Theory and Algorithms for Some Integer Least Squares Problems

Xiaohu Xie

Doctor of Philosophy

School Of Computer Science

McGill University

Montreal,Quebec

November 2014

A thesis submitted to McGill University
in partial fulfilment of the requirements of the degree of
Doctor of Philosophy in Computer Science

DEDICATION

To my beloved family

and especially

to my grandmother, Yan, Yuan-Ji

ACKNOWLEDGEMENTS

I would like to give my foremost and deepest appreciation to my supervisor, Professor Xiao-Wen Chang, who provided me with precious opinions and guidance through my Ph.D studies. From him, I not only learned knowledge and research skills, but also the rigorous attitude of an excellent scholar. The research experience with him is an invaluable fortune of my life.

I would like to thank my wife Wanru Lin. She has been constantly helping and caring me in various aspects of my life. Without her encouragement and support, it would be much harder for me to finish my study and research. My life is only complete with her being part of it.

I would like to thank my lab mates Mazen Al Borno, Stephen Breen, Sevan Hanssian, Wen-Yang Ku, Milena Scaccia, David Titley-Peloquin, Yancheng Xiao and Jinming Wen for all the inspiring discussions, helpful suggestions, and brilliant ideas in the course work and research. I appreciate all the joyful time we shared together. I would also like to thank my friends Lianxin He, Xu Li, Yanyan Mu, Shaowei Png and Yuhong Zhang for their precious friendship and so many other professors and fellow students of McGill University that offered me help.

Last but not least, I would like to thank my family for their support, love and always being there for me.

ABSTRACT

Integer least squares (ILS) is an important class of optimization problems in both pure mathematics and practical applications. In this thesis, we are particularly interested in the lattice reduction strategies, which can be used to improve the efficiency of typical approaches to solving ILS problems, and the sphere decoding approach, which is the most popular approach to solving ILS problems. We study three types of ILS problems in this thesis: the ordinary ILS (OILS) problems, the box constrained ILS (BILS) problems and the mixed ILS problems with box constraints on the real variables (MILSBR). For the OILS problems, we give a rigorous proof to show that the cost of sphere decoding is reduced by applying the widely used Lenstra-Lenstra-Lovász (LLL) lattice reduction algorithm. For the BILS problems, we first show that integer Gauss transformations (IGTs) can be used in the reduction process as long as certain conditions are satisfied. Then, we propose algorithms to compute an initial search radius and propose the restricted LLL (RLLL) lattice reduction, which incorporates IGTs. Numerical test results indicate that the RLLL reduction is very effective in improving the efficiency of sphere decoding and the quality of some approximate solutions of the BILS problems. For the MILSBR problems, we first show how to solve them using the lattice reduction-sphere decoding framework and then we propose a method to find an initial search radius and methods to compute lower bounds that are used to improve the efficiency of sphere decoding. Numerical tests show that our new algorithms can be much more efficient in solving the MILSBR problems than the CPLEX optimization studio.

ABRÉGÉ

Les moindres carrés en nombres entiers (ILS) est une classe importante de problèmes d'optimisation en mathématiques pures et dans les applications pratiques. Dans cette thèse, nous sommes intéressés particulièrement par les techniques de réduction des réseaux, qui peuvent être utilisés pour améliorer l'efficacité des méthodes typiques à résoudre les problèmes ILS, et la méthode de décodage par sphères qui est populaire pour résoudre les problèmes ILS. Nous étudions trois types de problèmes ILS dans cette thèse: les problèmes ILS ordinaires (OILS), les problèmes ILS sous constrainte de boîte (BILS) et les problèmes ILS mixtes avec variables réelles sous constraintes de boîte (MILSBR). Pour les problèmes OILS, nous provons rigoureusement que le coût du décodage par sphères est réduit par l'application de l'algorithme Lenstra-Lenstra-Lovász (LLL) pour la réduction des réseaux. Pour les problèmes BILS, nous montrons tout d'abord que les transformations de Gauss avec coefficients entiers (IGTs) peuvent être utilisés dans le processus de réduction pour autant que certaines conditions sont satisfait. Ensuite, nous proposons des algorithmes pour calculer un rayon initial de recherche et proposons l'algorithme LLL restreint (RLLL), qui incorpore IGTs. Les résultats des tests numériques indiquent que la réduction RLLL améliore l'efficacité du processus de décodage par sphères et la qualité des solutions approximatives des problèmes BILS. Pour les problèmes MILSBR, nous montrons comment les résoudre en utilisant le cadre de réduction des réseaux-décodage par sphères. Ensuite, nous proposons une méthode pour trouver un rayon initial de recherche et des méthodes pour calculer des minorants qui sont utilisés pour

améliorer l'efficacité du processus de décodage par sphères. Des tests numériques montrent que nos algorithmes sont plus efficaces pour résoudre les problèmes de MILSBR que CPLEX, un outil informatique d'optimisation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

CHAPTER 1
Background

1.1 Integer Least Squares Problems

An integer least squares (ILS) problem has the following form:

$$\min_{\boldsymbol{x}\in\mathbb{Z}^n} \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2\,, \tag{1.1}$$

where $\boldsymbol{A}\in\mathbb{R}^{m\times n}$ is a given real matrix, $\boldsymbol{y}\in\mathbb{R}^m$ is a given real vector, and $\boldsymbol{x}\in\mathbb{Z}^n$ is an integer vector. In some applications, the integer vector $\boldsymbol{x}$ is subject to some constraints. When there is no constraint, the problem (1.1) is referred to as an ordinary ILS (OILS) problem, in which $\boldsymbol{A}$ is often assumed to have full column rank.

The lattice theory offers a geometric insight into the OILS problem. Let $\boldsymbol{A}=\begin{bmatrix}\boldsymbol{a}_1 & \boldsymbol{a}_2 & \dots & \boldsymbol{a}_n\end{bmatrix}$ with full column rank. In lattice theory, the set defined by

$$\Lambda(\boldsymbol{A})=\{\boldsymbol{A}\boldsymbol{x} \;\mid\; \boldsymbol{x}\in\mathbb{Z}^n\}$$

is referred to as the lattice generated by $\boldsymbol{A}$, and $\{\boldsymbol{a}_1,\boldsymbol{a}_2,\dots,\boldsymbol{a}_n\}$ is called the basis of the lattice $\Lambda(\boldsymbol{A})$. Solving an OILS problem (1.1) is actually to find a lattice vector in $\Lambda(\boldsymbol{A})$ closest to the given target vector $\boldsymbol{y}$, see Figure 1–1. Thus OILS is also referred to as a closest vector problem (CVP), see, e.g., [2, 71].

1.2 Applications of ILS

The OILS problems have many important applications in various fields including combinatorial optimization [34], algorithmic number theory [61], cryptography

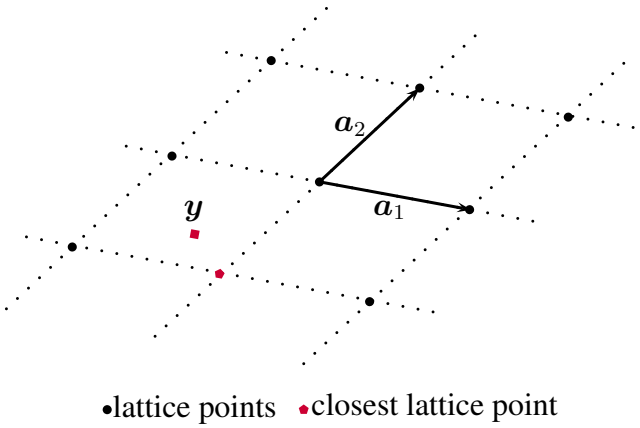


Figure 1–1: Closest lattice vector problem

and cryptanalysis [83, 4, 71, 72], GPS positioning [44, 54, 100], and wireless communications [78, 2, 77], etc. In the following, we use a few examples to illustrate how ILS problems arise in some practical areas.

1.2.1 GPS positioning

In the GPS positioning system, the navigation satellites broadcast two carrier signals: the L1 carrier signal (frequency = 1575.42 Mhz) and the L2 carrier signal (frequency = 1227.60 Mhz). Information carried by those two signals are encoded in two types of codes: the course-acquisition (C/A) code and the precision (P) code. In the standard code measurement based GPS positioning, the receiver uses the received code messages (containing the satellites clock time) to determine the transit time of each message and computes its distance to each satellite by multiplying the speed of light to the transit time. Because there are measurement errors, the computed distances between the satellites and the receiver are referred to as the code pseudoranges. For a specific satellite, the basic formula to compute the code

pseudorange p is [16]:

$$p = c(t_r - t_s),$$

where c is the speed of light, t_r is the time when the signal is received, t_s is the time when the signal was transmitted. Once the code pseudoranges corresponding to different satellites are obtained, a least squares problem is solved to estimate the position of the receiver.

Another way to compute the distances is to use the total phase difference of the transmitted signal and the received signal and multiply it by the wavelength, which is about 0.19 meters for the L1 carrier. The phase of a particular cycle of a carrier can be measured accurately with error less than 1% of a complete wave cycle [110]. However, because the signals transmitted by a GPS satellite can be thought of as pure sinusoidal waves, it is impossible to discriminate between different signal cycles. Thus the number of complete signal cycles between a satellite and the receiver remains unknown. This unknown integer component in the phase difference is referred to as the integer ambiguity. Based on the phase measurement, a phase pseudorange can be computed using the following formula

$$p = \left(\frac{\varphi}{2\pi} + N\right)\lambda,$$

where φ is the measured phase difference, $N \in \mathbb{Z}$ is the integer ambiguity and λ is the wavelength of the signal. Now, in our system, we not only have a real unknown position vector but also have integer unknown ambiguities. To resolve the integer ambiguities and estimate the receiver's position, one needs to solve a mixed ILS problem. When the real variables in a mixed ILS problem is unconstrained, it can

be shown that this mixed ILS problem can be reduced to an OILS problem. If the integer ambiguities are resolved correctly, this method reduces pseudorange error to as small as 2 millimetres, in contrast to 3 meters for the regular transit time based pseudorange measurement using the C/A code, and 0.3 meters using the P code [69].

As of November 2014, there are thirty-two satellites in the GPS constellation [102] and there are about ten satellites visible from any point on the ground at any one time. While using the widely employed double difference phase observation model (see, e.g., [86, 88]), the number of integer ambiguities to be resolved (the dimension of the OILS problem), which is twice the number of visible satellites on the sky, is about twenty. However, as other global navigation satellite systems (GNSS), e.g., BeiDou, GLONASS and Galileo, become available, and as the use of the triple difference phase observable (see, e.g., [86]) becomes the trend, the number of visible navigation satellites and the number of integer ambiguities we want to resolve increases rapidly.

In some specific positioning applications, the real variables in the mixed ILS problem to be solved are subject to constraints. For example, in the GNSS compass application, the real variables are subject to a baseline constraint (see, e.g., [101, 100]). In this application, one needs to solve a baseline constrained mixed ILS problem. In some other applications, ranges of the real position of the receiver can be obtained, e.g., on a bathymetric surveying boat where the real time altitudes can be obtained from the tide-gauge readings [111]. In this case, we suggest solving a mixed ILS problem with box constrained real variables (MILSBR). We propose an efficient algorithm to solve MILSBR in this thesis (see Chapter 4).

1.2.2 Digital communications

In multiple-input and multiple-output (MIMO) wireless communications, multiple transmitters and receivers are used to transfer data at the same time. In a narrowband flat fading MIMO system, the received signal vectors are given by linear combinations of transmit symbol vectors corrupted by additive Gaussian noise. We have

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{v}, \tag{1.2}$$

where $\boldsymbol{H} \in \mathbb{C}^{m\times n}$ is the channel matrix, $\boldsymbol{x}$ is an n-dimensional transmitted symbol vector where each transmitted symbol x_i is a complex number chosen from a known finite alphabet $\chi = \{s_1, s_2, \ldots, s_M\}$, $\boldsymbol{y} \in \mathbb{C}^m$ is the received signal vector and $\boldsymbol{v} \in \mathbb{C}^m$ is a zero-mean Gaussian noise vector. Furthermore, we assume that the transmitted symbol vectors $\boldsymbol{x}$ in (1.2) are points in an M-QAM constellation where $M = 4^k$ for some $k \in \mathbb{Z}^+$. So that we have $\chi = \left\{k_1 + k_2 i \mid k_1, k_2 = \pm 1, \pm 3, \ldots, \pm(2^k - 1)\right\}$ where $i^2 = -1$. The maximum-likelihood decoding approach to estimate the transmitted $\boldsymbol{x}$ is to solve the following problem:

$$\min_{\boldsymbol{x}\in\chi^n} \|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}\|_2^2 . \tag{1.3}$$

We let

$$\tilde{\boldsymbol{y}} = \begin{bmatrix} \boldsymbol{y}^{\mathrm{R}} \\ \boldsymbol{y}^{\mathrm{I}} \end{bmatrix}, \quad \tilde{\boldsymbol{H}} = \begin{bmatrix} \boldsymbol{H}^{\mathrm{R}} & -\boldsymbol{H}^{\mathrm{I}} \\ \boldsymbol{H}^{\mathrm{I}} & \boldsymbol{H}^{\mathrm{R}} \end{bmatrix}, \quad \tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x}^{\mathrm{R}} \\ \boldsymbol{x}^{\mathrm{I}} \end{bmatrix},$$

where the superscript "R" indicates the real part of a complex matrix or vector, and the superscript "I" indicates its imaginary part. We further let $\bar{\boldsymbol{x}} = (\tilde{\boldsymbol{x}} + (2^k - 1)\mathbf{1}_{2n})/2$, $\bar{\boldsymbol{H}} = 2\tilde{\boldsymbol{H}}$, and $\bar{\boldsymbol{y}} = \tilde{\boldsymbol{y}} + (2^k - 1)\tilde{\boldsymbol{H}}\mathbf{1}_{2n}$, where $\mathbf{1}_{2n}$ denotes a $2n$-vector with all entries one (see Section 1.7). It is easy to see that $\bar{\boldsymbol{x}} \in \left\{0, 1, 2, \ldots, 2^k - 1\right\}^{2n}$ and

$\|\boldsymbol{y}-\boldsymbol{H}\boldsymbol{x}\|_2^2 = \|\tilde{\boldsymbol{y}}-\tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}}\|_2^2 = \|\bar{\boldsymbol{y}}-\bar{\boldsymbol{H}}\bar{\boldsymbol{x}}\|_2^2$. To solve (1.3), we just solve the following box constrained ILS (BILS) problem (see, e.g., [30, 48, 24]):

$$\min_{\bar{\boldsymbol{x}}\in\mathcal{B}} \|\bar{\boldsymbol{y}}-\bar{\boldsymbol{H}}\bar{\boldsymbol{x}}\|_2^2, \quad \mathcal{B} = \{\boldsymbol{x} \mid \boldsymbol{x}\in\mathbb{Z}^{2n}; \mathbf{0}_{2n} \le \boldsymbol{x} \le (2^k-1)\mathbf{1}_{2n}\}.$$

The BILS problem is studied in Chapter 3. In MIMO, m and n are the number of transmit antennas and receive antennas. In typical applications of MIMO, m and n usually take values from 2 to 8 (see, e.g., [62]). However in massive MIMO, one of the most recent developments of the MIMO technology, the number of antennas can up to a few hundred (see, e.g., [87, 59]).

Other than MIMO, ILS problems arises in other areas of digital communications including packing, quantization, and signaling for the additive white Gaussian noise (AWGN) channel (see, e.g., [36, 95, 106]).

1.3 Hardness of Solving ILS Problems

The complexity of the general ILS problem has been studied by several researchers. It was shown in [103] that it is an NP-hard problem; the simple proof presented below was given in [70].

Consider a subset sum problem: given a set of integers $C = \{c_1, \ldots, c_n\}$ and an integer number s, is there a non-empty subset of C whose sum is s? The problem is an important NP-complete problem in complexity theory and cryptography. Now

we define a matrix $\boldsymbol{A} \in \mathbb{Z}^{(n+1)\times n}$ and vector $\boldsymbol{y} \in \mathbb{Z}^{n+1}$ as follows:

$$\boldsymbol{A} = \begin{bmatrix} c_1 & c_2 & \dots & c_n \\ 2 & & & \\ & 2 & & \\ & & \ddots & \\ & & & 2 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} s \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

Then for any $\boldsymbol{x} \in \mathbb{Z}^n$,

$$\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 = \left\| \begin{bmatrix} \sum_{i=1}^n c_i x_i - s \\ 2x_1 - 1 \\ 2x_2 - 1 \\ \dots \\ 2x_n - 1 \end{bmatrix} \right\|_2^2 = \left(\sum_{i=1}^n c_i x_i - s \right)^2 + \sum_{i=1}^n (2x_i - 1)^2 \geq n.$$

Assume that there exists a solution to the subset sum problem, i.e., there are $x_i \in \{0, 1\}$ such that $\sum_{i=1}^n c_i x_i = s$. Then for this specific $\boldsymbol{x}$, it is easy to verify that $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 = n$. Therefore, if $\boldsymbol{x}$ is a solution to the subset sum problem, then $\boldsymbol{x}$ is a solution to the constructed ILS problem $\min_{\boldsymbol{x}\in\mathbb{Z}^n} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$. On the other hand, if $\boldsymbol{x}$ is a solution to the constructed ILS problem and $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 = n$, then it is easy to see that $\boldsymbol{x}$ must be a solution to the subset sum problem; if $\boldsymbol{x}$ is a solution to the constructed ILS problem and $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 > n$, then the subset sum problem has no solution. Since the subset sum problem can be reduced to the ILS problem, we can conclude that the ILS problem is NP-hard. This means all known algorithms for solving (1.1) have exponential complexity.

Approximation of ILS is also difficult. Assume the optimal solution of (1.1) is $\boldsymbol{x}^*$. It is known that finding an approximate solution $\bar{\boldsymbol{x}}$ of (1.1) such that the

residual ratio $\|\boldsymbol{A}\bar{\boldsymbol{x}} - \boldsymbol{y}\| / \|\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{y}\|$ (also known as the approximation ratio) is upper-bounded by a constant is also NP-hard [9].

1.4 Lattice Reduction

Lattice reduction is a process of improving the "quality" of the lattice basis. It is pervasively used as preprocessing to improve the efficiency of typical solvers for ILS problems.

For any particular lattice $\Lambda(\boldsymbol{A})$ with $n > 1$, the choice of basis is not unique, i.e., we can find infinitely many $\bar{\boldsymbol{A}} \in \mathbb{R}^{m\times n}$ such that $\Lambda(\boldsymbol{A}) = \Lambda(\bar{\boldsymbol{A}})$. In the following, we give the general relation between $\boldsymbol{A}$ and $\bar{\boldsymbol{A}}$ when $\Lambda(\boldsymbol{A}) = \Lambda(\bar{\boldsymbol{A}})$. To do that, we will introduce some basic results with proofs, which will be used later.

Definition 1–1. A square matrix $\boldsymbol{Z}$ is unimodular if $\boldsymbol{Z}$ is an integer matrix, i.e., $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$, and $\det(\boldsymbol{Z}) = \pm 1$.

Proposition 1–1. Given a matrix $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$, $\boldsymbol{Z}$ is unimodular if and only if $\boldsymbol{Z}^{-1} \in \mathbb{Z}^{n\times n}$.

Proof. Since $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$, the adjugate matrix $\operatorname{adj}(\boldsymbol{Z}) \in \mathbb{Z}^{n\times n}$. If $\boldsymbol{Z}$ is unimodular, then $\boldsymbol{Z}^{-1} = \det(\boldsymbol{Z})^{-1} \operatorname{adj}(\boldsymbol{Z}) \in \mathbb{Z}^{n\times n}$.

On the other hand, suppose $\boldsymbol{Z}^{-1} \in \mathbb{Z}^{n\times n}$, then $\det(\boldsymbol{Z}), \det\left(\boldsymbol{Z}^{-1}\right) \in \mathbb{Z}$. But $\det(\boldsymbol{Z}) \det\left(\boldsymbol{Z}^{-1}\right) = 1$, so we must have $\det(\boldsymbol{Z}) = \det\left(\boldsymbol{Z}^{-1}\right) = \pm 1$. Thus $\boldsymbol{Z}$ must be unimodular, completing the proof. □

Proposition 1–2. Given a matrix $\boldsymbol{Z} \in \mathbb{R}^{n\times n}$, then $\Lambda(\boldsymbol{Z}) = \mathbb{Z}^n$ if and only if $\boldsymbol{Z}$ is unimodular.

Proof. Suppose $\boldsymbol{Z}$ is unimodular, then $\boldsymbol{Z}, \boldsymbol{Z}^{-1} \in \mathbb{Z}^{n\times n}$. For any $\boldsymbol{z} \in \Lambda(\boldsymbol{Z}) = \{\boldsymbol{Z}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{Z}^n\}$, it is obvious that $\boldsymbol{z} \in \mathbb{Z}^n$. Also, for any $\boldsymbol{z} \in \mathbb{Z}^n$, define $\boldsymbol{x} = \boldsymbol{Z}^{-1}\boldsymbol{z} \in \mathbb{Z}^n$. Then $\boldsymbol{z} = \boldsymbol{Z}\boldsymbol{x} \in \Lambda(\boldsymbol{Z})$. Thus, we must have $\Lambda(\boldsymbol{Z}) = \mathbb{Z}^n$.

On the other hand, suppose $\Lambda(\boldsymbol{Z}) = \mathbb{Z}^n$, we must have $\boldsymbol{Z}\boldsymbol{e}_i \in \mathbb{Z}^n$ for $i = 1, 2, \ldots, n$, where $\boldsymbol{e}_i$ is a unit vector whose i-th entry is one and other entries are zeros, see Section 1.7. Then $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$. Since $\boldsymbol{e}_i \in \mathbb{Z}^n = \Lambda(\boldsymbol{Z})$ for $i = 1, 2, \ldots, n$. Thus there must exist vectors $\boldsymbol{h}_i \in \mathbb{Z}^n$ such that $\boldsymbol{e}_i = \boldsymbol{Z}\boldsymbol{h}_i$. Let $\boldsymbol{H} = \begin{bmatrix}\boldsymbol{h}_1 & \boldsymbol{h}_2 & \ldots & \boldsymbol{h}_n\end{bmatrix} \in \mathbb{Z}^{n\times n}$, we have $\boldsymbol{Z}\boldsymbol{H} = \boldsymbol{I}_{n\times n}$. Thus $\boldsymbol{H} = \boldsymbol{Z}^{-1}$. From Proposition 1–1, $\boldsymbol{Z}$ must be unimodular. □

Proposition 1–3. (see, e.g., [2]) Given two matrices $\boldsymbol{A}, \bar{\boldsymbol{A}} \in \mathbb{R}^{m\times n}$ which have full column rank, then, $\Lambda(\boldsymbol{A}) = \Lambda(\bar{\boldsymbol{A}})$ if and only if there exists a unimodular matrix $\boldsymbol{Z}$ such that $\bar{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{Z}$.

Proof. We first prove that $\Lambda(\boldsymbol{A}) = \Lambda(\boldsymbol{A}\boldsymbol{Z})$ if and only if $\boldsymbol{Z}$ is a unimodular matrix. We can see that $\Lambda(\boldsymbol{A}) = \{\boldsymbol{A}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{Z}^n\}$ and $\Lambda(\boldsymbol{A}\boldsymbol{Z}) = \{\boldsymbol{A}\boldsymbol{x} \mid \boldsymbol{x} \in \Lambda(\boldsymbol{Z})\}$. When $\boldsymbol{A}$ has full column rank, $f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ is a bijection. Thus $\{\boldsymbol{A}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{Z}^n\} = \{\boldsymbol{A}\boldsymbol{x} \mid \boldsymbol{x} \in \Lambda(\boldsymbol{Z})\}$ holds if and only if $\Lambda(\boldsymbol{Z}) = \mathbb{Z}^n$. Then, from Proposition 1–2, we can conclude that $\Lambda(\boldsymbol{A}) = \Lambda(\boldsymbol{A}\boldsymbol{Z})$ if and only if $\boldsymbol{Z}$ is a unimodular matrix.

Now if $\Lambda(\boldsymbol{A}) = \Lambda(\bar{\boldsymbol{A}})$, we can always find a matrix $\boldsymbol{Z} \in \mathbb{R}^{n\times n}$ such that $\bar{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{Z}$, completing the proof. □

Given a lattice basis matrix $\boldsymbol{A}$, the process of lattice reduction is to find a unimodular matrix $\boldsymbol{Z}$ such that the column vectors of the new basis matrix $\bar{\boldsymbol{A}} \equiv \boldsymbol{A}\boldsymbol{Z}$ are short, i.e., more orthogonal to each other, see Figure 1–2.

Latice reductions have been studied for more than a century and many different types of reductions have been proposed. The first rigorous definition of lattice reduction in arbitrary dimension was suggested by Hermite in [47] in 1850. Other types of lattice reductions that are often considered in literature are the Korkin-Zolotarev (KZ) reduction [56] proposed in 1873, the Minkowski reduction [74] proposed in 1896 and the Lenstra-Lenstra-Lovász (LLL) reduction [60] proposed in 1982. Next, we briefly review the two most popular lattice reductions: the KZ reduction and the LLL reduction, and then review Schnorr's hierarchy of lattice reduction algorithms that stretch from the LLL reduction to the KZ reduction.

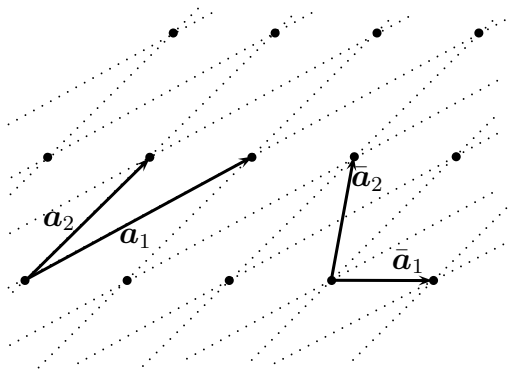


Figure 1–2: Lattice reduction

1.4.1 The Korkin-Zolotarev reduction

A basis matrix $\boldsymbol{A}$ is KZ reduced if

i $\boldsymbol{a}_1$ is the shortest non-zero lattice vector in $\Lambda(\boldsymbol{A})$ (see later);

ii For any $1 < k \leq n$, $\left|\boldsymbol{a}_1^{\mathsf{T}}\boldsymbol{a}_k\right| \leq \|\boldsymbol{a}_1\|_2^2/2$;

iii The basis matrix formed by the orthogonal projections of $\begin{bmatrix}\boldsymbol{a}_2 & \boldsymbol{a}_3 & \dots & \boldsymbol{a}_n\end{bmatrix}$ onto the orthogonal complement of the space spanned by $\boldsymbol{a}_1$ is KZ reduced.

To find the shortest non-zero lattice vector in $\Lambda(\boldsymbol{A})$, one needs to solve the following problem:

$$\min_{\boldsymbol{x}\in\mathbb{Z}^n\setminus\mathbf{0}} \|\boldsymbol{A}\boldsymbol{x}\|_2^2 . \tag{1.4}$$

This problem, (1.4), is referred to as the shortest vector problem (SVP) in lattice theory. The SVP is one of the two most fundamental problems in the lattice theory (the other one is CVP), and it has drawn great attention from researchers. Solving SVP is difficult (there is no polynomial-time solver yet). In [103], the SVP is conjectured to be NP-hard. Recently in [3], SVP is proven to be NP-hard under reverse unfaithful random reductions. Though this does not prove the NP-hardness of SVP, it implies the intractability of this problem. For more on solving SVP, we refer the reader to the survey paper [42].

The computation of the KZ reduced basis of a given lattice involves solving a series of SVP problems. Thus the KZ reduction is very expensive to compute.

1.4.2 The Lenstra-Lenstra-Lovász reduction

A breakthrough of the algorithmic study of lattice reduction was made in [60] in which the LLL reduction was proposed. The greatest advantage of the LLL reduction is that it can be computed efficiently in polynomial time (under certain conditions, see [60] and [49]). And thus it has become the most widely employed lattice reduction in practice.

As the LLL reduction is crucial for our later theoretical analysis and algorithm design, the definition and the algorithm will be introduced in Section 2.1. For a given lattice, the LLL-reduced basis is less orthogonal than the KZ-reduced basis. A KZ-reduced matrix is also LLL-reduced, but the reverse does not hold. If $\boldsymbol{A}$ is

LLL-reduced, the first basis vector $\boldsymbol{a}_1$ offers an approximate solution to the SVP problem. The ratio of $\|\boldsymbol{a}_1\|_2$ to the length of the shortest lattice vector is bounded by 2^n [80].

Many variants of the LLL reduction have been proposed in the literature. Note that the original LLL algorithm uses integer/rational arithmetic and adopts Gram-Schmidt orthogonalisation (GSO), see Section 2.1. Some of the variants follow the tradition of using integer/rational arithmetic and aim at reducing the computational complexity of the LLL reduction (see, e.g., [91, 79, 38, 82]). The problem is that using exact arithmetic may lead to the use of very large integers, slowing down the algorithm. A floating point LLL algorithm, named H-LLL, was proposed by Moreal, Stehlé and Villard in [76]. This algorithm relies on the computation of the QR-factorization of the basis using Householder's algorithm. H-LLL computes floating point approximations to the coefficients of the R-factor and uses them to perform exact operations on the basis. It was shown that if the precision is large enough, then H-LLL runs correctly. For some applications such as solving ILS problems, LLL is used to accelerate the search process (see Section 1.5.1). In this situation, a nearly LLL reduced basis is acceptable. So for solving ILS problems, one almost always uses floating point arithmetic (fpa), see, for example, [31, 28, 64, 109].

In [64], Ling and Howgrave-Graham showed that when the LLL reduction is used to improve the performance of the Babai point estimation [10], some steps of the reduction are not necessary and, based on this, proposed the effective LLL (ELLL) algorithm which is more computationally efficient. However, the ELLL algorithm can have numerical stability issue, see [64, 109]. In [109], we addressed this problem and

further improved the computational efficiency by proposing the partial LLL (PLLL) algorithm. An ELLL or PLLL reduced basis is not necessarily LLL reduced, but in [109] we proved that ELLL and PLLL are equivalent to LLL in improving the efficiency of Phost's discrete enumeration approach (see section 1.5.1) for solving (1.1).

1.4.3 Schnorr's hierarchy of lattice reductions

Schnorr's hierarchy of lattice reduction algorithms proposed in [90] compromises the KZ reduction's quality and the LLL reduction's efficiency by treating every k consecutive columns of $\boldsymbol{A}$ as a block. The KZ reduction is performed to reduce the basis vectors within a block, and the inter-block reduction is handled in the LLL-manner. The KZ reduction and the LLL reduction lie on the two opposite ends of Schnorr's hierarchy where the block size k equals to n and 2 respectively. Based on Schnorr's hierarchy of lattice reduction, the block KZ (BKZ) reduction was proposed in [89].

1.5 Methods for Solving ILS

OILS has been investigated from different perspectives for decades and many different algorithms for solving OILS have been developed. We refer the reader to the surveys [2] and [42]. Below, we briefly review some of the important approaches to solving OILS.

1.5.1 The discrete enumeration algorithms

Some of the earliest algorithms for solving OILS are the discrete enumeration algorithms, which enumerate all possible lattice vectors in a given vicinity region of the target vector $\boldsymbol{y}$. By orthogonal triangularization (e.g., using the Gram-Schmidt

process) of the lattice basis, the discrete enumeration algorithms can recursively bound the integer coordinates of the candidate solution. Distinguished by the different shapes of the search region, the enumeration-based solvers can be divided into two main branches. One branch follows Phost's approach [85, 35] which enumerates lattice vectors in a hyper sphere, and the other branch follows Kannan's approach [51, 52] which examines the lattice vectors in a parallelotope. An important improvement of Phost's approach was proposed by Schnorr and Euchner in [89]. Schnorr-Euchner's algorithm enumerates the lattice points in a hyper sphere as Phost's algorithm does, but in a more efficient order, see Section 2.2. In general, Phost's approach is more often used in practice and Kannan's approach serves more as a theoretical tool. The discrete enumeration algorithms following Phost's approach (which includes Schnorr-Euchner's algorithm) are referred to as the *sphere decoding* algorithms in communications [35]. In this thesis, we often use this term, and the search process of a sphere decoder is referred to as a sphere decoding process.

A crucial parameter that affects the performance of the enumeration algorithms is the initial size of the search region. Suggestions on choosing the initial search region size can be found in, e.g., [78, 105] for Phost's approach and in, e.g., [11] for Kannan's approach. The computational complexity analysis of the enumeration based approaches was first given in [45, 52] and it was refined in [43], which showed that a discrete enumeration based OILS solver has time complexity $O(n^{n/2+o(n)})$ (using Kannan's approach). Space complexity of the enumeration approach is polynomial in n (see, e.g., [42]).

1.5.2 The Monte Carlo algorithms

In [5, 6], nondeterministic algorithms have been proposed for solving the OILS problem in a singly exponential time complexity. This algorithm was improved in [17] later on. Those algorithms are Monte Carlo algorithms, which can find lattice vectors that are no more than $(1+\epsilon)$ times further away from the target vector than the optimal solution is, for arbitrary $\epsilon > 0$. The current best bound on the time and space complexities of those algorithms are both $(2+1/\epsilon)^{O(n)}$, see [42].

1.5.3 The Voronoi cell based algorithms

The deterministic algorithm that currently gives the best time complexity was proposed in [73]. It solves OILS based on the computation of Voronoi cell of the lattice [107]. The Voronoi cell $V(\boldsymbol{t})$ of a lattice vector $\boldsymbol{t} \in \Lambda(\boldsymbol{A})$ is the set of real vectors $\bar{\boldsymbol{t}} \in \mathbb{R}^n$ whose closest lattice vector is $\boldsymbol{t}$, i.e.,

$$V(\boldsymbol{t}) = \{\bar{\boldsymbol{t}} \mid \|\boldsymbol{t} - \bar{\boldsymbol{t}}\|_2 \leq \|\boldsymbol{t}' - \bar{\boldsymbol{t}}\|_2, \quad \forall \boldsymbol{t}' \in \Lambda(\boldsymbol{A})\}.$$

Figure 1–3 gives an example of a Voronoi cell. The Voronoi cells $V(\boldsymbol{t})$ are translations of $V(\boldsymbol{0})$, and they are convex polytopes which are symmetrical with respect to reflection in $\boldsymbol{t}$ [95]. By computing the Voronoi cells of the lattice, OILS can be solved in time $O(2^{2n+o(n)})$ and space $2^{n+o(n)}$ [73].

1.5.4 The real relaxation branch-and-bound approach

The real relaxation branch-and-bound (RRBB) approach was first considered in [57] for ILS problems and it was improved later in [8]. The RRBB approach is based on the branch-and-bound paradigm commonly used in mixed-integer programming (see, e.g., [14, 68]). In RRBB, an ILS is solved by solving a sequence of its real

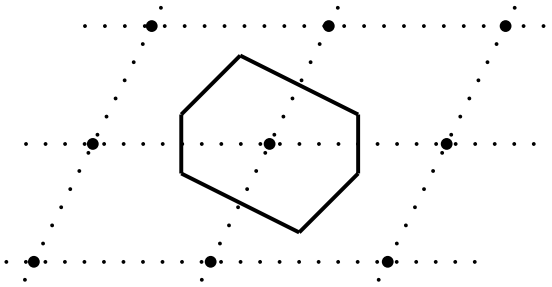


Figure 1–3: The Voronoi cell of a lattice vector

relaxations. It starts by solving problem P_0, the real relaxation of (1.1):

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2,$$

to obtain the real least squares solution $\check{\boldsymbol{x}}$. If $\check{\boldsymbol{x}}$ is an integer vector, then it is returned as the solution to (1.1). Otherwise, the solver chooses $i \in \{1,\ldots,n\}$ such that $\check{x}_i \notin \mathbb{Z}$, and creates two subproblems: P_1 with additional constraint $x_i \leq \lfloor \check{x}_i \rfloor$, and P_2 with additional constraints $x_i \geq \lceil \check{x}_i \rceil + 1$. Then the solver iterates on the subproblems P_1 and P_2. Thus the RRBB process forms a binary search tree. By exploring the search tree, the RRBB method finds the optimal solution of (1.1). In the attempt to improve computational efficiency, the search tree is usually pruned in the solving process using various strategies [8].

1.5.5 Solving ILS problems in practice

In terms of computational efficiency for currently handleable dimensions, the Voroni cell based algorithms are known to be uncompetitive, and the Monte-Carlo algorithms are outperformed by the discrete enumeration based algorithms, see [42]. Although the RRBB approach may be more efficient than the discrete enumeration approach for some types of ILS problems, often the latter is more efficient than

the former, see [8]. The high popularity and high efficiency of discrete enumeration algorithms in practice have led to some hardware implementations [32, 46].

Due to the NP-hardness of ILS, searching for the optimal solution of (1.1) may become time-prohibitive when, e.g., $\boldsymbol{A}$ is ill conditioned, the ILS residual is large, or the dimension of the problem is large [48]. So for some applications, an approximate solution, which can be produced more quickly, is computed instead (see, e.g., [63]).

One often used approximate solution in communications is the Babai point, produced by Babai's nearest plane algorithm [10], which achieves a polynomial time complexity $O(n^2)$ and an approximation ratio upper-bounded by $2\left(2/\sqrt{3}\right)^n$. The Babai point is also the first integer point found by Schnorr-Euchner's sphere decoding algorithm. In communications, a method for finding this approximate solution is referred to as a successive interference cancellation decoder. Lattice reduction can be used to improve the quality of the Babai point, see, e.g., [63, 25].

Another approximation method is the semidefinite programming (SDP) approach which was first considered in [99]. The performance of this approach was further analyzed in [55] where the optimality conditions for the SDP approach were given. The SDP approach is based on a semidefinite relaxation of a given ILS problem, which can be solved efficiently using the interior point methods, and cutting planes are introduced to strengthen the approximation [99]. The SDP approach offers a near-optimal solution (under certain conditions) with a complexity that is exponential in the worst case [55].

1.6 Organization and Contributions

In this thesis, we are mainly interested in solving the ILS problems exactly, focusing on the lattice reductions and discrete enumeration algorithms.

The rest of this thesis is organized as follows. In Chapter 2, we study the OILS problems. We first give details on the LLL reduction algorithm, which involves column permutations and size reductions, and the sphere decoding approach for solving OILS problems. Then, we discuss how the computational cost of the sphere decoding algorithms will change in each step of the LLL reduction process and show that the efficiency of sphere decoding is improved by the LLL reduction. It is well-known that LLL can improve the efficiency of sphere decoding. But we are the first to show it rigorously in theory. We also look at how will the value of δ, an important parameter in the LLL reduction, affects the cost of sphere decoding. Part of this chapter has been published in [25], which also includes other results that are not covered by this thesis.

In Chapter 3, we study the BILS problems. It is widely believed that incorporating size reductions in the reduction process for the BILS problems would make sphere decoding difficult. So only column permutations are used in the existing reduction algorithms. In this chapter, we introduce the concept of inactive set and based on that, we show that size reductions can be used to transform the BILS problems without bringing any difficulty to sphere decoding as long as certain conditions are satisfied. Then, we propose algorithms to find an inactive set for a BILS problem based on a given initial search radius. To find an initial search radius for a given BILS problem, we also propose some algorithms. Based on those algorithms, we

propose the restricted LLL (RLLL) reduction for the BILS problems which incorporates size reductions. Finally, results of numerical experiments are given to show the potential of the RLLL reduction in improving the efficiency of the sphere decoding approach and the quality of some approximate solutions. The main results given in this chapter will appear in [27].

In Chapter 4, we investigate the mixed ILS problems with box constraints on the real variables (MILSBR). We first show how the MILSBR problems can be solved using lattice reductions and the sphere decoding approach. Then, we propose an algorithm to compute an initial search radius which makes use of the information in the box constraint. After that, we show how to use lower bounds to improve the efficiency of sphere decoding, and propose algorithms to find the lower bounds. Finally, we give numerical experiments to show that using the new algorithms, we can solve the MILSBR problems much faster than the popular commercial software package CPLEX does. The results of this chapter will appear in [26].

In Chapter 5, we conclude the whole thesis and give directions for future research.

1.7 Notation

In the following, we define the notation used in this thesis.

Scalar: Scalars are represented by lower case Greek $\alpha, \beta, \ldots, \lambda, \mu, \nu$, or by lower case Roman $a, b, c, \ldots$ (usually with subscripts, e.g., a_i, a_{ij}).

Vector: Vectors are represented by bold lower case Roman $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \ldots$. Superscript T indicates a transposition or a row vector, e.g.,

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} = (a_i)_m, \quad \boldsymbol{c}^{\mathsf{T}} = \begin{bmatrix} c_1 & c_2 & \ldots & c_n \end{bmatrix} = (c_i)_n^{\mathsf{T}}.$$

We use $\boldsymbol{e}_i$ to denote the i-th standard basis vector (column vector), i.e.,

$$\boldsymbol{e}_i = \begin{bmatrix} \underbrace{0 & \ldots & 0}_{i-1} & 1 & 0 & \ldots & 0 \end{bmatrix}^{\mathsf{T}}.$$

We use $\mathbf{1}_n$ to denote the n-dimensional vector with all 1's, and $\mathbf{0}_n$ to denote the n-dimensional vector with all 0's, i.e.,

$$\mathbf{1}_n = (1)_n, \quad \mathbf{0}_n = (0)_n.$$

Matrix: Matrices are represented by bold upper case Roman $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \ldots$, e.g.,

$$\boldsymbol{A} = \underbrace{\begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \ldots & \boldsymbol{a}_n \end{bmatrix}}_{\text{column partition}} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} = (a_{ij})_{m \times n}.$$

The transposition of a matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^{\mathsf{T}}$. Given a matrix $\boldsymbol{A}$, we use $\boldsymbol{a}_{i:j,k}$ to denote the vector formed by entries in column k and rows i to j of $\boldsymbol{A}$ inclusively, i.e.,

$$\boldsymbol{a}_{i:j,k} = \begin{bmatrix} a_{i,k} & a_{i+1,k} & \ldots & a_{j,k} \end{bmatrix}^{\mathsf{T}}.$$

We use $\boldsymbol{A}_{i:j,p:q}$ to denote a submatrix of $\boldsymbol{A}$, i.e., $\boldsymbol{A}_{i:j,p:q} = \begin{bmatrix}\boldsymbol{a}_{i:j,p} & \boldsymbol{a}_{i:j,p+1} & \dots & \boldsymbol{a}_{i:j,q}\end{bmatrix}$. The unit matrix of dimension n is denoted by $\boldsymbol{I}_n$, e.g.,

$$\boldsymbol{I}_n = \begin{bmatrix}\boldsymbol{e}_1 & \boldsymbol{e}_2 & \dots & \boldsymbol{e}_n\end{bmatrix} = \begin{bmatrix}1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1\end{bmatrix}.$$

The matrix with only 0's is denoted by $\mathbf{0}$.

Rounding: The rounding of $x \in \mathbb{R}$ is represented by $\lfloor x \rceil$, which is the nearest integer to x, i.e., $\lfloor 1.6 \rceil = 2$. If two integers have the same the distances to x, then the one that is closer to 0 is chosen, e.g., $\lfloor 0.5 \rceil = 0$. When rounding is applied to a vector or matrix, it rounds the vector or matrix entry by entry, i.e., for $\boldsymbol{a} = (a_i)_n$, we have $\lfloor \boldsymbol{a} \rceil = (\lfloor a_i \rceil)_n$.

Boolean: Boolean variables are represented by lower case Roman in Fraktur font $\mathfrak{a}, \mathfrak{b}, \dots$. Each Boolean variable can take one of the two values: 1 – true, and 0 – false. Boolean vectors are represented using the bold Fraktur font $\boldsymbol{\mathfrak{a}}, \boldsymbol{\mathfrak{b}}, \dots$. Logical operations used in this thesis are: $\wedge$ – logic and; $\vee$ – logic or; $\neg$ – logic not; When a relational operator, e.g., $<$, is used in a Boolean context, it returns a Boolean value to indicate whether or not the relation is true. When a logical or comparison operator takes a vector/vectors as its operand(s), it works on the operand(s) entry by entry, i.e.,

$$\neg\boldsymbol{\mathfrak{a}} = (\neg\mathfrak{a}_i)_n, \quad \boldsymbol{\mathfrak{a}} \wedge \boldsymbol{\mathfrak{b}} = (\mathfrak{a}_i \wedge \mathfrak{b}_i)_n, \quad \boldsymbol{a} \leq \boldsymbol{b} = (a_i \leq b_i)_n.$$

CHAPTER 2
Effects of the LLL Reduction on the Efficiency of Sphere Decoding

The most efficient approach to solving an OILS problem in practice is Phost's discrete enumeration approach, which is also referred to as sphere decoding in digital communications (see Section 1.5). In this thesis, we often use the name sphere decoding to refer to Phost's discrete enumeration approach, and use sphere decoders to denote sphere decoding algorithms. The LLL reduction is often used as preprocessing (see Section 1.4). It is well-known that the LLL reduction can make sphere decoders faster (see, e.g., [44, 2]). But to our knowledge there has been no rigorous justification. In this chapter, we show rigorously that applying the LLL reduction algorithm will reduce the computational cost of sphere decoders, which is measured approximately by the number of nodes in the search tree in the literature. Finally, we discussed how different values of δ, an important coefficient of the LLL reduction, will affect the cost of sphere decoders. Part of this chapter was published in [25].

2.1 The Reduction of OILS Problems and the LLL Reduction Algorithm

For convenience, we rewrite the OILS problem here:

$$\min_{\boldsymbol{x}\in\mathbb{Z}^n} \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2 . \tag{2.1}$$

To solve (2.1) using the sphere decoding approach, one usually transforms it to a simpler form first. Typically, one computes the QRZ factorization of $\boldsymbol{A}$ (see, e.g.,

[23]):

$$\boldsymbol{Q}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{Z} = \begin{bmatrix}\boldsymbol{R}\\ \boldsymbol{0}\end{bmatrix}, \tag{2.2}$$

where $\boldsymbol{Q} = \underset{n}{\begin{bmatrix}\boldsymbol{Q}_1\end{bmatrix}} \quad \underset{m-n}{\begin{bmatrix}\boldsymbol{Q}_2\end{bmatrix}} \in \mathbb{R}^{m\times m}$ is orthogonal, $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ is upper triangular (without loss of generality, we assume its diagonal entries are positive throughout this thesis), and $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$ is a unimodular matrix. Then we have

$$\|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2 = \left\|\boldsymbol{Q}^{\mathsf{T}}\left(\boldsymbol{y}-\boldsymbol{Q}\begin{bmatrix}\boldsymbol{R}\\ \boldsymbol{0}\end{bmatrix}\boldsymbol{Z}^{-1}\boldsymbol{x}\right)\right\|_2^2 = \left\|\boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{y}-\boldsymbol{R}\boldsymbol{Z}^{-1}\boldsymbol{x}\right\|_2^2 + \left\|\boldsymbol{Q}_2^{\mathsf{T}}\boldsymbol{y}\right\|_2^2.$$

Let $\bar{\boldsymbol{y}} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{y}$ and $\boldsymbol{z} = \boldsymbol{Z}^{-1}\boldsymbol{x}$. Then the ILS problem (2.1) can be transformed to

$$\min_{\boldsymbol{z}\in\mathbb{Z}^n} \|\bar{\boldsymbol{y}}-\boldsymbol{R}\boldsymbol{z}\|_2^2. \tag{2.3}$$

The transformation from (2.1) to (2.3) is also referred to as reduction in the literature. Then we can apply a sphere decoder to find the solution of (2.3) (see Section 2.2). Once the solution $\boldsymbol{z}^*$ of (2.3) is found, the solution $\boldsymbol{x}^*$ of the original problem (2.1) can be computed using $\boldsymbol{x}^* = \boldsymbol{Z}\boldsymbol{z}^*$.

Different unimodular matrix $\boldsymbol{Z}$ in (2.2) will result in different $\boldsymbol{R}$, making the speed of search process by a sphere decoder different. It turns out that lattice reductions can often make the search process faster. The widely used lattice reduction for the ILS problem is the LLL reduction [60], which finds a QRZ factorization (2.2) such that the upper triangular matrix $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ satisfies the following two conditions for $k = 2, 3, \ldots, n$ and $i = k-1, k-2, \ldots, 1$:

$$\text{size-reduction condition:} \qquad |r_{ik}| \le r_{ii}/2, \tag{2.4a}$$

$$\text{Lovász condition:} \qquad \delta r_{k-1,k-1}^2 \le r_{k-1,k}^2 + r_{kk}^2, \tag{2.4b}$$

where the parameter δ is a constant and $\delta \in (1/4, 1]$. If $\boldsymbol{R}$ in (2.2) satisfies those two conditions, then $\boldsymbol{R}$ and $\boldsymbol{AZ}$ are said to be δ-LLL reduced or simply LLL reduced.

The original LLL algorithm [60] first applies the Gram-Schmidt orthogonalization (GSO) process to $\boldsymbol{A}$, getting the initial triangular matrix $\boldsymbol{R}$ (more precisely, to avoid square root computation, the original LLL algorithm gives a row scaled $\boldsymbol{R}$ which has unit diagonal entries). Then $\boldsymbol{R}$ is updated by orthogonal transformations from the left and unimodular transformations from the right to eventually form the LLL-reduced upper triangular matrix in (2.2). All those unimodular transformations form the factor $\boldsymbol{Z}$. All the orthogonal transformations form the factor $\boldsymbol{Q}$, which usually need not to be explicitly computed. Note that orthogonal transformations and unimodular transformations do not change the absolute value of the determinant of a matrix. Thus the product $r_{11}r_{22}\dots r_{nn}$ is a constant throughout the updating process. It is easy to show that $r_{11}r_{22}\dots r_{nn} = \sqrt{\det\left(\boldsymbol{A}^{\mathsf{T}}\boldsymbol{A}\right)} = \det(\boldsymbol{R})$.

In the LLL reduction, two types of unimodular matrices are used: one is the integer Gauss matrix and the other is the permutation matrix. In the following we introduce these two types of unimodular matrices and the corresponding operations when they are applied to $\boldsymbol{R}$.

Integer Gauss Transformations: An integer matrix is called an integer Gauss transformation (IGT) or an integer Gauss matrix if it has the following form

$$\boldsymbol{Z}_{ik} = \boldsymbol{I} - \zeta_{ik}\boldsymbol{e}_i\boldsymbol{e}_k^{\mathsf{T}}, \tag{2.5}$$

where ζ_{ik} is an integer and $i \neq k$.

