_ntru* überprüft werden, ob die gefundenen Lösungen tatsächlich dem geheimen Schlüssel des Verfahrens entsprechen.

Neben den *map*-Files wird während des Hauptprogrammablaufs in regelmäßigen Abständen eine Datei mit der Endung *.nrm_*PID** geschrieben. Diese beinhaltet die euklidischen Längen der Basisvektoren des aktuellen Gitters und der dazu orthogonalen Vektoren. Zur grafischen Darstellung dieser Datei dient das Skript *scripts/draw_norm.sh*. Allerdings wird hierfür das frei erhältliche Statistikprogramm R benötigt **[R92]**.

5.2. Kommandozeilenparameter von *rasa*

--*seed* (ganzzahlig – gültige Werte: ≥ 0 | Default: 0)

Diese Kenngröße legt den Wert fest, mit dem der Zufallszahlengenerator initialisiert wird.

--*delta* (reellwertig – gültige Werte: > 0.75 und < 1 | Default: 0.99)

Der Faktor δ dient zur Abschwächung bzw. Verschärfung der Lovász-Bedingung und wird unter anderem innerhalb der Sampling Algorithmen SHORT bzw. ESHORT ausgewertet, um abzuwägen, wann ein zufällig erzeugter Gittervektor kurz genug ist.

--*alpha* (reellwertig – gültige Werte: > 0 und < 1 | Default: 0.5)

Der Faktor α wird von den „Bounding Functions“ für das Extreme Pruning ausgewertet. Je geringer der Wert gewählt wird, desto mehr verkleinert sich der Enumerationsbaum. Damit diese Größe zum Tragen kommt, muss allerdings mit dem Parameter *eprune* eine „Bounding Function“ festgelegt werden.

--*beta* (ganzzahlig – gültige Werte: ≥ 2 | Default: 20)

Die Blockgröße β, die maßgebend für das BKZ-Verfahren ist.

--*btime* (ganzzahlig – gültige Werte: ≥ 0 | Default: 0)

Diese Einflußgröße entscheidet, wie viele Sekunden eine BKZ-Reduktion maximal dauern darf. Ist das Sekundenlimit erreicht, wird das BKZ-Verfahren beendet und eine abschließende LLL-Reduktion durchgeführt. Das Programm wird allerdings nicht abgebrochen, lediglich der Laufzeitanteil der BKZ-Reduktionen wird geregelt.

--*slimit* (ganzzahlig – gültige Werte: ≥ 0 | Default: 20000)

Die Anzahl der Vektoren, die pro Random Sampling - Aufruf höchstens erzeugt werden.

--*gsa* (ganzzahlig – gültige Werte: $0, 1, 2$ | Default: 0)

Das Random Sampling wird durch die Strategie zur Bestimmung des GSA-Verhaltens konfiguriert. Defaultmässig ist keine Strategie vorgegeben, d.h. Random Sampling ist deaktiviert. Soll die Strategie von Schnorr verwendet werden, muss der Parameter gleich 1 gesetzt werden. Bevorzugt man hingegen die Buchmann'sche Strategie, so ist er mit 2 zu initialisieren.

--*prj* (ganzzahlig – gültige Werte: $0, 1$ | Default: 0)

Bei Verwendung des *gsa* Parameters wird nur der Algorithmus zur Erzeugung kurzer Gittervektoren aktiviert. Um zusätzliche

Routinen zur Suche nach kurzen Projektionen (ESHORT) einzuschalten, ist prj gleich 1 zu setzen.

--*nprune* (ganzzahlig – gültige Werte: $0, 1, 2$ | Default: 0)

Das Abschneiden von Enumerationsbäumen ist in der Default-Einstellung nicht aktiviert. Bei Verwendung des Parameterwerts 1 werden nur die wenig zeitaufwändigen Pruning-Tests durchgeführt. Rechenintensivere Tests werden mit dem Parameterwert 2 eingeschaltet (vgl. Kapitel 2.4).

--*eprune* (ganzzahlig – gültige Werte: $0, 1, 2, 3$ | Default: 0)

Mit diesem Switch wird die Extreme Pruning - „Bounding Function“ festgelegt, die verwendet werden soll. Standardmäßig ist Extreme Pruning deaktiviert.

--*deep* (ganzzahlig – gültige Werte: ≥ 0 | Default: 0)

Ein Wert > 0 bedeutet, dass die LLL-Variante mit Tiefeneinfügungen verwendet wird. Der Wert von *deep* gibt dabei an, auf wieviele Vektoren der Lovász-Schritt ausgedehnt wird.

--*heu* (ganzzahlig – gültige Werte: $0, 1, 2, 3$ | Default: 0)

Mit dieser Größe wird geregelt, welche heuristische Strategie verwendet wird. Defaultmäßig ist die euklidische Strategie aktiv (vgl. Kapitel 4.1).

--*pwr* (ganzzahlig – gültige Werte: $0, 1$ | Default: 1)

Paarweise Reduktionen werden defaultmässig immer durchgeführt und können mit dieser Option deaktiviert werden.

--*sieve* (ganzzahlig – gültige Werte: ≥ -1 | Default: -1)

Zur Aktivierung der Sieb-Funktionalität muss dieser Parameter mit der gewünschten Siebgröße initialisiert werden. Soll ein Sieb verwendet werden, dessen Größe sich dynamisch anpasst, ist der Parameter gleich 0 zu setzen.

--*scale* (ganzzahlig – gültige Werte: ≥ 0 | Default: 0)

Diese Steuergröße entspricht der Schrittweite, die angibt wie stark das Gitter pro Skalierungsschritt verzerrt wird. Leider ist diese Funktion noch nicht vollständig implementiert, weshalb defaultmäßig kein Skalierungsversuch unternommen wird.

Die nächsten 3 Parameter sind relevant für das Abbruchverhalten von *rasa*.

--*rtime* (ganzzahlig – gültige Werte: ≥ 0 | Default: 0)

Hiermit wird festgelegt, nach wie vielen Sekunden Laufzeit das

Programm beendet wird. Standardmäßig ist der Parameter mit 0 initialisiert, was unendlich langer Laufzeit entspricht.

--*sol* (ganzzahlig – gültige Werte: ≥ 0 | Default: 1)

Bevorzugt man einen Programmabbruch aufgrund der Anzahl der gefundenen Lösungen, dann ist dieses Kriterium zu verwenden. Defaultmäßig terminiert das Programm, sobald eine Lösung gefunden wurde. Wird kein Limit gewünscht, ist der Wert gleich 0 zu setzen.

--*slk* (ganzzahlig – gültige Werte: ≥ 0 | Default: 0)

Für Gitter, die aus linearen Optimierungsproblemen mit Ungleichungen resultieren, ist diese Abbrucheinstellung gedacht. Das Programm terminiert, sobald ein Gittervektor gefunden wird, der die Gleichungen des zugrundeliegenden Optimierungsproblems exakt erfüllt und dessen Abweichung von den Grenzen in den Ungleichungskomponenten kleiner gleich *slk* ist (vgl. Kapitel 5.4).

Zur Begegnung von Programmabbrüchen wegen Genauigkeitsproblemen, dienen die nächsten beiden Einstellmöglichkeiten:

--*prec* (ganzzahlig – gültige Werte: ≥ 64 und $< 2^{32}$ | Default: 128)

Dieser Parameterwert kann nur mit der rr-Variante von *rasa* verwendet werden und legt fest, mit wie vielen Präzisions-Bits Gleitkommaoperationen durchgeführt werden sollen.

--*trans* (ganzzahlig – gültige Werte: $0, 1$ | Default: 1)

Es wird an verschiedenen Stellen innerhalb des Programms überprüft, ob das aktuelle Gitter mit der Transformation des ursprünglichen Gitters übereinstimmt. Soll die Laufzeit des Programms verringert werden, empfiehlt es sich diesen Parameter gleich 0 zu setzen, um die Transformations - Checks zu deaktivieren. Allerdings beraubt man sich damit der Möglichkeit Rechen-, bzw. Präzisionsfehler frühzeitig zu erkennen.

Der folgende Parameter ist relevant für das Fortsetzen eines zuvor durchgeführten *rasa*-Durchlaufs.

--*map* (Zeichenkette – gültige Werte: *.map_*PID* | Default: “”)

Damit abgebrochene Programm-Aufrufe zu einem späteren Zeitpunkt fortgesetzt werden können, muss lediglich beim Programmaufruf die korrespondierende *map*-Datei angegegeben werden.

Die letzte Einstellmöglichkeit beschäftigt sich mit dem Ausgabeverhalten:

--*verb* (ganzzahlig – gültige Werte: $0, 1, 2, 3$ | Default: 0)

Normalerweise werden zur Laufzeit keine Statusmeldungen ausgegeben. Um nach jedem Hauptschleifendurchlauf statistische Informationen zu erhalten, ist die Einstellung 1 zu wählen. Bevorzugt man Informationen, die auch den Status der unterschiedlichen Programmfunktionen reflektieren, ist der Wert gleich 2 zu setzen. Sollte es zu unerklärlichen Programmabbrüchen kommen, können durch Verwendung des letzten Parameterwerts zusätzliche Warnmeldungen aktiviert werden.

5.3. Eingabeformate

Das Hauptprogramm *rasa* erwartet als Eingabe eine Datei im lat-Format, dessen Aufbau am folgenden Beispiel erläutert wird:

beispiel.lat

```
[
[ 0 0 -80 30 0 0 ]
[ -16 420 0 0 -72 0 ]
[ 21 0 20 30 36 60 ]
]

[ -21 -60 -60 -60 -60 -60 ]
[ 21 60 60 60 60 60 ]
[ 0 0 0 0 0 -1 ]

[ 0 30 20 15 12 0 ]

60
0
0

1
```

Am Anfang steht die Basismatrix des zu reduzierenden Gitters. Danach folgen drei Vektoren mit den Anforderungen, die ein Lösungsvektor erfüllen muss. Der erste enthält die unteren Grenzen, der zweite die oberen. Der dritte Vektor reflektiert, in welchen Komponenten Null-Einträge erlaubt sind. Falls 0 ein gültiger Wert ist, steht in der Komponente eine 0, andernfalls -1.

Der vierte Vektor, der die Faktoren zur Umrechnung von Gittervektoren in Lösungen der zugehörigen linearen Probleme beinhaltet, ist nur dann relevant, wenn das Gitter durch einen Aufruf des Hilfsprogramms *prb2lat* erzeugt wurde. Ist man an dieser Umrechnung nicht interessiert, z.B. weil man direkt mit einem Gitter startet, ist es ratsam diesen Vektor leer zu lassen, d.h. einfach [ ] in die Datei zu schreiben.

Es folgen 3 einzelne Werte, die wiederum nur dann von Interesse sind, wenn die Datei durch einen Aufruf von *prb2lat* erzeugt wurde.

