An ADMM Method for Underdetermined Box-constrained Integer Least Squares Problems

Tianchi Ma

School of Computer Science

McGill University, Montreal

May, 2021

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Science

Abstract

Integer least squares (ILS) is an important class of optimization problems that may arise from estimating the integer parameter vector in a linear model with additive Gaussian noise in applications such as communications, control and global navigation satellite systems. This thesis is concerned with solving the underdetermined box constrained ILS (UBILS) problems. We propose a modified alternating direction method of multipliers (ADMM) algorithm as a heuristic approach. Simulation results show that such heuristic is much superior to original ADMM on UBILS problems. Based on the existing direct tree search (DTS) algorithm, we show how to incorporate ADMM methods for computing initial points and lower bounds with the aim of improving the efficiency. Numerical results indicate that in most cases our new approach is better than DTS and selected commercial solvers in terms of efficiency and accuracy. Besides, the advantage of our new approach over DTS becomes more significant when the search region becomes larger.

Abrégé

Les moindres carrés en nombres entiers (ILS) sont une classe importante de problèmes d'optimisation, qui peuvent résulter de l'estimation du vecteur de paramètre entier dans une modèle linéaire avec bruit gaussien additif dans des applications comme les communications, le contrôle et les systèmes mondiaux de navigation par satellite. Cette thèse s'intéresse à la résolution des problèmes d'ILS sous-déterminés avec les contraintes de boites (UBILS). Nous proposons une méthode modifiée de direction alternative des multiplicateurs (ADMM) comme une approche heuristique. Les résultats de la simulation montrent que de telles heuristiques sont bien supérieures à ADMM original sur les problèmes UBILS. Sur la base de l'algorithme de recherche directe dans l'arbre (DTS), nous suggérons d'incorporer des techniques ADMM pour calculer les points initiaux et les bornes inférieures dans le but d'améliorer l'efficacité. Les résultats numériques indiquent que dans la plupart des cas, notre approche de recherche est meilleur que DTS et certains solveurs commerciaux en termes d'efficacité et de précision. L'avantage de notre nouvelle approche par rapport au DTS devient plus significatif lorsque la zone de recherche devient plus grande.

Acknowledgements

I would like to first thank my supervisor Prof. Xiao-Wen Chang for his guidance, suggestion and support throughout my master studies at McGill. His expertise helps me formulate the research questions and methodology. His timely and insightful feedback can always inspire me. In addition, I would like to thank all my friends for bringing me fun, help and inspirations. Lastly, I wish to express my sincerest gratitude to my family for their consistent love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Integer Least Squares Problem

Suppose that we have the following linear model:

$$\mathbf{y} = \mathbf{A}\mathbf{x}^* + \mathbf{v}, \quad \mathbf{v} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}) \tag{1.1}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a real matrix with full column rank, $\mathbf{y} \in \mathbb{R}^m$ is an observation vector, $\mathbf{x}^* \in \mathbb{Z}^n$ is an integer parameter vector and may be subject to some constraints, $\mathbf{v} \in \mathbb{R}^n$ is a noise vector following the normal distribution $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. The goal is to recover $\mathbf{x}^*$ based on given $\mathbf{A}$ and $\mathbf{y}$. This is an estimation or detection problem and may arise from different applications such as wireless communications (see, e.g., [1, 25, 52]), GPS positioning (see, e.g., [10, 21, 41, 65]) and control (see, e.g., [22]).

One approach to estimating $\mathbf{x}^*$ in (1.1) is to solve the following integer least squares (ILS) problem:

$$\min_{\mathbf{x}\in\mathbb{Z}^n} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2. \tag{1.2}$$

It is easy to show that the solution is the maximum likelihood estimator of $\mathbf{x}^*$ (see, e.g., [41]).

To distinguish (1.2) from other ILS problems, we refer to it as an ordinary integer least squares (OILS) problem. In the literature, $\mathbf{A}$ is referred to as a lattice generator matrix, which can generate the lattice $\mathcal{L}(\mathbf{A}) = \{\mathbf{A}\mathbf{x}, \mathbf{x} \in \mathbb{Z}^n\}$ (see e.g., [1]). The OILS problem (1.2) is equivalent to finding the closest point to $\mathbf{y}$ in the lattice $\mathcal{L}(\mathbf{A})$, so it can also be referred to as the closest point problem. The OILS problem, which does not arise from the liner model estimation, has applications in lattice cryptography (see e.g., [39, 40, 49, 50]) and number theory (see, e.g., [53]).

Different from a real least squares problem, the OILS problem has been proven to be NP-hard (see, e.g., [48, 67]). Various algorithms have been proposed to solve the OILS problem, including deterministic and stochastic ones. The deterministic algorithms include the Voronoi cell based approach (see, e.g., [51, 61]), the relaxation-based branch-and-bound (see, e.g., [4]), and the enumeration approach which enumerates lattice vectors within the search space until the optimal solution is found (see e.g., [34, 40, 43, 59]). Apart from deterministic algorithms, some randomized algorithms have also been investigated (see, e.g., [2, 3, 9, 39]).

In practice, the enumeration approach is often used, as it is often more efficient than other approaches. An enumeration algorithm typically involves two phases, reduction and search. The reduction phase is to reduce the problem (1.2) to an equivalent problem which makes the search stage more efficient. The search phase aims at finding the optimal solution of the reduced problem. For reduction, there are two well-known strategies: the Lenstra-Lenstra-Lovász (LLL) reduction [46] and the Korkine-Zolotare (KZ) reduction [45]. Due to the expensive computational cost of the KZ reduction, the LLL reduction is often used in practice for solving OILS problems. However, since the KZ reduction can make the search process more efficient than the LLL reduction, it is usually employed when one solves multiple OILS problems with the same matrix $\mathbf{A}$. The details of the LLL reduction will be given in Section 2.1.1. The second phase is to find the optimal solution by enumerating integer vectors in a region. There are two well-known search strategies: the Phost strategy (see, e.g., [34]), which enumerates the integer vectors in a hyper-ellipsoid, and the Kannan strategy (see, e.g., [43, 44]), which enumerates the integer vectors in a rectangular parallelotope. An important improvement of the Phost strategy, known as the Schnorr-Euchner (SE) strategy, enumerates the integer vectors in a different order in the hyper-ellipsoid [59]. The hyper-ellipsoid which the Phost strategy or the SE strategy enumerates integer points in is also a hyper-sphere in terms of lattice points. For this reason, these search algorithms are also referred to as sphere decoding (SD) algorithms in communications. A detailed comparison between these search strategies can be found in [1]. As the SE strategy outperforms the other two, it will be introduced in Section 2.1.2 and in this thesis we focus on the SE based search strategies.

1.2 Box-constrained Integer Least Squares Problem

In some applications, such as wireless communications (see e.g., [25, 42, 54, 58]), the parameter vector $\mathbf{x}^*$ in the linear model (1.1) is constrained to a discrete set, which can be easily transformed into an equivalent constraint box $\mathcal{B}$:

$$\mathcal{B} = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{l}, \mathbf{u} \in \mathbb{Z}^n\}. \tag{1.3}$$

Then the ILS problem (1.2) becomes:

$$\min_{\mathbf{x} \in \mathcal{B}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2. \tag{1.4}$$

We refer to (1.4) as a box-constrained integer least squares (BILS) problem. If $m \geq n$ and $\mathbf{A}$ has full column rank, the BILS problem (1.4) is called an overdetermined box-constrained integer least squares (OBILS) problem. Otherwise, when $m < n$ and $\mathbf{A}$ has full row rank, (1.4) is referred to as an underdetermined box-constrained least squares (UBILS) problem. This thesis will focus on UBILS problems.

To solve the OBILS problem through the enumeration approach, the LLL reduction cannot be applied because it would transform the box $\mathcal{B}$ to a more complex geometry and make the search phase less efficient. In [70] and [72], the Vertical Bell Laboratories Layered Space-Time (V-BLAST) reduction algorithm and the sorted QR decomposition (SQRD) reduction algorithm were suggested respectively. Both V-BLAST and SQRD use only the information of $\mathbf{A}$ to determine column reordering. In [63], Su and Wassel proposed a new column reordering algorithm that uses all information of the problem from a geometric point of view. Independently, another reduction algorithm was given from an algebraic point of view by Chang and Han in [16]. In [13], Breen and Chang showed that the reduction algorithm in [63] and [16] give the same column reordering result in theory. They combined the best part of these two algorithms and proposed a new mathematically equivalent but computationally faster reduction algorithm. This algorithm is called all information based permutation (AIP) reduction in [73] and will be introduced in Section 2.2.1. For the search phase, several Schnorr-Euchner based strategies were proposed to find the optimal solutions of the reduced OBILS problems. In [11], a direct extension of the Schnorr-Euchner search algorithm was proposed by taking the box constraint into account. A more natural and understandable search algorithm was suggested in [25]. The one proposed by Chang and Han in [16] is an improvement of the former two. In Section 2.2.2, the search algorithm in [16] will be reviewed in detail.

For the UBILS problem, the regular sphere decoding based search algorithm for OBILS problems cannot be applied directly. Damen et al. proposed the so-called generalized sphere decoding (GSD) algorithm [24]. Its main idea is to partition the vector $\mathbf{x}$ in (1.4) into two subvectors $\mathbf{x}_{(1)} \in \mathbb{Z}^m$ and $\mathbf{x}_{(2)} \in \mathbb{Z}^{n-m}$. For each candidate of $\mathbf{x}_{(2)}$, it solves a corresponding OBILS problem to find the corresponding $\mathbf{x}_{(1)}$. After all the possible $\mathbf{x}_{(2)}$ are enumerated, the combination of $\mathbf{x}_{(1)}$ and $\mathbf{x}_{(2)}$ that gives the minimal residual, is the optimal solution to the UBILS problem. In [27], Dayal and Varanasi proposed another generalized sphere decoding, which can reduce the computational cost by splitting the candidate set for $\mathbf{x}_{(2)}$ into disjoint ordered subsets. Later, Yang et al. proposed a double-layer sphere decoder (DLSD) to solve UBILS problems in [77], which is faster than the algorithm in [27]. Then in [18], Chang and Yang proposed a recursive GSD algorithm, which modifies the algorithm in [27] and incorporates a column reordering strategy in reduction, and is faster than the algorithms in [27] and [77]. More recently, in [19], Chang and Yang proposed an direct tree search (DTS) algorithm, which integrates the two search processes in [77] into one process seamlessly and presents a novel column reordering strategy to make search process more efficient, and this DTS algorithm is faster than the algorithms in [18,77]. Since this DTS algorithm is employed in this thesis, its detail will be reviewed in Section 2.3.1. All the above algorithms mainly consider how to generate a sequence of sub-ILS problems and solve them to find the optimal solution. In [20], Chang et al. proposed a partial regularization (PR) approach, which can transform the UBILS problem into an OBILS problem through dimension expansion. But this PR approach was designed for some communication applications and requires the constraint box $\mathcal{B}$ in (1.4) to be in some specific forms. More details of this PR approach will be reviewed in Section 2.3.3. For certain applications, we can obtain $\|\mathbf{x}_{(2)}\|_2^2$ as a constant and the UBILS problem can be easily transformed, through adding a term associated with $\|\mathbf{x}_{(2)}\|_2^2$ in the objective function into OBILS problem, which takes much less computation (see e.g., [23, 60, 66]). The modified ADMM method proposed in thesis is related to this approach.

Since the computational cost for UBILS problems is very high, a number of near-optimal approaches have also been developed to reduce the complexity (see e.g., [26,29,32,42,54–57]). The main idea of these near-optimal methods is to replace exhaustive search with partial search. In this thesis, we mainly focus on optimal algorithms for solving the UBILS problem and some reviews of sub-optimal approaches will be given in Section 2.3.4.

The search methods for OBILS and UBILS problems mentioned above are both tree search approaches. The search cost is proportional to the number of tree nodes. If some techniques can be employed to prune the search tree, the search process can be accelerated. Various lower bound techniques that can prune the search tree have been proposed (see. e.g., [14, 36, 62, 78]). Most lower bound techniques can only deal with OBILS problems. In this thesis, a new lower bound technique, which can boost the search efficiency for UBILS problems, will be proposed in Section 4.2.

1.3 Alternating Direction Method of Multipliers

Alternating direction method of multipliers (ADMM) is a simple but powerful algorithm that is well suited to distributed convex optimization. It can be applied in many statistical fields (see e.g., [12]), such as constrained sparse regression, sparse signal recovery, image restoration and denoising, and so on. ADMM takes the form of a decomposition-coordination procedure, in which the solutions to small local sub-problems are coordinated to find a solution to a large global problem. It is worth mentioning that ADMM itself is not new, although it has become popular recently. It was first introduced in the mid-1970s by Gabay, Mercier, Glowinski, and Marrocco [35,37], though similar ideas emerged as early as the mid-1950s. ADMM can be viewed as an attempt to blend the benefits of dual decomposition and augmented Lagrangian methods for constrained optimization. It turns out to be equivalent or closely related to many other algorithms (see e.g., Chapter 3 of [12]). In Section 3.1, the theory of ADMM will be reviewed.

Although for convex problems ADMM is proved to show convergence under some mild assumptions, for non-convex problems it need not converge and when it does converge, it need not converge to an optimal point (see e.g., Chapter 9 of [12]). In [28], Diamond et al. discussed how to use ADMM as a heuristic to approximately solve various optimisation problems with convex objective and variables from a discrete set. In [64], Takapoui et al. proposed an ADMM-based heuristic approach for mixed integer quadratic programming. They showed that this approach has favourable computational costs and in many cases the global solution can be found although it is not guaranteed. In [33, 71], it is shown that for boolean optimisation problems where each variable takes either the value of 0 or 1, the 0-1 constraint can be transformed into continuous constraints and then ADMM with transformed constraints can be viewed as a global approach. In Section 3.2, these techniques mentioned above of ADMM on integer problems will be reviewed in detail.

In general, the performance of ADMM on problems with discrete constraints, including both speed (faster convergence) and accuracy (more likely to give the optimal solution), is closely related to the choice of initial point and penalty parameter (see e.g., [75, 76]). In Section 4.1, we will propose an useful and reasonable way of choosing initial point and penalty parameter while extending ADMM to UBILS problems as a heuristic.

1.4 Goal, Organization and Contributions

The main goal of this thesis is to improve the efficiency of the tree search method for UBILS problems. The thesis has two main contributions. The first one is that we propose a modified ADMM method to find a sub-optimal solution, which is used as an initial point of the tree search method. This is different from all the current approaches for the UBILS problems, except the method proposed in [64]. Comparing with the latter, our modified ADMM method is much more accurate. The second one is we propose an ADMM-based lower bound technique in the tree search to prune the search tree. To the best of our knowledge, there are no lower bounds that work well for general UBILS problem in literature. Numerical experiments indicate our improved tree search method can often beat well-know software packages for the UBILS problems.

The rest part of this thesis is organized as follows.

In Chapter 2, we will review existing algorithms for OILS, OBILS and UBILS problems.

In Chapter 3, we first review the general form of ADMM. Later we show how ADMM can be employed as a local method for problems with discrete constraints. Then we will discuss how ADMM can be considered as a heuristic approach for integer optimisation problems. Some ideas in recent literature on how ADMM can be used as a global method for certain integer problems will also be introduced.

In Chapter 4, we will propose a modified ADMM that can be viewed as a good heuristic for UBILS problems, and discuss how to choose parameters. Then we focus on how the modified ADMM can be incorporated to the tree search approach. We will propose ADMM-based lower bound techniques to prune the search tree and improve search efficiency.

In Chapter 5, simulation results for various algorithms as well as some well-known commercial solvers on UBILS problems are given to show the efficiency and effectiveness of our ADMM search approach.

In Chapter 6, we summarize our results and discuss some further research directions.

1.5 Notation

In this section, we introduce some notations and terms to be used in this thesis.

We denote scalars by normal type letters (usually lowercase letters, occasionally upper case letters), column vectors by boldface lowercase letters and matrices by boldface upper case letters.

We use $\mathbb{R}, \mathbb{Z}$ and $\mathbb{C}$ to denote the sets of real scalars, integer scalars and complex numbers respectively, $\mathbb{R}^n, \mathbb{Z}^n$ and $\mathbb{C}^n$ to denote the sets of n-dimensional real vectors, integer vectors and complex vectors respectively, and $\mathbb{R}^{m\times n}, \mathbb{Z}^{m\times n}$ and $\mathbb{C}^{m\times n}$ to denote the sets of $m \times n$ real matrices, integer matrices and complex matrices respectively.

For a column vector $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x}_{i:j}$ denotes the subvector composed of elements of $\mathbf{x}$ with indices from i to j. For a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{A}_{i:j,k:l}$ denotes the sub-matrix containing all the elements of $\mathbf{A}$ whose row indices are from i to j and column indices are from k to l. We use a_{ij} to denote the (i, j) element of $\mathbf{A}$ and $\mathbf{A}_{ij}$ to denote the (i, j) block of $\mathbf{A}$. We use $\mathbf{A}^\mathsf{T}$ to denote the transpose of matrix $\mathbf{A}$. If $\mathbf{A} \in \mathbb{R}^{n\times n}$, we use $\dim(\mathbf{A})$ to denote n.

For a scalar $\alpha \in \mathbb{R}$, $\lfloor \alpha \rceil$ denotes the nearest integer to α (if there is a tie, $\lfloor \alpha \rceil$ is the one with the smaller magnitude). Let $\lfloor \alpha \rfloor$ and $\lceil \alpha \rceil$ denote the integers we obtain after performing the floor and ceiling operations of α, respectively. $\lfloor \alpha \rceil_{\mathcal{B}}$ denotes the nearest integer in set $\mathcal{B}$ to α. Analogously, for a vector $\mathbf{x} \in \mathbb{R}^n$, $\lfloor \mathbf{x} \rceil$ denotes the closest integer vector to $\mathbf{x}$, i.e., its i-th entry is $\lfloor x_i \rceil$, for $i = 1, \ldots, n$.

We also define some special vectors and matrices here. We use $\mathbf{1}_{(n)}$ to denote the n-dimensional vector of ones and $\mathbf{0}_{(n)}$ to denote the n-dimensional vector of zeros. We use $\mathbf{I}$ to denote an identity matrix and $\mathbf{e}_k$ to denote the k-th column of the identity matrix $\mathbf{I}$.

For a random vector $\mathbf{x}$, we denote $\mathrm{E}\{\mathbf{x}\}$ as its expectation vector and $\mathrm{cov}\{\mathbf{x}\}$ as its covariance matrix. For a random variable X, we use $\mathrm{var}\{X\}$ to denote the variance of X. If a real-valued vector $\mathbf{x}$ is normally distributed with zero mean and covariance matrix $\sigma^2\mathbf{I}$, we write $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, and if a complex-valued vector is circularly complex normal distributed with zero mean and covariance matrix $\sigma^2\mathbf{I}$, we write $\mathbf{x} \sim \mathcal{CN}(\mathbf{0}, \sigma^2\mathbf{I})$.

For the sake of reading convenience, we provide a list of acronyms in Table 1.1.

OILS:	Ordinary Integer Least Squares
OBILS:	Overdetermined Box-constrained Integer Least Squares
UBILS:	Underdetermined Box-constrained Integer Least Squares
ADMM:	Alternating Direction Method of Multipliers

Table 1.1: Acronyms

Chapter 2

Algorithms for Solving Various ILS Problems

This chapter reviews enumeration algorithms for different ILS problems. In Sections 2.1, 2.2, 2.3, algorithms for solving OILS, OBILS and UBILS problems are introduced respectively. These algorithms provide a basis for the following chapters.

2.1 Ordinary ILS Problem

Given an OILS problem:

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

where $\mathbf{y} \in \mathbb{R}^n$ and $\mathbf{A} \in \mathbb{R}^{m\times n}$ has full column rank, a typical approach is the Schnorr-Euchner sphere decoding method, which consists of two phases: reduction and search. In Section 2.1.1 the well-known LLL reduction is reviewed, and the Schnorr-Euchner search strategy is covered in Section 2.1.2.

2.1.1 Reduction

A. QR and QRZ Factorization

The QR factorization of $\mathbf{A}$ has the following form:

$$\mathbf{A} = [\underset{n}{\mathbf{Q}_1}, \underset{m-n}{\mathbf{Q}_2}] \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix} \tag{2.2}$$

where $[\underset{n}{\mathbf{Q}_1}, \underset{m-n}{\mathbf{Q}_2}] \in \mathbb{R}^{m\times m}$ is orthogonal and $\mathbf{R} \in \mathbb{R}^{n\times n}$ is non-singular upper triangular. Without loss of generality, we assume that $r_{ii} > 0$ for $i = 1, \dots, n$ throughout the thesis. Under this assumption, both $\mathbf{Q}_1$ and $\mathbf{R}$ are uniquely determined by $\mathbf{A}$.

If the search strategy to be introduced later is used, the algorithm efficiency will depend on $\mathbf{R}$. Thus, we want to obtain a better $\mathbf{R}$ by using the so-called QRZ factorization. Note that what properties a good $\mathbf{R}$ should have will be introduced later. The QRZ factorization of $\mathbf{A}$ that always exists has the following form (see e.g., [15]):

$$\mathbf{Q}^{\mathsf{T}}\mathbf{A}\mathbf{Z} = \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix} \tag{2.3}$$

where $\mathbf{Q} = [\underset{n}{\mathbf{Q}_1}, \underset{m-n}{\mathbf{Q}_2}] \in \mathbb{R}^{m\times m}$ is an orthogonal matrix, $\mathbf{Z} \in \mathbb{Z}^{n\times n}$ is a unimodular matrix, i.e., $\det(\mathbf{Z}) = \pm 1$, and $\mathbf{R} \in \mathbb{R}^{n\times n}$ is an upper triangular matrix with positive diagonal entries.

Then we have:

$$\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 = \|\mathbf{Q}_1^{\mathsf{T}}\mathbf{y} - \mathbf{R}\mathbf{Z}^{-1}\mathbf{x}\|_2^2 + \|\mathbf{Q}_2^{\mathsf{T}}\mathbf{y}\|_2^2. \tag{2.4}$$

Define $\bar{\mathbf{y}} = \mathbf{Q}_1^{\mathsf{T}}\mathbf{y}$, $\mathbf{z} = \mathbf{Z}^{-1}\mathbf{x}$. Then (2.1) can be transformed to an equivalent problem:

$$\min_{\mathbf{z}\in\mathbb{Z}^n} \|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}\|_2^2. \tag{2.5}$$

Note that if the optimal solution to (2.5) is $\hat{\mathbf{z}}$, then the optimal solution to (2.1) is $\hat{\mathbf{x}} = \mathbf{Z}\hat{\mathbf{z}}$.

B. LLL Reduction

In the QRZ factorization (2.3), $\mathbf{Z}$ is an unimodular matrix. If we take $\mathbf{Z} = \mathbf{I}$, the QRZ factorization just becomes the QR factorization. The aim is to choose a unimodular matrix

Z so that **R** in (2.3) can make the search phase more efficient. The so-called LLL reduction, proposed by Lenstra, Lenstra and Lovász in [46], can give a good **R**. The upper triangular matrix **R** is called δ-LLL reduced if the following conditions hold

$$|r_{k-1,j}| \leq \frac{1}{2}|r_{k-1,k-1}|, \quad j = k, ..., n \tag{2.6}$$

$$\delta r_{k-1,k-1}^2 \leq r_{k-1,k}^2 + r_{kk}^2, \quad k = 2, ..., n \tag{2.7}$$

where $\frac{1}{4} < \delta \leq 1$. The inequality (2.6) is called the size reduction condition, and the inequality (2.7) is called the Lovász condition. Combining these two conditions, we can get:

$$|r_{k-1,k-1}| \leq \frac{2}{\sqrt{4\delta - 1}}|r_{k,k}|. \tag{2.8}$$

The LLL reduction can help to pursue the inequality:

$$|r_{11}| << |r_{22}| << ... << |r_{nn}|. \tag{2.9}$$

In [15], the benefits of **R** satisfying (2.9) are explained.

In [46], an algorithm was proposed for computing the LLL reduction. Under certain conditions, the cost of the algorithm in the worst case is a polynomial function of m and n. Chang et al. shown in [17] that the LLL reduction can reduce the complexity of sphere decoding. There have been some proposed improvements to the LLL reduction algorithm. Ling and Howgrave-Graham shown in [47] that when the LLL reduction is used to improve the performance of the Babai point in the linear model (1.1) (see [5]), the reduction for the off-diagonal entries of matrix **R** that are above the super-diagonal entries in the LLL algorithm is not necessary mathematically. Therefore, they proposed the so-called effective LLL (ELLL) algorithm by removing the unnecessary part from the LLL algorithm. The ELLL reduction algorithm is more efficient, but it has a numerical stability issue [74]. In [74], Xie et al. proposed the partial LLL (PLLL) reduction algorithm, which does size reductions for part of super-diagonal entries of **R** and the corresponding off-diagonal entries in the same columns.

Compared with ELLL, the PLLL reduction eliminates unnecessary size reduction on some super-diagonal entries such that the numerical stability problem in the ELLL reduction can be avoided. In addition, the PLLL reduction is faster than ELLL, because the minimum pivoting strategy is incorporated in the PLLL reduction process when computing the QR factorization to reduce the number of column permutations. In [74], Xie et al. also proved that the ELLL and PLLL reduction are equivalent to the LLL reduction in improving the efficiency of the search process for solving (2.5). Thus, the PLLL reduction algorithm will be used later in this thesis and its pseudocode is given in Algorithm 2.1.

Here we give some explanations for some lines in Algorithm 2.1:

- Line 1: The QR factorization with minimum column pivoting is applied to make the matrix $\mathbf{R}$ have small diagonal entries at the beginning (top left in $\mathbf{R}$) and large diagonal entries at the end (bottom right in $\mathbf{R}$).

- Line 7 and 10: The integer Gauss transformation (IGT) has the following form:

$$\mathbf{Z}_{ik} = \mathbf{I} - \zeta_{ik}\mathbf{e}_i\mathbf{e}_k^{\mathsf{T}}, \quad i \neq k, \quad \zeta_{ik} \in \mathbb{Z}.$$

  Here we apply $\mathbf{Z}_{ik}$ to $\mathbf{R}$ from the right where $\zeta_{ik} = \lfloor r_{ik}/r_{ii}\rceil$ to ensure $|r_{ik}| \leq |r_{ii}|/2$.

- Line 14: The goal of interchanging columns k and $k-1$ is to increase r_{kk} and decrease $r_{k-1,k-1}$.

- Line 15: After interchanging two columns of $\mathbf{R}$, $\mathbf{R}$ is not an upper triangular matrix and a Givens rotation is applied to $\mathbf{R}$ from the left to make $\mathbf{R}$ upper triangular again.

For more details on IGT, column permutation and Givens rotation matrix, see e.g., [15].

2.1.2 Search

In the following we introduce the well-known Schnorr-Euchner algorithm in [59], which is an enumeration search algorithm with the goal of finding the optimal solution in (2.5). The main reference for the following description is [15].

Algorithm 2.1 Partial LLL Reduction

Input: The matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with full column rank and the vector $\mathbf{y} \in \mathbb{R}^m$
Output: The PLLL-reduced upper triangular matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$, the vector $\bar{\mathbf{y}} \in \mathbb{R}^n$, the unimodular matrix $\mathbf{Z} \in \mathbb{Z}^{n\times n}$
Function : $(\mathbf{R}, \mathbf{Z}, \bar{\mathbf{y}}) = \text{PLLL-Reduction}(\mathbf{A}, \mathbf{y})$

```
1:  Compute the Householder QR factorization with minimum pivoting: $\mathbf{Q}^{\mathsf{T}}\mathbf{A}\mathbf{P} = \begin{bmatrix}\mathbf{R}\\ \mathbf{0}\end{bmatrix}$
    where $\mathbf{Q} = [\mathbf{Q}_1, \mathbf{Q}_2]$ is orthogonal, $\mathbf{P}$ is a permutation matrix and $\mathbf{R}$ is upper triangular
2:  Set $\mathbf{Z} = \mathbf{P}$, $k = 2$, $\bar{\mathbf{y}} = \mathbf{Q}_1^{\mathsf{T}}\mathbf{y}$
3:  while $k \le n$ do
4:      $\zeta = \lfloor r_{k-1,k}/r_{k-1,k-1} \rceil$, $\alpha = (r_{k-1,k} - \zeta r_{k-1,k-1})^2$
5:      if $\delta r_{k-1,k-1}^2 > (r_{kk}^2 + \alpha)$ then
6:          if $\zeta \neq 0$ then
7:              Apply the integer Gauss transformation $\mathbf{Z}_{k-1,k}$ to reduce $r_{k-1,k}$
8:              Update $\mathbf{Z}$: $\mathbf{Z} = \mathbf{Z}\mathbf{Z}_{k-1,k}$
9:              for $i = k-2, \ldots, 1$ do
10:                 Apply the integer Gauss transformation $\mathbf{Z}_{ik}$ to reduce $r_{ik}$
11:                 Update $\mathbf{Z}$: $\mathbf{Z} = \mathbf{Z}\mathbf{Z}_{ik}$
12:             end for
13:         end if
14:         Interchange columns $k-1$ and $k$ of $\mathbf{R}$
15:         Transform permuted $\mathbf{R}$ back to upper triangular by a Givens rotation
16:         Interchange columns $k-1$ and $k$ of $\mathbf{Z}$
17:         Apply the same Givens rotation to $\bar{\mathbf{y}}$
18:         if $k > 2$ then
19:             $k = k - 1$
20:         end if
21:     else
22:         $k = k + 1$
23:     end if
24: end while
```

Suppose that we have an upper bound β for the objective function in (2.5) such that its optimal solution satisfies:

$$\|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}\|_2^2 < \beta^2 \tag{2.10}$$

where β is referred to as the search radius. In fact, the inequality (2.10) is a hyper-ellipsoid with the center $\mathbf{R}^{-1}\bar{\mathbf{y}}$ in $\mathbf{z}$ space. Then the problem is to find the optimal integer point inside the hyper-ellipsoid. We first show the basic idea of the Schnorr-Euchner algorithm in a recursive fashion and then discuss the choice of β.

Suppose that we have the following partitioning for the matrix and the vectors in (2.10):

$$\bar{\mathbf{y}} = \begin{bmatrix} \hat{\mathbf{y}} \\ \bar{y}_n \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} \hat{\mathbf{R}} & \mathbf{r} \\ & r_{nn} \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} \hat{\mathbf{z}} \\ z_n \end{bmatrix}.$$

Then (2.10) is equivalent to:

$$\begin{aligned} \|(\hat{\mathbf{y}} - \mathbf{r}z_n) - \hat{\mathbf{R}}\hat{\mathbf{z}}\|_2^2 &< \beta^2 - (\bar{y}_n - r_{nn}z_n)^2, \\ (\bar{y}_n - r_{nn}z_n)^2 &< \beta^2. \end{aligned} \tag{2.11}$$

Define:

$$c_n := \bar{y}_n / r_{nn}, \quad \hat{\mathbf{y}}(z_n) := \hat{\mathbf{y}} - \mathbf{r}z_n, \quad \hat{\beta}^2(z_n) := \beta^2 - (\bar{y}_n - r_{nn}z_n)^2.$$

Then (2.11) can be rewritten as:

$$\|\hat{\mathbf{y}}(z_n) - \hat{\mathbf{R}}\hat{\mathbf{z}}\|_2^2 < \hat{\beta}^2(z_n), \tag{2.12}$$

$$c_n - \beta/r_{nn} < z_n < c_n + \beta/r_{nn}. \tag{2.13}$$

The equation (2.13) gives the search interval at level n. The search method first tries to find an integer z_n satisfying (2.13). If there is no solution, then (2.13) does not hold for any integer z_n, which means the hyper-ellipsoid (2.10) does not contain any integer points, and the search process stops. If an integer $z_n^\dagger$ satisfying (2.13) can be found, then the next step is to find an $(n-1)$-dimensional integer point within the $(n-1)$-dimensional hyper-ellipsoid (2.12), which is the solution to the following sub-problem:

$$\min_{\hat{\mathbf{z}} \in \mathbb{Z}^{n-1}} \|\hat{\mathbf{y}}(z_n) - \hat{\mathbf{R}}\hat{\mathbf{z}}\|_2^2. \tag{2.14}$$

We see that the problem (2.14) is still an OILS problem but the dimension is $n-1$. There are two possible cases for this sub-problem:

- There is a solution $\hat{\mathbf{z}}^\dagger$ to the OILS problem (2.14). In this case, we first update the search radius $\beta^2 = \left\| \bar{\mathbf{y}} - \mathbf{R} \begin{bmatrix} \hat{\mathbf{z}}^\dagger \\ z_n^\dagger \end{bmatrix} \right\|_2^2$, and then we go back to level n to find a new suitable value for z_n that satisfies (2.13). Then we continue the search process as before.