Because $\boldsymbol{Z}_{ik}^{-1} = \boldsymbol{I} + \zeta_{ik}\boldsymbol{e}_i\boldsymbol{e}_k^{\mathsf{T}}$, it is easy to verify that $\boldsymbol{Z}_{ik}$ is unimodular. Applying $\boldsymbol{Z}_{ik}$ (with $i < k$) to $\boldsymbol{R}$ from the right gives

$$\bar{\boldsymbol{R}} = \boldsymbol{R}\boldsymbol{Z}_{ik} = \boldsymbol{R} - \zeta_{ik}\boldsymbol{R}\boldsymbol{e}_i\boldsymbol{e}_k^{\mathsf{T}}.$$

Thus $\bar{\boldsymbol{R}}$ is the same as $\boldsymbol{R}$, except that

$$\bar{r}_{jk} = r_{jk} - \zeta_{ik}r_{ji}, \quad j = 1, \ldots, i.$$

If we set $\zeta_{ik} = \lfloor r_{ik}/r_{ii} \rceil$, then applying $\boldsymbol{Z}_{ik}$ to $\boldsymbol{R}$ reduces $|r_{ik}|$. Specifically, we have $|\bar{r}_{ik}| \leq \bar{r}_{ii}/2$, i.e., the size-reduction condition (2.4a) is satisfied for this specific $\bar{r}_{ik}$ in $\bar{\boldsymbol{R}}$. Applying IGT $\boldsymbol{Z}_{ik}$ with ζ_{ik} defined above to $\boldsymbol{R}$ is called a size reduction on $|r_{ik}|$, and ζ_{ik} is called the size-reduction coefficient.

Column Permutations: A column permutation is to interchange two consecutive columns of $\boldsymbol{R}$. When columns $k-1$ and k of $\boldsymbol{R}$ are interchanged, the upper triangular structure is no longer maintained. But we can apply GSO to update rows $k-1$ and k of $\boldsymbol{R}$ to bring it back to an upper triangular matrix. This column permutation process can be described as

$$\bar{\boldsymbol{R}} = \boldsymbol{G}_{k-1,k}\boldsymbol{R}\boldsymbol{P}_{k-1,k}, \tag{2.6}$$

where $\boldsymbol{P}_{k-1,k}$ is the permutation matrix and $\boldsymbol{G}_{k-1,k}$ is the orthogonal matrix. In the original LLL algorithm, row scaling is used to avoid square root in the GSO. For convenience, when we use the term "column permutation", the whole column permutation and orthogonalization process (2.6) is usually implied. It

is easy to show that after the column permutation, we have

$$\begin{aligned} \bar{r}_{k-1,k-1} &= \sqrt{r_{k-1,k}^2 + r_{kk}^2}, \quad \bar{r}_{k-1,k} = \frac{r_{k-1,k-1} r_{k-1,k}}{\sqrt{r_{k-1,k}^2 + r_{kk}^2}}, \\ \bar{r}_{k,k} &= \frac{r_{k-1,k-1} r_{k,k}}{\sqrt{r_{k-1,k}^2 + r_{kk}^2}}. \end{aligned} \tag{2.7}$$

Note that $\boldsymbol{G}_{k-1,k}$ and $\boldsymbol{P}_{k-1,k}$ do not change the diagonal entries of $\boldsymbol{R}$, except for $r_{k-1,k-1}$ and $r_{k,k}$, and do not change the determinant of the upper triangular matrix.

In LLL, a column permutation operation is applied when the Lovász condition (2.4b) does not hold for some k, i.e. $\delta r_{k-1,k-1}^2 > r_{k-1,k}^2 + r_{kk}^2$. From (2.7) we have

$$\bar{r}_{k-1,k-1}^2 = r_{k-1,k}^2 + r_{kk}^2 < \delta r_{k-1,k-1}^2 = \delta(\bar{r}_{k-1,k}^2 + \bar{r}_{kk}^2). \tag{2.8}$$

Since $\delta \leq 1$, (2.8) indicates that the Lovász condition is satisfied for $(k-1)$-th and k-th diagonal entries of $\bar{\boldsymbol{R}}$, i.e., $\delta \bar{r}_{k-1,k-1}^2 < \bar{r}_{k-1,k}^2 + \bar{r}_{kk}^2$. And, in this case, it is easy to verify:

$$\bar{r}_{k-1,k-1} < r_{k-1,k-1}, \quad \bar{r}_{kk} > r_{kk}. \tag{2.9}$$

The LLL algorithm checks and updates the initial upper triangular matrix $\boldsymbol{R}$ column by column starting from column 2. Assume the LLL algorithm is currently working on column $\boldsymbol{r}_k$, it first checks the size-reduction condition (2.4a) on $r_{k-1,k}$. If the condition is not satisfied, then the LLL algorithm applies $\boldsymbol{Z}_{k-1,k}$ to reduce $r_{k-1,k}$. After that, the LLL algorithm checks the Lovász condition (2.4b). If the Lovász condition does not hold, it performs the column permutation of columns $k-1$ and k to ensure that the condition is satisfied and then, if $k > 2$, it goes back

to column $k-1$. Otherwise, the LLL algorithm checks the size-reduction condition (2.4a) for r_{ik} for $i = k-2, k-3, \ldots, 1$ and applies IGT $\boldsymbol{Z}_{ik}$ to reduce r_{ik} when (2.4a) is not satisfied. After the size reductions, it goes to column $k+1$. The LLL algorithm starts from $k = 2$ and repeats the above process until $k = n+1$. The pseudocode of the LLL reduction algorithm is given in Algorithm 2–1. When $\delta < 1$, both the original rational-arithmetic-based LLL algorithm [60] and the floating point arithmetic based LLL algorithm, see, e.g., [91], are polynomial time in the dimensions m and n and the bit-length of the entries of $\boldsymbol{A}$, see [96]. When $\delta = 1$ whether a polynomial time complexity can be reached is still unknown.

Algorithm 2–1 The LLL reduction

function $[\boldsymbol{R}, \boldsymbol{Z}] = \text{LLL}(\boldsymbol{A})$

apply GSO to obtain $\boldsymbol{A} = \boldsymbol{Q}\begin{bmatrix}\boldsymbol{R}\\ \boldsymbol{0}\end{bmatrix}$;
let $\boldsymbol{Z} = \boldsymbol{I}_n$, $k = 2$;
while $k \le n$ **do**
    apply size reduction $\boldsymbol{Z}_{k-1,k}$ to reduce $r_{k-1,k}$: $\boldsymbol{R} = \boldsymbol{R}\boldsymbol{Z}_{k-1,k}$;
    update $\boldsymbol{Z}$: $\boldsymbol{Z} = \boldsymbol{Z}\boldsymbol{Z}_{k-1,k}$;
    if $\delta r_{k-1,k-1}^2 > \left(r_{k-1,k}^2 + r_{k,k}^2\right)$ **then**
        perform column permutation: $\boldsymbol{R} = \boldsymbol{G}_{k-1,k}\boldsymbol{R}\boldsymbol{P}_{k-1,k}$;
        update $\boldsymbol{Z}$: $\boldsymbol{Z} = \boldsymbol{Z}\boldsymbol{P}_{k-1,k}$;
        if $k > 2$ **then**
            $k = k-1$
        end if;
    else
        for $i = k-2, \ldots, 1$ **do**
            apply size reduction $\boldsymbol{Z}_{ik}$ to reduce r_{ik}: $\boldsymbol{R} = \boldsymbol{R}\boldsymbol{Z}_{ik}$;
            update $\boldsymbol{Z}$: $\boldsymbol{Z} = \boldsymbol{Z}\boldsymbol{Z}_{ik}$;
        end for
        $k = k+1$;
    end if
end while

2.2 Sphere Decoding Algorithms

Let $\boldsymbol{z}^*$ be the optimal solution of (2.3). Suppose we have $\rho \in \mathbb{R}^+$ such that for $\boldsymbol{z} = \boldsymbol{z}^*$ the following inequality holds:

$$\|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2 < \rho^2. \tag{2.10}$$

Note that (2.10) represents an n-dimensional hyper-ellipsoid in terms of $\boldsymbol{z} \in \mathbb{R}^n$, or a hyper-sphere in which the lattice point $\boldsymbol{R}\boldsymbol{z}$ lies. To find the optimal solution for (2.3), a sphere decoder enumerates (part of) integer points within region (2.10). The parameter ρ, which is called the radius of the search region, is crucial to the performance of sphere decoders. On one hand, if ρ is too large, then the cost of sphere decoding would be expensive. On the other hand, if ρ is too small, the optimal solution might be excluded from (2.10). Methods for choosing ρ can be found in, e.g., [105] and [78].

We partition $\bar{\boldsymbol{y}}$, $\boldsymbol{R}$ and $\boldsymbol{z}$ such that (2.10) becomes

$$\left\| \begin{bmatrix} \hat{\boldsymbol{y}} \\ \bar{y}_n \end{bmatrix} - \begin{bmatrix} \hat{\boldsymbol{R}} & \hat{\boldsymbol{r}} \\ & r_{nn} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{z}} \\ z_n \end{bmatrix} \right\|_2^2 < \rho^2, \tag{2.11}$$

where $\hat{\boldsymbol{R}} \in \mathbb{R}^{(n-1)\times(n-1)}$, $\hat{\boldsymbol{y}}, \hat{\boldsymbol{r}} \in \mathbb{R}^{n-1}$, and $\hat{\boldsymbol{z}} \in \mathbb{Z}^{n-1}$. From (2.11) it follows that

$$(\bar{y}_n - r_{nn} z_n)^2 < \rho^2, \tag{2.12a}$$

$$\left\| (\hat{\boldsymbol{y}} - \hat{\boldsymbol{r}} z_n) - \hat{\boldsymbol{R}}\hat{\boldsymbol{z}} \right\|_2^2 < \rho^2 - (\bar{y}_n - r_{nn} z_n)^2. \tag{2.12b}$$

A sphere decoder enumerates z_n for all possible values that satisfy (2.12a). Then for each of these values of z_n, it enumerate possible values of $\hat{\boldsymbol{z}}$ according to (2.12b). Notice that (2.12b) is an $(n-1)$-dimensional hyper-ellipsoid when z_n is known. Because

(2.12b) and (2.10) have the same structure, the enumeration of $\hat{\boldsymbol{z}}$ in (2.12b) can be handled recursively as an $(n-1)$-dimensional subproblem. When the enumeration recursion reaches dimension 1 and finds a valid value for z_1, then an integer point $\boldsymbol{z}$ inside the hyper-ellipsoid (2.10) is found. This integer point $\boldsymbol{z}$ is called a candidate solution of (2.3). The enumeration process stops when no more values of z_n can be found that satisfies (2.12a). After that, among all of the candidate solutions we found in the enumeration process, we choose the one that minimizes (2.3) as the optimal solution.

In the enumeration process, we can update the search radius ρ to $\|\boldsymbol{y}-\boldsymbol{R}\boldsymbol{z}\|_2$ every time a candidate solution $\boldsymbol{z}$ is found, and let the enumeration process continue with this smaller ρ. This ρ updating strategy serves to shrink the search region in the search process and it greatly improves the search efficiency. We refer to this strategy as the shrinking strategy. When the shrinking strategy is employed, the initial value for ρ can be set to ∞. The last candidate solution $\boldsymbol{z}$ found in the enumeration is the optimal solution.

In the recursive process of sphere decoding described above, we define the enumeration process of z_k as the k-th level enumeration. It can be seen that in the k-th level of the enumeration process, every z_k satisfies

$$\left(\bar{y}_k - \sum_{j=k+1}^{n} r_{kj}z_j - r_{kk}z_k\right)^2 < \rho^2 - \sum_{j=k+1}^{n}\left(\bar{y}_j - \sum_{i=j}^{n} r_{ji}z_i\right)^2, \tag{2.13}$$

where the value of z_j is already fixed in the j-th level enumeration for all $j = n, n-1, \ldots, k+1$. To simplify the above inequality, we define $c_n = \bar{y}_n / r_{nn}$ and

$$c_k = \left(\bar{y}_k - \sum_{j=k+1}^{n} r_{kj} z_j \right) / r_{kk}, \quad k = n-1, n-2, \ldots, 1. \tag{2.14}$$

Then, (2.13) becomes

$$r_{kk}^2 \left(z_k - c_k \right)^2 < \rho^2 - \sum_{j=k+1}^{n} r_{jj}^2 \left(z_j - c_j \right)^2 . \tag{2.15}$$

Note (2.12a) is just (2.15) with $k = n$. The vector $|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}|$ [1] is called the residual vector of $\boldsymbol{z}$. It can be seen that the left hand side of (2.15) is the square of the k-th entry of the residual vector, i.e., $\left(\boldsymbol{e}_k^{\mathsf{T}} (\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}) \right)^2$.

Note that the left hand side of (2.15) increases if z_k moves away from c_k. In Schnorr-Euchner's sphere decoding algorithm [89], the enumeration of z_k is executed in the following specified order until (2.15) does not hold any more:

$$z_k = \begin{cases} \lfloor c_k \rceil, \lfloor c_k \rceil - 1, \lfloor c_k \rceil + 1, \lfloor c_k \rceil - 2, \ldots, & \text{if } \lfloor c_k \rceil \geq c_k \\ \lfloor c_k \rceil, \lfloor c_k \rceil + 1, \lfloor c_k \rceil - 1, \lfloor c_k \rceil + 2, \ldots, & \text{if } \lfloor c_k \rceil < c_k \end{cases} .$$

In this order, the enumeration of z_k starts from the value that minimizes the left-hand side of (2.15). This maximizes the chance of the point found early being optimal. This can greatly benefit the search efficiency if the shrinking strategy is used [2]. When the initial value for ρ is set to ∞, the first integer candidate found

[1] $|\boldsymbol{x}|$ denotes the absolute vector of $\boldsymbol{x}$, i.e., $|\boldsymbol{x}| = (|x_i|)_n$.

in Schnorr-Euchner's algorithm is

$$\boldsymbol{z}^{\mathrm{B}} = \begin{bmatrix} \lfloor c_1 \rceil & \lfloor c_2 \rceil & \cdots & \lfloor c_n \rceil \end{bmatrix}^{\mathsf{T}}, \tag{2.16}$$

which is exactly the Babai point produced by Babai's nearest plane algorithm [10].

It can be seen that sphere decoding is actually a depth-first search in a tree of depth $(n+1)$. We let the root be at level $n+1$ of the search tree, and let the children of the level-$(k+1)$ tree nodes be at level k, see Figure 2–1 for an example where $n = 3$. Then, each node at level k of the search tree corresponds to a value assigned to z_k in the k-th level sphere decoding enumeration, except the root of the tree which is an artificial node in the sense that it does not correspond to any actual step in the sphere decoding,

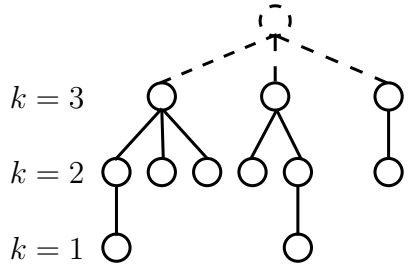


Figure 2–1: Search tree

2.3 The LLL Reduction and the Cost of Sphere Decoding

The complexity results of sphere decoders given in the literature are often about the cost of enumerating all integer points in the search region $\|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2 < \rho^2$, see (2.10). For simplicity, in the complexity analysis it is assumed that the shrinking strategy is not used in the search process. A typical measure of the cost is the number

of nodes enumerated by sphere decoders, which is just the size of the search tree, which we denote by η.

Recall that nodes at level k of the search tree correspond to the values of z_k satisfying (2.13) for different values of z_j for $j = k+1, k+1, \ldots, n$. Thus the total number of tree nodes at level k is the total number of combinations of $z_k, z_{k-1}, \ldots, z_n$ that satisfies (2.13). Inequality (2.13) can be rewritten as

$$\|\bar{\boldsymbol{y}}_{k:n} - \boldsymbol{R}_{k:n,k:n}\boldsymbol{z}_{k:n}\|_2^2 < \rho^2. \tag{2.17}$$

Thus the number of node at level k of the search tree is equal to the number of lattice points in the $(n-k+1)$-dimensional hyper-sphere defined by (2.17). For $k = n, n-1, \ldots, 1$, define E_k as the number of integer points satisfying (2.17) (and thus E_k is also the number of nodes at level k of the search tree), i.e.,

$$E_k = \left|\{\boldsymbol{z}_{k:n} \in \mathbb{Z}^{n-k+1} : \|\bar{\boldsymbol{y}}_{k:n} - \boldsymbol{R}_{k:n,k:n}\boldsymbol{z}_{k:n}\|_2 < \rho\}\right|,$$

where $|\cdot|$ denotes the number of elements in the set.

Via the Gaussian volume heuristic, the number of lattice points in a hyper-sphere, E_k, can be approximated by the ratio of the volume of the hyper-sphere and the volume of the Voroni cell of the lattice, see, e.g., [84]. Thus we have:

$$E_k \approx \frac{V_{n-k+1}\,\rho^{n-k+1}}{|\det(\boldsymbol{R}_{k:n,k:n})|} = \frac{V_{n-k+1}\,\rho^{n-k+1}}{r_{ii}r_{k+1,k+1}\ldots r_{nn}},$$

where V_{n-k+1} denotes the volume of an $(n-k+1)$-dimensional unit Euclidean ball. This approximation becomes the expected value to E_k if $\bar{\boldsymbol{y}}_{k:n}$ is uniformly distributed over some Voroni cells of the lattice generated by $\boldsymbol{R}_{k:n,k:n}$ (see [92]). Then we have

(see, e.g., [1, Sec 3.2] and [93]):

$$\eta = \sum_{k=1}^{n} E_k \approx \bar{\eta}(\boldsymbol{R}) \equiv \sum_{k=1}^{n} \frac{V_{n-k+1}\,\rho^{n-k+1}}{r_{ii} r_{k+1,k+1} \cdots r_{nn}}.$$

To make the search process work, we can simply use the QR factorization of $\boldsymbol{A}$ to transform (2.1) to (2.3), i.e., letting $\boldsymbol{Z} = \boldsymbol{I}$ in the QRZ factorization (2.2). Even though it has been well known that using the LLL reduction, instead of the QR factorization, in the transformation can make the sphere decoding faster (see e.g., [44, 2, 65]), there was no theoretical proof given in the literature until we published [25]. In this section, we rigorously show that transforming an ILS problem using the LLL reduction algorithm given in Algorithm 2–1 reduces the computational cost of sphere decoding for solving it, which is measured approximately by $\bar{\eta}$.

Suppose (2.1) is transformed to (2.3) by the QR factorization of $\boldsymbol{A}$, i.e., $\boldsymbol{Z} = \boldsymbol{I}$ in (2.2). Let the search region be

$$\|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2 < \rho^2. \tag{2.18}$$

And suppose we have the QRZ factorization $\bar{\boldsymbol{R}} = \bar{\boldsymbol{Q}}^{\mathsf{T}} \boldsymbol{R} \bar{\boldsymbol{Z}}$, where $\bar{\boldsymbol{Q}}$ is orthogonal and $\bar{\boldsymbol{Z}}$ is unimodular (from here until the end of this section, we do not assume that $\boldsymbol{R}$ is LLL reduced unless we state otherwise). Then with $\hat{\boldsymbol{y}} = \bar{\boldsymbol{Q}}^{\mathsf{T}} \bar{\boldsymbol{y}}$ and $\bar{\boldsymbol{z}} = \bar{\boldsymbol{Z}}^{-1} \boldsymbol{z}$, after the reduction, the search region (2.18) becomes

$$\left\|\hat{\boldsymbol{y}} - \bar{\boldsymbol{R}}\bar{\boldsymbol{z}}\right\|_2^2 < \rho^2. \tag{2.19}$$

As stated before, the cost of enumerating (2.18) and (2.19) can be measured approximately by $\bar{\eta}(\boldsymbol{R})$ and $\bar{\eta}(\bar{\boldsymbol{R}})$. In the following we look at how $\bar{\eta}(\boldsymbol{R})$ changes after some specific transformation of $\boldsymbol{R}$.

The following result shows that if the Lovász condition (2.4b) is not satisfied for some k, the cost $\bar{\eta}(\boldsymbol{R})$ decreases after the column permutation of columns $k-1$ and k.

Lemma 2–1. *Suppose that* $\delta r_{k-1,k-1}^2 > r_{k-1,k}^2 + r_{kk}^2$ *for matrix* $\boldsymbol{R}$ *in* (2.18) *and some* k. *After the permutation of columns* $k-1$ *and* k, $\boldsymbol{R}$ *becomes* $\bar{\boldsymbol{R}}$, *i.e.,* $\bar{\boldsymbol{R}} = \boldsymbol{G}_{k-1,k}\boldsymbol{R}\boldsymbol{P}_{k-1,k}$ (see (2.6)). *Then the cost* $\bar{\eta}(\boldsymbol{R})$ *of the search process decreases after the transformation, i.e.,*

$$\bar{\eta}(\boldsymbol{R}) > \bar{\eta}(\bar{\boldsymbol{R}}).$$

Proof. Since $\bar{r}_{ii} = r_{ii}$ for $i \neq k-1, k$, $\bar{r}_{k-1,k-1}\bar{r}_{kk} = r_{k-1,k-1}r_{kk}$, and $\bar{r}_{kk} > r_{kk}$ (see (2.9)) we have

$$\begin{aligned}
\bar{\eta}(\boldsymbol{R}) - \bar{\eta}(\bar{\boldsymbol{R}}) &= \sum_{i=1}^{n} \frac{V_{n-i+1}\,\rho^{n-i+1}}{r_{ii}r_{i+1,i+1}\dots r_{nn}} - \sum_{i=1}^{n} \frac{V_{n-i+1}\,\rho^{n-i+1}}{\bar{r}_{ii}\bar{r}_{i+1,i+1}\dots \bar{r}_{nn}} \\
&= \frac{V_{n-k+1}\,\rho^{n-k+1}}{r_{kk}r_{k+1,k+1}\dots r_{nn}} - \frac{V_{n-k+1}\,\rho^{n-k+1}}{\bar{r}_{kk}r_{k+1,k+1}\dots r_{nn}} \\
&= \left(\frac{1}{r_{kk}} - \frac{1}{\bar{r}_{kk}}\right) \frac{V_{n-k+1}\,\rho^{n-k+1}}{r_{k+1,k+1}\dots r_{nn}} > 0,
\end{aligned}$$

completing the proof. □

We can see that applying size reductions alone does not change the diagonal entries of $\boldsymbol{R}$ and thus does not change $\bar{\eta}(\boldsymbol{R})$. Suppose the Lovász condition (2.4b) does not hold for a specific k and furthermore $|r_{k-1,k}| > r_{k-1,k-1}/2$. The next lemma

shows that the size reduction on $r_{k-1,k}$ performed before the permutation operation can help to decrease the cost $\bar{\eta}(\boldsymbol{R})$ further.

Lemma 2–2. *Suppose that in* (2.18)*, matrix* $\boldsymbol{R}$ *satisfies* $\delta r_{k-1,k-1}^2 > r_{k-1,k}^2 + r_{kk}^2$ *and* $|r_{k-1,k}| > r_{k-1,k-1}/2$ *for some* k*. Let* $\bar{\boldsymbol{R}}$ *be defined as in Lemma 2–1. Suppose a size reduction on* $r_{k-1,k}$ *is performed first and then after the permutation of columns* $k-1$ *and* k *and triangularization,* $\boldsymbol{R}$ *becomes* $\hat{\boldsymbol{R}}$*, i.e.,* $\hat{\boldsymbol{R}} = \hat{\boldsymbol{G}}_{k-1,k}\boldsymbol{R}\boldsymbol{Z}_{k-1,k}\boldsymbol{P}_{k-1,k}$*. Then*

$$\bar{\eta}(\bar{\boldsymbol{R}}) > \bar{\eta}(\hat{\boldsymbol{R}}). \tag{2.20}$$

Proof. By the same argument given in the proof of Lemma 2–1, we have

$$\bar{\eta}(\bar{\boldsymbol{R}}) - \bar{\eta}(\hat{\boldsymbol{R}}) = \left(\frac{1}{\bar{r}_{kk}} - \frac{1}{\hat{r}_{kk}}\right) \frac{V_{n-k+1}\,\rho^{n-k+1}}{r_{k+1,k+1}\cdots r_{nn}}.$$

To show (2.20) we need only to prove $\bar{r}_{kk} < \hat{r}_{kk}$. Suppose that after the size reduction, $r_{k-1,k}$ becomes $\tilde{r}_{k-1,k}$. We have $|\tilde{r}_{k-1,k}| \le |r_{k-1,k}|$. From (2.7), we have

$$\hat{r}_{k-1,k-1} = \sqrt{\tilde{r}_{k-1,k}{}^2 + r_{kk}^2} < \sqrt{r_{k-1,k}^2 + r_{kk}^2} = \bar{r}_{k-1,k-1}.$$

Since $\bar{r}_{k-1,k-1}\bar{r}_{kk} = \hat{r}_{k-1,k-1}\hat{r}_{kk}$ and $\hat{r}_{k-1,k-1} < \bar{r}_{k-1,k-1}$, we have $\bar{r}_{kk} < \hat{r}_{kk}$, completing the proof. □

From Lemmas 2–1 and 2–2 we immediately obtain the following result.

Theorem 1. *Suppose that* $\bar{\boldsymbol{R}}$ *is obtained by applying the LLL reduction algorithm (Algorithm 2–1) to* $\boldsymbol{R}$*, then*

$$\bar{\eta}(\boldsymbol{R}) \ge \bar{\eta}(\bar{\boldsymbol{R}}),$$

where the equality holds if and only if no column permutation occurs during the LLL reduction process. Any size reductions on the superdiagonal entries of $\boldsymbol{R}$ *which is immediately followed by a column permutation during the LLL reduction process will reduce* $\bar{\eta}$*. All other size reductions have no effect on* $\bar{\eta}$*.*

Now we make some remarks on Theorem 1. Recall that in the LLL reduction (Algorithm 2–1), when the Lovász condition for two consecutive columns is not satisfied, then a column permutation takes places to ensure the Lovász condition to be satisfied. From Lemma 2–1, this column permutation strategy ensures that that the LLL reduction given in Algorithm 2–1 always reduces $\bar{\eta}$. If there is another reduction algorithm that LLL reduces the given matrices, but performs column permutations differently, e.g., the algorithm permutes two columns that are not consecutive or permutes two consecutive columns but the corresponding Lovász condition is not satisfied after the permutation, then we cannot guarantee this specific reduction algorithm will reduce $\bar{\eta}$ even though the resulting matrix is LLL reduced.

Note that size reductions do not change the diagonal entries of $\boldsymbol{R}$ and thus do not affect $\bar{\eta}$. This result is consistent with a result we gave in [109], which shows that all the size reductions on the off-diagonal entries above the superdiagonal of $\boldsymbol{R}$ and the size reductions on the superdiagonal entries of $\boldsymbol{R}$ which are not followed by column permutations do not have any effect on the search speed of the sphere decoders. This observation leads to the partial LLL (PLLL) algorithm, which avoids some unnecessary size reductions in the LLL reduction and has good numerical stability.

The result given here is also consistent with [7] where we showed what the reduction process should try to achieve to make the sphere decoding fast and corrected

some common misconceptions in the literature. For example, we explained why neither decreasing correlation coefficients of real least squares (RLS) estimates of the integer parameter vector $\boldsymbol{x}$ nor decreasing the condition number of the covariance matrix of the RLS estimate should be an objective of the reduction process.

2.4 Effects of δ on the Cost of Sphere Decoding

We know that with different values of δ, the LLL reduction algorithm given in Algorithm 2–1 generates different δ-LLL reduced matrices $\boldsymbol{R}$ for the same matrix $\boldsymbol{A}$. Suppose $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ are obtained by applying Algorithm 2–1 to $\boldsymbol{A}$ with $\delta = \delta_1$ and $\delta = \delta_2$ respectively, and $\delta_1 < \delta_2$. A natural question to ask is: what is the relation between $\bar{\eta}(\boldsymbol{R}_1)$ and $\bar{\eta}(\boldsymbol{R}_2)$? In the following, we try to address this question. First, we give the result for $n = 2$.

Theorem 2. *Suppose $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ are obtained by applying Algorithm 2–1 to $\boldsymbol{A}$ with $\delta = \delta_1$ and $\delta = \delta_2$ respectively, and $\delta_1 < \delta_2$. If $n = 2$, then*

$$\bar{\eta}(\boldsymbol{R}_1) \geq \bar{\eta}(\boldsymbol{R}_2). \tag{2.21}$$

Proof. Note that only two columns are involved in the reduction process and the value of δ only determines when the process should terminate. In the reduction process, the upper triangular matrix either first becomes δ_1-LLL reduced and then becomes δ_2-LLL reduced after some more permutations or becomes δ_1-LLL reduced and δ_2 -LLL reduced at the same time. Therefore, by Lemma 2–1, the conclusion holds. □

Now, let us consider the cases when $n \geq 3$. Note that when $\delta_1 < \delta_2$, $\boldsymbol{R}_2$ must be δ_1-LLL reduced, but $\boldsymbol{R}_1$ may not be δ_2-LLL reduced. If $\boldsymbol{R}_2$ can be obtained by

applying the Algorithm 2–1 (with $\delta = \delta_2$) to $\boldsymbol{R}_1$, then, by Theorem 1, we have (2.21). Otherwise, (2.21) may not hold. We use an example to illustrate this.

Example 2–1. *Let*

$$\boldsymbol{A} = \begin{bmatrix} 0.9675 & 0.4328 & 0.0935 & 0.9477 \\ & 0.5879 & 0.6792 & 0.4456 \\ & & 0.4295 & 0.0549 \\ & & & 0.0853 \end{bmatrix}. \tag{2.22}$$

Applying Algorithm 2–1 to $\boldsymbol{A}$ *with* $\delta = \delta_1 = 4/9$ *and* $\delta = \delta_2 = 25/36$*, we obtain the* δ_1*-LLL reduced* $\boldsymbol{R}_1$ *and the* δ_2*-LLL reduced* $\boldsymbol{R}_2$*, respectively:*

$$\boldsymbol{R}_1 = \begin{bmatrix} 0.4852 & 0.0678 & -0.0232 & -0.1236 \\ & 0.3485 & 0.0578 & -0.1235 \\ & & 0.3413 & -0.0990 \\ & & & 0.3612 \end{bmatrix},$$

$$\boldsymbol{R}_2 = \begin{bmatrix} 0.5549 & -0.2591 & 0.1280 & -0.0001 \\ & 0.3845 & -0.1278 & -0.0855 \\ & & 0.2960 & -0.0996 \\ & & & 0.3299 \end{bmatrix}.$$

Note that $\boldsymbol{R}_1$ *is not* δ_2*-LLL reduced. Applying Algorithm 1 to* $\boldsymbol{R}_1$ *with* $\delta = \delta_2$*, we obtain* δ_2*-LLL reduced* $\hat{\boldsymbol{R}}_2$*:*

$$\hat{\boldsymbol{R}}_2 = \begin{bmatrix} 0.3550 & 0.0523 & -0.1448 & 0.0926 \\ & 0.3429 & -0.0889 & -0.0470 \\ & & 0.3767 & -0.1346 \\ & & & 0.4544 \end{bmatrix}.$$

In Table 2–1, we give the values of $\bar{\eta}(\boldsymbol{R}_1)$*,* $\bar{\eta}(\boldsymbol{R}_2)$*,* $\bar{\eta}(\hat{\boldsymbol{R}}_1)$ *with different values of* ρ*. It can be seen that* $\bar{\eta}(\boldsymbol{R}_2) > \eta(\bar{\boldsymbol{R}}_1) > \bar{\eta}(\hat{\boldsymbol{R}}_2)$ *for different choices of* ρ*.*

To give a general n-dimensional example, define $\boldsymbol{A}^{(n)} = \begin{bmatrix} \alpha \boldsymbol{I}_{n-3} & \\ & \boldsymbol{A} \end{bmatrix}$ with $\boldsymbol{A}$ given in (2.22). When $|\alpha|$ is small enough, the LLL reduction will not change the

Table 2–1: Sphere decoding cost $\bar{\eta}$ versus search radius ρ for Example 2–1

ρ	0.2500	0.5000	0.7500	1.0000	1.2500
$\bar{\eta}(\boldsymbol{R}_1)$	5.4264	36.1325	134.5629	365.3632	815.3802
$\bar{\eta}(\boldsymbol{R}_2)$	6.1943	39.8179	144.6296	386.5897	853.8591
$\bar{\eta}(\hat{\boldsymbol{R}}_2)$	4.2876	30.5105	118.6606	330.9310	751.7156

first $n-3$ rows and columns of $\boldsymbol{A}^{(n)}$. In this case, it is easy to see that we still have $\bar{\eta}\big(\boldsymbol{R}_2^{(n)}\big) > \bar{\eta}\big(\boldsymbol{R}_1^{(n)}\big)$ where $\boldsymbol{R}_1^{(n)}$ and $\boldsymbol{R}_2^{(n)}$ are obtained by applying Algorithm 2–1 to $\boldsymbol{A}$ with $\delta = 4/9$ and $\delta = 25/36$ respectively.

Although the aforementioned example shows that larger δ may not guarantee to produce smaller $\bar{\eta}$ when $n \geq 3$. With a larger δ, we can expect that the chance of getting a smaller $\bar{\eta}$ is much higher than getting a larger $\bar{\eta}$. Here, we give some numerical tests to show in general, how will δ affect $\bar{\eta}$. We consider two cases where the matrix $\boldsymbol{A}$ are generated differently, as explained below.

- Case 1: $\boldsymbol{A} = \text{randn}(n, n)$, where $\text{randn}(n, n)$ is a MATLAB built-in function to generate a random $n \times n$ matrix with independent, identically distributed (i.i.d.) entries following a normal distribution $\mathcal{N}(0, 1)$, i.e., with zero-mean and unit variance.
- Case 2 (moderately ill conditioned): we first generate $\boldsymbol{A}_0 = \text{randn}(n, n)$ and compute the singular value decomposition $\boldsymbol{A}_0 = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathsf{T}}$. Then, we construct diagonal matrix $\bar{\boldsymbol{D}}$ with $\bar{d}_{11} = 15$, $\bar{d}_{nn} = 0.005$ and $\bar{d}_{ii} = d_{ii}$ for $i = 2, 3, \ldots, n-1$. Finally, we form $\boldsymbol{A} = \boldsymbol{U}\bar{\boldsymbol{D}}\boldsymbol{V}^{\mathsf{T}}$. The condition number of $\boldsymbol{A}$ is at least 3000.

In the tests for each case, we use 1000 runs to generate 1000 different matrices $\boldsymbol{A}$ for a fixed dimension $n = 30$. Figure 2–2 and 2–3 represent the average $\bar{\eta}$ over 1000

runs against different values of δ for cases 1 and 2, respectively. The three curves in the figures correspond to different values of ρ.

From Figure 2–2 and 2–3, we can see that the value of δ in the LLL reduction has a significant effect on $\bar{\eta}$. Figure 2–2 and 2–3 show that as δ increases, on average $\bar{\eta}$ decreases almost exponentially. But we point out that for tests in both case 1 and case 2, sometimes a larger δ results in a larger $\bar{\eta}$ in the tests. This phenomenon is more often observed in case 2. Table 2–2 gives the number of times out of those 1000 runs in which $\bar{\eta}$ increases when δ increases from t to $t + 0.1$ for $t = 0.3 : 0.1 : 0.9$. In our tests, we tried various dimension sizes n and other types of $\boldsymbol{A}$, and observed the same phenomenon.

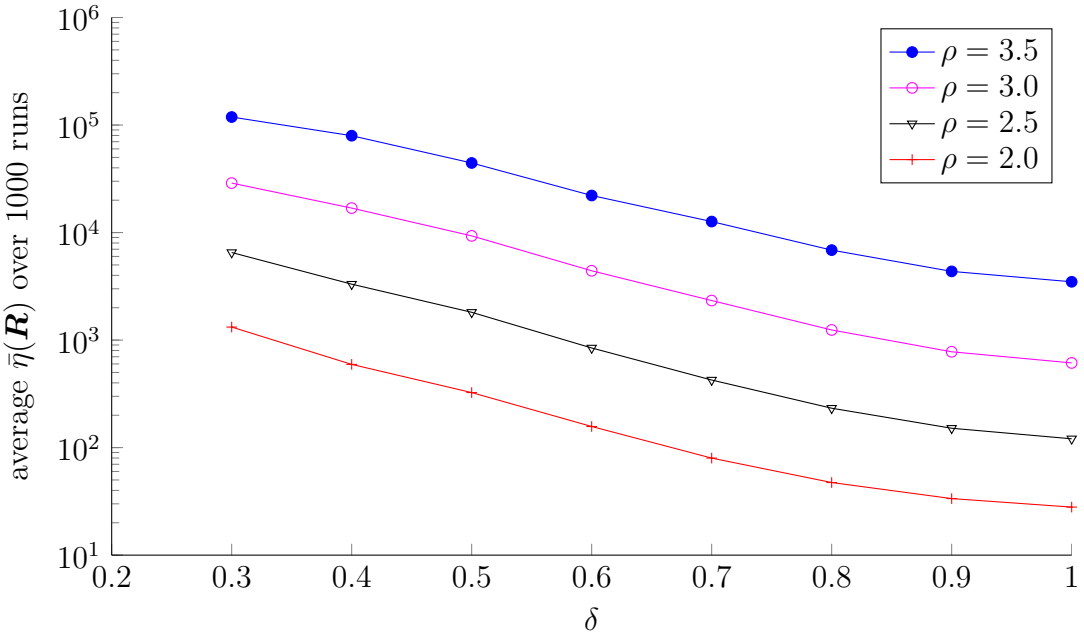


Figure 2–2: Case 1: Average $\bar{\eta}$ after the δ-LLL reduction

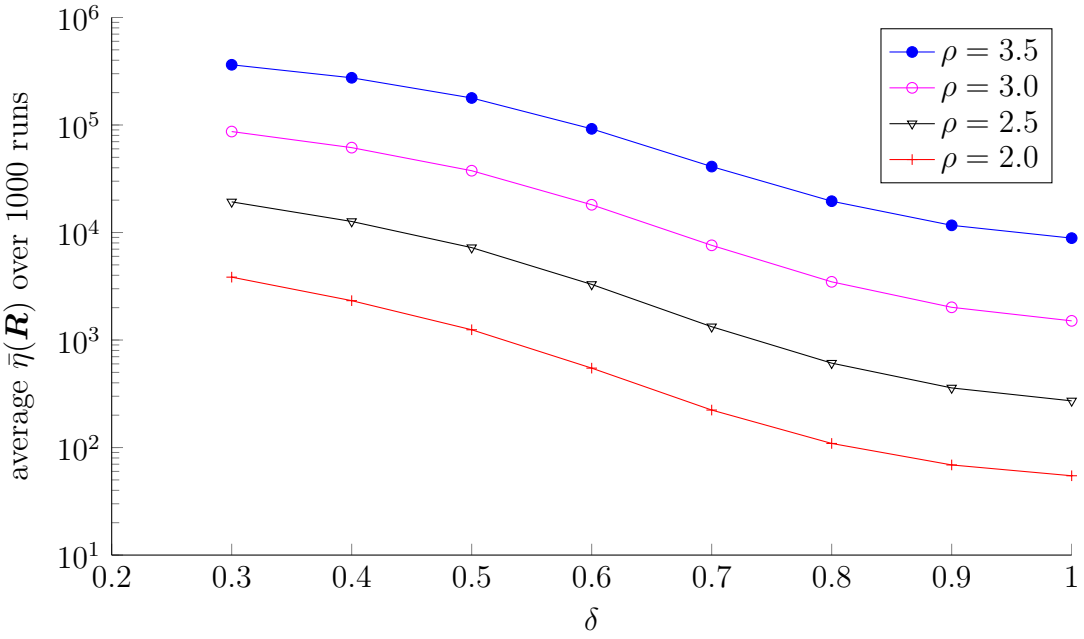


Figure 2–3: Case 2: Average $\bar{\eta}$ after the δ-LLL reduction

Table 2–2: Number of runs in which $\bar{\eta}$ increases when δ increases

	Case 1				Case 2			
δ \ ρ	2.0	2.5	3.0	3.5	2.0	2.5	3.0	3.5
0.3–0.4	21	22	24	24	77	86	87	92
0.4–0.5	42	46	48	56	91	97	103	108
0.5–0.6	64	64	67	69	108	108	113	116
0.6–0.7	53	61	65	66	96	97	96	98
0.7–0.8	50	50	52	63	101	99	90	87
0.8–0.9	60	60	63	70	92	85	87	92
0.9–1.0	53	56	62	72	90	85	89	102

CHAPTER 3
Box-Constrained Integer Least Squares Problems

In this chapter, we consider the box-constrained integer least squares (BILS) problems. The sphere decoding approach for the OILS problems we described in Section 2.2 can easily be modified to solve the BILS problems, and this approach is commonly used in practice (see, e.g., [24]). In Lemma 2–2, we showed that size reductions can help to decrease the cost of sphere decoding. However, size reductions are usually not used in the reduction process for a BILS problem because they tend to make the constraints of the problem too complicated to handle in sphere decoding (see, e.g., [108, 30, 98, 24, 20]). In this chapter, by introducing the concept of inactive constraints, we propose a novel reduction algorithm for BILS problems, which is able to incorporate size reductions without causing any difficultly to sphere decoding. The main results obtained in this chapter will appear in [27].

3.1 Introduction

A BILS problem has the following form:

$$\min_{\boldsymbol{x}\in\mathcal{B}} \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}\|_2^2, \quad \mathcal{B}=\{\boldsymbol{x} \mid \boldsymbol{x}\in\mathbb{Z}^n;\ \boldsymbol{u}\le\boldsymbol{x}\le\boldsymbol{l}\}, \tag{3.1}$$

where $\boldsymbol{A}\in\mathbb{R}^{m\times n}$ is a given matrix, $\boldsymbol{y}\in\mathbb{R}^m$ is a given vector, and $\boldsymbol{l},\boldsymbol{u}\in\mathbb{Z}^n$ with $\boldsymbol{l}<\boldsymbol{u}$ are given integer vectors which define the boundaries of $\boldsymbol{x}$. Here, $\mathcal{B}$ is a box. (In geometry, a box refers to a rectangular parallelotope with edges parallel to the coordinate axes. We abuse the notation a little bit here to refer to the set of integer

points in a conventional box). In this Chapter, we consider only the case that $\boldsymbol{A}$ has full column rank. And in this case, (3.1) is also referred to as an overdetermined BILS problem.

3.1.1 Reduction algorithms for BILS problems

Unlike the LLL reduction for OILS problems that uses both size reductions and column permutations (see Section 2.1 for details), the reduction process for BILS problems usually does not involve size reductions because the geometry of the constraint box would become complicated to make the search process difficult. Generally, applying a unimodular matrix $\boldsymbol{Z}$ which involves size reductions transforms the shape of the domain from a box $\{\boldsymbol{x} \mid \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}\}$ to a skewed parallelotope $\{\boldsymbol{z} \mid \boldsymbol{l} \leq \boldsymbol{Z}\boldsymbol{z} \leq \boldsymbol{u}\}$. In the process of sphere decoding for solving BILS (see later), it is easy to enumerate x_k in $[l_k, u_k]$, but it is much more difficult to enumerate z_k because all entries of $\boldsymbol{z}$ are entangled.

Thus, reduction strategies in the literature for BILS problems usually only do column reordering. In other words, for a BILS problem (3.1), the unimodular matrix $\boldsymbol{Z}$ in the QRZ factorization (2.2) is chosen to be a permutation matrix $\boldsymbol{P} \in \mathbb{R}^{n\times n}$, so we have

$$\boldsymbol{A}\boldsymbol{P} = \boldsymbol{Q}_1\boldsymbol{R}. \tag{3.2}$$

Define

$$\bar{\boldsymbol{y}} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{y}, \quad \boldsymbol{z} = \boldsymbol{P}^{\mathsf{T}}\boldsymbol{x}, \quad \bar{\boldsymbol{l}} = \boldsymbol{P}^{\mathsf{T}}\boldsymbol{l}, \quad \bar{\boldsymbol{u}} = \boldsymbol{P}^{\mathsf{T}}\boldsymbol{u}. \tag{3.3}$$

Then problem (3.1) can be transformed to

$$\min_{\boldsymbol{z}\in\bar{\mathcal{B}}} \|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2, \quad \bar{\mathcal{B}} = \big\{\boldsymbol{z} \mid \boldsymbol{z} \in \mathbb{Z}^n;\ \bar{\boldsymbol{l}} \leq \boldsymbol{z} \leq \bar{\boldsymbol{u}}\big\}. \tag{3.4}$$

After we get the solution $\boldsymbol{z}^*$ of (3.4), we can then get the solution $\boldsymbol{x}^*$ of (3.1) using $\boldsymbol{x}^* = \boldsymbol{P}\boldsymbol{z}^*$.

Different column reordering strategies for BILS problems can be found in [108, 30, 98, 24, 20]. In [30], the well-known V-BLAST (vertical Bell laboratories layered space-time) strategy originally introduced in [37] for getting good performance of the Babai point was proposed to reduce (3.1) for improving the efficiency of sphere decoding. The V-BLAST strategy determines the columns of the final $\boldsymbol{R}$ from last to first. When determining column k, V-BLAST chooses the column that maximizes r_{kk} from the columns that have not been chosen before. In [24], the sorted QR decomposition (SQRD) algorithm originally introduced in [108] was also proposed to determine $\boldsymbol{P}$. In contrast to V-BLAST, SQRD determines the columns of the final $\boldsymbol{R}$ from first to last. When determining column k, it seeks a column from the not-yet-chosen columns to minimize r_{kk}. Both SQRD and V-BLAST use only the information in $\boldsymbol{A}$, and both of them can be computed efficiently in $O(n^3)$ flops (SQRD costs less than V-BLAST). However the V-BLAST reduction is more effective in reducing the cost of sphere decoding, see, e.g., [25].

In [98], Su and Wassell considered the geometry of the BILS problem and, from this point of view, proposed a new column reordering algorithm which uses not only the information in $\boldsymbol{A}$, but also the information in $\boldsymbol{y}$ and $\mathcal{B}$ of the BILS problem (3.1). Independently, in [24] Chang and Han proposed another column reordering algorithm. Their algorithm also uses all information of (3.1) and the derivation is based on an algebraic point of view. In [20], Breen and Chang showed that the reduction strategy given in [98] is essentially the same as the one given in [24], in

the sense that they will always give the same column reordering. The cost of the former is less than the cost of the later if the former is implemented in an efficient way. In [20], another reduction algorithm (which will be referred to as the all-information based permutation (AIP) algorithm for the sake of convenience) that is mathematically equivalent to the above two reduction algorithms but faster was proposed by combining the best part from each of the two reduction algorithms. This approach is more sophisticated than SQRD and V-BLAST and provides the best known result in speeding up sphere decoding for solving BILS problems.

3.1.2 The AIP reduction algorithm

In the following, we briefly review the process of the AIP reduction which will be used in this chapter. The AIP reduction first computes the QRP factorization (3.2) with initial $\boldsymbol{P} = \boldsymbol{I_n}$, getting initial $\boldsymbol{R}$, $\bar{\boldsymbol{y}}$, $\bar{\boldsymbol{l}}$ and $\bar{\boldsymbol{u}}$ in (3.4). The AIP reduction then determines a new order for the columns of $\boldsymbol{R}$ from right to left and returns the AIP reduced BILS problem and the updated permutation matrix $\boldsymbol{P}$. In the following, we describe how the last column $\boldsymbol{r}_n$ is determined in the AIP reduction. Subsequent steps of determining other columns are the same but are applied to subproblems with smaller dimensions.

Let matrix $\boldsymbol{F} = \boldsymbol{R}^{-\mathsf{T}}$. It is easy to see that $\boldsymbol{F}$ is a lower-triangular matrix. Define $\check{\boldsymbol{z}} = (\check{z}_i)_n = \boldsymbol{F}^{\mathsf{T}}\bar{\boldsymbol{y}}$. Let $\boldsymbol{z}^{\mathrm{r}} = \left(z_i^{\mathrm{r}}\right)_n$ where z_i^{r} is the closest integer to $\check{z}_i$ in the interval $[\bar{l}_i, \bar{u}_i]$, i.e., $z_i^{\mathrm{r}} = \text{median}\left(\lfloor \check{z}_i \rceil, \bar{l}_i, \bar{u}_i\right)$ [1] , for $i = 1, 2, \ldots, n$. Let $\boldsymbol{z}^{\mathrm{s}} = \left(z_i^{\mathrm{s}}\right)_n$

[1] median (a, b, c) denotes the median value of a, b and c.

where $z_i^{\rm s}$ is the second closest integer to $\check{z}_i$ in $[\bar{l}_i, \bar{u}_i]$, i.e.,

$$z_i^{\rm s} = \begin{cases} z_i^{\rm r} + 1 & \text{if } z_i^{\rm r} = \bar{l}_i \text{ or } z_i^{\rm r} < \check{z}_i \\ z_i^{\rm r} - 1 & \text{if } z_i^{\rm r} = \bar{u}_i \text{ or } z_i^{\rm r} \geq \check{z}_i \end{cases} \qquad i = 1, 2, \ldots, n. \tag{3.5}$$

For an $i \in \{1, 2, \ldots, n\}$, assume we interchange columns i and n of $\boldsymbol{R}$ (thus entries i and n in the vector $\boldsymbol{z}$ and $\boldsymbol{z}^{\rm s}$ are also swapped), and return $\boldsymbol{R}$ to upper-triangular by a series of Givens rotations applied to $\boldsymbol{R}$ from the left. The same Givens rotations are also applied to $\bar{\boldsymbol{y}}$. To avoid confusion, we denote the new $\boldsymbol{R}$ by $\hat{\boldsymbol{R}}$, the new $\boldsymbol{z}$ by $\hat{\boldsymbol{z}}$, the new $\boldsymbol{z}^{\rm s}$ by $\hat{\boldsymbol{z}}^{\rm s}$ and the new $\bar{\boldsymbol{y}}$ by $\hat{\boldsymbol{y}}$. This process can be described as:

$$\hat{\boldsymbol{R}} = \boldsymbol{G}^{\mathsf{T}} \boldsymbol{R} \boldsymbol{P}_{in}, \quad \hat{\boldsymbol{y}} = \boldsymbol{G}^{\mathsf{T}} \bar{\boldsymbol{y}}, \quad \hat{\boldsymbol{z}} = \boldsymbol{P}_{in} \boldsymbol{z}, \quad \hat{\boldsymbol{z}}^{\rm s} = \boldsymbol{P}_{in} \boldsymbol{z}^{\rm s} \tag{3.6}$$

where $\boldsymbol{P}_{in}$ is the permutation matrix that interchanges columns i and n of $\boldsymbol{R}$, and $\boldsymbol{G}$ is an orthogonal matrix representing the product of the Givens rotations that bring $\boldsymbol{R}\boldsymbol{P}_{in}$ back to upper-triangular. Without loss of generality, we also assume that the diagonal entries of $\hat{\boldsymbol{R}}$ are positive. For this new problem, define

$$d_i = \hat{r}_{nn}^2 \left(\hat{z}_n^{\rm s} - \hat{y}_n / \hat{r}_{nn} \right)^2, \tag{3.7}$$

which is the n-th residual squared when $\hat{z}_n = \hat{z}_n^{\rm s}$, corresponding to column i. To determine the new last column of $\boldsymbol{R}$, the AIP reduction finds all d_1, d_2, $\ldots$, d_n and choose column $j = \operatorname{argmax}_i d_i$ to be the new last column.