Der letzte Wert entspricht der Anzahl der Ungleichungen, die im ursprünglichen linearen Problem vorhanden sind. In diesem Beispiel ist also nur eine Ungleichung beteiligt.

Startet man direkt mit einem Gitter können die letzten $3 + 1$ Werte auch weggelassen werden. In diesem Fall würde die Datei dann folgendes Aussehen haben:

beispiel.lat

```
[
[ 0 0 -80 30 0 0 ]
[ -16 420 0 0 -72 0 ]
[ 21 0 20 30 36 60 ]
]

[ -21 -60 -60 -60 -60 -60 ]
[ 21 60 60 60 60 60 ]
[ 0 0 0 0 0 -1 ]

[ ]
```

Hier würde nun also in dem Gitter, das von $b_1 = (0, 0, -80, 30, 0, 0)^T$, $b_2 = (-16, 420, 0, 0, -72, 0)^T$ und $b_3 = (21, 0, 20, 30, 36, 60)^T$ erzeugt wird, nach Lösungsvektoren v gesucht mit:

$$(-21, -60, -60, -60, -60, -60)^T \leq v \leq (21, 60, 60, 60, 60, 60)^T$$

und $v_6 \neq 0$.

Bemerkung 119. Da das Hauptprogramm nur Dateien im lat-Format einlesen kann, müssen andere Formate mit den jeweiligen Hilfsprogramme ins lat-Format konvertiert werden.

5.3.1. prb → lat. Das prb-Format fungiert als das Standardformat zur Speicherung eines linearen Problems der Form:

Finde ein $x \in \mathbb{Z}^n$, so dass $l_u \leq A_u \cdot x \leq r_u$, $A_g \cdot x = b_g$ und $0 \leq x \leq r$,
wobei $A_u \in \mathbb{Z}^{m_u \times n}, A_g \in \mathbb{Z}^{m_g \times n}, l_u, r_u \in \mathbb{Z}^{m_u}, b_g \in \mathbb{Z}^{m_g}$ und $r \in \mathbb{N}^n$.

Beispielsweise sei das folgende Problem gegeben:

Finde ein $x \in \mathbb{Z}^n$ mit $(0, 0, 0, 0)^T \leq x \leq (2, 3, 4, 5)^T$, so dass:

$$0 \leq x_1 + 5x_4 \leq 21$$
$$2 \leq x_2 + x_3 \leq 7$$
$$2x_2 + 4x_3 = 16$$
$$3x_1 + 7x_4 = 31$$

Man erhält daraus: $A_u = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 1 & 0 \end{bmatrix}$, $A_g = \begin{bmatrix} 0 & 2 & 4 & 0 \\ 3 & 0 & 0 & 7 \end{bmatrix}$, $l_u = (0,2)^T$, $r_u = (21,7)^T$, $b_g = (16,31)^T$ und $r = (2,3,4,5)^T$.

D.h. der Inhalt der zugehörigen prb-Datei wäre folgender:

beispiel.prb

```
[
[ 1 0 0 5 ]
[ 0 1 1 0 ]
]

[
[ 0 2 4 0 ]
[ 3 0 0 7 ]
]

[ 0 2 ]
[ 21 7 ]

[ 16 31 ]

[ 2 3 4 5 ]

1
```

Am Ende der Datei steht noch eine einzelne Zahl, die entweder gleich 0 oder 1 ist. Damit wird festgelegt, ob bei der Übersetzung des Problems in das lat-Format, die Dimension des resultierenden Gitters mit Hilfe des Aardal-Ansatzes verringert werden soll. Wenn dies gewünscht ist, muss der Wert gleich 1 gesetzt werden.

Um nun ein Problem in ein Gitter einzubetten (vgl. Kapitel 3) ist nachstehendes Kommando auszuführen:

```
# ./prb2lat beispiel.prb
```

Resultat dieses Aufrufs ist eine Datei mit dem Namen „beispiel.lat“, die den folgenden Inhalt besitzt:

beispiel.lat

```
[
[ 0 -2 0 -80 30 0 0 ]
[ -16 0 420 0 0 -72 0 ]
[ 21 1 0 20 30 36 60 ]
]
```

beispiel.lat

```
[ -21 -5 -60 -60 -60 -60 -60 ]
[ 21 5 60 60 60 60 60 ]
[ 0 0 0 0 0 0 -1 ]

[ 0 0 30 20 15 12 0 ]

60
0
0

2
```

REMARK. $R = 60$ ist das kleinste gemeinsame Vielfache der Komponenten von $r = (2, 3, 4, 5)^T$ und befindet sich in der Datei direkt unterhalb des Umrechnungsvektors c. Für diesen gilt unter anderem: $c_{m_u+j} = \frac{R}{r_j}, j = 1, 2, ..., n$. Die beiden anderen mit 0 belegten Werte, werden nur dann benötigt, wenn der *dia* Parameter von *prb2lat* verwendet wird.

5.3.2. lp → prb. Um ein lineares Problem im CPLEX lp-Format einzulesen, wird das Programm *lp2prb* benötigt, mit dessen Hilfe man das Problem ins prb-Format umwandeln kann.

Die einzulesende Datei „beispiel.lp“ besitze den folgenden Inhalt:

beispiel.lp

```
Minimize
 obj:
Subject To
 + 1 x1 + 5 x4 <= 21
 + 1 x1 + 5 x4 >= 0
 + 2 x2 + 4 x3 = 16
 + 3 x1 + 7 x4 = 31
Bounds
 0 <= x1 <= 2
 0 <= x2 <= 3
 0 <= x3 <= 4
 0 <= x4 <= 5
General
 x1 x2 x3 x4
End
```

Durch den folgenden Kommandoaufruf wird die Datei ins prb-Format konvertiert und unter dem Namen „beispiel.prb“ abgespeichert:

```
# ./lp2prb beispiel.lp
```

Die resultierende Datei „beispiel.prb“ besitzt den Inhalt:

beispiel.prb

```
[
[ 1 0 0 5 ]
]

[
[ 0 2 4 0 ]
[ 3 0 0 7 ]
]

[ 0 ]

[ 21 ]

[ 16 31 ]

[ 2 3 4 5 ]

1
```

Bemerkung 120. Falls der entgegengesetzte Weg gewünscht ist, d.h. eine Konvertierung vom prb-Format ins CPLEX lp-Format ist das Hilfsprogramm *prb2lp* zu verwenden.

5.3.3. code → prb. Das code-Format beinhaltet Daten zu Linearen Codes und wird von dem Shell-Skript *scripts/get_ code.sh* generiert. Das Skript lädt für alle bekannten q-(N, k, d) Codes die Inzidenzmatrizen [**B09**] herunter.

Remark. Die Matrizen sind bereits vereinfacht, d.h. sie sind Resultate einer Operation von Automorphismengruppen (vgl. Anhang).

Die $m \times n$ - Matrix M und der Vektor ω mit den zugehörigen Orbit-Längen werden in einer Datei mit der Endung *.code* gespeichert.

Das folgende Beispiel beschreibt einen 2-$(4, 3, 2)$ Code:

beispiel.code

```
[
[ 1 0 1 1 0 ]
[ 0 1 0 1 1 ]
[ 1 0 1 0 1 ]
[ 2 1 0 0 0 ]
[ 0 1 2 0 0 ]
]
```

```
[ 2 1 2 1 1 ]

4
2
```

Am Ende der Datei stehen die Dimension N und die Minimaldistanz d.

Die Suche nach einem q-(N, k, d) Code reduziert sich auf das Auffinden einer Lösung x für das ganzzahlig-lineare Gleichungssystem:

$$M \cdot x \leq \begin{pmatrix} N-d \\ N-d \\ \vdots \\ N-d \end{pmatrix},$$

$$\sum_{j=1}^{n} \omega_j \cdot x_j = N$$

$$\text{und } 0 \leq x_j \leq 1 \text{ für } j = 1, 2, \ldots, n$$

Es gibt zwei Varianten dieses Gleichungssystem zu behandeln - mit Schlupfvariablen - oder ohne. Möchte man ohne Schlupfvariablen arbeiten, ist der folgende Aufruf zu tätigen:

```
# ./code2prb --slacks=0 beispiel.code
```

Damit wird das Gleichungssystem unverändert in eine prb-Datei geschrieben.

Im anderen Fall wird das System durch die Einführung von Schlupfvariablen derart abgeändert, dass nur Gleichungen vorliegen und man erhält die äquivalente Darstellung:

$$\begin{bmatrix} & -1 & 0 & 0 \\ M & 0 & \ddots & 0 \\ & 0 & 0 & -1 \\ \omega & 0 & 0 & 0 \end{bmatrix} \cdot \tilde{x} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ N \end{pmatrix},$$

$$0 \leq \tilde{x}_j \leq 1 \text{ für } j = 1, 2, \ldots, n$$
$$\text{und } 0 \leq \tilde{x}_j \leq N - d \text{ für } j = n+1, n+2, \ldots, n+m$$

Mit dem Aufruf

```
# ./code2prb --slacks=1 beispiel.code
```

erhält man jetzt dieses Resultat:

beispiel.prb

```
[
]

[
[ 1 0 1 1 0 -1 0 0 0 0 ]
[ 0 1 0 1 1 0 -1 0 0 0 ]
[ 1 0 1 0 1 0 0 -1 0 0 ]
[ 2 1 0 0 0 0 0 0 -1 0 ]
[ 0 1 2 0 0 0 0 0 0 -1 ]
[ 2 1 2 1 1 0 0 0 0 0 ]
]

[ ]

[ ]

[ 0 0 0 0 0 4 ]

[ 1 1 1 1 1 2 2 2 2 2 ]

1
```

5.3.4. ntru → prb. Das *ntru*-Format dient zur Speicherung der Parameter des Public-Key Kryptosystems NTRU [**S98**].

Das Programm *ntru2prb* generiert aus dem öffentlichen Schlüssel h des Kryptosystems ein ganzzahliges, lineares Optimierungsproblem der Form:

$$\begin{bmatrix} & & & -1 & & & q & & \\ & H & & & \ddots & & & \ddots & \\ & & & & & -1 & & & q \\ 1 & \dots & 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & 1 & \dots & 1 & 0 & \dots & 0 \end{bmatrix} \cdot \begin{pmatrix} F \\ g \\ k_0 \\ k_1 \\ \vdots \\ k_{n-1} \end{pmatrix} = \begin{pmatrix} -\hat{h} \\ df \\ dg \end{pmatrix}$$

$$\text{wobei } H := \begin{bmatrix} h_o & h_1 & \dots & h_{n-1} \\ h_1 & h_2 & \dots & h_0 \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-1} & h_0 & \dots & h_{n-2} \end{bmatrix}$$

Dieses Problem besitzt modulo q genau eine 0/1 - Lösung (F, g), die gerade dem geheimen, privaten Schlüssel des Systems entspricht [**V05**].