- There is no solution to the OILS problem (2.14). In this case, we go back to level n and find a new suitable value for z_n satisfying (2.13). Then we continue the search process as before.

Then we discuss how to choose a value for z_n at level n. We first choose the nearest integer to c_n, i.e., $\lfloor c_n \rceil$, as this choice makes $(\bar{y}_n - r_{nn}z_n)^2$ minimal. Later on when we come back to level n from level $n-1$, we can choose z_n to be the next nearest integer, namely, in the order as follows:

$$\lfloor c_n \rceil, \lfloor c_n \rceil - 1, \lfloor c_n \rceil + 1, \lfloor c_n \rceil - 2, ..., \quad \text{if } c_n \leq \lfloor c_n \rceil$$

or

$$\lfloor c_n \rceil, \lfloor c_n \rceil + 1, \lfloor c_n \rceil - 1, \lfloor c_n \rceil + 2, ..., \quad \text{if } c_n > \lfloor c_n \rceil.$$

When there is no integer in the search interval at level n for the latest β, the algorithm stops, and the latest point found is the optimal solution to (2.5). Note that when we solve the $(n-1)$-dimensional OILS problem (2.14), we use exactly the same method. Whenever we find a new point inside the hyper-ellipsoid with radius β, we update the search radius β to be smaller. Thus the search sphere gradually shrinks, and when there is only one point left in the hyper-ellipsoid, that is the optimal solution. The pseudocode of the search process is described in Algorithm 2.2. According to [39], the time complexity of Algorithm 2.2 can be upper bounded by $O(n^{n/2+o(n)})$.

From inequalities (2.11), it is easy to understand that the initial choice of β is important for the efficiency of search process. If the initial value of β is too large, there will be too many integer points inside the hyper-ellipsoid, and the search speed would be slow. On the other

Algorithm 2.2 Schnorr-Euchner Search

```
Input: The upper triangular matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$ with full column rank, the vector $\bar{\mathbf{y}} \in \mathbb{R}^n$
Output: The optimal solution to (2.5) $\mathbf{z}^{\text{OILS}} \in \mathbb{Z}^n$
Function : $\mathbf{z}^{\text{OILS}} = \text{SE-Search}(\mathbf{R}, \bar{\mathbf{y}})$
 1: Set $k = n, \beta = \infty, \mathbf{z}^{\text{OILS}} = \mathbf{0}_{(n)}$
 2: Compute $c_k = (\bar{y}_k - \sum_{j=k+1}^{n} r_{kj} z_j)/r_{kk}$, $z_k = \lfloor c_k \rceil$, $d_k = \text{sign}(c_k - z_k)$
 3: if $r_{kk}^2 (z_k - c_k)^2 \geq \beta^2 - \sum_{j=k+1}^{n} r_{jj}^2 (z_j - c_j)^2$ then
 4:     if $k = n$ then
 5:         Terminate (The optimal solution has been found)
 6:     else
 7:         Set $k = k + 1$ and go to Line 13
 8:     end if
 9: else if $k > 1$ then
10:     Set $k = k - 1$ and go to Line 2
11: end if
12: ($k = 1$ and a valid point is found) Set $\beta = \|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}\|_2$, save $\mathbf{z}^{\text{OILS}} = \mathbf{z}$ and set $k = k + 1$
13: (Enumeration at level $k$) Set $z_k = z_k + d_k$, $d_k = -d_k - \text{sign}(d_k)$ and go to Line 3
```

hand, if the initial search radius β is too small, there may be no points inside the hyper-ellipsoid, and the algorithm will terminate without giving a solution. For the OILS problem (2.5), we usually set the initial search radius $\beta = \infty$, and the first integer point obtained in the search approach is called the (ordinary) Babai point [5]. Another simple method to initialize β is to solve the corresponding real least squares problem, i.e., $\mathbf{z}^{\text{RLS}} = \mathbf{R}^{-1}\bar{\mathbf{y}}$, and then set $\beta = \|\bar{\mathbf{y}} - \mathbf{R}\lfloor \mathbf{z}^{\text{RLS}} \rceil\|_2$. If there is no integer point inside this hyper-ellipsoid, it means that $\lfloor \mathbf{z}^{\text{RLS}} \rceil$ is the optimal solution of (2.5).

2.2 Overdetermined Box-constrained ILS Problem

We rewrite the OBILS problem in (1.4):

$$\min_{\mathbf{x}\in\mathcal{B}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 \tag{2.15}$$

where $\mathcal{B} = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{l}, \mathbf{u} \in \mathbb{Z}^n\}$, $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{m\times n}$ ($m \geq n$) has full column rank.

The main difference between (2.15) and the OILS problem is the constraint box $\mathcal{B}$, which makes it difficult for the reduction and search strategy mentioned in Section 2.1 to be applied directly. Thus in this section, we will first introduce the AIP reduction algorithm in [13] and then focus on the search algorithm proposed in [16] to solve the OBILS problem.

2.2.1 Reduction

The LLL reduction approach for OILS problems cannot be applied to OBILS problems because the geometry of the constraint box would become complicated after $\mathbf{x}$ is changed to $\mathbf{z}$ by the unimodular transformation and then the search process would be difficult. Therefore we apply a column permutation matrix $\mathbf{P}$ to $\mathbf{A}$ in the OBILS problem (2.15) from right such that:

$$\mathbf{AP} = [\underset{n}{\mathbf{Q}_1}, \underset{m-n}{\mathbf{Q}_2}] \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix} \tag{2.16}$$

where $\mathbf{P}$ is a $n \times n$ permutation matrix, $[\underset{n}{\mathbf{Q}_1}, \underset{m-n}{\mathbf{Q}_2}] \in \mathbb{R}^{m \times m}$ is orthogonal and $\mathbf{R} \in \mathbb{R}^{n \times n}$ is non-singular upper triangular. Then we have:

$$\|\mathbf{y} - \mathbf{Ax}\|_2^2 = \|\mathbf{Q}_1^{\mathsf{T}}\mathbf{y} - \mathbf{RP}^{\mathsf{T}}\mathbf{x}\|_2^2 + \|\mathbf{Q}_2^{\mathsf{T}}\mathbf{y}\|_2^2. \tag{2.17}$$

If we define:

$$\bar{\mathbf{y}} = \mathbf{Q}_1^{\mathsf{T}}\mathbf{y}, \quad \mathbf{z} = \mathbf{P}^{\mathsf{T}}\mathbf{x}, \quad \bar{\mathbf{l}} = \mathbf{P}^{\mathsf{T}}\mathbf{l}, \quad \bar{\mathbf{u}} = \mathbf{P}^{\mathsf{T}}\mathbf{u}, \tag{2.18}$$

then the original problem (2.15) can be transformed to:

$$\min_{\mathbf{z} \in \bar{\mathcal{B}}} \|\bar{\mathbf{y}} - \mathbf{Rz}\|_2^2 \tag{2.19}$$

where

$$\bar{\mathcal{B}} = \{\mathbf{x} \in \mathbb{Z}^n : \bar{\mathbf{l}} \leq \mathbf{x} \leq \bar{\mathbf{u}}, \bar{\mathbf{l}}, \bar{\mathbf{u}} \in \mathbb{Z}^n\}. \tag{2.20}$$

Then in the search process, we try to find the optimal solution $\hat{\mathbf{z}}$ for the problem (2.19) and get the solution for the problem (2.15) $\hat{\mathbf{x}} = \mathbf{P}\hat{\mathbf{z}}$.

There have been various column reordering strategies (different permutation matrices $\mathbf{P}$) (see e.g., [13,16,63,72,73]), which have the potential to improve search efficiency. Then we give an effective approach —— the AIP (all information permutation) reduction algorithm proposed in [13] in Algorithm 2.3.

Here we give some explanations for some lines in Algorithm 2.3:

- Line 4: AIP reduction first determines the nth column of $\mathbf{R}$, and then the $(n-1)$th column and so on.

- Line 6: AIP reduction chooses column $j = \mathrm{agrmax}_{i=1,2,..,k} d_i^{(k)}$ to be the new kth column. The motivation of this choice is to make the search radius for the $(n-1)-$dimensional subproblem small and at the same time to make r_{kk} large (for more explanations we refer to [16]).

2.2.2 Search

In this section, we will introduce the search algorithm for OBILS problems proposed by Chang and Han in [16]. We refer to it as CH search algorithm. Suppose for the OBILS problem (2.19) we can find β such that the optimal solution satisfies:

$$\|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}\|_2^2 < \beta^2. \tag{2.21}$$

The CH search algorithm is a based on the Schnorr-Euchner approach and takes the box constraint $\mathcal{B}$ in (2.20) into consideration. In the Schnorr-Euchner search process, at level k, integer candidates for z_k are enumerated by an increasing order of $|z_k - c_k|$ where $c_k = (\bar{y}_k - \sum_{j=k+1}^{n} r_{kj} z_j)/r_{kk}$. The same strategy is used in CH search except that z_k should be in the constrained interval $[\bar{l}_k, \bar{u}_k]$. We choose the first nearest integer to c_k in the interval $[\bar{l}_k, \bar{u}_k]$ as the first candidate of z_k, and choose the second nearest integer to c_k in the interval $[\bar{l}_k, \bar{u}_k]$ as the second candidate of z_k, and so on until each integer in $[\bar{l}_k, \bar{u}_k]$ has been chosen. The pseudocode of CH search method can be found in Algorithm 2.4.

Algorithm 2.3 AIP Reduction

Input: The matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with full column rank and the vector $\mathbf{y} \in \mathbb{R}^m$, the lower bound vector $\mathbf{l} \in \mathbb{Z}^n$ and upper bound vector $\mathbf{u} \in \mathbb{Z}^n$
Output: The reduced upper triangular matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$, the permutation matrix $\mathbf{P} \in \mathbb{Z}^{n\times n}$, the vector $\bar{\mathbf{y}} \in \mathbb{R}^n$, the permuted lower and upper bound vector $\bar{\mathbf{l}}$ and $\bar{\mathbf{u}}$
Function : $(\mathbf{R}, \mathbf{P}, \bar{\mathbf{y}}, \bar{\mathbf{l}}, \bar{\mathbf{u}}) = \text{AIP-Reduction}(\mathbf{A}, \mathbf{y}, \mathbf{l}, \mathbf{u})$

```
1: Compute the Householder QR factorization: $\mathbf{Q}^\mathsf{T}\mathbf{A} = \begin{bmatrix}\mathbf{R}\\ \mathbf{0}\end{bmatrix}$ where $\mathbf{Q} = [\mathbf{Q}_1, \mathbf{Q}_2]$ is
   orthogonal and $\mathbf{R}$ is upper triangular
2: Initialize $\bar{\mathbf{l}} = \mathbf{l}, \bar{\mathbf{u}} = \mathbf{u}, \bar{\mathbf{y}} = \mathbf{Q}_1^\mathsf{T}\mathbf{y}, \widetilde{\mathbf{y}} = \bar{\mathbf{y}}, \mathbf{P} = \mathbf{I}_n$ and set $\mathbf{L} = \mathbf{R}^{-\mathsf{T}}$
3: Compute $\breve{\mathbf{l}}$ with $\breve{l}_i = \|\mathbf{L}_{:,i}\|_2^2$ for $i = 1, 2, ..., n$
4: for $k = n : -1 : 2$ do
5:     Solve $\mathbf{L}^\mathsf{T}\breve{\mathbf{z}} = \breve{\mathbf{y}}$ and let $z_i^s$ be the second closest integer in $[\bar{l}_i, \bar{u}_i]$ to $\breve{z}_i$ for $i = 1, 2, ..., k$
6:     Compute $d_i^{(k)} = (z_i^s - \breve{z}_i)^2/\breve{l}_i$ for $i = 1, 2, ..., k$ and let $j = \text{agrmax}_{i=1,2,..,k} d_i^{(k)}$
7:     Compute $z_k = \text{median}(\lfloor \breve{z}_i \rceil, \bar{l}_i, \bar{u}_i)$ and $\widetilde{\mathbf{y}} = \widetilde{\mathbf{y}} - \mathbf{R}_{1:j,j} z_k$
8:     Remove column $j$ of $\mathbf{L}$ and entry $j$ of $\breve{\mathbf{l}}$
9:     if $i \neq k$ then
10:        Interchange columns $j$ and $k$ of $\mathbf{R}$
11:        Interchange entry $j$ and $k$ of $\bar{\mathbf{l}}$ and $\bar{\mathbf{u}}$
12:        Interchange columns $j$ and $k$ of $\mathbf{P}$
13:        Use Givens rotations to bring $\mathbf{R}$ back to upper triangular
14:        Apply the same Givens rotation to update $\mathbf{L}, \bar{\mathbf{y}}, \widetilde{\mathbf{y}}$
15:    end if
16:    Update $\breve{l}_i = \breve{l}_i - l_{ki}^2$ for $i = 1, 2, ..., k$
17:    Remove row $k$ of $\mathbf{L}$ and entry $k$ of $\widetilde{\mathbf{y}}$
18: end for
```

For the OBILS problem, we can also set the initial search radius β as ∞ like what we have done for the OILS problem in Section 2.1.2. We can also compute the so-called Babai point $\mathbf{z}^{\text{Babai}}$, which solves a sequence of n 1-dimensional OBILS problems (see e.g., [69]). Then we set the initial β to be $\|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}^{\text{Babai}}\|_2$. Note that if there is no valid integer point inside the hyper-ellipsoid with this β, then $\mathbf{z}^{\text{Babai}}$ is the optimal solution of (2.19). Another strategy is to solve the corresponding box-constrained real least squares problem:

$$\min_{\mathbf{z}\in\mathbb{R}^n, \bar{\mathbf{l}}\leq\mathbf{z}\leq\bar{\mathbf{u}}} \|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}\|_2^2. \tag{2.22}$$

The methods for solving the box-constrained real least squares problem can be found in [8]. Suppose the optimal solution of (2.22) is $\mathbf{z}^{\text{BRLS}}$, then we take $\beta = \|\bar{\mathbf{y}} - \mathbf{R}\lfloor\mathbf{z}^{\text{BRLS}}\rceil\|_2$. Note

Algorithm 2.4 CH Search

Input: The upper triangular matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$ with full column rank, the vector $\bar{\mathbf{y}} \in \mathbb{R}^n$, the lower bound vector $\bar{\mathbf{l}} \in \mathbb{Z}^n$, the upper bound vector $\bar{\mathbf{u}} \in \mathbb{Z}^n$, the initial search radius β
Output: The optimal solution to (2.19) $\mathbf{z}^{\text{OBILS}} \in \mathbb{Z}^n$
Function : $\mathbf{z}^{\text{OBILS}} = \text{CH-Search}(\mathbf{R}, \bar{\mathbf{y}}, \bar{\mathbf{l}}, \bar{\mathbf{u}}, \beta)$

```
1: Set $k = n, \mathbf{z} = \mathbf{0}_{(n)}$
2: Compute $c_k = (\bar{y}_k - \sum_{j=k+1}^{n} r_{kj} z_j)/r_{kk}$, $z_k = \lfloor c_k \rceil$
3: Set $lbound_k = 0$ and $ubound_k = 0$
4: if $z_k \le \bar{l}_k$ then
5:     Set $z_k = \bar{l}_k$, $lbound_k = 1$, $d_k = 1$
6: else if $z_k \ge \bar{u}_k$ then
7:     Set $z_k = \bar{u}_k$, $ubound_k = 1$, $d_k = -1$
8: else
9:     Set $d_k = \text{sign}(c_k - z_k)$
10: end if
11: if $r_{kk}^2 (z_k - c_k)^2 \ge \beta^2 - \sum_{j=k+1}^{n} r_{jj}^2 (z_j - c_j)^2$ then
12:     if $k = n$ then
13:         Terminate (The optimal solution has been found)
14:     else
15:         Set $k = k + 1$ and go to Line 21
16:     end if
17: else if $k > 1$ then
18:     Set $k = k - 1$ and go to Line 2
19: end if
20: ($k = 1$ and a valid point is found) Set $\beta = \|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}\|_2$, save $\mathbf{z}^{\text{OBILS}} = \mathbf{z}$ and set $k = k + 1$
21: (Enumeration at level $k$)
22: if $lbound_k = 1$ and $ubound_k = 1$ then
23:     Go to Line 12
24: end if
25: Set $z_k = z_k + d_k$
26: if $z_k = \bar{l}_k$ then
27:     Set $lbound_k = 1$ and $d_k = -d_k - \text{sign}(d_k)$
28: else if $z_k = \bar{u}_k$ then
29:     Set $ubound_k = 1$ and $d_k = -d_k - \text{sign}(d_k)$
30: else if $lbound_k = 1$ then
31:     Set $d_k = 1$
32: else if $ubound_k = 1$ then
33:     Set $d_k = -1$
34: else
35:     Set $d_k = -d_k - \text{sign}(d_k)$
36: end if
37: Go to Line 11
```

that if we cannot find any integer point with this β, it means that $\lfloor \mathbf{z}^{\mathrm{BRLS}} \rceil$ is the optimal solution to (2.19).

2.3 Underdetermined Box-constrained ILS Problem

The problem (1.4) is considered as the UBILS problem when $m < n$ and $\mathbf{A}$ has full row rank. We rewrite the UBILS problem:

$$\min_{\mathbf{x} \in \mathcal{B}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 \tag{2.23}$$

where $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{m \times n}$ $(m < n)$ has full row rank, $\mathcal{B} = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{l}, \mathbf{u} \in \mathbb{Z}^n\}$. Next, in Section 2.3.1, we introduce the direct tree search (DTS) method proposed in [19] to solve the UBILS problem and in Section 2.3.2, we introduce how lower bound techniques can improve the search efficiency. In Section 2.3.3, another approach that can transform certain types of UBILS problems into equivalent OBILS problems by dimension expansion, the partial regularization (PR) algorithm proposed in [20], is introduced. In Section 2.3.4 we review some sub-optimal approaches.

2.3.1 Direct Tree Search

Similar to the OILS and OBILS problem, the first phase is to reduce the UBILS problem in (2.23). Like the reduction introduced in Section 2.2.1, we can apply a permutation matrix $\mathbf{P}$ from the right to $\mathbf{A}$ such that the OR decomposition becomes:

$$\mathbf{A}\mathbf{P} = \mathbf{Q}\mathbf{R} \tag{2.24}$$

where $\mathbf{P}$ is a $n \times n$ permutation matrix, $\mathbf{Q} \in \mathbb{R}^{m \times m}$ is orthogonal and $\mathbf{R} \in \mathbb{R}^{m \times n}$ is upper trapezoidal ($r_{ij} = 0$ for all $i > j$). Then we have:

$$\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 = \|\mathbf{Q}^{\mathsf{T}}\mathbf{y} - \mathbf{R}\mathbf{P}^{\mathsf{T}}\mathbf{x}\|_2^2. \tag{2.25}$$

If we define:

$$\bar{\mathbf{y}} = \mathbf{Q}^{\mathsf{T}}\mathbf{y}, \quad \mathbf{z} = \mathbf{P}^{\mathsf{T}}\mathbf{x}, \quad \bar{\mathbf{l}} = \mathbf{P}^{\mathsf{T}}\mathbf{l}, \quad \bar{\mathbf{u}} = \mathbf{P}^{\mathsf{T}}\mathbf{u}, \tag{2.26}$$

then the original problem (2.23) can be transformed to:

$$\min_{\mathbf{z}\in\bar{\mathcal{B}}} \|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}\|_2^2 \tag{2.27}$$

where

$$\bar{\mathcal{B}} = \{\mathbf{x} \in \mathbb{Z}^n : \bar{\mathbf{l}} \leq \mathbf{x} \leq \bar{\mathbf{u}}, \bar{\mathbf{l}}, \bar{\mathbf{u}} \in \mathbb{Z}^n\}. \tag{2.28}$$

Then in the search process, we find the optimal solution $\hat{\mathbf{z}}$ for the problem (2.27) and get the solution for the problem (2.23) $\hat{\mathbf{x}} = \mathbf{P}\hat{\mathbf{z}}$.

We first introduce how the DTS approach in [19] conducts the search process, and then focus on how it generates the reduced form of the UBILS problem (obtain the permutation matrix $\mathbf{P}$ in (2.24) that helps to improve search efficiency).

2.3.1.1 DTS Search

Different from the OBILS problem (2.19), the matrix $\mathbf{R}$ in the UBILS problem (2.27) is upper trapezoidal. Thus the search approach given in Section 2.2.2 cannot work here because it has only one variable at each level but now we have, at level m, $n-m+1$ variables. It means that we have to determine $n-m+1$ variables first with only one inequality at the top level, and that is why the UBILS problem is much more difficult to solve than the OBILS problem in practice.

Suppose we have a constant $\beta > 0$ such that the optimal solution of (2.27) satisfies:

$$\|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}\|_2^2 < \beta^2. \tag{2.29}$$

A method of finding β can be found in [18], which first solves a box-constrained real least squares problem and then rounds the solution to an integer vector in $\bar{\mathcal{B}}$, say $\mathbf{z}^{\mathrm{BRLS}}$, ant then takes $\beta = \|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}^{\mathrm{BRLS}}\|_2$. Another method of finding the initial β will be proposed in Chapter 4.

We can expand the left side of (2.29) and rewrite the inequality as:

$$\sum_{i=1}^{m}(\bar{y}_i - \sum_{j=i}^{n} r_{ij}z_j)^2 < \beta^2. \tag{2.30}$$

(2.30) is equivalent to:

$$(\bar{y}_m - \sum_{j=m}^{n} r_{mj}z_j)^2 < \beta^2, \tag{2.31}$$

$$\sum_{i=1}^{m-1}(\bar{y}_i - \sum_{j=i}^{m-1} r_{ij}z_j - \sum_{j=m}^{n} r_{ij}z_j)^2 < \beta^2 - (\bar{y}_m - \sum_{j=m}^{n} r_{mj}z_j)^2. \tag{2.32}$$

It is easy to see that (2.32) is an OBILS problem if $\mathbf{z}_{m:n}$ bas been determined, which can be solved by CH search algorithm and is equivalent to the following set of inequalities:

$$(\bar{y}_k - \sum_{j=k}^{n} r_{kj}z_j)^2 < \beta^2 - \sum_{i=k+1}^{m}(\bar{y}_i - \sum_{j=i}^{n} r_{ij}z_j)^2, \quad k = m-1, ..., 1. \tag{2.33}$$

For simplicity, we define:

$$c_i = (\bar{y}_i - \sum_{j=i+1}^{n} r_{ij}z_j)/r_{ii}, \quad i = m-1, ..., 1. \tag{2.34}$$

Thus for level $k = m-1, ..., 1$, we can define $\bar{\mathcal{B}}_k = \{\bar{l}_k, \bar{l}_k + 1, ..., \bar{u}_k\}$ and have the feasible set:

$$\begin{aligned} z_k \in & \mathcal{Z}_k = \bar{\mathcal{B}}_k \cap (\lambda_k, \varphi_k) \\ \lambda_k = c_k - & \sqrt{\beta^2 - \sum_{i=k+1}^{m} r_{ii}^2(z_i - c_i)_2^2/|r_{kk}|} \\ \varphi_k = c_k + & \sqrt{\beta^2 - \sum_{i=k+1}^{m} r_{ii}^2(z_i - c_i)_2^2/|r_{kk}|}. \end{aligned} \tag{2.35}$$

Then we will show how the DTS approach deals with level m in inequality (2.31). Let $\mathcal{J}^+ = \{j|r_{mj} \geq 0, m \leq j \leq n\}$, $\mathcal{J}^- = \{j|r_{mj} < 0, m \leq j \leq n\}$. Then (2.31) is equivalent to:

$$\bar{y}_m - \beta < \sum_{j\in\mathcal{J}^+} r_{mj}z_j + \sum_{j\in\mathcal{J}^-} r_{mj}z_j < \bar{y}_m + \beta. \tag{2.36}$$

Define the following transformation for z_j, $j = m, m+1, ..., n$:

$$\bar{z}_j = \begin{cases} -\bar{l}_j + z_j & \text{if } j \in \mathcal{J}^+ \\ \bar{u}_j - z_j & \text{if } j \in \mathcal{J}^- \end{cases} \tag{2.37}$$

so that the constraint $\bar{\mathcal{B}}_j$ becomes $\bar{\mathcal{B}}'_j$:

$$\bar{z}_j \in \bar{\mathcal{B}}'_j = \{0, 1, ..., \bar{u}_j - \bar{l}_j\}. \tag{2.38}$$

Define:

$$\alpha = \bar{y}_m - \sum_{j\in\mathcal{J}^+} |r_{mj}\bar{l}_j| + \sum_{j\in\mathcal{J}^-} |r_{mj}\bar{u}_j|, \tag{2.39}$$

then (2.36) becomes:

$$\alpha - \beta < \sum_{j=m}^{n} |r_{mj}|\bar{z}_j < \alpha + \beta. \tag{2.40}$$

Assume $r_{mj} \neq 0$ for $j = m, m+1, ..., n$ (exceptions will be discussed later in this section), then from (2.38) and (2.40) it is easy to get the feasible region $\bar{\mathcal{Z}}_j$ for $j = n, n-1, ..., m$:

$$\begin{aligned} \bar{z}_j \in \bar{\mathcal{Z}}_j &= \bar{\mathcal{B}}'_j \cap (\bar{\lambda}_j, \bar{\varphi}_j) \\ \bar{\lambda}_j &= (\alpha - \beta - \sum_{i=j+1}^{n} |r_{mi}|\bar{z}_i - \sum_{i=m}^{j-1} |r_{mi}|(\bar{u}_i - \bar{l}_i))/|r_{mj}| \\ \bar{\varphi}_j &= (\alpha + \beta - \sum_{i=j+1}^{n} |r_{mi}|\bar{z}_i)/|r_{mj}|. \end{aligned} \tag{2.41}$$

With inequalities (2.35) and (2.41), a depth-first tree search algorithm can be developed to find the optimal solution of (2.27). Now we describe the DTS strategy starting from level

n. First we take $\bar{z}_n = \lfloor \frac{\alpha}{|r_{mn}|} \rceil |_{\bar{\mathcal{Z}}_n}$, which denotes the nearest integer to $\frac{\alpha}{|r_{mn}|}$ in the set $\bar{\mathcal{Z}}_n$. Then we move to level $n-1$ and compute the set $\bar{\mathcal{Z}}_{n-1}$. If $\bar{\mathcal{Z}}_{n-1}$ is empty, it means that the $\bar{z}_n$ we choose is invalid. Then we move back to level n and choose $\bar{z}_n$ to be next nearest integer to $\frac{\alpha}{|r_{mn}|}$ in the set $\bar{\mathcal{Z}}_n$ and go to level $n-1$ again. If $\bar{\mathcal{Z}}_{n-1}$ is not empty, we choose $\bar{z}_{n-1} = \lfloor \frac{\alpha - |r_{mn}\bar{z}_n|}{|r_{m,n-1}|} \rceil |_{\bar{\mathcal{Z}}_{n-1}}$. We continue this process until we reach level m and find a valid integer for $\bar{z}_m$. At this point, we have fixed $\bar{z}_n, ..., \bar{z}_m$ and can transform $\bar{\mathbf{z}}_{m:n}$ to the original integer vector $\mathbf{z}_{m:n}$ by following (2.37). Then we move to the level $m-1$ and compute $\mathcal{Z}_{m-1}$ with (2.35). If $\mathcal{Z}_{m-1}$ is empty, we move to level m and choose the next $\bar{z}_m$. Otherwise, we choose $z_{m-1} = \lfloor c_{m-1} \rceil |_{\mathcal{Z}_{m-1}}$ and move down to level $m-2$. Continue this procedure until we reach level 1 and choose a valid integer for z_1. At this time, we have fixed all the variables and a full integer point $\mathbf{z}^*$ is found. Then we update $\beta = \|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}^*\|_2$. Next, we search within the new hyper-ellipsoid. We go back to level 2 and choose z_2 to be the next nearest integer to c_2 in the set $\mathcal{Z}_2$. Note that the candidate set $\mathcal{Z}_2$ should have been updated using the new β in (2.35). If such z_2 exists, we move down to level 1 to update the value of z_1. Otherwise, we move up to level 3 to update the value of z_3, and so on. In general, in this tree search process, if we find a valid integer at level $j+1$ we move down to level j. If we fail to find an integer at level j, we move up to level $j+1$. Finally, if we fail to find a new valid integer for $\bar{z}_n$ at level n, the search process terminates and the latest found integer point is the optimal solution.

It is possible that in (2.41) $r_{mj} = 0$ for some j. In this case all the integers in the set $\bar{\mathcal{Z}}_j$ can be candidates for $\bar{z}_j$. When we go to level j from level $j+1$, we choose $\bar{z}_j$ to be the smallest integer in the set $\bar{\mathcal{Z}}_j$, i.e., $\bar{z}_j = 0$. And if we go to level j from level $j-1$, we set $\bar{z}_j$ to be the next smallest integer in the set $\bar{\mathcal{Z}}_j$. Fortunately, this case ($r_{mj} = 0$) rarely occurs in practice.

In Algorithm 2.5, the pseudocode of this DTS approach is presented.