Quantity d_i can be computed efficiently. From (3.6), we have $\hat{z}_n^{\rm s} = z_i^{\rm s}$ and

$$\hat{y}_n / \hat{r}_{nn} = \boldsymbol{e}_n^{\mathsf{T}} \hat{\boldsymbol{R}}^{-1} \hat{\boldsymbol{y}} = \boldsymbol{e}_n^{\mathsf{T}} \left(\boldsymbol{G}^{\mathsf{T}} \boldsymbol{R} \boldsymbol{P}_{in} \right)^{-1} \boldsymbol{G}^{\mathsf{T}} \bar{\boldsymbol{y}} = \boldsymbol{e}_n^{\mathsf{T}} \boldsymbol{P}_{in} \boldsymbol{R}^{-1} \bar{\boldsymbol{y}} = \boldsymbol{e}_i^{\mathsf{T}} \check{\boldsymbol{z}} = \check{z}_i.$$

Define $\hat{\boldsymbol{F}} = \hat{\boldsymbol{R}}^{-\mathsf{T}}$. Then we have $\hat{\boldsymbol{F}} = \boldsymbol{G}^{\mathsf{T}}\boldsymbol{R}^{-\mathsf{T}}\boldsymbol{P}_{in} = \boldsymbol{G}^{\mathsf{T}}\boldsymbol{F}\boldsymbol{P}_{in}$. Let column n of $\hat{\boldsymbol{F}}$ be $\hat{\boldsymbol{f}}_n$. Since $\hat{\boldsymbol{F}}$ is lower-triangular, $\hat{\boldsymbol{f}}_n = \hat{f}_{nn}\boldsymbol{e}_n$. This indicates that $\hat{\boldsymbol{f}}_n^{\mathsf{T}}\hat{\boldsymbol{r}}_n = \hat{f}_{nn}\hat{r}_{nn} = 1$. Recall that $\hat{r}_{nn} > 0$. We have

$$\hat{r}_{nn}^{-1} = \hat{f}_{nn} = \left\|\hat{\boldsymbol{f}}_n\right\|_2 = \left\|\hat{\boldsymbol{F}}\boldsymbol{e}_n\right\|_2 = \left\|\boldsymbol{G}^{\mathsf{T}}\boldsymbol{F}\boldsymbol{P}_{in}\boldsymbol{e}_n\right\|_2 = \left\|\boldsymbol{F}\boldsymbol{e}_i\right\|_2 = \left\|\boldsymbol{f}_i\right\|_2.$$

Thus, d_i in (3.7) can be computed using the following equation.

$$d_i = \frac{\left(z_i^{\mathrm{s}} - \check{z}_i\right)^2}{\left\|\boldsymbol{f}_i\right\|_2^2}. \tag{3.8}$$

After j is found, the AIP reduction algorithm interchanges column j and column n of $\boldsymbol{R}$ and update $\bar{\boldsymbol{y}}$ correspondingly (for convenience, we removed the hats). Then it interchanges entries j and n of $\bar{\boldsymbol{l}}$ and $\bar{\boldsymbol{u}}$ respectively, and updates $\boldsymbol{F}$ by interchanging columns j and n of $\boldsymbol{F}$ and applying to it the same Givens rotations that are used to update $\boldsymbol{R}$, so that $\boldsymbol{F} = \boldsymbol{R}^{-\mathsf{T}}$ still holds. After that, the AIP reduction algorithm fixes $z_n = z_n^{\mathrm{B}} = \text{median}\left(\lfloor \bar{y}_n/r_{nn} \rceil, \bar{l}_n, \bar{u}_n\right)$, which is just the original z_j^{r}, and updates $\tilde{\boldsymbol{y}} = \bar{\boldsymbol{y}} - \boldsymbol{r}_n z_n^{\mathrm{B}}$. Later, we will see that this z_n^{B} is the n-th entry of the box-constrained Babai point of the AIP reduced BILS problem. To determine the order of columns of $\boldsymbol{R}_{1:n-1,1:n-1}$, the AIP reduction algorithm recursively apply the above process to the following $(n-1)$-dimensional subproblem:

$$\min_{\boldsymbol{z}_{1:n-1} \in \bar{\mathcal{B}}_{n-1}} \left\|\tilde{\boldsymbol{y}}_{1:n-1} - \boldsymbol{R}_{1:n-1,1:n-1}\boldsymbol{z}_{1:n-1}\right\|_2^2, \tag{3.9}$$

where $\bar{\mathcal{B}}_{n-1} = \left\{\boldsymbol{z}_{1:n-1} \mid \bar{\boldsymbol{z}} \in \mathbb{Z}^{n-1};\ \bar{\boldsymbol{l}}_{1:n-1} \le \boldsymbol{z}_{1:n-1} \le \bar{\boldsymbol{u}}_{1:n-1}\right\}$. Because $\boldsymbol{F}_{1:n-1,1:n-1} = \left(\boldsymbol{R}_{1:n-1,1:n-1}\right)^{-\mathsf{T}}$, the inverse does not need to be recomputed for the subproblem.

Here we propose an improvement to the AIP reduction. Notice that in the AIP reduction, we need to compute $\|\boldsymbol{f}_i\|_2^2$ in (3.8) for $i = 1, 2, \ldots, n$. Then in the reduction of the $(n-1)$-dimensional subproblem (3.9), we need to compute the column vector norms $\left\|\boldsymbol{f}_{1:n-1,i}\right\|_2^2$ using the updated $\boldsymbol{F}$. Assume after we compute $\|\boldsymbol{f}_i\|_2^2$ for $i = 1, 2, \ldots, n$, we store the values in the vector $\check{\boldsymbol{f}}$, i.e., $\check{\boldsymbol{f}} = \left(\left\|\boldsymbol{f}_i\right\|_2^2\right)_n$. When we interchange columns of $\boldsymbol{F}$, we interchange entries of $\check{\boldsymbol{f}}$ accordingly to keep consistence. Note that when we apply Givens rotations to update $\boldsymbol{F}$, $\|\boldsymbol{f}_i\|_2^2$ will remain unchanged. Then in the $(n-1)$-dimensional subproblem, instead of computing $\left\|\boldsymbol{f}_{1:n-1,i}\right\|_2$ from scratch, we can simply update $\check{f}_i = \check{f}_i - f_{ni}^2$ for all i. In this way, we have $\left\|\boldsymbol{f}_{1:n-1,i}\right\|_2 = \check{f}_i$. By recursively updating $\check{\boldsymbol{f}}$ in the subproblems, the cost to compute (3.8) for the k-dimensional subproblems is then reduced from $O(k)$ to $O(1)$. Note that the above strategy is similar to what is used in the computation of the QR factorization of a matrix with standard column pivoting, see, e.g., [41, Sec. 5.4.2].

As stated, when determining the last column, the AIP reduction pursues a maximum n-th residual squared $\hat{r}_{nn}^2 \left(\hat{z}_n - \hat{y}_n/\hat{r}_{nn}\right)^2$ with $\hat{z}_n = \hat{z}_n^{\mathrm{s}}$ (see (3.7)) or equivalently $(z_i - \check{z}_i)^2/\|\boldsymbol{f}_i\|^2$ with $z_i = z_i^{\mathrm{s}}$ (see (3.8)). Following the description in [20], the motivation of doing this is to make the sphere decoding radius (i.e., $\rho^2 - (r_{nn}(z_n - \check{z}_n))^2$, see (3.11b)) for the $(n-1)$-dimensional sub-BILS-problems small, and at the same time make r_{nn} large (we have stated why we want large r_{nn} in OILS, see Section 2.3). Using z_i^{s} to compute d_i in (3.8) is because that $|z_i^{\mathrm{s}} - \check{z}_i|$ is never less than 0.5 and usually also not very large. This means that, if we choose column j to be the new column n where d_j is large, then the resulting r_{nn} is likely to be large as well

and the requirement to have large r_{nn} is met. Using z_i^{r}, instead of z_i^{s}, to compute d_i in (3.8) would not be a good choice because $|z_i^{\mathrm{r}} - \check{z}_i|$ might be very small or even 0. This means that a column i that leads to a large r_{nn} after the column permutation might correspond to a very small d_i. On the contrary a column j corresponding to a large d_j might have large $\left|z_j^{\mathrm{r}} - \check{z}_j\right|$ and a small r_{nn}.

The pseudocode of the AIP reduction is given in Algorithm 3–1. We can see that the last column of $\boldsymbol{F}$ is not used in the computation of the subproblem (3.9). In the implementation, instead of doing a column permutation, it simply drops column j of $\boldsymbol{F}$ to improve efficiency. After $\boldsymbol{F}$ is brought back to upper-triangular by the Givens rotations, the last row of $\boldsymbol{F}$ is dropped as well to bring it back to a square matrix.

3.1.3 Sphere decoding for BILS problems

Sphere decoding algorithms (see Section 2.2 for more details) can be modified to search for the optimal solution of a reduced BILS problem (3.4) (see, e.g., [30, 18, 24]).

A sphere decoder for BILS searches for the optimal solution within the following region:

$$\|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2 < \rho^2, \tag{3.10a}$$

$$\bar{\boldsymbol{l}} \leq \boldsymbol{z} \leq \bar{\boldsymbol{u}}, \tag{3.10b}$$

where ρ denotes the search radius. In geometry, (3.10) represents the intersection of a hyper-ellipsoid and a box, as exemplified in Figure 3–1 for the case $n = 2$. We can

Algorithm 3–1 The AIP reduction

function $[\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \boldsymbol{P}, \boldsymbol{z}^{\mathrm{B}}] = \mathrm{AIP}(\boldsymbol{A}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$

apply the QR factorization to obtain $\boldsymbol{A} = [\boldsymbol{Q}_1, \boldsymbol{Q}_2] \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{bmatrix}$ and $\bar{\boldsymbol{y}} = \boldsymbol{Q}_1^{\mathsf{T}} \boldsymbol{y}$;
initialize $\bar{\boldsymbol{l}} = \boldsymbol{l}$ and $\bar{\boldsymbol{u}} = \boldsymbol{u}$, let $\tilde{\boldsymbol{y}} = \bar{\boldsymbol{y}}$;
compute $\boldsymbol{F} = \boldsymbol{R}^{-\mathsf{T}}$ and set $\boldsymbol{P} = \boldsymbol{I}_n$;
compute $\check{\boldsymbol{f}} = (\|\boldsymbol{f}_i\|_2^2)_n$;
for $k = n$ to 2 **do**
    $\check{\boldsymbol{z}} = \boldsymbol{F}^{\mathsf{T}} \tilde{\boldsymbol{y}} \in \mathbb{R}^k$;
    let z_i^{s} be the second closest integer in $[\bar{l}_i, \bar{u}_i]$ to $\check{z}_i$, for $i = 1, \ldots, k$; ▷ See (3.5)
    $d_i = \left(z_i^{\mathrm{s}} - \check{z}_i\right)^2 / \check{f}_i$ for $i = 1, 2, \ldots, k$;
    let $j = \mathrm{argmax}_{i=1,2,\ldots,k}\, d_i$;
    $z_k^{\mathrm{B}} = \mathrm{median}\left(\lfloor \check{z}_i \rceil, \bar{l}_i, \bar{u}_i\right)$.
    $\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{y}} - \boldsymbol{r}_j z_k^{\mathrm{B}}$;
    remove column j of $\boldsymbol{F}$ and entry j of $\check{\boldsymbol{f}}$;
    if $i \neq k$ **then**
        interchange column j and k of $\boldsymbol{R}$;
        interchange entry j and k of $\bar{\boldsymbol{l}}$ and $\bar{\boldsymbol{u}}$;
        interchange columns j and k of $\boldsymbol{P}$;
        use Givens rotations to bring $\boldsymbol{R}$ back to upper triangular;
        apply the same Givens rotations to update $\boldsymbol{F}$, $\bar{\boldsymbol{y}}$ and $\tilde{\boldsymbol{y}}$;
    end if
    update $\check{f}_i = \check{f}_i - f_{ki}^2$ for $i = 1, 2, \ldots, k$;
    remove row k in $\boldsymbol{F}$ and remove entry k in $\tilde{\boldsymbol{y}}$; ▷ s.t. $\boldsymbol{F} \in \mathbb{R}^{k \times k}$ $\bar{\boldsymbol{y}} \in \mathbb{R}^k$
end for

partition (3.10a) like (2.11) and reorganize (3.10) to

$$(\bar{y}_n - r_{nn} z_n)^2 < \rho^2, \qquad \bar{l}_n \leq z_n \leq \bar{u}_n, \tag{3.11a}$$

$$\left\| (\hat{\boldsymbol{y}} - \hat{\boldsymbol{r}} z_n) - \hat{\boldsymbol{R}} \hat{\boldsymbol{z}} \right\|_2^2 < \rho^2 - (\bar{y}_n - r_{nn} z_n)^2, \qquad \hat{\boldsymbol{l}} \leq \hat{\boldsymbol{z}} \leq \hat{\boldsymbol{u}}, \tag{3.11b}$$

where $\hat{\boldsymbol{l}} = \bar{\boldsymbol{l}}_{1:n-1}$ and $\hat{\boldsymbol{u}} = \bar{\boldsymbol{u}}_{1:n-1}$. From (3.11a), we have:

$$\max(\bar{l}_n, \lfloor (\bar{y}_n - \rho)/r_{nn} \rfloor + 1) \leq z_n \leq \min(\bar{u}_n, \lceil (\bar{y}_n + \rho)/r_{nn} \rceil - 1). \tag{3.12}$$

For each possible integer value of z_n in (3.12), the n-dimensional search space (3.10) reduces to an $(n-1)$-dimensional search space (3.11b). The overall process of sphere decoding for a BILS problem is similar to the sphere decoding for the OILS problems described in Section 2.2.

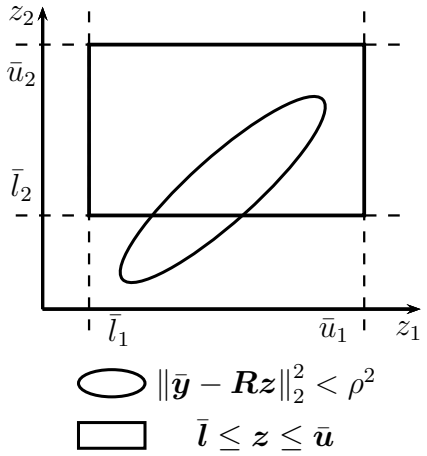


Figure 3–1: BILS search region

As in section 2.2, let $c_k = \left(\bar{y}_k - \sum_{j=k+1}^{n} r_{kj} z_j\right)/r_{kk}$. The Schnorr-Euchner sphere decoder for BILS enumerates the integer values of z_k satisfying (3.12) in ascending order with respect to their distances to c_k. At level k of the recursive decoding process, the first integer to be enumerated for z_k is median $\left(\lfloor c_k \rceil, \bar{l}_k, \bar{u}_k\right)$, which is the nearest integer in $[\bar{l}_k, \bar{u}_k]$ to c_k. Similar to the Babai point of an OILS problem (see (2.16)), for a reduced BILS problem we can define the box-constrained Babai point $\boldsymbol{z}^{\mathrm{B}} = (z_i^{\mathrm{B}})_n$ where

$$z_k^{\mathrm{B}} = \text{median}\left(\lfloor c_k \rceil, \bar{l}_k, \bar{u}_k\right), \quad k = n, n-1, \ldots, 1. \tag{3.13}$$

In (3.13), c_k is computed using (2.14) with $z_j = z_j^{\mathrm{B}}$ for $j = k+1, k+2, \ldots, n$. We can see that the AIP algorithm computes the box-constrained Babai point in the reduction process, see Algorithm 3–1.

3.2 Inactive Constraints and Partially Open Box Constrained ILS Problems

In this section, we first introduce the concepts of inactive constraints for a BILS problem, and how to reduce a BILS problem to a partially open box constrained ILS problem (POBILS). Unlike the reduction of regular BILS problems where only column permutations are allowed, we show that for a POBILS problem, size reductions can be incorporated in the reduction. In the end of this section, we give the general rule on the reduction of POBILS problems.

3.2.1 Inactive constraints and POBILS problems

Definition 3–2 (Inactive Constraints)**.** Given a BILS problem, if the interval constraint on a variable can be removed without altering the optimal solution, then we say this interval constraint is an inactive constraint. If a set of interval constraints can be removed from a BILS problem without altering the optimal solution, then we say this set is an inactive set of constraints.

An example of an inactive constraint can be found in Figure 3–1. Removing the interval constraint on z_1 does not change the search region of the sphere decoder, the intersection of the box and the ellipse. Thus it does not alter the optimal solution either. From Definition 3–2, the interval constraint $\bar{l}_1 \leq z_1 \leq \bar{u}_1$ is inactive, and itself forms an inactive set of the BILS problem.

One problem we are interested in is to find a large inactive set of constraints for a given BILS problem (see Section 3.2.3). However, finding the largest inactive set of a given BILS is generally difficult. To better understand the structure of inactive sets of a BILS problem, we give the following properties:

i Any subset of an inactive set of a BILS problem is also an inactive set of this BILS problem;

ii The empty set Φ is an inactive set for any BILS problem;

However, the inactive sets of a BILS problem do not possess the augmentation property, i.e., for two different inactive sets $\mathcal{A}$ and $\mathcal{B}$, there may not be such an element in $\mathcal{A}$ such that adding it to $\mathcal{B}$ gives an inactive set larger than $\mathcal{B}$. This can be seen using the example shown in Figure 3–2, where $\begin{bmatrix}4 & 3\end{bmatrix}^{\mathsf{T}}$ is the optimal solution of the given BILS problem. Removing either the interval constraint on x_1, or the interval constraint on x_2 will not introduce new integer point into the search region, and thus will not alter the optimal solution. So, each constraint itself forms an inactive set of this problem. However, adding the two inactive sets does not give a valid inactive set because when both of the interval constraints are removed, the optimal solution is then changed to $\begin{bmatrix}3 & 2\end{bmatrix}^{\mathsf{T}}$. Because of the lack of the augmentation property, the inactive sets of a BILS problem do not form a matroid. This implies that generally the problem of finding the largest inactive set of a given BILS problem cannot be solved by greedy algorithms.

For an inactive set of constraints, we will use $\mathcal{I}$ to denote the set of the corresponding indices. For convenience, $\mathcal{I}$ is also referred to as an inactive set. Define $\mathcal{J} \equiv \mathcal{I}^{\mathsf{c}}$. Removing the interval constraints in the inactive set from the box $\mathcal{B}$ in

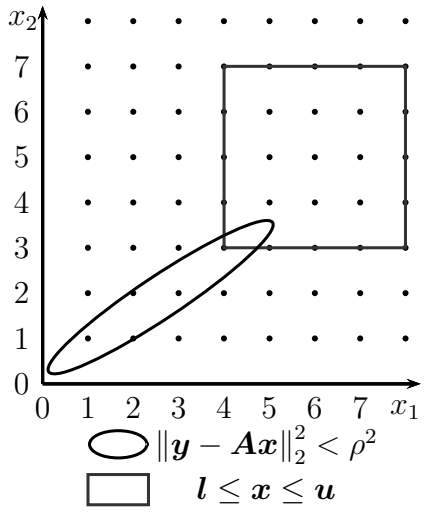


Figure 3–2: Example of inactive sets

(3.1) gives us a constraint-reduced BILS problem:

$$\min_{\boldsymbol{x}\in\mathcal{P}} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2, \tag{3.14}$$

where

$$\mathcal{P} = \{\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{Z}^n;\ l_i \leq x_i \leq u_i, \forall i \in \mathcal{J}\} \tag{3.15}$$

is a partially open box (we call it po-box for short). The constraint-reduced BILS problem (3.14) is also referred to as the po-box constrained ILS (POBILS) problem. Recall that for $\mathcal{B}$ in (3.1), we always have the strict inequality $l_i < u_i$. If this condition holds for a po-box $\mathcal{P}$, then we call this $\mathcal{P}$ a non-degenerate po-box. In a POBILS problem (3.14), $\mathcal{P}$ is always non-degenerative. For a po-box $\mathcal{P}$, we always assume either $l_i \neq -\infty$ or $u_i \neq \infty$ for all $i \in \mathcal{J}$. If this is not the case for some i, we can simply remove i from $\mathcal{J}$ without changing the box. We can regard a regular

BILS problem as a special case of of POBILS problem, i.e., $\mathcal{J} = \{1, 2, \ldots, n\}$, and regard a OILS problem as another special case of POBILS problem, i.e., $\mathcal{J} = \Phi$.

For a po-box $\mathcal{P}$, define a subset of lattice $\Lambda(\boldsymbol{A})$:

$$\Lambda_{\mathcal{P}}(\boldsymbol{A}) = \{\boldsymbol{A}\boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}\}. \tag{3.16}$$

Note that $\Lambda_{\mathcal{P}}(\boldsymbol{A})$ is a constrained lattice. If $\mathcal{J} = \phi$, then $\mathcal{P} = \mathbb{Z}^n$ and $\Lambda_{\mathcal{P}}(\boldsymbol{A})$ becomes the regular lattice $\Lambda(\boldsymbol{A})$. It is easy to see that solving (3.14) is actually to find the closest lattice point in $\Lambda_{\mathcal{P}}(\boldsymbol{A})$ to the given vector $\boldsymbol{y}$.

3.2.2 Using IGTs in the reduction of POBILS problems

Recall that in the reduction process for the BILS problem (3.1), usually the unimodular matrix $\boldsymbol{Z}$ in the QRZ factorization (2.2) is chosen to be a permutation matrix $\boldsymbol{P} \in \mathbb{R}^{n\times n}$, see (3.2). Other types of unimodular matrices are not used because they skew the constraint box, making sphere decoding difficult. In this subsection, we show that for the POBILS problem (3.14), however, when certain conditions are satisfied, IGTs can be used in the QRZ factorization of $\boldsymbol{A}$ without skewing its constraints. This indicates that size reductions can be incorporated in the reduction of POBILS problems.

Recall that an IGT has the following form:

$$\boldsymbol{Z}_{ij} = \boldsymbol{I} - \zeta_{ij}\boldsymbol{e}_i\boldsymbol{e}_j^{\mathsf{T}}, \qquad \zeta_{ij} \in \mathbb{Z}, \quad i \neq j,$$

and its inverse is

$$\boldsymbol{Z}_{ij}^{-1} = \boldsymbol{I} + \zeta_{ij}\boldsymbol{e}_i\boldsymbol{e}_j^{\mathsf{T}}. \tag{3.17}$$

Let $\bar{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{Z}_{ij}$ and $\bar{\boldsymbol{x}} = \boldsymbol{Z}_{ij}^{-1}\boldsymbol{x}$, so $\boldsymbol{A}\boldsymbol{x} = \bar{\boldsymbol{A}}\bar{\boldsymbol{x}}$. From (3.17), we have

$$\bar{\boldsymbol{x}} = \boldsymbol{Z}_{ij}^{-1}\boldsymbol{x} = (\boldsymbol{I} + \zeta_{ij}\boldsymbol{e}_i\boldsymbol{e}_j^{\mathsf{T}})\boldsymbol{x} = \boldsymbol{x} + \zeta_{ij}x_j\boldsymbol{e}_i,$$

leading to

$$\bar{x}_i = x_i + \zeta_{ij}x_j, \qquad \bar{x}_k = x_k, \ \forall k \neq i. \tag{3.18}$$

This indicates that only the i-th element of $\boldsymbol{x}$ is changed by the IGT, and other elements are not. Now suppose the i-th interval constraint $l_i \leq x_i \leq u_i$ is inactive (i.e. $i \in \mathcal{I}$), then x_i is not interval-bounded in $\mathcal{P}$ in (3.14). From (3.18), it is easy to see that for any $\boldsymbol{x} \in \mathcal{P}$, we still have $\bar{\boldsymbol{x}} \in \mathcal{P}$, and vice versa. Without complicating the interval constraints, IGT $\boldsymbol{Z}_{ij}$ can be applied to transform (3.14) to

$$\min_{\bar{\boldsymbol{x}} \in \mathcal{P}} \left\| \boldsymbol{y} - \bar{\boldsymbol{A}}\bar{\boldsymbol{x}} \right\|_2^2.$$

From the above we can see that if a BILS problem has a non-empty inactive set $\mathcal{I}$, then we can remove the i-th interval constraint of this problem for all $i \in \mathcal{I}$ to form a POBILS problem. For the POBILS problem, IGT $\boldsymbol{Z}_{ij}$ can then be applied when $i \in \mathcal{I}$.

3.2.3 Reduction of po-box constrained lattices

In this section, we consider the reduction of constrained lattices (3.16). Recall that in the regular lattice reduction, we use a unimodular matrix $\boldsymbol{Z}$ to reduce the lattice basis $\boldsymbol{A}$ to $\boldsymbol{A}\boldsymbol{Z}$ without changing the lattice, i.e., $\Lambda(\boldsymbol{A}) = \Lambda(\boldsymbol{A}\boldsymbol{Z})$ (see Section 1.4). In the constrained lattice reduction, we want to find a matrix $\boldsymbol{Z} \in \mathbb{R}^{n\times n}$ such that

$$\Lambda_{\mathcal{P}}(\boldsymbol{A}) = \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{A}\boldsymbol{Z}), \tag{3.19}$$

where $\bar{\mathcal{P}}$ is a po-box and the lattice basis $\boldsymbol{AZ}$ is reduced. Here $\bar{\mathcal{P}}$ is required to be a po-box so that sphere decoders can enumerate integer points in $\bar{\mathcal{P}}$ efficiently. Next, we show what conditions $\boldsymbol{Z}$ should satisfy so that there exists a po-box $\bar{\mathcal{P}}$ making (3.19) hold.

Lemma 3–1. *Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with full column rank and a matrix $\boldsymbol{Z} \in \mathbb{R}^{n\times n}$, for any po-boxes $\mathcal{P}$ and $\bar{\mathcal{P}}$, $\Lambda_{\mathcal{P}}(\boldsymbol{A}) = \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{AZ})$ if and only if $\mathcal{P} = \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{Z})$.*

Proof. It is easy to see that $\Lambda_{\bar{\mathcal{P}}}(\boldsymbol{AZ}) = \left\{\boldsymbol{AZx} \mid \boldsymbol{x} \in \bar{\mathcal{P}}\right\} = \{\boldsymbol{A}\bar{\boldsymbol{x}} \mid \bar{\boldsymbol{x}} \in \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{Z})\}$. When $\boldsymbol{A}$ has full column rank, $f(\boldsymbol{x}) = \boldsymbol{Ax}$ is a bijection. Thus $\{\boldsymbol{Ax} \mid \boldsymbol{x} \in \mathcal{P}\} = \{\boldsymbol{A}\bar{\boldsymbol{x}} \mid \bar{\boldsymbol{x}} \in \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{Z})\}$ if and only if $\mathcal{P} = \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{Z})$. □

Lemma 3–2. *Given a non-degenerate po-box $\mathcal{P}$ as in* (3.15)*, if there exists another po-box $\bar{\mathcal{P}}$ and a matrix $\boldsymbol{Z} \in \mathbb{R}^{n\times n}$ such that $\mathcal{P} = \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{Z})$, then $\boldsymbol{Z}$ is unimodular.*

Proof. To prove $\boldsymbol{Z}$ is unimodular, we first prove that $\boldsymbol{Z}$ is invertible and $\boldsymbol{Z}^{-1}$ is an integer matrix. Construct a vector $\boldsymbol{x} = (x_i)_n$ where

$$x_i = \begin{cases} l_i & \text{when } i \in \mathcal{J} \\ 0 & \text{otherwise} \end{cases}. \tag{3.20}$$

Recall that in a non-degenerate po-box $\mathcal{P}$, $l_i < u_i$ for $i \in \mathcal{J}$. From (3.15), it is easy to see that

$$\boldsymbol{x} \in \mathcal{P}; \qquad (\boldsymbol{x} + \boldsymbol{e}_i) \in \mathcal{P}, \quad i = 1, 2, \ldots, n. \tag{3.21}$$

Thus there must exist vectors $\bar{\boldsymbol{x}} \in \bar{\mathcal{P}}$ and $\bar{\boldsymbol{x}}'_i \in \bar{\mathcal{P}}$ for $i = 1, 2, \ldots, n$ such that

$$\boldsymbol{Z}\bar{\boldsymbol{x}} = \boldsymbol{x}, \tag{3.22a}$$

$$\boldsymbol{Z}\bar{\boldsymbol{x}}'_i = \boldsymbol{x} + \boldsymbol{e}_i. \tag{3.22b}$$

Define $\boldsymbol{h}_i = \bar{\boldsymbol{x}}'_i - \bar{\boldsymbol{x}}$. Subtracting (3.22a) from (3.22b) gives us $\boldsymbol{Z}\boldsymbol{h}_i = \boldsymbol{e}_i$. Define $\boldsymbol{H} = \begin{bmatrix}\boldsymbol{h}_1 & \boldsymbol{h}_2 & \dots & \boldsymbol{h}_n\end{bmatrix} \in \mathbb{Z}^{n\times n}$. We have $\boldsymbol{Z}\boldsymbol{H} = \boldsymbol{I}$, and thus $\boldsymbol{Z}^{-1} = \boldsymbol{H}$ and it is an integer matrix.

Write $\bar{\mathcal{P}}$ as

$$\bar{\mathcal{P}} = \left\{\bar{\boldsymbol{x}} \mid \bar{\boldsymbol{x}} \in \mathbb{Z}^n;\ \bar{l}_j \le \bar{x}_j \le \bar{u}_j\ \forall j \in \bar{\mathcal{J}}\right\}.$$

Next we prove that $\bar{\mathcal{P}}$ is also non-degenerate, i.e., $\bar{l}_j < \bar{u}_j$ for all $j \in \bar{\mathcal{J}}$ in $\bar{\mathcal{P}}$. Assume that there exists $j \in \bar{\mathcal{J}}$ such that $\bar{l}_j = \bar{u}_j$. Then, it can be seen that $\boldsymbol{e}_j^\mathsf{T}\bar{\boldsymbol{x}} = \bar{l}_j$ for any $\bar{\boldsymbol{x}} \in \bar{\mathcal{P}}$. For the vector $\boldsymbol{x}$ defined in (3.20), from (3.21) and (3.22) we have $\boldsymbol{H}\boldsymbol{x} \in \bar{\mathcal{P}}$ and $\boldsymbol{H}(\boldsymbol{x}+\boldsymbol{e}_i) \in \bar{\mathcal{P}}$ for $i = 1, 2, \dots, n$. Let $\boldsymbol{h}_j^\mathsf{T}$ be the j-th row of $\boldsymbol{H}$, i.e., $\boldsymbol{h}_j^\mathsf{T} = \boldsymbol{e}_j^\mathsf{T}\boldsymbol{H}$. We then have

$$\boldsymbol{h}_j^\mathsf{T}\boldsymbol{x} = \bar{l}_j, \tag{3.23a}$$

$$\boldsymbol{h}_j^\mathsf{T}(\boldsymbol{x} + \boldsymbol{e}_i) = \bar{l}_j, \quad i = 1, 2, \dots, n. \tag{3.23b}$$

Subtracting (3.23a) from (3.23b) gives $\boldsymbol{h}_j^\mathsf{T}\boldsymbol{e}_i = 0$ for all $i = 1, 2, \dots, n$, indicating $\boldsymbol{h}_j^\mathsf{T} = \boldsymbol{0}$, which contradicts to the fact that $\boldsymbol{H}$ is invertible. Thus $\bar{l}_j < \bar{u}_j$. In the rest part of this chapter, when we refer to a po-box, we always assume that it is non-degenerate.

Then we prove $\boldsymbol{Z}$ is an integer matrix. Let us define $\bar{\boldsymbol{x}} = (\bar{x}_i)_n \in \bar{\mathcal{P}}$ where $\bar{x}_i = \bar{l}_i$ when $i \in \bar{\mathcal{J}}$ and $\bar{x}_i = 0$ otherwise. It is easy to see that $(\bar{\boldsymbol{x}} + \boldsymbol{e}_i) \in \bar{\mathcal{P}}$ for $i = 1, 2, \dots, n$. We must have

$$\boldsymbol{Z}\bar{\boldsymbol{x}} \in \mathcal{P} \subseteq \mathbb{Z}^n, \tag{3.24a}$$

$$\boldsymbol{Z}(\bar{\boldsymbol{x}} + \boldsymbol{e}_i) \in \mathcal{P} \subseteq \mathbb{Z}^n. \tag{3.24b}$$

Subtracting (3.24a) from (3.24b) gives us $\boldsymbol{Z}\boldsymbol{e}_i \in \mathbb{Z}^n$ for all $i = 1, 2, \ldots, n$. Thus, we have $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$. Since $\boldsymbol{Z}$ and $\boldsymbol{Z}^{-1}$ are both integral, by Proposition 1–1, $\boldsymbol{Z}$ must be unimodular. □

It is obvious that when a box $\mathcal{B}$ is transformed by a permutation matrix, we still get a box. This observation has been pervasively used in solving BILS problems, see, e.g., [108, 30, 98, 24, 20]. Next, we give the other direction of this observation.

Observation 3–1. Given a box $\mathcal{B}$, if there exists a box $\bar{\mathcal{B}}$ and a square matrix $\boldsymbol{Z} \in \mathbb{R}^{n\times n}$ such that

$$\mathcal{B} = \Lambda_{\bar{\mathcal{B}}}(\boldsymbol{Z}), \tag{3.25}$$

then $|\boldsymbol{Z}|$ is a permutation matrix.

Proof. Because a box is a special po-box, from Lemma 3–1, we know $\boldsymbol{Z}$ is unimodular. Next, we prove by contradiction that each column of $\boldsymbol{Z}$ has at most one non-zero entry. Let $\boldsymbol{z}_i^{\mathsf{T}}$ denote the i-th row of $\boldsymbol{Z}$. Assume that there exist $i, j, k \in \{1, 2, \ldots, n\}$ such that $i \neq j$, $z_{ik}, z_{jk} \neq 0$. Without loss of generality, we assume that both $z_{ik}, z_{jk} > 0$. If, for example, $z_{ik} < 0$, we can multiply $\boldsymbol{z}_i^{\mathsf{T}}$ by -1 and interchange l_i, u_i in $\mathcal{B}$ so that (3.25) still holds. Due to (3.25), for any $\boldsymbol{x} \in \mathcal{B}$, we can find $\bar{\boldsymbol{x}} \in \bar{\mathcal{B}}$ such that $\boldsymbol{x} = \boldsymbol{Z}\bar{\boldsymbol{x}}$. It follows that $x_i = \boldsymbol{z}_i^{\mathsf{T}}\bar{\boldsymbol{x}} = \sum_{t=1}^{n} z_{it}\bar{x}_t$. Since $l_i \leq x_i \leq u_i$, we must have the following equalities:

$$l_i = \sum_{t=1}^{n} \min_{\bar{l}_t \leq \bar{x}_t \leq \bar{u}_t} (z_{it}\bar{x}_t), \qquad u_i = \sum_{t=1}^{n} \max_{\bar{l}_t \leq \bar{x}_t \leq \bar{u}_t} (z_{it}\bar{x}_t).$$

Then, under the assumption $\boldsymbol{z}_{ik} > 0$, when $x_i = l_i$, we must have $\bar{x}_k = \bar{l}_k$, i.e., $z_{ik}\bar{x}_k$ reaches its minimum. Similarly, under the assumption $\boldsymbol{z}_{jk} > 0$, $x_j = u_j$ requires

$\bar{x}_k = \bar{u}_k$, i.e., $z_{jk}\bar{x}_k$ reaches its maximum. Since $\bar{l}_k \neq \bar{u}_k$, this implies that we cannot have $x_i = l_i$ and $x_j = u_j$ at the same time, contradicting to the assumption that $\boldsymbol{x}$ is an arbitrary vector in $\mathcal{B}$. Thus $\boldsymbol{Z}$ has at most one non-zero entry each column. Since $\boldsymbol{Z}$ is non-singular, we can further conclude that $\boldsymbol{Z}$ has exactly one non-zero entry in each column and each row. Since $\det(\boldsymbol{Z}) = \pm 1$ (from the definition of unimodular matrix), it is easy to see that those non-zero entries of $\boldsymbol{Z}$ has to be ± 1, completing the proof. □

Theorem 3. *Given a po-box $\mathcal{P}$ with the form of* (3.15) *and a matrix $\boldsymbol{Z} \in \mathbb{R}^{n\times n}$. We can construct another po-box $\bar{\mathcal{P}}$ such that*

$$\mathcal{P} = \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{Z}), \tag{3.26}$$

if and only if $\boldsymbol{Z}$ is unimodular and for any $i \in \mathcal{J}$, there exists $j \in \{1, 2, \ldots, n\}$ such that

$$\left|\boldsymbol{z}_i^{\mathsf{T}}\right| = \boldsymbol{e}_j^{\mathsf{T}}, \tag{3.27}$$

where $\boldsymbol{z}_i^{\mathsf{T}}$ is the i-th row of $\boldsymbol{Z}$.

Proof. To prove the "if" part, we first give a process to construct $\bar{\mathcal{P}}$ and then prove (3.26) holds with the constructed $\bar{\mathcal{P}}$.

Define map $p(i) = j$ for each of those pairs of i and j in (3.27). Let

$$\bar{\mathcal{P}} = \{\bar{\boldsymbol{x}} \mid \bar{\boldsymbol{x}} \in \mathbb{Z}^n;\ l_i \leq z_{ij}\bar{x}_j \leq u_i \quad \forall i \in \mathcal{J}, j = p(i)\}, \tag{3.28}$$

where $z_{ij} = \pm 1$, see (3.27). With the $\bar{\mathcal{P}}$ constructed above, in the following we prove (3.26) by showing $\mathcal{P} \supseteq \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{Z})$ and $\mathcal{P} \subseteq \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{Z})$.

To show $\mathcal{P} \supseteq \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{Z})$, it suffices to show that for every $\bar{\boldsymbol{x}} \in \bar{\mathcal{P}}$, $\boldsymbol{x} = \boldsymbol{Z}\bar{\boldsymbol{x}} \in \mathcal{P}$. Since $\bar{\boldsymbol{x}} \in \mathbb{Z}^n$ and $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$, we have $\boldsymbol{x} \in \mathbb{Z}^n$. For any $i \in \mathcal{J}$, $x_i = \boldsymbol{z}_i^{\mathsf{T}}\bar{\boldsymbol{x}} = z_{ij}\boldsymbol{e}_j^{\mathsf{T}}\bar{\boldsymbol{x}} = z_{ij}\bar{x}_j$. From (3.28), we know that $l_i \le z_{ij}\bar{x}_j \le u_i$, proving $\boldsymbol{x} \in \mathcal{P}$.

To show $\mathcal{P} \subseteq \Lambda_{\bar{\mathcal{P}}}(\boldsymbol{Z})$, it suffices to show that for every $\boldsymbol{x} \in \mathcal{P}$, $\bar{\boldsymbol{x}} = \boldsymbol{Z}^{-1}\boldsymbol{x} \in \bar{\mathcal{P}}$. Since $\boldsymbol{x} \in \mathbb{Z}^n$ and $\boldsymbol{Z}$ is unimodular, we have $\bar{\boldsymbol{x}} \in \mathbb{Z}^n$ immediately. Let $\boldsymbol{H} = \boldsymbol{Z}^{-1}$ and let row i of $\boldsymbol{H}$ be $\boldsymbol{h}_i^{\mathsf{T}}$. From $\boldsymbol{ZH} = \boldsymbol{I}_n$, we can see that $\boldsymbol{z}_i^{\mathsf{T}}\boldsymbol{H} = \boldsymbol{e}_i^{\mathsf{T}}$. For any $i \in \mathcal{J}$ and $j = p(i)$, we also have $\boldsymbol{z}_i^{\mathsf{T}}\boldsymbol{H} = z_{ij}\boldsymbol{e}_j^{\mathsf{T}}\boldsymbol{H} = z_{ij}\boldsymbol{h}_j^{\mathsf{T}}$, indicating $\boldsymbol{h}_j^{\mathsf{T}} = z_{ij}\boldsymbol{e}_i^{\mathsf{T}}$. We have $z_{ij}\bar{x}_j = z_{ij}\boldsymbol{e}_j^{\mathsf{T}}\bar{\boldsymbol{x}} = z_{ij}\boldsymbol{e}_j^{\mathsf{T}}\boldsymbol{H}\boldsymbol{x} = z_{ij}\boldsymbol{h}_j^{\mathsf{T}}\boldsymbol{x} = z_{ij}^2\boldsymbol{e}_i^{\mathsf{T}}\boldsymbol{x} = x_i$. From (3.15), $l_i \le z_{ij}\bar{x}_j \le u_i$, proving $\bar{\boldsymbol{x}} \in \bar{\mathcal{P}}$.

Now we prove the "only if" part of the theorem. By Lemma 3–2, we can conclude that $\boldsymbol{Z}$ is unimodular. Next, we prove that (3.27) must hold. Let $\boldsymbol{P}$ be a permutation matrix such that for any $i \in \mathcal{J}$, we have $\boldsymbol{P}\boldsymbol{e}_i = \boldsymbol{e}_k$ for some $k \in \{1, 2, \ldots, |\mathcal{J}|\}$. Similarly, let $\bar{\boldsymbol{P}}$ be a permutation matrix such that for any $i \in \bar{\mathcal{J}}$, we have $\bar{\boldsymbol{P}}\boldsymbol{e}_i = \boldsymbol{e}_k$ for some $k \in \left\{1, 2, \ldots, \left|\bar{\mathcal{J}}\right|\right\}$. Note that $\Lambda_{\mathcal{P}}(\boldsymbol{P}) = \{\boldsymbol{P}\boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}\}$ and $\Lambda_{\bar{\mathcal{P}}}(\bar{\boldsymbol{P}}) = \left\{\bar{\boldsymbol{P}}\bar{\boldsymbol{x}} \mid \bar{\boldsymbol{x}} \in \bar{\mathcal{P}}\right\}$. Let matrix $\bar{\boldsymbol{Z}} = \boldsymbol{P}\boldsymbol{Z}\bar{\boldsymbol{P}}^{\mathsf{T}}$. From (3.26), it is easy to see that for any $\bar{\boldsymbol{s}} \in \Lambda_{\bar{\mathcal{P}}}(\bar{\boldsymbol{P}})$, there exists an $\boldsymbol{s} \in \Lambda_{\mathcal{P}}(\boldsymbol{P})$ such that

$$\boldsymbol{s} = \bar{\boldsymbol{Z}}\bar{\boldsymbol{s}}.$$

We note that s_k is interval-constrained if and only if $k \in \{1, 2, \ldots, |\mathcal{J}|\}$ and $\bar{s}_k$ is interval-constrained if and only if $k \in \left\{1, 2 \ldots, \left|\bar{\mathcal{J}}\right|\right\}$. Partition $\bar{\boldsymbol{Z}}$ as follows:

$$\bar{\boldsymbol{Z}} = \begin{matrix} \\ |\mathcal{J}| \\ n - |\mathcal{J}| \end{matrix} \begin{matrix} \begin{matrix} |\bar{\mathcal{J}}| & n - |\bar{\mathcal{J}}| \end{matrix} \\ \begin{bmatrix} \bar{\boldsymbol{Z}}_{11} & \bar{\boldsymbol{Z}}_{12} \\ \bar{\boldsymbol{Z}}_{21} & \bar{\boldsymbol{Z}}_{22} \end{bmatrix} \end{matrix}.$$

We can claim $\bar{\boldsymbol{Z}}_{12} = \mathbf{0}_{|\mathcal{J}|\times(n-|\bar{\mathcal{J}}|)}$. To prove this, assume $\bar{z}_{ik} \neq 0$ for some $i \in \{1, 2, \ldots, |\mathcal{J}|\}$ and $k \in \left\{\left|\bar{\mathcal{J}}\right| + 1, \left|\bar{\mathcal{J}}\right| + 2, \ldots, n\right\}$. For any integer α and a given vector $\bar{\boldsymbol{s}}' \in \Lambda_{\bar{\mathcal{P}}}(\bar{\boldsymbol{P}})$, we define $\bar{\boldsymbol{s}} = (\bar{\boldsymbol{s}}' + \alpha \boldsymbol{e}_k) \in \Lambda_{\bar{\mathcal{P}}}(\bar{\boldsymbol{P}})$ (because $\bar{s}_k$ is unbounded). Thus vector $\boldsymbol{s} = \bar{\boldsymbol{Z}}\bar{\boldsymbol{s}} \in \Lambda_{\mathcal{P}}(\boldsymbol{P})$. Let $\boldsymbol{s}' = \bar{\boldsymbol{Z}}\bar{\boldsymbol{s}}'$. We have $s_i = s'_i + \alpha z_{ik}$. Since $\bar{z}_{ik} \neq 0$ and α is an arbitrary integer, this equation indicates that s_i is unbounded, contradicting the fact that s_i is interval-constrained.

Since $\bar{\boldsymbol{Z}}_{12} = \mathbf{0}_{|\mathcal{J}|\times(n-|\bar{\mathcal{J}}|)}$ and $\bar{\boldsymbol{Z}}$ is non singular, it is easy to see that $|\mathcal{J}| \leq \left|\bar{\mathcal{J}}\right|$. Similary, from $\bar{\mathcal{P}} = \Lambda_{\mathcal{P}}(\boldsymbol{H})$ with $\boldsymbol{H} = \boldsymbol{Z}^{-1}$, we can show $\left|\bar{\mathcal{J}}\right| \leq |\mathcal{J}|$, giving us that $|\mathcal{J}| = \left|\bar{\mathcal{J}}\right|$. Thus, $\bar{\boldsymbol{Z}}_{11}$ is a square matrix. From Observation 3–1, $\left|\bar{\boldsymbol{Z}}_{11}\right|$ is a permutation matrix. Let $\bar{\boldsymbol{z}}_k^{\mathsf{T}}$ be the k-th row of $\bar{\boldsymbol{Z}}$. Then $\left|\bar{\boldsymbol{z}}_k^{\mathsf{T}}\right|$ is a standard unit vector for any $k \in \{1, 2, \ldots, |\mathcal{J}|\}$. From the definition of $\boldsymbol{P}$, for any $i \in \mathcal{J}$, we can find some $k \in \{1, 2, \ldots, |\mathcal{J}|\}$ such that $\boldsymbol{P}\boldsymbol{e}_i = \boldsymbol{e}_k$. Thus

$$\left|\boldsymbol{z}_i^{\mathsf{T}}\right| = \boldsymbol{e}_i^{\mathsf{T}}\boldsymbol{Z} = \boldsymbol{e}_i^{\mathsf{T}}\boldsymbol{P}^{\mathsf{T}}\bar{\boldsymbol{Z}}\bar{\boldsymbol{P}} = \boldsymbol{e}_k^{\mathsf{T}}\bar{\boldsymbol{Z}}\bar{\boldsymbol{P}} = \bar{\boldsymbol{z}}_k^{\mathsf{T}}\bar{\boldsymbol{P}}.$$

Obviously, $\left|\boldsymbol{z}_i^{\mathsf{T}}\right|$ is also a standard unit vector, completing the proof. □

3.2.4 Reduction of the POBILS problem

We have shown that in the reduction of the POBILS problem (3.14) with $|\mathcal{J}| < n$, the unimodular matrix $\boldsymbol{Z}$ in the QRZ factorization can incorporate IGTs. In the reduction of the POBILS problem (3.14), by Theorem 3, we compute the following QRZ factorization of $\boldsymbol{A}$

$$\boldsymbol{Q}^{\mathsf{T}}\boldsymbol{A}\boldsymbol{Z} = \begin{bmatrix}\boldsymbol{R}\\ \mathbf{0}\end{bmatrix}, \tag{3.29}$$

where $\boldsymbol{Q} = \underset{n}{\begin{bmatrix}\boldsymbol{Q}_1 & \boldsymbol{Q}_2\end{bmatrix}}_{m-n} \in \mathbb{R}^{m\times m}$ is orthogonal, $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ is an upper triangular matrix with positive diagonal entries, and $\boldsymbol{Z} \in \mathbb{Z}^{n\times n}$ is a unimodular matrix whose

i-th row $\boldsymbol{z}_i^{\mathsf{T}}$ satisfies

$$\boldsymbol{z}_i^{\mathsf{T}} = \boldsymbol{e}_j^{\mathsf{T}} \tag{3.30}$$

for any $i \in \mathcal{J}$ and some $j \in \{1, 2, \ldots, n\}$. Here, without bringing in any inconvenience, we do not take $\boldsymbol{z}_i^{\mathsf{T}} = -\boldsymbol{e}_j^{\mathsf{T}}$, although it is allowed in (3.27). Then the POBILS problem (3.14) is reduced to

$$\min_{\boldsymbol{z} \in \bar{\mathcal{P}}} \|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2, \tag{3.31}$$

where $\bar{\boldsymbol{y}} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{y}$, $\boldsymbol{z} = \boldsymbol{Z}^{-1}\boldsymbol{x}$, and

$$\bar{\mathcal{P}} = \left\{ \boldsymbol{z} \mid \boldsymbol{z} \in \mathbb{Z}^n;\ l_i \leq z_{p(i)} \leq u_i\ \forall i \in \mathcal{J} \right\}$$

with $p(i) = j$ for each pair of (i, j) in (3.30), c.f., (3.28).

In Section 3.2.2, we see that an IGT $\boldsymbol{Z}_{ij}$ can be applied in reducing matrix $\boldsymbol{A}$ in (3.14) if $i \in \mathcal{I}$. It is easy to see that such a $\boldsymbol{Z}_{ij}$ satisfies (3.30). Also from (3.30), we see that if there is no inactive constraint in (3.14), i.e., $\mathcal{I} = \phi$ and $\mathcal{J} = \{1, 2, \ldots, n\}$, then the QRZ factorization (3.29) becomes the QRP factorization (3.2). Theorem 3 indicates that the larger the inactive set, the more freedom we have in choosing $\boldsymbol{Z}$ in (3.29). If all the constraints are inactive, i.e., $\mathcal{I} = \{1, 2, \ldots, n\}$ and $\mathcal{J} = \mathcal{I}^{\mathsf{c}} = \phi$, then $\boldsymbol{Z}$ can be any unimodular matrix and (3.29) becomes (2.2), the regular QRZ factorization for OILS problems.

Now the questions are: how do we find an inactive set $\mathcal{I}$ as large as possible for a given BILS problem, and based on the inactive set found, how do we construct an appropriate $\boldsymbol{Z}$ that satisfies (3.27) to reduce the lattice? These questions are to be addressed in the following sections.

3.3 Finding Inactive Constraints

Given a valid search radius ρ, the optimal solution of a BILS problem is in the intersection of (3.10a) and (3.10b). After an inactive set of constraints are removed, the optimal solution does not change. Note that removing an inactive set of constraints may or may not introduce new points into the search region (3.10). For the efficiency of sphere decoding, we want to find an inactive set of constraints such that removing the set will not introduce new points into the search region.

Base on a given search radius ρ (we will show how to find it later), we propose two efficient algorithms to find a set of inactive constraints: the level-1 inactive-set-finding algorithm (ISF-1) and the level-2 inactive-set-finding algorithm (ISF-2). Both of the two algorithms are based on the triangular form of the BILS problem (3.4), which can be obtained by applying the QR factorization from the original BILS problem (3.1). For the convenience of algorithmic descriptions, we represent inactive sets using Boolean vectors.

Definition 3–3 (inactive vector)**.** For an inactive set $\mathcal{I}$, we define a Boolean vector $\mathfrak{s} = (\mathfrak{s}_i)_n \in \{0,1\}^n$ such that $\mathfrak{s}_i = 1$ if $i \in \mathcal{I}$, and $\mathfrak{s}_i = 0$ otherwise. We call the vector $\mathfrak{s}$ an inactive vector.

