Combinatorial Algorithms for Subset Sum Problems

Dissertation
an der Fakultät für Mathematik
der Ruhr-Universität Bochum

vorgelegt von
Ilya Ozerov
Erstgutachter: Prof. Dr. Alexander May
Zweitgutachter: Prof. Dr. Gregor Leander
Tag der mündlichen Prüfung: 05.02.2016
# Contents

1 Introduction  

2 Consistency Problem  
   2.1 High-Level Idea  
      2.1.1 NN Problem  
      2.1.2 zeroAND Problem  
      2.1.3 A Joint Solution  
   2.2 Consistency Problem  
      2.2.1 Preliminaries  
      2.2.2 Problem and Algorithm  
      2.2.3 Analysis  
   2.3 Weight Match Problem  
      2.3.1 General Case  
      2.3.2 Random Weight Match Problem  
   2.4 Nearest Neighbor Problem  
      2.4.1 Analysis  

3 Subset Sum Problem  
   3.1 Generalized Problem  
      3.1.1 Brute Force  
      3.1.2 Meet-in-the-Middle  
   3.2 Random Subset Sum Problem  
      3.2.1 Tools  
   3.3 Known Results  
      3.3.1 Meet-in-the-Middle Resivited  
      3.3.2 Classical Representations  
   3.4 Consistent Representations  
      3.4.1 Group Weight Match Problem  
      3.4.2 Algorithm  

4 Binary Subset Sum Problem  
   4.1 Known Results  
      4.1.1 Meet-in-the-Middle  
      4.1.2 Representations I  
      4.1.3 Representations II  
   4.2 Novel Results
## CONTENTS

4.2.1 Consistent Representations I ........................................... 68
4.2.2 Consistent Representations II ......................................... 71
4.3 Results in Special Groups ............................................... 74
  4.3.1 Algorithms .......................................................... 74

5 Knapsack Problem .......................................................... 79
  5.1 Results .......................................................................... 80

6 Decoding Problem ........................................................... 81
  6.1 Classical Algorithms ...................................................... 83
    6.1.1 Prange’s Information Set Decoding ............................... 83
    6.1.2 Stern’s Decoding ...................................................... 86
    6.1.3 MMT .................................................................. 89
    6.1.4 BJMM ................................................................. 91
  6.2 Application of the Nearest Neighbor Technique .................. 95
    6.2.1 Basic Approach ...................................................... 95
    6.2.2 Using Representations ............................................ 98
    6.2.3 Nearest Neighbor with MMT .................................... 100
    6.2.4 Nearest Neighbor with BJMM ................................. 102
  6.3 Conclusion ..................................................................... 106

7 Discrete Logarithm Problem ............................................... 107
  7.1 Known Generic Algorithm .............................................. 109
  7.2 New Generic Algorithm ............................................... 109
  7.3 Optimal Splitting ......................................................... 114

Einen außerordentlichen Dank möchte ich an meine Mutter richten, die sich entschieden hat mit mir nach Deutschland auszuwandern und mir ermöglicht hat in diesem Land zur Schule zu gehen und zu studieren. Vielen vielen Dank an meine ganze Familie, Edith, Det, Andi, Maxim und besonders an meine Großmutter für die dauerhafte Unterstützung und dass ihr immer für mich da seid!


Ein großes Thanks an Saqib für die ganzen Nachtsessions an der Uni, die ganzen Ausflüge zu Burger King und die teilweise hundert km Zugfahrten, um einen Film in OV zu sehen. Durch die Zeit mit dir haben sich meine not the yellow of the egg Englischkenntnisse gefühlt mindestens verdoppelt, wofür ich dir besonders danke.


Mein abschließender Dank geht an Latex, das es geschafft hat eine Zweierpotenz an Seiten zu setzen. Sieht doch sicher aus.
Zusammenfassung  In dieser Arbeit wird eine verallgemeinerte Version wichtiger kombinatorischer Probleme, sogenannte Subset Sum Probleme, analysiert. Es werden neue Algorithmen für diese Klasse von Problemen entwickelt, indem ein sogenanntes Konsistenzproblem untersucht wird, was zu besseren Algorithmen für das zeroAND Problem und das Nearest Neighbor Problem führt. Dies impliziert die bestbekannten Algorithmen für das Knapsack Problem mit einer Komplexität von $2^{0.287n}$ und das Dekodierproblem mit Laufzeit $2^{0.097n}$. Es wird gezeigt, dass die verwendeten Techniken ebenfalls in einem Spezialfall des Diskreten Logarithmusproblems angewandt werden können.