In [78], Zhu found that $\frac{\alpha}{|r_{mn}|}$ is usually larger than $\bar{u}_n$ in practice, which means that the search process often starts with branch $\bar{z}_n = \bar{u}_n$ and enumerates all the branches by the order of $\bar{z}_n = \bar{u}_n, \bar{z}_n = \bar{u}_n - 1, ..., \bar{z}_n = \bar{l}_n$. Zhu suggested that if the optimal solution is

Algorithm 2.5 DTS Search

Input: The trapezoidal matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$ with full row rank, the vector $\bar{\mathbf{y}} \in \mathbb{R}^m$, the lower bound vector $\bar{\mathbf{l}} \in \mathbb{Z}^n$, the upper bound vector $\bar{\mathbf{u}} \in \mathbb{Z}^n$, the initial search radius β

Output: The optimal solution to (2.27) $\mathbf{z}^{\text{UBILS}} \in \mathbb{Z}^n$

Function : $\mathbf{z}^{\text{UBILS}} = \text{DTS-Search}(\mathbf{R}, \bar{\mathbf{y}}, \bar{\mathbf{l}}, \bar{\mathbf{u}}, \beta)$

```
1: Step 1(Initialization)
2:     Set $k = n$
3: end Step 1
4: Step 2
5:     Compute $\bar{\lambda}_k$ and $\bar{\mu}_k$ by following (2.41)
6:     Set $\bar{\mathcal{Z}}_k = \{0, 1, ..., \bar{u}_k - \bar{l}_k\} \cap (\bar{\lambda}_k, \bar{\mu}_k)$
7:     if $\bar{\mathcal{Z}}_k$ is empty then Go to Step 4
8:     else
9:         Compute $c_k = \frac{\alpha - \sum_{j=k+1}^{n} |r_{mj}| \bar{z}_j}{|r_{mk}|}$, $\bar{z}_k = \lfloor c_k \rceil |_{\bar{\mathcal{Z}}_k}$
10:    end if
11: end Step 1
12: Step 3
13:     if $k > m - 1$ then Set $k = k - 1$ and Go to Step 2
14:     else
15:         Transform $\bar{\mathbf{z}}_{m:n}$ back to $\mathbf{z}_{m:n}$ using (2.37)
16:         Compute $\widetilde{\mathbf{y}} = \bar{\mathbf{y}}_{1:m-1} - \mathbf{R}_{1:m-1,m:n}\mathbf{z}_{m:n}$ and $T_m = (\bar{y}_m - \mathbf{R}_{m,m:n}\mathbf{z}_{m:n})^2$
17:         Compute $\mathbf{z}_{1:m-1}$ = CH-Search($\mathbf{R}_{1:m-1,1:m-1}, \widetilde{\mathbf{y}}, \bar{\mathbf{l}}_{1:m-1}, \bar{\mathbf{u}}_{1:m-1}, \sqrt{\beta^2 - T_m}$) (See
    Algorithm 2.4)
18:         Set $\mathbf{z}^{\text{UBILS}} = \mathbf{z}$ and $\beta = \sqrt{(\widetilde{\mathbf{y}} - \mathbf{R}_{1:m-1,1:m-1}\mathbf{z}_{1:m-1})^2 + T_m}$
19:     end if
20: end Step 3
21: Step 4
22:     if $k = n$ then
23:         Terminate
24:     else
25:         Set $k = k + 1$
26:     end if
27: end Step 4
28: Step 5
29:     Choose $\bar{z}_k \in \bar{\mathcal{Z}}_k$ to be the next nearest integer to $c_k$
30:     if $\bar{z}_k$ does not exist then
31:         Go to Step 4
32:     else
33:         Go to Step 3
34:     end if
35: end Step 5
```

in a branch with small value for $\bar{z}_n$, then the search process will come to this branch at a late stage, making the search process inefficient. Thus Zhu proposed a new search ordering strategy at level n in [78], which is based on upper bounds of the minimal value of the objective function. The details are omitted here because for our ADMM search algorithm, such reordering strategy may not have the potential to help improve the search efficiency, which will be shown through numerical experiments in Chapter 5.

2.3.1.2 DTS Reduction

We introduce the reduction method proposed in [19] which transforms the original UBILS problem (2.23) to the reduced form (2.27).

In the reduction, we first determine how to choose $n-m+1$ columns as the right part of the permuted $\mathbf{A}$, and the remaining $m-1$ columns as the left part. Then we determine how to order the columns for each part.

We suppose that the two parts of $\mathbf{A}$ have been determined, and we have a permuted $\mathbf{A}$ and an associated upper trapezoidal matrix $\mathbf{R}$. We show how to reorder the right part of $\mathbf{R}$. At level j ($m \leq j \leq n$), the integer set $\bar{\mathcal{Z}}_j$ can be computed in (2.41). To make the search process more efficient, we prefer $\bar{\mathcal{Z}}_j$ to be smaller, which motivates the following reordering strategy. Define:

$$L_j = \min\{u_j - l_j, \lfloor \bar{\varphi}_j \rfloor + \operatorname{sign}(\bar{\varphi}_j + \lfloor \bar{\varphi}_j \rfloor) - 1\} - \max\{0, \lceil \bar{\lambda}_j \rceil + \operatorname{sign}(\bar{\lambda}_j + \lceil \bar{\lambda}_j \rceil) - 1\} \quad (2.42)$$

where L_j is the number of integers in $\bar{\mathcal{Z}}_j$ if greater than zero and otherwise, it means that $\bar{\mathcal{Z}}_j$ is empty. Suppose that we have chosen the last $n-j$ columns of $\mathbf{R}$ and now we need to determine column j of $\mathbf{R}$ where $j \geq m$. We first compute L_j with the current jth column of $\mathbf{R}$ and then interchange jth and ith column of $\mathbf{R}$ for $i = m, m+1, ..., j-1$. Then we find the smallest L_j and the corresponding column with this smallest L_j is chosen to be the jth column of $\mathbf{R}$ before we move on to level $j-1$. This procedure is repeated until we order all the last $n-m+1$ columns or we encounter a column whose smallest L_j is non-positive. The detailed procedure is described in Algorithm 2.6.

Algorithm 2.6 Reorder columns of $\mathbf{R}_{:,m:n}$

Input: The trapezoidal matrix $\mathbf{R} \in \mathbb{R}^{m\times n}$ with full row rank, $\bar{y}_m$, β
Output: permuted $\mathbf{R}$, the permutation matrix $\mathbf{P} \in \mathbb{Z}^{n\times n}$, variable $index$ for which $\bar{\mathcal{Z}}_{index}$ is empty, variable $Prod = \prod_{j=index+1}^{n} L_j$
Function : $(\mathbf{R}, \mathbf{P}, index, Prod) = \text{Reorder}(\mathbf{R}, \bar{\mathbf{y}}_m, \beta)$

```
1: Set $\mathbf{P} = \mathbf{I}_n$, $index = m-1$, $Prod = 1$, $j = n$
2: while $j \ge m$ do
3:     Compute $L_j$ in (2.42) and set $p = j$
4:     for $i = m : j-1$ do
5:         Set $\mathbf{R}' = \mathbf{R}$ and interchange column $i$ and $j$ of $\mathbf{R}'$, and then compute the
       corresponding $L'_j$
6:         if $L'_j < L_j$ then Set $p = i$, $L_j = L'_j$
7:         end if
8:     end for
9:     if $p \ne j$ then Set $index = j$ and break the while loop
10:    end if
11:    Compute $\bar{z}_j = \lfloor \frac{\alpha - \sum_{l=i+1}^{n} |r_{ml}\bar{z}_l|}{|r_{m,j}|} \rceil|_{\bar{\mathcal{Z}}_j}$ with $\alpha$ defined in (2.39)
12:    Compute $Prod = Prod \times L_j$ and $j = j-1$
13: end while
```

We make some explanations on two outputs in Algorithm 2.6. The output *index* is the largest column number which have not been reordered. Another output *Prod* is the product of numbers of candidates at levels higher than *index*. To make the search process more efficient, we would like to get a larger *index* and smaller *Prod*.

In the following we describe the steps of the whole DTS reduction process:

1. We first compute the Householder OR decomposition with standard column pivoting to ensure that the first m columns of $\mathbf{R}$ are linearly independent, i.e., rank$(\mathbf{R}(:, 1 : m)) = m$. After this step, the last $n-m$ columns are as a group determined.

2. For $j = 1 : m$, we interchange columns j and m of $\mathbf{R}$ and bring $\mathbf{R}$ back to upper trapezoid by Givens rotations. Note that as before we need to simultaneously update $\bar{\mathbf{y}}$ using the same rotation. Then we use Algorithm 2.6 to reorder the last $n-m+1$ columns of the new $\mathbf{R}$ and get the output *index* and *Prod*. After those m steps, we

find the largest one among the m values of *index* and the corresponding order. If there are more than one orderings having the same *index*, we choose the one that gives the smallest *Prod*. Then if $index > m+1$, we reorder columns $m, m+1, ..., index-1$ of $\mathbf{R}$ such that $|r_{mm}| \leq |r_{m,m+1}| \leq ... \leq |r_{m,index-1}|$ since a larger $|r_{mj}|$ is likely to lead to a smaller L_j.

3. In this step we reorder the first $m-1$ columns of $\mathbf{R}$. Note that the last $n-m+1$ columns have been determined and we can get the first valid $\mathbf{z}_{(2)}$ by the DTS search algorithm. Then the UBILS problem becomes an OBILS problem and we can employ AIP reduction (see Algorithm 2.3) to reorder the first $m-1$ columns of $\mathbf{R}$.

The detailed pseudocode of the above reduction method is shown in Algorithm 2.7.

2.3.2 Lower Bounds

An example of the search tree for the DTS algorithm is depicted in Figure 2.1. Note that the root node does not correspond to any search operation and is an artificial node. In this example, the search tree has three different branches at level n (red, blue and green respectively). In order to improve the search efficiency, we should prune the tree so that less nodes will be traversed during search process. For example, if we can remove the green node at level n, the whole green branch can be deleted throughout the search phase.

Then, we introduce lower bound techniques that can help to prune the search tree for the DTS approach to UBILS problems. Suppose that we have computed the initial search radius β, (2.30) is equivalent to the following set of inequalities:

$$(\bar{y}_k - \sum_{j=k}^{n} r_{kj} z_k)^2 < \beta^2 - \sum_{i=k+1}^{m} (\bar{y}_i - \sum_{j=i}^{n} r_{ij} z_j)^2 - \sum_{i=1}^{k-1} (\bar{y}_i - \sum_{j=i}^{n} r_{ij} z_j)^2, \quad k = 1, 2, \ldots, m. \quad (2.43)$$

Note that when $k = m$ the second term on the right hand side of (2.43) vanishes and when $k = 1$ the third term vanishes. It is clear that the inequalities (2.43) are at least as tight as the corresponding inequalities (2.31) and (2.33) of the previous DTS algorithm. The last

Algorithm 2.7 DTS Reduction

```
Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{y} \in \mathbb{R}^n$ of the UBILS problem (2.23)and an initial search radius $\beta \in \mathbb{R}$
Output: Permutation matrix $\mathbf{P} \in \mathbb{R}^{n\times n}$, upper trapezoidal matrix $\mathbf{R} \in \mathbb{R}^{m\times n}$ and $\bar{\mathbf{y}} \in \mathbb{R}^m$ satisfying $\mathbf{AP} = \mathbf{QR}$ and $\bar{\mathbf{y}} = \mathbf{Q}^{\mathsf{T}}\mathbf{y}$ where $\mathbf{Q} \in \mathbb{R}^{m\times m}$ is orthogonal
Function : $(\mathbf{P}, \mathbf{R}, \bar{\mathbf{y}}) = $ DTS-Reduction$(\mathbf{A}, \mathbf{y}, \beta)$
1: Compute $\mathbf{AP} = \mathbf{QR}$ with standard pivoting and set $\bar{\mathbf{y}} = \mathbf{Q}^{\mathsf{T}}\mathbf{y}$, $index = 0$ and $Prod = \infty$
2: for $j = 1 : m$ do
3:     Set $\mathbf{R}' = \mathbf{R}$ and $\bar{\mathbf{y}}' = \bar{\mathbf{y}}$
4:     if $j \neq m$ then
5:         Interchange columns $j$ and $m$ of $\mathbf{R}'$ and transform $\mathbf{R}'$ to an upper trapezoidal matrix by Givens rotation. Apply the same Givens rotation to $\bar{\mathbf{y}}'$
6:     end if
7:     $(\mathbf{R}', \mathbf{P}', index_{tmp}, Prod_{tmp}) = $ Reorder$(\mathbf{R}', \bar{y}'_m, \beta)$ (See Algorithm 2.6)
8:     if $index_{tmp} > index$ then
9:         Set $index = index_{tmp}$, $p = j$, $\mathbf{R}_{tmp} = \mathbf{R}'$, $\bar{\mathbf{y}}_{tmp} = \bar{\mathbf{y}}'$, $\mathbf{P}_{tmp} = \mathbf{P}'$
10:    else if $index_{tmp} = index$ then
11:        if $Prod_{tmp} < Prod$ then
12:            Set $Prod = Prod_{tmp}$, $p = j$, $\mathbf{R}_{tmp} = \mathbf{R}'$, $\bar{\mathbf{y}}_{tmp} = \bar{\mathbf{y}}'$, $\mathbf{P}_{tmp} = \mathbf{P}'$
13:        end if
14:    end if
15: end for
16: if $p \neq m$ then
17:     Interchange columns $p$ and $m$ of $P$
18:     Set $\mathbf{P} = \mathbf{PP}_{tmp}$, $\mathbf{R} = \mathbf{R}_{tmp}$ and $\bar{\mathbf{y}} = \bar{\mathbf{y}}_{tmp}$
19: end if
20: if $index > m + 1$ then
21:     Reorder the columns of $\mathbf{R}_{:,m:index-1}$ such that $|r_{mm}| \leq |r_{m,m+1}| \leq \ldots \leq |r_{m,index-1}|$, and reorder the columns of $\mathbf{P}$ correspondingly
22: end if
23: Use the DTS search algorithm to find the first $\mathbf{z}_{(2)}$ and then use Algorithm 2.3 to reorder the first $m-1$ columns of $\mathbf{R}$, leading to final $\mathbf{R}$, $\bar{\mathbf{y}}$ and $\mathbf{P}$
```

term $\sum_{i=1}^{k-1}(\bar{y}_i - \sum_{j=i}^{n} r_{ij}z_j)^2$ in (2.43) is considered as zero in the inequalities (2.31) and (2.33). It is because during the search process, for example, at level k variables $z_{k-1}, \ldots, z_1$ have not been determined and thus $\sum_{i=1}^{k-1}(\bar{y}_i - \sum_{j=i}^{n} r_{ij}z_j)^2$ cannot be computed directly. However, if we can find a good lower bound T_k satisfying (for $k = m, m-1, \ldots, 2$):

$$T_k \leq \sum_{i=1}^{k-1}(\bar{y}_i - \sum_{j=i}^{n} r_{ij}z_j)^2 = \|\bar{\mathbf{y}}_{1:k-1} - \mathbf{R}_{1:k-1,k:n}\mathbf{z}_{k:n} - \mathbf{R}_{1:k-1,1:k-1}\mathbf{z}_{1:k-1}\|_2^2, \tag{2.44}$$

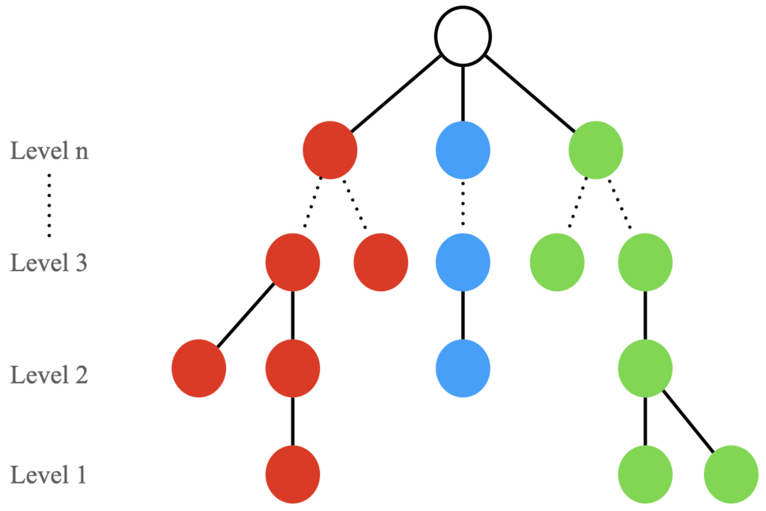

Figure 2.1: An example of search tree

we can clearly shrink the search region through replacing the inequalities (2.31) and (2.33) with the following inequalities:

$$(\bar{y}_k - \sum_{j=k}^{n} r_{kj} z_k)^2 < \beta^2 - \sum_{i=k+1}^{m} (\bar{y}_i - \sum_{j=i}^{n} r_{ij} z_j)^2 - T_k, \quad k = 1, 2, \ldots, m. \tag{2.45}$$

Obviously, the tighter T_k is, the more nodes we can prune on the search tree. However, if the cost of computing lower bounds is significant, we cannot achieve a good performance even if the lower bounds are tight. Thus there always exists a trade-off between the tightness and the computational cost of lower bounds. Note that $T_1, T_2, ..., T_{m-1}$ are different from T_m in that for the former the right hand side of (2.44) is an OBILS problem because $\mathbf{z}_{k:n}$ have already been determined, but for T_m the right hand side can be an UBILS problem since for example, when determining z_n at the top level, all variables from z_{n-1} to z_1 are still unknown.

There have been various methods that can compute lower bounds of OBILS problems (see e.g., [14, 36, 62]). The key ideas are as follows. To simplify notation, we consider lower bounds for the following OBILS model:

$$\min_{\mathbf{z}\in\mathcal{B}} \|\mathbf{y}-\mathbf{R}\mathbf{z}\|_2^2 \tag{2.46}$$

where $\mathbf{R} \in \mathbb{R}^{n\times n}$ has full column rank and other quantities are defined as before. Define $\mathbf{z}^{\text{RLS}} = \mathbf{R}^{-1}\mathbf{y}$ as the real least squares solution to (2.46) if $\mathcal{B}$ is replaced by $\mathbb{R}^n$. Now we would like to find a lower bound T such that:

$$0 \le T \le \|\mathbf{y}-\mathbf{R}\mathbf{z}\|_2^2. \tag{2.47}$$

Let $\mathbf{C} \in \mathbb{R}^{k\times n}$ be any arbitrary matrix. Then we have:

$$\|\mathbf{C}(\mathbf{z}-\mathbf{z}^{\text{RLS}})\|_2 = \|\mathbf{C}\mathbf{R}^{-1}(\mathbf{R}\mathbf{z}-\mathbf{y})\|_2 \le \|\mathbf{C}\mathbf{R}^{-1}\|_2\|\mathbf{y}-\mathbf{R}\mathbf{z}\|_2, \tag{2.48}$$

leading to a lower bound on the objective function of the OBILS problem (2.46):

$$\min_{\mathbf{z}\in\mathcal{B}} \|\mathbf{y}-\mathbf{R}\mathbf{z}\|_2^2 \ge \frac{\min_{\mathbf{z}\in\mathcal{B}} \|\mathbf{C}(\mathbf{z}-\mathbf{z}^{\text{RLS}})\|_2}{\|\mathbf{C}\mathbf{R}^{-1}\|_2}. \tag{2.49}$$

The inequality (2.49) gives a framework for a class of lower bounds. There are different lower bounds based on various choices of $\mathbf{C}$. In [62], Stojnic et al. proposed a norm-wise based lower bound where $\mathbf{C} = \mathbf{I}$. In [36], Garcia et al. proposed a component-wise based lower bound where $\mathbf{C} = \mathbf{e}_i^{\mathsf{T}}$ for $i = 1, 2, ..., n$. In [14], Buchheim et al. suggested a basis reduction based lower bound where $\mathbf{C} = \mathbf{t}_i^{\mathsf{T}} \in \mathbb{Z}^{1\times n}$ for $i = 1, 2, ..., n$ and $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_n] \in \mathbb{Z}^{n\times n}$ is a unimodular matrix which reduces the lattice basis matrix $\mathbf{R}^{-\mathsf{T}}$.

For more details on computing lower bounds for OBILS models (2.46), we refer to [78]. In Chapter 4, we will propose an effective approach for calculating lower bounds of UBILS problems.

2.3.3 Partial Regularization Algorithm

Different from the direct tree search algorithm introduced in Section 2.3.1, in this subsection, the partial regularization (PR) approach proposed in [20] will be presented to solve the following problem arising from multi-input multi-output (MIMO) communication applications:

$$\begin{gathered}\min_{\mathbf{x}\in\mathcal{X}_p^n} \|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2, \\ \mathcal{X}_p^n = \mathcal{X}_p \times \mathcal{X}_p \times \cdots \times \mathcal{X}_p, \quad \mathcal{X}_p = \{\pm 1, \pm 3, ..., \pm(2^p-1)\}.\end{gathered} \tag{2.50}$$

Note that (2.50) is the same as (2.23) except that the constraint box $\mathcal{B}$ is now replaced by $\mathcal{X}_p^n$. Partition $\mathbf{A}$ and $\mathbf{x}$ as follows:

$$\mathbf{A} = [\underset{m}{\mathbf{A}_1} \quad \underset{l}{\mathbf{A}_2}], \quad \mathbf{x} = \begin{bmatrix}\mathbf{x}_1 \\ \mathbf{x}_2\end{bmatrix} \tag{2.51}$$

where $l = n - m$, $\mathbf{x}_1 \in \mathcal{X}_p^m$ and $\mathbf{x}_2 \in \mathcal{X}_p^l$. Following [20], we can write $\mathbf{x}_2$ as a linear combination of $\mathbf{x}_2^{(i)} \in \mathcal{X}_1^l$ for $0 \le i \le p-1$:

$$\mathbf{x}_2 = \sum_{i=0}^{p-1} 2^i \mathbf{x}_2^{(i)}. \tag{2.52}$$

Define:

$$\bar{\mathbf{A}}_2 = \begin{bmatrix}\mathbf{A}_2 & 2\mathbf{A}_2 & \cdots & 2^{p-1}\mathbf{A}_2\end{bmatrix} \in \mathbb{R}^{m\times pl}, \quad \bar{\mathbf{x}}_2 = \begin{bmatrix}\mathbf{x}_2^{(0)} \\ \mathbf{x}_2^{(1)} \\ \vdots \\ \mathbf{x}_2^{(p-1)}\end{bmatrix} \in \mathbb{R}^{pl}. \tag{2.53}$$

Note that $\|\bar{\mathbf{x}}_2\|_2^2 = pl$ is a constant. Then we obtain an equivalent form of problem (2.50) as:

$$\min_{\mathbf{x}_1\in\mathcal{X}_p^m, \bar{\mathbf{x}}_2\in\mathcal{X}_p^l} \left\| \mathbf{y} - \begin{bmatrix}\mathbf{A}_1 & \bar{\mathbf{A}}_2\end{bmatrix} \begin{bmatrix}\mathbf{x}_1 \\ \bar{\mathbf{x}}_2\end{bmatrix} \right\|_2^2 + \alpha^2 \|\bar{\mathbf{x}}_2\|_2^2 \tag{2.54}$$

where α is a regularization parameter. Therefore with

$$\bar{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_1 & \bar{\mathbf{A}}_2 \\ \mathbf{0} & \alpha\mathbf{I} \end{bmatrix} \in \mathbb{R}^{(m+pl)\times(m+pl)}, \quad \bar{\mathbf{x}} = \begin{bmatrix} \mathbf{x}_1 \\ \bar{\mathbf{x}}_2 \end{bmatrix} \in \mathbb{R}^{(m+pl)}, \quad \bar{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{(m+pl)},$$
$$\bar{\mathcal{X}} = \left\{ \begin{bmatrix} \mathbf{x}_1 \\ \bar{\mathbf{x}}_2 \end{bmatrix} : \mathbf{x}_1 \in \mathcal{X}_p^m, \bar{\mathbf{x}}_2 \in \mathcal{X}_1^{pl} \right\}, \tag{2.55}$$

then the problem (2.54) can be written as:

$$\min_{\bar{\mathbf{x}}\in\bar{\mathcal{X}}} \|\bar{\mathbf{y}} - \bar{\mathbf{A}}\bar{\mathbf{x}}\|_2^2. \tag{2.56}$$

Obviously, (2.56) is an overdetermined ILS problem and we can solve it by slightly modifying the method introduced in Section 2.2.

It is difficult to extend the PR approach to a general UBILS problem, however, we can extend it to the UBILS problem (2.23) with the following box constraint:

$$\mathcal{B} = \{\mathbf{x} \in \mathbb{Z}^n, \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \mathbf{l}, \mathbf{u} \in \mathbb{Z}^n, u_j - l_j = 2^{p_j} - 1, p_j \in \mathbb{Z}^+, j = 1, 2, ..., n\}. \tag{2.57}$$

First, for each $x_j, j = m+1, m+2, ..., n$, we do the following shift:

$$\bar{x}_j = x_j - l_j \in \{0, 1, ..., u_j - l_j\}. \tag{2.58}$$

Since $\bar{x}_j$ is non-negative we can write it as a linear combination of $\bar{x}_j^{(i)} \in \{0,1\}$ for $i = 0, 1, ..., p_j - 1$:

$$\bar{x}_j = \sum_{i=0}^{p_j-1} 2^i \bar{x}_j^{(i)}. \tag{2.59}$$

Note that $\bar{x}_j^{(p_j-1)}\bar{x}_j^{(p_j)} \cdots \bar{x}_j^{(1)}\bar{x}_j^{(0)}$ is the binary representation of $\bar{x}_j$. Then we define:

$$\bar{\mathbf{A}}_2 = \begin{bmatrix} \mathbf{a}_{m+1}^{(0)} & \cdots & \mathbf{a}_{m+1}^{(p_{m+1}-1)} & \mathbf{a}_{m+2}^{(0)} & \cdots & \mathbf{a}_{m+2}^{(p_{m+2}-1)} & \cdots & \mathbf{a}_n^{(0)} & \cdots & \mathbf{a}_n^{(p_n-1)} \end{bmatrix} \tag{2.60}$$

$$\bar{\mathbf{x}}_2 = \begin{bmatrix} \bar{x}_{m+1}^{(0)} & \cdots & \bar{x}_{m+1}^{(p_{m+1}-1)} & \bar{x}_{m+2}^{(0)} & \cdots & \bar{x}_{m+2}^{(p_{m+2}-1)} & \cdots & \bar{x}_{n}^{(0)} & \cdots & \bar{x}_{n}^{(p_n-1)} \end{bmatrix}^{\mathsf{T}} \in \{0,1\}^q \tag{2.61}$$

where $\mathbf{a}_j^{(i)} = 2^i \mathbf{A}_{:,j}$ and $q = \sum_{j=m+1}^{n} p_j$. It is easy to see that the UBILS problem (2.23) with the constraint box (2.57) can be transformed to the following equivalent problem:

$$\min_{\mathbf{x}_1 \in \mathcal{B}_{1:m}, \bar{\mathbf{x}}_2 \in \{0,1\}^q} \left\| \mathbf{y} - \begin{bmatrix} \mathbf{A}_1 & \bar{\mathbf{A}}_2 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \bar{\mathbf{x}}_2 \end{bmatrix} \right\|_2^2 \tag{2.62}$$

Since $\|\bar{\mathbf{x}}_2 - (1/2)\mathbf{1}_{(q)}\|_2^2$ is a constant, the optimal solution to this problem is exactly the solution to the following OBILS problem:

$$\min_{\bar{\mathbf{x}} \in \bar{\mathcal{X}}} \|\bar{\mathbf{y}} - \bar{\mathbf{A}}\bar{\mathbf{x}}\|_2^2. \tag{2.63}$$

where

$$\begin{gathered} \bar{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_1 & \bar{\mathbf{A}}_2 \\ \mathbf{0} & \alpha \mathbf{I} \end{bmatrix} \in \mathbb{R}^{(m+q)\times(m+q)}, \quad \bar{\mathbf{x}} = \begin{bmatrix} \mathbf{x}_1 \\ \bar{\mathbf{x}}_2 \end{bmatrix} \in \mathbb{R}^{m+q}, \quad \bar{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ (1/2)\mathbf{1}_q \end{bmatrix} \in \mathbb{R}^{m+q}, \\ \bar{\mathcal{X}} = \left\{ \begin{bmatrix} \mathbf{x}_1 \\ \bar{\mathbf{x}}_2 \end{bmatrix} : \mathbf{x}_1 \in \mathcal{B}_{1:m}, \bar{\mathbf{x}}_2 \in \{0,1\}^q \right\}, \quad \alpha \text{ is a constant parameter.} \end{gathered} \tag{2.64}$$

This OBILS problem can be solved by the method introduced in Section 2.2.

2.3.4 Sub-optimal Methods

The DTS algorithm for UBILS problems discussed above usually costs a large amount of computation due to insufficient information to determine variables in the underdetermined part. Hence, researchers have investigated various sub-optimal methods with less complexity for UBILS problems. We fist introduce the so-called λ generalized sphere decoding (λ-GSD) approach proposed by Wang and Le-Ngoc in [68], since most sub-optimal methods are based on it.

The λ-GSD approach estimates $\mathbf{x}^*$ in the following linear model from communications:

$$\mathbf{y} = \mathbf{A}\mathbf{x}^* + \mathbf{v} \tag{2.65}$$

where $\mathbf{y} \in \mathbb{R}^n$ is an observation vector, $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m < n)$ is a channel matrix with full row rank, $\mathbf{x}^* \in \mathbb{Z}^n$ is an integer parameter vector and subject to the box constraint $\mathcal{B} = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{l}, \mathbf{u} \in \mathbb{Z}^n\}$, $\mathbf{v} \in \mathbb{R}^n$ is a noise vector following the normal distribution $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. In order to avoid solving the associated UBILS problem $\min\limits_{\mathbf{x}\in\mathcal{B}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2$ directly, λ-GSD partitions $\mathbf{A} = [\underset{m}{\mathbf{A}_1}, \underset{n-m}{\mathbf{A}_2}] \in \mathbb{R}^{m\times n}$ and $\mathbf{x}^* = [\underset{m}{\mathbf{x}_1^*}, \underset{n-m}{\mathbf{x}_2^*}] \in \mathbb{R}^n$. By introducing a trivial equation $\lambda\bar{\mathbf{x}}_2^* = \lambda\mathbf{x}_2^* - \lambda(\mathbf{x}_2^* - \bar{\mathbf{x}}_2^*)$ where $\bar{\mathbf{x}}_2^*$ is the mean of $\mathbf{x}_2^*$ and λ is a weighting factor. Then (2.65) becomes:

$$\begin{gathered}\widetilde{\mathbf{y}} = \widetilde{\mathbf{A}}\mathbf{x}^* + \widetilde{\mathbf{v}} \\ \widetilde{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \lambda\bar{\mathbf{x}}_2^* \end{bmatrix}, \quad \widetilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{0} & \lambda\mathbf{I} \end{bmatrix}, \quad \widetilde{\mathbf{v}} = \begin{bmatrix} \mathbf{v} \\ \lambda(\mathbf{x}_2^* - \bar{\mathbf{x}}_2^*) \end{bmatrix}.\end{gathered} \tag{2.66}$$

Typically, the noise vector $\mathbf{v}$ and parameter vector $\mathbf{x}_2^*$ are independent. Hence, $\widetilde{\mathbf{v}}$ is also a zero-mean noise vector with variance of $\text{diag}(\sigma^2, ..., \sigma^2, \lambda^2 P_{m+1}, ..., \lambda^2 P_n)$, where P_j represents the variance of the j-th element of $\mathbf{x}^*$ for $j = m+1, ..., n$. Consequently, estimating $\mathbf{x}^*$ in (2.66) is to solve the following ILS problem:

$$\min_{\mathbf{x}\in\mathcal{B}} \|\widetilde{\mathbf{y}} - \widetilde{\mathbf{A}}\mathbf{x}\|_2^2. \tag{2.67}$$

Clearly, (2.67) is an OBILS problem which can be solved by the approach discussed in Section 2.2. Compared with OBILS problems, UBILS problems are much more difficult to solve because at top level, we have to pick values for $(n-m+1)$ variables with information of only one inequality. Note that the transformed OBILS problem (2.67) is not mathematically equivalent to the original UBILS problem (2.23) unless the term $\|\mathbf{x}_2^* - \bar{\mathbf{x}}_2^*\|_2^2$ is constant, and hence the optimal solution to (2.67) is not necessarily to be that of (2.23). Note that our modified ADMM algorithm to be introduced in Chapter 4, which can also be viewed as a sub-optimal approach for UBILS problems, has its first iteration very similar to this λ-GSD approach. In [68], Wang and Le-Ngoc showed through numerical experiments on sphere decoding problems that for the λ-GSD approach, when the parameter λ goes beyond certain limits, the performances evaluated by symbol-error rate deteriorate significantly. Note that such limits are different for each specific problem. Such experimental results guide us to choose penalty parameter for our modified ADMM algorithm in Chapter 4.

Based on λ-GSD, researchers have proposed some other sub-optimal methods for UBILS problems with faster speed and less accuracy. In general, these methods mostly focus on replacing exhaustive search with partial search while solving the transformed OBILS problem (2.67) for less computational complexity. In [26], Datta et al. applied Tabu search, which is a heuristic method used to solve combinatorial optimization problems for further acceleration. In [54], Qian er al. divided the tree structure of the transformed OBILS problem (2.67) into two parts (level n to $m+1$ and level m to 1), and utilized an exhaustive tree search for the top part while a constrained tree search for the bottom part. In [55], Qian et al. proposed a way of adjusting the radius of the tree search based on the channel condition in order to avoid exhaustive search while determining $\mathbf{x}_{m+1;n}$ in (2.67). In [57], Qian et al. proposed another scheme that explores only a subset of candidates when determining $\mathbf{x}_{m+1:n}$. Note that all these sub-optimal approaches mentioned above have been shown to own faster speed and acceptable degradation of accuracy compared with the standard λ-GSD approach for UBILS problems arising from communication applications.