3.3.1 The level-1 inactive-set-finding (ISF-1) algorithm

For the BILS problem (3.4), suppose that we can find the smallest box $\left\{\boldsymbol{z} \mid \boldsymbol{z} \in \mathbb{Z}^n,\ \hat{\boldsymbol{l}} \le \boldsymbol{z} \le \hat{\boldsymbol{u}}\right\}$ that covers all the integer points in the search ellipsoid (3.10a), see Figure 3–3. It can be seen that the constraint $z_i \le \bar{u}_i$ is redundant if $\hat{u}_i \le \bar{u}_i$, and the constraint $z_i \ge \bar{l}_i$ is redundant if $\hat{l}_i \ge \bar{l}_i$. Thus $\mathfrak{s} = (\hat{\boldsymbol{l}} \ge \bar{\boldsymbol{l}}) \wedge (\hat{\boldsymbol{u}} \le \bar{\boldsymbol{u}})$ is an inactive vector. In the example shown in Figure 3–3, we have $\mathfrak{s} = \begin{bmatrix}1 & 0\end{bmatrix}^{\mathsf{T}}$. Actually,

removing the interval constraint $\bar{l}_1 \leq z_1 \leq \bar{u}_1$ does not introduce new integer points into the search region.

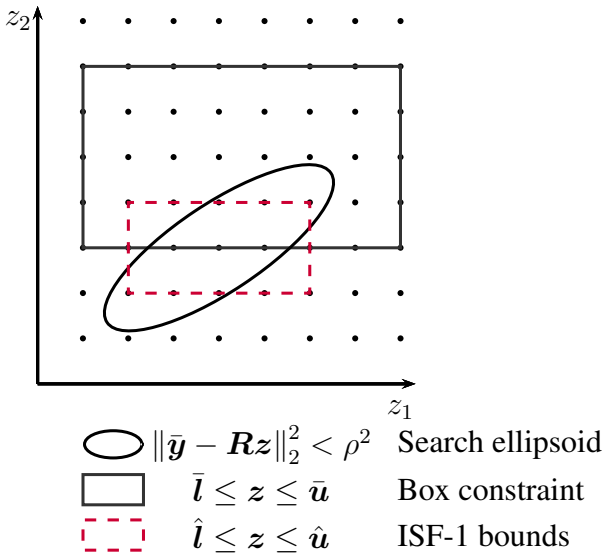


Figure 3–3: Example for ISF-1

Computing the covering box $[\hat{\boldsymbol{l}}, \hat{\boldsymbol{u}}]$ actually amounts to determine the range of each entry of $\boldsymbol{z} \in \mathbb{Z}^n$ in the ellipsoid (3.10a). As in Section 3.1.2, let $\boldsymbol{F} = \begin{bmatrix} \boldsymbol{f}_1 & \boldsymbol{f}_2 & \dots & \boldsymbol{f}_n \end{bmatrix} = \boldsymbol{R}^{-\mathsf{T}}$, and let $\check{\boldsymbol{z}}$ be the unconstrained real least squares solution of (3.4), i.e., $\check{\boldsymbol{z}} = \boldsymbol{F}^{\mathsf{T}}\bar{\boldsymbol{y}}$. For any $\boldsymbol{z}$ satisfying (3.10a), define vector $\boldsymbol{d} = (\boldsymbol{R}\boldsymbol{z} - \bar{\boldsymbol{y}})/\rho$. Then we have $\|\boldsymbol{d}\|_2 \leq 1$ and

$$\boldsymbol{z} = \check{\boldsymbol{z}} + \rho \boldsymbol{F}^{\mathsf{T}} \boldsymbol{d}. \tag{3.32}$$

Equation (3.32) can give us the range of possible values for $\boldsymbol{z}$. Multiplying both sides of (3.32) by $\boldsymbol{e}_i^{\mathsf{T}}$ form the left, we have

$$z_i = \boldsymbol{e}_i^{\mathsf{T}}(\check{\boldsymbol{z}} + \rho \boldsymbol{F}^{\mathsf{T}} \boldsymbol{d}) = \check{z}_i + \rho \boldsymbol{f}_i^{\mathsf{T}} \boldsymbol{d}. \tag{3.33}$$

Using the Cauchy-Schwarz inequality, we have $\left|\boldsymbol{f}_i^{\mathsf{T}}\boldsymbol{d}\right| \le \|\boldsymbol{f}_i\|_2 \|\boldsymbol{d}\|_2 \le \|\boldsymbol{f}_i\|_2$. Thus, from (3.33) we have $\check{z}_i - \rho\|\boldsymbol{f}_i\|_2 \le z_i \le \check{z}_i + \rho\|\boldsymbol{f}_i\|_2$, leading to the covering box:

$$\hat{\boldsymbol{l}} \le \boldsymbol{z} \le \hat{\boldsymbol{u}}, \quad \hat{l}_i = \lceil \check{z}_i - \rho\|\boldsymbol{f}_i\|_2 \rceil, \ \hat{u}_i = \lfloor \check{z}_i + \rho\|\boldsymbol{f}_i\|_2 \rfloor, \ i = 1, 2 \ldots, n. \tag{3.34}$$

In [23], the same bound as (3.34) has been deduced for solving the ellipsoid-constrained ILS problems. Note that this method of finding an inactive set checks the constraints on each z_i separately. The determination of one inactive constraint has no effect on the determination of any other inactive constraint. We call this algorithm the level-1 inactive-set-finding (ISF-1) algorithm.

The pseudocode of computing $\hat{\boldsymbol{l}} = (\hat{l}_i)_n$ and $\hat{\boldsymbol{u}} = (\hat{u}_i)_n$ is given in Algorithm 3–2. The pseudocode of the ISF-1 algorithm is given in Algorithm 3–3. Here, we could put Algorithm 3–2 and Algorithm 3–3 into one algorithm. But for later uses, we separate them. The main cost of ISF-1 comes from the computation of $\boldsymbol{F}$, which needs $O(n^3)$ flops.

Algorithm 3–2 Compute $\hat{\boldsymbol{l}}$ and $\hat{\boldsymbol{u}}$

function $[\hat{\boldsymbol{l}}, \hat{\boldsymbol{u}}] = \text{ISF1-bounds}(\boldsymbol{F}, \check{\boldsymbol{z}}, \rho)$

1: $\tilde{\boldsymbol{f}} = [\|\boldsymbol{f}_1\|_2 \ \ \|\boldsymbol{f}_2\|_2 \ \ \ldots \ \ \|\boldsymbol{f}_n\|_2]^{\mathsf{T}}$; ▷ $\tilde{f}_i = \|\boldsymbol{f}_i\|_2$

2: $\hat{\boldsymbol{l}} = \lceil \check{\boldsymbol{z}} - \rho\tilde{\boldsymbol{f}} \rceil$ and $\hat{\boldsymbol{u}} = \lfloor \check{\boldsymbol{z}} + \rho\tilde{\boldsymbol{f}} \rfloor$; ▷ see (3.34)

Algorithm 3–3 ISF-1: level-1 inactive-set-finding algorithm

function $\mathfrak{s} = \text{ISF-1}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \rho)$

1: $\boldsymbol{F} = \boldsymbol{R}^{-\mathsf{T}}$ and $\check{\boldsymbol{z}} = \boldsymbol{R}^{-1}\bar{\boldsymbol{y}}$;

2: $[\hat{\boldsymbol{l}}, \hat{\boldsymbol{u}}] = \text{ISF1-bounds}(\boldsymbol{F}, \check{\boldsymbol{z}}, \rho)$;

3: $\mathfrak{s} = \left(\hat{\boldsymbol{l}} \ge \bar{\boldsymbol{l}}\right) \wedge (\hat{\boldsymbol{u}} \le \bar{\boldsymbol{u}})$;

3.3.2 The level-2 inactive-set-finding (ISF-2) algorithm

In this section, we propose another algorithm which will be referred as the level-2 inactive-set-finding (ISF-2) algorithm that can find a larger inactive set than ISF-1. In ISF-1 we compute the range of each entry z_i in the hyper-ellipsoid (3.10a) separately. However, the range of possible values of z_i in the actual search region is usually smaller because it is affected by some other interval constraints. In the example shown in Figure 3–4, it can be seen that the range of z_1 within the box constraint $\bar{\mathcal{B}}$ is $[2, 8]$. The range $[\hat{l}_1, \hat{u}_1]$ for z_1 found by ISF-1 is $[1, 6]$ and thus constant $\bar{l}_1 \leq z_1 \leq \bar{u}_1$ cannot be determined as inactive by ISF-1. Notice that in the box constraint, $\bar{l}_2 = 3$. From the figure, for any integer point $\begin{bmatrix} z_1 & z_2 \end{bmatrix}^{\mathsf{T}}$ within the ellipse satisfying $z_2 \geq \bar{l}_2 = 3$, we must have $3 \leq z_1 \leq 6$. So, removing the interval constraint $\bar{l}_1 \leq z_1 \leq \bar{u}_1$ will not introduce any new integer point into the actual search region. This indicates that $\mathfrak{s} = \begin{bmatrix} 1 & 0 \end{bmatrix}^{\mathsf{T}}$ is an inactive vector for this BILS problem. Here, we can use the search ellipsoid and one interval constraint to determine whether another interval constraint can be removed. In the following, we first present a general method to do this. Then, we propose the ISF-2 algorithm which finds an inactive set based on this method. Compared with the ISF-1 algorithm, the ISF-2 algorithm can find a bigger inactive set.

Let $\boldsymbol{z}$ be an integer point in the hyper-ellipsoid (3.10a). For any pair of indices i, k with $i, k \in \{1, 2, \ldots, n\}$ and $i \neq k$, suppose z_k satisfies the interval constraint $\bar{l}_k \leq z_k \leq \bar{u}_k$, we would like to determine the range of z_i. The integer point $\boldsymbol{z}$ should

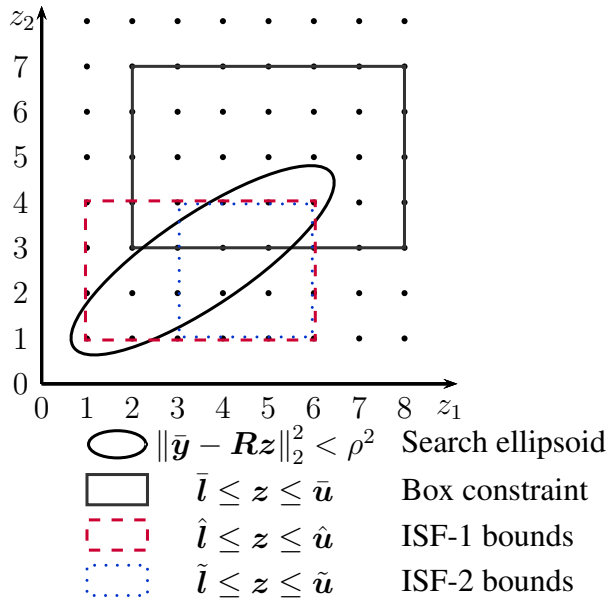


Figure 3–4: Example for ISF-2

satisfy (3.32). Multiplying both sides of (3.32) by $\begin{bmatrix}\boldsymbol{e}_k & \boldsymbol{e}_i\end{bmatrix}^{\mathsf{T}}$ from the left, we obtain

$$\begin{bmatrix} z_k \\ z_i \end{bmatrix} = \begin{bmatrix}\boldsymbol{e}_k & \boldsymbol{e}_i\end{bmatrix}^{\mathsf{T}} (\check{\boldsymbol{z}} + \rho \boldsymbol{F}^{\mathsf{T}} \boldsymbol{d}) = \begin{bmatrix} \check{z}_k \\ \check{z}_i \end{bmatrix} + \rho \begin{bmatrix}\boldsymbol{f}_k & \boldsymbol{f}_i\end{bmatrix}^{\mathsf{T}} \boldsymbol{d}. \tag{3.35}$$

Let the QR factorization of $\begin{bmatrix}\boldsymbol{f}_k & \boldsymbol{f}_i\end{bmatrix}$ be

$$\begin{bmatrix}\boldsymbol{f}_k & \boldsymbol{f}_i\end{bmatrix} = \begin{bmatrix}\boldsymbol{Q}_1^{(ki)} & \boldsymbol{Q}_2^{(ki)}\end{bmatrix} \begin{bmatrix}\boldsymbol{S}^{(ki)} \\ \boldsymbol{0}\end{bmatrix},$$

where $\boldsymbol{Q}^{(ki)} = \begin{bmatrix}\underset{2}{\boldsymbol{Q}_1^{(ki)}} & \underset{n-2}{\boldsymbol{Q}_2^{(ki)}}\end{bmatrix} \in \mathbb{R}^{n\times n}$ is orthogonal, $\boldsymbol{S}^{(ki)} = \begin{bmatrix} s_{11}^{(ki)} & s_{12}^{(ki)} \\ & s_{22}^{(ki)} \end{bmatrix} = \mathbb{R}^{2\times 2}$ is an upper triangular matrix. Without loss of generality, we assume $s_{11}^{(ki)}, s_{22}^{(ki)} > 0$.

Let $\boldsymbol{T} = (t_{ij})_{n\times n} = \boldsymbol{F}^{\mathsf{T}}\boldsymbol{F}$. We can easily show

$$s_{11}^{(ki)} = \|\boldsymbol{f}_k\|_2 = \sqrt{t_{kk}}, \tag{3.36a}$$

$$s_{12}^{(ki)} = \boldsymbol{f}_k^{\mathsf{T}}\boldsymbol{f}_i / \|\boldsymbol{f}_k\|_2 = t_{ki}/\sqrt{t_{kk}}, \tag{3.36b}$$

$$s_{22}^{(ki)} = \sqrt{\|\boldsymbol{f}_i\|_2^2 - \left(\boldsymbol{f}_k^{\mathsf{T}}\boldsymbol{f}_i / \|\boldsymbol{f}_k\|_2\right)^2} = \sqrt{t_{ii} - t_{ki}^2/t_{kk}}. \tag{3.36c}$$

Define $\bar{\boldsymbol{d}} = \begin{bmatrix}\bar{d}_1 & \bar{d}_2\end{bmatrix}^{\mathsf{T}} = \boldsymbol{Q}_1^{(ki)\mathsf{T}}\boldsymbol{d}$. It can be seen that

$$\begin{bmatrix}\boldsymbol{f}_k & \boldsymbol{f}_i\end{bmatrix}^{\mathsf{T}}\boldsymbol{d} = \begin{bmatrix}\boldsymbol{f}_k & \boldsymbol{f}_i\end{bmatrix}^{\mathsf{T}}\boldsymbol{Q}^{(ki)}\boldsymbol{Q}^{(ki)\mathsf{T}}\boldsymbol{d} = \begin{bmatrix}\boldsymbol{S}^{(ki)} \\ \boldsymbol{0}\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}\boldsymbol{Q}_1^{(ki)\mathsf{T}}\boldsymbol{d} \\ \boldsymbol{Q}_2^{(ki)\mathsf{T}}\boldsymbol{d}\end{bmatrix} = \boldsymbol{S}^{(ki)^{\mathsf{T}}}\bar{\boldsymbol{d}}.$$

Thus (3.35) can be rewritten as

$$z_k = \check{z}_k + \rho\bar{d}_1 s_{11}^{(ki)}, \tag{3.37a}$$

$$z_i = \check{z}_i + \rho\bar{d}_1 s_{12}^{(ki)} + \rho\bar{d}_2 s_{22}^{(ki)}. \tag{3.37b}$$

Since $\bar{l}_k \le z_k \le \bar{u}_k$ and z_k is an integer, from (3.37a) we have $\check{z}_k + \rho\bar{d}_1 s_{11}^{(ki)} \in \left\{\bar{l}_k, \bar{l}_k + 1, \ldots, \bar{u}_k\right\}$. Let $z_k^{\mathrm{r}} = \text{median}\left(\lfloor\check{z}_k\rceil, \bar{l}_k, \bar{u}_k\right)$. Since z_k^{r} is the closest element in $\left\{\bar{l}_k, \bar{l}_k + 1, \ldots, \bar{u}_k\right\}$ to $\check{z}_k$, we must have $\left|\rho\bar{d}_1 s_{11}^{(ki)}\right| \ge |z_k^{\mathrm{r}} - \check{z}_k|$. Then, by (3.36a), we have

$$\bar{d}_1^2 \ge \left(\left(z_k^{\mathrm{r}} - \check{z}_k\right)/\left(\rho s_{11}^{(ki)}\right)\right)^2 = \left(z_k^{\mathrm{r}} - \check{z}_k\right)^2/\left(\rho^2 t_{kk}\right).$$

Since $\left\|\bar{\boldsymbol{d}}\right\|_2 \le \left\|\boldsymbol{Q}_1^{(ki)}\right\|_2 \|\boldsymbol{d}\|_2 \le 1$, we have $\bar{d}_1^2 + \bar{d}_2^2 \le 1$. So, we have the following inequality:

$$\left|\bar{d}_2\right| \le \sqrt{1 - \frac{\left(z_k^{\mathrm{r}} - \check{z}_k\right)^2}{\rho^2 t_{kk}}}. \tag{3.38}$$

Since $\bar{l}_k \le z_k \le \bar{u}_k$, from (3.36a) and (3.37a) we have

$$\frac{\bar{l}_k - \check{z}_k}{\rho\sqrt{t_{kk}}} \le \bar{d}_1 \le \frac{\bar{u}_k - \check{z}_k}{\rho\sqrt{t_{kk}}}. \tag{3.39}$$

Define two non-negative numbers $t_{ki}^+ = \max(t_{ki}, 0)$ and $t_{ki}^- = \max(-t_{ki}, 0)$. We can rewrite (3.36b) as $s_{12}^{(ki)} = t_{ki}^+/\sqrt{t_{kk}} - t_{ki}^-/\sqrt{t_{kk}}$. Then, from (3.39), we have

$$\begin{aligned}
\rho\bar{d}_1 s_{12}^{(ki)} &\ge \frac{\left(\bar{l}_k - \check{z}_k\right) t_{ki}^+}{t_{kk}} - \frac{\left(\bar{u}_k - \check{z}_k\right) t_{ki}^-}{t_{kk}} = \frac{\bar{l}_k t_{ki}^+ - \bar{u}_k t_{ki}^- - \check{z}_k t_{ki}}{t_{kk}}, \\
\rho\bar{d}_1 s_{12}^{(ki)} &\le \frac{\left(\bar{u}_k - \check{z}_k\right) t_{ki}^+}{t_{kk}} - \frac{\left(\bar{l}_k - \check{z}_k\right) t_{ki}^-}{t_{kk}} = \frac{\bar{u}_k t_{ki}^+ - \bar{l}_k t_{ki}^- - \check{z}_k t_{ki}}{t_{kk}}.
\end{aligned} \tag{3.40}$$

Now from (3.37b), (3.36c), (3.38) and (3.40), we have

$$\tilde{l}_{ki} \le z_i \le \tilde{u}_{ki}, \tag{3.41}$$

where

$$\tilde{l}_{ki} = \left\lceil \check{z}_i + \frac{\bar{l}_k t_{ki}^+ - \bar{u}_k t_{ki}^- - \check{z}_k t_{ki}}{t_{kk}} - \sqrt{\rho^2 - \frac{\left(z_k^{\mathrm{r}} - \check{z}_k\right)^2}{t_{kk}}}\sqrt{t_{ii} - \frac{t_{ki}^2}{t_{kk}}} \right\rceil, \tag{3.42a}$$

$$\tilde{u}_{ki} = \left\lfloor \check{z}_i + \frac{\bar{u}_k t_{ki}^+ - \bar{l}_k t_{ki}^- - \check{z}_k t_{ki}}{t_{kk}} + \sqrt{\rho^2 - \frac{\left(z_k^{\mathrm{r}} - \check{z}_k\right)^2}{t_{kk}}}\sqrt{t_{ii} - \frac{t_{ki}^2}{t_{kk}}} \right\rfloor. \tag{3.42b}$$

Here we want to emphasize that the interval constraint we used to derive (3.41) is $\bar{l}_k \le z_k \le \bar{u}_k$. Thus, if there exist i, k_1, k_2 where $i \ne k_1, k_2$ such that $\tilde{l}_{k_2 i} \ge \bar{l}_i$ and $\tilde{u}_{k_1 i} \le \bar{u}_i$, then removing the i-th interval constraint $\bar{l}_i \le z_i \le \bar{u}_i$ will not change the search region as long as the k_1-th and k_2-th interval constraints are preserved.

We assumed $k \neq i$ in the derivation of (3.41). If we set $k = i$, we obtain

$$\frac{\bar{l}_k t_{ki}^+ - \bar{u}_k t_{ki}^+ - \check{z}_k t_{ki}}{t_{kk}} = \bar{l}_i - \check{z}_i,$$
$$\frac{\bar{u}_k t_{ki}^+ - \bar{l}_k t_{ki}^+ - \check{z}_k t_{ki}}{t_{kk}} = \bar{u}_i - \check{z}_i,$$

and $t_{ii} - t_{ki}^2/t_{kk} = 0$. Then, from (3.42), we have

$$\tilde{l}_{ii} = \bar{l}_i, \quad \tilde{u}_{ii} = \bar{u}_i. \tag{3.43}$$

This indicates that when $k = i$, (3.41) is just the original interval constraint on $\boldsymbol{z}_i$.

Define $\tilde{\boldsymbol{L}} = (\tilde{l}_{ij})_{n\times n}$ and $\tilde{\boldsymbol{U}} = (\tilde{u}_{ij})_{n\times n}$. The pseudocode for computing $\tilde{\boldsymbol{L}}$ and $\tilde{\boldsymbol{U}}$ using (3.42) is given in Algorithm 3–4. The main cost of this algorithm is the cost of computing $\boldsymbol{T}$, which needs $O(n^3)$ flops. The cost of all other computations is $O(n^2)$ flops.

Algorithm 3–4 Compute $\tilde{\boldsymbol{L}}$ and $\tilde{\boldsymbol{U}}$

function $[\tilde{\boldsymbol{L}}, \tilde{\boldsymbol{U}}] = \text{ISF2-bounds}(\boldsymbol{F}, \check{\boldsymbol{z}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \rho)$

1: $\boldsymbol{z}^{\mathrm{r}} = \max\left(\min\left(\lfloor \check{\boldsymbol{z}} \rceil, \bar{\boldsymbol{u}}\right), \bar{\boldsymbol{l}}\right)$;
2: $\boldsymbol{T} = \boldsymbol{F}^{\mathsf{T}}\boldsymbol{F}$;
3: $\boldsymbol{T}^+ = \max(\boldsymbol{T}, \boldsymbol{0})$, $\boldsymbol{T}^- = \max(-\boldsymbol{T}, \boldsymbol{0})$;
4: $\boldsymbol{B} = (b_{ki})_{n\times n}$ with $b_{ki} = \sqrt{\rho^2 - \left(z_k^{\mathrm{r}} - \check{z}_k\right)^2 / t_{kk}}\sqrt{t_{ii} - t_{ki}^2/t_{kk}}$;
5: $\tilde{\boldsymbol{L}} = (\tilde{l}_{ki})_{n\times n}$ with $\tilde{l}_{ki} = \left\lceil \check{z}_i + \left(\bar{l}_k t_{ki}^+ - \bar{u}_k t_{ki}^+ - \check{z}_k t_{ki}\right)/t_{kk} - b_{ki} \right\rceil$; ▷ see (3.42a)
6: $\tilde{\boldsymbol{U}} = (\tilde{u}_{ki})_{n\times n}$ with $\tilde{u}_{ki} = \left\lfloor \check{z}_i + \left(\bar{u}_k t_{ki}^+ - \bar{l}_k t_{ki}^+ - \check{z}_k t_{ki}\right)/t_{kk} + b_{ki} \right\rfloor$; ▷ see (3.42b)

Next, we present an algorithm to construct an inactive vector $\boldsymbol{\mathfrak{s}}$ using $\tilde{\boldsymbol{L}}$, $\tilde{\boldsymbol{U}}$. Write $\tilde{\boldsymbol{L}} = \begin{bmatrix}\tilde{\boldsymbol{l}}_1 & \tilde{\boldsymbol{l}}_2 & \dots & \tilde{\boldsymbol{l}}_n\end{bmatrix}$ and $\tilde{\boldsymbol{U}} = \begin{bmatrix}\tilde{\boldsymbol{u}}_1 & \tilde{\boldsymbol{u}}_2 & \dots & \tilde{\boldsymbol{u}}_n\end{bmatrix}$. First we construct the following Boolean vectors:

$$\mathfrak{l}_i = \left(\tilde{\boldsymbol{l}}_i < \boldsymbol{1}_n \bar{l}_i\right), \quad \mathfrak{u}_i = \left(\tilde{\boldsymbol{u}}_i > \boldsymbol{1}_n \bar{u}_i\right), \quad \text{for } i = 1, 2, \dots, n. \tag{3.44}$$

Because of (3.43), $\mathfrak{l}_i, \mathfrak{u}_i \neq \mathbf{1}_n$ for all i.

Theorem 4. *For the BILS problem* (3.4)*, define a set of Boolean vectors*

$$\mathfrak{B} = \big\{\mathfrak{l}_i \mid \hat{l}_i < \bar{l}_i, i \in \{1, 2, \ldots, n\}\big\} \cup \big\{\mathfrak{u}_i \mid \hat{u}_i > \bar{u}_i, i \in \{1, 2, \ldots, n\}\big\}, \tag{3.45}$$

where $\mathfrak{l}_i$*,* $\mathfrak{u}_i$ *are defined in* (3.44)*, and* $\hat{l}_i$*,* $\hat{u}_i$ *are defined in* (3.34)*. Any Boolean vector* $\mathfrak{s} \in \{0, 1\}^n$ *which satisfies*

$$\mathfrak{s} \vee \mathfrak{b} \neq \mathbf{1}_n, \quad \forall \mathfrak{b} \in \mathfrak{B} \tag{3.46}$$

is an inactive vector.

Proof. First we give a remark. Because $\mathfrak{b} \neq \mathbf{1}_n$ for any $\mathfrak{b} \in \mathfrak{B}$, there exists at least one $\mathfrak{s}$ such that (3.46) holds (i.e., $\mathfrak{s} = \mathbf{0}_n$).

Let i be any index such that $\mathfrak{s}_i = 1$. If $\hat{l}_i \geq \bar{l}_i$, obviously the lower bound $\bar{l}_i \leq z_i$ can be removed from the box constraint without affecting the solution to (3.4) (see Section 3.3.1). If $\hat{l}_i < \bar{l}_i$, by (3.45), $\mathfrak{l}_i \in \mathfrak{B}$. By (3.46) we have $\mathfrak{s} \vee \mathfrak{l}_i \neq \mathbf{1}_n$, which indicates that there must exist $k \neq i$ such that $\mathfrak{s}_k = 0$ and $\boldsymbol{e}_k^{\mathsf{T}} \mathfrak{l}_i = 0$ ($\boldsymbol{e}_k^{\mathsf{T}} \mathfrak{l}_i$ is the k-th entry of $\mathfrak{l}_i$), implying $\tilde{l}_{ki} \geq \bar{l}_i$ by (3.44). From (3.41), we can still claim that the lower bound $\bar{l}_i \leq z_i$ can be removed without affecting the solution to (3.4). Here, we would like to emphasize that the k-th interval constraint corresponding to $\mathfrak{s}_k = 0$ will not be removed.

Similarly, for any i such that $\mathfrak{s}_i = 1$, we can prove that the upper bound $z_i \leq \bar{u}_i$ can be removed without affecting the solution to (3.4). Then by Definition 3–3, we know that $\mathfrak{s}$ is an inactive vector, completing the proof. □

In the following we use an example to illustrate Theorem 4.

Example 3–1. *Suppose in the BILS problem* (3.4)*,* $n = 4$*,* $\bar{\boldsymbol{l}} = \mathbf{0}_4$ *and* $\bar{\boldsymbol{u}} = 7 \times \mathbf{1}_4$*. And suppose after applying Algorithm 3–2 and Algorithm 3–4, we obtain*

$$\hat{\boldsymbol{l}} = \begin{bmatrix} -1 \\ -2 \\ 1 \\ 0 \end{bmatrix}, \ \hat{\boldsymbol{u}} = \begin{bmatrix} 8 \\ 6 \\ 9 \\ 9 \end{bmatrix}, \ \tilde{\boldsymbol{L}} = \begin{bmatrix} 0 & -2 & 2 & -1 \\ 1 & 0 & -2 & 0 \\ -1 & -1 & 0 & -3 \\ 0 & 1 & -3 & 0 \end{bmatrix}, \ \tilde{\boldsymbol{U}} = \begin{bmatrix} 7 & 9 & 8 & 6 \\ 8 & 7 & 9 & 11 \\ 5 & 9 & 7 & 8 \\ 12 & 9 & 10 & 7 \end{bmatrix}. \quad (3.47)$$

The ISF-1 algorithm (Algorithm 3–3) cannot find any inactive constraints since $\big(\hat{\boldsymbol{l}} \geq \bar{\boldsymbol{l}}\big) \wedge (\hat{\boldsymbol{u}} \leq \bar{\boldsymbol{u}}) = \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}^{\mathsf{T}} \wedge \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^{\mathsf{T}} = \mathbf{0}$*. However, from* (3.45)*, we have*

$$\mathfrak{B} = \{\mathfrak{l}_1, \mathfrak{l}_2, \mathfrak{u}_1, \mathfrak{u}_3, \mathfrak{u}_4\} = \left\{ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right\}. \quad (3.48)$$

From Theorem 4, vector $\mathfrak{s} = \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}^{\mathsf{T}}$ *satisfies* (3.46) *and thus* $\{1, 2\}$ *is an inactive set. This can be verified as follows. Let* $\boldsymbol{z}$ *be any integer point in the ellipsoid* (3.10a)*. From* (3.34)*, we have*

$$-1 = \hat{l}_1 \leq z_1 \leq \hat{u}_1 = 8, \quad -2 = \hat{l}_1 \leq z_2 \leq \hat{u}_2 = 6.$$

Since the interval constraint on z_3 *is preserved, from* (3.41)*, we have*

$$-1 = \tilde{l}_{31} \leq z_1 \leq \tilde{u}_{31} = 5, \quad -1 = \tilde{l}_{32} \leq z_2 \leq \tilde{u}_{32} = 9.$$

Since the interval constraint on z_4 *is preserved, again from* (3.41)*, we have*

$$0 = \tilde{l}_{41} \leq z_1 \leq \tilde{u}_{41} = 12, \quad 1 = \tilde{l}_{42} \leq z_2 \leq \tilde{u}_{42} = 9.$$

From the above inequalities, it follows that

$$0 \leq z_1 \leq 5, \quad 1 \leq z_2 \leq 6.$$

Thus the 1st interval constraint $0 \leq z_1 \leq 7$ *and the 2nd interval constraint* $0 \leq z_2 \leq 7$ *are inactive at the same time, i.e., they form an inactive set.*

To get a large inactive set, ideally we would like to find the $\mathfrak{s}$ satisfing (3.46) with the largest number of 1's. However, to find such an $\mathfrak{s}$ is actually to solve a hitting set problem, which is known to be NP-hard [29]. Here we give a polynomial time greedy algorithm that finds a suboptimal solution.

First, from Theorem 4, we must have $\mathfrak{s} \vee \mathfrak{b} \neq \mathbf{1}_n$ for any $\mathfrak{b} \in \mathfrak{B}$. Thus, if $\mathfrak{b} = \mathbf{1}_n - \boldsymbol{e}_i$ for some $\mathfrak{b} \in \mathfrak{B}$ and $i \in \{1, 2, \ldots, n\}$, then we must have $\mathfrak{s}_i = 0$. From this observation, we first initialize $\mathfrak{s} = (\mathfrak{s}_i)_n$ such that

$$\mathfrak{s}_i = \begin{cases} 0 & \text{if } (\mathbf{1}_n - \boldsymbol{e}_i) \in \mathfrak{B}, \\ 1 & \text{otherwise} \end{cases}, \quad i = 1, 2, \ldots, n. \tag{3.49}$$

Then, we update $\mathfrak{B}$ by removing all $\mathfrak{b}$ from $\mathfrak{B}$ such that $\mathfrak{s} \vee \mathfrak{b} \neq \mathbf{1}_n$. If the updated $\mathfrak{B}$ is an empty set, then by Theorem 4, the current $\mathfrak{s}$ is an inactive vector. If not, then we must have $\mathfrak{s} \neq \mathbf{0}_n$ (because $\mathfrak{b} \neq \mathbf{1}_n$ for any $\mathfrak{b} \in \mathfrak{B}$). Thus we can always find an index k such that $\mathfrak{s}_k = 1$. Then we flip $\mathfrak{s}_k$ to 0 and remove all $\mathfrak{b}$ from $\mathfrak{B}$ where $\mathfrak{b}_k = 0$. Specifically, we choose

$$k = \underset{i \in \{1, \ldots, n\}}{\operatorname{argmin}} \sum_{\mathfrak{b} \in \mathfrak{B}} \mathfrak{b}_i, \tag{3.50}$$

so that the number of elements removed from $\mathfrak{B}$ is maximized in this step (a greedy strategy). This flipping-and-removing step is repeated until $\mathfrak{B}$ is empty. Note that

it is possible that we may have more than one choices for k in (3.50). In this case, it does not matter which value is chosen (in our implementation, we simple choose the smallest one).

In the following, we use an example to illustrate the above process of finding an inactive vector $\mathfrak{s}$.

Example 3–2. *Assume we have the same* $\hat{\boldsymbol{l}}$, $\hat{\boldsymbol{u}}$, $\tilde{\boldsymbol{L}}$ *and* $\tilde{\boldsymbol{U}}$ *as in* (3.47). *To find a vector* $\mathfrak{s}$ *that satisfies* (3.46), *we first initialize* $\mathfrak{s}$ *by* (3.49) *where* $\mathfrak{B}$ *is given in* (3.48). *Note that in* $\mathfrak{B}$, *only* $\mathfrak{u}_3 = \mathbf{1}_4 - \boldsymbol{e}_3$. *Thus we have* $\mathfrak{s} = \begin{bmatrix}1 & 1 & 0 & 1\end{bmatrix}^{\mathsf{T}}$. *Since* $\mathfrak{s} \vee \mathfrak{u}_2 \neq \mathbf{1}_4$ *and* $\mathfrak{s} \vee \mathfrak{u}_3 \neq \mathbf{1}_4$, *we remove* $\mathfrak{u}_2$ *and* $\mathfrak{u}_3$ *from* $\mathfrak{B}$, *giving the updated* $\mathfrak{B}$ *as follows:*

$$\mathfrak{B} = \left\{ \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\\0\end{bmatrix} \right\}.$$

For the above $\mathfrak{B}$, *we can easily see that* $\sum_{\mathfrak{b}\in\mathfrak{B}} \mathfrak{b}_i$ *is minimum when* $i = 4$. *So we set* $\mathfrak{s}_4 = 0$, *giving the updated* $\mathfrak{s} = \begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix}^{\mathsf{T}}$. *It is easy to check that* $\mathfrak{s} \vee \mathfrak{b} \neq \mathbf{1}_4$ *for all* $\mathfrak{b} \in \mathfrak{B}$, *so they can all be removed from* $\mathfrak{B}$, *leading to* $\mathfrak{B} = \phi$. *Then,* $\mathfrak{s} = \begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix}^{\mathsf{T}}$ *is our inactive vector. This* $\mathfrak{s}$ *is exactly the inactive vector we used in Example 3–1.*

A formal description of ISF-2 (the level 2 inactive-set-finding algorithm) is given in Algorithm 3–5. If Algorithm 3–5 is implemented properly, the initialization of $\mathfrak{B}$ and the first update of $\mathfrak{B}$ cost $O(n^2)$ flops, and in each iteration of the "while" loop, computing k costs $O(n^2)$ flops and the update of $\mathfrak{B}$ costs $O(n)$ flops. We know that in each iteration, one element of $\mathfrak{s}$ is flipped from 1 to 0. So the total number of iterations is not more than n. The total cost of the algorithm is thus $O(n^3)$ flops. In

our numerical experiments, we found that usually $\mathfrak{B}$ becomes empty after the first update (line 6 in Algorithm 3–5) and thus the while loop is usually not executed.

Algorithm 3–5 ISF-2: level-2 inactive-set-finding algorithm

function $\mathfrak{s} = \text{ISF-2}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \rho)$

1: $\boldsymbol{F} = \boldsymbol{R}^{-\mathsf{T}}$ and $\check{\boldsymbol{z}} = \boldsymbol{R}^{-1}\bar{\boldsymbol{y}}$;
2: $[\hat{\boldsymbol{l}}, \hat{\boldsymbol{u}}] = \text{ISF1-bounds}(\boldsymbol{F}, \check{\boldsymbol{z}}, \rho)$; ▷ see Algorithm 3–2
3: $[\tilde{\boldsymbol{L}}, \tilde{\boldsymbol{U}}] = \text{ISF2-bounds}(\boldsymbol{F}, \check{\boldsymbol{z}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \rho)$; ▷ see Algorithm 3–4
4: Construct $\mathfrak{B}$ using (3.45);
5: initialize $\mathfrak{s} = (\mathfrak{s}_i)_n$ using (3.49);
6: remove any $\mathfrak{b}$ from $\mathfrak{B}$, if $\mathfrak{s} \vee \mathfrak{b} \neq \mathbf{1}_n$;
7: **while** $\mathfrak{B} \neq \phi$ **do**
8: $\quad k = \text{argmin}_{i\in\{1,\ldots,n\}} \sum_{\mathfrak{b}\in\mathfrak{B}} \mathfrak{b}_i$;
9: $\quad \mathfrak{s}_k = 0$;
10: $\quad$ remove any $\mathfrak{b}$ from $\mathfrak{B}$, if $\mathfrak{b}_k = 0$;
11: **end while**

In the following, we make two remarks about Algorithm 3–5.

The cost of ISF-2 is of the same order as the cost of ISF-1. The reason that ISF-1 does not cost less is because, like ISF-2, it involves the computation of $\boldsymbol{R}^{-1}$, which needs $O(n^3)$ flops.

In this algorithm, we pursue only the objective of maximizing the size of the inactive set. This is because the larger the inactive set, the less restrictions we will have in choosing the unimodular matrix $\boldsymbol{Z}$ in (3.29). However, the impact of an inactive set on the lattice reduction not only depends on it size, but also depends on the nature of specific constraints in this inactive set, e.g., removing some interval constraints may lead to a better lattice reduction than removing some others. The reduction would be more effective if we could take both aspects into account. This is a challenging problem for future studies.

3.4 Computing an Initial Search Radius ρ

To find an inactive set of constraints for a BILS problem, both ISF-1 and ISF-2 introduced in the last section rely on a given search radius ρ. A small ρ is preferred as both ISF-1 and ISF-2 tend to find a larger inactive set with a smaller ρ. A small initial ρ also makes sphere decoding faster (see Section 3.1.3). However, ρ cannot be arbitrary small, as the optimal solution has to be in the search region (3.10). In this section, we consider how to find an initial ρ for a given BILS problem. We propose three methods for this purpose. All of the three methods involve finding an approximate solution of (3.4). In the first method, the approximate solution is computed based on the OILS relaxation of (3.4). In the second method, the box-constrained Babai point is used as the approximate solution. In the last method, the approximate solution is computed based on the real relaxation of (3.4). We will compare the three different methods in the numerical experiments, see Section 3.6.

3.4.1 An OILS relaxation based method

Suppose we have transformed the BILS problem (3.1) to the problem (3.4) by the standard QR factorization, i.e., $\boldsymbol{P} = \boldsymbol{I}_n$ in (3.2) and (3.3). Note that in this case, $\boldsymbol{z}$ in (3.4) can be replaced by $\boldsymbol{x}$.

In this method of computing ρ, we first remove the box constraint in (3.4) and get an OILS problem:

$$\min_{\boldsymbol{z}\in\mathbb{Z}^n} \|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2 .$$

Then, we solve the OILS problem and get the solution $\boldsymbol{z}^*$. For efficiency, we use the PLLL reduction [7] to reduce the OILS problem and use Schnorr-Euchner's sphere decoding algorithm to find $\boldsymbol{z}^*$, see Section 2.2. If $\boldsymbol{z}^*$ is in the box $\bar{\mathcal{B}}$, then certainly

$\boldsymbol{z}^*$ is also the solution of the BILS problem (3.4). Otherwise, we project $\boldsymbol{z}^*$ onto $\bar{\mathcal{B}}$, obtaining $\bar{\boldsymbol{z}}^* = \text{median}\left(\boldsymbol{z}^*, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}\right)$, the nearest integer point in $\bar{\mathcal{B}}$ to $\boldsymbol{z}^*$, which we use as an approximate solution to (3.4). Then we can define the initial search radius as $\rho = \|\bar{\boldsymbol{y}} - \boldsymbol{R}\bar{\boldsymbol{z}}^*\|_2$.

This method is more than just finding an initial ρ for solving a BILS problem (3.4). Actually, it provides a method for solving (3.4). We will formally describe this method in Algorithm 3–8 after we introduce a new reduction algorithm for (3.4).

3.4.2 A Babai point based method

Another way of finding ρ for the BILS problem (3.4) is to compute the box-constrained Babai point $\boldsymbol{z}^{\text{B}}$, an approximate solution to (3.4), which is often used in practice, see Section 3.1.3. Then, the residual of the Babai point, i.e., $\|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}^{\text{B}}\|_2$, can be used as the initial search radius ρ.

It is known that the AIP reduction introduced in Section 3.1.2 can reduce the residual of the Babai point for BILS problems [24]. Thus, to get a small search radius ρ, we apply the AIP reduction on (3.1) to get (3.4). Recall that the AIP reduction actually computes the Babai point $\boldsymbol{z}^{\text{B}}$ in the reduction process, see Algorithm 3–1.

3.4.3 A real relaxation based method

Assume $\boldsymbol{z}^*$ is the optimal solution of (3.4). Clearly the value of z_n^* is affected by all interval constraints $\bar{l}_i \le z_i \le \bar{u}_i$ for $i = 1, 2, \ldots, n$. In the previously introduced process to compute z_n^{B}, the last entry of the box constrained Babai point, we can see that the information in the interval constraint on z_n is used, i.e., $z_n^{\text{B}} = \text{median}\left(\lfloor \check{z}_n \rceil, \bar{l}_n, \bar{u}_n\right)$. However, in this computation, the information of other interval constraints are not taken into account at all.

Suppose we have transformed the BILS problem (3.1) to the problem (3.4) by the standard QR factorization. Here we propose a real relaxation based method which finds an approximate solution $\hat{\boldsymbol{z}} \in \mathbb{Z}^n$ based on $\tilde{\boldsymbol{z}} \in \mathbb{R}^n$, the solution of the real relaxation of the BILS problem (3.4), i.e.,

$$\tilde{\boldsymbol{z}} = \underset{\bar{\boldsymbol{l}} \le \boldsymbol{z} \le \bar{\boldsymbol{u}}}{\operatorname{argmin}} \|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2 . \tag{3.51}$$

The constrained real least squares solution $\tilde{\boldsymbol{z}}$ in (3.51) can be computed efficiently using various numerical algorithms (for classic algorithms, see [81, Chapter 16]). In this thesis, we use the algorithm proposed in [75], which is based on the gradient projection algorithm [13] combined with the standard active set strategy for solving constrained optimization problems, see, e.g., [81, Section 16.5] and [12].

Clearly, the computation of $\tilde{\boldsymbol{z}}$ uses information in all interval constraints. To find an approximate solution $\hat{\boldsymbol{z}}$ of (3.4), we let $\hat{z}_n = \lfloor \tilde{z}_n \rceil$. Then, like in the AIP reduction, we update $\bar{\boldsymbol{y}} = \bar{\boldsymbol{y}} - \boldsymbol{r}_n \hat{z}_n$ and reduce the dimension of (3.51) by 1 to get the following subproblem (see Section 3.1.2):

$$\min_{\bar{\boldsymbol{l}}_{1:k} \le \boldsymbol{z}_{1:k} \le \bar{\boldsymbol{u}}_{1:k}} \|\bar{\boldsymbol{y}}_{1:k} - \boldsymbol{R}_{1:k,1:k}\boldsymbol{z}_{1:k}\|_2^2 , \tag{3.52}$$

where $k = n - 1$. Then, by applying the above procedure recursively to (3.52), we determine $\hat{z}_k$ for $k = n-1, n-2, \ldots, 1$.

In the above process of finding $\hat{\boldsymbol{z}}$, we need to solve n real constrained least squares subproblems (3.52) of dimension k where $k = n, n-1, \ldots, 1$. Denote the solution of (3.52) by $\tilde{\boldsymbol{z}}^{(k)}$. When a numerical algorithm is used to find $\tilde{\boldsymbol{z}}^{(k)}$, an initial guess $\tilde{\boldsymbol{z}}_0^{(k)}$

needs to be provided. The better the initial guess is, the faster the algorithm converges [75]. Assume that we have computed $\tilde{\boldsymbol{z}}^{(k+1)} = \begin{bmatrix} \tilde{z}_1^{(k+1)} & \tilde{z}_2^{(k+1)} & \dots & \tilde{z}_{k+1}^{(k+1)} \end{bmatrix}^{\mathsf{T}}$ for the $(k+1)$-dimensional subproblem. We use only $\tilde{z}_{k+1}^{(k+1)}$ to determine $\hat{z}_{k+1}$, i.e., $\hat{z}_{k+1} = \left\lfloor \tilde{z}_{k+1}^{(k+1)} \right\rceil$. However, the computation of the other entries of $\tilde{\boldsymbol{z}}^{(k+1)}$ will not be wasted. We use $\tilde{\boldsymbol{z}}_0^{(k)} = \begin{bmatrix} \tilde{z}_1^{(k+1)} & \tilde{z}_1^{(k+1)} & \dots & \tilde{z}_k^{(k+1)} \end{bmatrix}^{\mathsf{T}}$ as the initial point to find $\tilde{\boldsymbol{z}}^{(k)}$. Likely, this $\tilde{\boldsymbol{z}}_0^{(k)}$ is a good initial point for finding $\tilde{\boldsymbol{z}}^{(k)}$.

Recall we mentioned before that the AIP reduction can reduce the residual of the Babai point $\boldsymbol{z}^{\mathrm{B}}$. To further improve the quality of $\hat{\boldsymbol{z}}$, we also blend the column interchange strategy of the AIP reduction into the above basic structure of the real relaxation based method. Specifically, before we determine the last entry of $\hat{\boldsymbol{z}}$, we first compute

$$j = \underset{i \in \{1,\dots,n\}}{\operatorname{argmax}} \left(\tilde{z}_i^{\mathrm{S}} - \tilde{z}_i \right) / \left\| \boldsymbol{f}_i \right\|_2^2, \tag{3.53}$$

where $\boldsymbol{f}_i$ is the i-th column of $\boldsymbol{F} = \boldsymbol{R}^{-\mathsf{T}}$ and $\tilde{z}_i^{\mathrm{S}}$ is the second closest integer in $[\bar{l}_i, \bar{u}_i]$ to $\tilde{z}_i$. Then, like in the AIP reduction (see Section 3.1.2), we interchange columns j and n of $\boldsymbol{R}$ and update $\bar{\boldsymbol{y}}$, $\bar{\boldsymbol{l}}$, $\bar{\boldsymbol{u}}$ and $\tilde{\boldsymbol{z}}$ accordingly. After that, we determine $\hat{z}_n$, and reduce the original problem to the subproblem (3.52) as described before. This column permutation strategy is recursively applied to (3.52).

Since column permutations are adopted, the resulting $\hat{\boldsymbol{z}}$ needs to be permuted back to give an approximate solution to the original BILS problem. We call this approximate solution as the R-approximate of (3.4), as it is computed based on the real relaxation of (3.4). An efficient implementation of the above algorithm to find an R-approximate $\hat{\boldsymbol{z}}$ and a search radius ρ is given in Algorithm 3–6.

Algorithm 3–6 Find an initial radius for a BILS problem

function $[\hat{\boldsymbol{z}}, \rho] = \text{R-approximate}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}})$

compute $\boldsymbol{F} = \boldsymbol{R}^{-\mathsf{T}}$, $\check{\boldsymbol{f}} = (\|\boldsymbol{f}_i\|_2^2)_n$ and initialize $\boldsymbol{P} = \boldsymbol{I}_n$, $\rho = 0$;
for $k = n$ to 1 **do**
  $\check{\boldsymbol{z}} = \boldsymbol{F}^{\mathsf{T}}\bar{\boldsymbol{y}} \in \mathbb{R}^k$;
  compute $\tilde{\boldsymbol{z}} = \text{argmin}_{\bar{\boldsymbol{l}} \le \boldsymbol{z} \le \bar{\boldsymbol{u}}} \|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2$;
  let $\tilde{z}_i^{\text{s}}$ be the second closest integer in $[\bar{l}_i, \bar{u}_i]$ to $\tilde{z}_i$, for $i = 1, 2, \ldots, k$;
  $j = \text{argmax}_{i \in \{1,\ldots,k\}} \left(\tilde{z}_i^{\text{s}} - \tilde{z}_i\right)^2 / \check{f}_i$ for $i = 1, 2, \ldots, k$; ▷ see (3.53)
  $\hat{z}_k = \lfloor \tilde{z}_j \rceil$;
  $\bar{\boldsymbol{y}} = \bar{\boldsymbol{y}} - \boldsymbol{r}_j \hat{z}_k$;
  remove column j of $\boldsymbol{F}$ and entry j of $\bar{\boldsymbol{l}}$, $\bar{\boldsymbol{u}}$ and $\check{\boldsymbol{f}}$;
  if $j \neq k$ **then**
    interchange columns j and k of $\boldsymbol{R}$ and $\boldsymbol{P}$;
    use Givens rotations to bring $\boldsymbol{R}$ back to upper triangular;
    apply the same Givens rotations to update $\boldsymbol{F}$ and $\bar{\boldsymbol{y}}$;
  end if
  $\rho = \rho + (r_{kk}(\hat{z}_k - \check{z}_k))^2$;
  remove column k and row k of $\boldsymbol{R}$; ▷ s.t. $\boldsymbol{R} \in \mathbb{R}^{k \times k}$
  update $\check{f}_i = \check{f}_i - f_{ki}^2$ for $i = 1, 2, \ldots, k$;
  remove row k of $\boldsymbol{F}$ and entry k in $\bar{\boldsymbol{y}}$; ▷ s.t. $\boldsymbol{F} \in \mathbb{R}^{k \times k}$ $\bar{\boldsymbol{y}} \in \mathbb{R}^k$
end for
$\hat{\boldsymbol{z}} = \boldsymbol{P}\hat{\boldsymbol{z}}$, $\rho = \sqrt{\rho}$;

3.5 New Reduction Algorithms for BILS Problems

In this section, we show how to transform the BILS problem (3.1) to a reduced POBILS problem (3.31) using the theories and algorithms introduced in the previous sections.

Assume we have a po-box constrained lattice $\Lambda_{\mathcal{P}}(\boldsymbol{A})$ as defined in (3.16). Recall that to reduce $\Lambda_{\mathcal{P}}(\boldsymbol{A})$, we want to compute the QRZ factorization (3.29) (see Section 3.2.4). Without lost of generality, we further assume that we have obtained an initial $\boldsymbol{R}$ in (3.29) by applying the QR factorization to $\boldsymbol{A}$ and have transformed the original lattice to $\Lambda_{\mathcal{P}}(\boldsymbol{R})$. In this section, we first propose a restricted lattice reduction

algorithm that lattice reduces the initial $\boldsymbol{R}$. This algorithm can be regarded as an extension of the LLL algorithm, see Algorithm 2–1. As the use of size reductions in the algorithm is restricted by the set $\mathcal{J}$ in (3.15), we refer to the algorithm as the restricted LLL (RLLL) algorithm.