5.4. Aufruf des Programms

Nachdem das Gitterproblem in Form einer lat-Datei vorliegt, wird es Zeit für den ersten *rasa*-Aufruf:

```
# ./rasa --beta=25 --heu=1 --sol=4 beispiel.lat
```

Diese Kommandozeile veranlasst das Programm BKZ - Reduktionen mit Blockweite 25 durchzuführen und für heuristische Tests die erste Bewertungsfunktion zu verwenden. Sobald 4 unterschiedliche, vollständige Lösungen (*sol*) gefunden wurden, terminiert das Programm.

Wenn bei dem zugrundeliegendem Optimierungsproblem Ungleichungen vorhanden waren, so kann mit Hilfe des *slk* Parameters auch nach unvollständigen Lösungen gesucht werden, also nach Lösungen, die zwar dem Gleichungsanteil genügen, aber den Ungleichungs-Part nicht komplett erfüllen:

```
# ./rasa --slk=13 beispiel.lat
```

Nun wird abgebrochen, sobald eine (unvollständige) Lösung x gefunden wird, d.h. x erfüllt alle Gleichungen und für den Ungleichungsteil gilt:

$$s := \sum_{i=1}^{m_u} \max(0, [l_u - A_u x]_i, [A_u x - r_u]_i) \leq 13$$

($[z]_i$ bezeichnet die i-te Komponente des Vektors z).

Während des Programmablaufs werden zwei Dateien generiert, die mit der PID des *rasa*-Prozesses enden:

```
beispiel.map_PID, beispiel.nrm_PID
```

Die map_PID - Datei enthält die aktuelle Gitterbasis B' und unimodulare Transformationsmatrizen $T_1, T_2, ..., T_t$. Desweiteren enthält die Datei Informationen über die verwendeten Kommandozeilenparameter und die gefundenen Lösungen.

Die Konsistenz des map-Files läßt sich mit *check_map* überprüfen.

```
# ./check_map beispiel.map_PID
```

Hierbei wird verifiziert, ob $B' = T_t \cdot ... \cdot T_2 \cdot T_1 \cdot B$, wenn B die anfängliche Gitterbasis bezeichnet. Es werden außerdem sämtliche gefundene exakte bzw. unvollständige Lösungen in Dateien mit der Endung *sol* bzw. *slk* vermerkt. Heuristisch kürzeste Vektoren landen in einer Datei mit der Endung *heu*.

Soll das Programm zu einem späteren Zeitpunkt (evtl. mit veränderten Argumenten) an genau der Stelle weiterrechnen, an der abgebrochen wurde, ist der *map*-Parameter zu verwenden:

```
# ./rasa --map=beispiel.map_PID beispiel.lat
```

Bemerkung 121. Die Ausgabe von Statusmeldungen wird durch den Kommandozeilenparameter *verb* geregelt.

Die nrm_PID - Datei beinhaltet die euklidische Längeninformation der Basis und der zugehörigen Orthogonalvektoren. Diese Information und das entsprechende GSA-Verhalten lässt sich mithilfe des Statistikprogramms *R* [**R92**] und dem folgenden Skriptaufruf veranschaulichen:

```
# ./scripts/draw.sh beispiel.nrm_PID
```

5.4.1. Ablauf des Hauptprogramms. Schematisch lässt sich der Programmablauf folgendermaßen darstellen:

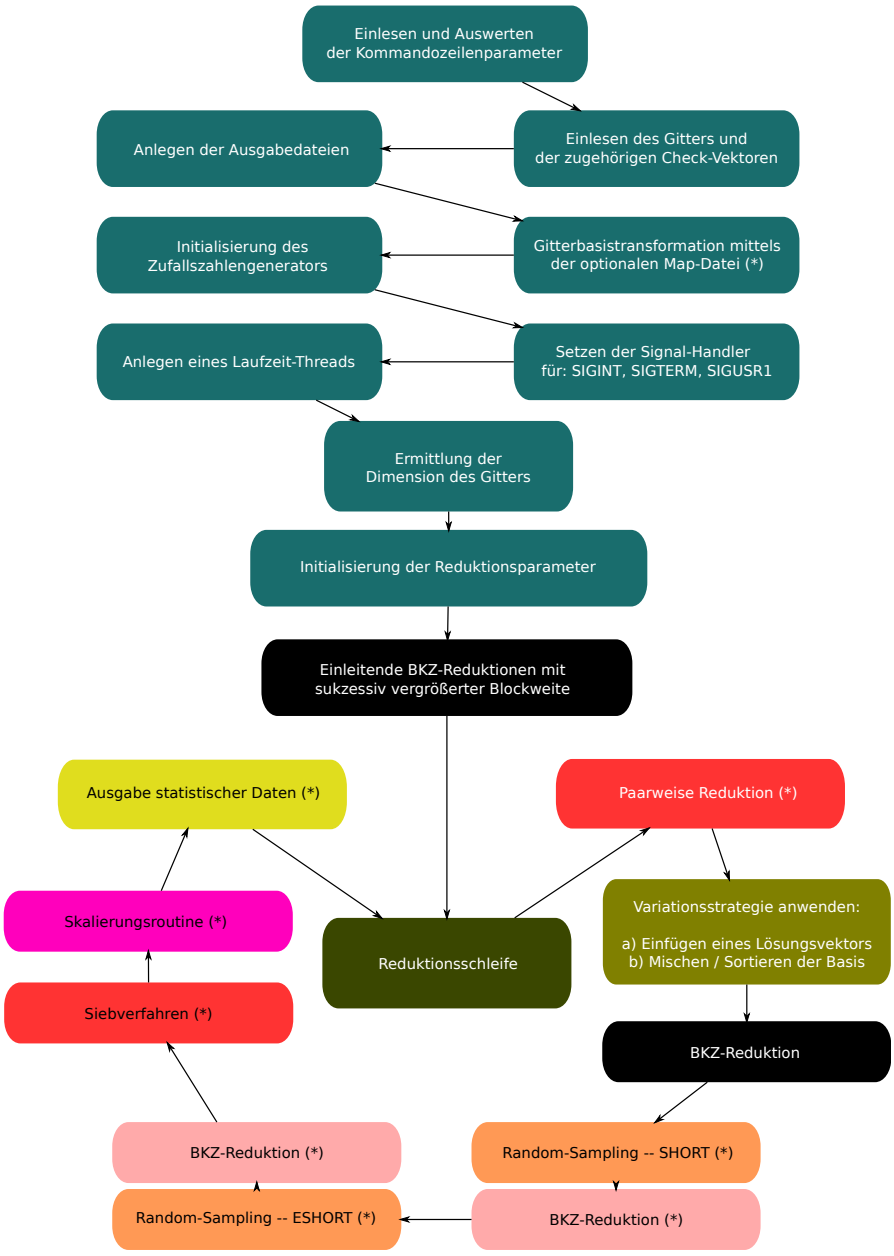


ABBILDUNG 5.4.1. Darstellung des Programmfluss

Nachdem das Gitter eingelesen wurde und der Initialisierungsvorgang beendet ist, werden in einer Schleife über β (beginnend bei $\beta = 2$) BKZ-Reduktionen durchgeführt, bis die gewünschte Blockweite erreicht ist. Danach beginnt die Hauptprogramm-Schleife, die endlos wiederholt wird, derweil nicht eines der Abbruchkriterien (*rtime, sol*, *slk*) erfüllt ist.

Innerhalb der Hauptprogramm-Schleife werden zunächst einfache Paarweise Reduktionen durchgeführt, bis keine Verbesserung der heuristischen Bewertung der Basisvektoren mehr möglich ist. Zwischen zwei Reduktionsschritten wird außerdem jedesmal der heuristisch kürzeste Gittervektor $b^{(heu)}$ in die Basis eingefügt.

Als nächstes wird überprüft, ob sich bereits Vektoren in dem Pool mit den gefundenen Lösungsvektoren befinden. Ist dies der Fall, wird ein Vektor aus dem Pool in die Basis eingefügt. Auf diese Art können Lösungscluster aufgespürt werden, die sich in direkter Nachbarschaft zur eingefügten Lösung befinden. Wenn der Pool leer ist oder bereits alle Lösungen getestet wurden, wird per Zufallsentscheid bestimmt, ob die Basisvektoren sortiert oder durchgemischt werden sollen, um wenigstens auf diese Weise etwas Bewegung in die Algorithmen zu bringen und um Endlosschleifen vorzubeugen.

Nach der folgenden BKZ-Reduktion werden zwei Indizes $1 \leq i < j \leq m$ zufällig bestimmt und anschließend mit dem Untergitter $L(b_i, b_{i+1}, ..., b_j)$ Random-Sampling betrieben. Dies geschieht durch Aufruf der Sampling Routinen zur Erzeugung kurzer Gittervektoren bzw. Projektionen. Nach einem erfolgreichen Sampling-Schritt wird der gefundene Vektor in die Basis eingefügt und zusätzlich in die Menge der zu siebenden Vektoren kopiert. Direkt im Anschluss wird wieder BKZ-reduziert. Dieses Vorgehen wird nun solange iteriert, bis der Sampling-Algorithmus keine guten Vektoren mehr liefert.

Danach ist es Zeit für die Durchführung des Siebverfahrens. Anfänglich wird überprüft, ob die Kapazität der zu siebenden Menge ausgeschöpft ist. Ist dies nicht der Fall, werden mit dem Babai'schen Rundungsalgorithmus zufälligverteilte Gittervektoren erzeugt, die in die Menge eingefügt werden, bis die maximale Elementanzahl erreicht ist. Sodann startet die Siebprozedur und liefert als Resultat einen Vektor b. Nun wird überprüft, ob die heuristische Bewertung des Vektors b geringer ausfällt, als die heuristische Bewertung von $b^{(heu)}$. Sollte dies der Fall sein, wird $b^{(heu)}$ durch b ersetzt.

Abschließend wird eine Skalierungsroutine aufgerufen, verschiedene statistische Daten ausgegeben und die Reduktions-Schleife beginnt von vorne.

Bemerkung 122. Bei der schematischen Darstellung des Programmflusses in Abbildung 5.4.1 sind die optionalen Schritte mit einem Stern (*) gekennzeichnet - erst durch das Setzen entsprechender Kommandozeilenparameter werden diese aktiviert.

KAPITEL 6

Laufzeitvergleiche

Im letzten Kapitel dieser Dissertation wird der Einfluß spezieller Parameterkombinationen auf das Laufzeitverhalten des Programms dokumentiert.