Chapter 3

ADMM for Integer Optimisation Problems

This chapter first reviews the standard form of ADMM in Section 3.1. Then in Section 3.2 we introduce ADMM as a heuristic approach to problems with discrete constraints and discuss techniques of extending ADMM to integer optimisation problems.

3.1 Standard ADMM Algorithm

ADMM is an approach that is intended to blend the decomposability of dual ascent with the superior convergence properties of the method of multipliers. For dual ascent and method of multipliers, we refer to Chapter 2 of [12]. The algorithm solves problems in the following form:

$$\begin{aligned} &\min\ f(\mathbf{x}) + g(\mathbf{z}) \\ &\text{s.t. }\ \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{c} \end{aligned} \tag{3.1}$$

where f and g are convex functions, variables $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{z} \in \mathbb{R}^m$, matrices $\mathbf{A} \in \mathbb{R}^{p\times n}$, $\mathbf{B} \in \mathbb{R}^{p\times m}$ and constant vector $\mathbf{c} \in \mathbb{R}^p$. We form the augmented Lagrangian:

$$L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y}) = f(\mathbf{x}) + g(\mathbf{z}) + \mathbf{y}^\mathsf{T}(\mathbf{Ax} + \mathbf{Bz} - \mathbf{c}) + (\rho/2)\|\mathbf{Ax} + \mathbf{Bz} - \mathbf{c}\|_2^2 \tag{3.2}$$

where $\rho > 0$ is the penalty parameter, $\mathbf{y} \in \mathbb{R}^p$ is the dual variable. Note that the augmented Lagrangian can be viewed as the standard Lagrangian associated with the problem:

$$\begin{aligned} &\min\ f(\mathbf{x}) + g(\mathbf{z}) + (\rho/2)\|\mathbf{Ax} + \mathbf{Bz} - c\|_2^2, \\ &\text{s.t. } \mathbf{Ax} + \mathbf{Bz} = \mathbf{c}. \end{aligned} \tag{3.3}$$

This problem is clearly equivalent to the original problem (3.1) because for any feasible $\mathbf{x}$ and $\mathbf{z}$ the term added to the objective function is zero. The associated dual objective function is $h_\rho(\mathbf{y}) = \inf_{(\mathbf{x},\mathbf{z})} L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y})$ and the dual problem is maximizing $h_\rho(\mathbf{y})$. The benefit of including the penalty term in (3.2) is that h_ρ can be shown to be differentiable under rather mild condition on the original problem.

In the method of multipliers, we first minimize the augmented Lagrangian jointly with respect to the two primal variables $\mathbf{x}$ and $\mathbf{z}$. After finding $(\mathbf{x}^+, \mathbf{z}^+) = \underset{\mathbf{x},\mathbf{z}}{\operatorname{argmin}}\, L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y})$, we solve the dual problem using gradient ascent with the step size of ρ where the gradient $\nabla h_\rho(\mathbf{y})$ can be evaluated as $\mathbf{Ax}^+ + \mathbf{Bz}^+ - \mathbf{c}$, which is the residual of the equality constraint. The method of multipliers consists of iterating the updates:

$$(\mathbf{x}^{(k+1)}, \mathbf{z}^{(k+1)}) = \underset{\mathbf{x},\mathbf{z}}{\operatorname{argmin}}\, L_\rho(\mathbf{x}, \mathbf{z}, \mathbf{y}^{(k)}) \tag{3.4}$$

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \rho(\mathbf{Ax}^{(k+1)} + \mathbf{Bz}^{(k+1)} - \mathbf{c}). \tag{3.5}$$

For ADMM, different from the method of multipliers, $\mathbf{x}$ and $\mathbf{z}$ are updated in an alternating fashion, which accounts for the term alternating direction. Separating the minimization over $\mathbf{x}$ and $\mathbf{z}$ into two steps is precisely what allows for decomposition when f

and g are separable. Thus ADMM consists of the iterations:

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}}{\operatorname{argmin}}\, L_\rho(\mathbf{x}, \mathbf{z}^{(k)}, \mathbf{y}^{(k)}) \tag{3.6}$$

$$\mathbf{z}^{(k+1)} = \underset{\mathbf{z}}{\operatorname{argmin}}\, L_\rho(\mathbf{x}^{(k+1)}, \mathbf{z}, \mathbf{y}^{(k)}) \tag{3.7}$$

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \rho(\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}). \tag{3.8}$$

We usually write ADMM in a slightly different form (scaled form) for convenience, by combining the linear and quadratic terms in the augmented Lagrangian and scaling the dual variable. Defining the residual $\mathbf{r} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}$ and the scaled dual variable $\mathbf{u} = (1/\rho)\mathbf{y}$, we have:

$$\mathbf{y}^\mathsf{T}\mathbf{r} + (\rho/2)\|\mathbf{r}\|_2^2 = (\rho/2)\|\mathbf{r} + \mathbf{u}\|_2^2 - (\rho/2)\|\mathbf{u}\|_2^2.$$

Then we can express ADMM as:

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}}{\operatorname{argmin}}(f(\mathbf{x}) + (\rho/2)\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z}^{(k)} - \mathbf{c} + \mathbf{u}^{(k)}\|_2^2) \tag{3.9}$$

$$\mathbf{z}^{(k+1)} = \underset{\mathbf{z}}{\operatorname{argmin}}(g(\mathbf{z}) + (\rho/2)\|\mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z} - \mathbf{c} + \mathbf{u}^{(k)}\|_2^2) \tag{3.10}$$

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c}. \tag{3.11}$$

Defining the residual at iteration k as $\mathbf{r}^{(k)} = \mathbf{A}\mathbf{x}^{(k)} + \mathbf{B}\mathbf{z}^{(k)} - \mathbf{c}$, we can see that $\mathbf{u}^{(k)} = \mathbf{u}^{(0)} + \sum_{j=1}^{k} \mathbf{r}^{(j)}$ is the running sum of residuals. We call (3.6)-(3.8) the unscaled form of ADMM and (3.9)-(3.11) the scaled form. The two are clearly equivalent.

ADMM is proved to possess convergence properties for convex problems under some mild assumptions. There are various convergence analyses in the literature (see e.g., Chapter 3 of [12] and [31]). In practice, ADMM can be slow to converge to high accuracy, but it is often the case that ADMM converges to modest accuracy within a few tens of iterations, which makes it practically useful in large-scale optimisation problems where modest accuracy is sufficient. Many variations on the classic ADMM algorithm have been explored in the literature, such as varying penalty parameter, other general augmenting terms, over-relaxation, inexact minimization and so on, and some of these methods can give superior convergence in practice compared to the standard ADMM presented above (see e.g., Chapter 3 of [12]).

Define the primal residual $\mathbf{r}$ and the dual residual $\mathbf{s}$ at iteration $k+1$ as:

$$\begin{aligned} \mathbf{r}^{(k+1)} &= \mathbf{A}\mathbf{x}^{(k+1)} + \mathbf{B}\mathbf{z}^{(k+1)} - \mathbf{c} \\ \mathbf{s}^{(k+1)} &= \rho \mathbf{A}^{\mathsf{T}}\mathbf{B}(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)}). \end{aligned} \tag{3.12}$$

These two residuals represent for primal and dual feasibility respectively. It can be proved that these two residuals converge to zero as ADMM converges and the necessary and sufficient optimality condition of ADMM on the problem (3.1) is that both primal and dual residual reach zero (see e.g., Chapter 3 of [12]). In practice, a reasonable termination criterion is that the primal and dual residuals are small, i.e., $\|\mathbf{r}^{(k+1)}\|_2^2 \leq \sigma_1$ and $\|\mathbf{s}^{(k+1)}\|_2^2 \leq \sigma_2$ where σ_1 and σ_2 are tolerances for the primal and dual feasibility.

The penalty parameter ρ in the ADMM algorithm (3.9)-(3.11) plays an important role in the practical performance of ADMM. In practice, this parameter, fixed over iterations, usually needs to be manually tuned by users for their particular problem instances. With the goal of making performances less dependent on the initial choice of the penalty parameter as well as accelerating convergence, an adaptive method has been proposed to automatically tune this parameter as the algorithm runs (see e.g., Chapter 3 of [12] and [76]). The scheme is as follows:

$$\rho^{(k+1)} = \begin{cases} \tau^{\text{incr}}\rho^{(k)} & \text{if } \|\mathbf{r}^{(k)}\|_2 > \mu\|\mathbf{s}^{(k)}\|_2 \\ \rho^{(k)}/\tau^{\text{decr}} & \text{if } \|\mathbf{s}^{(k)}\|_2 > \mu\|\mathbf{r}^{(k)}\|_2 \\ \rho^{(k)} & \text{otherwise} \end{cases} \tag{3.13}$$

where $\mathbf{r}^{(k)}$ and $\mathbf{s}^{(k)}$ are the primal and dual residual at iteration k defined in (3.12), $\mu > 1$, τ^{incr}, τ^{decr} are parameters. Typical choices might be $\mu = 10$ and $\tau^{\text{incr}} = \tau^{\text{decr}} = 2$. The idea behind this penalty parameter update is to try to keep the primal and dual residual norms within a factor of μ of one another as they both go to zero. The ADMM update equations (3.9)-(3.11) suggest that large values of ρ place a large penalty on violations of primal feasibility and so tend to produce small primal residuals. Conversely, the definition of $\mathbf{s}^{(k)}$ suggests that small values of ρ tend to reduce the dual residual, but at the expense of reducing the penalty on primal feasibility, which may result in a larger primal residual.

3.2 ADMM for Integer Optimisation Problems

In this section, we first show how ADMM can be applied to problems with discrete constraints and then focus on extending ADMM to integer optimisation problems.

3.2.1 ADMM as a Heuristic for Integer Problems

We now explore the use of ADMM for problems with discrete constraints. Note that in this case, ADMM need not converge and when it does converge, it need not converge to an optimal point. Consider the following constrained optimisation problem:

$$\begin{aligned} &\min\ f(\mathbf{x}) \\ &\text{s.t. } \mathbf{x} \in \mathcal{S} \end{aligned} \tag{3.14}$$

where f is a convex function and $\mathcal{S}$ is a discrete constraint set. Defining g as an indicator function of the constraint set $\mathcal{S}$:

$$g(\mathbf{z}) = \begin{cases} 0, & \text{if } \mathbf{z} \in \mathcal{S} \\ \infty, & \text{otherwise} \end{cases} \tag{3.15}$$

then (3.14) can be rewritten in the standard form (3.1) as:

$$\begin{aligned} &\min\ f(\mathbf{x}) + g(\mathbf{z}) \\ &\text{s.t. } \mathbf{x} - \mathbf{z} = \mathbf{0}. \end{aligned} \tag{3.16}$$

Applying the ADMM iteration formula (3.9)-(3.11) to this problem gives:

$$\mathbf{x}^{(k+1)} = \operatorname*{argmin}_{\mathbf{x}}(f(\mathbf{x}) + (\rho/2)\|\mathbf{x} - \mathbf{z}^{(k)} + \mathbf{u}^{(k)}\|_2^2) \tag{3.17}$$

$$\mathbf{z}^{(k+1)} = \Pi_{\mathcal{S}}(\mathbf{x}^{(k+1)} + \mathbf{u}^{(k)}) \tag{3.18}$$

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)} \tag{3.19}$$

where $\Pi_{\mathcal{S}}$ is the projector onto the set $\mathcal{S}$ and (3.18) is due to:

$$\mathbf{z}^{(k+1)} = \operatorname*{argmin}_{\mathbf{z}}(g(\mathbf{z})+(\rho/2)\|\mathbf{x}^{(k+1)}-\mathbf{z}+\mathbf{u}^{(k)}\|_2^2) = \operatorname*{argmin}_{\mathbf{z}\in\mathcal{S}} \|\mathbf{x}^{(k+1)}+\mathbf{u}^{(k)}-\mathbf{z}\|_2^2 = \Pi_{\mathcal{S}}(\mathbf{x}^{(k+1)}+\mathbf{u}^{(k)}).$$

Many hard problems can be written in the form (3.14), including maximum coverage problem, job selection, 3-satisfiability (see e.g., [28]), as well as the UBILS problem. For different problems, the discrete set $\mathcal{S}$ can take various forms. For the integer optimisation problems, the constraint set $\mathcal{S}$ becomes an integer set.

In [64], Takapoui et al. applied this ADMM algorithm (3.17)-(3.19) to mixed-integer quadratic programming (MIQP) problems, which are NP-complete and have the following form:

$$\begin{aligned} &\min\ (1/2)\mathbf{x}^{\mathsf{T}}\mathbf{P}\mathbf{x} + \mathbf{q}^{\mathsf{T}}\mathbf{x} + r \\ &\text{s.t.}\ \ \mathbf{A}\mathbf{x} = \mathbf{b} \\ &\qquad \mathbf{x} \in \mathcal{X} \end{aligned} \tag{3.20}$$

where $\mathbf{P} \in \mathbb{R}^{n\times n}$ is a symmetric positive semidefinite matrix, $\mathbf{q} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_n$, with each $\mathcal{X}_i$ being an integer set or a convex subset of $\mathbb{R}$. Rewriting the constraint of (3.20) as:

$$\begin{bmatrix}\mathbf{A}\\ \mathbf{I}\end{bmatrix}\mathbf{x} - \begin{bmatrix}\mathbf{0}\\ \mathbf{I}\end{bmatrix}\mathbf{z} = \begin{bmatrix}\mathbf{b}\\ \mathbf{0}\end{bmatrix}, \quad \mathbf{z} \in \mathcal{X}$$

and applying the ADMM approach with scaled dual variable $\mathbf{u} \in \mathbb{R}^{m+n}$ lead to the method for solving (3.20) given in [64], which iterates the following updates:

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}}{\operatorname{argmin}} \left((1/2)\mathbf{x}^\mathsf{T}\mathbf{P}\mathbf{x} + \mathbf{q}^\mathsf{T}\mathbf{x} + (\rho/2) \left\| \begin{bmatrix} \mathbf{A} \\ \mathbf{I} \end{bmatrix} \mathbf{x} - \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \mathbf{z}^{(k)} - \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix} + \mathbf{u}^{(k)} \right\|_2^2 \right) \tag{3.21}$$

$$\mathbf{z}^{(k+1)} = \Pi_{\mathcal{X}}(\mathbf{x}^{(k+1)} + [\underset{m}{\mathbf{0}}, \underset{n}{\mathbf{I}}]\mathbf{u}^{(k)}) \tag{3.22}$$

$$\mathbf{u}^{(k+1)} = \mathbf{u}^{(k)} + \begin{bmatrix} \mathbf{A} \\ \mathbf{I} \end{bmatrix} \mathbf{x}^{(k+1)} - \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \mathbf{z}^{(k+1)} - \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}. \tag{3.23}$$

Thus the $\mathbf{x}$-update is to solve a strongly convex quadratic programming problem in real space through solving a linear system and $\mathbf{z}$-update involves projection onto the set $\mathcal{X}$. For more details on solving this real quadratic programming problem, we refer to [64]. This MIQP example is illustrated here because it is easy to see that the UBILS problem (2.23) can also be written into the form of (3.20) by rewriting the objective function $\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 = \mathbf{x}^\mathsf{T}\mathbf{A}^\mathsf{T}\mathbf{A}\mathbf{x} - 2\mathbf{y}^\mathsf{T}\mathbf{A}\mathbf{x} + \mathbf{y}^\mathsf{T}\mathbf{y}$ and removing the linear constraint (note that the matrix $\mathbf{A}$ here is different from the one in (3.20)). Takapoui et al. showed in [64] through numerical experiments that this ADMM algorithm is an effective tool for MIQP problems arising from many embedded optimisation applications, such as hybrid vehicle control, power converter and control, since ADMM usually finds a feasible point with reasonable objective value and is substantially faster than global optimisation methods. They also applied this approach to a particular OBILS problem arising from message decoding in communications and shown that satisfactory bit error rates can be obtained with substantially less computing time than the relax-and-round method. However, in Chapter 5, our experimental results show that this approach in [64] may not be a good heuristic for the UBILS problem in that it hardly converges to the optimal solution. In Section 4.1, we will propose a modified ADMM algorithm, which is different from the approach in [64] in that ours imposes integer requirement for $\mathbf{x}$-update, and can be a much more effective heuristic for UBILS problems. For OBILS problems, our numerical experiments suggest that the modified ADMM is more accurate than the standard ADMM in [64], but it is not as fast as the CH search approach introduced in Section 2.3 and it does not always give the optimal solution like the CH search approach. Thus we will focus on UBILS problems in the following chapters.

3.2.2 ADMM for Boolearn Problems

For the problem (3.14), a special case is the constraint set $\mathcal{S}$ is or can be transformed to $\{\mathbf{x}|x_i \in \{0,1\}, \forall i\}$. It is a special case of integer optimisation problems and we refer to such kind of problems as boolean or binary optimisation problems. The boolean problems arise from many applications such as 3-SAT, image segmentation, feature selection and so on. The UBILS problem can be considered a boolean problem when the lower and upper bound satisfy $u_i - l_i = 1$ for $i = 1, 2, ..., n$.

An approach has been proposed to transform the boolean constraint into the intersection of two continuous constraints (see e.g., [33, 71]). Let $\mathbf{1}_{(n)}$ to denote the n-dimensional vector of ones, we can transform the constraint $\mathcal{S}$ into:

$$\mathcal{S} = \{0,1\}^n \Leftrightarrow \{\mathbf{x}|0 \leq x_i \leq 1\} \cap \left\{\mathbf{x}\middle|\|\mathbf{x} - (1/2)\mathbf{1}_{(n)}\|_p^p = n/4\right\}. \tag{3.24}$$

It means that the 0-1 constraint $\mathcal{S}$ can be equivalently replaced by the intersection of a box and a sphere (defined through the ℓ_p norm). Then we can infuse this equivalence into the ADMM framework, and ADMM update steps consist of manageable sub-problems in the continuous domain. Researchers have demonstrated its efficacy and effectiveness in [33, 71]. Note that this technique of transforming 0-1 integer constraint into equivalent continuous constraint can only be applied to certain UBILS problems where $u_i - l_i = 1$ for $i = 1, 2, .., n$. However, for such UBILS problems, since the penalty term $\|\mathbf{x} - (1/2)\mathbf{1}_{(n)}\|_2^2$ is a constant (x_i can only be 0 or 1 for $i = 1, 2, ..., n$) , a common approach is to add this penalty term to the objective function of UBILS problems and to transform UBILS problems into equivalent OBILS problems that are much easier to solve as we have discussed in Section 2.3.

Chapter 4

ADMM for UBILS Problems

In this chapter, we will first propose a modified ADMM algorithm, which can be viewed as an effective heuristic for the UBILS problems in Section 4.1.1, and then discuss a reasonable way of choosing parameters in Section 4.1.2. Later, in Section 4.2, we will incorporate ADMM to the tree search algorithm for the UBILS problems which can boost search efficiency.

4.1 Modified ADMM Algorithm

4.1.1 Modified Form

For the sake of readability, we rewrite the UBILS problem in (2.23) here:

$$\min_{\mathbf{x}\in\mathcal{B}} \|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 \tag{4.1}$$

where $\mathcal{B}=\{\mathbf{x}\in\mathbb{Z}^n : \mathbf{l}\leq\mathbf{x}\leq\mathbf{u}, \mathbf{l},\mathbf{u}\in\mathbb{Z}^n\}$ is the constraint box, $\mathbf{y}\in\mathbb{R}^m$, $\mathbf{A}\in\mathbb{R}^{m\times n}$ has full row rank with $m<n$. Applying the ADMM algorithm in (3.9)-(3.11), we have the formula

at step $k+1$:

$$\begin{aligned}
\mathbf{x}^{(k+1)} &= \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}}(\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 + (\rho/2)\|\mathbf{x}-\mathbf{z}^{(k)}+\mathbf{w}^{(k)}\|_2^2) \\
\mathbf{z}^{(k+1)} &= \Pi_{\mathcal{B}}(\mathbf{x}^{(k+1)}+\mathbf{w}^{(k)}) \\
\mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)}+\mathbf{x}^{(k+1)}-\mathbf{z}^{(k+1)}.
\end{aligned} \tag{4.2}$$

Here we solve a real least squares problem to get the real solution $\mathbf{x}^{(k+1)}$, compute $\mathbf{z}^{(k+1)}$ by projecting $\mathbf{x}^{(k+1)}+\mathbf{w}^{(k)}$ onto the constraint set $\mathcal{B}$ and then update the scaled dual variable $\mathbf{w}^{(k+1)}$.

In Chapter 5, we will show that the ADMM algorithm (4.2) given in [64] is not effective for the UBILS problem in that it can hardly give the optimal solution. Thus we propose to impose the integer requirement when updating $\mathbf{x}$. For simplicity, we replace ρ by $2\lambda^2$ ($\lambda>0$), and then ADMM for the UBILS problem consists of the following updates:

$$\begin{aligned}
\mathbf{x}^{(k+1)} &= \underset{\mathbf{x}\in\mathbb{Z}^n}{\operatorname{argmin}}(\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 + \lambda^2\|\mathbf{x}-\mathbf{z}^{(k)}+\mathbf{w}^{(k)}\|_2^2) \\
\mathbf{z}^{(k+1)} &= \Pi_{\mathcal{B}}(\mathbf{x}^{(k+1)}+\mathbf{w}^{(k)}) \\
\mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)}+\mathbf{x}^{(k+1)}-\mathbf{z}^{(k+1)}.
\end{aligned} \tag{4.3}$$

Since

$$\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 + \lambda^2\|\mathbf{x}-\mathbf{z}^{(k)}+\mathbf{w}^{(k)}\|_2^2 = \left\| \begin{bmatrix} \mathbf{y} \\ \lambda(\mathbf{z}^{(k)}-\mathbf{w}^{(k)}) \end{bmatrix} - \begin{bmatrix} \mathbf{A} \\ \lambda\mathbf{I} \end{bmatrix} \mathbf{x} \right\|_2^2 \tag{4.4}$$

it can be seen that in (4.3), the $\mathbf{x}$-update is to solve an OILS problem instead of a real least squares problem in the standard form, and thus we refer to it as the modified ADMM algorithm. The PLLL reduction and SE search algorithm for solving OILS problems have been introduced in Section 2.1. In general, solving an UBILS problem is much more difficult than solving an OILS problem mainly because, as discussed in Section 2.3, for the UBILS problem, the tree search method has to determine $(n-m+1)$ variables with the information of only one equation at the beginning, and the number of equations is larger than the number of variables while it is the opposite for the OILS problem. Besides, the efficient LLL reduction, which improves search speed, can only be applied to OILS problems as discussed in Chapter 2. Recall that the optimality condition of the standard ADMM on convex problems is that the primal residual and dual residual both reach zero, for the modified ADMM (4.3) on the UBILS problem it is clear that when the primal residual $\mathbf{r}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)}$ and the dual residual $\mathbf{s}^{(k+1)} = 2\lambda^2(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)})$ both reach zero, the algorithm stops ($\mathbf{x}$ and $\mathbf{z}$ update will not change their values over further iterations). We find that in practice if a constant λ is used over all iterations, the modified ADMM does not always converge and may end in an infinite loop, which means that the above conditions are not always achieved. We have observed that the parameter λ tends to trade off convergence and optimality: if λ is too small, the algorithm may not converge, which means that the primal and dual residual often cannot reach zero, but if the algorithm converges, the solution of the last iteration tends to have small objective value; and if λ is too large, the algorithm is more likely to converge but the last iteration usually corresponds to a larger objective value. It is reasonable because with larger λ, when updating $\mathbf{x}$ in (4.3) we penalize more the second term $\lambda^2\|\mathbf{x} - \mathbf{z}^{(k)} + \mathbf{w}^{(k)}\|_2^2$, making $\mathbf{x}^{(k+1)}$ get closer to $\mathbf{z}^{(k)} - \mathbf{w}^{(k)}$. Then in (4.3) $\mathbf{z}^{(k+1)}$ gets closer to $\mathbf{z}^{(k)}$ and $\mathbf{w}^{(k+1)}$ closer to zero, and in the following iteration, we have $\mathbf{x}^{(k+2)}$ closer to $\mathbf{z}^{(k+1)}$ and so does $\mathbf{z}^{(k+2)}$. In this way, larger λ helps the term $\mathbf{x}^{(k+2)} - \mathbf{z}^{(k+2)}$ and $\mathbf{z}^{(k+2)} - \mathbf{z}^{(k+1)}$ move toward to zero, and makes it easier for the algorithm to stop. Larger λ also means we put smaller weight on the first term $\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2$, which is the objective function of original UBILS problem, is likely to produce iterates resulting in larger objective values. In the next section, we will discuss how to set λ and get the final solution for the UBILS problem from iterates as well.

4.1.2 Parameter Setting

In this section, we will discuss how to set parameters including $\mathbf{z}^{(0)}$, $\mathbf{w}^{(0)}$ and penalty parameter λ when applying our modified ADMM algorithm to the UBILS problem.

It has been shown that in the convex case the choice of parameters in ADMM only affects the speed of the convergence, while in the non-convex case the choice can have a critical role in the quality of approximate solution, as well as the speed at which this solution is found (see e.g., [28, 75, 76]). Note that the integer constraint is a special type of non-convex constraint.

For $\mathbf{w}^{(0)}$, it is reasonable to set it as $\mathbf{0}_{(n)}$ because the scaled dual variable $\mathbf{w}$ represents the running sum of residuals and should start from $\mathbf{0}_{(n)}$. Then for $\mathbf{z}^{(0)}$ and λ, we first introduce the linear minimum mean square error estimator (LMMSE) (see e.g., [15]).

Throughout the rest of this thesis, we assume that the UBILS problem arises from estimating the integer parameter vector in the linear model (1.1). For the sake of convenience, we rewrite the linear model here:

$$\mathbf{y} = \mathbf{A}\mathbf{x}^* + \mathbf{v} \tag{4.5}$$

where $\mathbf{y} \in \mathbb{R}^m$ is an observation vector, $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m < n)$ is a real matrix with full row rank, $\mathbf{x}^*$ is assumed to be uniformly distributed over the discrete box $\mathcal{B} = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{l}, \mathbf{u} \in \mathbb{Z}^n\}$, $\mathbf{v} \in \mathbb{R}^n$ is a noise vector following the normal distribution $\mathcal{N}(\mathbf{0}, \sigma_{\mathbf{v}}^2\mathbf{I})$. Without loss of generality, we assume that $u_i - l_i$ are identical for $i = 1, 2, ..., n$. As discussed in Chapter 1, the maximum likelihood estimator (MLE) $\mathbf{x}^{\mathrm{MLE}}$ for the parameter vector $\mathbf{x}^*$ is the solution to the UBILS problem (4.5):

$$\mathbf{x}^{\mathrm{MLE}} = \underset{\mathbf{x}\in\mathcal{B}}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2. \tag{4.6}$$

Since sometimes MLE is too time consuming to calculate, one may use other estimators with worse performance but less computational cost in general. One such an estimator is linear minimum mean square error (LMMSE) estimator $\mathbf{x}^{\mathrm{LMMSE}}$, which is a solution of

$$\min_{\mathbf{x}=\mathbf{B}\mathbf{y}+\mathbf{c}} \ \mathrm{E}\{(\mathbf{x} - \mathbf{x}^*)(\mathbf{x} - \mathbf{x}^*)^{\mathsf{T}}\} \tag{4.7}$$

where $\mathbf{x} = \mathbf{B}\mathbf{y}+\mathbf{c}$ restricts $\mathbf{x}$ to be a linear function of $\mathbf{y}$. It can be shown that (see e.g., [15]):

$$\mathbf{x}^{\mathrm{LMMSE}} = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \frac{\sigma_{\mathbf{v}}^2}{\sigma_{\mathbf{x}}^2}\left\|\mathbf{x} - \frac{1}{2}(\mathbf{l} + \mathbf{u})\right\|_2^2 \tag{4.8}$$

where $\mathrm{E}(\mathbf{x}^*) = \frac{1}{2}(\mathbf{l}+\mathbf{u})$ (the center of the constraint box $\mathcal{B}$) and $\sigma_{\mathbf{x}}^2 = \frac{(u_i-l_i+1)^2-1}{12}$ is the variance of the i-th element of $\mathbf{x}^*$ (note that $u_i - l_i$ is independent of i according to our earlier assumption).

The LMMSE guides us to set $\mathbf{z}^{(0)}$ as $\frac{1}{2}(\mathbf{l}+\mathbf{u})$ and $\lambda^{(0)}$ as $\sqrt{\sigma_{\mathbf{v}}^2/\sigma_{\mathbf{x}}^2}$, and then $\mathbf{x}^{(1)}$ given by the modified ADMM in the first iteration is exactly the integer LMMSE of $\mathbf{x}^*$ in the model (4.5). Like that observed in [68] (see the discussion in Section 2.3.4), in our numerical tests, we find that if $\lambda^{(0)}$ is larger than $\sqrt{\sigma_{\mathbf{v}}^2/\sigma_{\mathbf{x}}^2}$, the accuracy of the solution drops dramatically. While $\alpha\sqrt{\sigma_{\mathbf{v}}^2/\sigma_{\mathbf{x}}^2}$ (α is a factor smaller than one) is as good as $\sqrt{\sigma_{\mathbf{v}}^2/\sigma_{\mathbf{x}}^2}$ in terms of accuracy. Therefore, we refer to $\sqrt{\sigma_{\mathbf{v}}^2/\sigma_{\mathbf{x}}^2}$ as a critical value for $\lambda^{(0)}$. The experimental verification will be shown in Chapter 5. We give some theoretical explanations here. Intuitively we prefer the modified ADMM algorithm stops with a small λ. Suppose that the modified ADMM algorithm stops at iteration k^*+1, which means that $\mathbf{x}^{(k^*+1)} = \mathbf{z}^{(k^*+1)}$ and $\mathbf{z}^{(k^*+1)} = \mathbf{z}^{(k^*)}$, then we have from the update formula (4.3):

$$\|\mathbf{y}-\mathbf{A}\mathbf{x}^{(k^*+1)}\|_2^2 + \lambda^2\|\mathbf{w}^{(k^*)}\|_2^2 \le \|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 + \lambda^2\|\mathbf{x}-\mathbf{x}^{(k^*+1)}+\mathbf{w}^{(k^*)}\|_2^2, \quad \forall \mathbf{x}\in\mathbb{Z}^n. \tag{4.9}$$

Let the optimal solution to the UBILS problem (4.1) be denoted by $\mathbf{x}^{\mathrm{MLE}}$, where MLE is for maximum-likelihood-estimator. Then we have:

$$\|\mathbf{y}-\mathbf{A}\mathbf{x}^{(k^*+1)}\|_2^2 - \|\mathbf{y}-\mathbf{A}\mathbf{x}^{\mathrm{MLE}}\|_2^2 \le -\lambda^2\|\mathbf{w}^{(k^*)}\|_2^2 + \lambda^2\|\mathbf{x}^{\mathrm{MLE}}-\mathbf{x}^{(k^*+1)}+\mathbf{w}^{(k^*)}\|_2^2. \tag{4.10}$$

Considering that $\mathbf{x}^{\mathrm{MLE}} \in \mathcal{B} = \{\mathbf{x}\in\mathbb{Z}^n : \mathbf{l}\le\mathbf{x}\le\mathbf{u}, \mathbf{l},\mathbf{u}\in\mathbb{Z}^n\}$, we can obtain an upper bound for the gap between the objective values at $\mathbf{x}^{(k^*+1)}$ and $\mathbf{x}^{\mathrm{MLE}}$:

$$\begin{aligned}&\|\mathbf{y}-\mathbf{A}\mathbf{x}^{(k^*+1)}\|_2^2 - \|\mathbf{y}-\mathbf{A}\mathbf{x}^{\mathrm{MLE}}\|_2^2\\ \le& -\lambda^2\|\mathbf{w}^{(k^*)}\|_2^2 + \lambda^2\sum_{i=1}^{n}[\max(|u_i - x_i^{(k^*+1)} + w_i^{(k^*)}|, |l_i - x_i^{(k^*+1)} + w_i^{(k^*)}|)]^2.\end{aligned} \tag{4.11}$$

Intuitively, the upper bound is likely to be tighter when λ is smaller, which means that more probably $\mathbf{x}^{(k^*+1)}$ can give a better objective value or more probably we can obtain $\mathbf{x}^{(k^*+1)} = \mathbf{x}^{\mathrm{MLE}}$ as the optimal solution.