In Sections 3.5.1 and 3.5.2, we describe two new size reduction operations that are used to update $\boldsymbol{R}$ in the RLLL algorithm. Then based on the two operations, we describe the RLLL algorithm in Section 3.5.3. Based on the RLLL algorithm, we give three versions of RLLL-based BILS reduction algorithms that transform the BILS problem (3.1) to a reduced POBILS problem (3.31) in Section 3.5.4. The main difference of these versions is that they compute different initial search radius ρ.

3.5.1 Extended size reduction

Recall that in the LLL reduction (see Algorithm 2–1), every off-diagonal entry r_{ik} is size reduced by IGT $\boldsymbol{Z}_{ik}$ defined in (2.5). In the RLLL reduction, however, an IGT $\boldsymbol{Z}_{ik}$ is only allowed if $i \notin \mathcal{J}$ (see Section 3.2.2). So, we would like to look at how to apply only the allowed IGTs $\boldsymbol{Z}_{ik}$ to reduce off-diagonal entries in $\boldsymbol{r}_k$, the k-th column of $\boldsymbol{R}$.

For a given index set $\mathcal{J}$, define the Boolean vector $\mathfrak{s} = (\mathfrak{s}_i)_n$ where $\mathfrak{s}_i = 1$ if $i \notin \mathcal{J}$ and $\mathfrak{s} = 0$ otherwise. Given $\mathfrak{s}$ and $k \in \{1, 2, \ldots, n\}$, we denote

$$\{i \mid \mathfrak{s}_i = 1, i < k\} = \{i_1, i_2, \ldots, i_p\},$$

where $1 \leq i_1 < i_2 < \cdots < i_p \leq k-1$. The most straightforward way to reduce the entries in $\boldsymbol{r}_k$ is to use the IGT $\boldsymbol{Z}_{ik} = \boldsymbol{I} - \zeta_{ik}\boldsymbol{e}_i\boldsymbol{e}_k^{\mathsf{T}}$ with $\zeta_{ik} = \lfloor r_{ik}/r_{ii} \rceil$ to update $\boldsymbol{r}_k$ for $i = i_p, i_{p-1}, \ldots, i_1$, in the same way as the LLL reduction does (see Algorithm 2–1).

However, this straightforward method has a flaw. We know that after applying $\boldsymbol{Z}_{ik}$ in the above way, $|r_{ik}|$ is reduced. But, $|r_{jk}|$ might be increased for some $j < i$. In the LLL algorithm, this does not cause problem because for any $j < i$, $|r_{jk}|$ will be reduced later by the IGT $\boldsymbol{Z}_{jk}$. Here, if $\mathfrak{s}_j = 0$, r_{jk} will fail to be reduced. In this case, some off-diagonal entries in $\boldsymbol{R}$ may progressively grow too large and this can cause severe numerical problems, see [109]. Also, the goal of lattice reduction is to make the column vectors of $\boldsymbol{R}$ shorter. If we only reduce $|r_{ik}|$ where $\mathfrak{s}_i = 1$ and let other entries in $\boldsymbol{r}_k$ grow, the overall length of $\boldsymbol{r}_k$ might be greatly increased.

In our algorithm, instead of reducing some entries of $\boldsymbol{r}_k$, we try to reduce $\|\boldsymbol{r}_k\|_2$ using IGT $\boldsymbol{Z}_{ik}$ where $i < k$ and $\mathfrak{s}_i = 1$. To achieve this, we compute the size-reduction coefficients ζ_{ik} differently. The IGTs we can apply to reduce vector $\boldsymbol{r}_k$ are $\boldsymbol{Z}_{i_p,k}$, $\boldsymbol{Z}_{i_{p-1},k}$, $\ldots$, $\boldsymbol{Z}_{i_1,k}$. And those IGTs only affect entries in $\boldsymbol{r}_{1:i_p,k}$. Define $i_0 = 0$ and $\boldsymbol{r}_{(j),k} = \boldsymbol{r}_{i_{j-1}+1:i_j,k}$. Then $\boldsymbol{r}_{1:i_p,k}$ can be partitioned to

$$\boldsymbol{r}_{1:i_p,k} = \begin{bmatrix} \boldsymbol{r}_{(1),k} \\ \boldsymbol{r}_{(2),k} \\ \vdots \\ \boldsymbol{r}_{(p),k} \end{bmatrix} \begin{matrix} i_1 \\ i_2 - i_1 \\ \vdots \\ i_p - i_{p-1} \end{matrix} .$$

Generally, applying IGT $\boldsymbol{Z}_{i_j,k}$ will alter $\boldsymbol{r}_{(1),k}, \boldsymbol{r}_{(2),k}, \ldots, \boldsymbol{r}_{(j),k}$. In the size reduction, we first let $j = p$ and apply $\boldsymbol{Z}_{i_j,k}$ to $\boldsymbol{R}$. Then, $\boldsymbol{r}_{(j),k}$ becomes $\boldsymbol{r}_{(j),k} - \zeta_{i_j,k}\boldsymbol{r}_{(j),i_j}$. Instead of choosing $\zeta_{i_j,k} \in \mathbb{Z}$ such that $\left|r_{i_j,k} - \zeta_{i_j,k} r_{i_j,i_j}\right|$ is minimized as the LLL does, we choose $\zeta_{i_j,k}$ to be the solution to

$$\min_{\zeta \in \mathbb{Z}} \left\| \boldsymbol{r}_{(j),k} - \zeta \boldsymbol{r}_{(j),i_j} \right\|_2 . \tag{3.54}$$

It can be easily shown that the optimal $\zeta_{i_j,k}$ is

$$\zeta_{i_j,k} = \left\lfloor \frac{\boldsymbol{r}_{(j),i_j}^{\mathsf{T}} \boldsymbol{r}_{(j),k}}{\boldsymbol{r}_{(j),i_j}^{\mathsf{T}} \boldsymbol{r}_{(j),i_j}} \right\rceil . \tag{3.55}$$

After $\zeta_{i_j,k}$ is computed, we construct $\boldsymbol{Z}_{i_p,k} = \boldsymbol{I} - \zeta_{i_p,k} \boldsymbol{e}_{i_p} \boldsymbol{e}_k^{\mathsf{T}}$ and update $\boldsymbol{r}_k$ by applying $\boldsymbol{Z}_{i_p,k}$ to $\boldsymbol{R}$. Other size-reduction coefficients $\zeta_{i_j,k}$ for $j = p-1, p-2, \ldots, 1$ are computed using the same method. We would like to point out that solving (3.54) is one step of the Gaussian algorithm for finding a reduced base for a 2-dimensional lattice, see, e.g., [71, section 1.2].

The regular size reduction reduces one element by an IGT, while the above size reduction reduces a vector by an IGT. So the latter can be regarded as an extension of the former. For this reason, we call the latter the extended size reduction. Applying the sequence of extended size reductions to reduce a column as shown above is also a natural extension of applying a sequence of regular size reductions to reduce a column as the LLL reduction. To see this, write $\hat{\boldsymbol{R}} = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \ldots & \boldsymbol{r}_{k-1} \end{bmatrix}$ and let $\boldsymbol{\zeta} = \begin{bmatrix} \zeta_1 & \zeta_2 & \ldots & \zeta_{k-1} \end{bmatrix}^{\mathsf{T}} \in \mathbb{Z}^{k-1}$ be the integer vector consisting of size-reduction coefficients. In the LLL reduction process for column k, $\boldsymbol{r}_k$ is transformed to $\boldsymbol{r}_k - \hat{\boldsymbol{R}}\boldsymbol{\zeta}$. To make the length of $\boldsymbol{r}_k$ small after the reduction, it is easy to verify that $\boldsymbol{\zeta}$ in the LLL reduction is chosen to be the Babai point, see (2.16), of the following OILS problem:

$$\min_{\boldsymbol{\zeta} \in \mathbb{Z}^{k-1}} \left\| \boldsymbol{r}_k - \hat{\boldsymbol{R}}\boldsymbol{\zeta} \right\|_2^2. \tag{3.56}$$

In the restricted lattice reduction, we can only use columns $\boldsymbol{r}_{i_1}, \boldsymbol{r}_{i_2}, \ldots, \boldsymbol{r}_{i_p}$ to reduce $\boldsymbol{r}_k$. Redefine $\hat{\boldsymbol{R}} = \begin{bmatrix} \boldsymbol{r}_{i_1} & \boldsymbol{r}_{i_2} & \ldots & \boldsymbol{r}_{i_p} \end{bmatrix}$, and $\boldsymbol{\zeta} = \begin{bmatrix} \zeta_{i_1} & \zeta_{i_2} & \ldots & \zeta_{i_p} \end{bmatrix}^{\mathsf{T}}$. Problem (3.56)

in this situation becomes

$$\min_{\boldsymbol{\zeta}\in\mathbb{Z}^p} \left\|\boldsymbol{r}_k - \hat{\boldsymbol{R}}\boldsymbol{\zeta}\right\|_2^2. \tag{3.57}$$

Note that $\hat{\boldsymbol{R}}$ in (3.57) may not be upper triangular. We can use the QR factorization to transform (3.57) to a triangular form as (3.56) so that the Babai point of (3.57) can still be computed using the formula (2.16) given before. It can be shown that, the size-reduction coefficients computed using (3.55) form the exact Babai point of (3.57).

3.5.2 Backward size reduction

In all size reductions we have seen so far, we alway use column $\boldsymbol{r}_i$ to reduce r_{ik} in column $\boldsymbol{r}_k$ where $k > i$. In this subsection, we propose the backward size reduction which uses column $k+1$ to reduce r_{kk} in column k, which will be used later.

A backward size reduction matrix has the form $\boldsymbol{Z}_{k+1,k} = \boldsymbol{I} - \zeta_{k+1,k}\boldsymbol{e}_{k+1}\boldsymbol{e}_k^{\mathsf{T}}$ where $\zeta_{k+1,k} \in \mathbb{Z}$. Unlike the regular size reductions we have seen before, when applying $\boldsymbol{Z}_{k+1,k}$ to an upper triangular matrix, the resulting matrix is no longer upper triangular. A Givens rotation $\boldsymbol{G}_{k,k+1}$ is needed to bring the matrix back to upper triangular. We can describe the process of applying a backward size reduction using the following equation:

$$\bar{\boldsymbol{R}} = \boldsymbol{G}_{k,k+1}\boldsymbol{R}\boldsymbol{Z}_{k+1,k}.$$

It is easy to see that

$$\bar{r}_{kk}^2 = \left\| \begin{bmatrix} r_{kk} \\ 0 \end{bmatrix} - \zeta_{k+1,k} \begin{bmatrix} r_{k,k+1} \\ r_{k+1,k+1} \end{bmatrix} \right\|_2^2.$$

To minimize $\bar{r}_{k,k}^2$, we choose $\zeta_{k+1,k}$ such that:

$$\zeta_{k+1,k} = \underset{\zeta \in \mathbb{Z}}{\operatorname{argmin}} \left\| \begin{bmatrix} r_{kk} \\ 0 \end{bmatrix} - \zeta \begin{bmatrix} r_{k,k+1} \\ r_{k+1,k+1} \end{bmatrix} \right\|_2^2. \tag{3.58}$$

Note that the right hand side of (3.58) is actually (3.54) with $\boldsymbol{r}_{(j),k}$ replaced by $\begin{bmatrix} r_{kk} & 0 \end{bmatrix}^{\mathsf{T}}$ and $\boldsymbol{r}_{(j),i_j}$ by $\begin{bmatrix} r_{k,k+1} & r_{k+1,k+1} \end{bmatrix}^{\mathsf{T}}$. Thus, from (3.55), we have

$$\zeta_{k+1,k} = \left\lfloor \frac{r_{kk} r_{k,k+1}}{r_{k,k+1}^2 + r_{k+1,k+1}^2} \right\rceil. \tag{3.59}$$

It can be verified that the backward reduction is actually equivalent to first interchanging columns k and $k+1$ of $\boldsymbol{R}$, then applying regular size reduction $\boldsymbol{Z}_{k,k+1}$ and interchanging columns k and $k+1$ back. In the following example, we demonstrate how the backward size reduction is applied to a po-box constrained lattice.

Example 3–3. *Assume for $\Lambda_{\mathcal{P}}(\boldsymbol{R})$ we have*

$$\boldsymbol{R} = \begin{bmatrix} 5.0 & 4.0 \\ 0 & 3.1 \end{bmatrix}, \quad \mathfrak{s} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Since $\mathfrak{s}_1 = 0$, we cannot use $\boldsymbol{r}_1$ to reduce $\boldsymbol{r}_2$. It is easy to see that the Lovász condition is satisfied for $\boldsymbol{R}$, thus no column permutation will happen if the LLL permutation strategy is used. However, from (3.59)*, we have $\zeta_{2,1} = 1$. This means $\boldsymbol{R}$ can be reduced by the backward size reduction. After the backward size reduction, $\boldsymbol{R}$ becomes*

$$\bar{\boldsymbol{R}} = \boldsymbol{G}_{1,2} \begin{bmatrix} 5.0 & 4.0 \\ 0 & 3.1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} = \boldsymbol{G}_{1,2} \begin{bmatrix} 1.0 & 4.0 \\ -3.1 & 3.1 \end{bmatrix} = \begin{bmatrix} 3.2573 & -1.7223 \\ 0 & 4.7585 \end{bmatrix}.$$

Next, we make a remark on the backward size reduction.

Proposition 3–1. If an upper triangular matrix $\boldsymbol{R}$ is δ-LLL reduced with $\delta = 1$, then $\zeta_{k+1,k} = 0$ in (3.59) for all $k = 1, 2, \ldots, n-1$.

Proof. Since $\boldsymbol{R}$ is δ-LLL reduced, we have (see (2.4))

$$|r_{k,k+1}| \leq r_{kk}/2, \tag{3.60a}$$

$$\delta r_{kk}^2 \leq r_{k,k+1}^2 + r_{k+1,k+1}^2, \tag{3.60b}$$

for all $k = 1, \ldots, n-1$. Then it follows from (3.60) that

$$|\zeta_{k+1,k}| = \left\lfloor \frac{|r_{kk} r_{k,k+1}|}{r_{k,k+1}^2 + r_{k+1,k+1}^2} \right\rceil \leq \left\lfloor \frac{r_{kk}^2}{2\delta r_{kk}^2} \right\rceil = \left\lfloor \frac{1}{2\delta} \right\rceil .$$

When $\delta = 1$, we have $|\zeta_{k+1,k}| \leq \lfloor 0.5 \rceil = 0$ (we defined $\lfloor 0.5 \rceil = 0$, see Section 1.7). $\square$

From Proposition 3–1 we can see that the backward size reduction $\boldsymbol{Z}_{k+1,k}$ is not very useful in the LLL reduction (Algorithm 2–1) where δ is usually set to 1 or very close to 1. However, in the restricted lattice reduction, the size-reduction condition (3.60a) cannot always be satisfied, e.g., when $\mathfrak{s}_{k+1} = 1$ and $\mathfrak{s}_k = 0$, backward size reduction $\boldsymbol{Z}_{k+1,k}$ is helpful to reduce r_{kk}.

3.5.3 The RLLL algorithm

Given a po-box constrained lattice $\Lambda_{\mathcal{P}}(\boldsymbol{R}) = \{\boldsymbol{R}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{Z}^n,\ l_i \leq x_i \leq u_i,\ \mathfrak{s}_i = 0\}$, the RLLL algorithm reduces the lattice basis $\boldsymbol{R}$ following the framework of the LLL algorithm with two main differences. First, in RLLL, IGT $\boldsymbol{Z}_{ij}$ is allowed only if $\mathfrak{s}_i = 1$, i.e., the i-th interval constraint is inactive, see Section 3.2.2. Second, when it is possible, RLLL performs backward size reductions on columns of $\boldsymbol{R}$ where neither regular size reductions nor the LLL column permutations can be applied.

Specifically, the RLLL algorithm checks and updates the upper triangular matrix $\boldsymbol{R}$ column by column. Assume the current column it works on is $\boldsymbol{r}_k$. The RLLL algorithm first checks $\mathfrak{s}_{k-1}$. If $\mathfrak{s}_{k-1} = 1$, then it reduces $r_{k-1,k}$ using an IGT $\boldsymbol{Z}_{k-1,k}$. Otherwise, it skips this size reduction step and checks the Lovász condition (2.4b). If the Lovász condition does not hold, then it interchanges columns $k-1$ and k of $\boldsymbol{R}$ and entries $k-1$ and k of $\boldsymbol{l}$, $\boldsymbol{u}$ and $\mathfrak{s}$. However, if the Lovász condition holds, then it checks if $\mathfrak{s}_{k-1} = 0$, $\mathfrak{s}_k = 1$ and the backward size-reduction coefficient $\zeta_{k,k-1} \neq 0$. If these criteria are satisfied, then it reduces $r_{k-1,k-1}$ using the backward IGT $\boldsymbol{Z}_{k,k-1}$.

When either the column permutation or the backward size reduction occurs in the above step, column $\boldsymbol{r}_{k-1}$ is modified. In this case, the RLLL algorithm moves backward to column $k-1$ of $\boldsymbol{R}$ (if $k > 2$). Otherwise, it applies the extended size reductions to reduce $\|\boldsymbol{r}_k\|_2^2$, see Section 3.5.1, and then moves forward to the column $k+1$. After it moves to a new column, it updates k accordingly. Like the LLL algorithm, the above process starts from $k = 2$ and ends once $k > n$. The pseudocode of the overall RLLL algorithm is given in Algorithm 3–7. The inputs of the algorithm are $\boldsymbol{R}$, $\boldsymbol{l}$, $\boldsymbol{u}$ and $\mathfrak{s}$ that define $\Lambda_{\mathcal{P}}(\boldsymbol{R})$.

Here we make some remarks. Algorithm 3–7 behaves exactly the same as the LLL algorithm if all constraints are inactive, i.e., $\mathfrak{s} = \mathbf{1}_n$. On the other end, if there is no inactive constraint, i.e., $\mathfrak{s} = \mathbf{0}_n$, it becomes a column permutation algorithm that follows the LLL permutation strategy, see [25]. Also, like ELLL [64] and PLLL [109] which reduces the cost of LLL, the RLLL reduction could be modified to reduce the cost too. We leave this topic to future research.

Algorithm 3–7 The restricted LLL algorithm

function $[\boldsymbol{R}, \boldsymbol{l}, \boldsymbol{u}, \mathfrak{s}, \boldsymbol{Z}] = \text{RLLL}(\boldsymbol{R}, \boldsymbol{l}, \boldsymbol{u}, \mathfrak{s})$

let $\boldsymbol{Z} = \boldsymbol{I}_n$, $k = 2$;
while $k \leq n$ **do**
  if $\mathfrak{s}_{k-1} = 1$ **then**
    reduce $r_{k-1,k}$: $\boldsymbol{R} = \boldsymbol{R}\boldsymbol{Z}_{k-1,k}$; ▷ regular size reduction
    update $\boldsymbol{Z}$: $\boldsymbol{Z} = \boldsymbol{Z}\boldsymbol{Z}_{k-1,k}$;
  end if
  if $\delta r_{k-1,k-1}^2 > r_{k-1,k}^2 + r_{k,k}^2$ **then**
    do column permutation: $\boldsymbol{R} = \boldsymbol{G}_{k-1,k}\boldsymbol{R}\boldsymbol{P}_{k-1,k}$;
    update $\boldsymbol{Z} = \boldsymbol{Z}\boldsymbol{P}_{k-1,k}$;
    update $\boldsymbol{l} = \boldsymbol{P}_{k-1,k}\boldsymbol{l}$, $\boldsymbol{u} = \boldsymbol{P}_{k-1,k}\boldsymbol{u}$ and $\mathfrak{s} = \boldsymbol{P}_{k-1,k}\mathfrak{s}$;
    $k = \max(k-1, 2)$;
  else
    if $\mathfrak{s}_{k-1} = 0$ and $\mathfrak{s}_k = 1$ **then**
      compute $\zeta_{k,k-1}$ using (3.59); ▷ backward size reduction
      if $\zeta_{k,k-1} \neq 0$ **then**
        apply backward size reduction: $\boldsymbol{R} = \boldsymbol{G}_{k,k-1}\boldsymbol{R}\boldsymbol{Z}_{k,k-1}$;
        update $\boldsymbol{Z} = \boldsymbol{Z}\boldsymbol{Z}_{k,k-1}$;
        $k = \max(k-1, 2)$;
        continue
      end if
    end if
    apply the extended size reduction to $\boldsymbol{r}_k$;
    $k = k + 1$;
  end if
end while

3.5.4 RLLL based reductions for BILS problems

Now we put all relevant methods together to propose three RLLL based reduction algorithms to transform the original BILS problem (3.1) to the reduced POBILS problem (3.31).

Given the BILS problem (3.1), the three reduction algorithms, named RLLL-O, RLLL-A, and RLLL-R first apply the QR factorization on $\boldsymbol{A}$ to get the initial $\boldsymbol{R}$ and

$\bar{\boldsymbol{y}}$ in (3.31) with initial $\bar{\boldsymbol{l}} = \boldsymbol{l}$ and $\bar{\boldsymbol{u}} = \boldsymbol{u}$, i.e., by letting $\boldsymbol{Z} = \boldsymbol{I}$ in (3.29). Then they compute an initial search radius ρ using different methods:

RLLL-O: using the OILS relaxation based method, see Section 3.4.1;

RLLL-A: using the AIP reduction and the Babai point, see Section 3.4.2;

RLLL-R: using the real relaxation based method, see Section 3.4.3.

After that they apply ISF-2 (see Algorithm 3–5) to find an inactive vector $\mathfrak{s}$ of the problem. Based on $\mathfrak{s}$, define $\mathcal{P} = \{\boldsymbol{z} \mid \boldsymbol{z} \in \mathbb{Z}^n;\ \bar{l}_i \leq z_i \leq \bar{u}_i \text{ where } \mathfrak{s}_i = 0\}$. The original BILS problem is then transformed to the following POBILS problem:

$$\min_{\boldsymbol{z}\in\mathcal{P}} \|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2 .$$

The RLLL algorithm is then applied to perform the restricted lattice reduction on the upper triangular $\boldsymbol{R}$ obtained above.

To further improve the efficiency of sphere decoding, the AIP reduction (see Algorithm 3–1) is used in these reduction algorithms to reorder the columns of $\boldsymbol{R}$ after the RLLL reduction. Although the RLLL algorithm applies lattice reduction to $\boldsymbol{R}$, it determines the column order of $\boldsymbol{R}$ based on the diagonal entries of $\boldsymbol{R}$, unlike the AIP reduction which also uses the information in $\mathcal{B}$ and $\bar{\boldsymbol{y}}$. We use two extreme cases to explain why combining RLLL with AIP is beneficial. In one extreme case where we have a full inactive set, i.e., $\mathfrak{s} = \mathbf{1}_n$, the POBILS problem becomes an OILS problem and the RLLL reduction becomes the LLL reduction. It is observed in [19, chapter 3] that applying the AIP reduction after the LLL reduction is helpful in improving the efficiency of sphere decoding for solving the OILS problems. In the opposite extreme case where the inactive set is empty, i.e., $\mathfrak{s} = \mathbf{0}_n$, no size reduction

can be performed in the RLLL reduction. In this case, the RLLL reduction merely uses the LLL permute strategy to reorder the columns of $\boldsymbol{R}$ according to its diagonal entries. And we know that in general this column ordering is not as effective as the V-BLAST ordering (see [25]), which is not as effective as the AIP ordering (see [20, 24]). In the implementation, if $\boldsymbol{\mathfrak{s}} = \boldsymbol{0}_n$, we actually skip the RLLL reduction and perform only the AIP reduction.

Note that the RLLL-O reduction has a chance to find the solution $\boldsymbol{x}^*$ of the original BILS problem in the computation of ρ. If this happens, it returns the solution $\boldsymbol{x}^*$, and the rest of the reduction and sphere decoding are no longer needed. The pseudocode of RLLL-O, RLLL-A and RLLL-R is given in Algorithm 3–8, Algorithm 3–9 and Algorithm 3–10, respectively.

Algorithm 3–8 Reduction based on RLLL and the OILS relaxation solution

function $[\boldsymbol{x}^*, \boldsymbol{R}, \bar{\boldsymbol{y}}, \boldsymbol{Z}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \boldsymbol{\mathfrak{s}}, \rho] = \text{RLLL-O}(\boldsymbol{A}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$

apply the PLLL reduction: $\boldsymbol{R} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{A}\boldsymbol{Z}$, $\bar{\boldsymbol{y}} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{y}$;
compute $\boldsymbol{z}^* = \text{argmin}_{\boldsymbol{z}\in\mathbb{Z}^n} \|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}\|_2^2$;
$\bar{\boldsymbol{x}}^* = \boldsymbol{Z}\boldsymbol{z}^*$;
if $\boldsymbol{l} \le \bar{\boldsymbol{x}}^* \le \boldsymbol{u}$ **then**
    $\boldsymbol{x}^* = \bar{\boldsymbol{x}}^*$; ▷ x^* is the solution
else
    $\bar{\boldsymbol{x}}^* = \text{median}\,(\bar{\boldsymbol{x}}^*, \boldsymbol{l}, \boldsymbol{u})$;
    apply QR factorization: $\boldsymbol{R} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{A}$, $\bar{\boldsymbol{y}} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{y}$;
    $\rho = \|\bar{\boldsymbol{y}} - \boldsymbol{R}\bar{\boldsymbol{x}}^*\|_2$;
    $\boldsymbol{\mathfrak{s}} = \text{ISF-2}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \boldsymbol{l}, \boldsymbol{u}, \rho)$;
    if $\boldsymbol{\mathfrak{s}} \neq \boldsymbol{0}$ **then**
        $[\boldsymbol{R}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \boldsymbol{\mathfrak{s}}, \boldsymbol{Z}] = \text{RLLL}(\boldsymbol{R}, \boldsymbol{l}, \boldsymbol{u}, \boldsymbol{\mathfrak{s}})$
        update $\bar{\boldsymbol{y}}$ accordingly;
    end if
    $[\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \boldsymbol{P}] = \text{AIP}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}})$;
    update $\boldsymbol{Z} = \boldsymbol{Z}\boldsymbol{P}$ and $\boldsymbol{\mathfrak{s}} = \boldsymbol{P}^{\mathsf{T}}\boldsymbol{\mathfrak{s}}$;
end if

Algorithm 3–9 Reduction based on RLLL and the Babai point

function $[\boldsymbol{R}, \bar{\boldsymbol{y}}, \boldsymbol{Z}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \mathfrak{s}, \rho] = \text{RLLL-A}(\boldsymbol{A}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$

$[\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \boldsymbol{P}_0, \boldsymbol{z}^{\text{B}}] = \text{AIP}(\boldsymbol{A}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$; ▷ see Algorithm 3–1
$\rho = \|\bar{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{z}^{\text{B}}\|_2$;
$\mathfrak{s} = \text{ISF-2}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \rho)$;
if $\mathfrak{s} \neq \boldsymbol{0}$ **then**
    $[\boldsymbol{R}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \mathfrak{s}, \boldsymbol{Z}] = \text{RLLL}(\boldsymbol{R}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \mathfrak{s})$;
    update $\bar{\boldsymbol{y}}$ accordingly;
    $[\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \boldsymbol{P}_1] = \text{AIP}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}})$;
    update $\boldsymbol{Z} = \boldsymbol{P}_0\boldsymbol{Z}\boldsymbol{P}_1$ and $\mathfrak{s} = \boldsymbol{P}_1^{\mathsf{T}}\mathfrak{s}$;
else
    $\boldsymbol{Z} = \boldsymbol{P}_0$;
end if

Algorithm 3–10 Reduction based on RLLL and the real relaxation solutions

function $[\boldsymbol{R}, \bar{\boldsymbol{y}}, \boldsymbol{Z}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \mathfrak{s}, \rho] = \text{RLLL-R}(\boldsymbol{A}, \boldsymbol{y}, \boldsymbol{l}, \boldsymbol{u})$

apply QR factorization: $\boldsymbol{R} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{A}$, $\bar{\boldsymbol{y}} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{y}$;
initialize $\bar{\boldsymbol{l}} = \boldsymbol{l}$, $\bar{\boldsymbol{u}} = \boldsymbol{u}$ and $\boldsymbol{Z} = \boldsymbol{I}_n$;
$[\hat{\boldsymbol{z}}, \rho] = \text{R-approximate}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}})$; ▷ see Algorithm 3–6
$\mathfrak{s} = \text{ISF-2}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \rho)$;
if $\mathfrak{s} \neq \boldsymbol{0}$ **then**
    $[\boldsymbol{R}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \mathfrak{s}, \boldsymbol{Z}] = \text{RLLL}(\boldsymbol{R}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \mathfrak{s})$;
    update $\bar{\boldsymbol{y}}$ accordingly;
end if
$[\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}}, \boldsymbol{P}] = \text{AIP}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}})$;
update $\boldsymbol{Z} = \boldsymbol{Z}\boldsymbol{P}$ and $\mathfrak{s} = \boldsymbol{P}^{\mathsf{T}}\mathfrak{s}$;

3.6 Numerical experiments

In this section, we compare the performance of the RLLL based algorithms proposed in Section 3.5.4 and the AIP reduction described in Section 3.1.2 through numerical experiments. All of the experiments were performed in MATLAB 7.14 (R2012a) on a PC with 3.30GHz quadcore CPU and 4GB memory running Ubuntu 12.04 (Linux 3.2.0).

We designed two experiments to compare the AIP reduction and the RLLL based reductions. In the first experiment, we focus on general BILS problems and compare the effectiveness of the reduction algorithms on Schnorr-Euchner's sphere decoding algorithm. In MIMO applications, one is often interested in finding approximate solutions to BILS problems. In the second experiment, we focus on MIMO model and show how the reduction algorithms affect the symbol error rate (SER) of the approximate solutions.

3.6.1 Effects of reductions on the efficiency of sphere decoding

In this numerical experiment, we generate instances of the BILS problem (3.1) according to the following linear model:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{v}, \quad \mathbf{0}_n \leq \boldsymbol{x} \leq (2^k - 1)\mathbf{1}_n, \tag{3.61}$$

where $\boldsymbol{v} \in \mathbb{R}^{n\times 1}$ is the noise vector following $\mathcal{N}(\mathbf{0}_n, \sigma^2 \boldsymbol{I}_n)$, $k \in \mathbb{Z}$ is the scale factor of the box constraint, $\boldsymbol{x} = (x_i)_n$ where $x_i \in \mathbb{Z}$ follows the i.i.d. uniform distribution over the range $[0, 2^k - 1]$, and $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is generated differently in the tests as follows:

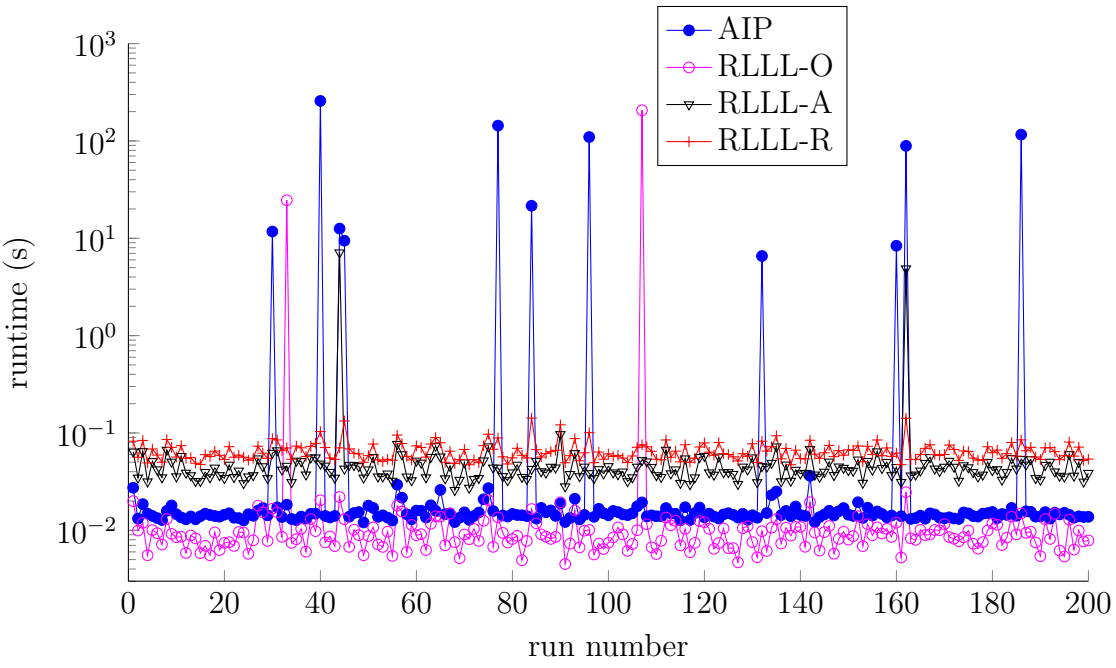
- Case 1: $\boldsymbol{A} = \text{randn}(n, n)$, where $\text{randn}(n, n)$ is a MATLAB built-in function to generate a random $n \times n$ matrix with independent, identically distributed (i.i.d.) entries following normal distribution $\mathcal{N}(0, 1)$.
- Case 2 (moderately ill conditioned): we first generate $\boldsymbol{A}_0 = \text{randn}(n, n)$ and compute the singular value decomposition $\boldsymbol{A}_0 = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\mathsf{T}$. Then, we construct a diagonal matrix $\bar{\boldsymbol{D}}$ with $\bar{d}_{11} = 15$, $\bar{d}_{nn} = 0.005$ and $\bar{d}_{ii} = d_{ii}$ for $i = 2, 3, \ldots, n-1$. Finally, we form $\boldsymbol{A} = \boldsymbol{U}\bar{\boldsymbol{D}}\boldsymbol{V}^\mathsf{T}$. The condition number of $\boldsymbol{A}$ is at least 3000.

The same two cases have been used in Section 2.4.

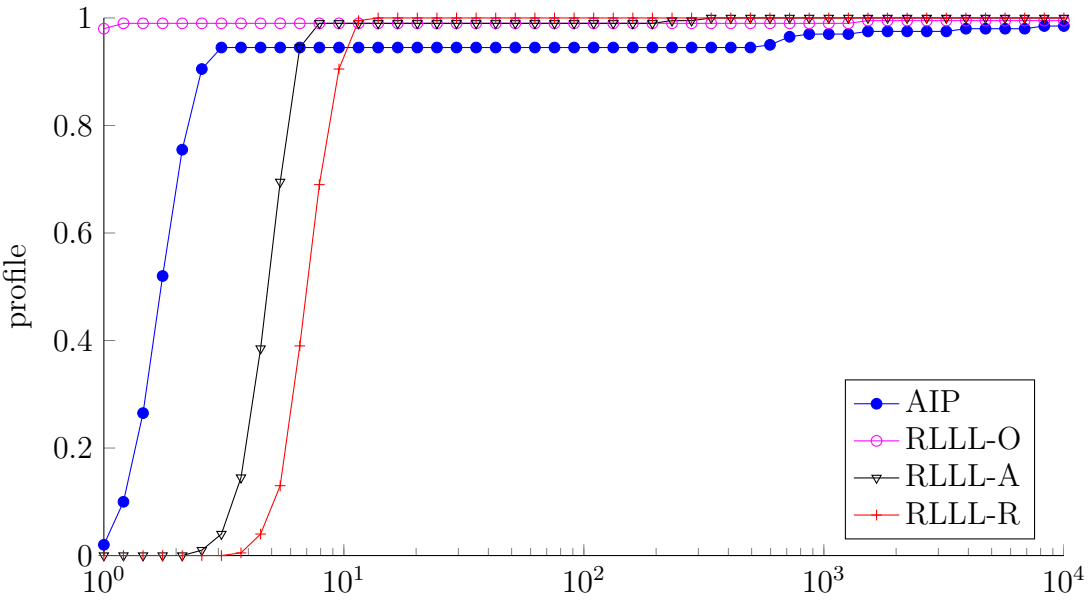
We first compare the effectiveness of the four reduction algorithms, AIP, RLLL-O, RLLL-A and RLLL-R, in improving the efficiency of solving BILS problems. The test instances of BILS problem (3.1) are generated using model (3.61). For each of Case 1 and Case 2, we generate 200 instances independently. For each instance, we reduce it using the four reduction algorithms mentioned above and solve the reduced problems using the Schnorr-Euchner's sphere decoding algorithm for BILS, see Section 3.1.3. The per-instance runtime (including the computation time of reduction and sphere decoding) corresponding to each reduction algorithm and their performance profiles are shown in Figure 3–5 and Figure 3–6 for Case 1 and Case 2 respectively. Performance profiles provide an effective means to compare solver performance for several solvers at once, eliminating some of the bias certain comparisons have (see, e.g., [15, 33]). The runtime statistics is reported in Table 3–1.

Table 3–1: Runtime statistics of the 200 instances

	Case 1 (Figure 3–5a)			Case 2 (Figure 3–6a)		
Solver	Min (s)	Average (s)	Max (s)	Min (s)	Average (s)	Max (s)
AIP	0.012	3.946	257.650	0.011	6.410	164.810
RLLL-O	0.005	1.166	206.690	0.009	0.398	35.390
RLLL-A	0.026	0.103	7.121	0.031	0.971	86.710
RLLL-R	0.047	0.065	0.142	0.047	0.071	1.274

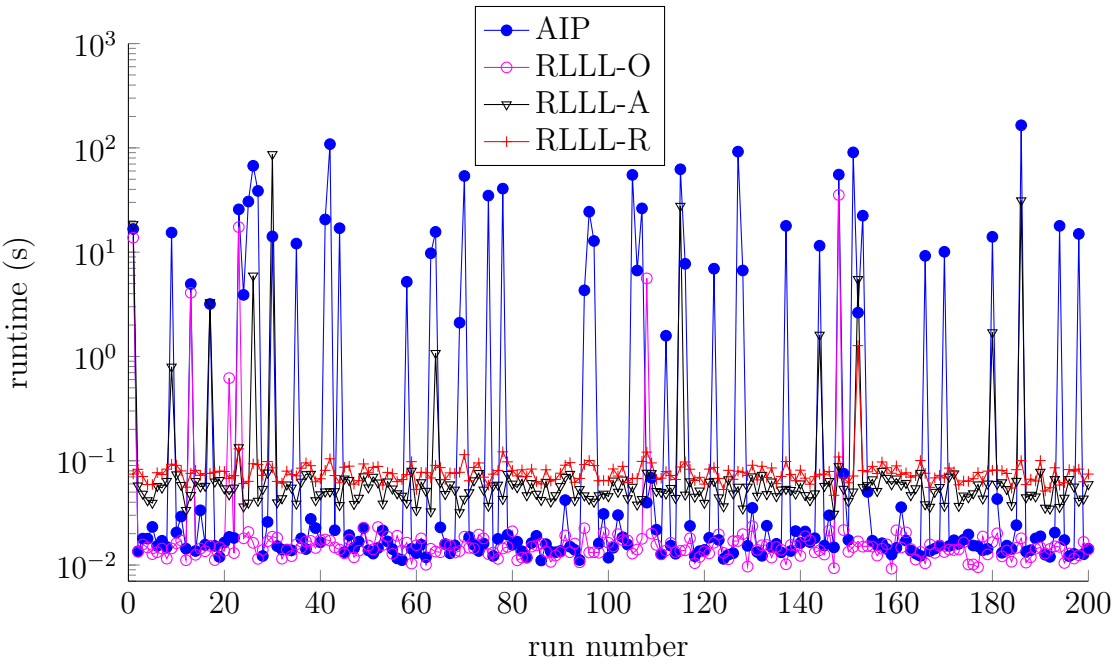


(a) Runtime of 200 instances

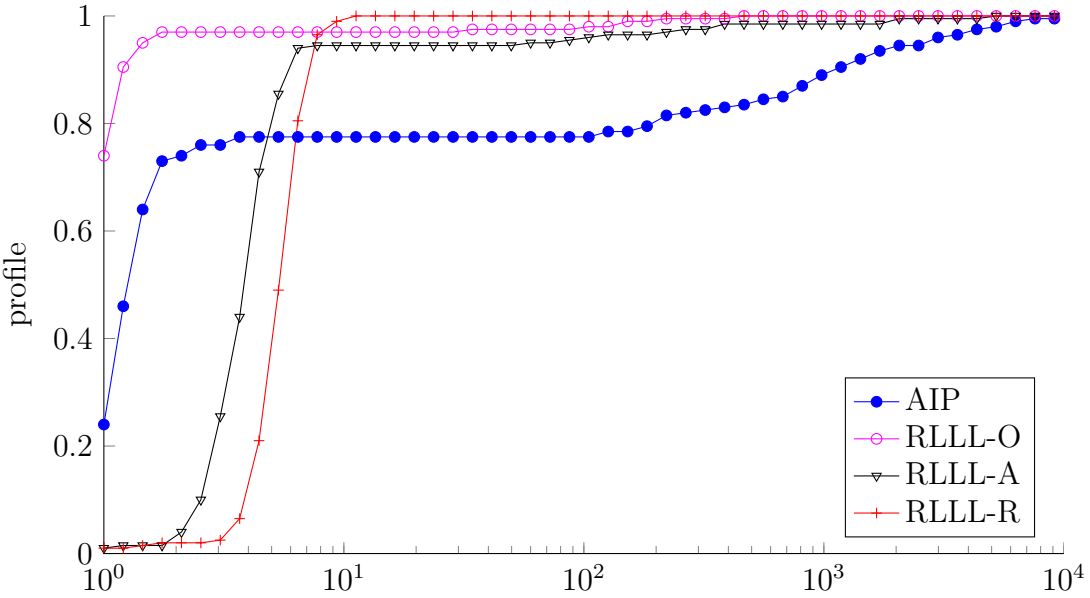


(b) Performance profile

Figure 3–5: Case 1: $\sigma = 0.35$, $k = 5$ and $n = 50$



(a) Runtime of 200 instances



(b) Performance profile

Figure 3–6: Case 2: $\sigma = 0.35$, $k = 5$ and $n = 45$

We can see that generally the RLLL-O reduction gives us the best performance, followed by the AIP reduction. Solving the BILS problems using RLLL-A and RLLL-R as the reduction algorithms are generally slower because the two reduction algorithms themselves are more time consuming. However, the runtime curves of the AIP reduction has many spikes, especially in Case 2 (see Figure 3–6a). In contrast, the runtime curves corresponding to the RLLL based reductions have significantly less spikes, especially for RLLL-R. If we look at the average runtime over the 200 instance of each cases in Table 3–1, we see that the RLLL-R reductions gives the best average performance.

To give a full comparison of the four reduction algorithms under different dimensions, noise levels and box sizes, we further tested the four reductions in three different scenarios. Let us consider the BILS problems of Case 1 first. In the first scenario, we take $\sigma = 0.35$, $k = 5$ (so that $0 \leq x_i \leq 31$) and $n = 40, 41, \ldots, 60$. For each dimension n, we independently generate 100 instances of BILS problem (3.1) using model (3.61). For each instance, we solve it using the four reduction algorithms and the Schnorr-Euchner's sphere decoding algorithm. The average computation time (including the reduction time and the sphere decoding time) corresponding to each reduction algorithm over the 100 instances is shown in Figure 3–7.

In Figure 3–7, we can see that for most n, using RLLL-O as reduction enables us to solve the BILS problems much faster than using the other three algorithms on average, but there are many spikes in the corresponding curve. Recall that RLLL-O solves an OILS problem to find the initial search radius ρ. We found that, these spikes on the curve of RLLL-O are mainly due to the OILS sphere decoding. The

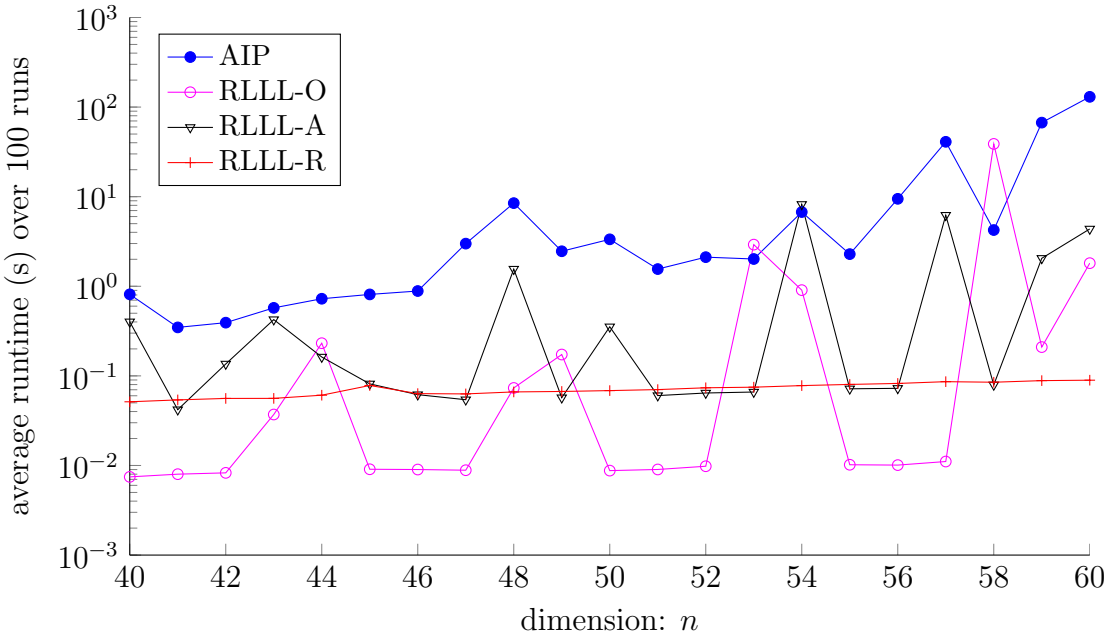


Figure 3–7: Case 1: $\sigma = 0.35$, $k = 5$ and $n = 40 : 60$

RLLL-A algorithm also has many spikes on its curve. Recall that the RLLL-A computes an initial search radius ρ using the AIP reduction and then applies the RLLL algorithm (Algorithm 3–7) based on this ρ. When this initial ρ is large and the ISF-2 algorithm cannot find enough number of inactive constraints, then the RLLL algorithm does not bring significant benefit. It can be observed that the curve of RLLL-A shares some spikes with the curve of AIP, e.g., at dimensions 48, 57 and 60. On the other hand, RLLL-R is more stable than the three other algorithms, even though sometimes it is outperformed by RLLL-O. Compared to the AIP reduction, the RLLL-R reduction lets us solve the BILS problems about 10 times to 1000 times faster (from dimension 40 to dimension 60).

In the second scenario, we take $n = 50$, $k = 5$ and let $\sigma = 0.05 : 0.05 : 0.7$. For each value of σ, we independently generate 100 instances of BILS problem (3.1) using model (3.61). Then, we solve the 100 instances using the four reduction algorithms and sphere decoding as in scenario 1. The average runtime is shown in Figure 3–8 for Case 1.

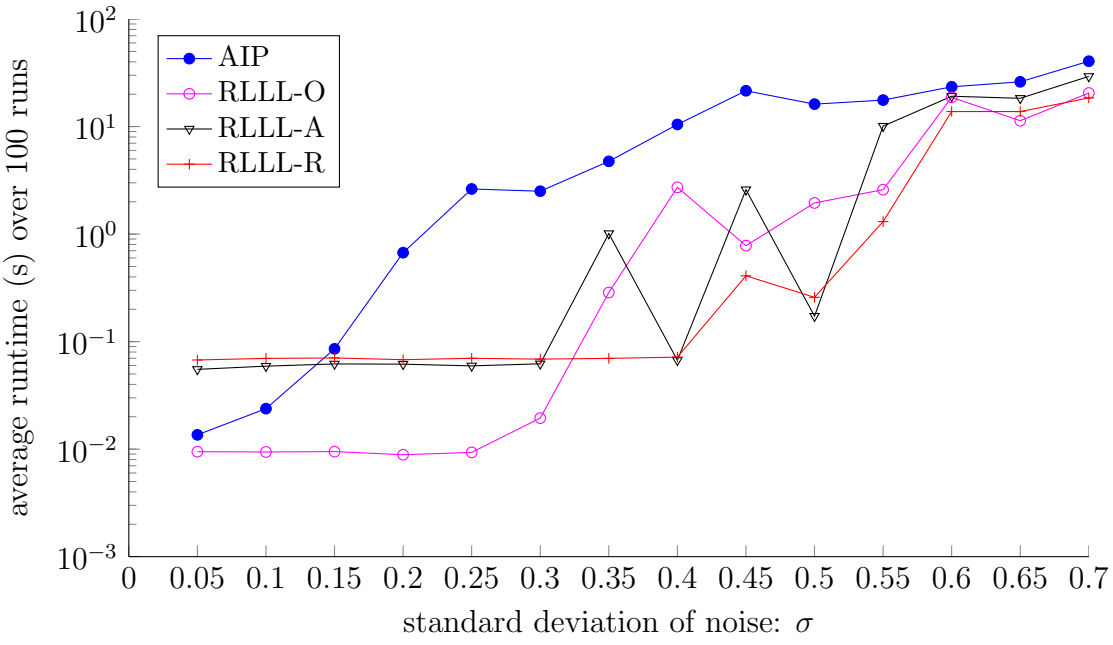


Figure 3–8: Case 1: $n = 50$, $k = 5$ and $\sigma = 0.05 : 0.05 : 0.7$

In Figure 3–8, we can see that the RLLL-O is the best reduction when the noise is small. This is because Schnorr-Euchner's sphere decoding algorithm combined with the PLLL reduction is very efficient in solving the OILS problems when the noise is small. And in this case, we often observe that the solution of the OILS relaxation is just the solution of the original BILS problem. Thus, we can save time to reduce and solve the BILS problem. However, when the noise is large, the cost for

solving the OILS relaxation increases rapidly. And the chance of the OILS solution being different from the BILS solution also increases. In this case, the RLLL-O based solver is less efficient than the RLLL-R based solver.

The efficiency of reduction and the efficiency of sphere decoding are usually two competing considerations in solving ILS problems. We can see that the RLLL-A reduction and RLLL-R reduction are outperformed by the RLLL-O reduction and the AIP reduction when the noise is small. The reason is that RLLL-R and RLLL-O tend to spend a lot more time in reduction, trying to get a higher efficiency for sphere decoding. However, when the noise is small, the sphere decoding process is usually not expensive. In this case, the gain in sphere decoding does not pay off the extra effort spent on reduction. However, when the noise increases, the cost of sphere decoding raises rapidly if the AIP reduction is used. The RLLL-R algorithm becomes the better choice when $\sigma \geq 0.35$. As the noise level keeps increasing, the initial search radius ρ inevitably increases, and the number of inactive constraints we could find decreases. In Figure 3–8, we can observe that the performance differences between the AIP reduction and the RLLL based reductions decrease when σ is big.

In the third scenario, we take $n = 50$, $\sigma = 0.35$ and let k take different values from 1 to 10 (so that each x_i takes values from 0 to 1023). For each value of k, we independently generate 100 instances of BILS problem (3.1) using model (3.61). Then, we compute the average computation time of the 100 instances using the four reduction algorithms and sphere decoding as in scenario 1 and scenario 2. The results are shown in Figure 3–9 for Case 1.