Die folgenden Parametersätze werden verwendet:

Run ID	Kommandozeile
H0	–heu=0 –sol=1
H1	–heu=1 –sol=1
H2	–heu=2 –sol=1
H3	–heu=3 –sol=1
D0	–deep=50 –heu=0 –sol=1
D1	–deep=50 –heu=1 –sol=1
D2	–deep=50 –heu=2 –sol=1
D3	–deep=50 –heu=3 –sol=1
NP1	–beta=40 –nprune=1 –sol=1
NP2	–beta=40 –nprune=2 –sol=1
EP1	–beta=40 –alpha=0.5 –eprune=1 –sol=1
EP2	–beta=40 –alpha=0.5 –eprune=2 –sol=1
EP3	–beta=40 –alpha=0.25 –eprune=3 –sol=1
R1	–slimit=20000 –gsa=1 –prj=0 –sol=1
R2	–slimit=20000 –gsa=1 –prj=1 –sol=1
R3	–slimit=20000 –gsa=2 –prj=0 –sol=1
R4	–slimit=20000 –gsa=2 –prj=1 –sol=1
S0	–sieve=0 –heu=0 –sol=1
S1	–sieve=0 –heu=1 –sol=1
S2	–sieve=0 –heu=2 –sol=1
S3	–sieve=0 –heu=3 –sol=1
K1	–heu=1 –deep=50 –sieve=0 –sol=1
K2	–beta=50 –deep=100 –alpha=0.5 –eprune=1 –sol=1
K3	–heu=3 –sieve=0 –gsa=1 –prj=1 –sol=1
K4	–beta=40 –deep=100 –alpha=0.5 –eprune=2 –sieve=0 –sol=1

- Falls ein Parameter nicht in der Kommandozeile angegeben wird, hat dies eine Initalisierung mit dem zugehörigen Standardwert zur Folge (siehe Abschnitt 5.2).

- Um die Reproduzierbarkeit der Ergebnisse gewährleisten zu können, wurde bei sämtlichen Parametersätzen der Startwert des Zufallszahlengenerators mit „–seed=17101978“ vorbelegt.

Im Rahmen der Laufzeitvergleiche sollte insbesondere überprüft werden, inwiefern sich das Programm zur Lösung von ganzzahlig-linearen Gleichungssystemen eignet. Hierzu wurden einige zufällig erzeugte Market-Split Problemen aus der 9-er, 10-er und 11-er Kategorie getestet, die mit den folgenden Bash-Kommandos erzeugt wurden:

```
# cd rasa
# for i in 9 10 11 ; do
#   for n in $(seq 1 5) ; do
#     j=$((${i}-1)*10))
#     ./gen_prb --M=${i} --N=${j} ${i}x${j}_${n}.prb
#     ./prb2lat ${i}x${j}_${n}.prb
#   done
# done
```

Danach wurden Testreihen mit harten Market-Sharing Referenz-Problemen durchgeführt (`http://www.tamiu.edu/~hwang/research/marketshare/`).

Die Dateien liegen im CPLEX lp-Format vor und müssen deshalb zunächst ins lat-Format gebracht werden:

```
# cd rasa
# for i in 7 8 9 10 ; do
#   for n in $(seq 1 5) ; do
#     ./lp2prb markshare_${i}_${n}.lp
#     ./prb2lat markshare_${i}_${n}.prb
#   done
# done
```

Bemerkung 123. Die Dimension der resultierenden Gitter-Probleme bewegt sich zwischen 50 und 100.

Schließlich wurden noch ausgewählte Challenge-Probleme angegangen, die von der TU Darmstadt (`http://www.latticechallenge.org/`) bereitgestellt wurden, um die Effizienz von Algorithmen zur Gitterbasenreduktion vergleichen zu können. Aufgrund der großen Dimension (≥ 500) und der damit verbundenen Laufzeiterhöhung wurden diese Probleme allerdings nur mit der Parameterkombination K3 angegangen.

Sämtliche Problemdaten (und die zugehörigen Lösungen) sind in den Unterordnern „msp“, „msh“ und „challenge“ des Verzeichnis „test“ wiederzufinden.

Es folgen die Tabellen mit den Laufzeitvergleichen. Jede Zelle enthält die Zeit in Sekunden, die bis zur Lösungsfindung benötigt wurde. Falls hinter

der Sekundenangabe noch ein eingeklammerter Wert steht, konnte das FP-Problem in der vorgebenen Gesamtlaufzeit nicht gelöst werden - stattdessen wurde versucht eine Lösung für die IP-Version zu finden. Der Wert in der Klammer gibt dann den zugehörigen Schlupf wieder (vgl. Anhang).

Run	MSP9-1	MSP9-2	MSP9-3	MSP9-4	MSP9-5
CPLEX	2700 (16)	2700 (13)	2700 (14)	2700 (15)	2700 (16)
H0	464	406 (0)	730	166	1105 (0)
H1	695	54 (0)	482	12 (1)	321 (0)
H2	682	279	1667	1074	2546
H3	729	123	799	123 (1)	1271
D0	412	58	336	192	100
D1	80	40 (0)	372	122	429
D2	175	50 (0)	569	311	260
D3	399	104 (0)	256	464	263
NP1	299	509 (1)	181	740	205 (1)
NP2	1993	64 (2)	1201	59 (1)	1363 (1)
EP1	345 (0)	451 (0)	206	93 (1)	48 (1)
EP2	64	146	245	735 (0)	1383
EP3	642	148	359	581	251 (1)
R1	505	137	334	245 (0)	752
R2	924	62	1000	246 (0)	729
R3	803	168	769	175 (1)	769
R4	569	1393 (0)	716	176 (1)	767
S0	477	405 (0)	702	168	1113 (0)
S1	702	54 (0)	450	261	326 (0)
S2	162 (1)	1739 (0)	524 (1)	32 (1)	40 (1)
S3	732	123	729	9 (1)	1260
K1	192	39 (0)	397	101	62
K2	126	181	140	92	629
K3	453 (0)	30	263	710	264
K4	136	93 (0)	96	153	223

TABELLE 1. Market-Sharing Probleme: M=9, N=80, Laufzeit: 45m

Run	MSP10-1	MSP10-2	MSP10-3	MSP10-4	MSP10-5
CPLEX	14400 (27)	14400 (24)	14400 (24)	14400 (23)	14400 (22)
H0	581 (0)	134 (1)	3011 (0)	561 (1)	112 (1)
H1	1220 (0)	30 (1)	4737 (0)	132 (1)	104 (1)
H2	970 (0)	64 (1)	11151 (0)	185 (1)	188 (1)
H3	838 (0)	66 (1)	942 (0)	501 (1)	38 (1)
D0	1624 (0)	170 (1)	719 (0)	1311 (1)	119 (1)
D1	37 (0)	33 (1)	1252 (0)	1290 (0)	233 (1)
D2	2940 (0)	21 (1)	2339 (0)	1413 (1)	7 (1)
D3	363 (0)	5626 (0)	3779 (0)	1137 (1)	98 (1)
NP1	1341 (0)	41 (1)	7754 (0)	2975 (1)	13690 (1)
NP2	9999 (0)	289 (1)	182 (1)	5791 (2)	1495 (2)
EP1	411 (0)	4069	6593 (0)	1155 (1)	187 (1)
EP2	1308 (0)	120 (1)	2042 (0)	802 (1)	130 (1)
EP3	411 (0)	110 (1)	6599 (0)	1156 (1)	188 (1)
R1	2600 (0)	546	2907	399 (1)	15 (1)
R2	4845 (0)	254 (1)	6422 (0)	399 (1)	15 (1)
R3	4269 (0)	80 (1)	6989	1113 (1)	11908 (0)
R4	8658 (0)	81 (1)	29 (1)	745 (1)	906 (1)
S0	581 (0)	135 (1)	3001 (0)	10530	113 (1)
S1	1221 (0)	19 (1)	4763 (0)	132 (1)	105 (1)
S2	986 (0)	68 (1)	32 (1)	194 (1)	204 (1)
S3	48 (0)	146 (1)	943 (0)	502 (1)	26 (1)
K1	38 (0)	61 (1)	1262 (0)	1305 (0)	10206
K2	1436	8335 (0)	377 (0)	6034	7517
K3	3306 (0)	205 (1)	755 (0)	474 (1)	84 (1)
K4	1888 (0)	78 (1)	608 (0)	2831	1224

TABELLE 2. Market-Split Probleme: M=10, N=90, Laufzeit: 240m

Run	MSP11-1	MSP11-2	MSP11-3	MSP11-4	MSP11-5
CPLEX	18000 (30)	18000 (21)	18000 (35)	18000 (30)	18000 (29)
H0	982 (1)	259 (0)	799 (0)	503 (1)	4198 (0)
H1	2975 (0)	819 (0)	1027 (0)	864 (1)	5185 (0)
H2	961 (0)	97 (0)	1720 (0)	80 (1)	4058 (0)
H3	1707 (0)	1337 (0)	39 (0)	335 (1)	3359 (0)
D0	266 (1)	514 (0)	590 (0)	12520	2107 (0)
D1	1221 (1)	195 (0)	31 (0)	4682 (0)	127 (0)
D2	277 (1)	1194 (0)	689 (0)	6514	1878 (0)
D3	5954 (0)	484 (0)	224 (0)	6246	885 (0)
NP1	1154 (2)	7793 (2)	7982 (0)	1865 (0)	1520 (1)
NP2	9398 (2)	2549 (3)	410 (2)	15474 (0)	11931 (1)
EP1	562 (1)	1067 (0)	11644 (0)	13233 (0)	4563 (0)
EP2	490 (1)	6180	878 (0)	2628 (0)	1619 (0)
EP3	3169 (1)	1068 (0)	11656 (0)	13240 (0)	4562 (0)
R1	221 (1)	7395 (0)	576 (0)	5597 (1)	1411 (0)
R2	230 (1)	1765 (0)	1594 (0)	1153 (1)	4395 (0)
R3	2107 (0)	2431 (0)	139 (0)	391 (2)	16917 (0)
R4	7501 (0)	4764 (0)	139 (0)	392 (2)	8446 (0)
S0	774 (1)	256 (0)	800 (0)	505 (1)	4173 (0)
S1	6245 (0)	163 (0)	1026 (0)	866 (1)	5182 (0)
S2	999 (0)	288 (0)	1744 (0)	3138 (1)	4131 (0)
S3	784 (0)	1339 (0)	676 (0)	336 (1)	3359 (0)
K1	1215 (1)	193 (0)	32 (0)	5232	126 (0)
K2	7849	5890 (0)	112 (0)	10680	3806 (0)
K3	819 (1)	290 (0)	383 (0)	1963 (1)	772 (0)
K4	2457 (0)	7000 (0)	104 (0)	9611	655 (0)