Summary  In this thesis, we present a generalized framework for the study of combinatorial problems, so-called Subset Sum Problems. In this framework, we improve the best known technique for solving this class of problems by identifying a Consistency Problem. We present a more efficient algorithm for this problem, which leads to algorithms for the special cases of the zeroAND Problem and the Nearest Neighbor Problem. This implies the best known algorithm for the Knapsack Problem with time complexity $2^{0.287n}$ and the Decoding Problem with time complexity $2^{0.097n}$. We show that the studied combinatorial techniques can also be applied to a special case of the Discrete Logarithm Problem.
Chapter 1
Introduction

The results by Cook [Coo71], Karp [Kar72], and Levin [Lev73] on NP-completeness identify a large class of problems that are equally hard, i.e. once there is an efficient algorithm for only one of these problems, it directly implies an efficient algorithm for all the problems from the class. An algorithm for a problem is called efficient if it runs in time polynomially in the input size for all instances of the problem. An unsolved question “P \neq NP” in computer science is, if there either exist efficient algorithms for this class of problems, or if it can be shown that there are none. NP-hard problems are a larger class of problems that are at least as hard as NP-complete problems. A decision problem simply asks if a problem instance is solvable or not, whereas a computational problem also asks to find a solution. In this thesis, we want to concentrate on two NP-hard computational problems, the Knapsack Problem [Kar72] and the Decoding Problem [BMvT78]. The Knapsack Problem asks to find a subset of \( n \) given integers \( a_1, \ldots, a_n \in \mathbb{Z} \) that sums to a target integer \( s \in \mathbb{Z} \). The Decoding Problem asks to find a closest codeword of a linear code \( C \) to a given vector \( x \).

Notice that even though there might be instances of NP-hard problems that are indeed difficult to solve, this is clearly not true for all instances. An important task for the field of cryptography is therefore to identify a certain subset of hard instances. Ideally, one would like to link the worst-case hardness (i.e. the hardness of the hardest instance) to some average-case hardness, which is the hardness of an instance chosen from some distribution. Indeed, Ajtai [Ajt96] was able to show a reduction between the average-case and the worst-case of a so-called Unique Shortest Vector Problem (USVP). That is, if one is able to break a certain average-case instance (e.g. an instance of a cryptographic scheme like [AD97]) efficiently, one is able to solve any and, therefore, also the hardest instance of the USVP efficiently. However, Ajtai wasn’t able to show that the USVP is NP-hard, which is only provable for the Shortest Vector Problem. This indicates that it doesn’t seem to be possible to link the average-case hardness and the worst-case hardness of NP-hard problems.

However, even if such a link could be established, it is still unclear how fast the problems can be solved for practical instances. Therefore, one has to rely on identifying instances that are practically hard to solve by analyzing the best algorithms for these instances. This analysis can then be used to derive parameters for cryptographic schemes in order to achieve a certain security bound. Constructing cryptographic primitives, Impagliazzo and Naor [IN96] analyzed several algorithms for the Knapsack Problem in order to identify a certain subset of instances that appear to be hard. An important quantity for that is the density, which links the number of the integers \( a_1, \ldots, a_n \) and its size. For both small and high density, it can be shown that
the problem becomes easy, whereas for a density close to 1, the problem appears to be hard. Concretely, for small density, there are lattice-based algorithms [LOS85, CJL+92] that solve the Knapsack Problem in polynomial time. For high density, the problem can be solved by lattice-based methods [JG94], as well as list-based techniques [Wag02, MS09]. The latter can also be used to find more efficient algorithms for the Learning Parity with Noise Problem [BKW03], as well as the Learning with Errors Problem [ACF+15].

In this thesis, we want to concentrate on densities that are close to 1 both for the Knapsack Problem and the Decoding Problem, in which case the presented techniques are either not applicable or perform badly. It is assumed that for these special instances no efficient algorithm exists. The best algorithms are assumed to have an exponential time complexity $2^{cn}$, where $c > 0$ is some constant. One major goal of this thesis is to find the best constant $c$, introducing algorithms with novel techniques that lead to asymptotic improvement. The studied algorithms are based on purely combinatorial ideas, building on recent developments in this direction.

Classically, the Knapsack Problem with density 1 was only known to be solvable in complexity $2^{0.5n}$ with an algorithm by Horowitz and Sahni [HS74]. Howgrave-Graham and Joux [HJ10] introduced the so-called representation technique with a complexity of $2^{0.340n}$, which was further improved by Becker, Coron and Joux [BCJ11] to a complexity of $2^{0.291n}$. In Chapter 5, a novel algorithm with best known complexity $2^{0.287n}$ is presented.