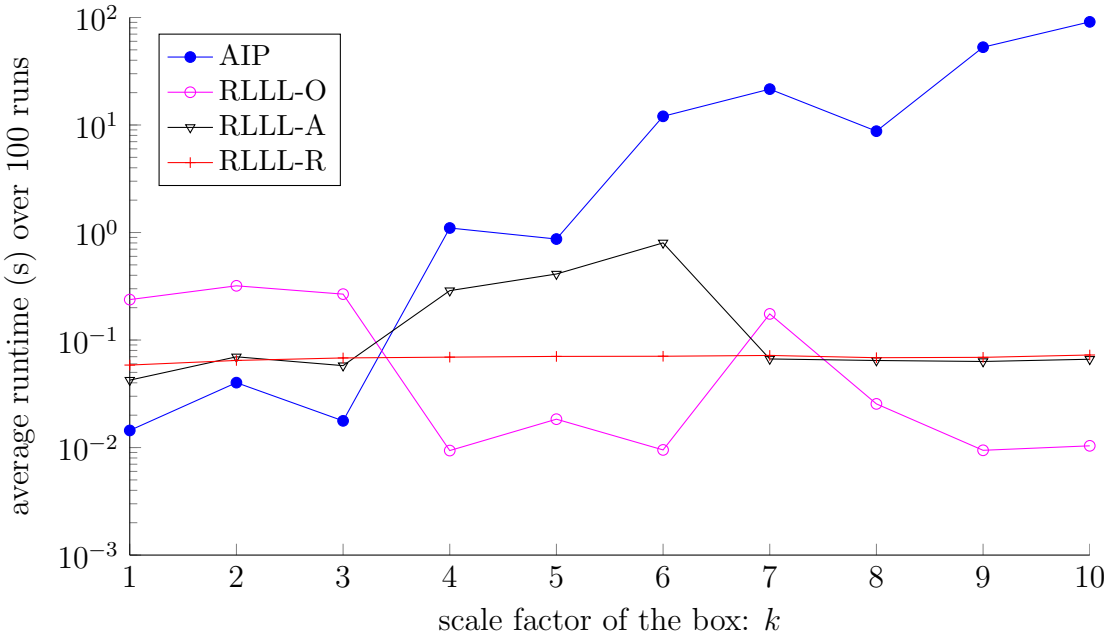


Figure 3–9: Case 1: $n = 50$, $\sigma = 0.35$ and $k = 1 : 10$

In Figure 3–9, we can observe that when the AIP reduction is used, the solving time increases when the size of the box increases. Recall that the search region of sphere decoding for a BILS problem is the intersection of the ellipsoid (3.10a) and the box (3.10b). When the size of the box is large, the volume of the intersection is expected to be large as well, leading to less efficient sphere decoding. However, when the box is large, we have a better chance to find more inactive constraints, which means more size reductions are allowed in the RLLL reduction. Affected by these two factors, the curve corresponding to the RLLL-A reduction in Figure 3–9 tends to increase when k changes form 1 to 6, and then it tends to decrease or remain unchanged when k changes from 6 to 10. For RLLL-O, when the size of the box is small, the OILS solution is less likely to be (or close to) the BILS solution. Thus,

when the box is very small, the RLLL-O reduction does not perform very well. We can see that RLLL-R, even though is not always the best one, generally performs well and is very stable in this scenario. It will be an interesting topic to study why RLLL-R is much more stable than the other reductions in the tests. We leave it to future research.

For Case 2, we also compare the runtime of solving the BILS problems (now with a more ill conditioned $\boldsymbol{A}$) using the four reductions and sphere decoding in the above three scenarios. The results are shown in Figure 3–10, 3–11 and 3–12, In Case 2, we noticed that some sphere decoders run very slowly for large n. For efficiency, we use slightly smaller dimensions n in the tests, i.e., $n = 35 : 55$ in scenario 1 (see Figure 3–10) and $n = 45$ in scenarios 2 and 3 (see Figure 3–11 and 3–12). We also limit the search tree size of sphere decoding to 10^8. If a sphere decoder does not finish within this limit, we terminate it and set the runtime to NaN, which will be shown as blank in the figures. All other setups are identical to the setups used in the scenarios for Case 1. From the figures, we can observe that all curves are more spiky compared with Case 1, indicating larger instance-to-instance variations. But, when we compare the different reduction algorithms in Case 2, we still see the similar patterns as we saw in Case 1.

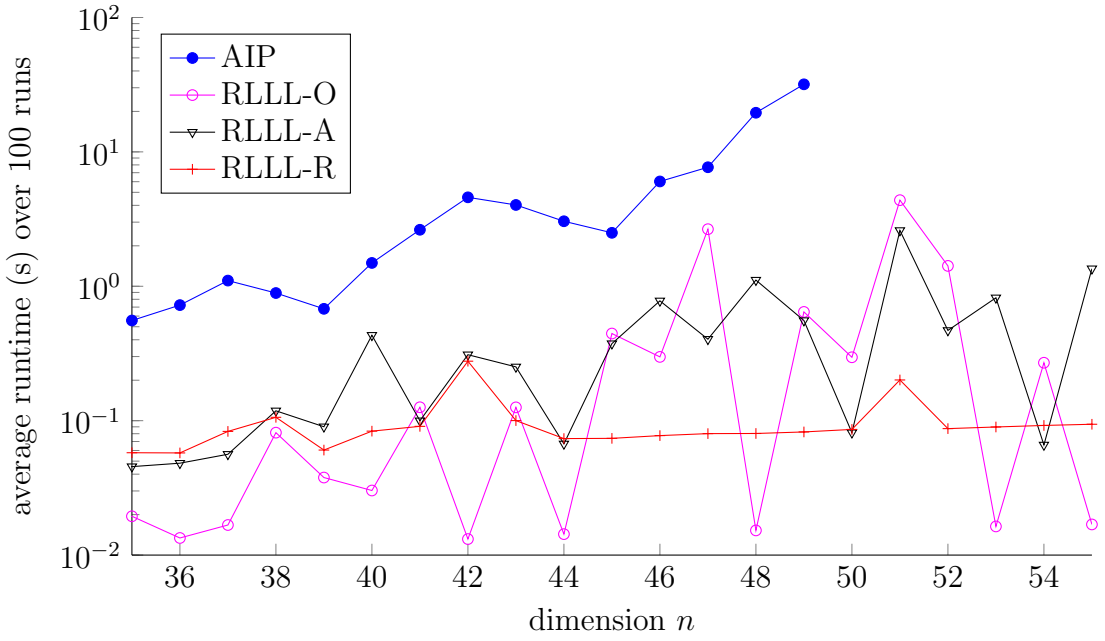


Figure 3–10: Case 2: $\sigma = 0.35$, $k = 5$ and $n = 35 : 55$

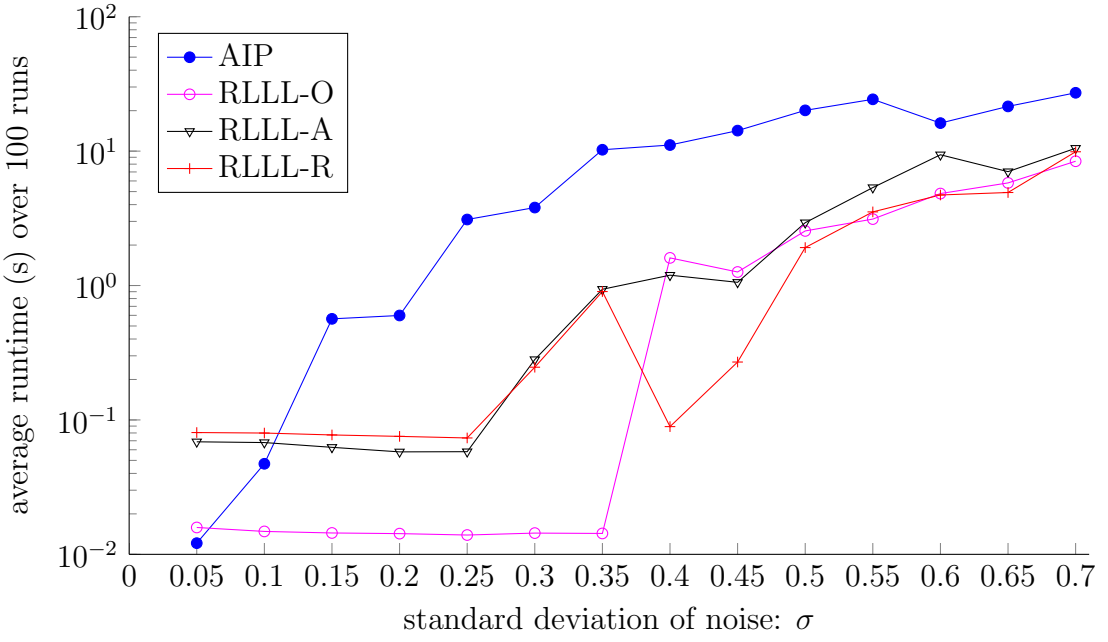


Figure 3–11: Case 2: $n = 45$, $k = 5$, and $\sigma = 0.05 : 0.05 : 0.7$

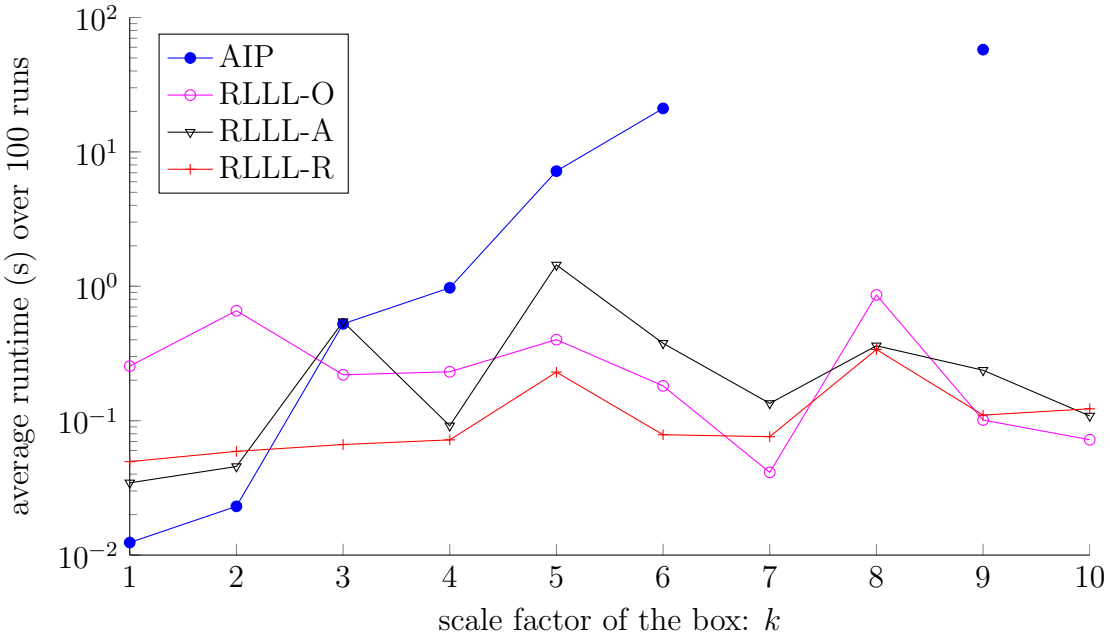


Figure 3–12: Case 2: $n = 45$, $\sigma = 0.35$ and $k = 1 : 10$

3.6.2 Effects of reductions on the quality of approximate solutions

Solving the BILS problem (3.1) exactly can be very time consuming. In applications like MIMO communication, approximate solutions to (3.1) are acceptable, but they are required to be computed very quickly. In the following numerical tests, we compare the performance of the approximation solutions obtained in the reduction algorithms introduced before. However, since the RLLL-O reduction algorithm involves solving an OILS problem exactly, which is very time consuming sometimes, we consider only the AIP, RLLL-A and RLLL-R reduction algorithms in the tests. For the AIP reduction, we use the box-constrained Babai point $\boldsymbol{z}^{\mathrm{B}}$ as the approximate solution. Recall that the RLLL-A reduction and the RLLL-R reduction compute an approximate solution $\hat{\boldsymbol{z}}$ to get the initial search radius ρ before the RLLL algorithm

is applied. To not waste the computation, for RLLL-A and RLLL-R, we compare the residual of $\hat{\boldsymbol{z}}$ and the residual of $\boldsymbol{z}^{\text{B}}$, the box-constrained Babai point of the RLLL-A/RLLL-R reduced problem, and choose the one that has a smaller residual as the approximate solution.

In the tests, we use the MIMO model to generate the BILS problems. Recall that in the MIMO application with m transmit and n receive antennas, we have the complex linear model

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{v}, \quad \boldsymbol{x} \in \left\{k_1 + k_2 i \mid k_1, k_2 = \pm 1, \pm 3, \ldots, \pm(2^k - 1)\right\}^n, \tag{3.62}$$

where $\boldsymbol{H} \in \mathbb{C}^{m\times n}$ is the channel matrix, $\boldsymbol{x}$ is an n-dimensional transmitted symbol vector, $\boldsymbol{y} \in \mathbb{C}^m$ is the received signal vector and $\boldsymbol{v} \in \mathbb{C}^m$ is a Gaussian noise vector with distribution $\mathcal{CN}(\boldsymbol{0}_m, \sigma^2 \boldsymbol{I}_m)$. Note that the entries of $\boldsymbol{x}$ in (3.62) correspond to the M-QAM (quadrature amplitude modulation) constellation where $M = 4^k$.

The correlation between different MIMO channel elements is usually modelled under the assumption that the correlation among the receive antennas is independent of the correlation among the transmit antennas (and vice versa), see, e.g., [94, 53]. Under this assumption, the MIMO channel matrix $\boldsymbol{H}$ for Rayleigh flat-fading like channels with signal correlation can be described by (e.g., [104, 40])

$$\boldsymbol{H} = \boldsymbol{R}_{\text{R}}^{1/2} \boldsymbol{H}_0 \boldsymbol{R}_{\text{T}}^{1/2},$$

where $\boldsymbol{H}_0 \in \mathbb{C}^{m\times n}$ is a random matrix with i.i.d. complex Gaussian zero-mean unit variance elements, i.e., $h_{ij} \sim \mathcal{CN}(0, 1)$, $\boldsymbol{R}_{\text{T}} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{R}_{\text{R}} \in \mathbb{R}^{n\times n}$ are the correlations observed on the transmitter and receiver side respectively.

In [104], a single coefficient spatial correlation model is proposed to generate $\boldsymbol{R}_{\mathrm{T}}$ and $\boldsymbol{R}_{\mathrm{R}}$ for simulation purposes. Specifically, the model sets

$$\boldsymbol{R}_{\mathrm{R}} = \begin{bmatrix} 1 & r_{\mathrm{R}} & r_{\mathrm{R}}^4 & \dots & r_{\mathrm{R}}^{(n-1)^2} \\ r_{\mathrm{R}} & 1 & r_{\mathrm{R}} & \ddots & \vdots \\ r_{\mathrm{R}}^4 & r_{\mathrm{R}} & 1 & \ddots & r_{\mathrm{R}}^4 \\ \vdots & \ddots & \ddots & \ddots & r_{\mathrm{R}} \\ r_{\mathrm{R}}^{(n-1)^2} & \dots & r_{\mathrm{R}}^4 & r_{\mathrm{R}} & 1 \end{bmatrix}, \quad \boldsymbol{R}_{\mathrm{T}} = \begin{bmatrix} 1 & r_{\mathrm{T}} & r_{\mathrm{T}}^4 & \dots & r_{\mathrm{T}}^{(n-1)^2} \\ r_{\mathrm{T}} & 1 & r_{\mathrm{T}} & \ddots & \vdots \\ r_{\mathrm{T}}^4 & r_{\mathrm{T}} & 1 & \ddots & r_{\mathrm{T}}^4 \\ \vdots & \ddots & \ddots & \ddots & r_{\mathrm{T}} \\ r_{\mathrm{T}}^{(n-1)^2} & \dots & r_{\mathrm{T}}^4 & r_{\mathrm{T}} & 1 \end{bmatrix},$$

where $r_{\mathrm{R}}, r_{\mathrm{T}} \in [0,1]$ reflect the level of correlation on the receiver side and the transmitter side. Matrix $\boldsymbol{H}$ changes from a full rank matrix (usually $\boldsymbol{H}_0$ has full column rank) to a rank 1 matrix as r_{R} and r_{T} increase from 0 to 1.

In our simulation, we transform the above complex model to a real model. We define

$$\tilde{\boldsymbol{v}} = \begin{bmatrix} \boldsymbol{v}^{\mathrm{R}} \\ \boldsymbol{v}^{\mathrm{I}} \end{bmatrix}, \quad \tilde{\boldsymbol{H}}_0 = \begin{bmatrix} \boldsymbol{H}_0^{\mathrm{R}} & -\boldsymbol{H}_0^{\mathrm{I}} \\ \boldsymbol{H}_0^{\mathrm{I}} & \boldsymbol{H}_0^{\mathrm{R}} \end{bmatrix}, \quad \tilde{\boldsymbol{H}} = \begin{bmatrix} \boldsymbol{R}_{\mathrm{R}}^{1/2} & \\ & \boldsymbol{R}_{\mathrm{R}}^{1/2} \end{bmatrix} \tilde{\boldsymbol{H}}_0 \begin{bmatrix} \boldsymbol{R}_{\mathrm{T}}^{1/2} & \\ & \boldsymbol{R}_{\mathrm{T}}^{1/2} \end{bmatrix}, \quad (3.63)$$

where the superscript R indicates the real part of a complex matrix or vector, and the superscript I indicates its imaginary part. It is easy to see that the entries of $\boldsymbol{H}_0^{\mathrm{R}}$ and $\boldsymbol{H}_0^{\mathrm{I}}$ follow the i.i.d. distribution $\mathcal{N}(0, 1/2)$ and the entries of $\bar{\boldsymbol{v}}$ follow the i.i.d. distribution $\mathcal{N}(0, \sigma^2/2)$. Let $\bar{\boldsymbol{x}} = (\tilde{\boldsymbol{x}} + (2^k - 1)\mathbf{1}_{2n})/2$, and $\bar{\boldsymbol{y}} = \tilde{\boldsymbol{y}} + (2^k - 1)\tilde{\boldsymbol{H}}\mathbf{1}_{2n}$. We can transform the complex model (3.62) to the following linear model

$$\bar{\boldsymbol{y}} = 2\tilde{\boldsymbol{H}}\bar{\boldsymbol{x}} + \tilde{\boldsymbol{v}}, \quad \mathbf{0}_{2n} \le \bar{\boldsymbol{x}} \le (2^k - 1)\mathbf{1}_{2n}. \quad (3.64)$$

In our simulation, we set $m = n$, i.e., $\tilde{\boldsymbol{H}}$ is a square matrix, and let $r_{\mathrm{R}} = r_{\mathrm{T}} = \gamma$ for simplicity. To generate an instance of the above linear model, we uniformly generate $\bar{x}_i$ in $[0, 2^k - 1]$ for $i = 1, 2, \dots, 2n$, generate $\boldsymbol{H}_0^{\mathrm{R}}$ and $\boldsymbol{H}_0^{\mathrm{I}}$ using $1/\sqrt{2} * \mathrm{randn}(2n, 2n)$,

and generate $\tilde{\boldsymbol{v}}$ using $\sigma/\sqrt{2} * \text{randn}(2n, 1)$. Then we compute $\tilde{\boldsymbol{H}}$ using (3.63) and $\bar{\boldsymbol{y}}$ using (3.64).

In Figure 3–13, Figure 3–14 and Figure 3–15, we show the average symbol error rates (SERs) of the approximate solutions obtained by using the above mentioned reduction algorithms against the signal-to-noise-ratios (SNRs) in different setups. Here the SER of an approximate solution $\hat{\boldsymbol{x}}$ is the defined by $d/(2n)$, where d is the number of different entries between $\hat{\boldsymbol{x}}$ and the transmitted vector $\bar{\boldsymbol{x}}$. For a M-QAM constellation, the SNR is defined by $\text{SNR} = 10\log_{10} 2n(M-1)/(3\sigma^2)$. Each point in the figures represents the average SER over 5000 runs.

In Figure 3–13, we take $k = 4$, i.e., 256-QAM, $m = n = 8$ and $\gamma = 0, 0.5, 0.7$. We can see that for all AIP, RLLL-A and RLLL-R, the average SERs of the approximate solutions increase when γ increases. Compared with AIP, RLLL-R significantly reduces the average SER of the approximate solutions. And the performance of RLLL-A is always in-between the performance of the other two. When γ is large ($\tilde{\boldsymbol{H}}$ is ill-conditioned), RLLL-A performs similarly to AIP. This is because that when $\tilde{\boldsymbol{H}}$ is ill conditioned, the initial ρ computed in RLLL-A is usually large. In this case, the inactive set found by ISF-2 will be small and the RLLL reduction used in RLLL-A will be less effective. On the contrary, when γ is small ($\tilde{\boldsymbol{H}}$ is well-conditioned), RLLL-A performs similarly to RLLL-R. This is because that when $\tilde{\boldsymbol{H}}$ is well conditioned, the initial residuals computed in RLLL-A and RLLL-R are close to each other.

In Figure 3–14, we take $\gamma = 0.5$, $m = n = 8$ and $k = 3, 4, 5$ (corresponding to 64-QAM, 256-QAM and 1024-QAM). We can see that the average SERs of the

approximate solutions increase when k increases. RLLL-R still performs better than RLLL-A, and they both reduce SER more than AIP does. When a larger QAM constellation is used, i.e., when k is larger, the benefit of using RLLL-R and RLLL-A becomes more significant and the performance of RLLL-R and RLLL-A becomes closer. This is expected because when k is large, RLLL-R and RLLL-A can usually find larger inactive sets which will make the RLLL reduction more effective. Also, when k is large, the R-approximate computed by Algorithm 3–6 becomes close to the Babai point computed by the AIP algorithm Algorithm 3–1. In the extreme case when k approaches to infinity, it can be seen that $\tilde{\boldsymbol{z}}$ (the solution to the real relaxation of the BILS problem, see (3.51)) used in Algorithm 3–6 is just $\check{\boldsymbol{z}}$ (the unconstrained RLS solution) used in Algorithm 3–1, and thus the R-approximate computed by Algorithm 3–6 becomes the Babai point computed Algorithm 3–1. In this case, RLLL-R becomes RLLL-A.

In Figure 3–15, we take $\gamma = 0.5$, $k = 4$ and $m = n = 4, 8, 16$. Because the curves tend to be very close to each other for different values of m and n in this setup, we separate the results into three sub-figures for clearness. We can see that for all of the three algorithms, when the SNR is low, SER increases as the number of antennas increases, and when the SNR is high, SER decreases as the number of antennas increases (this can be seen more clearly in Figure 3–16 where we use AIP as an example). This indicates that in the current spacial correlation model, using less antennas is preferred in a noisy environment. Otherwise, using more antennas is preferred. In Figure 3–15, we can also see that RLLL-R still performs the best,

AIP still performs the worst and RLLL-A performs similarly to RLLL-B when the dimension is small and it performs similarly to AIP when the dimension is large.

The drawback of computing the approximate solution using RLLL-A and RLLL-R is that the cost of the RLLL based reductions is higher than the cost the AIP reduction. In Table 3–2, we list the average cost of computing the the approximate solutions in Figure 3–13, Figure 3–14 and Figure 3–15. Because of the space limitation, in the table, we use A and R to represent RLLL-A and RLLL-R, respectively. We can see that compared with AIP, using RLLL-A to compute approximate solutions generally costs about 3 to 5 times more, and using RLLL-R costs about 4 to 6 times more.

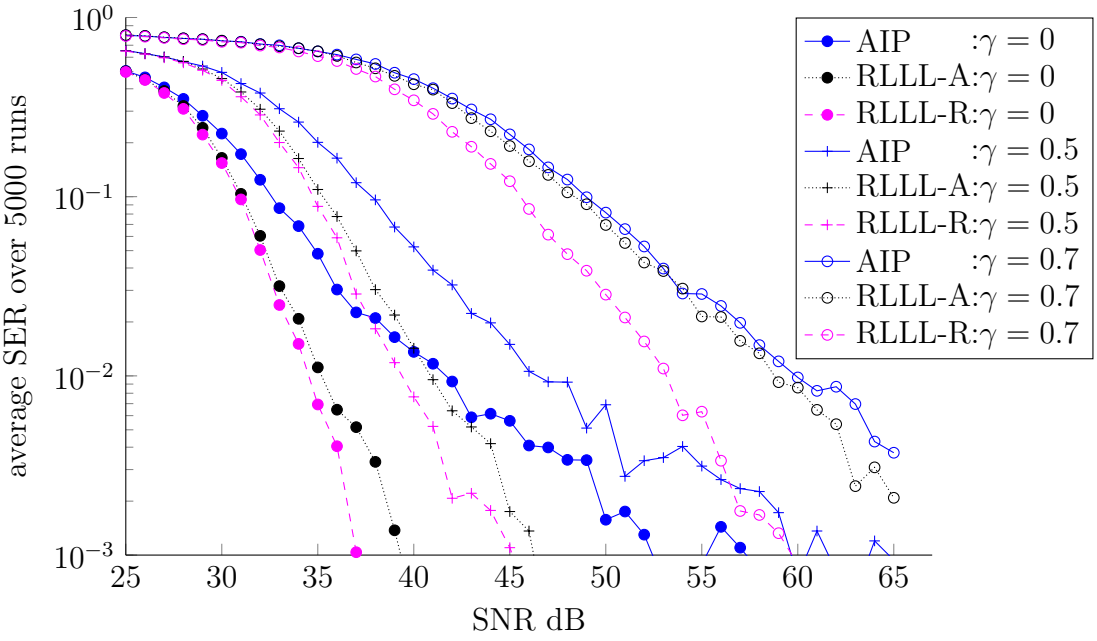


Figure 3–13: 256-QAM, $m = n = 8$

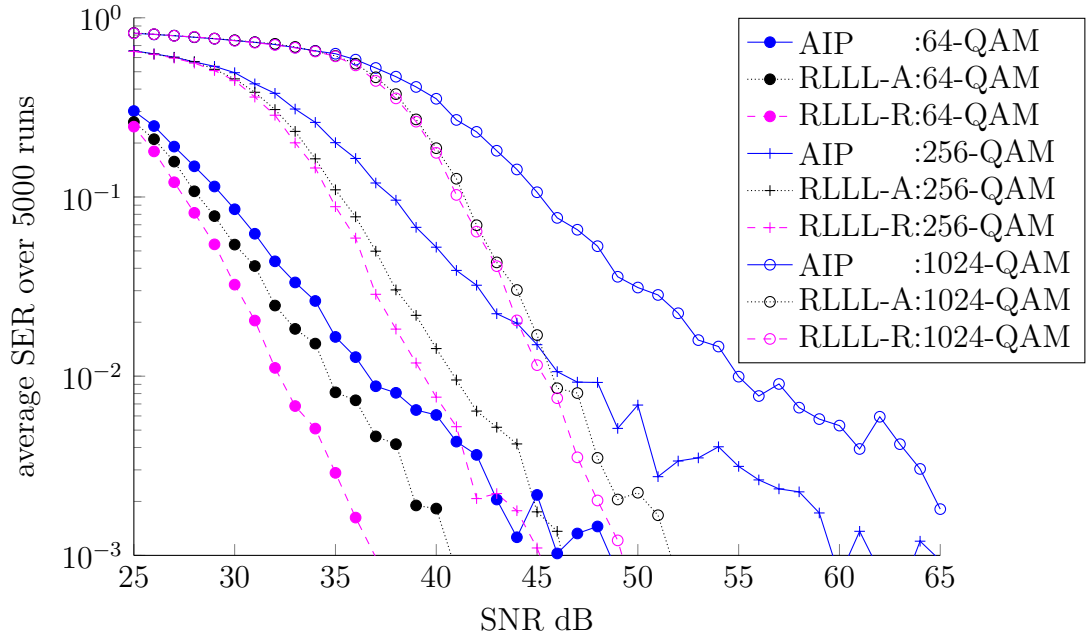


Figure 3–14: $\gamma = 0.5$, $m = n = 8$

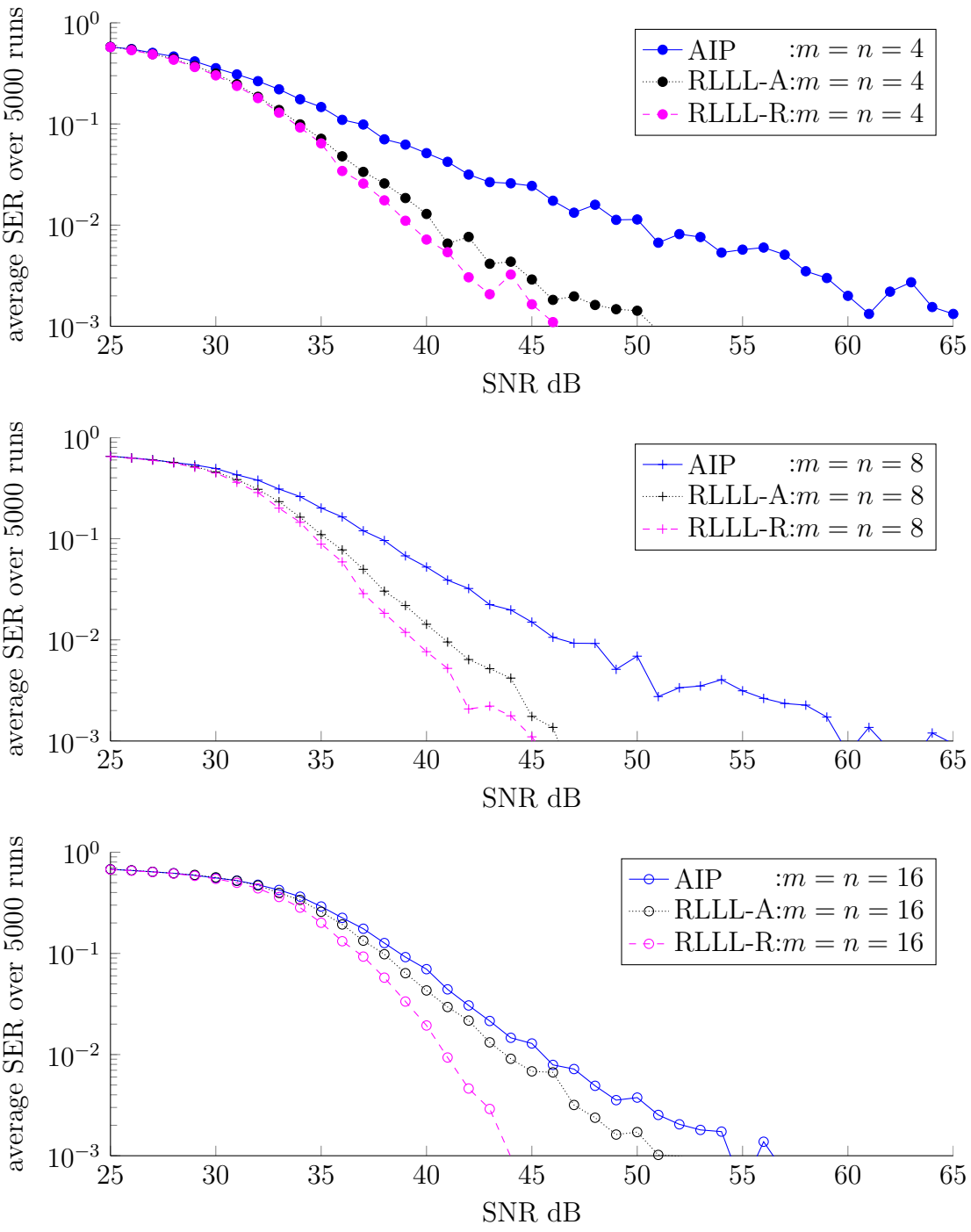


Figure 3–15: 256-QAM, $\gamma = 0.5$

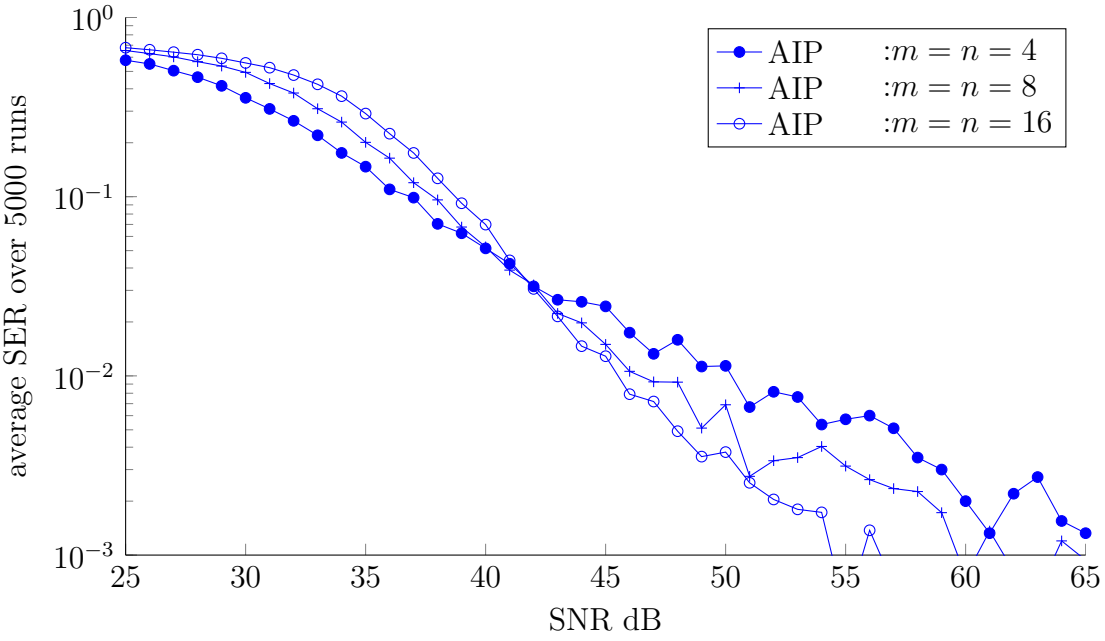


Figure 3–16: 256-QAM, $\gamma = 0.5$

Table 3–2: Average runtime to compute the approximate solutions

	AIP	A	R	AIP	A	R	AIP	A	R
Fig. 3–13	$\gamma = 0$			$\gamma = 5$			$\gamma = 7$		
time (ms)	2.5	8.6	10.6	2.5	10.8	13.2	2.5	10.0	14.6
Fig. 3–14	64-QAM			256-QAM			1024-QAM		
time (ms)	2.5	10.9	13.5	2.5	10.8	13.2	2.5	10.7	13.3
Fig. 3–15	$m = n = 4$			$m = n = 8$			$m = n = 16$		
time (ms)	1.2	4.2	5.1	2.5	10.8	13.2	6.3	27.4	34.3

CHAPTER 4
Mixed ILS Problems With Box Constraints On The Real Variables

4.1 Introduction

This chapter is concerned with the mixed ILS problems with box constraints on the real variables (MILSBR). An MILSBR problem has the following form:

$$\min_{\substack{\boldsymbol{x}\in\mathbb{R}^{n_r},\ \boldsymbol{l}\le\boldsymbol{x}\le\boldsymbol{u}\\ \boldsymbol{z}\in\mathbb{Z}^{n_i}}} \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}-\boldsymbol{B}\boldsymbol{z}\|_2^2, \tag{4.1}$$

where $\boldsymbol{y}\in\mathbb{R}^m$ is a given real vector, $\boldsymbol{A}\in\mathbb{R}^{m\times n_r}$ and $\boldsymbol{B}\in\mathbb{R}^{\boldsymbol{m}\times n_i}$ are given real matrices, $\boldsymbol{x}\in\mathbb{R}^{n_r}$ is a real vector, $\boldsymbol{l},\boldsymbol{u}\in\mathbb{R}^{n_r}$ are given lower and upper bounds on $\boldsymbol{x}$, and $\boldsymbol{z}\in\mathbb{Z}^{n_i}$ is an integer vector. Here, we assume that $\begin{bmatrix}\boldsymbol{A} & \boldsymbol{B}\end{bmatrix}\in\mathbb{R}^{m\times n}$, where $n=n_r+n_i$, has full column rank.

One application of MILSBR lies in the GPS positioning with some prior information of the actual position. In the double-difference carrier phase GPS positioning, one needs to solve a mixed ILS problem with an unknown variable $\boldsymbol{z}$ as the integer carrier-phase ambiguity vector and $\boldsymbol{x}$ as the real position vector. In some situations, additional information can be obtained to determine a range for the possible position vector $\boldsymbol{x}$, e.g., on a bathymetric surveying boat where the real time altitudes can be obtained (within certain measurement error range) from the tide-gauge readings [111]. Higher position accuracy can be achieved by using the additional information. In this case, we can solve an MILSBR problem.

In this chapter, we propose an algorithm to solve (4.1) based on the sphere decoding approach (see Section 2.10). In Sections 4.2 and 4.3, we describe the framework of applying reduction and sphere decoding to solve (4.1). In Section 4.4, we propose an algorithm to find an initial search radius, which is an important parameter used in sphere decoding. To solve an MILSBR problem, the sphere decoding approach recursively solves a series of subproblems. Each of the subproblems is also an MILSBR problem with a lower dimension. In Section 4.5, we propose a method to find some lower bounds on the objective functions in the sub-MILSBR problems, and we will show how to improve the efficiency of the sphere decoding approach by using those lower bounds. Finally, in Section 4.6, we use numerical experiments to show that our algorithms are very effective to find a good initial search radius and good lower bounds of the sub-MILSBR problems, which can significantly improve the efficiency of sphere decoding. Numerical results indicate that the new algorithm can solve the MILSBR problems much faster than the popular commercial software package CPLEX does. The main results obtained in this chapter will appear in [26].

4.2 Reduction

Like solving OILS problems, when we solve an MILSBR problem (4.1), we first reduce the problem in order to facilitate the solving process and improve the solving efficiency. Specifically, for the MILSBR problem (4.1), we find the following matrix decomposition

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}\boldsymbol{Z} \end{bmatrix} = \boldsymbol{Q} \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{0} \end{bmatrix}, \tag{4.2}$$

where $\boldsymbol{Q} = \underset{n}{\begin{bmatrix}\boldsymbol{Q}_1 & \underset{m-n}{\boldsymbol{Q}_2}\end{bmatrix}} \in \mathbb{R}^{m\times m}$ is an orthogonal matrix, $\boldsymbol{R} = \underset{n_r}{\begin{bmatrix}\boldsymbol{R}_1 & \boldsymbol{R}_2 \\ & \boldsymbol{R}_3\end{bmatrix}}_{n_i}^{n_r}$ is an upper triangular matrix, and $\boldsymbol{Z} \in \mathbb{Z}^{n_i\times n_i}$ is a unimodular matrix. Once we have (4.2), we can reduce (4.1) to the following problem

$$\min_{\substack{\boldsymbol{x}\in\mathbb{R}^{n_r},\ \boldsymbol{l}\le\boldsymbol{x}\le\boldsymbol{u} \\ \bar{\boldsymbol{z}}\in\mathbb{Z}^{n_i}}} \left\| \begin{bmatrix}\bar{\boldsymbol{y}}_1 \\ \bar{\boldsymbol{y}}_2\end{bmatrix} - \begin{bmatrix}\boldsymbol{R}_1 & \boldsymbol{R}_2 \\ & \boldsymbol{R}_3\end{bmatrix} \begin{bmatrix}\boldsymbol{x} \\ \bar{\boldsymbol{z}}\end{bmatrix} \right\|_2^2, \tag{4.3}$$

where $\bar{\boldsymbol{y}} = \underset{n_i}{\begin{bmatrix}\bar{\boldsymbol{y}}_1 \\ \bar{\boldsymbol{y}}_2\end{bmatrix}}^{n_r} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{y} \in \mathbb{R}^n$ and $\bar{\boldsymbol{z}} = \boldsymbol{Z}^{-1}\boldsymbol{z} \in \mathbb{Z}^{n_i}$. After we find the solution $\boldsymbol{x}^*$ and $\bar{\boldsymbol{z}}^*$ of (4.3), we can let $\boldsymbol{z}^* = \boldsymbol{Z}\bar{\boldsymbol{z}}^*$, and $\boldsymbol{x}^*$, $\boldsymbol{z}^*$ give the solution of (4.1).

In the reduction, the unimodular matrix $\boldsymbol{Z}$ is not unique and different $\boldsymbol{Z}$ will give different $\boldsymbol{R}$ which will affect the efficiency of sphere decoding. In Section 4.3, we will see that the MILSBR problem (4.1) can be solved based on a sphere decoding process on the lattice $\Lambda(\boldsymbol{R}_3) = \{\boldsymbol{R}_3\bar{\boldsymbol{z}} \mid \bar{\boldsymbol{z}} \in \mathbb{Z}^{n_i}\}$. To make sphere decoding more efficient, we want to choose an appropriate $\boldsymbol{Z}$ that improves the orthogonality of the bases of $\Lambda(\boldsymbol{R}_3)$, see Section 1.4. In this thesis, we propose to use the well-known LLL reduction algorithm [60] to form $\boldsymbol{Z}$.

To use the LLL algorithm to form $\boldsymbol{Z}$, we first use the QR factorization to compute $\boldsymbol{R}_1$. Assume we have the following QR factorization,

$$\boldsymbol{A} = \boldsymbol{Q}_A \begin{bmatrix}\boldsymbol{R}_1 \\ \boldsymbol{0}\end{bmatrix}, \tag{4.4}$$

where $\boldsymbol{Q}_A \in \mathbb{R}^{m\times m}$ is orthogonal and $\boldsymbol{R}_1 \in \mathbb{R}^{n_r\times n_r}$ is upper triangular. Let $\bar{\boldsymbol{B}} = \boldsymbol{Q}_A^{\mathsf{T}}\boldsymbol{B} = \underset{m-n_r}{\begin{bmatrix}\bar{\boldsymbol{B}}_1 \\ \bar{\boldsymbol{B}}_2\end{bmatrix}}^{n_r}$. Then, we apply the LLL algorithm (see Algorithm 2–1) to $\bar{\boldsymbol{B}}_2$

to get the following QRZ factorization:

$$\bar{\boldsymbol{B}}_2\boldsymbol{Z} = \boldsymbol{Q}_B \begin{bmatrix} \boldsymbol{R}_3 \\ \boldsymbol{0} \end{bmatrix}, \tag{4.5}$$

where $\boldsymbol{Z} \in \mathbb{Z}^{n_i \times \boldsymbol{n}_i}$ is unimodular, $\boldsymbol{Q}_B \in \mathbb{R}^{(m-n_r)\times(m-n_r)}$ is orthogonal and $\boldsymbol{R}_3 \in \mathbb{R}^{n_i \times n_i}$ is an LLL reduced upper triangular matrix. Finally, we define

$$\boldsymbol{R}_2 = \bar{\boldsymbol{B}}_1\boldsymbol{Z}, \quad \boldsymbol{Q} = \underset{n}{\begin{bmatrix} \boldsymbol{Q}_1 & \boldsymbol{Q}_2 \end{bmatrix}}_{m-n} = \boldsymbol{Q}_A \begin{bmatrix} \boldsymbol{I}_{n_r \times n_r} & \\ & \boldsymbol{Q}_B \end{bmatrix}, \quad \bar{\boldsymbol{y}} = \boldsymbol{Q}_1^{\mathsf{T}}\boldsymbol{y}. \tag{4.6}$$

Equations (4.4), (4.5) and (4.6) give us $\boldsymbol{R}$ and $\bar{\boldsymbol{y}}$ in (4.3).

Note that in the implementation of the above process, we do not need to explicitly form the orthogonal matrices $\boldsymbol{Q}_A$ and $\boldsymbol{Q}_B$. Instead, we obtain $\bar{\boldsymbol{B}}$ by updating $\boldsymbol{B}$ in the process of QR factorization (4.4) and obtain $\bar{\boldsymbol{y}}$ by updating $\boldsymbol{y}$ in the QR factorization (4.4) and in the LLL reduction described in (4.5). The pseudocode to compute (4.2) using the LLL reduction is given in Algorithm 4–1.

Algorithm 4–1 The LLL reduction for MILSBR problems

function $[\boldsymbol{R}, \boldsymbol{Z}] = \text{LLL\_MILSBR}(\boldsymbol{A}, \boldsymbol{B})$

1: compute QR factorization $\boldsymbol{A} = \boldsymbol{Q}_A \begin{bmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{bmatrix}$;

2: in the QR factorization, update $\bar{\boldsymbol{B}} = \boldsymbol{Q}_A^{\mathsf{T}}\boldsymbol{B} = \underset{m-n_r}{\begin{bmatrix} \bar{\boldsymbol{B}}_1 \\ \bar{\boldsymbol{B}}_2 \end{bmatrix}}^{n_r}$;

3: $[\boldsymbol{R}_3, \boldsymbol{Z}] = \text{LLL}(\bar{\boldsymbol{B}}_2)$; ▷ see Algorithm 2–1

4: $\boldsymbol{R}_2 = \bar{\boldsymbol{B}}_1\boldsymbol{Z}$;

5: $\boldsymbol{R} = \begin{bmatrix} \boldsymbol{R}_1 & \boldsymbol{R}_2 \\ & \boldsymbol{R}_3 \end{bmatrix}$;

4.3 Sphere Decoding for MILSBR

In this section, we propose a method to solve the reduced MILSBR problem (4.3) using sphere decoding algorithms.

Recall that to solve the reduced OILS problem (2.3), a sphere decoder enumerates a series of integer points $\boldsymbol{z}$ in a hyper-ellipsoid (2.10). These integer points enumerated by the sphere decoder are referred to as the candidate solutions. Then, among all candidates, the decoder finds the one that minimizes the objective function and returns it as the optimal solution. To find the solution of the MILSBR problem (4.3), we can also use a sphere decoder to find a series of candidate solutions first, and then choose the optimal solution among them. Below we show how to do it.

Suppose $\begin{bmatrix}\boldsymbol{x}^*\\ \bar{\boldsymbol{z}}^*\end{bmatrix}$ is the optimal solution to (4.3). Assume that we have $\rho \in \mathbb{R}^+$ such that for $\boldsymbol{x} = \boldsymbol{x}^*$ and $\bar{\boldsymbol{z}} = \bar{\boldsymbol{z}}^*$, the following inequality holds

$$\left\| \begin{bmatrix}\bar{\boldsymbol{y}}_1\\ \bar{\boldsymbol{y}}_2\end{bmatrix} - \begin{bmatrix}\boldsymbol{R}_1 & \boldsymbol{R}_2\\ & \boldsymbol{R}_3\end{bmatrix} \begin{bmatrix}\boldsymbol{x}\\ \bar{\boldsymbol{z}}\end{bmatrix} \right\|_2^2 < \rho^2. \tag{4.7}$$

The left hand side of (4.7) can be expanded to $\|\bar{\boldsymbol{y}}_2 - \boldsymbol{R}_3\bar{\boldsymbol{z}}\|_2^2 + \|\bar{\boldsymbol{y}}_1 - \boldsymbol{R}_2\bar{\boldsymbol{z}} - \boldsymbol{R}_1\boldsymbol{x}\|_2^2$. Since $\boldsymbol{x} \in \mathbb{R}^{n_r}$, there are an infinity number of points $\begin{bmatrix}\boldsymbol{x}\\ \bar{\boldsymbol{z}}\end{bmatrix}$ that satisfy (4.7). However, when $\bar{\boldsymbol{z}}$ is fixed to a specific value, e.g., $\bar{\boldsymbol{z}} = \bar{\boldsymbol{z}}'$, the optimal value for $\boldsymbol{x}$ for minimizing the left hand side of (4.7) is uniquely fixed, i.e., $\boldsymbol{x} = \boldsymbol{x}^*_{\bar{\boldsymbol{z}}'}$ where

$$\boldsymbol{x}^*_{\bar{\boldsymbol{z}}} = \underset{\boldsymbol{l}\le\boldsymbol{x}\le\boldsymbol{u}}{\operatorname{argmin}} \left\|(\bar{\boldsymbol{y}}_1 - \boldsymbol{R}_2\bar{\boldsymbol{z}}) - \boldsymbol{R}_1\boldsymbol{x}\right\|_2^2. \tag{4.8}$$

For any vector $\bar{\boldsymbol{z}} \in \mathbb{Z}^{n_i}$, define the following residual function

$$f(\bar{\boldsymbol{z}}) = \left\|(\bar{\boldsymbol{y}}_1 - \boldsymbol{R}_2\bar{\boldsymbol{z}}) - \boldsymbol{R}_1\boldsymbol{x}^*_{\bar{\boldsymbol{z}}}\right\|_2. \tag{4.9}$$

To find the optimal solution in (4.7), we actually only need to enumerate a finite number of integer points $\bar{\boldsymbol{z}}$ that satisfy

$$\|\bar{\boldsymbol{y}}_2 - \boldsymbol{R}_3\bar{\boldsymbol{z}}\|_2^2 + f^2(\bar{\boldsymbol{z}}) < \rho^2. \tag{4.10}$$

Because $f^2(\bar{\boldsymbol{z}}) \geq 0$, the inequality (4.10) indicates

$$\|\bar{\boldsymbol{y}}_2 - \boldsymbol{R}_3\bar{\boldsymbol{z}}\|_2^2 < \rho^2. \tag{4.11}$$

Note that (4.11) defines a hyper-ellipsoid, which has the same form as the OILS search region (2.10). We can use sphere decoding algorithms to enumerate all integer points that satisfy (4.11). For each integer point $\bar{\boldsymbol{z}}'$ enumerated, we compute the corresponding real vector $\boldsymbol{x}' = \boldsymbol{x}^*_{\bar{\boldsymbol{z}}'}$ as defined in (4.8). If (4.10) is satisfied with $\bar{\boldsymbol{z}} = \bar{\boldsymbol{z}}'$, we call such a pair $(\boldsymbol{x}', \bar{\boldsymbol{z}}')$ a candidate solution of (4.3). Among all the candidate solutions, the pair $(\boldsymbol{x}', \bar{\boldsymbol{z}}')$ that minimizes the objective function $\left\|\bar{\boldsymbol{y}} - \boldsymbol{R}\begin{bmatrix}\boldsymbol{x}\\ \bar{\boldsymbol{z}}\end{bmatrix}\right\|_2^2$ is the optimal solution of (4.3).

In general, the solution $\boldsymbol{x}^*_{\bar{\boldsymbol{z}}}$ in (4.8) cannot be found analytically. In this thesis, we propose to compute $\boldsymbol{x}^*_{\bar{\boldsymbol{z}}}$ numerically using an efficient algorithm given in [75] which combines the gradient projection method (see [13]) and the standard active set strategy for solving the convex constrained optimization problem (see [12] and [81, Chapter 16]).