It has been shown that using appropriate varying penalty parameter over iterations can make ADMM more efficient and effective to both convex and non-convex optimisation problems (see e.g., [12, 28, 75, 76]). The typical scheme used by the ADMM method for solving a general problem is given in (3.13). But for the modified ADMM algorithm for the UBILS problem, we propose to increase the value of penalty parameter λ for every certain number of iterations until the algorithm stops. The scheme is as follows:

$$\lambda^{(k+1)} = \begin{cases} \tau\lambda^{(k)} & \text{if } (k \bmod q) = 0 \\ \lambda^{(k)} & \text{otherwise} \end{cases} \tag{4.12}$$

where τ is a factor of inflating λ and $(k \bmod q)$ is the remainder of the division of the current iteration k by an integer parameter q. Note that when q is set as 1, it means that we inflate λ for every iteration. In the following, several reasons explaining such approach will be given. In Chapter 5, we will show through numerical experiments that for our modified ADMM on UBILS problems, this proposed way of varying λ outperforms fixed λ that is used in [64].

First, it is inspired by the quadratic penalty function method (see e.g., [7]), whose idea is to eliminate some or all of the constraints by adding to the objective function a penalty term which prescribes a high cost to infeasible points. Associated with this method is a penalty parameter λ, which determines the severity of the penalty and as a consequence the extent to which the resulting unconstrained problem approximates the original constrained problem. Thus the quadratic penalty method consists of solving a sequence of problems with $\{\lambda^{(k)}\}$ as a sequence of penalty parameters satisfying:

$$\forall k, \quad 0 < \lambda^{(k)} < \lambda^{(k+1)}, \quad \lambda^{(k+1)} \to \infty.$$

Similarly, for the modified ADMM algorithm, if such sequence of penalty parameter $\{\lambda^{(k)}\}$ is adopted, it can easily be seen that the algorithm is guaranteed to stop, which means that it will eventually achieve $\mathbf{x}^{(k+1)} = \mathbf{z}^{(k+1)}$ and $\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)}$. Otherwise, if λ does not change over iterations, we find that in practice the algorithm may never converge and be stuck in an endless loop. Specifically, we propose to increase the penalty parameter λ every certain number of iterations instead of every iteration because we would like the algorithm to stop with a relatively small λ, and the rational will be explained later in this section. Another advantage of keeping λ unchanged over some iterations is that we benefit from skipping the reduction phase while solving the associated OILS problem for $\mathbf{x}-$update in (4.3) since the channel matrix is the same as previous iteration.

Second, as we have discussed in Chapter 3, with the goal of making the performance less dependent on the initial choice of the penalty parameter and accelerating convergence as well, we would like to inflate the penalty parameter when the primal residual appears large compared to the dual residual, and deflate it when the primal residual seems too small relative to the dual residual. This scheme of varying the penalty parameter for the standard ADMM (3.13) suggests us to increase our penalty parameter $\lambda^{(k)}$ over iterations because for our modified ADMM algorithm on the UBILS problem, we find that in practice the primal residual $\mathbf{r}^{(k+1)} = \mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)}$ is much larger than the dual residual $\mathbf{s}^{(k+1)} = 2\lambda^2(\mathbf{z}^{(k+1)} - \mathbf{z}^{(k)})$ over iterations. Thus our proposed approach of varying λ (4.12) gives similar performances to the scheme (3.13) in practice, and we adopt (4.12) since it ensures that the algorithm stops while the scheme in literature (3.13) does not.

All in all, for parameter setting of the modified ADMM algorithm, we suggest to set $\mathbf{w}^{(0)}$ as $\mathbf{0}_{(n)}$, $\mathbf{z}^{(0)}$ as the center of the constraint box $\frac{\mathbf{l}+\mathbf{u}}{2}$ and $\lambda^{(0)}$ as $\alpha\sqrt{\sigma_{\mathbf{v}}^2/\sigma_{\mathbf{x}}^2}$ where α is a factor smaller than or equal to one, and to increase the value of penalty parameter λ by a factor of τ for every q iterations until the algorithm stops. The pseudocode of the modified ADMM algorithm on the UBILS problem can be found in Algorithm 4.1.

Here we give some explanations for some lines in Algorithm 4.1:

Algorithm 4.1 Modified ADMM

```
Input: The matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m < n)$ with full row rank, the vector $\mathbf{y} \in \mathbb{R}^m$, the lower bound vector $\mathbf{l} \in \mathbb{Z}^n$, the upper bound vector $\mathbf{u} \in \mathbb{Z}^n$, the variance of the noise vector in the linear model (4.5) $\sigma^2$
Output: The solution (not necessarily optimal) to the UBILS problem (4.1) $\mathbf{x}^{\text{MA}}$ and the associated residual norm $\beta = \|\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}\|_2$
Function: $(\mathbf{x}^{\text{MA}}, \beta) = \text{Modified-ADMM}(\mathbf{A}, \mathbf{y}, \mathbf{l}, \mathbf{u}, \sigma^2)$
1: Set $\mathbf{z}^{(0)} = \frac{\mathbf{l}+\mathbf{u}}{2}$, $\mathbf{w}^{(0)} = \mathbf{0}_{(n)} \in \mathbb{R}^n$, $\beta = \infty$
2: Select constants $\alpha$, $q$ and $\tau$ for initializing and updating penalty parameter $\lambda$
3: Select maximal number of iterations $K$
4: Compute var$\{X\}$ where $X \sim \mathcal{U}\{l_1, l_1+1, ..., u_1\}$ and set initial $\lambda = \alpha\sqrt{\frac{\sigma^2}{\text{var}\{X\}}}$
5: for $k = 1 : K$ do
6:     $\mathbf{x}^{(k)} = \underset{\mathbf{x}\in\mathbb{Z}^n}{\text{argmin}} \left\| \begin{bmatrix} \mathbf{y} \\ \lambda(\mathbf{z}^{(k-1)} - \mathbf{w}^{(k-1)}) \end{bmatrix} - \begin{bmatrix} \mathbf{A} \\ \lambda\mathbf{I} \end{bmatrix} \mathbf{x} \right\|_2^2$ (See Algorithm 2.1 and 2.2)
7:     $\mathbf{z}^{(k)} = \lfloor \mathbf{x}^{(k)} + \mathbf{w}^{(k-1)} \rceil_{\mathcal{B}}$
8:     $\mathbf{w}^{(k)} = \mathbf{w}^{(k-1)} + \mathbf{x}^{(k)} - \mathbf{z}^{(k)}$
9:     if $\beta > \|\mathbf{y} - \mathbf{A}\mathbf{z}^{(k)}\|_2$ then
10:        Set $\mathbf{x}^{\text{MA}} = \mathbf{z}^{(k)}$ and $\beta = \|\mathbf{y} - \mathbf{A}\mathbf{z}^{(k)}\|_2$
11:    end if
12:    if $\mathbf{x}^{(k)} = \mathbf{z}^{(k)} = \mathbf{z}^{(k-1)}$ then
13:        Terminate
14:    end if
15:    if $k \pmod{q} = 0$ then
16:        $\lambda = \lambda\tau$
17:        $\mathbf{w}^{(k)} = \mathbf{w}^{(k)}/\tau^2$
18:    end if
19: end for
```

- Line 2: α is the factor associated with initializing the penalty parameter λ. $\alpha < 1$ is for setting initial $\lambda^{(0)} = \alpha\sqrt{\sigma_{\mathbf{v}}^2/\sigma_{\mathbf{x}}^2}$ as we have discussed above. Note that if the variance is unknown or the UBILS problem does not arise form linear model (4.5), we find that in practice, it is usually acceptable to set initial λ as a small value (e.g., 1e-2) since λ will inflate over further iterations. The parameters $q \in \mathbb{Z}$ and $\tau > 1$ are for varying λ over iterations. We inflate λ by a factor of τ every q iterations.

- Line 6: Every $\mathbf{x}$-update is associated with solving an OILS problem since we have $\begin{bmatrix} \mathbf{A} \\ \lambda\mathbf{I} \end{bmatrix} \in \mathbb{R}^{(m+n)\times n}$, $\begin{bmatrix} \mathbf{y} \\ \lambda(\mathbf{z}^{(k-1)} - \mathbf{w}^{(k-1)}) \end{bmatrix} \in \mathbb{R}^{(m+n)}$. The PLLL reduction and SE search algorithm for solving OILS problems have been introduced in Section 2.1. Note that if λ is unchanged compared to last iteration, we can reuse the reduction result of last iteration and start the search phase directly.

- Lines 9–11: Since the objective function may not decrease monotonically, after each iteration, we check if the latest iterate $\mathbf{z}^{(k)}$ is better than the current best solution $\hat{\mathbf{x}}$. If so, we update $\hat{\mathbf{x}}$. This means that the final solution $\hat{\mathbf{x}}$ may not be the last iterate.

- Line 17: Recall that $\mathbf{w}$ is the scaled form of the dual variable. Therefore, after updating the penalty parameter λ, the scaled dual variable $\mathbf{w}$ must also be updated.

4.2 Using ADMM in the Tree Search

In this section, we incorporate the modified ADMM into the direct tree search (DTS) approach for UBILS problems in order to improve the search efficiency.

Note that the modified ADMM algorithm shown in Section 4.1 is a heuristic approach to the UBILS problem, which means that the optimal solution is not guaranteed as the DTS algorithm introduced in Section 2.3. We propose to employ the solution given by the modified ADMM as the initial point for the DTS algorithm.

Next, we show how to employ the modified ADMM algorithm to find lower bounds for both OBILS and UBILS problems. Suppose we would like to compute a lower bound on $\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2$, where $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m\times n}$ (m can be smaller, greater or equal to n), $\mathbf{x} \in \mathcal{B} = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \mathbf{l}, \mathbf{u} \in \mathbb{Z}^n\}$. Recall that our modified ADMM for solving $\min\limits_{\mathbf{x}\in\mathcal{B}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2$ consists of the following updates:

$$
\begin{aligned}
\mathbf{x}^{(k+1)} &= \underset{\mathbf{x}\in\mathbb{Z}^n}{\operatorname{argmin}}(\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \lambda^2\|\mathbf{x} - \mathbf{z}^{(k)} + \mathbf{w}^{(k)}\|_2^2) \\
\mathbf{z}^{(k+1)} &= \Pi_{\mathcal{B}}(\mathbf{x}^{(k+1)} + \mathbf{w}^{(k)}) \\
\mathbf{w}^{(k+1)} &= \mathbf{w}^{(k)} + \mathbf{x}^{(k+1)} - \mathbf{z}^{(k+1)}.
\end{aligned}
\tag{4.13}
$$

From $\mathbf{x}$-update in every iteration, $\mathbf{x}^{(k+1)}$ minimizes $(\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2+\lambda^2\|\mathbf{x}-\mathbf{z}^{(k)}+\mathbf{w}^{(k)}\|_2^2)$ in $\mathbb{Z}^n$. Thus,

$$\forall \mathbf{x}\in\mathbb{Z}^n,\ \|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 \geq -\lambda^2\|\mathbf{x}-\mathbf{z}^{(k)}+\mathbf{w}^{(k)}\|_2^2+\|\mathbf{y}-\mathbf{A}\mathbf{x}^{(k+1)}\|_2^2+\lambda^2\|\mathbf{x}^{(k+1)}-\mathbf{z}^{(k)}+\mathbf{w}^{(k)}\|_2^2. \tag{4.14}$$

Then we have the lower bound:

$$\begin{aligned}\forall \mathbf{x}\in\mathcal{B},\ \|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 \geq &-\lambda^2\sum_{i=1}^{n}[\max(|u_i-z_i^k+w_i^k|,|l_i-z_i^k+w_i^k|)]^2\\ &+\|\mathbf{y}-\mathbf{A}\mathbf{x}^{(k+1)}\|_2^2+\lambda^2\|\mathbf{x}^{(k+1)}-\mathbf{z}^{(k)}+\mathbf{w}^{(k)}\|_2^2.\end{aligned} \tag{4.15}$$

In Chapter 5, we will show that for UBILS problems such lower bounds can be computed within insignificant time compared to solving the corresponding UBILS problem and are relatively tight as well in general. The pseudocode of computing such ADMM-based lower bounds can be found in Algorithm 4.2.

Here we give some explanations for some lines in Algorithm 4.2:

- Line 2: If the entities $\mathbf{A},\mathbf{y},\mathbf{l},\mathbf{u}$ are from linear model (4.5), we set $\lambda^{(0)}$ as $\alpha\sqrt{\sigma_{\mathbf{v}}^2/\mathrm{var}\{X\}}$ as we have done in Algorithm 4.1. Otherwise, another strategy is to set $\lambda^{(0)}$ as a small value, i.e., 1e-3 is usually good in practice because on the one hand, from (4.15) it is clear that smaller λ may potentially lead to a tighter lower bound, and on the other hand, it is reasonable to start with a small λ since it will be inflated over iterations.

- Line 7: We update the lower bound if the lower bound obtained from the current iteration in (4.15) is larger than the existing one.

- Line 8: The algorithm terminates early if the current lower bound is larger than the maximal allowed residual since the current lower bound has already been sufficiently tight.

Hence, the modified ADMM algorithm can help us to find lower bounds for both OBILS and UBILS problems. Then we can obtain the lower bound T_m satisfying $T_m \leq \sum_{i=1}^{m-1}(\bar{y}_i -$

Algorithm 4.2 ADMM-based Lower Bound

Input: The matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ (either $m \geq n$ or $m < n$), the vector $\mathbf{y} \in \mathbb{R}^m$, the lower bound vector $\mathbf{l} \in \mathbb{Z}^n$, the upper bound vector $\mathbf{u} \in \mathbb{Z}^n$, the variance of the noise vector in the linear model (4.5) σ^2, the maximal possible residual res

Output: The lower bound lb of $\min_{\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^n} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2$ and $Flag$. If the lower bound is found to be larger than res, $Flag$ is 1 and otherwise 0

Function: $(lb, Flag) = \text{ADMM-LowerBound}(\mathbf{A}, \mathbf{y}, \mathbf{l}, \mathbf{u}, \sigma^2, res)$

```
1:  Set $\mathbf{z}^{(0)} = \frac{\mathbf{l}+\mathbf{u}}{2}$, $\mathbf{w}^{(0)} = \mathbf{0}_{(n)} \in \mathbb{R}^n$, $lb = 0$, $Flag = 0$
2:  Select $\lambda^{(0)}$ for initializing penalty parameter $\lambda$
3:  Select $q$ and $\tau$ for updating penalty parameter $\lambda$
4:  Select maximal number of iterations $K$
5:  for $k = 1 : K$ do
6:      $\mathbf{x}^{(k)} = \underset{\mathbf{x} \in \mathbb{Z}^n}{\operatorname{argmin}} \left\| \begin{bmatrix} \mathbf{y} \\ \lambda(\mathbf{z}^{(k-1)} - \mathbf{w}^{(k-1)}) \end{bmatrix} - \begin{bmatrix} \mathbf{A} \\ \lambda \mathbf{I} \end{bmatrix} \mathbf{x} \right\|_2^2$ (See Algorithm 2.1 and 2.2)
7:      $lb = \max(lb, \|\mathbf{y} - \mathbf{A}\mathbf{x}^{(k)}\|_2^2 + \lambda^2 \|\mathbf{x}^{(k)} - \mathbf{z}^{(k-1)} + \mathbf{w}^{(k-1)}\|_2^2 - \lambda^2 \sum_i \max((l_i - z_i^{(k-1)} + w_i^{(k-1)})^2, (u_i - z_i^{(k-1)} + w_i^{(k-1)})^2))$
8:      if $lb > res$ then
9:          $Flag = 1$
10:         Terminate
11:     end if
12:     $\mathbf{z}^{(k)} = \lfloor \mathbf{x}^{(k)} + \mathbf{w}^{(k-1)} \rceil_{\mathcal{B}}$
13:     $\mathbf{w}^{(k)} = \mathbf{w}^{(k-1)} + \mathbf{x}^{(k)} - \mathbf{z}^{(k)}$
14:     if $\mathbf{x}^{(k)} = \mathbf{z}^{(k)} = \mathbf{z}^{(k-1)}$ then
15:         Terminate
16:     end if
17:     if $k$ (mod) $q = 0$ then
18:         $\lambda = \lambda \times \tau$
19:         $\mathbf{w}^{(k)} = \mathbf{w}^{(k)} / \tau^2$
20:     end if
21: end for
```

$\sum_{j=i}^{n} r_{ij} z_j)^2$ in (2.44) and shrink the search region when determining $\mathbf{z}_{m:n}$ with the inequality set (2.45) to prune the search tree and improve search efficiency. The lower bound T_m is independent of any value of $\mathbf{z}_{m:n}$ we choose in the search process and it needs to be computed only once. In (2.45) we have the lower bounds $T_1, T_2, ..., T_m$, but here we only compute T_m because we observe that others are usually not worth calculating in terms of reducing the total cost. Note the UBILS problem is much more difficult to solve than the OBILS problem since we have to determine $(n - m + 1)$ variables with information of only one inequality.

Computing the lower bounds T_m is a way of including information of other levels into the top level when determining $\mathbf{z}_{m:n}$.

Suppose we are now in the underdetermined part during the search process for solving the reduced UBILS problem $\min\limits_{\mathbf{z}\in\bar{\mathcal{B}}} \|\bar{\mathbf{y}} - \mathbf{R}\mathbf{z}\|_2^2$ with entities defined the same as in (2.27) and the search radius of β, after choosing a $z_k \in \mathcal{Z}_k$ $(m+1 < k \leq n)$, the current value of $\mathbf{z}_{k:n}$ is known and the following inequality holds:

$$\min_{\mathbf{z}_{1:k-1}\in\bar{\mathcal{B}}_{1:k-1}} \|(\bar{\mathbf{y}} - \mathbf{R}_{1:m,k:n}\mathbf{z}_{k:n}) - \mathbf{R}_{1:m,1:k-1}\mathbf{z}_{1:k-1}\|_2^2 < \beta^2 \tag{4.16}$$

where $\bar{\mathcal{B}}_{1:k-1} = \{\mathbf{x} \in \mathbb{Z}^{k-1} : \bar{\mathbf{l}}_{1:k-1} \leq \mathbf{x} \leq \bar{\mathbf{u}}_{1:k-1}\}$. It is clear that the left hand side of the above inequality is an UBILS problem since $\mathbf{z}_{k:n}$ is known. Then we can employ Algorithm 4.2 to compute a lower bound T_k on the minimum of the objective function in (4.16). If T_k is found to be less than β^2, then the choice of z_k may be good and we can move down to choose z_{k-1}. Otherwise, it means that the choice of z_k is invalid and the branch $\mathbf{z}_{k:n}$ cannot be a part of the optimal solution and it should be pruned from the search tree. Then we should choose the next integer in $\mathcal{Z}_k$ for z_k. Note that the process of computing lower bounds can also provide solutions $\mathbf{z}_{1:k-1}$ ($\mathbf{z}_{k:n}$ has been fixed) to the original UBILS problems, but in practice we find that such solutions almost never help to reduce the search radius β and thus we do not make the use of this information for simplicity.

The marginal benefits of such pruning nodes techniques decrease with k because for example in Figure 2.1, considering that computing a lower bound at levels n and $n-1$ take similar computation time, if we can prune the green node at level n then we remove the whole green branch in the search tree, but when lower bound techniques help us to prune a green node at level $n-1$ we can only remove a part of the green branch. Thus in practice, we select $level_{\text{lb}}$ for calculating lower bound when determining $z_n, z_{n-1}, ..., z_{n-level_{\text{lb}}+1}$. In general, $level_{\text{lb}}$ satisfies $0 \leq level_{\text{lb}} \leq n-m$. When $level_{\text{lb}} = 0$ we do not apply such lower bound techniques and when $level_{\text{lb}} = n-m$, it means that such techniques are employed for determining $z_n, z_{n-1}, ..., z_{m+1}$.

In all, we have discussed how the modified ADMM algorithm can be incorporated into the direct tree search approach for UBILS problems, which includes giving initial points and computing lower bounds to shrink search region and to prune nodes. We refer to our new algorithm for the UBILS problem (4.1) as ADMM tree search and its pseudocode can be found in Algorithm 4.3.

Here we give some explanations for some lines in Algorithm 4.3 that make it different from Algorithm 2.5:

- Line 2: We employ the modified ADMM appoach in Algorithm 4.1 to calculate the initial point and search radius. If the UBILS problem (4.1) does not arise from the linear model (4.5) or the noise variance is unknown, we can also set the initial λ in Algorithm 4.1 as a small value, i.e., 1e-2 or 1e-3 as we have discussed above. If there is no point found within this search radius β in the following tree search process, then $\mathbf{x}^{\text{UBILS}}$ is proved to be the optimal solution.

- Line 5: We use Algorithm 4.2 to compute the lower bound T_m as defined in (2.44) to shrink the search region before determining $\mathbf{z}_{m:n}$. Note that T_m must not be greater than β^2.

- Line 6: We have discussed above that the marginal benefits of using lower bounds to prune nodes decrease as level k goes down. Below certain level, the costs of computing lower bounds may be even higher than the benefits of removing branches. Hence it is reasonable to select a value for $level_{\text{lb}}$ so that we compute lower bounds only when determining $\mathbf{z}_{n-level_{\text{lb}}+1:n}$.

- Line 8-20: When the lower bounds fail to prune nodes, we may have to compute lower bounds for more nodes when moving down to the next level. Thus the costs of computing lower bounds may increase significantly. To avoid such snowball effect, we adopt an adaptive strategy of computing lower bounds: we compute lower bounds at level n for each $\bar{z}_n \in \bar{\mathcal{Z}}_n$ where $\bar{\mathcal{Z}}_n$ is the feasible set defined in (2.41), and if those lower bounds can help to prune a certain percentage ($\gamma = 50\%$ in our experiments) of nodes we continue to compute lower bounds at level $n-1$, otherwise we discard the lower bound techniques since they may not be effective.

- Line 9 and 23: Note that now when computing $\bar{\lambda}_k$ and $\bar{\mu}_k$ for $k = n, ..., m$, since we have computed the lower bound T_m, β in (2.41) becomes $\sqrt{\beta^2 - T_m}$.

- Line 15 and 50-62 (Step 6): When determining $z_n, ..., z_{n-level_{\text{lb}}+1}$, we first compute a lower bound using Algorithm 4.2 as in (4.16). If such lower bound is greater than β^2 ($flag = 1$), it means that the choice of z_k is invalid and we move to the next integer in $\mathcal{Z}_k$.

Algorithm 4.3 ADMM Tree Search

Input: The matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ ($m < n$) with full row rank, the vector $\mathbf{y} \in \mathbb{R}^m$, the lower bound vector $\mathbf{l} \in \mathbb{Z}^n$, the upper bound vector $\mathbf{u} \in \mathbb{Z}^n$, the variance of the noise vector in the linear model (4.5) σ

Output: The solution to the UBILS problem (4.1) $\mathbf{x}^{\text{UBILS}}$

Function: $\mathbf{x}^{\text{UBILS}} = \text{ADMM-TreeSearch}(\mathbf{A}, \mathbf{y}, \mathbf{l}, \mathbf{u}, \sigma^2)$

```
1: **Step** 1(Initialization and Reduction)
2:     $(\mathbf{x}^{\text{UBILS}}, \beta) = \text{Modified-ADMM}(\mathbf{A}, \mathbf{y}, \mathbf{l}, \mathbf{u}, \sigma^2)$ (See Algorithm 4.1)
3:     $(\mathbf{P}, \mathbf{R}, \bar{\mathbf{y}}) = \text{DTS-Reduction}(\mathbf{A}, \mathbf{y}, \beta)$ (See Algorithm 2.7)
4:     Set $\bar{\mathbf{l}} = \mathbf{P}^\mathsf{T}\mathbf{l}$, $\bar{\mathbf{u}} = \mathbf{P}^\mathsf{T}\mathbf{u}$, $\mathbf{z}^{\text{UBILS}} = \mathbf{P}^\mathsf{T}\mathbf{x}^{\text{UBILS}}$
5:     $(T_m, \sim) = \text{ADMM-LowerBound}(\mathbf{R}_{1:m-1,1:n}, \bar{\mathbf{y}}_{1:m-1}, \bar{\mathbf{l}}, \bar{\mathbf{u}}, \beta^2)$ (See Algorithm 4.2)
6:     Select $level_{\text{lb}}$ for calculating lower bound with the goal of pruning nodes early when
       determining $z_n, z_{n-1}, ..., z_{n-level_{\text{lb}}+1}$
7:     Set $k = n$
8:     Create an array $Flags_n$ of size $\bar{u}_n - \bar{l}_n + 1$ and fill it with initial values of 0
9:     Compute $\bar{\lambda}_n$ and $\bar{\mu}_n$ by following (2.41). Note that $\beta$ in that formula is now replaced
       by $\sqrt{\beta^2 - T_m}$ and set $\bar{\mathcal{Z}}_n = \{0, 1, ..., \bar{u}_n - \bar{l}_n\} \cap (\bar{\lambda}_n, \bar{\mu}_n)$
10:    **if** $\bar{\mathcal{Z}}_n$ is empty **then** Terminate
11:    **else**
12:        **for** $\bar{z}_n \in \bar{\mathcal{Z}}_n$ **do**
13:            Transform $\bar{z}_n$ back to $z_n$ using (2.37)
14:            Set $\mathbf{R}' = \mathbf{R}_{1:m,1:n-1}$, $\bar{\mathbf{y}}' = \bar{\mathbf{y}} - \mathbf{R}_{1:m,n}\mathbf{z}_n$, $\bar{\mathbf{l}}' = \bar{\mathbf{l}}_{1:n-1}$, $\bar{\mathbf{u}}' = \bar{\mathbf{u}}_{1:n-1}$
15:            $(\sim, Flags_n[\bar{z}_n]) = \text{ADMM-LowerBound}(\mathbf{R}', \bar{\mathbf{y}}', \bar{\mathbf{l}}', \bar{\mathbf{u}}', \beta^2)$ (See Algorithm 4.2)
16:        **end for**
17:        **if** number of entries with the value of 1 in $Flags_n < \gamma|\bar{\mathcal{Z}}_n|$ **then**
18:            $level_{\text{lb}} = 0$
19:        **end if**
20:    **end if**
21: **end Step** 1
```

```
22: Step 2
23:     Compute $\bar{\lambda}_k$ and $\bar{\mu}_k$ by following (2.41). Note that $\beta$ in that formula is now replaced
    by $\sqrt{\beta^2 - T_m}$. Set $\bar{\mathcal{Z}}_k = \{0, 1, ..., \bar{u}_k - \bar{l}_k\} \cap (\bar{\lambda}_k, \bar{\mu}_k)$
24:     if $\bar{\mathcal{Z}}_k$ is empty then Go to Step 4
25:     else
26:         Compute $c_k = \frac{\alpha - \sum_{j=k+1}^{n} |r_{mj}| \bar{z}_j}{|r_{mk}|}$, $\bar{z}_k = \lfloor c_k \rceil |_{\bar{\mathcal{Z}}_k}$
27:         Go to Step 6
28:     end if
29: end Step 2
30: Step 3
31:     if $k > m - 1$ then Set $k = k - 1$ and go to Step 2
32:     else
33:         Transform $\bar{\mathbf{z}}_{m:n}$ back to $\mathbf{z}_{m:n}$ using (2.37)
34:         Compute $\widetilde{\mathbf{y}} = \bar{\mathbf{y}}_{1:m-1} - \mathbf{R}_{1:m-1,m:n}\mathbf{z}_{m:n}$ and $T = (\bar{y}_m - \mathbf{R}_{m,m:n}\mathbf{z}_{m:n})^2$
35:         Compute $\mathbf{z}_{1:m-1}$ = CH-Search($\mathbf{R}_{1:m-1,1:m-1}, \widetilde{\mathbf{y}}, \bar{\mathbf{l}}_{1:m-1}, \bar{\mathbf{u}}_{1:m-1}, \sqrt{\beta^2 - T}$) (See
    Algorithm 2.4)
36:         Set $\mathbf{x}^{\text{UBILS}} = \mathbf{Pz}$ and $\beta = \sqrt{(\widetilde{\mathbf{y}} - \mathbf{R}_{1:m-1,1:m-1}\mathbf{z}_{1:m-1})^2 + T}$
37:     end if
38: end Step 3
39: Step 4
40:     if $k = n$ then Terminate
41:     else
42:         Set $k = k + 1$
43:     end if
44: end Step 4
45: Step 5
46:     Choose $\bar{z}_k \in \bar{\mathcal{Z}}_k$ to be the next nearest integer to $c_k$
47:     if $\bar{z}_k$ does not exist then Go to Step 4
48:     end if
49: end Step 5
50: Step 6
51:     if $k > n - level_{\text{lb}}$ then
52:         if $k = n$ then $Flag = Flags_n[\bar{z}_n]$
53:         else
54:             Transform $\bar{\mathbf{z}}_{k:n}$ back to $\mathbf{z}_{k:n}$ using (2.37)
55:             Set $\mathbf{R}' = \mathbf{R}_{1:m,1:k-1}$, $\bar{\mathbf{y}}' = \bar{\mathbf{y}} - \mathbf{R}_{1:m,k:n}\mathbf{z}_{k:n}$, $\bar{\mathbf{l}}' = \bar{\mathbf{l}}_{1:k-1}$, $\bar{\mathbf{u}}' = \bar{\mathbf{u}}_{1:k-1}$
56:             $(\sim, Flag)$ = ADMM-LowerBound($\mathbf{R}', \bar{\mathbf{y}}', \bar{\mathbf{l}}', \bar{\mathbf{u}}', \beta^2$) (See Algorithm 4.2)
57:         end if
58:         if $Flag = 1$ then Go to Step 5
59:         end if
60:     end if
61:     Go to Step 3
62: end Step 6
```

Chapter 5

Numerical Experiments

In this chapter, we will demonstrate the effectiveness and efficiency of the modified ADMM and ADMM tree search approach to the UBILS problem proposed in Chapter 4 through numerical experiments. In Section 5.1, how we set up the experiments will be introduced and in Section 5.2, comparisons among our proposed algorithms and existing algorithms as well as selected commercial optimisation packages will be conducted in terms of both computational cost and accuracy.

Our proposed algorithms to be tested are implemented in MATLAB 2019a and all tests are run on a laptop with 2.3 GHz 8-Core Intel i9 CPU, 16 GB memory and Mac OS.