TABELLE 3. Market-Split Probleme: M=11, N=100, Laufzeit: 300m

Problem	MSH7-0	MSH7-1	MSH7-2	MSH7-3	MSH7-4
CPLEX	3600 (5)	3600 (7)	3600 (9)	3600 (5)	3600 (6)
H0	102	43	79	100	651
H1	92 (0)	29	84	21	97
H2	16 (0)	81	27	57	235 (0)
H3	14 (0)	21	26	14 (0)	107
D0	53	83	25	26	549
D1	59 (0)	84	26	95	404
D2	36	81	28	21	150
D3	36	41	7	40	42
NP1	55	6	89 (1)	408	6 (1)
NP2	377	12	768 (1)	2403 (1)	18 (1)
EP1	253	284	221	46	703
EP2	4	61	8	8	215 (0)
EP3	69	12	14	80	231 (0)
R1	65	70	136	62	1419 (1)
R2	128	167	147	62	477 (1)
R3	8	172	17	27	633
R4	8	165	17	28	631
S0	106	42	73	48	663
S1	93	28	84	21	89
S2	16 (0)	9 (2)	611 (1)	25 (1)	419 (1)
S3	9	5	6	2	26
K1	19	39	13	49	210
K2	9	16	7	4	14
K3	18 (0)	7	204	6 (0)	493
K4	10	33	5	14	121

TABELLE 4. Market-Sharing Probleme: M=7, N=60, Laufzeit: 60m

Problem	MSH8-0	MSH8-1	MSH8-2	MSH8-3	MSH8-4
CPLEX	7200 (7)	7200 (10)	7200 (10)	7200 (9)	7200 (8)
H0	138	1186 (2)	282 (1)	13 (2)	638 (1)
H1	63	133 (1)	914 (1)	3240 (1)	223 (1)
H2	31	335 (1)	337 (1)	12 (2)	142 (1)
H3	48	5063 (1)	348 (1)	1320 (1)	551 (1)
D0	184	67 (1)	1135 (1)	926 (2)	8 (1)
D1	91	430 (1)	835 (1)	167 (2)	215 (1)
D2	106	383 (1)	1641 (1)	3384 (1)	928 (1)
D3	40	122 (1)	170 (1)	14 (2)	159 (1)
NP1	755	95 (1)	1325 (1)	36 (2)	1116 (1)
NP2	6274 (1)	1375 (1)	780 (4)	258 (2)	3215 (2)
EP1	18	1348 (1)	864 (1)	1038 (1)	68 (1)
EP2	73	916 (1)	409 (1)	2902 (1)	16 (1)
EP3	9	29 (1)	1216 (1)	119 (2)	52 (1)
R1	414	486 (1)	1445 (1)	2382 (1)	229 (1)
R2	389	611 (1)	3256 (1)	125 (2)	541 (1)
R3	46	440 (1)	410 (1)	45 (2)	412 (1)
R4	46	437 (1)	407 (1)	56 (2)	421 (1)
S0	136	1175 (2)	381 (1)	12 (2)	635 (1)
S1	63	130 (1)	920 (1)	3273 (1)	226 (1)
S2	48 (0)	865 (1)	1995 (1)	9 (2)	199 (1)
S3	15	2672 (1)	102 (1)	425 (1)	227 (1)
K1	31 (0)	7 (1)	520 (1)	111 (2)	345 (1)
K2	61	329 (1)	684 (1)	195 (2)	259 (1)
K3	42	617 (1)	2404 (1)	4443 (1)	67 (1)
K4	22	543 (1)	1469 (1)	7 (2)	35 (1)

TABELLE 5. Market-Sharing Probleme: M=8, N=70, Laufzeit: 120m

Problem	MSH9-0	MSH9-1	MSH9-2	MSH9-3	MSH9-4
CPLEX	14400 (20)	14400 (16)	14400 (15)	14400 (20)	14400 (16)
H0	2365 (4)	2550 (7)	7989 (6)	1962 (3)	1453 (4)
H1	259 (4)	5864 (7)	7002 (4)	2776 (2)	1591 (2)
H2	111 (4)	12854 (7)	2547 (4)	1407 (2)	256 (5)
H3	458 (4)	0 (18011)	1435 (6)	2407 (2)	4817 (5)
D0	2175 (4)	1637 (8)	4986 (7)	148 (3)	6537 (2)
D1	169 (4)	0 (18011)	3061 (3)	1954 (2)	1906 (5)
D2	117 (4)	1775 (7)	10533 (3)	665 (3)	10515 (2)
D3	6308 (4)	10650 (8)	8918 (3)	2378 (2)	14313 (2)
NP1	10301 (4)	0 (18011)	15 (7)	5898 (2)	2258 (5)
NP2	12421 (6)	0 (18011)	79 (7)	6091 (7)	0 (17401)
EP1	65 (4)	2053 (7)	626 (5)	168 (2)	341 (5)
EP2	2352 (4)	11446 (7)	656 (3)	337 (3)	1656 (2)
EP3	6228 (4)	6579 (7)	38 (3)	169 (3)	8125 (4)
R1	7427 (4)	11897 (8)	5551 (6)	1394 (2)	3692 (5)
R2	6605 (7)	0 (18011)	5683 (3)	1324 (2)	10831 (5)
R3	1125 (4)	5194 (7)	3160 (5)	9322 (2)	13435 (4)
R4	1111 (4)	5126 (7)	3161 (5)	536 (3)	13411 (4)
S0	2404 (4)	2579 (7)	8199 (5)	1953 (3)	1533 (4)
S1	254 (4)	5812 (8)	7096 (4)	2737 (2)	1546 (2)
S2	99 (4)	519 (8)	899 (9)	2178 (2)	326 (5)
S3	112 (4)	0 (18011)	1401 (6)	2553 (2)	4716 (5)
K1	817 (4)	7918 (6)	14060 (3)	3637 (2)	278 (4)
K2	13 (4)	14189 (6)	13334 (5)	55 (1)	4813 (2)
K3	1168 (4)	725 (5)	5599 (3)	10563 (2)	11444 (4)
K4	26 (4)	5027 (6)	1921 (3)	2058 (2)	74 (5)

TABELLE 6. Market-Sharing Probleme: M=9, N=80, Laufzeit: 240m

Problem	MSH10-0	MSH10-1	MSH10-2	MSH10-3	MSH10-4
CPLEX	18000 (21)	18000 (26)	18000 (26)	18000 (19)	18000 (26)
H0	12278 (7)	0 (22597)	0 (22104)	664 (4)	1799 (11)
H1	0 (23283)	5185 (7)	0 (22104)	5748 (4)	0 (22154)
H2	10689 (9)	6200 (7)	0 (22104)	12851 (4)	6690 (8)
H3	5056 (7)	5394 (8)	3922 (9)	896 (4)	0 (22154)
D0	0 (23283)	0 (22597)	0 (22104)	58 (5)	0 (22154)
D1	9105 (7)	3804 (8)	8541 (7)	5484 (4)	0 (22154)
D2	12482 (11)	2337 (8)	12706 (6)	15162 (3)	0 (22154)
D3	0 (23283)	0 (22597)	17812 (8)	176 (4)	0 (22154)
NP1	0 (23283)	3327 (8)	0 (22104)	11435 (6)	0 (22154)
NP2	1 (23283)	1 (22597)	1 (22104)	0 (22428)	0 (22154)
EP1	3777 (8)	0 (22597)	612 (7)	7537 (4)	12592 (11)
EP2	0 (23283)	0 (22597)	0 (22104)	2311 (4)	448 (9)
EP3	0 (23283)	0 (22597)	0 (22104)	4452 (4)	0 (22154)
R1	5920 (9)	1560 (8)	0 (22104)	9120 (4)	0 (22154)
R2	0 (23283)	7547 (8)	0 (22104)	5889 (4)	10205 (8)
R3	14053 (9)	2226 (7)	15191 (11)	10410 (3)	0 (22154)
R4	14034 (9)	2315 (7)	15110 (11)	10374 (3)	0 (22154)
S0	12421 (7)	0 (22597)	0 (22104)	682 (4)	1885 (11)
S1	0 (23283)	5120 (7)	0 (22104)	5716 (4)	0 (22154)
S2	0 (23283)	0 (22597)	0 (22104)	1558 (7)	0 (22154)
S3	5139 (7)	5347 (8)	3962 (9)	892 (4)	0 (22154)
K1	8680 (9)	5217 (6)	1 (22104)	1486 (4)	0 (22154)
K2	7749 (6)	5766 (6)	9192 (7)	391 (4)	0 (22154)
K3	13749 (7)	0 (22597)	8731 (6)	7428 (3)	596 (8)
K4	0 (23283)	13702 (6)	1332 (9)	9596 (5)	0 (22154)

TABELLE 7. Market-Sharing Probleme: M=10, N=90, Laufzeit: 300m

Innerhalb jeder Tabellenspalte wird mit den Farben blau und rot markiert, welcher Parametersatz in Gegenüberstellung mit den restlichen Parametersätzen das schlechteste (blau) bzw. beste (rot) Ergebnis liefert.

Die summierten Anzahlen der Erfolge bzw. Mißerfolge über alle Tabellen ergibt schließlich das folgende Bild:

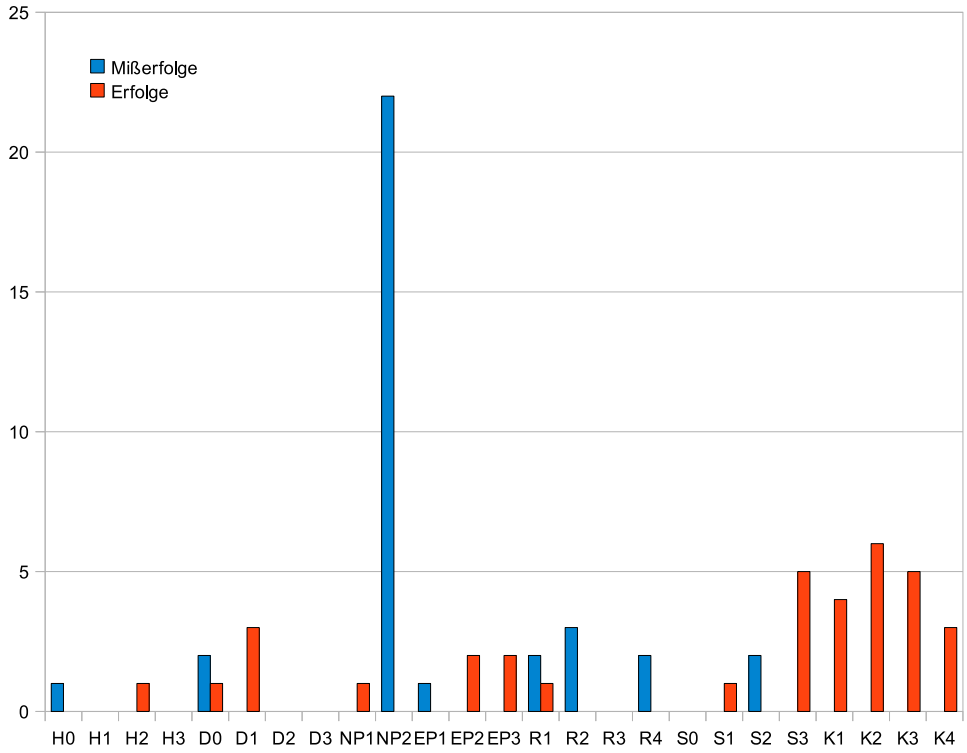


Abbildung 6.0.1. Auswertung der Benchmarks

Fazit

Aus den Testläufen lassen sich unterschiedliche Schlussfolgerungen ziehen:

Zum einen ist festzustellen, dass das heuristische Sieb-Verfahren in Kombination mit einer dynamischen Füllstrategie, die vielversprechendste Technik darstellt. Dynamische Füllstrategie heißt, dass die zu siebende Menge S nur mit den Vektoren gefüllt wird, die während der Abarbeitung der anderen Algorithmen abfallen. Dies wird erreicht durch Setzen des Kommandozeilenparameters „–seed=0“. In der Theorie hängt der Erfolg des Sieb-Algorithmus zwar von der Größe von $|S|$ ab, allerdings scheint ein künstliches Aufblähen der zu siebenden Menge durch den Babai'schen Rundungsalgorithmus (zur Erzeugung zusätzlicher, gleichverteilter Gittervektoren) nicht unbedingt der richtige Ansatz zu sein. Dies wurde im Vorfeld (durch leider undokumentierte Tests) bestätigt. Bei der in den Parametersätzen S0 - S3 verwendeten dynamischen Recycling-Strategie ergibt sich allerdings ein anderes Bild. Insbesondere in Verbindung mit der dritten heuristischen Bewertungsfunktion resultieren hieraus die besten Ergebnisse, was durch die Resultate der Parametersätze S3 und K3 belegt wird.

Das in Kapitel (2.4) geschilderte Konzept den Enumerationsbaum an geeigneten Stellen abzuschneiden (NP1 / 2) scheint nur dann von Erfolg gekrönt zu sein, wenn die zugrundeliegende BKZ-Reduktion mit voller Blockweite durchgeführt wird. Bei den hier durchgeführten Laufzeitvergleichen, mit geringen Blockweiten, konnte jedensfalls kein erwähnenswerter Erfolg verzeichnet werden. Dies scheint damit erklärbar, dass diese Methode durchaus einiges an Rechenoverhead erfordert, der bei kleinen Blockweiten ($\beta \leq 40$) zu stark zum Tragen kommt.

Ganz anders sieht das Bild jedoch beim Extreme Pruning aus. Hier halten sich die zusätzlichen Berechnungen in vertretbaren Grenzen, womit die Zeitersparnis durch das Abschneiden, wesentlich besser zum Tragen kommt. Es konnten hierbei besonders die Parametersätze EP2 und EP3 überzeugen.

Die Auswirkungen von Tiefeneinfügungen (D0 - D3) machen sich ebenfalls positiv bemerkbar. Besonders hervorzuheben sind die Parametersätze D1 und K1. Unter anderem ließ sich feststellen, dass zwar die Anzahl der Iterationen, die während einer BKZ-Reduktion durchgeführt werden, abnehmen, wenn bei gleichbleibender Blocklänge β der Wert des Parameters *deep* vergrößert wird, sich aber die Laufzeit der durchgeführten LLL-Reduktionen erhöht. Die Erklärung hierfür ist allerdings recht einfach. Der *deep* Reduktionsschritt ist innerhalb des LLL-Verfahrens gekapselt und vergrößert damit

dessen Laufzeit und da der Reduktionsschritt in gewisser Weise die Aufgaben des BKZ-Verfahrens übernimmt ist es eigentlich nur natürlich, das die Anzahl der Iterationen abnimmt.

Die Benchmarks zur Paarweisen Reduktion (H0 - H3) lassen Rückschlüsse zu, welche heuristische Bewertungsfunktion die besseren Resultate liefert, wenn kein Sieb verwendet wird. Es zeigt sich, dass die Default-Heuristik (H0) nicht unbedingt die beste Wahl ist - die zweite heuristische Bewertungsfunktion (H2) liefert auf lange Sicht die besseren Ergebnisse.

Die Random Sampling Testläufe bleiben leider eher unscheinbar - was wohl auch an der Sampling-Rate von 20000 gelegen haben mag. Es ist allerdings ersichtlich, dass die allgemeingültige Strategie von Buchmann (R3 / R4) der Schnorr'schen (R1 / R2) leicht überlegen ist.

Zusammenfassend kann man feststellen, das die in dem Programm implementierten Methoden, in ihrer Kombination und mit Hinblick auf die Qualität der erreichten Gitter-Reduktion für die Market-Share/Split - Probleme hervorragende Ergebnisse liefern und für die schwierigeren Challenge-Probleme, die in Dimensionen > 500 angesiedelt sind, die Resultate durchaus noch zufriedenstellend sind. Die Höherdimensionalität im Vergleich zu den Market-Sharing Problemen (< 120) ist allerdings spürbar, da wesentlich längere Laufzeiten (> 7 Tage) nötig waren, bis genügend kurze Vektoren gefunden werden konnten.

Beim Vergleich mit CPLEX, das einen völlig anderen Ansatz zur Lösung von ganzzahlig-linearen Gleichungsysstemen verwendet (Branch & Bound), konnten sogar sehr gute Ergebnisse erzielt werden. In nahezu allen Testreihen waren die von RASA gefundenen Lösungen qualitativ besser, als beim kommerziellen Produkt. Es konnten insbesondere einige Probleme optimal gelöst werden, für die CPLEX (sogar bei unbeschränkter Laufzeit) fälschlicherweise „Optimallösungen“ mit $||Ax - d||_1 > 0$ lieferte. Leider wird in der verwendeten Interactive Version von CPLEX der Zeitpunkt der Lösungsfindung nicht vermerkt. Es konnte deshalb in den entsprechenden Tabellenzeilen keine Angabe gemacht werden, wann genau innerhalb des vorgegebenen Zeitrahmens die Lösung gefunden wurde.

Danksagung

Abschließend möchte ich mich bei Herrn PD Dr. Alfred Wassermann herzlich für die beständige und umfangreiche Unterstützung bedanken. Insbesondere gilt mein Dank Herrn Prof. Dr. Baptist, dem Team des Lehrstuhls für Mathematik und ihrer Didaktik, meinen Freunden und meiner Familie, die mich mit ihrer aufmunternden Art während der gesamten Promotionszeit tatkräftig unterstützt haben und mir mit zahlreichen Ablenkungen, Aufheiterungen und Korrekturen zur Seite standen.

Anhang

Lösungsstrategien: FP und IP

Zur Lösung von ganzzahlig-linearen Gleichungssystemen wird normalerweise der Feasability-Ansatz (FP) verwendet:

Man sucht zu vorgegebenem $A = (a_{ij}) \in \mathbb{Z}^{h \times k}, d \in \mathbb{Z}^h$ und $r \in \mathbb{Z}^k$ einen Vektor $x \in \mathbb{Z}^k$, so dass $A \cdot x = d$ und $0 \leq x \leq r$.

Was tut man aber, wenn keine solche Lösung existiert? In diesen Fällen wird das Problem durch die Einführung von Schlupfvariablen zunächst als allgemeines IP-Problem reformuliert:

Finde einen Vektor $x \in \mathbb{Z}^k$ und Schlupfvektoren $s^{(1)}, s^{(2)} \in \mathbb{Z}^h$ mit $s_j^{(i)} \geq 0$ für $i = 1, 2$ und $j = 1, 2, ..., h$, so dass $A \cdot x + s^{(1)} - s^{(2)} = d$, $0 \leq x \leq r$ und $\sum_{j=1}^{h}(s_j^{(1)} + s_j^{(2)})$ minimal ist.

Aufgrund dieser leicht veränderten Betrachtungsweise besitzt das System jetzt immer eine Lösung. Allerdings lässt es sich wegen der zusätzlichen Minimierungsbedingung nicht mehr ohne weiteres ins *prb*-Format bringen.

Es wären jetzt mehrere Wege möglich, dieser Sachlage zu begegnen:

Man könnte die Minimierungsbedingung als zusätzliche Ungleichung schreiben. Hierzu müssten lediglich die untere und obere Grenze passend gewählt werden - was allerdings recht statisch wäre. Auch ist der Umstand, mit zusätzlichen Variablen für den Schlupf zu arbeiten, nicht gerade wünschenswert, da sich dadurch insbesondere die Problem-Dimension vergrößert.

Es gibt aber einen eleganten Ausweg. Man fasst das ursprüngliche Gleichungsystem, als System von Ungleichungen auf, mit gleichgewählter unterer und oberer Grenze:

Finde einen Vektor $x \in \mathbb{Z}^k$ so dass $d \leq A \cdot x \leq d$ und $0 \leq x \leq r$.

Damit ändert sich zwar an der theoretischen Formulierung des Problems überhaupt nichts, entscheidend wird aber sein, das dieses System eine veränderte *prb*-Datei zur Folge hat.

Hierzu ein kurzes Beispiel:

Es soll das lineare Gleichungssystem $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \cdot x = \begin{pmatrix} 10 \\ 20 \\ 30 \end{pmatrix}$

mit $(0, 0, 0)^T \leq x \leq (40, 50, 60)$ gelöst werden.

Die resultierende prb-Datei für die FP-Version besitzt den Inhalt:

beispiel.prb

```
[
]

[
[ 1 2 3 ]
[ 4 5 6 ]
[ 7 8 9 ]
]

[ ]

[ ]

[ 10 20 30 ]

[ 40 50 60 ]

1
```

Um das System als IP-Problem aufzufassen werden die Gleichungen als Ungleichungen geschrieben. Hieraus resultiert die folgende Datei:

beispiel_slk.prb

```
[
[ 1 2 3 ]
[ 4 5 6 ]
[ 7 8 9 ]
]

[
]

[ 10 20 30 ]

[ 10 20 30 ]

[ ]

[ 40 50 60 ]

1
```

Nach der Konvertierung ins lat-Format (siehe Kapitel 1.4.1), kann mithilfe des Kommandozeilenparameters *slk* von RASA nach Lösungen mit Schlupf

gesucht werden, was eben nur dann möglich ist, wenn das zugrundeliegende System Ungleichungen besitzt.

Sobald ein Gittervektor gefunden wird, für dessen Lösungsanteil x gilt:

$$||A \cdot x - d||_1 \leq slk$$

wird abgebrochen (siehe Kapitel 5.4).