The shrink strategy can greatly improve the search efficiency of the sphere decoding for OILS problems. For the mixed ILS problem, we can also shrink the search region in the process of sphere decoding. Assume we start sphere decoding using an initial search radius ρ. Once a candidate integer solution $\bar{\boldsymbol{z}}'$ is found, i.e., (4.10) holds with $\bar{\boldsymbol{z}} = \bar{\boldsymbol{z}}'$, we then shrink the search radius by letting $\rho^2 = \|\bar{\boldsymbol{y}}_2 - \boldsymbol{R}_3\bar{\boldsymbol{z}}'\|_2^2 + f^2(\bar{\boldsymbol{z}}')$. In this way, the sphere decoder only needs to enumerate part of the candidate solutions in the initial search region (4.11). Different sphere decoding algorithms may enumerate the points in (4.11) in different order. Some of the orders can make ρ

shrink faster than the others and thus can improve the efficiency of sphere decoding. To get the most out of shrinking, we use Schnorr-Euchner's sphere decoding algorithm (see Section 2.2) to enumerate (4.11).

4.4 Computing an Initial Search Radius

In this section, we propose an algorithm to find an initial search radius ρ for the search region (4.7). The smaller the ρ is, the more efficient the sphere decoding would be. But ρ should not be too small so that the optimal solution is excluded from the search region. To find a valid ρ, we first find a suboptimal solution to (4.3) and then use the corresponding residual as the initial search radius ρ.

To find a suboptimal solution of (4.3), a straightforward way is to first use Babai's nearest plane algorithm (see Section 2.2) to approximately solve

$$\min_{\bar{\boldsymbol{z}}\in\mathbb{Z}^{n_i}} \|\bar{\boldsymbol{y}}_2 - \boldsymbol{R}_3\bar{\boldsymbol{z}}\|_2^2 . \tag{4.12}$$

Denote the resulting approximate solution as $\bar{\boldsymbol{z}}^{\mathrm{B}}$ (which is also known as the Babai point). Then, we compute the real vector $\boldsymbol{x}^*_{\bar{\boldsymbol{z}}^{\mathrm{B}}}$ defined by (4.8). Then we can initialize

$$\rho^2 = \|\bar{\boldsymbol{y}}_2 - \boldsymbol{R}_3\bar{\boldsymbol{z}}^{\mathrm{B}}\|_2^2 + f^2(\bar{\boldsymbol{z}}^{\mathrm{B}}), \tag{4.13}$$

where f is defined in (4.9).

When computing the Babai point $\bar{\boldsymbol{z}}^{\mathrm{B}}$ in the above process, we are actually trying to minimize $\|\bar{\boldsymbol{y}}_2 - \boldsymbol{R}_3\bar{\boldsymbol{z}}\|_2$ alone without considering the interval constraints on $\boldsymbol{x}$ at all. Thus, it is possible that we end up with a large $f(\bar{\boldsymbol{z}}^{\mathrm{B}})$, which will lead to a large search radius ρ.

To improve the initial search radius, we propose a new method to compute an approximate solution for $\bar{\boldsymbol{z}}$ in (4.3) with the box constraint on $\boldsymbol{x}$ being taken into account. Assume the solution of (4.3) is $\begin{bmatrix}\boldsymbol{x}^*\\ \bar{\boldsymbol{z}}^*\end{bmatrix}$. In this method, we first compute a vector $\tilde{\boldsymbol{x}}$ as a rough approximation to $\boldsymbol{x}^*$ and use $\tilde{\boldsymbol{x}}$ to guide the computation of a suboptimal $\bar{\boldsymbol{z}}$.

To compute $\tilde{\boldsymbol{x}}$, we first interchange $\boldsymbol{R}_{:,1:n_r}$ and $\boldsymbol{R}_{:,n_r+1:n}$ in $\boldsymbol{R}$, and bring the resulting matrix back to upper triangular using an orthogonal transformation $\boldsymbol{Q}^{\mathsf{T}}$. We denote the resulting matrix as $\tilde{\boldsymbol{R}}$. Then, we update $\bar{\boldsymbol{y}}$ to $\tilde{\boldsymbol{y}}$ correspondingly using the same orthogonal transformation $\boldsymbol{Q}^{\mathsf{T}}$. This process can be described as

$$\tilde{\boldsymbol{R}} = \boldsymbol{Q}^{\mathsf{T}}\begin{bmatrix}\boldsymbol{R}_2 & \boldsymbol{R}_1\\ \boldsymbol{R}_3 & \end{bmatrix} = \begin{bmatrix}\tilde{\boldsymbol{R}}_1 & \tilde{\boldsymbol{R}}_2\\ & \tilde{\boldsymbol{R}}_3\end{bmatrix}\begin{matrix}n_i\\ n_r\end{matrix}, \quad \tilde{\boldsymbol{y}} = \boldsymbol{Q}^{\mathsf{T}}\bar{\boldsymbol{y}} = \begin{bmatrix}\tilde{\boldsymbol{y}}_1\\ \tilde{\boldsymbol{y}}_2\end{bmatrix}\begin{matrix}n_i\\ n_r\end{matrix}. \tag{4.14}$$

$$\begin{matrix} & \\ n_i & n_r\end{matrix}$$

Then, the original reduced MILSBR problem (4.3) is equivalent to

$$\min_{\substack{\tilde{\boldsymbol{z}}\in\mathbb{Z}^{n_i}\\ \boldsymbol{l}\le\boldsymbol{x}\le\boldsymbol{u}}} \left\| \begin{bmatrix}\tilde{\boldsymbol{y}}_1\\ \tilde{\boldsymbol{y}}_2\end{bmatrix} - \begin{bmatrix}\tilde{\boldsymbol{R}}_1 & \tilde{\boldsymbol{R}}_2\\ & \tilde{\boldsymbol{R}}_3\end{bmatrix}\begin{bmatrix}\tilde{\boldsymbol{z}}\\ \boldsymbol{x}\end{bmatrix}\right\|_2^2, \tag{4.15}$$

and $\begin{bmatrix}\bar{\boldsymbol{z}}^*\\ \boldsymbol{x}^*\end{bmatrix}$ is the optimal solution to (4.15). Let

$$\tilde{\boldsymbol{x}} = \operatorname*{argmin}_{\boldsymbol{l}\le\boldsymbol{x}\le\boldsymbol{u}} \left\|\tilde{\boldsymbol{y}}_2 - \tilde{\boldsymbol{R}}_3\boldsymbol{x}\right\|_2^2. \tag{4.16}$$

Like (4.8), $\tilde{\boldsymbol{x}}$ in (4.16) can be solved numerically using the algorithm proposed in [75]. If we fix $\boldsymbol{x} = \tilde{\boldsymbol{x}}$, (4.15) can be transformed to the following OILS problem

$$\min_{\tilde{\boldsymbol{z}}\in\mathbb{Z}^{n_i}} \left\|(\tilde{\boldsymbol{y}}_1 - \tilde{\boldsymbol{R}}_2\tilde{\boldsymbol{x}}) - \tilde{\boldsymbol{R}}_1\bar{\boldsymbol{z}}\right\|_2^2. \tag{4.17}$$

We use the Babai point $\tilde{\boldsymbol{z}}^{\mathrm{B}}$ as a suboptimal solution to (4.17). After $\tilde{\boldsymbol{z}}^{\mathrm{B}}$ is computed, we recompute the approximation to the real vector $\boldsymbol{x}^*_{\tilde{\boldsymbol{z}}^{\mathrm{B}}}$. Finally, the initial search

radius is computed as follows:

$$\rho^2 = \|\bar{\boldsymbol{y}}_2 - \boldsymbol{R}_3\tilde{\boldsymbol{z}}^{\mathrm{B}}\|_2^2 + f^2(\tilde{\boldsymbol{z}}^{\mathrm{B}}). \tag{4.18}$$

Obviously, $\tilde{\boldsymbol{z}}^{\mathrm{B}}$ is affected by the box constraint $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$. This new method essentially uses the information of the box constraint on $\boldsymbol{x}$ to guide the computation of the approximate solution $\tilde{\boldsymbol{z}}^{\mathrm{B}}$. In contrast, $\bar{\boldsymbol{z}}^{\mathrm{B}}$, the Babai point of (4.12), is totally independent of the box constraint. Because of the use of the information in the box constraint, the new method is able to find a much smaller search radius than the straightforward Babai based method does. We use Example 4–1, where $n_i = n_r = 1$, to illustrate this. The effectiveness of the new method is also observed in the numerical experiments, see Section 4.6. The pseudocode of the new algorithm is given in Algorithm 4–2.

Algorithm 4–2 Find an initial search radius for a MILSBR problem

function $\rho = \text{initial\_radius}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \boldsymbol{l}, \boldsymbol{u})$

1: Compute $\tilde{\boldsymbol{R}}$ and $\tilde{\boldsymbol{y}}$ using (4.14);
2: Solve (4.16) for $\tilde{\boldsymbol{x}}$;
3: Compute the Babai point $\tilde{\boldsymbol{z}}^{\mathrm{B}}$ for (4.17);
4: Compute $\boldsymbol{x}^*_{\tilde{\boldsymbol{z}}^{\mathrm{B}}}$ by solving (4.8);
5: $\rho = \left\| \bar{\boldsymbol{y}} - \boldsymbol{R} \begin{bmatrix} \boldsymbol{x}^*_{\tilde{\boldsymbol{z}}^{\mathrm{B}}} \\ \tilde{\boldsymbol{z}}^{\mathrm{B}} \end{bmatrix} \right\|_2$;

Example 4–1. *Here we use a 2-dimensional example to show that* (4.18) *is better than* (4.13) *in finding a small* ρ*. Assume we have*

$$\boldsymbol{R} = \begin{bmatrix} 2.5 & 4 \\ & 3 \end{bmatrix}, \quad \bar{\boldsymbol{y}} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}, \quad l = 4, \quad u = 5.$$

And we want to find $\boldsymbol{x} \in [l, u]$ *and* $\bar{z} \in \mathbb{Z}$ *to make* $\left\|\bar{\boldsymbol{y}} - \boldsymbol{R}\begin{bmatrix} x & \bar{z} \end{bmatrix}^{\mathsf{T}}\right\|_2^2$ *small. In the first method, we first compute the Babai point* $\bar{z}^{\mathrm{B}} = \lfloor 5/3 \rceil = 2$*. Based on this Babai point, we compute*

$$x^*_{\bar{z}^{\mathrm{B}}} = \underset{4 \le x \le 5}{\operatorname{argmin}} \left((0 - 4\bar{z}^{\mathrm{B}}) - 2.5x\right)^2 = \operatorname{median}\left(-8/2.5, 4, 5\right) = 4.$$

Using $\bar{z}^{\mathrm{B}}$ *and* $x^*_{\bar{z}^{\mathrm{B}}}$*, we obtain* $\rho^2 = \left\|\bar{\boldsymbol{y}} - \boldsymbol{R}\begin{bmatrix} 4 & 2 \end{bmatrix}^{\mathsf{T}}\right\|_2^2 = 18^2 + 1^2 = 325.$

If we use the new method, we first permute the columns of $\boldsymbol{R}$ *to get* $\tilde{\boldsymbol{R}}$*, i.e.,*

$$\tilde{\boldsymbol{R}} = \begin{bmatrix} 5 & 2 \\ & 1.5 \end{bmatrix} \quad \tilde{\boldsymbol{y}} = \begin{bmatrix} 3 \\ -4 \end{bmatrix}.$$

Then we compute $\tilde{x} = \operatorname{median}\left(-4/1.5, 4, 5\right) = 4$*. Based on* $\tilde{x}$*, we compute the Babai point* $\tilde{z}^{\mathrm{B}} = \lfloor (3 - 2\tilde{x})/5 \rceil = -1$*. And finally, we compute*

$$x^*_{\tilde{z}^{\mathrm{B}}} = \underset{4 \le x \le 5}{\operatorname{argmin}} \left((0 - 4\tilde{z}^{\mathrm{B}}) - 2.5x\right)^2 = \operatorname{median}\left(4/2.5, 4, 5\right) = 4.$$

Based on $\tilde{z}^{\mathrm{B}}$ *and* $x^*_{\tilde{z}^{\mathrm{B}}}$*, we get* $\rho^2 = \left\|\bar{\boldsymbol{y}} - \boldsymbol{R}\begin{bmatrix} 4 & -1 \end{bmatrix}^{\mathsf{T}}\right\|_2^2 = 6^2 + 8^2 = 100.$

Here we can see that because the computation of $\bar{z}^{\mathrm{B}}$ *does not consider the interval constraints on* x*, we end up with a large overall residual. Since the computation of* $\tilde{z}^{\mathrm{B}}$ *uses the information of the interval constraints on* x*, the new method gives us a smaller initial search radius.*

4.5 Improving the Efficiency of Sphere Decoding using Lower Bounds

In the sphere decoding process to find candidate solutions in the search region (4.11), $\bar{z}_{n_i}$ is enumerated over the integer values that satisfy the following inequality (see (2.12a)):

$$(\bar{y}_n - r_{nn}\bar{z}_{n_i})^2 < \rho^2. \tag{4.19}$$

Sphere decoding is essentially a depth-first search in a search tree (see Section 2.2), and each value of $\bar{z}_{n_i}$ satisfying (4.19) corresponds to a branch at the n-th level of the search tree.

Recall that (4.7) is a necessary condition for the optimal solution to (4.3). Let $\boldsymbol{w} = \begin{bmatrix} \boldsymbol{x} \\ \bar{\boldsymbol{z}} \end{bmatrix}$ be a vector satisfying (4.7). Define the following partitions:

$$\bar{\boldsymbol{y}} = \begin{bmatrix} \hat{\boldsymbol{y}} \\ \bar{y}_n \end{bmatrix} \begin{matrix} n-1 \\ 1 \end{matrix}, \quad \boldsymbol{R} = \begin{bmatrix} \hat{\boldsymbol{R}} & \hat{\boldsymbol{r}} \\ & r_{nn} \end{bmatrix} \begin{matrix} n-1 \\ 1 \end{matrix}, \quad \boldsymbol{w} = \begin{bmatrix} \hat{\boldsymbol{w}} \\ w_n \end{bmatrix} \begin{matrix} n-1 \\ 1 \end{matrix}. \tag{4.20}$$

$$\begin{matrix} & n-1 & 1 \end{matrix}$$

Here w_n is just $\bar{z}_{n_i}$. With (4.20), the inequality (4.7) can be rewritten as

$$\left\| \begin{bmatrix} \hat{\boldsymbol{y}} \\ \bar{y}_n \end{bmatrix} - \begin{bmatrix} \hat{\boldsymbol{R}} & \hat{\boldsymbol{r}} \\ & r_{nn} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{w}} \\ w_n \end{bmatrix} \right\| < \rho^2. \tag{4.21}$$

Define function

$$g_{w_n}(\hat{\boldsymbol{w}}) = \left\| (\hat{\boldsymbol{y}} - w_n \hat{\boldsymbol{r}}) - \hat{\boldsymbol{R}} \hat{\boldsymbol{w}} \right\|_2^2. \tag{4.22}$$

The inequality (4.21) can be rewritten as $(\bar{y}_n - r_{nn} w_n)^2 + g_{w_n}(\hat{\boldsymbol{w}}) < \rho^2$. Since $g_{w_n}(\hat{\boldsymbol{w}}) \geq 0$, (4.19) follows immediately. However, if we can find a lower bound $\underline{g}_{w_n}$ for (4.22), i.e.,

$$0 \leq \underline{g}_{w_n} \leq \min_{\substack{\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ \hat{\boldsymbol{z}} \in \mathbb{Z}^{n_i - 1}}} g_{w_n} \left(\begin{bmatrix} \boldsymbol{x} \\ \hat{\boldsymbol{z}} \end{bmatrix} \right), \tag{4.23}$$

then from (4.21), we only need to enumerate values for w_n that satisfy

$$(\bar{y}_n - r_{nn} w_n)^2 < \rho^2 - \underline{g}_{w_n}. \tag{4.24}$$

With a good lower bound $\underline{g}_{w_n}$, the number of different w_n that satisfy (4.24) is expected to be less than the number of those that satisfy (4.19). This means that we could cut some of those branches in the search tree. In an n-dimensional enumeration

process of sphere decoding, for each value w_n enumerated, an $(n-1)$-dimensional subprocess is recursively invoked to enumerate $\hat{\boldsymbol{w}}$ over the region $g_{w_n}(\hat{\boldsymbol{w}}) \leq \rho^2 - (\bar{y}_n - r_{nn}w_n)^2$. This lower-bound based branch-cutting method is also applied recursively to the k-dimensional subproblems for $k = n-1, n-2, \ldots, n_r$ to improve the overall search efficiency. To reduce the overhead introduced by the computation of the lower bounds, for a given value of w_n in the enumeration, we check the inequality (4.19) first. If (4.19) is not satisfied, then we do not need to compute the lower bound $\underline{g}_{w_n}$. The above process of using lower bounds in sphere decoding in described in Algorithm 4–3.

In (4.23), the right hand side of the second inequality is actually an $(n-1)$-dimensional MILSBR problem. Thus $\underline{g}_{w_n}$ is a lower bound on the residual of an MILSBR problem. Some algorithms in literature that compute lower bounds of the residuals of OILS and BILS problems can be adopted to compute $\underline{g}_{w_n}$. In Subsections 4.5.1 and 4.5.2, we give two algorithms to compute lower bounds based on the algorithms in the literature and in Subsection 4.5.3 we propose a new algorithm that can find a tighter lower bound.

4.5.1 The vector norm based method to find a lower bound

In [50], a method is proposed to find lower bounds for binary quadratic optimization problems based on the minimal singular value of $\boldsymbol{R}$. This method is later improved and used in solving the ILS and BILS problems, see, e.g., [97] and [67]. In this subsection, we give a method to obtain $\underline{g}_{w_n}$ based on the same idea.

Algorithm 4–3 Sphere decoding for solving MILSBR problems

function $[\boldsymbol{x}^*, \bar{\boldsymbol{z}}^*] = \text{sphere\_decoding\_MILSBR}(\boldsymbol{R}, \bar{\boldsymbol{y}}, \boldsymbol{l}, \boldsymbol{u}, \rho)$

Initialize $\hat{\boldsymbol{y}}^{(n)} = \bar{\boldsymbol{y}}$, $s_n = 0$, $k = n$ and $k_0 = n$;
while $k \leq n$ **do**
  if $k > n_r$ **then**
    if $k \leq k_0$ **then**
      $c_k = \hat{y}_k^{(k)} / r_{kk}$, $w_k = \lfloor c_k \rceil$, $d_k = \text{sign}(c_k - w_k)$;
    else
      $w_k = w_k + d_k$, $d_k = -d_k - \text{sign}(d_k)$; ▷ Shenorr-Euchner's search order
    end if
    $s_{k-1} = s_k + r_{kk}^2 (w_k - c_k)^2$; ▷ partial residual
    if $s_{k-1} < \rho^2$ **then**
      $k_1 = k - 1$;
      $\hat{\boldsymbol{y}}^{(k_1)} = \hat{\boldsymbol{y}}_{1:k_1}^{(k)} - \boldsymbol{r}_{1:k_1,k} w_k$;
      find $\underline{g}_{\boldsymbol{w}_{k:n}}$ as a lower bound of $\left\| \hat{\boldsymbol{y}}^{(k_1)} - \hat{\boldsymbol{R}}_{1:k_1,1:k_1} \boldsymbol{w}_{1:k_1} \right\|_2^2$; ▷ see later
      if $s_{k_1} + \underline{g}_{\boldsymbol{w}_{k:n}} < \rho^2$ **then**
        $k_0 = k$, $k = k - 1$; ▷ move down in the search tree
        continue
      end if
    end if
    $k_0 = k$, $k = k + 1$; ▷ move up in the search tree
  else
    solve $\boldsymbol{x}' = \text{argmin}_{\boldsymbol{u} \leq \boldsymbol{x} \leq \boldsymbol{l}} \left\| \hat{\boldsymbol{y}}^{(n_r)} - \boldsymbol{R}_{1:n_n,1:n_r} \boldsymbol{x} \right\|_2^2$;
    if $s_{n_r} + \left\| \hat{\boldsymbol{y}}^{(n_r)} - \boldsymbol{R}_{1:n_n,1:n_r} \boldsymbol{x}' \right\|_2^2 < \rho^2$ **then**
      $\boldsymbol{x}^* = \boldsymbol{x}'$, $\bar{\boldsymbol{z}}^* = \boldsymbol{w}_{n_r+1:n}$; ▷ found a solution candidate
      $\rho^2 = s_{n_r} + \left\| \hat{\boldsymbol{y}}^{(n_r)} - \boldsymbol{R}_{1:n_r,1:n_r} \boldsymbol{x}' \right\|_2^2$; ▷ shrink the search radius
    end if
    $k_0 = k$, $k = k + 1$; ▷ move up in the search tree
  end if
end while

Define $\check{\boldsymbol{w}}^{(n-1)} = \hat{\boldsymbol{R}}^{-1} (\hat{\boldsymbol{y}} - w_n \hat{\boldsymbol{r}})$. For simplicity, we denote $\check{\boldsymbol{w}} = \check{\boldsymbol{w}}^{(n-1)}$. From (4.22), $g_{w_n}(\hat{\boldsymbol{w}})$ can be rewritten as

$$g_{w_n}(\hat{\boldsymbol{w}}) = \left\| \hat{\boldsymbol{R}} \big(\hat{\boldsymbol{w}} - \check{\boldsymbol{w}} \big) \right\|_2^2.$$

Let $\boldsymbol{d} = \hat{\boldsymbol{R}}(\hat{\boldsymbol{w}} - \check{\boldsymbol{w}})$ and $\hat{\boldsymbol{F}} = \hat{\boldsymbol{R}}^{-\mathsf{T}}$ with column partition $\hat{\boldsymbol{F}} = \begin{bmatrix} \hat{\boldsymbol{f}}_1 & \hat{\boldsymbol{f}}_2 & \dots & \hat{\boldsymbol{f}}_{n-1} \end{bmatrix}$. Then we have

$$\hat{\boldsymbol{F}}^{\mathsf{T}} \boldsymbol{d} = \hat{\boldsymbol{w}} - \check{\boldsymbol{w}}. \tag{4.25}$$

Applying the Cauchy-Schwartz inequality to (4.25) gives

$$\left\|\hat{\boldsymbol{F}}\right\|_2 \left\|\boldsymbol{d}\right\|_2 \geq \left\|\hat{\boldsymbol{F}}^{\mathsf{T}} \boldsymbol{d}\right\|_2 = \left\|\hat{\boldsymbol{w}} - \check{\boldsymbol{w}}\right\|_2.$$

Note that $\left\|\hat{\boldsymbol{F}}\right\|_2 = \sigma_{\min}^{-1}(\hat{\boldsymbol{R}})$, where $\sigma_{\min}(\hat{\boldsymbol{R}})$ is the minimal singular value of $\hat{\boldsymbol{R}}$. Thus

$$g_{w_n}(\hat{\boldsymbol{w}}) = \|\boldsymbol{d}\|_2^2 \geq \frac{\|\hat{\boldsymbol{w}} - \check{\boldsymbol{w}}\|_2^2}{\left\|\hat{\boldsymbol{F}}\right\|_2^2} = \sigma_{\min}^2(\hat{\boldsymbol{R}}) \, \|\hat{\boldsymbol{w}} - \check{\boldsymbol{w}}\|_2^2 . \tag{4.26}$$

Recall that $\hat{w}_i = x_i$ when $i \leq n_r$, and $\hat{w}_i = \bar{z}_{i-n_r}$ otherwise. Thus we have:

$$\begin{aligned} & l_i \leq \hat{w}_i \leq u_i, \quad i = 1, 2, \dots, n_r, \\ & \hat{w}_i \in \mathbb{Z}, \qquad\quad i = n_r + 1, n_r + 2, \dots, n - 1. \end{aligned} \tag{4.27}$$

Define the $(n-1)$-dimensional vector $\hat{\boldsymbol{w}}'$ as the closest $\hat{\boldsymbol{w}}$ to $\check{\boldsymbol{w}}$, i.e.,

$$\hat{w}_i' = \begin{cases} \text{median}(\check{w}_i, l_i, u_i) & \text{if } i \leq n_r, \\ \lfloor \check{w}_i \rceil & \text{otherwise.} \end{cases} \tag{4.28}$$

This gives us the following lower bound,

$$\begin{aligned} \underline{g}_{w_n} &= \sigma_{\min}^2(\hat{\boldsymbol{R}}) \min_{\hat{\boldsymbol{w}}} \|\hat{\boldsymbol{w}} - \check{\boldsymbol{w}}\|_2^2 \\ &= \sigma_{\min}^2(\hat{\boldsymbol{R}}) \sum_{i=1}^{n-1} \min_{\hat{w}_i} (\hat{w}_i - \check{w}_i)_2^2 \\ &= \sigma_{\min}^2(\hat{\boldsymbol{R}}) \sum_{i=1}^{n-1} (\hat{w}_i' - \check{w}_i)_2^2. \end{aligned} \tag{4.29}$$

Now we discuss how to compute $\underline{g}_{w_n}$ in an efficient way. Computing $\underline{g}_{w_n}$ requires $\check{\boldsymbol{w}} = \check{\boldsymbol{w}}^{(n-1)} = \hat{\boldsymbol{R}}^{-1}(\hat{\boldsymbol{y}} - w_n\hat{\boldsymbol{r}})$. Assume that we have computed $\check{\boldsymbol{w}}^{(n)} = \boldsymbol{R}^{-1}\bar{\boldsymbol{y}}$, which is the n-dimensional counter part of $\check{\boldsymbol{w}}^{(n-1)}$. From partition (4.20), we have

$$\hat{\boldsymbol{y}} = \begin{bmatrix} \hat{\boldsymbol{R}} & \hat{\boldsymbol{r}} \end{bmatrix} \begin{bmatrix} \check{\boldsymbol{w}}^{(n)}_{1:n-1} \\ \check{w}^{(n)}_n \end{bmatrix} = \hat{\boldsymbol{R}}\check{\boldsymbol{w}}^{(n)}_{1:n-1} + \hat{\boldsymbol{r}}\check{w}^{(n)}_n.$$

Define $\hat{\boldsymbol{h}} = \hat{\boldsymbol{R}}^{-1}\hat{\boldsymbol{r}}$. We have

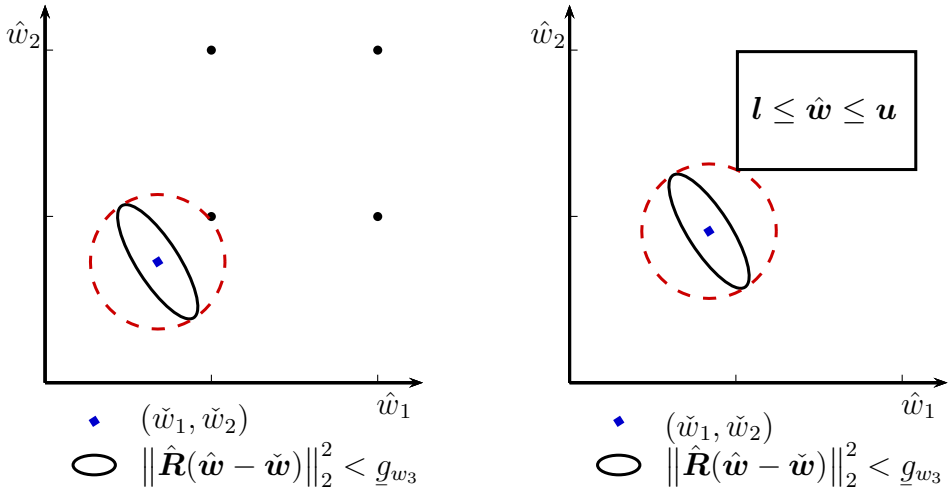
$$\begin{aligned} \check{\boldsymbol{w}}^{(n-1)} &= \hat{\boldsymbol{R}}^{-1}\hat{\boldsymbol{y}} - w_n\hat{\boldsymbol{R}}^{-1}\hat{\boldsymbol{r}} \\ &= \check{\boldsymbol{w}}^{(n)}_{1:n-1} + (\check{w}^{(n)}_n - w_n)\hat{\boldsymbol{h}}. \end{aligned} \tag{4.30}$$

Since $\hat{\boldsymbol{h}}$ is constant in the enumeration process of sphere decoding and can be precomputed, (4.30) provides us an efficient way to compute $\check{\boldsymbol{w}}$ in $O(n)$. With $\hat{\boldsymbol{h}}^{(k)} = \boldsymbol{R}^{-1}_{1:k,1:k}\boldsymbol{r}_{1:k,k+1}$ precomputed, we can recursively compute $\check{\boldsymbol{w}}^{(k)}$ for the k-dimensional sphere decoding subprocess at a cost of $O(k)$. With (4.28), (4.30) and (4.29), the lower bound $\underline{g}_{w_n}$ given in (4.29) can be computed. Noting that $\sigma_{\min}(\hat{\boldsymbol{R}})$ is also constant through out the whole process of sphere decoding and it can be precomputed, the complexity of computing $\underline{g}_{w_n}$ for each value of w_n is $O(n)$ (here we do not include the cost of any precomputations because this cost is constant to the number of values of w_n in the enumeration).

To understand this method from geometric perspective, let us assume we have an hyper-ellipsoid $g_{w_n}(\hat{\boldsymbol{w}}) = \left\|\hat{\boldsymbol{R}}(\hat{\boldsymbol{w}} - \check{\boldsymbol{w}})\right\|_2^2 < \hat{\rho}^2$. The inequality (4.26) indicates that if $\hat{\boldsymbol{w}}$ is in the above hyper-ellipsoid, then it is also in the following hyper-sphere

$$\sigma^2_{\min}(\hat{\boldsymbol{R}})\left\|\hat{\boldsymbol{w}} - \check{\boldsymbol{w}}\right\|_2^2 < \hat{\rho}^2. \tag{4.31}$$

This means that the hyper-sphere (4.31) covers the hyper-ellipsoid $g_{w_n}(\hat{\boldsymbol{w}}) < \hat{\rho}^2$. We also know that in the MILSBR problem, the entries of $\hat{\boldsymbol{w}}$ are subject to interval constraints or integer constraints, see (4.27). The norm-wise lower bound algorithm computes $\underline{g}_{w_n}$ as the largest value for $\hat{\rho}$ such that the hyper-sphere (4.31) contains no valid $\hat{\boldsymbol{w}}$ that satisfies (4.27). In this way, it ensures that $g_{w_n}(\hat{\boldsymbol{w}}) \geq \underline{g}_{w_n}$ for any $\hat{\boldsymbol{w}}$ that satisfies (4.27). To illustrate this geometry relationship of the hyper-ellipsoid, the hyper-sphere and the valid values for $\hat{\boldsymbol{w}}$, we give two examples in Figure 4–1 where $\hat{\boldsymbol{w}}$ are 2-dimensional vectors which are subject to the integer constraint $\hat{\boldsymbol{w}} \in \mathbb{Z}^2$ or the box constraint $\boldsymbol{l} \leq \hat{\boldsymbol{w}} \leq \boldsymbol{u}$ respectively.



(a) Integer constraint (b) Box constraint

Figure 4–1: Norm-wise lower bound

It is easy to see that the lower-bound computed using the above method is tight when the shape of the hyper-ellipsoid $\boldsymbol{g}_{w_n}(\hat{\boldsymbol{w}}) < \hat{\rho}^2$ is close to the hyper-sphere (4.31), i.e., $\sigma_{\min}(\hat{\boldsymbol{R}})/\sigma_{\max}(\hat{\boldsymbol{R}})$ is close to 1. To improve the lower bound, [50] propose to precondition $\hat{\boldsymbol{R}}$ before computing the lower bound. Let $\boldsymbol{C} \in \mathbb{R}^{n \times n}$ be a diagonal

matrix. Multiplying both sides of (4.25) by $\boldsymbol{C}$ gives us

$$\tilde{\boldsymbol{F}}^{\mathsf{T}}\boldsymbol{d} = \boldsymbol{C}(\hat{\boldsymbol{w}} - \check{\boldsymbol{w}}),$$

where $\tilde{\boldsymbol{F}} = \hat{\boldsymbol{F}}\boldsymbol{C}$. Following the same deduction as (4.26), we have:

$$g_{w_n}(\hat{\boldsymbol{w}}) = \|\boldsymbol{d}\|_2^2 \geq \frac{\|\boldsymbol{C}(\hat{\boldsymbol{w}} - \check{\boldsymbol{w}})\|_2^2}{\left\|\tilde{\boldsymbol{F}}^{\mathsf{T}}\right\|_2^2} = \frac{\sum_{i=1}^{n-1} c_{ii}^2 \left(\hat{w}_i - \check{w}_i\right)_2^2}{\left\|\tilde{\boldsymbol{F}}\right\|_2^2}. \tag{4.32}$$

This inequality gives us the following lower bound:

$$\underline{g}_{w_n,\boldsymbol{C}} = \left\|\tilde{\boldsymbol{F}}\right\|_2^{-2} \sum_{i=1}^{n-1} c_{ii}^2 \left(\hat{w}_i' - \check{w}_i\right)_2^2. \tag{4.33}$$

It is easy to see that (4.29) is a special case of (4.33) where $\boldsymbol{C} = \boldsymbol{I}$. We can adjust the value of the above lower bound by choosing a different matrix $\boldsymbol{C}$. In, e.g., [50, 66], it is suggested to solve a semi-definite programming (SDP) problem to find $\boldsymbol{C}$. Other methods of choosing $\boldsymbol{C}$ can be found in, e.g., [97, 21, 22]. Here we choose $c_{ii} = \left\|\hat{\boldsymbol{f}}_i\right\|_2^{-1}$, where $\hat{\boldsymbol{f}}_i$ is the i-th column of $\hat{\boldsymbol{F}}$ for $i = 1, 2, \ldots, n-1$, as [21] does. Then (4.33) becomes

$$\underline{g}_{w_n} = \left\|\tilde{\boldsymbol{F}}\right\|_2^{-2} \sum_{i=1}^{n-1} \frac{\left(\hat{w}_i' - \check{w}_i\right)_2^2}{\left\|\hat{\boldsymbol{f}}_i\right\|_2^2}, \tag{4.34}$$

where matrix $\tilde{\boldsymbol{F}}$ has unit column vectors. Computing $\boldsymbol{C}$ in the above way requires much less computations than the SDP based methods, and can be done prior to the enumeration process of sphere decoding. This meets our need to compute the lower bounds efficiently. The lower bound computed by (4.34) is optimal if matrix $\hat{\boldsymbol{F}}$ has orthogonal column vectors. To see this, we note that $\tilde{\boldsymbol{F}}$ becomes an orthogonal matrix in this case, and thus the inequality in (4.32) takes only the equality sign. This means that the lower bound $\underline{g}_{w_n}$ computed by (4.34) is reachable, i.e., $\underline{g}_{w_n} = g_{w_n}(\hat{\boldsymbol{w}}')$.

4.5.2 The component-wise method to find a lower bound

In this subsection, we show how to compute the lower bound $\underline{g}_{w_n}$ based on a method proposed in [39]. The original method in [39] is proposed to compute lower bounds for solving the BILS problems. The same method is also used in [21] to compute lower bounds for convex constrained quadratic integer programming.

Equation (4.25) indicates that

$$\left|\hat{\boldsymbol{f}}_i^{\mathsf{T}}\boldsymbol{d}\right| = \left|\boldsymbol{e}_i^{\mathsf{T}}\hat{\boldsymbol{F}}^{\mathsf{T}}\boldsymbol{d}\right| = \left|\boldsymbol{e}_i^{\mathsf{T}}(\hat{\boldsymbol{w}} - \check{\boldsymbol{w}})\right| = \left|\hat{w}_i - \check{w}_i\right|. \tag{4.35}$$

From (4.35), by the Cauchy-Schwartz inequality we have

$$\left\|\hat{\boldsymbol{f}}_i\right\|_2 \left\|\boldsymbol{d}\right\|_2 \geq \left|\hat{\boldsymbol{f}}_i^{\mathsf{T}}\boldsymbol{d}\right| = \left|\hat{w}_i - \check{w}_i\right|, \quad i = 1, 2 \ldots, n-1. \tag{4.36}$$

Recall that the above inequality is essentially what we used in designing the ISF-1 to derive the covering box that gives the boundaries of $\hat{w}_i$ for $i = 1, 2, \ldots, n$, where $\|\boldsymbol{d}\|_2$ is upper bounded, i.e., $\|\boldsymbol{d}\|_2^2 < \rho^2$, see Subsection 3.3.1. Now, we will reversely use it to derive a lower bound on $\|\boldsymbol{d}\|_2$, given the constraint on $\hat{w}_i$. From (4.36), we have

$$g_{w_n}(\hat{\boldsymbol{w}}) = \|\boldsymbol{d}\|_2^2 \geq \max_{i\in\{1,\ldots,n-1\}} \frac{(\hat{w}_i - \check{w}_i)^2}{\left\|\hat{\boldsymbol{f}}_i\right\|_2^2}.$$

Since $\hat{w}_i$ is either integer constrained or interval constrained, this inequality gives us the following formula to compute the lower bound:

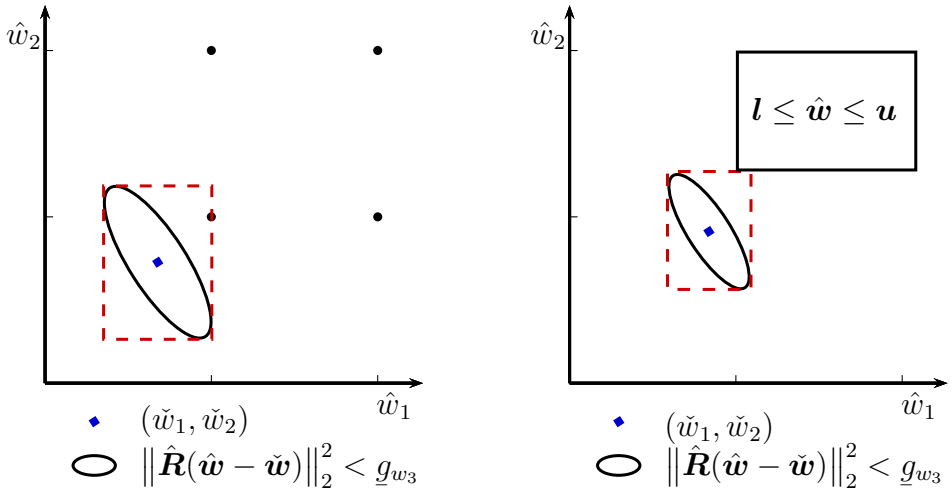
$$\underline{g}_{w_n} = \max_{i\in\{1,\ldots,n-1\}} \frac{\min_{\hat{w}_i}(\hat{w}_i - \check{w}_i)^2}{\left\|\hat{\boldsymbol{f}}_i\right\|_2^2} = \max_{i\in\{1,\ldots,n-1\}} \frac{(\hat{w}_i' - \check{w}_i)^2}{\left\|\hat{\boldsymbol{f}}_i\right\|_2^2}, \tag{4.37}$$

where $\hat{w}_i'$ is defined in (4.28). Vector $\check{\boldsymbol{w}}^{n-1}$ can be computed using (4.37). With $\left\|\hat{\boldsymbol{f}}_i\right\|_2^2$ precomputed, the lower bound $\underline{g}_{w_n}$ can be computed in $O(n)$.

To understand the component-wise method geometrically, we still consider a $(n-1)$-dimensional hyper-ellipsoid $g_{w_n}(\hat{\boldsymbol{w}}) < \hat{\rho}^2$. Then inequalities in (4.36) indicate that for any $\hat{\boldsymbol{w}}$ in the above hyper-ellipsoid, we have,

$$\frac{(\hat{w}_i - \check{w}_i)^2}{\left\|\hat{\boldsymbol{f}}_i\right\|_2^2} < \hat{\rho}^2, \quad i = 1, 2, \ldots n-1. \tag{4.38}$$

In geometry, (4.38) defines an $(n-1)$-dimensional box that covers the hyper-ellipsoid $g_{w_n}(\hat{\boldsymbol{w}}) < \hat{\rho}^2$. The component-wise method then computes $\underline{g}_{w_n}$ as the largest value for $\hat{\rho}$ such that the box (4.38) contains no valid $\hat{\boldsymbol{w}}$ that satisfies (4.27). This guarantees that $g_{w_n}(\hat{\boldsymbol{w}}) \geq \underline{g}_{w_n}$ for any $\hat{\boldsymbol{w}}$ that satisfies (4.27). In Figure 4–2, we use the same two examples we gave in Figure 4–1 to illustrate how $\underline{g}_{w_n}$ is computed based on the box (4.38).



(a) Integer constraint (b) Box constraint

Figure 4–2: Component-wise lower bound

4.5.3 The second-order component-wise method to find a lower bound

Recall that the norm-wise method introduced in Section 4.5.1 takes vector norms of each sides of (4.25), and the component-wise method introduced in Section 4.5.2 considers the components on each side of (4.25) one by one. In this subsection, we propose a new method to compute $\underline{g}_{w_n}$ which considers two components on each side of (4.25) at a time. We call this method the second-order component-wise method. As we have seen that the component-wise method is to reversely apply the ISF-1 algorithm, the second-order component-wise method is to reversely apply the ISF-2 algorithm introduced in Section 3.3.2.

For two different integer indices $i, j \leq n_r$, multiplying both sides of (4.25) by $\begin{bmatrix} \boldsymbol{e}_i & \boldsymbol{e}_j \end{bmatrix}^{\mathsf{T}}$ gives us

$$\begin{bmatrix} \hat{\boldsymbol{f}}_i & \hat{\boldsymbol{f}}_j \end{bmatrix}^{\mathsf{T}} \boldsymbol{d} = \begin{bmatrix} \hat{w}_i - \check{w}_i \\ \hat{w}_j - \check{w}_j \end{bmatrix}. \tag{4.39}$$

Let the QR factorization of $\begin{bmatrix} \boldsymbol{f}_i & \boldsymbol{f}_j \end{bmatrix}$ be

$$\begin{bmatrix} \hat{\boldsymbol{f}}_i & \hat{\boldsymbol{f}}_j \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_1^{(ij)} & \boldsymbol{Q}_2^{(ij)} \end{bmatrix} \begin{bmatrix} \boldsymbol{S}^{(ij)} \\ \boldsymbol{0} \end{bmatrix},$$

where $\boldsymbol{Q}^{(ij)} = \underset{2}{\begin{bmatrix} \boldsymbol{Q}_1^{(ij)} & \boldsymbol{Q}_2^{(ij)} \end{bmatrix}}_{n-2} \in \mathbb{R}^{n \times n}$ is orthogonal, and $\boldsymbol{S}^{(ij)} = \begin{bmatrix} s_{11}^{(ij)} & s_{12}^{(ij)} \\ & s_{22}^{(ij)} \end{bmatrix} \in \mathbb{R}^{2 \times 2}$. Without loss of generality, we assume $s_{11}^{(ij)}, s_{22}^{(ij)} > 0$. Let $\boldsymbol{T} = (t_{ij})_{n_r \times n_r} = \left(\hat{\boldsymbol{F}}_{:,1:n_r}\right)^{\mathsf{T}} \hat{\boldsymbol{F}}_{:,1:n_r}$. We have (see (3.36))

$$s_{11}^{(ij)} = \left\| \hat{\boldsymbol{f}}_i \right\|_2 = \sqrt{t_{ii}}, \tag{4.40a}$$

$$s_{12}^{(ij)} = \hat{\boldsymbol{f}}_i^{\mathsf{T}} \hat{\boldsymbol{f}}_i / \left\| \hat{\boldsymbol{f}}_i \right\|_2 = t_{ij} / \sqrt{t_{ii}}, \tag{4.40b}$$

$$s_{22}^{(ij)} = \sqrt{\left\| \hat{\boldsymbol{f}}_j \right\|_2^2 - \left(\hat{\boldsymbol{f}}_i^{\mathsf{T}} \hat{\boldsymbol{f}}_j / \left\| \hat{\boldsymbol{f}}_i \right\|_2 \right)^2} = \sqrt{t_{jj} - t_{ij}^2 / t_{ii}}. \tag{4.40c}$$

Define $\tilde{\boldsymbol{d}} = \begin{bmatrix}\tilde{d}_1 & \tilde{d}_2\end{bmatrix}^\mathsf{T} = \boldsymbol{Q}_1^{(ij)\mathsf{T}}\boldsymbol{d}$. It can be seen that

$$\begin{bmatrix}\hat{\boldsymbol{f}}_i & \hat{\boldsymbol{f}}_j\end{bmatrix}^\mathsf{T}\boldsymbol{d} = \begin{bmatrix}\hat{\boldsymbol{f}}_i & \hat{\boldsymbol{f}}_j\end{bmatrix}^\mathsf{T}\boldsymbol{Q}^{(ij)}\boldsymbol{Q}^{(ij)\mathsf{T}}\boldsymbol{d} = \begin{bmatrix}\boldsymbol{S}^{(ij)}\\ \boldsymbol{0}\end{bmatrix}^\mathsf{T}\begin{bmatrix}\boldsymbol{Q}_1^{(ij)\mathsf{T}}\boldsymbol{d}\\ \boldsymbol{Q}_2^{(ij)\mathsf{T}}\boldsymbol{d}\end{bmatrix} = \boldsymbol{S}^{(ij)^\mathsf{T}}\tilde{\boldsymbol{d}}.$$

Replacing the left hand side of (4.39) with $\boldsymbol{S}^{(ij)^\mathsf{T}}\tilde{\boldsymbol{d}}$, it can be rewritten as

$$\begin{aligned}\tilde{d}_1 &= \big(\hat{w}_i - \check{w}_i\big)/s_{11}^{(ij)},\\ \tilde{d}_2 &= \big(\hat{w}_j - \check{w}_j - \tilde{d}_1 s_{12}^{(ij)}\big)/s_{22}^{(ij)}.\end{aligned} \tag{4.41}$$

Since $l_i \le \hat{w}_i \le u_i$ and $l_j \le \hat{w}_j \le u_j$ for $i, j \le n_r$, from (4.41), we have

$$\frac{l_i - \check{w}_i}{\sqrt{t_{ii}}} \le \tilde{d}_1 \le \frac{u_i - \check{w}_i}{\sqrt{t_{ii}}}, \tag{4.42a}$$

$$\frac{l_j - \check{w}_j - \tilde{d}_1 t_{ij}/\sqrt{t_{ii}}}{\sqrt{t_{jj} - t_{ij}^2/t_{ii}}} \le \tilde{d}_2 \le \frac{u_j - \check{w}_j - \tilde{d}_1 t_{ij}/\sqrt{t_{ii}}}{\sqrt{t_{jj} - t_{ij}^2/t_{ii}}}. \tag{4.42b}$$

Variable t_{ij} can be represented as the difference of two non-negative numbers $t_{ij}^+ = \max(t_{ij}, 0)$ and $t_{ij}^- = \max(-t_{ij}, 0)$, i.e., $t_{ij} = t_{ij}^+ - t_{ij}^-$. From (4.42a), we have

$$(l_i t_{ij}^+ - u_i t_{ij}^- - t_{ij}\check{w}_i)/t_{ii} \le \tilde{d}_1 t_{ij}/\sqrt{t_{ii}} \le (u_i t_{ij}^+ - l_i t_{ij}^- - t_{ij}\check{w}_i)/t_{ii}. \tag{4.43}$$

Define

$$\alpha_{ij} = t_{ij}/t_{ii}, \quad \beta_{ij} = t_{jj} - t_{ij}^2/t_{ii}, \quad \tilde{w}_{ij} = \check{w}_j - \alpha_{ij}\check{w}_i,$$

$$\tilde{l}_{ij} = l_j - (u_i t_{ij}^+ - l_i t_{ij}^-)/t_{ii}, \quad \tilde{u}_{ij} = \tilde{u}_j - (l_i t_{ij}^+ - u_i t_{ij}^-)/t_{ii}.$$

From (4.42b) and (4.43), we have

$$\frac{\tilde{l}_{ij} - \tilde{w}_{ij}}{\sqrt{\beta_{ij}}} \le \tilde{d}_2 \le \frac{\tilde{u}_j - \tilde{w}_{ij}}{\sqrt{\beta_{ij}}}. \tag{4.44}$$

Define

$$\bar{s}_i = \left(\text{median}\left(\frac{l_i - \check{w}_i}{\sqrt{t_{ii}}}, \frac{u_i - \check{w}_i}{\sqrt{t_{ii}}}, 0\right)\right)^2 = \frac{\left(\text{median}\left(\check{w}_i, l_i, u_i\right) - \check{w}_i\right)^2}{t_{ii}}, \tag{4.45a}$$

$$\bar{\bar{s}}_{ij} = \left(\text{median}\left(\frac{\tilde{l}_{ij} - \tilde{w}_{ij}}{\sqrt{\beta_{ij}}}, \frac{\tilde{u}_{ij} - \tilde{w}_{ij}}{\sqrt{\beta_{ij}}}, 0\right)\right)^2 = \frac{\left(\text{median}\left(\tilde{w}_{ij}, \tilde{l}_{ij}, \tilde{u}_{ij}\right) - \tilde{w}_{ij}\right)^2}{\beta_{ij}}. \tag{4.45b}$$

From (4.42a) and (4.44), we can see that $\bar{s}_i$ and $\bar{\bar{s}}_{ij}$ are lower bounds on $\tilde{d}_1^2$ and $\tilde{d}_2^2$ respectively. It immediately follows that

$$\|\boldsymbol{d}\|_2^2 \geq \frac{\left\|\left(\boldsymbol{Q}_1^{(ij)}\right)^{\mathsf{T}}\boldsymbol{d}\right\|_2^2}{\left\|\boldsymbol{Q}_1^{(ij)}\right\|_2^2} = \frac{\left\|\tilde{\boldsymbol{d}}\right\|_2^2}{1} = \tilde{d}_1^2 + \tilde{d}_2^2 \geq \bar{s}_i + \bar{\bar{s}}_{ij}, \quad i, j \in \{1, 2, \ldots, n_r\},\ i \neq j$$

Thus, we can get the following lower bound:

$$g_{w_n}(\hat{\boldsymbol{w}}) = \|\boldsymbol{d}\|_2^2 \geq \max_{\substack{i,j \in \{1,2,\ldots,n_r\} \\ i \neq j}} \bar{s}_i + \bar{\bar{s}}_{ij}.$$

When α_{ij}, β_{ij}, $\tilde{l}_{ij}$, $\tilde{u}_{ij}$ are precomputed for all $i, j \in \{1, 2, \ldots, n_r\}$ and $i \neq j$, the cost of computing the above lower bound is $O(n_r^2)$ flops. In the above mentioned GPS positioning application, $n_r = 3$ is the dimension of the position vector, and n_i is typically around 20, when dual frequencies of GPS signals are used.

From (4.45a), (4.28) and (4.40a), we actually have $\bar{s}_i = \left(\hat{w}_i' - \check{w}_i\right)^2 / \left\|\hat{\boldsymbol{f}}_i\right\|_2^2$, which is what we used to compute the component-wise lower bound in (4.37). Thus, we can extend the definition of $\bar{s}_i$ to include $i \in \{n_r + 1, n_r + 2, \ldots, n - 1\}$ by using this formula and we always have $g_{w_n}(\hat{\boldsymbol{w}}) \geq \bar{s}_i$ for $i \in \{1, 2, \ldots, n - 1\}$. However, $\bar{\bar{s}}_{ij}$ is only defined for the real part of the MILSBR problem, i.e., $i, j \in \{1, 2, \ldots, n_r\}$ (note that it uses the box constraints on the real variables). To fully use the information in the integer constraint on $\bar{\boldsymbol{z}}$ and the box constraint on $\boldsymbol{x}$, we take the following

lower bound:

$$\underline{g}_{w_n} = \max \left(\max_{i \in \{n_r+1,\ldots,n-1\}} \bar{s}_i \ , \ \max_{\substack{i,j \in \{1,\ldots,n_r\} \\ i \neq j}} \bar{s}_i + \bar{\bar{s}}_{ij} \right). \tag{4.46}$$

This lower bound can be computed in $O(n_r^2 + n_i)$ flops. We call this method the second-order component-wise method of computing lower bound.