5.1 Experiment Setup

We rewrite the linear model:

$$\mathbf{y} = \mathbf{A}\mathbf{x}^* + \mathbf{v} \tag{5.1}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m < n)$ is a real matrix with full row rank, $\mathbf{y} \in \mathbb{R}^m$ is an observation vector, $\mathbf{x}^* \in \mathbb{Z}^n$ is an integer parameter vector and is subject to the box constraint $\mathcal{B} = \{\mathbf{x} \in \mathbb{Z}^n : \mathbf{l} \le \mathbf{x} \le \mathbf{u}, \mathbf{l}, \mathbf{u} \in \mathbb{Z}^n\}$, $\mathbf{x}^*$ is assumed to be uniformly distributed inside the box $\mathcal{B}$, $\mathbf{v} \in \mathbb{R}^n$ is a noise vector following the normal distribution $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$. The maximum likelihood estimator $\mathbf{x}^{\text{MLE}}$ for the parameter vector $\mathbf{x}^*$ in model (5.1) is the optimal solution to the following UBILS problem:

$$\mathbf{x}^{\text{MLE}} = \operatorname*{argmin}_{\mathbf{x} \in \mathcal{B}} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2. \tag{5.2}$$

We introduce some MATLAB built-in functions which will be used in our experiments: `randn`(p,q) generates a $p \times q$ matrix of normally distributed random numbers; `randi`$([i,j],p,q)$ generates a $p \times q$ matrix of random integers drawn from the discrete uniform distribution on the interval $[i,j]]$; `ones`(p,q) generates a $p \times q$ matrix of ones. We consider two different cases:

Case 1 (random case): We use $\sigma *$`randn`$(m,1)$ to generate a m-dimensional noise vector $\mathbf{v}$, use `ones`$(n,1)*l$ and `ones`$(n,1)*u$ to generate constraint box $[\mathbf{l}, \mathbf{u}]$ where $l, u \in \mathbb{Z}$ and use `randi`$([l,u],n,1)$ to generate the true parametric vector $\mathbf{x}^*$. Then $\mathbf{y}$ can be generated by following (5.1) after $\mathbf{A}$ is generated by `randn`(m,n).

Case 2 (MIMO flat-fading in communications) We consider a real-world application in this case. In the multi-input multi-output (MIMO) linear flat-fading channel system, the relation between received signal vectors and transmit signal vectors of this system can be written as a complex linear system:

$$\mathbf{y}_c = \mathbf{A}_c \mathbf{x}_c^* + \mathbf{v}_c \tag{5.3}$$

where $\mathbf{A}_c \in \mathbb{C}^{N_r \times N_t}$ represents the channel matrix with N_t transmitter antennas and N_r receiver antennas. The elements of $\mathbf{A}_c$ are complex i.i.d Gaussian variables with distribution $\mathcal{CN}(\mathbf{0}, \mathbf{I})$ and $\mathbf{v}_c \in \mathbb{C}^{N_r}$ is the white Gaussian noise vector with distribution $\mathcal{CN}(\mathbf{0}, 2\sigma^2\mathbf{I})$. The elements of the unknown vector $\mathbf{x}_c^*$ are odd numbers in set $\mathcal{X}(p) = \{p_1 + p_2 j : p_1, p_2 = \pm 1, \pm 3, ..., \pm(2^p - 3), \pm(2^p - 1)\}$ where $j^2 = -1$, $p = 1, 2, 3$ correspond to 4QAM, 16QAM, 64QAM constellations respectively. QAM stands for quadrature amplitude modulation, which is a modulation scheme used by network operators for transmitting data. To deal with the complex case, we first transform (5.3) into a real linear model. Let

$$\mathbf{A}_c = \mathbf{A}_c^R + j\mathbf{A}_c^I, \quad \mathbf{y}_c = \mathbf{y}_c^R + j\mathbf{y}_c^I, \quad \mathbf{x}_c^* = \mathbf{x}_c^{*R} + j\mathbf{x}_c^{*I}, \quad \mathbf{v}_c = \mathbf{v}_c^R + j\mathbf{v}_c^I.$$

Then (5.3) is equivalent to the real linear model:

$$\mathbf{y} = \mathbf{A}\mathbf{x}^* + \mathbf{v} \tag{5.4}$$

where

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_c^R \\ \mathbf{y}_c^I \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \mathbf{A}_c^R & -\mathbf{A}_c^I \\ \mathbf{A}_c^I & \mathbf{A}_c^R \end{bmatrix}, \quad \mathbf{x}^* = \begin{bmatrix} \mathbf{x}_c^{*R} \\ \mathbf{x}_c^{*I} \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} \mathbf{v}_c^R \\ \mathbf{v}_c^I \end{bmatrix}. \tag{5.5}$$

Thus $\mathbf{A} \in \mathbb{R}^{m \times n}$ with $m = 2N_r$, $n = 2N_t$ and $a_{ij} \sim \mathcal{N}(0, 1/2)$, $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, $\mathbf{x}^* \in \mathcal{X}(p)^n = \mathcal{X}(p) \times \mathcal{X}(p) \times ... \times \mathcal{X}(p)$ with:

$$\mathcal{X}(p) = \{\pm 1, \pm 3, ..., \pm(2^p - 3), \pm(2^p - 1)\} \tag{5.6}$$

and estimating $\mathbf{x}_c^*$ in (5.3) is equivalent to estimating $\mathbf{x}^*$ in (5.4).

In this case, we can construct the matrix $\mathbf{A}$ by setting $\mathbf{A}_c^R = \frac{1}{\sqrt{2}}\texttt{randn}(N_r, N_t)$, $\mathbf{A}_c^I = \frac{1}{\sqrt{2}}\texttt{randn}(N_r, N_t)$. Then we show how to construct $\mathbf{x}^*$. First we generate each element of vector $\bar{\mathbf{x}}^*$ as $\bar{x}_i^* = 2 * \texttt{randi}([1, 2^{p-1}]) - 1$. Then x_i^* can be generated as $x_i^* = (3 - 2^{\texttt{randi}([1,2])}) * \bar{x}_i^*$ for $i = 1, 2, ..., n$. The method of generating $\mathbf{y}$ and $\mathbf{v}$ is the same as Case 1. To get the maximum likelihood estimator of $\mathbf{x}^*$, we solve the following problem:

$$\min_{\mathbf{x} \in \mathcal{X}(p)^n} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2. \tag{5.7}$$

Note that the above problem is not a standard UBILS problem since the constraint on $\mathbf{x}$ is not a box. However, we can transform it into a standard UBILS problem by defining for $i = 1, 2, ..., n$ the following transformation:

$$\bar{x}_i = \frac{2^p - 1 + x_i}{2} \tag{5.8}$$

so that $\bar{x}_i \in \bar{\mathcal{X}}(p) = \{0, 1..., 2^p - 1\}$. Then the problem (5.7) becomes a standard UBILS problem:

$$\min_{\bar{\mathbf{x}} \in \bar{\mathcal{X}}(p)} \|\bar{\mathbf{y}} - \bar{\mathbf{A}}\bar{\mathbf{x}}\|_2^2 \tag{5.9}$$

where $\bar{\mathbf{y}} = \mathbf{y} + (2^p - 1)\mathbf{A}\mathbf{1}_{(n)}$, $\bar{\mathbf{A}} = 2\mathbf{A}$.

Here we give the information of computing the associated signal-to-noise-ratios (SNR):

$$\text{SNR} = 10\log_{10}\left(\frac{(M-1)/3}{2\sigma^2}\right) \tag{5.10}$$

which will be used later for M-QAM (see e.g., [78]).

We will use the existing MATLAB code for the Partial LLL reduction and SE search algorithm on OILS problems from:

https://www.cs.mcgill.ca/~chang/MILES_routine1.php.

We use the MATLAB code for the AIP reduction and CH search on the OBILS problem to implement the PR approach from:

https://www.cs.mcgill.ca/~chang/MILES_routine3.php.

5.2 Simulation Results

5.2.1 Performance of Modified ADMM with Different Parameters

In this section, we demonstrate the effectiveness of our parameter setting strategies for Modified ADMM. We focus on Case 1 random generated problems in various scenarios throughout this section.

We consider the UBILS problem (5.2) and generate 100 random instances of Case 1 for each scenario as we have discussed in Section 5.1. These instances are generated with fixed $m = 15$, $n = 20$, $l = 0$, $u = 10$ and varying noise standard deviation $\sigma = 0.1 : 0.1 : 0.5$ as well as $\sigma = 0.01$.

In running Modified ADMM (Algorithm 4.1), we choose values for the four parameters α, q, τ and K. The parameter $\alpha = \lambda^{(0)}/\lambda^*$, where $\lambda^{(0)}$ is the initial value of λ, and $\lambda^* = \sqrt{\sigma^2/\text{var}\{X\}}$ with $\text{var}\{X\} = ((u-l+1)^2-1)/12 = 10$, and we take $\alpha = 0.2, 0.5, 1, 2, 5$, leading to five different initial values for $\lambda^{(0)}$. We take $q = 2$, $\tau = 1.05$ and $K = 200$, see Line 2, Line 2, and Line 3 of Algorithm 4.1, respectively.

In our tests, we record the experimental probability of the first iterate $\mathbf{z}^{(1)}$ equal to the ILS solution $\mathbf{x}^{\text{ILS}}$, denoted by $\Pr(\mathbf{z}^{(1)} = \mathbf{x}^{\text{ILS}})$, the experimental probability of the final output $\mathbf{x}^{\text{MA}}$ equal to the ILS solution $\mathbf{x}^{\text{ILS}}$, denoted by $\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$, and the average number of iterations over 100 tries under different scenarios. The optimal solution $\mathbf{x}^{\text{ILS}}$ to the UBILS problem, which is the maximum likelihood estimator of $\mathbf{x}^*$ in (5.1), can be found by the DTS algorithm (Algorithm 2.5). The number of iterations is the number of steps Modified ADMM takes to find the final output $\mathbf{x}^{\text{MA}}$ and if the algorithm fails to stop within the maximal number of iterations K, the number of iterations is just K. The experimental results are shown in Table 5.1. Note that bold values in tables of this thesis represent for the best results (e.g., in Table 5.1 they mark the highest probability and least number of iterations). For comparison, we also give the results for the fixed λ strategy (i.e., λ is not changed in Algorithm 4.1) in the last three columns of Table 5.1. Specifically, we take $\lambda = 0.5\lambda^*, \lambda^*, 2\lambda^*$.

As the goal of the application is to estimate the true integer parameter vector $\mathbf{x}^*$ in (5.4), we also record the experimental probability that $\mathbf{x}^{\text{MA}}$ is equal to $\mathbf{x}^*$, denoted by $\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^*)$, and the experimental probability that $\mathbf{x}^{\text{ILS}}$ is equal to $\mathbf{x}^*$, denoted by $\Pr(\mathbf{x}^{\text{ILS}} = \mathbf{x}^*)$ over the 100 tries. Such probabilities are referred to as the success rates in the literature. The results are displayed in Table 5.2.

Some interesting points can be seen from from Tables 5.1 and 5.2:

1. The performance of Modified ADMM drops as the noise standard deviation σ increases, and when $\sigma = 0.01$ and $\sigma = 0.1$ Modified ADMM can usually give the optimal solution in only one iteration.

Table 5.1: Performance of Modified ADMM ($K = 200, q = 2, \tau = 1.05$)

σ		α					λ/λ^*		
		0.2	0.5	1	2	5	0.5	1	2
0.01	$\Pr(\mathbf{z}^{(1)} = \mathbf{x}^{\text{ILS}})$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	# of iterations	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**
0.1	$\Pr(\mathbf{z}^{(1)} = \mathbf{x}^{\text{ILS}})$	**1.00**	**1.00**	**1.00**	**1.00**	0.95	**1.00**	**1.00**	**1.00**
	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**1.00**	**1.00**	**1.00**	**1.00**	0.95	**1.00**	**1.00**	**1.00**
	# of iterations	**2.00**	**2.00**	**2.00**	**2.00**	2.41	**2.00**	**2.00**	**2.00**
0.2	$\Pr(\mathbf{z}^{(1)} = \mathbf{x}^{\text{ILS}})$	0.87	**0.99**	**0.99**	0.97	0.34	**0.99**	**0.99**	0.97
	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**1.00**	**1.00**	**1.00**	0.97	0.48	**1.00**	**1.00**	0.99
	# of iterations	4.21	**2.20**	2.70	2.67	6.42	2.24	2.34	2.52
0.3	$\Pr(\mathbf{z}^{(1)} = \mathbf{x}^{\text{ILS}})$	0.40	0.75	**0.87**	0.66	0.07	0.75	**0.87**	0.66
	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	0.90	0.90	**0.93**	0.76	0.14	0.83	**0.93**	0.84
	# of iterations	27.34	11.41	**5.51**	7.02	6.69	35.29	17.01	7.02
0.4	$\Pr(\mathbf{z}^{(1)} = \mathbf{x}^{\text{ILS}})$	0.10	0.41	**0.54**	0.26	0.01	0.41	**0.54**	0.26
	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	0.73	**0.74**	0.73	0.37	0.08	0.46	0.73	0.43
	# of iterations	58.67	29.92	14.49	9.90	**5.91**	111.74	46.61	23.56
0.5	$\Pr(\mathbf{z}^{(1)} = \mathbf{x}^{\text{ILS}})$	0.02	0.17	**0.29**	0.14	0.00	0.17	**0.29**	0.14
	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**0.51**	0.48	0.45	0.23	0.04	0.25	0.42	0.28
	# of iterations	77.07	43.69	18.26	9.51	**4.85**	152.15	70.54	28.47

Table 5.2: Success rates of $\mathbf{x}^{\text{MA}}$ and $\mathbf{x}^{\text{ILS}}$

σ	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^*)$						$\Pr(\mathbf{x}^{\text{ILS}} = \mathbf{x}^*)$
	α					$\lambda = \lambda^*$	
	0.2	0.5	1	2	5		
0.01	1.00	1.00	1.00	1.00	1.00	1.00	1.00
0.1	1.00	1.00	1.00	1.00	0.95	1.00	1.00
0.2	1.00	1.00	1.00	0.97	0.48	1.00	1.00
0.3	0.90	0.89	0.90	0.75	0.14	0.90	0.95
0.4	0.67	0.66	0.67	0.33	0.07	0.64	0.76
0.5	0.28	0.29	0.29	0.18	0.02	0.28	0.42

2. In general, compared with varying λ with $\alpha = 1$ or $\alpha = 0.5$ (i.e., $\lambda^{(0)} = \lambda^*$ or $\lambda^{(0)} = 0.5\lambda^*$, the fixed λ strategy with $\lambda = \lambda^*$ or $\lambda = 0.5\lambda^*$ needs more iterations and gives slightly worse results. We noticed that for fixed $\lambda = \lambda^*$, the algorithm sometimes fails to converge (e.g., when $\sigma = 0.5$, 32% instances do not stop within 200 iterations.) while for varying λ with different α, the algorithm always converges for all instances. Thus, these test results favor varying λ.

3. For each σ, $\Pr(\mathbf{z}^{(1)} = \mathbf{x}^{\text{ILS}})$ is the highest when $\alpha = 1$. This is understandable because when $\alpha = 1$, $\lambda^{(0)} = \lambda^*$ and $\mathbf{x}^{(1)}$ is exactly the integer LMMSE estimator as we have discussed in Section 4.1.2, while $\mathbf{z}^{(1)}$ is the projection of $\mathbf{x}^{(1)}$ into $\mathcal{B}$ (note that $\mathbf{z}^{(0)} = (\mathbf{l} + \mathbf{u})/2$ and $\mathbf{w}^{(0)} = \mathbf{0}$)

4. From number of iterations, we can see that larger α leads to faster convergence but worse results especially when σ is large. In contrast, smaller α typically needs more iterations but leads to more accurate results. We also observe in the experiments that for the fixed λ strategy, the probability of $\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}}$ drops significantly especially for large σ when the fixed λ moves away from λ^*. Thus if λ^* is unknown, to avoid fine-tuning λ as the fixed λ strategy requires, we can employ the varying λ strategy with a relatively small $\lambda^{(0)}$ with $\alpha = 1, 0.5$ or 0.2. If we know the noise variance is relatively high, we can set α as 0.2 or 0.5.

5. The trend of the success rate $\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^*)$ is similar to that of $\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$. In addition, the optimal solution $\mathbf{x}^{\text{ILS}}$, which is both the maximum a posterior estimator and the maximum likelihood estimator of $\mathbf{x}^*$, gives the highest success rate, which decreases when σ increases.

From Table 5.1 and Table 5.2, we see $\alpha = 1$ is a usually good choice. Then we run Modified ADMM on the same test instances with fixed $\alpha = 1$ but now we take different $\tau = 1.02, 1.05, 1.1, 1.2, 1.5$, $q = 1, 2, 4, 8$, and record $\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$ and the average number of iterations. The results are displayed in Table 5.3 and 5.4.

In the following we give some observations and comments.

1. We notice that there exists a general trend: if we increase λ more slowly over iterations (i.e., choose smaller τ and larger q), we may (not necessarily) get higher probability of obtaining the optimal solution $\mathbf{x}^{\text{ILS}}$ but need more iterations. The performance of Modified ADMM is less sensitive on τ and q than on α and when σ is small, changing τ and q hardly influences the results.

Table 5.3: Performance of Modified ADMM ($K = 200, q = 2, \alpha = 1$ and varying τ)

σ		τ				
		1.02	1.05	1.1	1.2	1.5
0.01	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	# of iterations	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**
0.1	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
	# of iterations	**2.00**	**2.00**	**2.00**	**2.00**	**2.00**
0.2	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**1.00**	**1.00**	0.99	0.99	0.99
	# of iterations	2.63	2.70	2.53	2.35	**2.16**
0.3	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**0.94**	0.93	0.93	0.89	0.88
	# of iterations	6.63	5.51	4.66	4.56	**3.56**
0.4	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**0.77**	0.73	0.61	0.56	0.54
	# of iterations	20.28	14.49	12.32	10.01	**6.89**
0.5	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	0.44	**0.45**	0.40	0.35	0.30
	# of iterations	29.32	18.26	13.38	10.37	**7.81**

Table 5.4: Performance of Modified ADMM ($K = 200, \tau = 1.05, \alpha = 1$ and varying q)

σ		q			
		1	2	4	8
0.01	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**1.00**	**1.00**	**1.00**	**1.00**
	# of iterations	**2.00**	**2.00**	**2.00**	**2.00**
0.1	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	**1.00**	**1.00**	**1.00**	**1.00**
	# of iterations	**2.00**	**2.00**	**2.00**	**2.00**
0.2	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	0.99	**1.00**	**1.00**	**1.00**
	# of iterations	**2.51**	2.70	2.68	3.82
0.3	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	0.92	0.93	0.93	**0.94**
	# of iterations	**4.75**	5.51	5.97	6.93
0.4	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	0.62	0.73	**0.78**	0.76
	# of iterations	**12.08**	14.49	19.04	26.27
0.5	$\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$	0.41	0.45	**0.48**	**0.48**
	# of iterations	**13.33**	18.26	29.96	40.57

2. Our numerical tests indicate that there is no fixed choice for the parameters that can work well for all problems. For each specific problem, to achieve high probability of obtaining the optimal solution, we can fine-tune these three parameters α, τ and q (e.g., for the case with $\sigma = 0.4$ and 0.5, if we choose $\alpha = 0.5$, $\tau = 1.05$, $q = 3$ and $\alpha = 0.2$, $\tau = 1.05$, $q = 4$ we can further increase $\Pr(\mathbf{x}^{\text{MA}} = \mathbf{x}^{\text{ILS}})$ to 0.81 and 0.57 respectively).

3. In the ADMM tree search approach, which first uses Modified ADMM to get the initial point, we will not spend efforts on fine-tuning these parameters for each specific problem. Instead, as long as the choice of parameters usually gives acceptable results, we use it.

5.2.2 Comparisons of Modified ADMM and the Original ADMM

We compare Modified ADMM with the original ADMM algorithm in [64] (see Section 3.2.1). Note that we have λ as the parameter of Modified ADMM, and the penalty parameter of the original ADMM algorithm $\rho = 2\lambda^2$ in (3.21) is set as fixed $2(\lambda^*)^2$ over iterations where λ^* has the same definition as in Section 5.2.1. The main difference between these two is that our Modified ADMM imposes the integer constraint while updating $\mathbf{x}$ in each iteration but the original one computes $\mathbf{x}$ in the real space. The performances are evaluated in terms of how average objective values or residuals change over iterations. We generate 100 experimental instances for each scenario with fixed $m = 15$, $n = 20$, $l = 0$, $u = 10$ and $\sigma = 0.01, 0.1, 0.5$. We set $K = 200$, $\alpha = 1$, $\tau = 1.05$ and $q = 2$ for Modified ADMM, which has been shown before to be a good choice. The results of average objective value versus 200 iterations are illustrated in Figures 5.1-5.3. Note that if the algorithm stops before iteration 200, we keep its current objective value when the algorithm converges until iteration 200. To see how good the two algorithms, we also plot the average optimal objective value, which is obtained via computing the optimal solutions of the UBILS problems by the DTS approach.

It can be seen that for these cases, Modified ADMM is clearly much better than the original one. When the noise increases, although Modified ADMM may take more iterations before convergence, it can eventually give solutions with much smaller objective value than the original ADMM can achieve. For these cases, Modified ADMM can give the optimal solution for 100%, 100% and 45% instances respectively while the original ADMM algorithm for mere 7%, 8% and 3% instances for $\sigma = 0.01, 0.1, 0.5$. In Figure 5.1 and 5.2, the objective value of Modified ADMM coincides with the optimal value because for this case, Modified ADMM gives the optimal solution for all instances in only one iteration.

5.2.3 Performance of ADMM-based Lower Bounds

We check the tightness of ADMM-based lower bounds for UBILS problems that are proposed in Section 4.2. The test set for each scenario includes 100 random Case 1 problems with fixed $m = 15, l = 0$ and varying u, n, σ. When we run ADMM-based lower bound (Algorithm 4.2), we need to choose values for four parameters $\lambda^{(0)}$, q, τ and maximal number of iterations K (see Line 2, 3, 3 and 4 in Algorithm 4.2 respectively). Different from previous experiments, we prefer smaller λ when computing lower bounds as discussed in Chapter 4. Thus we set $\lambda^{(0)} = 0.02\lambda^*$, where $\lambda^* = \sqrt{\sigma^2/\text{var}\{X\}}$ is the same as before. We then set q and τ as 2 and 1.5, K as 20, making λ vary from $0.02\lambda^*$ to roughly $1.2\lambda^*$ in the last iteration. Note that we set less number of iterations and larger τ than Modified ADMM because in general calculating lower bounds needs less accuracy than obtaining a solution.

The results are shown in Table 5.5. Note that in this table, "minimal value" refers to the average optimal objective value of UBILS problems which can be computed through the

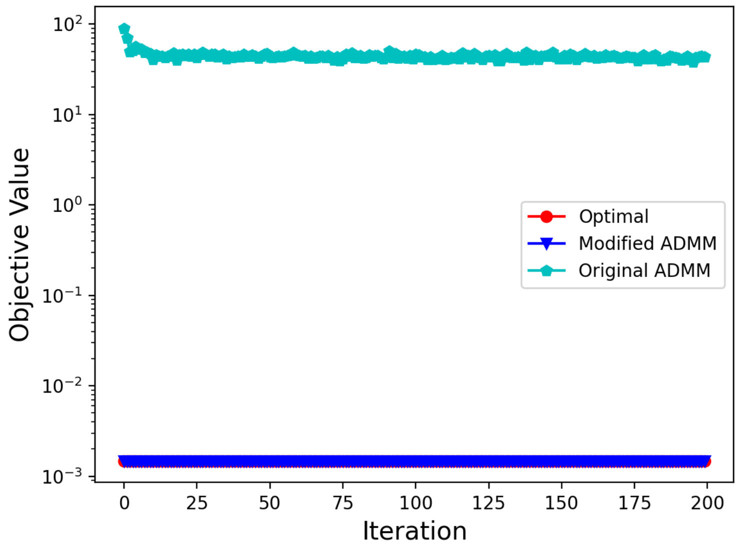

Figure 5.1: Comparison between modified and original ADMM for $\sigma = 0.01, m = 15, n = 20, l = 0, u = 10$

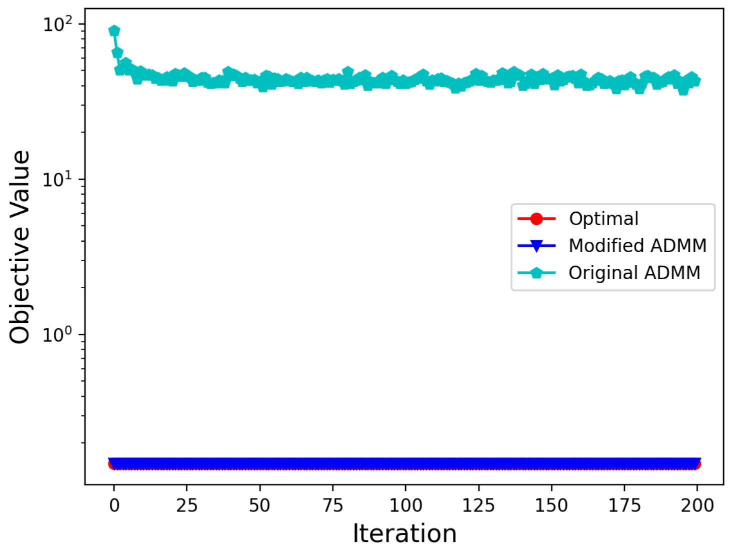

Figure 5.2: Comparison between modified and original ADMM for $\sigma = 0.1, m = 15, n = 20, l = 0, u = 10$

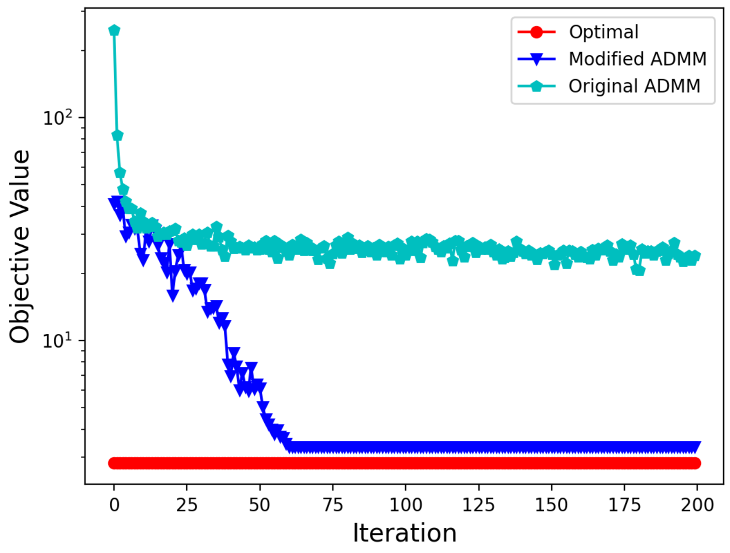

Figure 5.3: Comparison between modified and original ADMM with $\sigma = 0.5, m = 15, n = 20, l = 0, u = 10$

DTS approach, "lower bound" refers to the average calculated lower bounds for the objective value of UBILS problems, and "ratio" is the value of $\frac{\text{lower bound}}{\text{minimal value}}$.

From Table 5.5 we can see that the ADMM-based lower bounds are extremely tight when σ is relatively small and the tightness is hardly affected by the length of constraint box $(u-l)$ and the number of underdetermined levels $(n-m)$. However, when the noise increases, the effectiveness of such lower bounds for UBILS problems drops. As for the time cost, computing such a lower bound takes significantly less time than solving the corresponding UBILS problem via the DTS approach (e.g., for the case $m = 15, n = 20, \sigma = 0.1, u = 10$, the latter needs around 400 times more CPU time than the former and the ratio enlarges as the UBILS problem becomes more difficult to solve).

Table 5.5: ADMM-based lower bounds for UBILS problems

varying u and fixed $n = 20, \sigma = 0.1$					
u	2	4	6	8	10
minimal value	0.1479	0.1479	0.1479	0.1479	0.1479
lower bound	0.1478	0.1478	0.1476	0.1475	0.1472
ratio	99.93%	99.93%	99.78%	99.73%	99.53%

varying n and fixed $u = 10, \sigma = 0.1$					
n	16	17	18	19	20
minimal value	0.1561	0.1556	0.1433	0.1557	0.1479
lower bound	0.1560	0.1555	0.1431	0.1555	0.1472
ratio	99.94%	99.94%	99.86%	99.87%	99,53%

varying σ and fixed $n = 20, u = 10$					
σ	0.1	0.2	0.3	0.4	0.5
minimal value	0.1479	0.5916	1.3213	2.1385	2.8261
lower bound	0.1472	0.5392	0.8236	0.9066	0.9006
ratio	99.53%	91.14%	62.33%	42.39%	31.87%

5.2.4 Comparisons of ADMM Search and DTS

In this section, we compare our ADMM tree search algorithm (Algorithm 4.3) with the DTS algorithm (see [19], [78])[1] to show that how techniques proposed in Chapter 4 can improve the search efficiency. Note that for all the following simulation, we employ MATLAB Coder toolbox to generate C functions from the original MATLAB code for all algorithms

[1]The MATLAB implementation is available from https://www.cs.mcgill.ca/ ~chang/MILES_routine4.php

because all commercial optimisation packages that we will use later are implemented in C/C++.

The test set includes 100 Case 1 random instances with $l = 0$, $m = 15$, $n = 20$, two different u (10 and 20) and three different σ (0.01, 0.1 and 0.5). For convenience, we use the following abbreviations for different solvers:

DTS: The direct tree search algorithm (see Algorithm 2.7 for the reduction part and 2.5 for the search part) with top level search reordering technique in [78].

ADMM-Search-1: **DTS** with initial point and radius given by Modified ADMM.

ADMM-Search-2: **ADMM-Search-1** plus ADMM-based lower bound techniques proposed in Section 4.2. Note that for pruning nodes, we compute lower bounds for half of the underdetermined part (e.g., when $m = 15$ and $n = 20$, we have $level_{\text{lb}} = 2$ and calculate lower bounds when determining z_{20} and z_{19}), and the reason has been explained in Section 4.2.

ADMM-Search-3: Algorithm 4.3. Note that it is different from **ADMM-Search-2** in that it removes the top level search reordering technique in [78]. We check whether this technique will help improve search efficiency for our ADMM tree search algorithm.

Note that all three ADMM-Search algorithms run Modified ADMM to obtain an initial point. Instead of fine-tuning the parameters of Modified ADMM for each problem we will set K α, τ and q as 100, 0.2, 1.1 and 2, which usually gives acceptable outcomes for our problems. ADMM-Search-2 and ADMM-Search-3 also run ADMM-based lower bound (Algorithm 4.2), where we set parameters K, τ, q, $\lambda^{(0)}$ as $20, 1.5, 2$ and $0.02\lambda^*$ when computing T_m (See Line 5 of Algorithm 4.3) and change them to $5, 2, 1$ and $0.05\lambda^*$ when pruning the nodes (see Line 15 and 56 of Algorithm 4.3) since pruning nodes usually requires less accuracy. All choices above will be kept for all of the rest experiments in this thesis.

Tables 5.6 and 5.7 summarize the minimum, maximum, median and average running time of the same 100 Case 1 random instances for these four solvers.

In the following we give our observations from the two tables and some comments.

Table 5.6: Time comparison among different algorithms on Case 1 problems ($u = 10$)

100 Case 1 instances with $m = 15, n = 20, l = 0, u = 10, \sigma = 0.01$				
	Min(s)	Max(s)	Median(s)	Average(s)
DTS	0.0048	4.1596	0.1362	0.2433
ADMM-Search-1	**0.0026**	0.0508	**0.0160**	**0.0175**
ADMM-Search-2	0.0162	0.0332	0.0216	0.0215
ADMM-Search-3	0.0143	**0.0268**	0.0206	0.0203

100 Case 1 instances with $m = 15, n = 20, l = 0, u = 10, \sigma = 0.1$				
	Min(s)	Max(s)	Median(s)	Average(s)
DTS	0.0233	3.9949	0.1647	0.2619
ADMM-Search-1	**0.0062**	0.1242	0.0410	0.0431
ADMM-Search-2	0.0142	0.0307	0.0220	0.0217
ADMM-Search-3	0.0137	**0.0298**	**0.0207**	**0.0204**

100 Case 1 instances with $m = 15, n = 20, l = 0, u = 10, \sigma = 0.5$				
	Min(s)	Max(s)	Median(s)	Average(s)
DTS	**0.0114**	4.6039	0.3248	0.4652
ADMM-Search-1	0.0418	**0.8619**	**0.2735**	**0.2955**
ADMM-Search-2	0.0549	0.9445	0.2994	0.3298
ADMM-Search-3	0.0506	0.7270	0.2919	0.3097

1. DTS is always slower in terms of maximal, median and average time than three other algorithms which incorporate ADMM and the difference enlarges when u increases or σ drops.