The same technique of [HJ10] can also be applied to obtain new results for the Decoding Problem. In the cryptographically relevant scenario of Bounded Distance Decoding, the classical results of $2^{0.0576n}$ by Prange [Pra62] and $2^{0.0575n}$ by Stern [Ste88] were improved to $2^{0.0537n}$ by May, Meurer and Thomae [MMT11] and $2^{0.0494n}$ by Becker, Joux, May and Meurer [BJMM12]. In Chapter 6, a novel algorithm with the best known result of $2^{0.0473n}$ is presented.

Both results rely on techniques developed in Chapter 2. In that chapter, an algorithm for the so-called Consistency Problem is presented. In this problem, we are given two lists of vectors and have to find a pair of one vector of the first list and one vector of the second list that has a certain distribution. The problem is solved with a combinatorial technique that is based on a result by Dubiner [Dub10] and samples subsets of the components of the vectors to check them for certain distributions. The main contribution of the chapter is an extension and rigorous analysis of the result by Dubiner, which allows to handle exponential size input lists that appear in our applications. As a special case of the analysis, we obtain algorithms for the so-called zeroAND Problem, which leads to the result for the Knapsack Problem in Chapter 5 and the Nearest Neighbor Problem, which leads to the improvement for the Decoding Problem in Chapter 6.

Notice that both the algorithms for the Knapsack Problem and the Decoding Problem rely on very similar, purely combinatorial techniques. As already noticed in the original work [HJ10], as well as [BGLS14], this technique can be extended to arbitrary abelian groups. Indeed, an application of this technique also leads to improved results for the NTRU Problem [Oze12] as well as the Inhomogeneous Short Integer Solution Problem [BGLS14]. Bai, Galbraith, Li and Sheffield [BGLS14] propose a generalization of the problem that also covers algorithms for high density.

In Chapter 3, a similar generalization for arbitrary abelian groups, a so-called Subset Sum Problem, is presented. Compared to [BGLS14], it only covers the case of density 1, but instead gives a rigorous analysis of the algorithm in the generalized scenario. Additionally, it uses the results of Chapter 2 to construct a novel algorithm with the best known time complexity. The Subset Sum Problem is both a generalization of the Knapsack Problem and the Decoding
Problem, which allows to use the results in both Chapter 5 and Chapter 6.

In Chapter 4, the special case of a Binary Subset Sum Problem is considered, which follows directly from the results of Chapter 3. In Chapter 5 and Chapter 6, it is shown that both the Knapsack Problem, as well as the Decoding Problem are a special case of this Binary Subset Sum Problem, finally leading to the novel results.

The results on the Subset Sum Problem and ideas from the algorithms for the Decoding Problem are finally used in Chapter 7. This leads to a novel result for the Discrete Logarithm Problem in the special case of a composite group and a low-weight discrete logarithm $x$.

Overview

Let us summarize the main results of the chapters. The thesis is based on joint work with Alex May, presented in [MO], [MO14] and [MO15].

- Chapter 2 introduces a novel algorithm for the Consistency Problem. This leads to new results for the zeroAND Problem and the Nearest Neighbor Problem. The main contribution of this chapter is Theorem 17, which proves the correctness and time complexity of the algorithm for the Consistency Problem. The results are based on [MO] and [MO15].

- Chapter 3 introduces a novel framework with rigorous analysis for solving the generalized Subset Sum Problem and uses the results of Chapter 2 (zeroAND Problem) to improve the known results. The main contribution is Theorem 47, which proves the correctness and time complexity of the novel algorithm ConsistencyRep that solves the Subset Sum Problem more efficiently. This is based on [MO].

- Chapter 4 computes explicit results for the Binary Subset Sum Problem based on the results of Chapter 3. The main contribution is a Corollary 56, which follows directly from Theorem 47 and derives numerical results for the binary case of the Subset Sum Problem. This is also based on [MO].

- Chapter 5 applies the results of Chapter 4 to the special case of the Knapsack Problem, leading to the best known complexities for this problem in Corollary 65. This is once again based on [MO].

- Chapter 6 uses the results of both Chapter 2 (Nearest Neighbor Problem) as well as the results for the Binary Subset Sum Problem in Chapter 4 to present the best known algorithm for the Decoding Problem. This leads to Corollary 78 which is based on [MO15].

- Chapter 7 uses ideas of Chapter 3 and Chapter 6 to present a novel algorithm for a special case of the Discrete Logarithm Problem. The correctness and time complexity is proven in Theorem 79 based on [MO14].
A chapter overview is presented in Fig. 1.1 where an incoming arrow indicates the usage of a result of that chapter. Fig. 1.2 shows relevant lemmas, theorems and corollaries of the thesis. Notice that new results are marked in gray and an incoming arrow indicates usage of a result.