To understand the second-order component-wise method in geometrical terms, let us assume that a vector $\hat{\boldsymbol{w}}$ is in the hyper-ellipsoid $\|\boldsymbol{R}(\hat{\boldsymbol{w}} - \check{\boldsymbol{w}})\|_2^2 < \hat{\rho}^2$. Recall that in the component-wise method introduced in Section 4.5.2, we compute the range of each component of $\hat{\boldsymbol{w}}$ separately and get a covering box (4.38) of the hyper-ellipsoid. In the second-order component-wise method however, we also consider the effect of the value of one component on the range of another component. These inter-components effects are reflected in the computation of $\bar{\bar{s}}_{ij}$. As an example, the dashed red parallelogram in Figure 4–3 shows how the value of x_2 can affect the range of x_1 (we actually only consider the "linear effect" in our method). Since the second-order component-wise method also considers $\bar{s}_i$, it actually finds a covering polyhedron of the hyper-ellipsoid (e.g., the shadowed area in Figure 4–3). If we let $\underline{g}_{w_n}$ be the largest value of $\hat{\rho}^2$ such that this polyhedron contains no valid $\hat{\boldsymbol{w}}$ that satisfies (4.27), then we can ensure that $\boldsymbol{g}_{w_n}(\hat{\boldsymbol{w}}) \geq \underline{g}_{w_n}$ for any valid $\hat{\boldsymbol{w}}$. The setup of the example in Figure 4–3 is exactly the same as in Figure 4–1b and Figure 4–2b. However, with the second-order component-wise method, we are able to find a much tighter lower bound.

We can combine the second-order component-wise method with the preconditioned norm-wise method (4.34). Note that the right hand side of (4.34) is actually

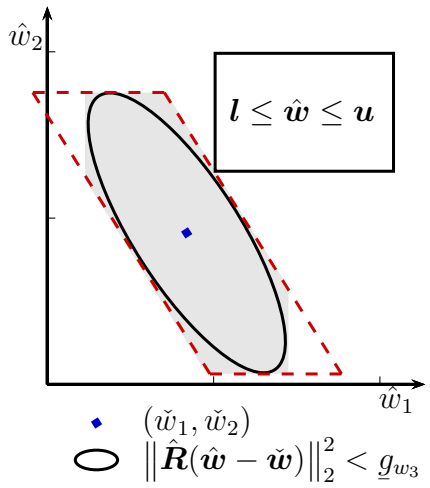


Figure 4–3: Second-order component-wise lower bound

$\left\|\tilde{\boldsymbol{F}}\right\|_2^{-2} \sum_{i=1}^{n-1} \bar{s}_i$. Combining (4.34) and (4.46) gives us

$$\underline{g}_{w_n} = \max\left(\left\|\tilde{\boldsymbol{F}}\right\|_2^{-2} \sum_{i=1}^{n-1} \bar{s}_i \ , \ \max_{i\in\{n_r+1,\ldots,n-1\}} \bar{s}_i \ , \ \max_{\substack{i,j\in\{1,\ldots,n_r\} \\ i\neq j}} \bar{s}_i + \bar{\bar{s}}_{ij} \right). \tag{4.47}$$

We refer to this method as the combined method of computing lower bound. Theoretically, (4.47) enables us to cut more branches than (4.46) does. However, as we see form the numerical tests results, the overhead introduced by (4.47) offsets the benefits. When compared with (4.46), and the overall efficiency of sphere decoding is usually not improved when we use (4.47), see Section (4.6).

4.6 Numerical Experiments

In this section, we use numerical experiments to show the effectiveness of the proposed algorithms in solving MILSBR problems (4.1). The algorithms to be tested in this section are implemented in C using the Matlab executable (MEX) library. In the tests, we compare our solvers with the IBM ILOG CPLEX Optimization Studio

(version 12.6) which is probably the most popular commercial optimization software package [1] . All tests were performed on a PC with 3.30GHz quadcore CPU and 4GB memory running Ubuntu 12.04 (Linux 3.2.0).

In the numerical experiments, we construct the data according to the following linear model

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{z} \end{bmatrix} + \boldsymbol{v}, \quad \boldsymbol{x} \in \mathbb{R}^{n_r}, \quad \boldsymbol{z} \in \mathbb{Z}^{n_i}, \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \tag{4.48}$$

where $\boldsymbol{v} \in \mathbb{R}^m$ is the Gaussian noise vector following the distribution $\mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$. To generate instances of (4.48), we let $\boldsymbol{l} = \text{randn}(n_r, 1)$, where $\text{randn}(m, n)$ is the Matlab function to generate an $m \times n$ matrix whose entries are drawn from the standard normal distribution, $\boldsymbol{u} = \boldsymbol{l} + d\boldsymbol{1}_n$, where $d \in \mathbb{R}^+$ is the width of the interval constraints, $\boldsymbol{x} = (x_i)_{n_r}$ where $x_i \in \mathbb{R}$ follows the i.i.d. uniform distribution over the range $[l_i, u_i]$, $\boldsymbol{z} = \text{round}(10 * \text{randn}(n_i, 1))$, and $\boldsymbol{v} = \sigma * \text{randn}(n, 1)$. Matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are generated in the following ways:

Case 1: $\boldsymbol{A} = \text{randn}(m, n_r)$, $\boldsymbol{B} = \text{randn}(m, n_i)$. Matrices $\boldsymbol{A}$, $\boldsymbol{B}$ generated in this way are generally well conditioned.

Case 2: $\boldsymbol{A} = \text{rand}(m, n_r)$, $\boldsymbol{B} = \text{rand}(m, n_i)$, where $\text{rand}(m, n)$ is a Matlab function to generate an $m \times n$ matrix whose entries follow the i.i.d. uniform distribution

[1] CPLEX solves the MILSBR problems based on the branch and bound methods [58] and incorporates many advanced optimization algorithms to improve the solving efficiency.

on the interval $(0, 1)$. Matrices $\boldsymbol{A}$, $\boldsymbol{B}$ generated in this way are generally more ill conditioned than the ones generated in Case 1.

In our simulations, we explicitly specify n and n_r, and take $m = n$ and $n_i = n - n_r$.

We compared CPLEX and five different sphere decoders in the tests. The sphere decoders are all based on Shenorr-Euchner's sphere decoding strategy but use different methods to find lower bounds as shown below

none: using no lower bound, i.e., $\underline{g}_{w_n} = 0$;

norm: the improved norm-wise method (4.33);

component: the regular component-wise method (4.37);

component-2: the second-order component-wise method (4.46);

combined: the combined method (4.47).

In the numerical tests, we always use LLL to reduce the original MILSRB problems first (see Section 4.2). When the sphere decoders are used, we use Algorithm 4–2 to find an initial search radius ρ.

Figure 4–4, 4–5 and 4–6 show the average runtime over 200 runs (including the time for reduction and finding an initial ρ, when applicable) of the sphere decoders and CPLEX for Case 1. We can see that, the sphere decoder uses no lower bound runs faster than CPLEX only when the dimension n is small and the standard deviation σ of the noise is small. However, when lower bounds are used in sphere decoding, the sphere decoders become much more efficient and outperform CPLEX in all of the tests in Case 1. From the figures, we can see that the sphere decoders based on the component-wise lower bounds or the second-order component-wise lower bounds

are usually faster than the ones based on the norm-wise lower bounds or the combined lower bounds. This is mostly because the norm-wise method and the combined method need to precompute n_i matrix norms $\left\|\tilde{\boldsymbol{F}}\right\|_2$ (see (4.34) and (4.47)) for subproblems of dimension k for $k = n-1, n-2, \ldots, n_r$. Computing the matrices norms introduces a big overhead. However, the sphere decoder based on the combined lower bounds can outperform the one based on the component-wise lower bounds when the noise is large. This is because the combined method can find tighter lower bounds than the component-wise method does, and thus can cut more branches from the search tree. Overall, the sphere decoder based on the second-order component-wise lower bounds gives the best performance.

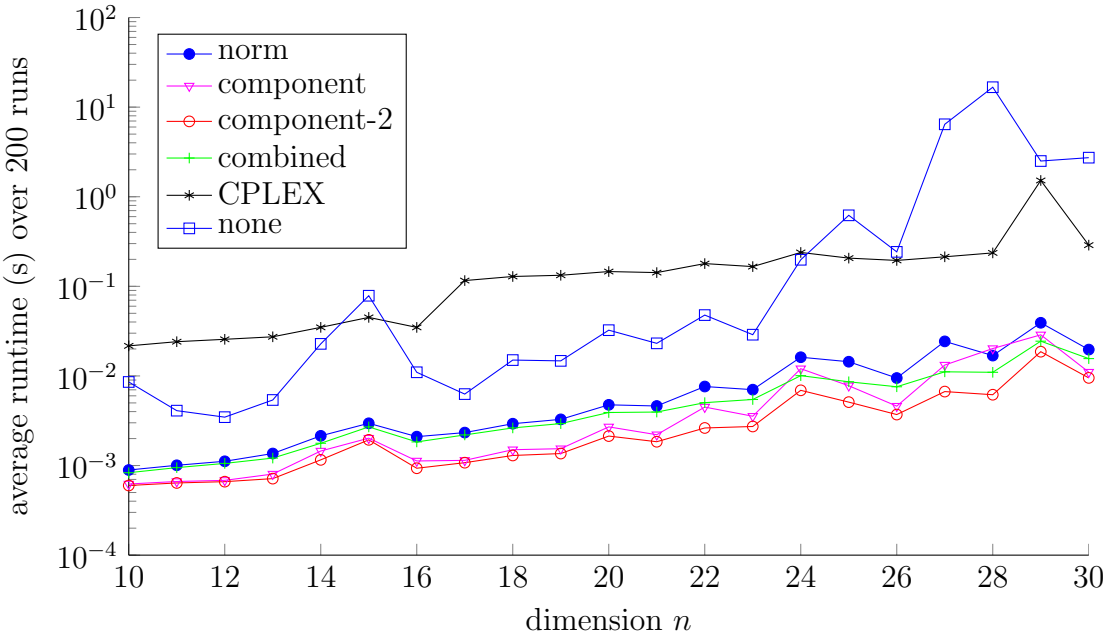


Figure 4–4: Case 1, $n_r = 3$, $\sigma = 0.75$, $d = 0.25$

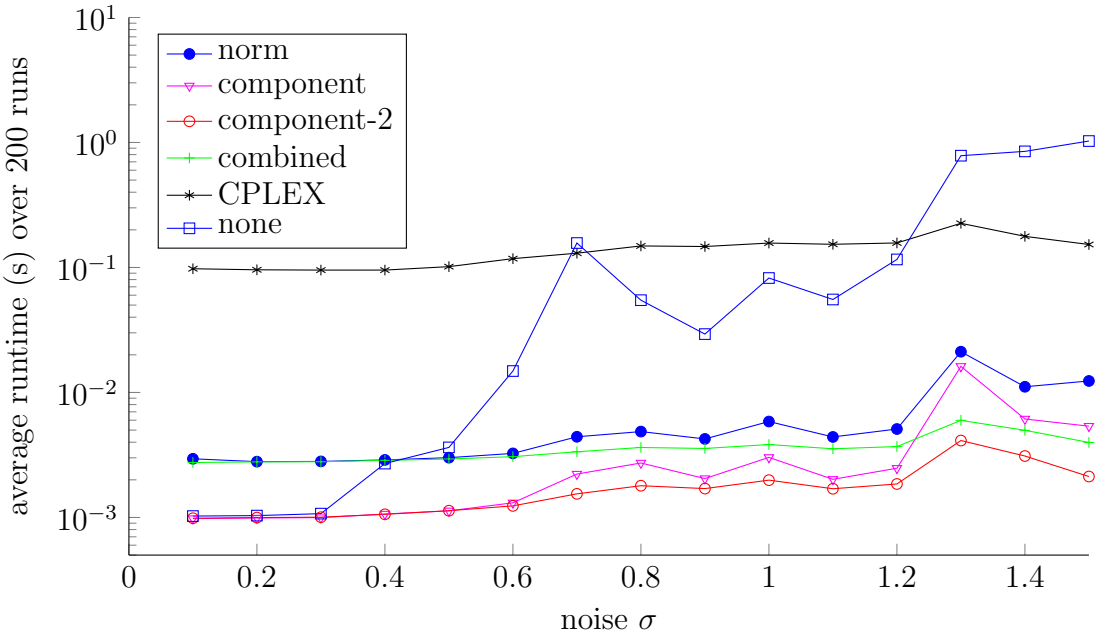


Figure 4–5: Case 1, $n = 20$, $n_r = 3$, $d = 0.25$

Figure 4–7, 4–8 and 4–9 show the average runtime over 200 runs of the sphere decoders and CPLEX in Case 2. In this case, we did not test the sphere decoder which uses no lower bound because this decoder is extremely inefficient in Case 2. In the figures, we can still see that the sphere decoder based on the second-order component-wise lower bounds is more efficient than the other sphere decoders and CPLEX. And the sphere decoder based on the combined lower bounds is faster than the method based on the component-wise lower bounds, when the noise is not too small. In Case 2, the sphere decoder based on the combined method can be much faster than the decoder based on the component-wise and norm-wise lower bounds.

In both Case 1 and Case 2, we saw in our numerical tests that even though the combined method can cut more branches in the search tree than the second-order

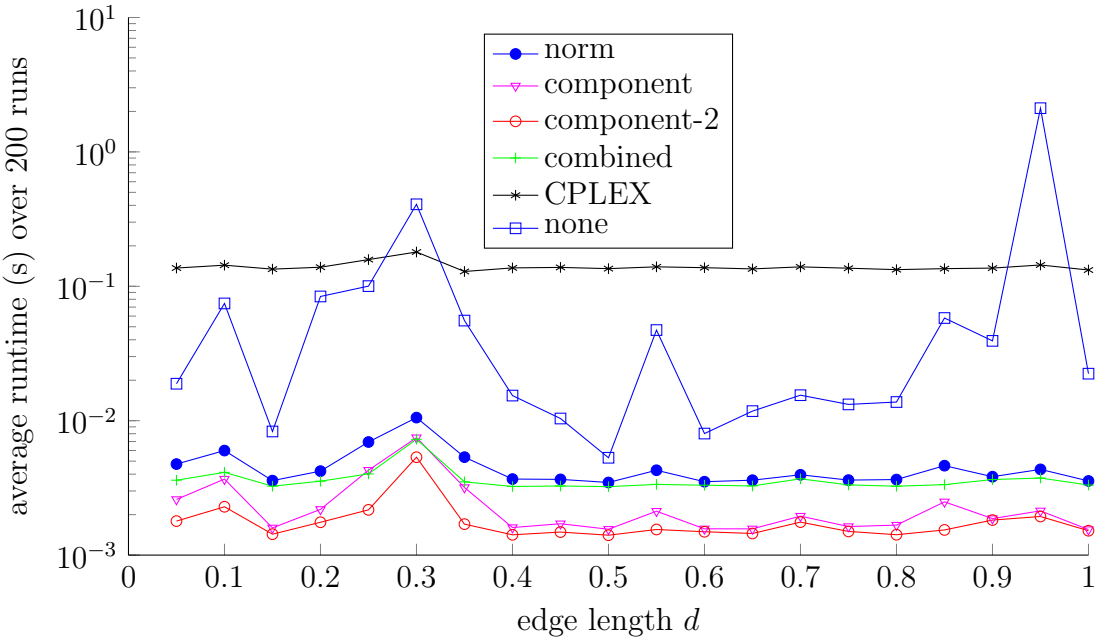


Figure 4–6: Case 1, $n = 20$, $n_r = 3$, $\sigma = 0.75$

component-wise method does, the former is usually slower in practice because of the overhead it introduces. Nevertheless, they are both much faster than CPLEX.

As we stated above, we always use Algorithm 4–2 to compute an initial radius ρ for the sphere decoders. Next, we show how Algorithm 4–2 helps in improving the efficiency of the sphere decoders. Specifically, we focus on the sphere decoder that uses the second-order component-wise method to compute the lower bounds.

In Figure 4–10a, we show the initial search radius found by the straightforward Babai's algorithm (marked as "Babai" in the figure) and by Algorithm 4–2 (marked as "New" in the figure). We can see that, in both Case 1 and Case 2, Algorithm 4–2 can find a much smaller initial search radius than the straightforward Babai's algorithms does. This is because that Algorithm 4–2 is able to use the information of

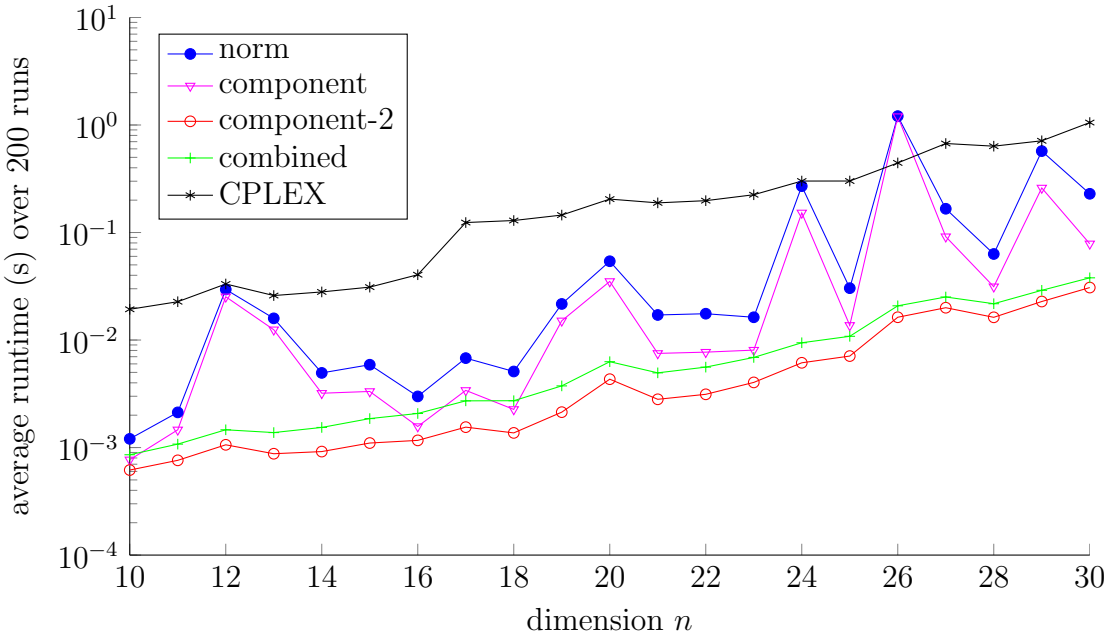


Figure 4–7: Case 2, $n_r = 3$, $\sigma = 0.5$, $d = 1$

the box constraint. We know that when computing the lower bounds in the process of sphere decoding, especially the second-order component-wise lower bounds, the information in the box constraint is also used. Next, we show that even when the lower bounds are used, Algorithm 4–2 is still useful in improving the efficiency of sphere decoding. In Figure 4–10b, we show the runtime of the sphere decoder (using the second-order component-wise lower bounds) where the initial search radius is computed by the straightforward Babai's algorithm or by Algorithm 4–2. We can see that the solving time is decreased when Algorithm 4–2 is used to compute the initial search radius.

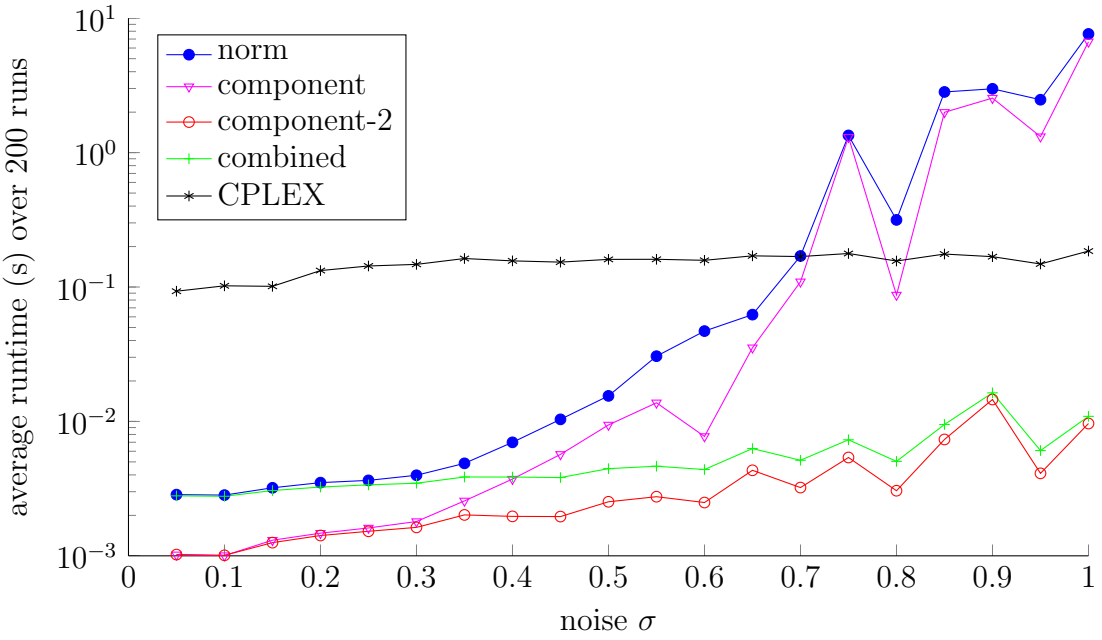


Figure 4–8: Case 2, $n = 20$, $n_r = 3$, $d = 1$

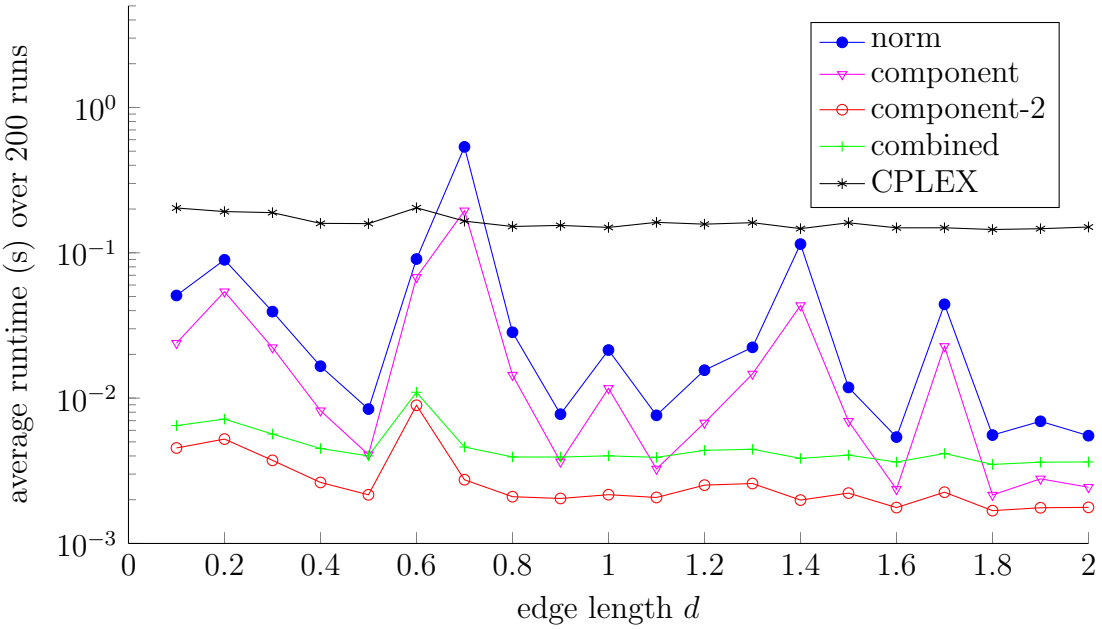
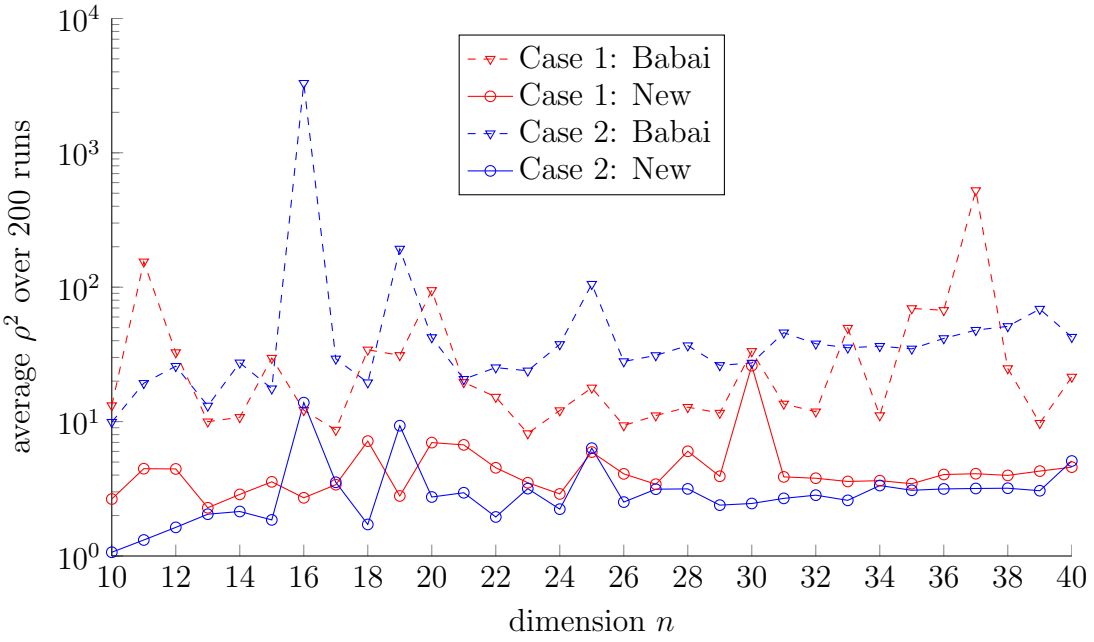
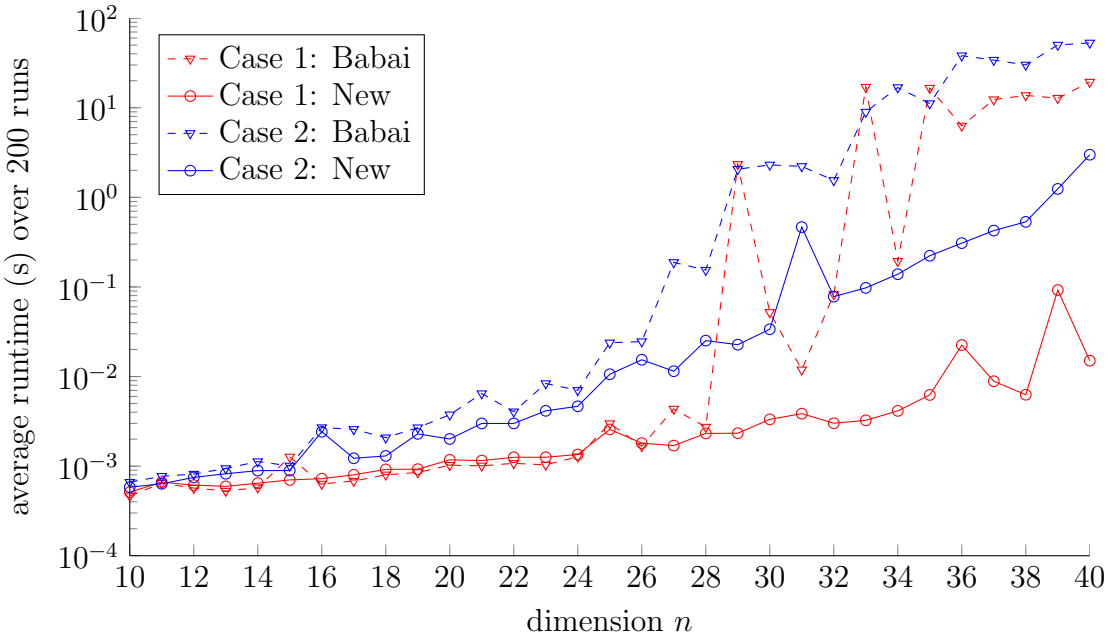


Figure 4–9: Case 2, $n = 20$, $n_r = 3$, $\sigma = 0.5$



(a) Initial search radius ρ



(b) Effects of initial ρ on sphere decoding using component-2 lower bounds

Figure 4–10: $n_r = 3$, $\sigma = 0.5$, $d = 1$

CHAPTER 5
Conclusion and Future Work

This thesis covered a range of theoretical results and algorithms for various ILS problems. In Chapter 1, we first gave an extensive background introduction to the ILS problems and their applications in practice. Then we introduced the concept of lattice reduction, which is commonly applied in the solving process of ILS problems. To give a state-of-art overview, various lattice reductions and approaches for solving ILS problems were reviewed and their advantages and disadvantages were commented.

In Chapter 2, we looked at the cost of the sphere decoding approach for solving the OILS problems. In this chapter, we discussed how the cost of the sphere decoding algorithms change in the process of the LLL reduction [60] and showed that this cost is reduced after the LLL reduction. We also discussed the effect of the parameter δ of the LLL reduction on the efficiency of sphere decoding. Examples and results of numerical tests were given to show that though increasing the value of δ generally improves the efficiency of sphere decoding for solving the OILS problems, this conclusion does not always hold.

There are some interesting questions we want to answer in future research. First, we assumed that the model matrix is deterministic. If the matrix is random following some probability distribution, what is the expected cost of sphere decoding? And how does the LLL reduction affect it? Second, the KZ reduction can usually make sphere

decoding faster than LLL does. Can some implementations of the KZ reduction always decrease the cost of sphere decoding, and decrease it more than the LLL reduction algorithm does?

In Chapter 3, the BILS problems were studied. First, we introduced the concept of inactive set of constraints for the BILS problems. Based on this concept, we showed that IGTs can be applied in the reduction of the BILS problems as long as certain conditions are satisfied, contrary to the common belief. After that, we proposed three algorithms to compute an initial search radius ρ for a given BILS problem, and two algorithms, ISF-1 and ISF-2, to find an inactive set based on the value of ρ. Based on these algorithms, we proposed the RLLL reduction algorithm which incorporates IGTs and applies the restricted LLL lattice reduction to the BILS problems. Numerical experiments were given to show that the RLLL based reductions can significantly improve the efficiency of the sphere decoding approach for solving BILS problems and the quality of some approximate solutions.

Here we want to emphasize that the RLLL algorithm is only one implementation of the QRZ factorization. In the reduction of the BILS problem, in additional to the RLLL algorithm, we also used the AIP reduction algorithm separately. It would be more ideal if the two reduction algorithms could be combined more naturally. More work is required to get a deeper understanding on how the factor $\boldsymbol{Z}$ in the QRZ factorization affects the efficiency of sphere decoding of the BILS problems. With a deeper understanding, more effective reduction algorithms could be designed for the BILS problems.

In Chapter 4, we studied the MILSBR problems. We proposed to use the LLL reduction and the sphere decoding approach to solve the MILSBR problems. To improve the efficiency of sphere decoding, we then proposed a method to find an initial search radius for a given MILSBR problem using the information in the box constraint. Compared to the straightforward Babai's nearest plane algorithm, this new method is able to find a much smaller initial search radius. It is known that the efficiency of sphere decoding can be improved using the lower bounds of the search radius of the sub sphere decoding processes. We proposed three efficient methods to compute lower bounds for the sphere decoding process: the norm-wise method, the component-wise method and the second-order component-wise method. The first two of them are adopted from the methods that compute the lower bounds of search radius for OILS and BILS problems and the third method is specifically for BILSBR. As shown in the numerical experiments, these methods improve the efficiency of sphere decoding significantly. When using the second-order component wise method to compute the lower bounds, the sphere decoding method is much faster than CPLEX.

Recall that when solving the OILS problems, Schnorr-Euchner's sphere decoding algorithm enumerates $\bar{z}_k$ in a specific order, trying to find good solution candidates early. However, this order is not the best for solving the MILSBR problems because it does not take the box constraint into account. This can lead to bad solution candidates being found early in the enumeration and result in poor efficiency for sphere decoding. It would be an interesting topic to find out if we can use the

information in the box constraint to guide the enumeration order of $\bar{z}_k$ and improve the efficiency of sphere decoding.

One practical application of the MILSBR problems is the GPS positioning where the position of the receiver is in a known range. In the future, we would like to apply the ILS approach to these GPS positioning problems and compare it with the Bayesian approach given in [111].

References

[1] W. Abediseid. *Efficient Lattice Decoders for the Linear Gaussian Vector Channel: Performance & Complexity Analysis*. PhD thesis, Department of Electrical and Computer Engineering, University of Waterloo, 2011.

[2] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, 2002.

[3] M. Ajtai. The shortest vector problem in L2 is NP-hard for randomized reductions. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 10–19, 1998.

[4] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 284–293, 1997.

[5] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 601–610, 2001.

[6] M. Ajtai, R. Kumar, and D. Sivakumar. Sampling short lattice vectors and the closest lattice vector problem. In *Proceedings of the 17th IEEE Annual Conference on Computational Complexity*, pages 41–45, 2002.

[7] M. Al Borno, X.-W. Chang, and X. Xie. On "decorrelation" in solving integer least-squares problems for ambiguity determination. *Survey Review*, 46(334):37–49, 2014.

[8] M. F. Anjos, X.-W. Chang, and W.-Y. Ku. Lattice preconditioning for the real relaxation branch-and-bound approach for integer least squares problems. *Journal of Global Optimization*, 59(1):227–242, 2014.

[9] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. In *Proceedings of*

34th Annual Symposium on Foundations of Computer Science, pages 724–733, 1993.

[10] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[11] A. Banihashemi and A. Khandani. On the complexity of decoding lattices using the Korkin-Zolotarev reduced basis. *IEEE Transactions on Information Theory*, 44(1):162–171, 1998.

[12] D. P. Bertsekas. On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Transactions on Automatic Control*, 21(2):174–184, 1976.

[13] D. P. Bertsekas. Projected Newton methods for optimization problems with simple constraints. *SIAM Journal on Control and Optimization*, 20(2):221–246, 1982.

[14] D. Bienstock. Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74(2):121–140, 1996.

[15] S. C. Billups, S. P. Dirkse, and M. C. Ferris. A comparison of large scale mixed complementarity problem solvers. *Computational Optimization and Applications*, 7(1):3–25, 1997.

[16] G. Blewitt. Basics of the GPS technique: observation equations. *Geodetic Applications of GPS*, pages 10–54, 1997.

[17] J. Blömer and S. Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. In L. Arge, C. Cachin, T. Jurdziński, and A. Tarlecki, editors, *Automata, Languages and Programming*, volume 4596 of *Lecture Notes in Computer Science*, pages 65–77. Springer Berlin Heidelberg, 2007.

[18] J. Boutros, N. Gresset, L. Brunel, and M. Fossorier. Soft-input soft-output lattice sphere decoder for linear channels. In *Proceedings of the IEEE Global Telecommunications Conference*, pages 1583–1587, 2003.

[19] S. Breen. Integer least squares search and reduction strategies. Master's thesis, McGill University, 2011.

[20] S. Breen and X.-W. Chang. Column reordering for box-constrained integer least squares problems. In *Proceedings of the IEEE Global Telecommunications Conference*, number of pages 6, 2011.

[21] C. Buchheim, A. Caprara, and A. Lodi. An effective branch-and-bound algorithm for convex quadratic integer programming. *Mathematical Programming*, 135(1-2):369–395, 2012.

[22] C. Buchheim, M. De Santis, L. Palagi, and M. Piacentini. An exact algorithm for nonconvex quadratic integer minimization using ellipsoidal relaxations. *SIAM Journal on Optimization*, 23(3):1867–1889, 2013.

[23] X.-W. Chang and G. Golub. Solving ellipsoid-constrained integer least squares problems. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1071–1089, 2009.

[24] X.-W. Chang and Q. Han. Solving box-constrained integer least squares problems. *IEEE Transactions on Wireless Communications*, 7(1):277–287, 2008.

[25] X.-W. Chang, J. Wen, and X. Xie. Effects of the LLL reduction on the success probability of the Babai point and on the complexity of sphere decoding. *IEEE Transactions on Information Theory*, 59(8):4915–4926, 2013.

[26] X.-W. Chang and X. Xie. A discrete enumeration approach for mixed integer least squares problems with box constraints on real variables, in preparation.

[27] X.-W. Chang and X. Xie. Restricted LLL lattice reduction and its application in solving box-constrained integer least squares problems, in preparation.

[28] X.-W. Chang, X. Yang, and T. Zhou. MLAMBDA: A modified LAMBDA method for integer least-squares estimation. *Journal of Geodesy*, 79(9):552–565, 2005.

[29] P. Crescenzi and V. Kann. Approximation on the web: A compendium of NP optimization problems. In J. Rolim, editor, *Randomization and Approximation Techniques in Computer Science*, volume 1269 of *Lecture Notes in Computer Science*, pages 111–118. Springer Berlin Heidelberg, 1997.

[30] M.-O. Damen, H. El-Gamal, and G. Caire. On maximum-likelihood detection and the search for the closest lattice point. *IEEE Transactions on Information Theory*, 49(10):2389–2402, 2003.

[31] P. De Jonge and C. Tiberius. LAMBDA method for integer ambiguity estimation: implementation aspects. In *Delft Geodetic Computing Center LGR-Series, No.12*, 1996.

[32] J. Detrey, G. Hanrot, X. Pujol, and D. Stehlé. Accelerating lattice reduction with FPGAs. In M. Abdalla and P. S. Barreto, editors, *Progress in Cryptology – LATINCRYPT*, volume 6212 of *Lecture Notes in Computer Science*, pages 124–143. Springer Berlin Heidelberg, 2010.

[33] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[34] F. Eisenbrand. Integer programming and algorithmic geometry of numbers. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 505–559. Springer Berlin Heidelberg, 2010.

[35] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–471, 1985.

[36] G. D. Forney. On the duality of coding and quantizing. In *Coding and Quantizatiion: DIMACS*, volume 14 of *Discr. Math. Theory Comp. Sci.* AMS Bookstore, 1993.

[37] G. Foschini, G. Golden, R. Valenzuela, and P. Wolniansky. Simplified processing for high spectral efficiency wireless communication employing multi-element arrays. *IEEE Journal on Selected Areas in Communications*, 17(11):1841–1852, 1999.

[38] Y. H. Gan, C. Ling, and W.-H. Mow. Complex lattice reduction algorithm for low-complexity full-diversity MIMO detection. *IEEE Transactions on Signal Processing*, 57(7):2701–2710, 2009.

[39] V. M. Garcia, S. Roger, R. A. Trujillo, A. M. Vidal, and A. Gonzalez. A deterministic lower bound for the radius in sphere decoding search. In *Proceedings of International Conference on Advanced Technologies for Communications*, pages 11–16, 2010.

[40] D. Gesbert and J. Akhtar. Transmitting over ill-conditioned MIMO channels: From spatial to constellation multiplexing. In T. Kaiser, editor, *Smart Antennas in Europe – State-of-the-Art*. EURASIP Publishing, 2005.

[41] G. H. Golub and C. F. Van Loan. *Matrix Computations*, 4th edition. The Johns Hopkins University Press, Baltimore, Maryland, 2013.

[42] G. Hanrot, X. Pujol, and D. Stehlé. Algorithms for the shortest and closest lattice vector problems. In Y. Chee, Z. Guo, S. Ling, F. Shao, Y. Tang, H. Wang, and C. Xing, editors, *Coding and Cryptology*, volume 6639 of *Lecture Notes in Computer Science*, pages 159–190. Springer Berlin Heidelberg, 2011.

[43] G. Hanrot and D. Stehlé. Improved analysis of Kannan's shortest lattice vector algorithm. In *Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*, pages 170–186. Springer-Verlag, 2007.

[44] A. Hassibi and S. Boyd. Integer parameter estimation in linear models with applications to GPS. *IEEE Transactions on Signal Processing*, 46(11):2938–2952, 1998.

[45] B. Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. *Theoretical Computer Science*, 41:125–139, 1985.

[46] J. Hermans, M. Schneider, J. Buchmann, F. Vercauteren, and B. Preneel. Parallel shortest lattice vector enumeration on graphics cards. In D. J. Bernstein and T. Lange, editors, *Progress in Cryptology – AFRICACRYPT*, volume 6055 of *Lecture Notes in Computer Science*, pages 52–68. Springer Berlin Heidelberg, 2010.

[47] C. Hermite. Extraits de lettres de M. Hermitea M. Jacobi sur différents objets de la théorie des nombres, deuxième lettre. *J. reine angew. Math*, 40:279–290, 1850.

[48] J. Jalden and B. Ottersten. On the complexity of sphere decoding in digital communications. *IEEE Transactions on Signal Processing*, 53(4):1474–1484, 2005.

[49] J. Jalden, D. Seethaler, and G. Matz. Worst- and average-case complexity of LLL lattice reduction in MIMO wireless systems. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2685–2688, 2008.

[50] A. Kamath and N. Karmarkar. A continuous method for computing bounds in integer quadratic optimization problems. *Journal of Global Optimization*, 2(3):229–241, 1992.

[51] R. Kannan. Improved algorithms for integer programming and related lattice problems. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 193–206, 1983.

[52] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12:415–550, 1987.

[53] J. P. Kermoal, L. Schumacher, P. E. Mogensen, and K. I. Pedersen. Experimental investigation of correlation properties of MIMO radio channels for indoor picocell scenarios. In *Proceedings of the 52nd IEEE Vehicular Technology Conference*, pages 14–21, 2000.

[54] D. Kim and R. B. Langley. GPS ambiguity resolution and validation: methodologies, trends and issues. In *Proceedings of the 7th GNSS Workshop–International Symposium on GPS/GNSS*, volume 30, 2000.

[55] M. Kisialiou and Z.-Q. Luo. Performance analysis of quasi-maximum-likelihood detector based on semi-definite programming. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 433–436, 2005.

[56] A. Korkine and G. Zolotareff. Sur les formes quadratiques. *Mathematische Annalen*, 6(3):366–389, 1873.

[57] W.-Y. Ku. Lattice preconditioning for the real relaxation based branch and bound method for integer least squares problems. Master's thesis, School of Computer Science, McGill University, 2011.

[58] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, 28(3):497–520, 1960.

[59] E. G. Larsson, O. Edfors, F. Tufvesson, and T. L. Marzetta. Massive MIMO for next generation wireless systems. *IEEE Communications Magazine*, 52(2):186–195, 2014.

[60] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[61] H. Lenstra. Lattices. In J. P. Buhler and P. Stevenhagen, editors, *Algorithmic Number Theory*, MSRI Publications, pages 127–181. Cambridge University Press, 2011.

[62] Q. Li, G. Li, W. Lee, M.-i. Lee, D. Mazzarese, B. Clerckx, and Z. Li. Mimo techniques in wimax and lte: a feature overview. *Communications Magazine, IEEE*, 48(5):86–92, 2010.

[63] C. Ling. On the proximity factors of lattice reduction-aided decoding. *IEEE Transactions on Signal Processing*, 59(6):2795–2808, 2011.

[64] C. Ling and N. Howgrave-Graham. Effective LLL reduction for lattice decoding. In *IEEE International Symposium on Information Theory*, pages 196–200, 2007.

[65] C. Ling, W.-H. Mow, K. H. Li, and A. C. Kot. Multiple-antenna differential lattice decoding. *IEEE Journal on Selected Areas in Communications*, 23(9):1821–1829, Sept 2005.

[66] W.-K. Ma, T. Davidson, K. M. Wong, Z.-Q. Luo, and P.-C. Ching. Quasi-maximum-likelihood multiuser detection using semi-definite relaxation with application to synchronous CDMA. *IEEE Transactions on Signal Processing*, 50(4):912–922, 2002.

[67] J. Maurer, J. Jaldén, D. Seethaler, and G. Matz. Achieving a continuous diversity-complexity tradeoff in wireless MIMO systems via pre-equalized sphere-decoding. *Selected Topics in Signal Processing, IEEE Journal of*, 3(6):986–999, 2009.

[68] R. D. McBride and J. S. Yormark. An implicit enumeration algorithm for quadratic integer programming. *Management Science*, 26(3):282–296, 1980.

[69] K. D. McDonald. The modernization of GPS: plans, new capabilities and the future relationship to Galileo. *Journal of Global Positioning Systems*, 1(1):1–17, 2002.

[70] D. Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, 2001.

[71] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems: A Cryptographic Perspective*, volume 671. Springer, 2002.

[72] D. Micciancio and O. Regev. Lattice-based cryptography. In D. J. Bernstein, J. Buchmann, and E. Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191. Springer Berlin Heidelberg, 2009.

[73] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 351–358, 2010.

[74] H. Minkowski. *Geometrie der Zahlen*. Teubner-Verlag, Leipzig, 1896.

[75] J. J. Moré and G. Toraldo. Algorithms for bound constrained quadratic programming problems. *Numerische Mathematik*, 55(4):377–400, 1989.

[76] I. Morel, D. Stehlé, and G. Villard. H-LLL: using householder inside LLL. In *Proceedings of the 34th International Symposium on Symbolic and Algebraic Computation*, pages 271–278, 2009.

[77] W. Mow. Universal lattice decoding: principle and recent advances. *Wireless Communications and Mobile Computing*, 3(5):553–569, 2003.

[78] W.-H. Mow. Maximum likelihood sequence estimation from the lattice viewpoint. *IEEE Transactions on Information Theory*, 40(5):1591–1600, 1994.

[79] P. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009.

[80] P. Q. Nguyen and D. Stehlé. LLL on the average. In F. Hess, S. Pauli, and M. Pohst, editors, *Algorithmic Number Theory*, volume 4076 of *Lecture Notes in Computer Science*, pages 238–256. Springer Berlin Heidelberg, 2006.

[81] J. Nocedal and S. J. Wright. *Numerical Optimization*, 2nd edition. Springer Series in Operations Research and Financial Engineering. Springer New York, 2006.

[82] A. Novocin, D. Stehlé, and G. Villard. An LLL-reduction algorithm with quasi-linear time complexity. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 403–412, 2011.

[83] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. *Cryptology and Computational Number Theory*, 42:75–88, 1990.

[84] J. M. W. P. M. Gruber, editor. *Handbook of Convex Geometry*. North-Holland, Amsterdam, 1993.

[85] M. Pohst. On the computation of lattice vectors of minimal length, successive minima and reduced bases with applications. *ACM Sigsam Bulletin*, 15(1):37–44, 1981.

[86] B. Remondi. Global positioning system carrier phase: Description and use. *Bulletin Géodésique*, 59(4):361–377, 1985.

[87] F. Rusek, D. Persson, B. K. Lau, E. Larsson, T. Marzetta, O. Edfors, and F. Tufvesson. Scaling up MIMO: Opportunities and challenges with very large arrays. *Signal Processing Magazine, IEEE*, 30(1):40–60, Jan 2013.

[88] S. Schaer, G. Beutler, L. Mervart, M. Rothacher, and U. Wild. Global and regional ionospheric models using the GPS double difference phase observable. In *Proceedings of the IGS Workshop on Special Topics and New Directions*, pages 77–92, 1995.

[89] C. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1):181–199, 1994.

[90] C. P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53:201–224, 1987.

[91] C. P. Schnorr. A more efficient algorithm for lattice basis reduction. *Journal of Algorithms*, 9:47–62, 1988.

[92] C. P. Schnorr and H. H. Hörner. Attacking the Chor-Rivest cryptosystem by improved lattice reduction. In *Proceedings of Advances in Cryptology, International Conference on the Theory and Application of Cryptographic Techniques*, volume 921 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1995.

[93] D. Seethaler, J. Jalden, C. Studer, and H. Bolcskei. On the complexity distribution of sphere decoding. *IEEE Transactions on Information Theory*, 57(9):5754–5768, 2011.

[94] D.-S. Shiu, G. J. Foschini, M. J. Gans, and J. M. Kahn. Fading correlation and its effect on the capacity of multielement antenna systems. *IEEE Transactions on Communications*, 48(3):502–513, 2000.

[95] N. J. Sloane and J. Conway. *Sphere Packings, Lattices and Groups*. Springer, 1999.

[96] D. Stehlé. Floating-point LLL: Theoretical and practical aspects. In P. Q. Nguyen and B. Vallée, editors, *The LLL Algorithm*, Information Security and Cryptography, pages 179–213. Springer Berlin Heidelberg, 2010.

[97] M. Stojnic, H. Vikalo, and B. Hassibi. Speeding up the sphere decoder with h^{∞} and SDP inspired lower bounds. *IEEE Transactions on Signal Processing*, 56(2):712–726, 2008.

[98] K. Su and I. Wassell. A new ordering for efficient sphere decoding. In *Proceedings of IEEE International Conference on Communications*, volume 3, pages 1906–1910, 2005.

[99] P. H. Tan and L. Rasmussen. The application of semidefinite programming for detection in CDMA. *IEEE Journal on Selected Areas in Communications*, 19(8):1442–1449, 2001.

[100] P. Teunissen. Integer least-squares theory for the GNSS compass. *Journal of Geodesy*, 84(7):433–447, 2010.

[101] P. J. Teunissen. The Lambda method for the GNSS compass. *Artificial Satellites*, 41(3):89–103, 2006.

[102] U.S. Naval Observatory. Current GPS constellation. `http://tycho.usno.navy.mil/gpscurr.html`. Accessed: 2014-11-1.

[103] P. van Emde Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Technical Report 81-04, Mathematics Department, University of Amsterdam, Amsterdam, The Netherlands, 1981.

[104] A. Van Zelst and J. S. Hammerschmidt. A single coefficient spatial correlation model for multiple-input multiple-output (MIMO) radio channels. In *Proceedings of URSI General Assembly*, pages 17–24, 2002.

[105] E. Viterbo and E. Biglieri. A universal algorithm for decoding lattice codes. In *Proceedings of the 14th Symposium on Signal Processing and Images*. ICIP, Study Group of Signal Processing and Images, 1993.

[106] E. Viterbo and J. Boutros. A universal lattice code decoder for fading channels. *IEEE Transactions on Information Theory*, 45(5):1639–1642, 1999.

[107] G. Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les paralléllοèdres

primitifs. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.

[108] D. Wubben, R. Bohnke, J. Rinas, V. Kuhn, and K. Kammeyer. Efficient algorithm for decoding layered space-time codes. *Electronics Letters*, 37(22):1348–1350, 2001.

[109] X. Xie, X.-W. Chang, and M. Al Borno. Partial LLL reduction. In *Proceedings of the IEEE Global Telecommunications Conference*, number of pages 5, 2011.

[110] G. Xu. GPS observables. In *GPS: Theory, Algorithms, and Applications*, pages 37–42. Springer Berlin Heidelberg, 2007.

[111] J.-J. Zhu, R. Santerre, and X.-W. Chang. A Bayesian method for linear, inequality-constrained adjustment and its application to GPS positioning. *Journal of Geodesy*, 78(9):528–534, 2005.