2. Compared with other three approaches that incorporate ADMM, DTS has larger differences between minimal and maximal time and between median and average time.

3. ADMM-Search-1 takes less time than DTS, which indicates that the initial search radius given by Mdified ADMM can improve the search efficiency.

4. ADMM-Search-2 and ADMM-Search-3 are slower than ADMM-Search-1 when $\sigma = 0.5$, meaning that for this case, ADMM-based lower bounds, which are less tight when σ is large as shown in Table 5.5, do not help to improve the search efficiency.

5. ADMM-Search-1 is also faster than other two ADMM-Search for the case $u = 10, \sigma = 0.01$ (the first sub-table in Table 5.6) because in this case although the lower bounds

Table 5.7: Time comparison among different algorithms on Case 1 problems ($u = 20$)

100 Case 1 instances with $m = 15, n = 20, l = 0, u = 20, \sigma = 0.01$

	Min(s)	Max(s)	Median(s)	Average(s)
DTS	0.2187	161.9328	2.7011	5.3550
ADMM-Search-1	0.0290	1.0347	0.3104	0.3593
ADMM-Search-2	0.0297	0.2882	0.0438	0.0476
ADMM-Search-3	**0.0273**	**0.1066**	**0.0375**	**0.0381**

100 Case 1 instances with $m = 15, n = 20, l = 0, u = 20, \sigma = 0.1$

	Min(s)	Max(s)	Median(s)	Average(s)
DTS	0.2961	174.9038	3.0593	6.2585
ADMM-Search-1	0.0932	2.6411	0.8482	0.8937
ADMM-Search-2	0.0281	0.0636	0.0434	0.0439
ADMM-Search-3	**0.0264**	**0.0528**	**0.0377**	**0.0374**

100 Case 1 instances with $m = 15, n = 20, l = 0, u = 20, \sigma = 0.5$

	Min(s)	Max(s)	Median(s)	Average(s)
DTS	1.3706	24.1324	5.3079	6.3041
ADMM-Search-1	**0.5064**	**10.2416**	**3.9285**	**4.2512**
ADMM-Search-2	0.6964	12.8465	4.6955	5.1634
ADMM-Search-3	0.5609	11.1708	4.2607	4.3266

are tight enough to prune nodes, the search region is quite small and it may not be rewarding to compute such lower bounds. From the case $u = 20, \sigma = 0.01$ (the first sub-table in Table 5.7), we can see that when the search region increases, the lower bounds become helpful.

6. Except the cases mentioned above ($\sigma = 0.5$ and $\sigma = 0.01, u = 10$), ADMM-Search-2 and ADMM-Search-3 are faster than ADMM-Search-1, indicating that for these cases, lower bounds help to reduce the total costs.

7. ADMM-Search-3 is always slightly faster than ADMM-Search-2, which means that the top level search reordering technique in [78] may even have a negative effect on the ADMM tree search algorithm. Thus, in the following experiments, we will employ ADMM-Search-3, which is Algorithm 4.3, for more comparisons and refer to ADMM-Search-3 as the ADMM-Search algorithm.

5.2.5 Comparisons of ADMM Search and Other Solvers

In the section, we compare our ADMM tree search algorithm (Algorithm 4.3) with selected commercial optimisation packages (Cplex[1], Gurobi[2], and Mosek[3]) for both Case 1 and Case 2 problems. These three commercial solvers are chosen because they all declare to provide global method for the mixed-integer quadratic programming (MIQP) problems (see [28]). Note that the UBILS problem can be viewed as a type of MIQP problems. Besides, Gurobi self-claims as the fastest solver for MIQP problems in benchmark datasets[4]. Note that since Cplex, Gurobi and Mosek all employ multi-thread computing, in our ADMM tree search algorithm we now compute lower bounds in parallel with the help of MATLAB's Parallel Computing toolbox (change "for"-loop to "parfor"-loop in MATLAB), i.e., when moving down to level k and computing lower bounds, for each $z_k \in \bar{\mathcal{Z}}_k$ where $\bar{\mathcal{Z}}_k$ is the associated feasible set defined in (2.41), every thread runs ADMM-based lower bound (Algorithm 4.2) independently and prune unfeasible nodes (those unfeasible z_k are then removed from the feasible set $\bar{\mathcal{Z}}_k$.

5.2.5.1 Case 1 problems

Comparisons for different box constraints

Figure 5.4 displays the average running time of five solvers versus the constraint box length $u - l$ ($l = 0$) and Figure 5.5 shows the accuracy of solutions obtained by three solvers Cplex, Gurobi and Mosek (note that the solvers DTS and ADMM-Search give the optimal solutions). The test set for each u includes 100 Case 1 random instances with fixed $l = 0, m = 15, n = 20, \sigma = 0.1$. Note the accuracy is defined as out of 100 instances, the number of instances where the computed solution is equal to the optimal one.

From the figure we can see that only when $u = 2$, the DTS algorithm is the fastest one in terms of average time but its running time grows significantly with u. In contrast, our

[1]IBM ILOG Cplex Optimization Studio V12.10.0
[2]Gurobi Optimizer 9.1
[3]Mosek Optimization Suite Release 9.2.32
[4]https://www.gurobi.com/wp-content/uploads/2018/12/benchmarks.pdf

ADMM-Search algorithm becomes the fastest solver when $u \geq 4$ and its average time grows relatively smoothly with u. The advantage of ADMM-Search over DTS becomes much more significant as u increases. The average running time for Cplex and Gurobi drops when u is larger than 18 and it is because in that case, these two solvers provide much less robust approach. Figure 5.5 shows that the accuracy of Cplex and Gurobi drops dramatically with u while Mosek, despite of being slower than the former two solvers, can always give the global solution. When u is 22, the accuracy of Cplex and Gurobi drops to mere 16% and 9% respectively. Note that the accuracy comparison does not include ADMM-Search and DTS because both algorithms are tree search based and always guarantee to provide the optimal solution.

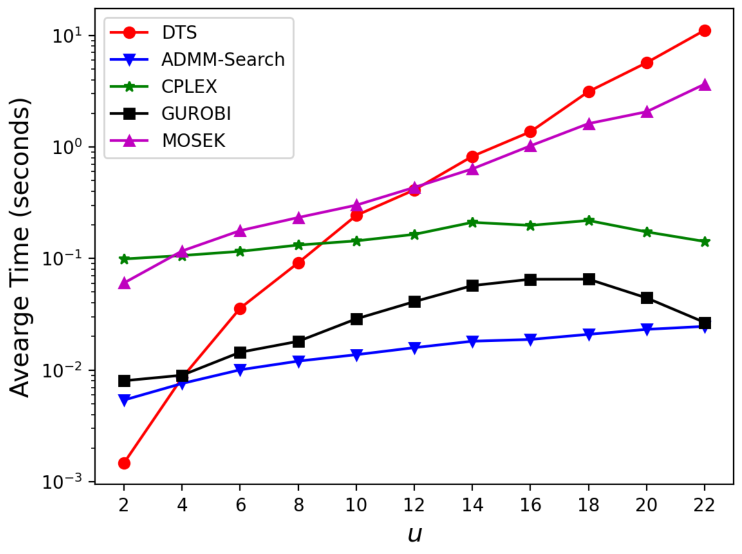

Figure 5.4: Case 1: Average Time vs. u with $l = 0, m = 15, n = 20, \sigma = 0.1$

Comparisons for different column dimensions

Figure 5.6 shows the average running time of all five solvers when n varies from 16 to 23. Here we generate 100 Case 1 random instances and m, l, u, σ are fixed as $15, 0, 10, 0.1$ respectively, and $n - m$ changes from 1 to 8 (n goes from 16 to 23), which is the number of underdetermined levels. It can be seen that DTS has the least average running time when $n \leq 17$ and after the turning point $n = 17$, ADMM-Search remains the fastest solver.

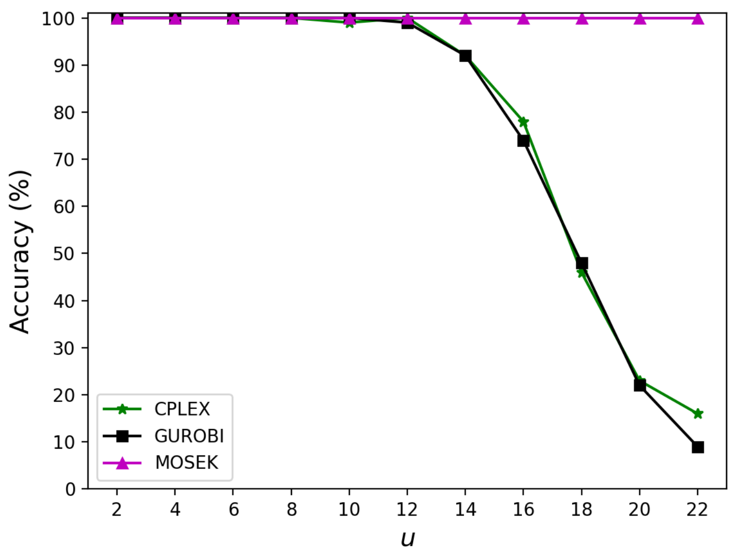

Figure 5.5: Case 1: Accuracy vs. u with $l = 0, m = 15, n = 20, \sigma = 0.1$

Similar to the length of box constraint, the advantage of ADMM-Search over DTS becomes much more significant as the number of underdetermined levels $(m - n)$ increases. That can be a huge advantage of ADMM-Search over DTS. As for three commercial solvers, Gurobi is faster than the other two and all these three are slower than ADMM-Search for all n. Besides, from Figure 5.7 we can see that as n increases, the accuracy of Guorbi and Cplex drops. For example, when $n = 23$, Gurobi and Cplex only give the optimal solution for 87% and 89% instances. So similar to Figure 5.5, when the problem becomes more difficult to solve, Cplex and Gurobi are less robust.

Comparisons for different row dimensions

Figure 5.8 shows the average running time of all five solvers when m varies from 13 to 19. Here we generate 100 Case 1 random instances and n, l, u, σ are fixed as $20, 0, 10, 0.1$ respectively, and $n - m$ changes from 7 to 1 (m goes from 13 to 19), which is the number of underdetermined levels. The associated accuracy is displayed in Figure 5.9. A similar trend can be seen here: when the number of underdetermined levels $(n - m)$ increases, the advantage of ADMM-Search over other solvers becomes larger.

Comparisons for different noise standard deviations

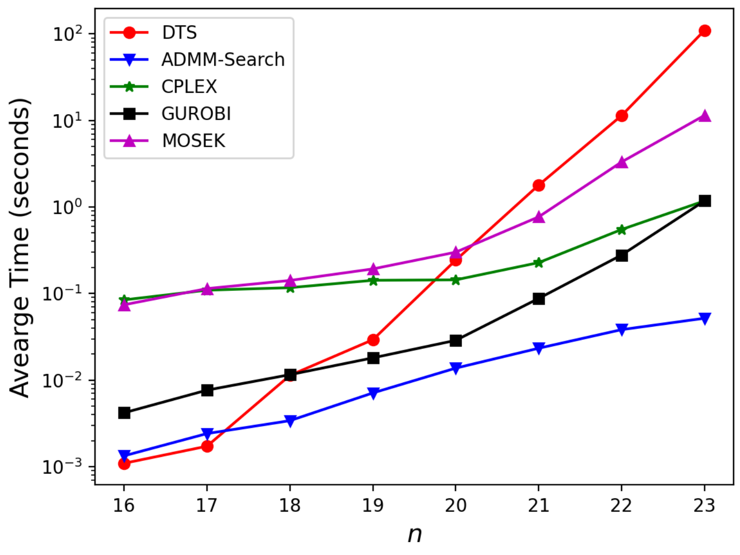

Figure 5.6: Case 1: Average Time vs. n with $l = 0, u = 10, m = 15, \sigma = 0.1$

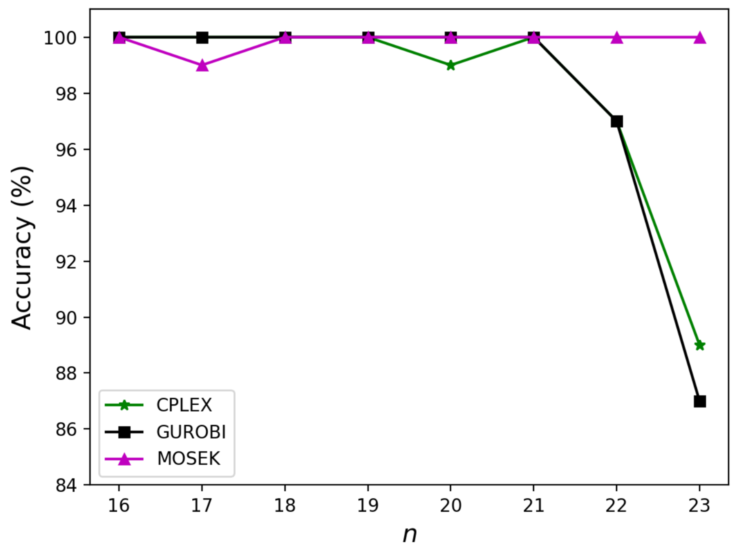

Figure 5.7: Case 1: Accuracy vs. n with $l = 0, u = 10, m = 15, \sigma = 0.1$

Figure 5.10 displays the average running time of five solvers versus σ and Figure 5.11 shows how the associated accuracy of Cplex, Gurobi and Mosek changes with σ. Here we fix other parameters $m = 15, n = 20, l = 0, u = 10$ and for each $\sigma = 0.05 : 0.05 : 0.5$ we generate 100 Case 1 random instances. From Figure 5.10 we can see that ADMM-Search

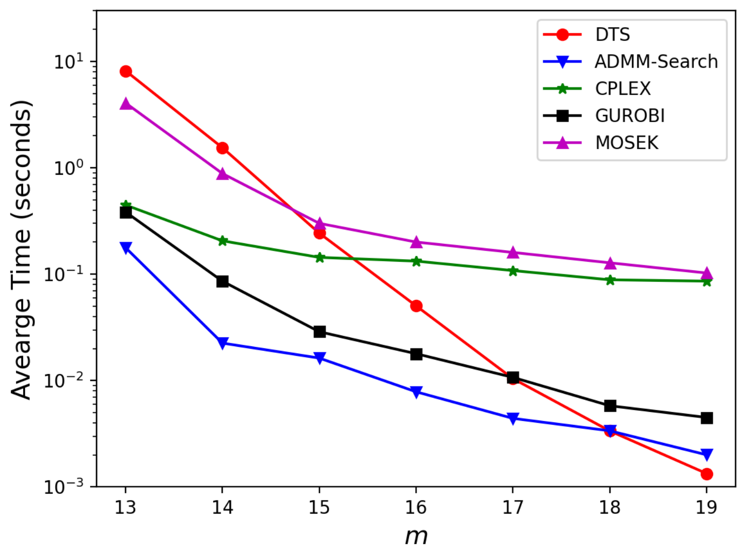

Figure 5.8: Case 1: Average Time vs. m with $l = 0, u = 10, n = 20, \sigma = 0.1$

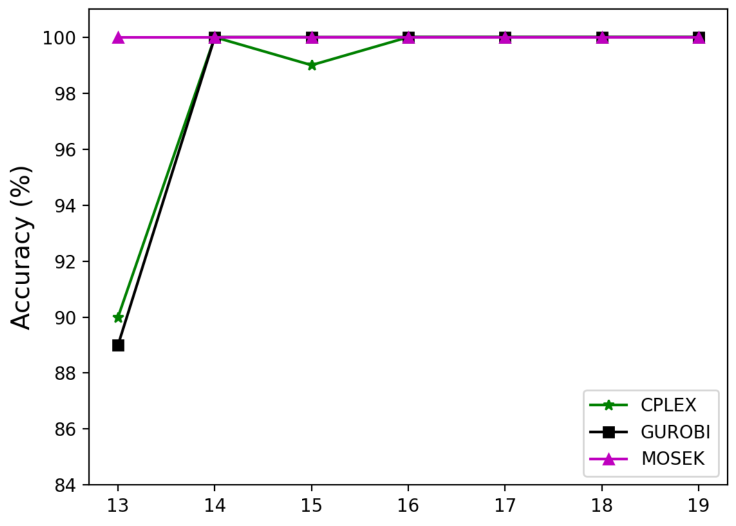

Figure 5.9: Case 1: Accuracy vs. m with $l = 0, u = 10, n = 20, \sigma = 0.1$

has the shortest average running time when $\sigma \leq 0.25$ and after that point, Gurobi remains the fastest solver. All solvers' average running time increase with σ, which is reasonable because when σ increases, more probably the residual becomes larger, making the search region larger as well. While other four solvers' running time grow smoothly with σ, ADMM-

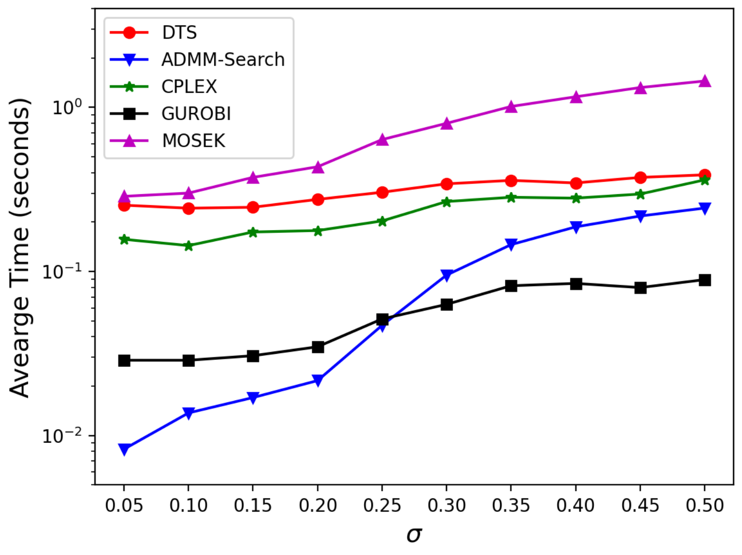

Figure 5.10: Case 1: Average Time vs. σ with $l = 0, u = 10, m = 15, n = 20$

Search's time increases more dramatically. It is because when σ becomes large, the lower bound techniques may not help to prune the search tree as we have shown in Table 5.5. Note that with the adaptive strategy, we discard computing lower bound early if they are shown to be non-effective. ADMM-Search is always faster than DTS as well as Cplex and Mosek owing to its good initial search radius given by Modified ADMM. From Figure 5.11, we can see that unlike DTS and ADMM-Search, all three commercial solvers cannot always guarantee to provide the optimal solution.

Comparisons on performance and accuracy profiles

Above we use average running time over all experimental instances as a performance metric to evaluate various solvers. However, the main drawback of this metric is that, a small number of the most difficult instances can dominate the results. Therefore, to compare various solvers further, we use a different metric, performance profiles to evaluate them.

Performance profiles provide an effective mean to compare performances for several solvers, eliminating some of the bias that computing averages has. It was first proposed by Dolan and More in [30], and is defined as a cumulative distribution function for a performance metric. Then we show how performance profiles can be obtained from raw data. Suppose we

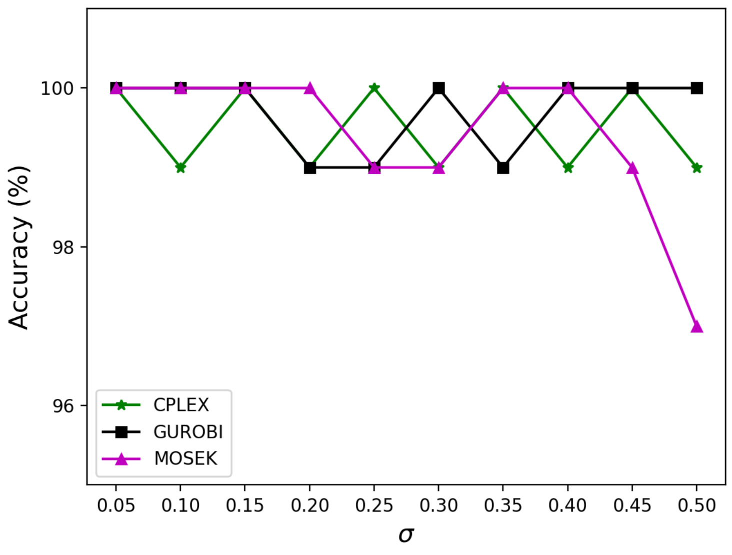

Figure 5.11: Case 1: Accuracy vs. σ with $l = 0, u = 10, m = 15, n = 20$

solve a set of n_p problems denoted as P with n_s different solvers denoted as S. Let p denote a particular problem, s denote a particular solver and $t_{p,s}$ denote the computing time required to solve problem p by solver s. We compare the performance on problem p by solver s with the best performance by any solver in S on this problem, and thus we have the performance ratio as:

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in S\}}. \tag{5.11}$$

For practical purposes, if a solver does not solve a problem, which means in our experiments exceeding the chosen time limit we set its $t_{p,s}$ as infinity. In order to obtain an overall assessment of a solver on the given test set, we define a cumulative distribution function $\rho_s(\tau)$ as:

$$\rho_s(\tau) = \frac{1}{n_p}\text{size}\{p \in P : r_{p,s} \leq \tau\}. \tag{5.12}$$

So $\rho_s(\tau)$ is the probability that a performance ratio $r_{p,s}$ is within a factor of τ of the fastest solver. Plotting $\rho_s(\tau)$ for all solvers in a chosen interval of τ gives the so-called performance profiles.

Note that one drawback of performance profiles is that the relative performance of the solvers may not be assessed. If performance profiles are used to compare more than two solvers, we can determine which solver has the highest probability of being within a factor τ of the best solver, but we cannot necessarily assess the performance of one solver relative to another that is not the best, which means that we are unable to rank the solvers and tell which solver is the second or third best (see e.g., [38]).

Apart from performance profiles, accuracy profiles are designed for fixed cost test sets (see e.g., [6]). Accuracy profiles begin by defining for each problem $p \in P$ and solver $s \in S$, an accuracy measure as

$$\gamma_{p,s} = \begin{cases} -f_{p,s}^{\text{acc}} & \text{if } -f_{p,s}^{\text{acc}} \leq M \\ M & \text{otherwise} \end{cases} \tag{5.13}$$

where $f_{p,s}^{\text{acc}} = \log_{10} \frac{f(\bar{x}_{p,s}) - f(x_p^*)}{f(x_p^0) - f(x_p^*)}$, f is the objective function, $\bar{x}_{p,s}$ is the solution obtained by solver s on problem p within the allowed time, x_p^* is the optimal solution for problem p and x_p^0 is the initial point. Then the performance of the solver s on the test set P is measured using the following function:

$$R_s(\tau) = \frac{1}{n_p} \text{size}\{p \in P : \gamma_{p,s} \leq \tau\}. \tag{5.14}$$

The accuracy profile $R_s(\tau)$ shows the proportion of problems such that the solver $s \in S$, given fixed computing time, is able to obtain a solution within an accuracy of τ of the optimal solution (e.g., when τ is three, we say that the solution achieves three digits of accuracy compared to the optimal solution). Note that for our simulation, the initial vector x_p^0 is selected as a vector with all zeros and the maximal allowed time of each instance for all solvers is the time when the fastest solver terminates, and the parameter M is set as 6.

Figure 5.12 shows the performance profiles of these five solvers. The test set includes 100 random Case 1 problems with $l = 0, u = 10, m = 15, n = 20, \sigma = 0.1$. It shows that ADMM-Search can first finish around 75% of all instances when τ in (5.12) is one, which means that for 75% instances, ADMM-Search is the fastest solver, and Gurobi can only finish around 20% when τ is one and other three solvers close to zero. ADMM-Search keeps this advantage until the end (finish all instances). From this point, ADMM-Search is clearly the best solver for this case.

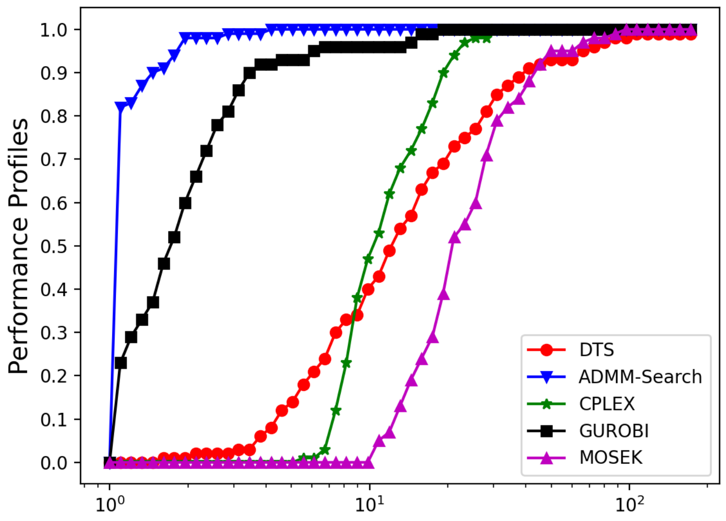

Figure 5.12: Case 1: Performance Profiles with $l = 0, u = 10, m = 15, n = 20, \sigma = 0.1$

The corresponding accuracy profiles are shown in Figure 5.13. We can see that ADMM-Search can always give the optimal solution within the allowed time, which is owing to the effective initial points given by Modified ADMM as we have shown in Table 5.1. Besides, Gurobi can give the optimal solution within the allowed time for around 40% instances, followed by DTS (around 5%) and other two solvers (nearly 0%). For remaining instances where Cplex, Gurobi, Mosek and DTS cannot provide optimal solution, most solutions are within around 2-3 digits of accuracy. Hence, in this case, we can see that ADMM-Search is the best one among these five solvers.

Next, we consider UBILS problems with larger search region. Figure 5.14 shows the performance profiles of these five solvers on 100 random Case 1 instances with $l = 0, u = 15, m = 15, n = 21, \sigma = 0.1$. It is clear that now UBILS problems should be more difficult to solve than the previous example. The associated accuracy profiles are illustrated in Figure 5.15. It shows that ADMM-Search can first finish around 90% of all instances at

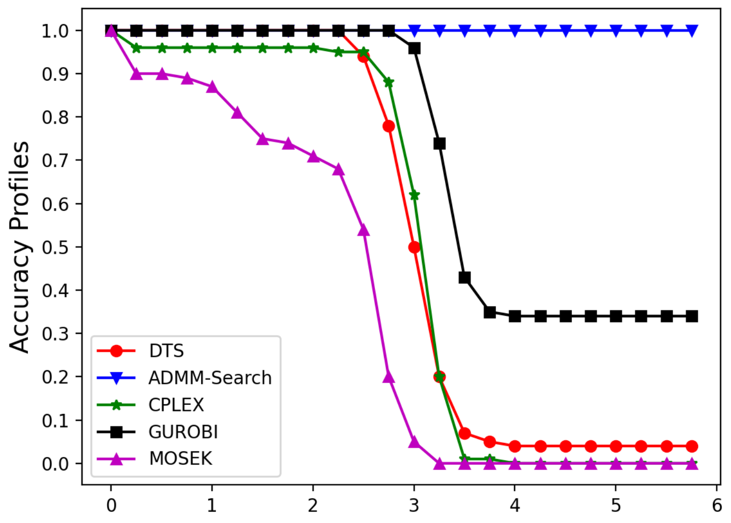

Figure 5.13: Case 1: Accuracy Profiles with $l = 0, u = 10, m = 15, n = 20, \sigma = 0.1$

the beginning and solve all test problems before τ increases to three. In contrast, Mosek and DTS are much slower and they can finish all problems when τ increases to around 1000 and 4000. We also find in the tests that in this case Cplex and Gurobi, similar to what is shown in Figure 5.4, can only give the optimal solution for 56% and 53% instances. From this point, the advantage of ADMM-Search over other four solvers becomes more significant for UBILS problems with greater search region.

5.2.5.2 Case 2 problems

In this section, we consider Case 2 problems, which arise from real-world applications: MIMO flat-fading systems. We first focus on 4QAM, 16QAM and 64QAM systems with fixed SNR $= 25, N_r = 8, N_t = 12$, and compare these five solvers on randomly generated experimental instances. Note that for the following comparisons, we also include the PR approach introduced in Section 2.3.3 since PR can deal with UBILS problems of Case 2.

4QAM

Figures 5.16 and 5.17 show the performance and accuracy profiles of these solvers for 4QAM problems. The test set includes 100 random 4QAM instances with $N_r = 8, N_t =$

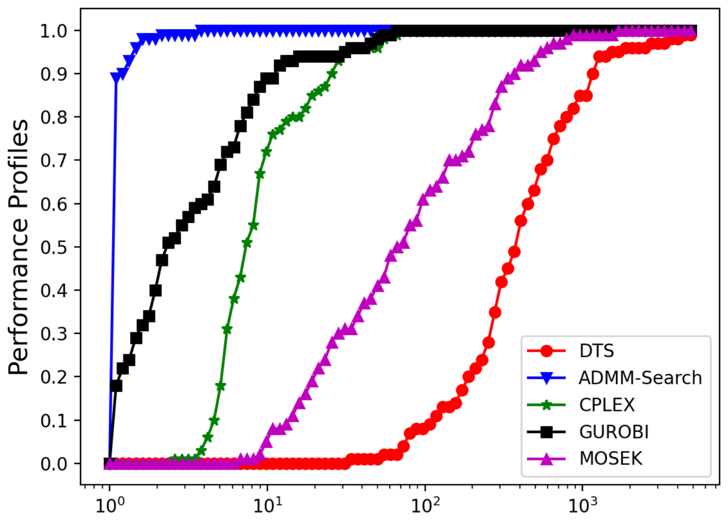

Figure 5.14: Case 1: Performance Profiles with $l = 0, u = 15, m = 15, n = 21, \sigma = 0.1$

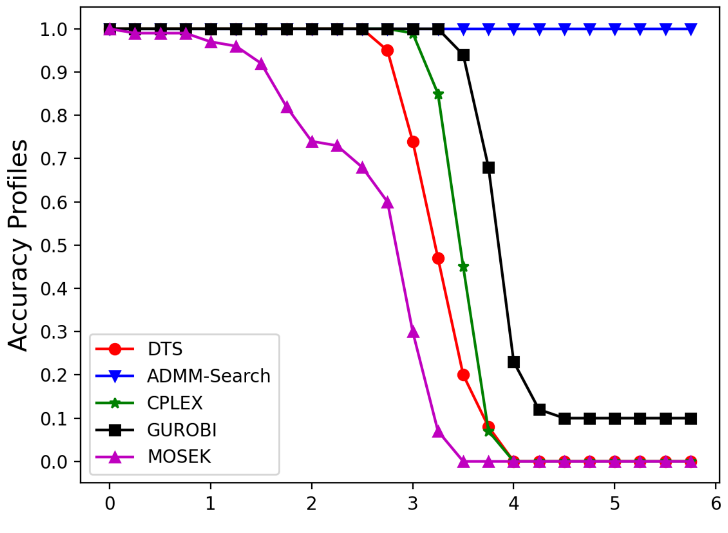

Figure 5.15: Case 1: Accuracy Profiles with $l = 0, u = 15, m = 15, n = 21, \sigma = 0.1$

$12, \mathrm{SNR} = 25$. It shows that for around 90% instances, ADMM-Search remains the fastest solver. Note that for 4QAM problems, $\mathbf{x}^{(1)}$ given by Modified ADMM (see (4.3)) of ADMM-Search in the first iteration is guaranteed to be the optimal solution if it is inside the constrained box because the penalty term $\|\mathbf{x} - \mathbf{z}^{(0)} + \mathbf{w}^{(0)}\|_2^2$ in (4.3) is a constant $\|\frac{\mathbf{1}_{(n)}}{2}\|_2^2$ and thus we can terminate ADMM-Search early in this case. From the associated accuracy profiles shown in Figure 5.17 we can see that ADMM-Search can always give the optimal solution within the allowed time, followed by DTS which can solve around 90% instances. In this case, other four solvers PR, Cplex, Gurobi and Mosek can only achieve 0-1 digits of accuracy within the allowed running time. Hence, in this case, it is clear that ADMM-Search is the best one among these solvers and ADMM-Search, DTS and PR are all superior to three commercial solvers.