Konstruktion von $q-(n,k,d)$ Codes

$GF(q)$ bezeichne einen endlichen Körper mit Primzahlpotenzordnung $q = p^a$.

Es wird eine Generatormatrix $G \in GF(q)^{k\times n}$ gesucht, die einen $q-(n,k)$ Code mit Minimaldistanz $\geq d$ erzeugt.

In $GF(q)^k$ existieren $s := \frac{q^k-1}{q-1}$ verschiedene eindimensionale Unterräume. Demnach gibt es also auch s mögliche Spalten für G, die mit $x_1, x_2, ..., x_s$ bezeichnet seien.

Es sei nun eine $s \times s$ Matrix M definiert mit

$$m_{i,j} := \begin{cases} 0, \text{ falls } \langle x_i, x_j\rangle = 0 \bmod q \\ 1, \text{ falls } \langle x_i, x_j\rangle \neq 0 \bmod q \end{cases}$$

Damit reduziert sich die Suche nach einer passenden Generatormatrix auf das Problem ein $y \in \{0,1\}^s$ zu finden, mit:

$$M \cdot y \geq \begin{pmatrix} d \\ d \\ \vdots \\ d \end{pmatrix} \text{ und } \sum_{i=1}^{s} y_i = n$$

Insbesondere gilt dann außerdem: $M \cdot y \leq (n, n, ..., n)^T$.

Man kann das Gleichungssystem auch anders schreiben, durch Definition einer Matrix $\widetilde{M}$ mit

$$\widetilde{m}_{i,j} := \begin{cases} 1, \text{ falls } \langle x_i, x_j\rangle = 0 \bmod q \\ 0, \text{ falls } \langle x_i, x_j\rangle \neq 0 \bmod q \end{cases}$$

Jetzt reduziert sich die Suche nach einer passenden Generatormatrix auf das Auffinden eines $y \in \{0,1\}^s$ mit:

$$\widetilde{M} \cdot y \leq \begin{pmatrix} n-d \\ n-d \\ \vdots \\ n-d \end{pmatrix} \text{ und } \sum_{i=1}^{s} y_i = n$$

Da die Dimension s normalerweise recht groß ist, versucht man diese mit einer gruppentheoretischen Überlegung zu verringern:

Es sei H eine Gruppe, die auf $GF(q)^k$ operiert. $GF(q)^k$ zerfällt damit in disjunkte Bahnen, d.h. $GF(q)^k = \sigma_1 \dot\cup \sigma_2 \dot\cup ... \dot\cup \sigma_l$. Insbesondere landen auch die $x_1, x_2, ..., x_s$ in (unterschiedlichen) Bahnen. Ohne Einschränkung seien $x_1, x_2, ..., x_s$ derart durchnummeriert, dass $x_1, x_2, ..., x_l$ ein Repräsentantensystem der Orbits $\sigma_1, \sigma_2, ..., \sigma_l$ bilden und für $i, j = 1, 2, ..., l$ sei definiert:

$$\overline{m}_{i,j} := \sum_{x_k \in \sigma_j} \widetilde{m}_{i,k}$$

Damit vereinfacht sich die Suche dahingehend, dass jetzt nach einem $y \in \{0,1\}^l$ gesucht wird, für das gilt:

$$\overline{M} \cdot y \leq \begin{pmatrix} n-d \\ n-d \\ \vdots \\ n-d \end{pmatrix} \text{ und } \sum_{i=1}^{l} |\sigma_i| \cdot y_i = n$$

Die gesuchte Generatormatrix G resultiert nun daraus, dass man für alle y_i mit $y_i = 1$ die $x_j \in \sigma_i$ als Spalten von G verwendet.

BEMERKUNG 124. Für die Operation werden normalerweise Gruppen verwendet, die von invertierbaren Matrizen erzeugt werden, etwa: $\mathrm{PGL}(k,q)$.

Lösungen der Benchmarks

Problem	Lösungsvektor(en)	$\|\|Ax - d\|\|_1$
MSP9_1	[0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 0 0 0 1 1 0 0 1 1 1 0 0 0 1 1 0 1 0 1 0 1 0 1 0 0 0 1 1 1 0 1 1 0 1 1 1 1]	0
MSP9_2	[0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 0 0 1 0 0 1 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 1 0]	0
MSP9_3	[1 0 1 0 0 0 1 1 1 0 0 0 1 1 0 0 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 1 0 1 1 0 1 1 1 0 1 0 0 1]	0
MSP9_4	[0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 0 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 1 1 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1]	0
MSP9_5	[0 0 0 1 0 1 1 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 1 0 1 1 1 0 1 1 1 1 0 0 1 1 0 1 0 0 1 1 1 1 1 0 0 1 1 0 0 0 1 1 0 0 0 1 0 1 0 1 1 1 0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 1]	0

Problem	Lösungsvektor(en)	$\|\|Ax - d\|\|_1$
MSP10_1	[0 1 0 1 1 0 1 1 1 0 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 1 1]	0
MSP10_2	[0 0 0 1 1 0 0 1 1 1 1 1 0 1 0 1 0 0 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 1 1 1 0 0 0 1 1 1 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 1 1 1 1 0 0 1 0 1 0 0 0 1 1]	0
MSP10_3	[0 1 0 0 0 0 0 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 0 0 1 0 1 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 0 0 1 1 0 1]	0
MSP10_4	[0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 1 0 1 0 1 0 1 1 0 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 0 1]	0
MSP10_5	[0 0 0 0 1 1 0 1 0 1 1 1 1 0 0 0 0 1 0 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0]	0

Problem	Lösungsvektor(en)	$\|\|Ax - d\|\|_1$
MSP11_1	[0 0 0 0 0 1 1 0 0 1 0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 1 1 1 1 0 1 1 0 0 1]	0
MSP11_2	[0 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 0 0 1 0 0 1 0 1 1 0 1 0 1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 1]	0
MSP11_3	[0 0 0 1 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 0 1 0 0 0 0 0 1 1 1 1 1 1 0 1 1 0 0 1 1 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 0 0 1]	0
MSP11_4	[0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 1 1 0 0 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 0 0 0 1 1 1 1 0 1 0 0]	0
MSP11_5	[0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1 1 0 0 1 0 1 0 0 1 1 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 0 1 0 1 1 0 1 0 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1 1 1 0 1 1 0 1 1 1 1 0 1 0 1 0 0 0 0 1 1 1 1 1 1 1 1 0 1 0 1 0 0 1 0 1 0 1]	0

Problem	Lösungsvektor(en)	$\|\|Ax - d\|\|_1$
MSH7-0	[1 1 0 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 0 0 1 1 1 0 0 1 1 0 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 0 1 0 1 0 1 1 0 1 0 1 0 1 1 1 0 0]	0
MSH7-1	[1 0 1 0 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 1 0 0 0 0 1 1 0 1 1 1 0 0 0 1 1 1 1 1 0 1 1 1 1 0]	0
MSH7-2	[0 0 1 1 1 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 0 0 1 1 0 1 0 1 1 1 1 1 0 0 0 1 0 0 0 1 0 0 1 1 1 1 0 0]	0
MSH7-3	[1 0 1 0 1 1 0 0 0 1 0 1 1 1 0 0 1 1 0 0 1 1 1 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1]	0
MSH7-4	[1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 0 1 0 0 1 1 1 1 1 0 1 1 1 0 1 0 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 1 0 0 0 1 0]	0

Problem	Lösungsvektor(en)	$\|\|Ax - d\|\|_1$
MSH8-0	[0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 0 1 1 0 1 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 1 1 0 1 1 1 1 1 0]	0
MSH8-1	[1 1 1 0 0 1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0 1 0 0 0 1 0 1 1 0 0 1 1 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0]	1
MSH8-2	[0 0 1 1 1 0 0 0 0 0 1 0 0 1 1 0 0 1 1 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0 1 0 1 1 1 0 1 1 1 0 1 1 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 1 0 1 1 0 0 0 0 0 0 1]	1
MSH8-3	[0 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 1 0 0 1 0 0 0 1 0 1 1 1 1 1 0 1 1 0 1 1 0 1 0 0 0 0 0 1 0 0 1 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0]	1
MSH8-4	[0 0 0 1 0 1 0 0 1 0 1 1 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 1 0 0 1 0 1 1 1 0]	1

Problem	Lösungsvektor(en)	$\|\|Ax - d\|\|_1$
MSH9-0	[1 0 1 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 0 0 0 1 0 0 1 0 0 1 0 1 1 1 0 0 1 0 1]	4
MSH9-1	[0 0 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1 1 1 0 1 0 0 1 0 1 0 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 0]	7
MSH9-2	[0 1 0 1 0 1 1 1 0 1 1 0 1 0 1 1 0 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 1 0 1 0 1 0 1 1 0 1 0 1 1 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0 0 0 1 1 0 0 1 0 1]	3
MSH9-3	[1 1 1 1 0 0 1 1 0 0 1 0 0 1 1 1 0 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 1 0 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0 1 1 0 0]	2
MSH9-4	[1 1 1 1 0 0 0 0 0 1 1 1 0 1 0 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1 0 1 1 1 0 0 1 0 0 0 0 0 1 0 1 1 1 1 1 0 1 0 0 1 0 0 0 0 0 0 1 1 0 0 1 1 0 1 1 0 1 1 0 0 0 1 1 0 0 0]	2

Problem	Lösungsvektor(en)	$\|\|Ax - d\|\|_1$
MSH10-0	[1 1 0 1 0 1 1 1 0 0 1 1 0 1 0 1 0 0 1 1 1 1 1 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 0 1 1 0 1 1 1 1 0 1 0]	7
MSH10-1	[0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1 1 0 1 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 0 0 0 1 0]	7
MSH10-2	[0 0 1 0 0 1 1 0 1 1 0 0 0 0 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 1 1 0 1 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 1 1 1 0]	6
MSH10-3	[0 1 1 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 1 1 1 0 1]	3
MSH10-4	[0 0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1 1 1 1 0 0 0 1 1 0 1 1 0 0 1 0 1 1 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 0 0 0 0 1 1 0 0 0 1 1 0 1 1 0 0 1]	8