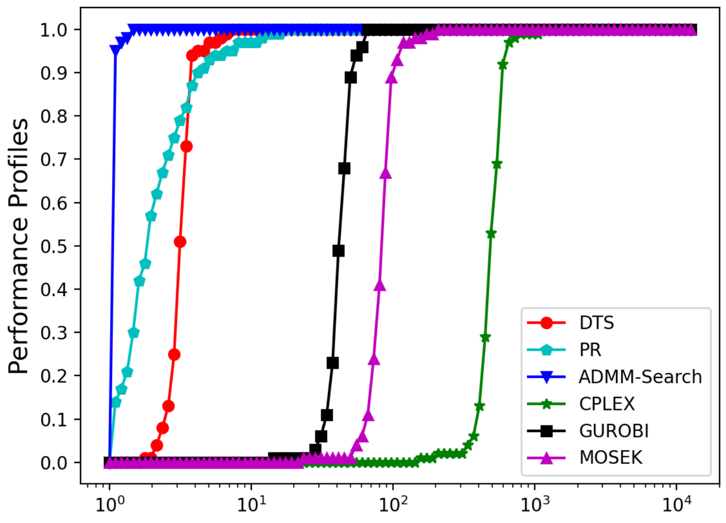

Figure 5.16: 4QAM: Performance Profiles with $N_r = 8, N_t = 12, \mathrm{SNR} = 25$

16QAM

Figures 5.18 and 5.19 display the performance and accuracy profiles of these solvers for 16QAM problems. The test set includes 100 random 16QAM instances with $N_r = 8, N_t = 12, \mathrm{SNR} = 25$. Combining two figures, we can see that Gurobi is the fastest solver for around 80% problems. Gurobi can finish all instances first when τ is around three,

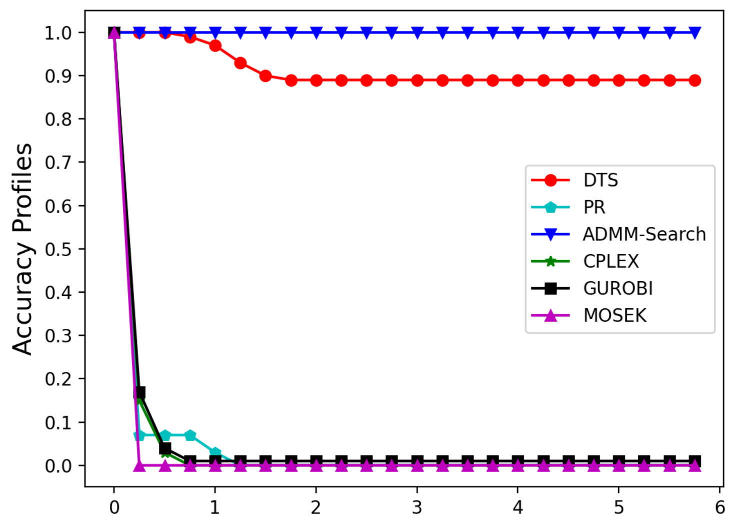

Figure 5.17: 4QAM: Accuracy Profiles with $N_r = 8, N_t = 12, \text{SNR} = 25$

followed by ADMM-Search when τ is roughly ten. But when required to terminate within the allowed time for each instance, ADMM-Search can provide the optimal solution for nearly all problems while Gurobi only roughly 90%. It is owing to the effective initial points of ADMM-Search given by Modified ADMM. For other solvers, they can solve all problems with almost the same τ as roughly 40 except PR that solves all instances when τ increases to around 200, and they can achieve around 1-2 digits of accuracy in most cases within allowed running time for each instance.

64QAM

Figures 5.20 and 5.21 show the performance and accuracy profiles of these solvers for 64QAM problems. The test set includes 100 random 64QAM instances with $N_r = 8, N_t = 12, \text{SNR} = 25$. Note that for this test, since PR can be extremely slow for certain experiment instances, we set the time limit of single instance as 300 seconds, which means that all solvers are forced to terminate when exceeding this limit. It shows that for around 70% instances, ADMM-Search is the fastest solver and for the remaining 30%, Gurobi is the fastest one. When τ is around 30, ADMM-Search can provide optimal solution for all test problems, followed by Cplex, Gurobi, Mosek. DTS can solve all all instances when τ is larger than

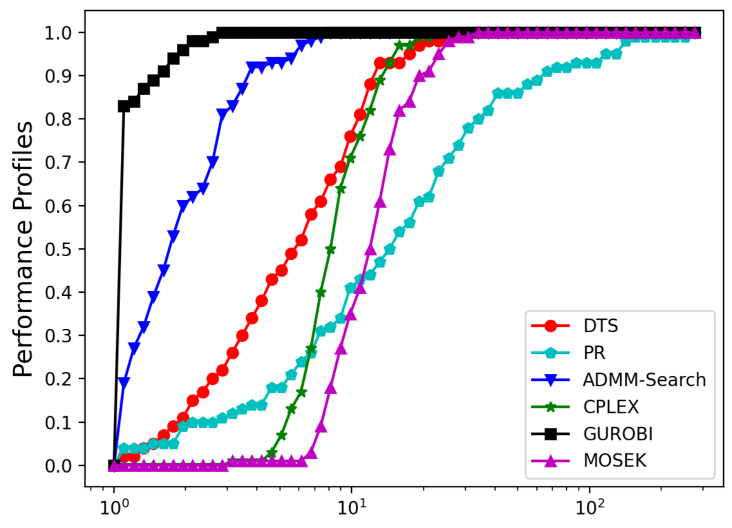

Figure 5.18: 16QAM: Performance Profiles with $N_r = 8, N_t = 12, \text{SNR} = 25$

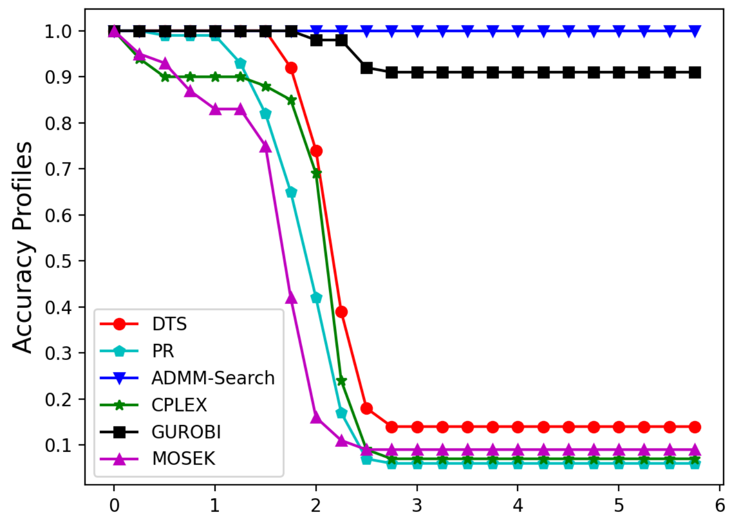

Figure 5.19: 16QAM: Accuracy Profiles with $N_r = 8, N_t = 12, \text{SNR} = 25$

2000 and PR terminates for only 60% instances within the time limit (300 seconds) for each instance. From the associated accuracy profiles shown in Figure 5.21, we can see that ADMM-Search can always give the optimal solution within the allowed time, followed by Gurobi which can solve around 40% instances. In this case, other four solvers DTS, PR, Cplex and Mosek can only achieve around 1-3 digits of accuracy for most instances within the allowed running time. Hence, in this case, we can say that ADMM-Search is the best one among these six solvers.

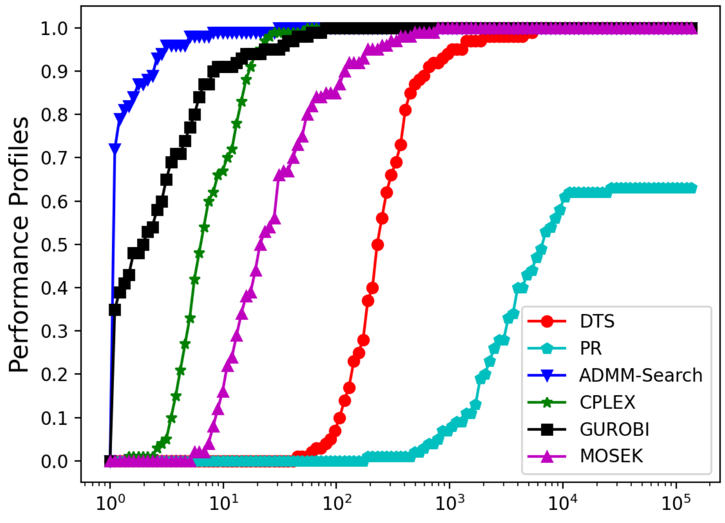

Figure 5.20: 64QAM: Performance Profiles with $N_r = 8, N_t = 12, \text{SNR} = 25$

Table 5.8 displays the minimal, maximal, median and average running time of these six solvers for 4QAM, 16QAM and 64QAM problems. For each of 4, 16 and 64QAM, we generate 100 random instances with $N_r = 8, N_t = 12, \text{SNR} = 25$. Note that for 64QAM problems, we set time limit for each instance as 300 seconds as we have done before. We can see that for 4QAM, ADMM-Search is the fastest solver and for 16QAM Gurobi is the fastest in terms of all four running time metrics except maximal time. For 64QAM problems, Gurobi can give the smallest minimal and Cplex the maximal time, while ADMM-Search is the fastest solver in terms of median and average time. It can also be seen that ADMM-Search is significantly faster than DTS and PR approach, especially for 64QAM problems. Such results are similar

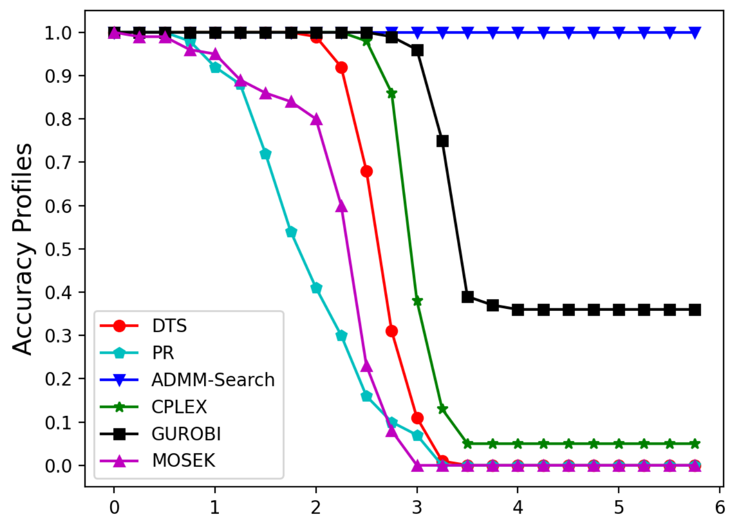

Figure 5.21: 64QAM: Accuracy Profiles with $N_r = 8, N_t = 12, \text{SNR} = 25$

to what we have obtained from performance profiles and state that ADMM-Search is better than other solvers in general.

In the following, we will focus on 64QAM problems and investigate how the running time or performance for these five solvers changes with N_t and SNR. Note that as we have shown before PR is much slower than other solvers on 64QAM problems, and therefore we exclude PR approach in the following comparison.

Comparisons for different transmitter antennas N_t

Figure 5.22 shows how average running time changes with N_t for these five solvers. The test set includes 100 random generated 64QAM instances for each N_t with fixed $N_r = 8, \text{SNR} = 25$. Note that for all instances, all five solvers provide the optimal solution. We can see that when $N_t = 9$, DTS is faster than other four solvers on average but its running time increases dramatically with N_t since the size of the search tree grows exponentially with $N_t - N_r$. When $N_t \geq 10$, ADMM-Search is faster than DTS as well as other three solvers and keeps this advantage . It is understandable that when N_t keeps increasing, the advantage ADMM-Search over DTS in terms of average time will become greater.

Comparisons for different SNR

Table 5.8: Running time for 4,16,64QAM with $N_r = 8, N_t = 12, \text{SNR} = 25$

	Min(s)	Max(s)	Median(s)	Average(s)
4QAM				
DTS	0.0008	**0.0092**	0.0012	0.0013
PR	0.0003	0.0164	0.0007	0.0011
ADMM-Search	**0.0002**	0.0110	**0.0004**	**0.0005**
Cplex	0.1687	0.4637	0.1863	0.1945
Gurobi	0.0031	0.0241	0.0038	0.0047
Mosek	0.0255	0.1023	0.0305	0.0321
16QAM				
DTS	0.0073	0.4093	0.0769	0.0918
PR	0.0051	5.1090	0.1750	0.3430
ADMM-Search	0.0147	**0.0419**	0.0250	0.0255
Cplex	0.0684	0.2015	0.1141	0.1100
Gurobi	**0.0050**	0.0978	**0.0138**	**0.0164**
Mosek	0.0355	0.2996	0.1655	0.1649
64QAM				
DTS	1.5516	251.8253	8.8669	14.6173
PR	8.2537	300.0000	215.8300	195.5577
ADMM-Search	0.0299	3.6939	**0.0442**	**0.0835**
Cplex	0.0722	**2.8594**	0.2274	0.3454
Gurobi	**0.0068**	3.7462	0.0976	0.2409
Mosek	0.1196	32.8078	0.9479	2.2650

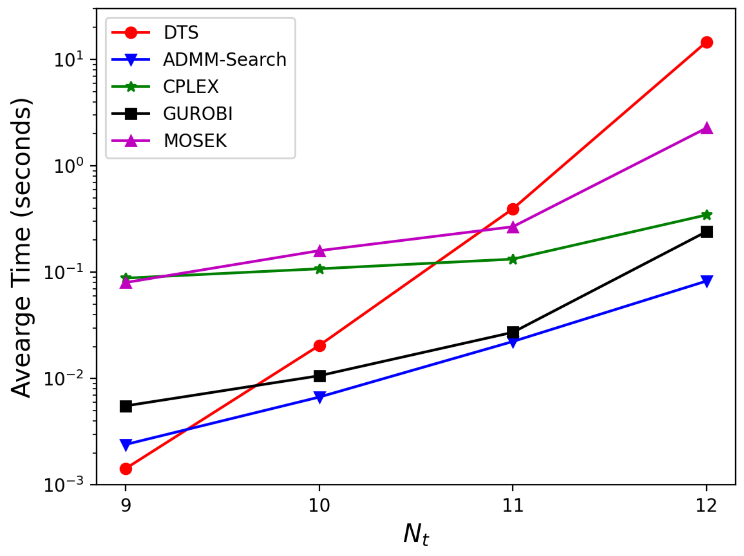

Figure 5.22: 64QAM: Average Time vs. N_t with $N_r = 8, \text{SNR} = 25$

Figure 5.23 shows how average running time of 100 instances changes with SNR for these five solvers. The test set includes 100 randomly generated 64QAM instances for SNR=15:5:35 with fixed $N_r = 8, N_t = 12$. Note that for all instances, all five solvers provide the optimal solution except when SNR is euqal to 15, Mosek fails two instances. We can see that ADMM-Search is the fastest solver when SNR$\geq$ 25 but its running time increases significantly when SNR drops. It is considered as the main drawback of ADMM-Search: when SNR is low or noises become large, less probably the ADMM-based lower bounds can help prune the search tree. For these cases, ADMM-Search is faster than DTS owing the effective initial search radius given by Modified ADMM. Note that the success rate of the optimal solution becomes less as SNR drops (e.g., when SNR is beyond 20, the optimal solution to the UBILS problem is equal to the true parameter vector $\mathbf{x}^*$ in (5.4) for 100% instances while when SNR is 15, the percentage drops to 58%).

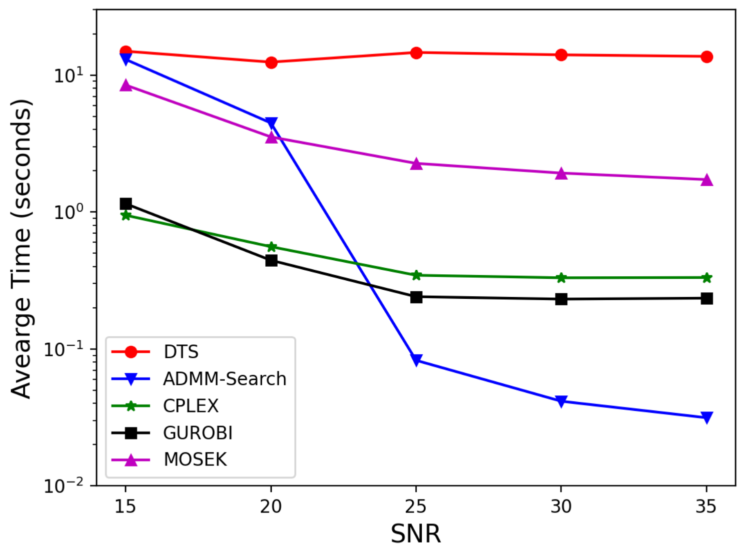

Figure 5.23: 64QAM: Average Time vs. SNR with $N_r = 8, N_t = 12$

Chapter 6

Summary and Future Work

In this chapter we summarize the thesis and propose some related ideas that can be investigated in the future.

In Chapter 2, we reviewed the existing sphere decoding algorithms for solving OILS, OBILS and UBILS problems respectively.

In Chapter 3, we reviewed the ADMM approach for solving optimization problems with linear constraints and for solving integer optimisation problems as a heuristic method. In particular we introduced the existing ADMM algorithm for mixed-integer quadratic programming problems.

The main contributions of this thesis are as follows.

In Section 4.1, we proposed a modified ADMM algorithm for UBILS problems which imposes integer requirement for $\mathbf{x}-$update. Then every iteration of the modified ADMM algorithm consists of solving an OILS problem. We suggest how to choose associated parameters for the modified ADMM algorithm and provide theoretical explanations.

In Section 4.2, we proposed to incorporate the modified ADMM algorithm into the DTS algorithm for UBILS problems with the aim of improving search efficiency. First we used the modified ADMM as a heuristic to provide a good initial search radius. Then we proposed an ADMM-based method of find lower bounds for UBILS problems to prune the search tree.

In Chapter 5, we conducted numerical simulation to first verify that the modified ADMM is much superior to the original ADMM on UBILS problems in terms of accuracy. Then we compared our ADMM-Search algorithm with DTS and PR approach plus three selected commercial solvers (Cplex, Gurobi and Mosek) in terms of different metrics including running time, performance and accuracy profiles. We test these solvers under two different cases: random cases and MIMO flat-fading systems in communications. The experiment results show that in most cases ADMM-Search is the best solver among these solvers in terms of both speed and accuracy. The advantage of ADMM-Search over DTS becomes much more significant as the size of the constrained box $(u-l)$ or the number of underdetermined levels $(m-n)$ increases.

One main limitation of ADMM-Search is that when the noise variance is large or when the residual is large its lower bound techniques may not help to reduce the total cost.

In the future work, we shall investigate the following problems:

- Since the ADMM tree search approach may not work well for UBILS problems with large residual or noise variance if such problems arise form the linear model, how can we improve the tree search approach to obtain better performance in this case? Can we find some efficient and effective methods of computing lower bounds for UBILS problems with large residuals?

- The theoretical analysis can be done to show the convergence properties of the modified ADMM on UBILS problems.

Bibliography

[1] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Trans. Inf. Theory*, 48(8):2201–2214, 2002.

[2] 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.

[3] 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.

[4] 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(2-3):227–242, 2014.

[5] L. Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[6] V. Beiranvand, W. Hare, and Y. Lucet. Best practices for comparing optimization algorithms. *Optimization and Engineering*, 18(4):815–848, 2017.

[7] D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[8] Å. Björck. *Numerical methods for least squares problems*. SIAM, 1996.

[9] J. Blömer and S. Naewe. Sampling methods for shortest vectors, closest vectors and successive minima. *Theor. Comput. Sci.*, 410(18):1648–1665, 2009.

[10] M. 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.

[11] J. Boutros, N. Gresset, L. Brunel, and M. Fossorier. Soft-input soft-output lattice sphere decoder for linear channels. In *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, volume 3, pages 1583–1587. IEEE, 2003.

[12] S. Boyd, N. Parikh, and E. Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers.* Now Publishers Inc, 2011.

[13] S. Breen and X.-W. Chang. Column reordering for box-constrained integer least squares problems. *arXiv preprint arXiv:1204.1407*, 2012.

[14] C. Buchheim, A. Caprara, and A. Lodi. An effective branch-and-bound algorithm for convex quadratic integer programming. *Mathematical programming*, 135(1):369–395, 2012.

[15] X.-W. Chang. *Numerical methods for estimation.* Lecture Notes. McGill University, 2019.

[16] X.-W. Chang and Q. Han. Solving box-constrained integer least squares problems. *IEEE Trans. Wireless Commun.*, 7(1):277–287, 2008.

[17] 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.

[18] X.-W. Chang and X. Yang. A new fast generalized sphere decoding algorithm for under-determined mimo systems. In *23rd Biennial Symposium on Communications, 2006*, pages 18–21. IEEE, 2006.

[19] X.-W. Chang and X. Yang. An efficient tree search decoder with column reordering for underdetermined mimo systems. In *IEEE GLOBECOM 2007-IEEE Global Telecommunications Conference*, pages 4375–4379. IEEE, 2007.

[20] X.-W. Chang, X. Yang, T. Le-Ngoc, and P. Wang. Partial regularisation approach for detection problems in underdetermined linear systems. *IET communications*, 3(1):17–24, 2009.

[21] X.-W. Chang, X. Yang, and T. Zhou. MLAMBDA: A modified LAMBDA method for integer least-squares estimation. *J. Geod.*, 79(9):552–565, 2005.

[22] X. Chen, X. Chang, and X. Liu. Syrafa: Synchronous rate and frequency adjustment for utilization control in distributed real-time embedded systems. *IEEE Transactions on Parallel and Distributed Systems*, 24(5):1052–1061, 2013.

[23] T. Cui and C. Tellambura. An efficient generalized sphere decoder for rank-deficient mimo systems. In *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, volume 5, pages 3689–3693. IEEE, 2004.

[24] M. O. Damen, K. Abed-Meraim, and J.-C. Belfiore. Generalized sphere decoder for asymmetrical space-time communication architecture. *Electronics letters*, 36(2):166–167, 2000.

[25] 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.

[26] T. Datta, N. Srinidhi, A. Chockalingam, and B. S. Rajan. Low-complexity near-optimal signal detection in underdetermined large-mimo systems. In *2012 National Conference on Communications (NCC)*, pages 1–5. IEEE, 2012.

[27] P. Dayal and M. K. Varanasi. A fast generalized sphere decoder for optimum decoding of under-determined mimo systems. In *Proceedings Of the Annual Allerton Conference on*

Communication Control AND Computing, volume 41, pages 1216–1225. The University; 1998, 2003.

[28] S. Diamond, R. Takapoui, and S. Boyd. A general system for heuristic minimization of convex functions over non-convex sets. *Optimization Methods and Software*, 33(1):165–193, 2018.

[29] Y. Ding, Y. Wang, J.-F. Diouris, and Z. Yao. Robust fixed-complexity sphere decoders for rank-deficient mimo systems. *IEEE transactions on wireless communications*, 12(9):4297–4305, 2013.

[30] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[31] J. Eckstein and D. P. Bertsekas. On the douglas—rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1):293–318, 1992.

[32] Y. Fadlallah, A. Aïssa-El-Bey, K. Amis, D. Pastor, and R. Pyndiah. New decoding strategy for underdetermined mimo transmission using sparse decomposition. In *21st European Signal Processing Conference (EUSIPCO 2013)*, pages 1–5. IEEE, 2013.

[33] M. Fan, X. Chang, X. Zhang, D. W. 0008, and L. Du. Top-k supervise feature selection via admm for integer programming. In *IJCAI*, pages 1646–1653, 2017.

[34] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Math. Comput.*, 44(170):463–471, 1985.

[35] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & mathematics with applications*, 2(1):17–40, 1976.

[36] V. M. García, S. Roger, R. A. Trujillo, A. M. Vidal, and A. Gonzalez. A deterministic lower bound for the radius in sphere decoding search. In *The 2010 International Conference on Advanced Technologies for Communications*, pages 11–16. IEEE, 2010.

[37] R. Glowinski and A. Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, 9(R2):41–76, 1975.

[38] N. Gould and J. Scott. A note on performance profiles for benchmarking software. *ACM Transactions on Mathematical Software (TOMS)*, 43(2):1–5, 2016.

[39] G. Hanrot, X. Pujol, and D. Stehlé. Algorithms for the shortest and closest lattice vector problems. In *International Conference on Coding and Cryptology*, pages 159–190. Springer, 2011.

[40] G. Hanrot and D. Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In *Annual international cryptology conference*, pages 170–186. Springer, 2007.

[41] A. Hassibi and S. Boyd. Integer parameter estimation in linear models with applications to GPS. *IEEE Trans. Signal Process.*, 46(11):2938–2952, 1998.

[42] C.-J. Huang, C.-S. Lee, W.-H. Chung, and T.-S. Lee. A geometry based efficient decoder for underdetermined MIMO systems. *Digital Signal Processing*, 41:60–69, 2015.

[43] 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.

[44] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, 12(3):415–440, 1987.

[45] A. Korkine and G. Zolotareff. Sur les formes quadratiques. *Math. Ann.*, 6(3):366–389, 1873.

[46] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261(4):515–534, 1982.

[47] C. Ling and N. Howgrave-Graham. Effective LLL reduction for lattice decoding. In *IEEE International Symposium on Information Theory, 2007*, pages 196–200.

[48] D. Micciancio. The hardness of the closest vector problem with preprocessing. *IEEE Transactions on Information Theory*, 47(3):1212–1215, 2001.

[49] D. Micciancio and S. Goldwasser. *Complexity of lattice problems: a cryptographic perspective*, volume 671. Springer Science & Business Media, 2012.

[50] D. Micciancio and O. Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, 2009.

[51] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. *SIAM Journal on Computing*, 42(3):1364–1391, 2013.

[52] W. H. Mow. Maximum likelihood sequence estimation from the lattice viewpoint. *IEEE Trans. Inf. Theory*, 40(5):1594–1600, 1994.

[53] V. Platonov, A. Rapinchuk, and R. Rowen. Algebraic groups and number theory. 1993.

[54] C. Qian, J. Wu, Y. R. Zheng, and Z. Wang. A modified fixed sphere decoding algorithm for under-determined mimo systems. In *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 4482–4487. IEEE, 2012.

[55] C. Qian, J. Wu, Y. R. Zheng, and Z. Wang. Two-stage list sphere decoding for under-determined multiple-input multiple-output systems. *IEEE Transactions on Wireless Communications*, 12(12):6476–6487, 2013.

[56] C. Qian, J. Wu, Y. R. Zheng, and Z. Wang. Low complexity detection algorithm for under-determined mimo systems. In *2014 IEEE International Conference on Communications (ICC)*, pages 5580–5585. IEEE, 2014.

[57] C. Qian, J. Wu, Y. R. Zheng, and Z. Wang. Simplified parallel interference cancelation for underdetermined mimo systems. *IEEE Transactions on Vehicular Technology*, 63(7):3196–3208, 2014.

[58] G. Romano, D. Ciuonzo, P. S. Rossi, and F. Palmieri. Low-complexity dominance-based sphere decoder for mimo systems. *Signal processing*, 93(9):2500–2509, 2013.

[59] C. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Math Program*, 66:181–191, 1994.

[60] K. Shahnaz and C. Ali. An efficient sphere decoding algorithm for rank-deficient ofdm/sdma systems. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2201–2204. IEEE, 2014.

[61] N. Sommer, M. Feder, and O. Shalvi. Finding the closest lattice point by iterative slicing. *SIAM J. Discrete Math*, 23(2):715–731, 2009.

[62] M. Stojnic, H. Vikalo, and B. Hassibi. An h-infinity based lower bound to speed up the sphere decoder. In *IEEE 6th Workshop on Signal Processing Advances in Wireless Communications, 2005.*, pages 751–755. IEEE, 2005.

[63] K. Su and I. J. Wassell. A new ordering for efficient sphere decoding. In *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, volume 3, pages 1906–1910. IEEE, 2005.

[64] R. Takapoui, N. Moehle, S. Boyd, and A. Bemporad. A simple effective heuristic for embedded mixed-integer quadratic programming. *International Journal of Control*, 93(1):2–12, 2020.

[65] P. J. Teunissen. Integer least-squares theory for the gnss compass. *Journal of Geodesy*, 84(7):433–447, 2010.

[66] M. Vameghestahbanati, E. Bedeer, I. Marsland, R. H. Gohary, and H. Yanikomeroglu. Enabling sphere decoding for scma. *IEEE Communications Letters*, 21(12):2750–2753, 2017.

[67] P. van Emde-Boas. *Another NP-complete partition problem and the complexity of computing short vectors in a lattice*. Report. Department of Mathematics. University of Amsterdam. 1981.

[68] P. Wang and T. Le-Ngoc. A low-complexity generalized sphere decoding approach for underdetermined linear communication systems: performance and complexity evaluation. *IEEE transactions on communications*, 57(11):3376–3388, 2009.

[69] J. Wen, X. W. Chang, and C. Tellambura. On the success probability of the box-constrained rounding and babai detectors. In *Proc. 2017 IEEE Int. Symp. Inf. Theory (ISIT)*, pages 526–530, June 2017.

[70] P. W. Wolniansky, G. J. Foschini, G. D. Golden, and R. A. Valenzuela. V-blast: An architecture for realizing very high data rates over the rich-scattering wireless channel. In *1998 URSI international symposium on signals, systems, and electronics. Conference proceedings (Cat. No. 98EX167)*, pages 295–300. IEEE, 1998.

[71] B. Wu and B. Ghanem. ℓ_p-box ADMM: A versatile framework for integer programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(7):1695–1708, 2019.

[72] D. Wübben, R. Bohnke, J. Rinas, V. Kuhn, and K. Kammeyer. Efficient algorithm for decoding layered space-time codes. *Electron. Lett.*, 37(22):1348–1350, 2001.

[73] X. Xie. *Theory and Algorithms for Some Integer Least Squares Problems*. PhD thesis, School of Computer Science, McGill University, 2014.

[74] X. Xie, X.-W. Chang, and M. A. Borno. Partial LLL reduction. In *the Proceedings of IEEE GLOBECOM 2011, 5 pages.*

[75] Z. Xu, S. De, M. Figueiredo, C. Studer, and T. Goldstein. An empirical study of admm for nonconvex problems. *arXiv preprint arXiv:1612.03349*, 2016.

[76] Z. Xu, M. Figueiredo, and T. Goldstein. Adaptive admm with spectral penalty parameter selection. In *Artificial Intelligence and Statistics*, pages 718–727. PMLR, 2017.

[77] Z. Yang, C. Liu, and J. He. A new approach for fast generalized sphere decoding in mimo systems. *IEEE Signal Processing Letters*, 12(1):41–44, 2004.

[78] J. Zhu. *Numerical methods for underdetermined box-constrained integer least squares problems*. Master's thesis, McGill University, 2016.