Problem	Lösungsvektor(en)	$\|\|x\|\|_2$
ch-575 (K3)	[-3 0 3 -4 2 6 -3 -1 1 -2 1 1 7 4 -8 -4 -3 -2 -1 2 -2 -1 0 3 -5 -4 0 3 -7 -8 -3 -4 7 -2 -2 3 0 3 0 3 2 5 2 -8 7 2 7 -1 -1 3 1 -4 3 4 0 -5 12 3 2 -3 3 3 3 3 -1 1 1 -1 -2 -2 6 -3 -1 3 -4 6 3 -2 3 2 1 6 -7 0 -1 -7 -1 1 -2 0 5 2 1 -3 1 -1 4 -1 8 6 -2 2 0 -1 -3 -7 -4 6 -1 -4 3 0 -4 4 -3 4 -5 5 0 2 3 0 -5 -2 2 0 -6 -3 -3 -1 -5 0 -1 0 0 0 0 0 0 0 4 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 7 0 -7 0 0 0 0 0 0 -3 6 -2 1 3 -6 2 6 -1 -3 -3 -7 -1 7 -1 0 6 -6 -12 1 5 7 -4 -2 6 -6 2 -6 0 1 -3 2 4 5 -3 6 7 4 3 -3 2 0 1 0 11 1 0 -4 2 1 -1 7 0 3 -2 -3 -2 1 -7 -1 -2 -3 1 -1 -2 -2 4 5 2 0 -5 4 1 -7 -2 9 -1 1 5]	58.5
ch-550 (K3)	[3 4 0 3 3 3 -2 -3 3 -7 1 0 0 4 1 -2 2 0 3 -3 -1 2 3 8 -1 -2 -4 7 -2 -4 3 -1 0 0 -5 2 -4 5 5 -2 -3 -3 0 1 -1 -3 2 -1 0 -7 -4 4 -9 0 -1 0 -1 2 3 5 -2 -7 -1 6 1 1 3 -5 -1 6 5 3 -4 5 3 -2 4 1 2 -1 3 2 6 2 -3 -3 0 1 -3 0 -3 1 0 -1 -6 0 -2 -3 4 -6 -3 2 2 0 -1 2 7 4 6 -2 0 0 3 2 -1 7 -6 -5 0 1 4 3 4 0 0 0 0 1 0 0 0 0 -3 4 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -2 0 6 -2 0 -4 2 3 -1 -2 3 -1 -6 1 3 -1 6 -3 1 2 4 4 -3 -5 -1 -3 1 -2 -3 5 -2 1 1 2 3 -3 0 1 2 5 5 2 0 3 -2 -1 2 -1 1 -4 -6 0 0 2 -11 1 -5 -3 0 5 4 -2 2 -2 1 -1 -4 0 -2 1 -3 -7 0 5 -3 0 -1]	47.7

Problem	Lösungsvektor(en)	$\|\|x\|\|_2$
ch-525 (K3)	[0 -1 4 -2 1 4 0 0 -1 -3 -4 -2 -3 4 4 -4 3 -2 -2 3 0 4 2 1 -2 2 -2 -1 2 0 -2 0 3 4 0 9 -2 -1 1 2 0 1 -3 -3 1 0 -1 8 1 1 2 3 5 2 -1 1 -5 0 -2 1 -6 -1 1 0 -3 -2 0 4 -2 0 5 -2 -1 -5 1 6 0 1 -3 -6 -1 -1 0 5 4 2 -3 2 0 1 -2 -4 6 -2 -1 1 -5 -4 2 -4 1 0 2 2 -1 -2 6 1 5 -1 -5 0 0 0 3 0 3 0 0 -2 3 1 0 4 0 4 0 0 0 0 4 0 6 0 2 1 4 3 -1 -5 1 -6 2 2 -5 -1 1 1 2 1 0 -2 -3 3 -1 2 -1 -3 4 -4 6 -5 2 4 -1 -1 3 0 -2 0 4 0 0 0 3 -1 -6 0 2 4 2 -4 0 -3 1 -5 -3 2 -2 -2 2 -1 -3 -3 0 3 -4 0 -2 -1 -1 2 2]	40
ch-500 (K3)	[-2 -3 -1 2 3 -2 -2 -1 0 -4 1 5 2 -1 0 -1 -2 -1 1 -2 -1 -2 0 1 0 3 5 1 -5 2 -1 -1 0 -3 2 1 2 0 -2 4 0 2 0 4 2 -1 -1 1 -3 -3 -2 -2 0 1 3 1 -2 0 0 1 2 -4 3 0 2 2 3 0 -4 0 1 0 1 5 -2 2 0 2 1 3 3 0 0 1 4 3 2 4 -3 -1 5 -1 2 3 4 -1 0 2 -2 -1 0 3 1 0 -4 0 0 0 2 2 -5 0 0 0 1 3 0 0 0 -2 0 0 0 0 0 0 0 0 0 -4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -3 0 2 1 0 -1 -1 1 -3 -2 1 5 0 0 3 3 -1 1 2 -1 -1 0 0 -2 -2 -4 -2 0 2 -1 -1 1 -3 2 0 2 -2 0 1 1 3 0 -1 0 0 -4 2 0 -3 -1 -2 -5 4 0 -1 -1 -2 0 3 4 0 -2 -1 0 -1]	29.4

Literaturverzeichnis

[A98] K. Aardal, C. Hurkens, A. K. Lenstra: „Solving a system of diophantine equations with lower and upper bounds on the variables“, Integer Programming and Combinatorial Optimization 6, Springer Lecture Notes in Computer Science 1412, 1998

[Aj98] M. Ajtai: „The Shortest Vector Problem in L_2 is NP-hard for Randomized Reductions“, Proceedings of the 30th Annual ACM Symposium on Theory of Computing, ACM Press, 1998

[A01] M. Ajtaj, R. Kumar, D. Sivakumar: „A sieve algorithm for the shortest lattice vector problem“, Proceedings of 47th Annual ACM Symposium on Theory of Computing, ACM Press, 2001

[B09] Faculty of Mathematics, Physics and Applied Computer Science: Coding Theory – Best Linear Codes, `http://www.algorithm.uni-bayreuth.de/en/research/Coding_Theory/Linear_Codes_BKW/index.html`

[C96] H. Cohen: „A Course in Computational Algebraic Number Theory“, Springer, 1996

[C00] J-Y. Cai: „The Complexity of Some Lattice Problems“, Algorithmic Number Theory 4, Springer Lecture Notes in Computer Science 1838, 2000

[E94] C. P. Schnorr, M. Euchner: „Lattice Basis Reduction: Improved Algorithms and solving Subset Sum Problems“, Mathematical Programming 66, 1994

[F97] G. Fischer: „Lineare Algebra“, 11. Auflage, Vieweg, 1997

[G79] M. R. Garey, D. S. Johnson: „Computers and Intractability: A Guide to the Theory of NP-Completeness“, W. H. Freeman & Company (New York), 1979

[G87] The GNU Project: „The GNU Compiler Collection“, erhältlich unter `http://gcc.gnu.org/`

[G99] O. Goldreich, D. Micciancio, S. Safra, J.-P. Seifert: „Approximating shortest lattice vectors is not harder than approximation closest vectors“, Report No. 2, The Electronic Colloquium on Computational Complexity, 1999

[G10] N. Gama, P. Q. Nguyen, O. Regev: „Lattice Enumeration Using Extreme Pruning“, EUROCRYPT 2010, International Association for Cryptologic Research, 2010

[H94] H. H. Hörner: „Verbesserte Gitterbasenreduktion; getestet am Chor-Rivest Kryptosystem und an allgemeinen Rucksack-Problemen“, Diplomarbeit im Fachbereich Mathematik an der Universität Frankfurt/Main, 1994

[J98] K. Jänich: „Lineare Algebra“, 7. Auflage, Springer, 1998

[K73] A. Korkine, G. Zolotarev: „Sur les formes quadratiques“, Mathematische Annalen 6, Springer, 1873

[K95] K. Königsberger: „Analysis I“, 3. Auflage, Springer, 1995

[L82] A. Lenstra, H. W. Lenstra, L. Lovász: „Factoring Polynomials with Rational Coefficients“, Mathematische Annalen 261, Springer, 1982

[L83] H. W. Lenstra: „Integer Programming with a Fixed Number of Variables“, Mathematics of Operations Research 8, 1983

[L05] J. Buchmann, C. Ludwig: „Practical Lattice Basis Sampling Reduction“, erhältlich unter `http://eprint.iacr.org/`, 2005

[N07] P. Q. Nguyen, T. Vidick: „Sieve Algorithms for the Shortest Vector Problem are Practical“, Journal of Math. Crypt. 1, de Gruyter, 2007

[M98] D. Micciancio: „On the Hardness of the Shortest Vector Problem“, PhD, Department of Electrical Engineering and Computer Science , Massachusetts Institute of Technology, 1998

[M99] D. Micciancio: „The Shortest Vector in a Lattice is Hard to Approximate to within Some Constant“, Report No. 16, The Electronic Colloquium on Computational Complexity, 1998

[P92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery: „Numerical Recipes in C. 2nd edn.“, Cambridge University Press, 1992

[R92] The R Project: „The R Project for Statistical Computing“, erhältlich unter `http://www.r-project.org/`

[R97] H. Ritter: „Aufzählung von Gittervektoren in allgemeiner Norm“, Dissertation im Fachbereich Mathematik an der Universität Frankfurt/Main, 1997

[S87] C. P. Schnorr: „A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms“, Theoretical Computer Science 53, 1987

[S02] C. P. Schnorr: „Gittertheorie und algorithmische Geometrie“, Vorlesung im WS 2001/2002 an der Universität Frankfurt/Main, erhältlich unter `http://www.mi.informatik.uni-frankfurt.de/teaching/lecture_notes/schnorr.gitter.ps`

[S03] C. P. Schnorr: „Lattice Reduction by Random Sanpling and Birthday Methods“, Symposium on Theoretical Aspects of Computer Science, Springer Lecture Notes in Computer Science 2607, 2003

[S04] C. P. Schnorr: „Fast LLL-Type Lattice Reduction“, erhältlich unter `http://www.mi.informatik.uni-frankfurt.de/research/papers.html`, 2004

[S08] V. Shoup: „NTL: A Library for doing Number Theory“, erhältlich unter `http://shoup.net/ntl/`

[S98] J. Hoffstein, J. Pipher, J. H. Silverman: „NTRU: A Ring Based Public Key Cryptosystem“, Algorithmic Number Theory 3, Springer Lecture Notes in Computer Science 1423, 1998

[V05] H. Vogel: „Gitterbasenreduktion mit Random Sampling“, Diplomarbeit am Lehrstuhl II für Mathematik der Universität Bayreuth, 2005, erhältlich unter `http://www.birdworx.de/math/dld/diplom_hv.pdf`

[W02] A. Wassermann: „Attacking the Market Split Problem with Lattice Point Enumeration“, Journal of Combinatorial Optimization 6, Kluwer Academic Publishers, 2002

[W06] A. Wassermann: „Lattice Point Enumeration and Applications“, Bayreuther Mathematische Schriften, Mathematisches Institut der Universität Bayreuth, 2006

[W08] H. Wang: „A Note on Solving the Market Split Problem Via Branch-and-Cut“, Manuscript No. 251, Annuals of Operations Research, 2008

Index