Figure 1.2: Overview: lemmas, theorems, corollaries
Chapter 2

Consistency Problem

In this chapter, we study a special case of a class of problems that are built as follows. Given two lists $L_1, L_2$ of $n$-dimensional vectors, the task is to find one vector $x_1$ of the first list and one vector $x_2$ of the second list that fulfill a certain criterion, i.e. $C(x_1, x_2) = 1$ for some function $C$ that outputs 1 if the criterion is fulfilled. Usually, there are no restrictions on the dependence of $L_1$ and $L_2$. In particular, the lists can also be identical. These kinds of problems can always be solved by a naive method with $|L_1| \cdot |L_2|$ evaluations of the function $C$, by trying each pair of one vector in the first list and one vector in the second list. Without having any additional information on the structure of the vectors or the criterion $C$, this is also the best we can hope for. However, if we introduce some more specific structure to the elements in the lists and the criterion $C$, more efficient algorithms can be constructed.

Starting in the early 70’s, interesting special cases were discussed. Usually, the problem is defined such that $L_1$ is a list of many $n$-dimensional vectors, whereas $L_2$ has only one point, the so-called query point. The most common way to define the problem is to choose the list elements from some metric space and to define $C(x_1, x_2)$ to output 1, if $x_1, x_2$ have at most some specific distance in a certain metric. This in turn can lead to algorithms for the so-called Nearest Neighbor Search, the problem of finding the closest points. Starting from the first definitions by Minsky and Papert [MP69] as well as Knuth [Knu73], this line of research leads to several interesting results in various metrics, most prominently in the Euclidean or the Hamming metrics.

An important tool for developing algorithms for the Nearest Neighbor Search is the technique of Locality Sensitive Hashing (LSH) introduced by Indyk and Motwani [IM98]. The idea is to preprocess the list $L_1$ by hashing the elements with many different hash functions such that the problem can be solved in sub-linear time for the query point in $L_2$. The hash functions have to be chosen such that points that are close in the given metric, have similar hash values. Despite the different choice of the hash functions, new results in the Euclidean metric lead directly to new results in the Hamming metric and vice versa. The most recent result for both metrics is by Andoni et al. [AINR14]. In the following, we want to assume that the lists $L_1$ and $L_2$ are roughly of the same size. In this case, the technique can be applied to the first list, after which each element in the second list can be seen as a query point and checked individually.

An additional advantage of the LSH technique is that it doesn’t require any distribution of the elements in the two lists. However, if we assume that the elements are uniformly distributed, there are better algorithms for the problem. In the Hamming metric, we have two lists of roughly the same size and with uniformly random binary vectors. The problem is to find two vectors...
2. Consistency Problem

(one in each list) with a certain distance of $\gamma n$ for some $0 \leq \gamma \leq \frac{1}{2}$. Naively, the problem can be solved in time $|L_1| \cdot |L_2| \approx |L_1|^2$. This is also the best we can hope for if $\gamma = \frac{1}{2}$, since the number of solutions in this case is (up to factors polynomially in $n$) also $|L_1|^2$. If $\gamma = 0$, there is a very efficient algorithm involving sorting and binary search that solves the problem (up to logarithmic factors) in time the size of the list. For $0 < \gamma < \frac{1}{2}$, Dubiner [Dub10] solves the problem in time roughly $|L_1|^{1-\frac{\gamma}{2}}$. With the help of fast matrix multiplication, Valiant [Val15] presents an algorithm with time complexity $|L_1|^{1.62}$, with an exponent independent of $\gamma$. Whereas the latter is an interesting result for large values of $\gamma$, for smaller values it is outperformed by the former.

Although the result by Dubiner is the best known result for small values of $\gamma$ – which is the most interesting scenario throughout this thesis – it doesn’t apply in the cryptographically relevant case of exponential (in $n$) size lists, which we want to assume in the following. An algorithm for this particular scenario is the main contribution of this thesis and was developed in [MO15]. This algorithm for the so-called Nearest Neighbor (NN) Problem is applicable to constructing algorithms for the Decoding Problem of Chapter 6. A problem related to that is the so-called zeroAND Problem, which asks to find vector pairs that do not have any matching ones and was introduced in [MO]. This algorithm is applicable in developing improved algorithms for the Subset Sum Problem of Chapter 3, as well as for the special cases of the Binary Subset Sum Problem of Chapter 4 and the Knapsack Problem of Chapter 5.

In Sect. 2.1, we want to introduce the Nearest Neighbor Problem and the zeroAND Problem. The problems are firstly discussed in a high-level manner, presenting ideas that finally lead to a joint solution for both problems. These problems are generalized to a so-called Consistency Problem in Sect. 2.2. An algorithm for this generalized problem that is applicable to both the special (and additional) cases is presented and analyzed rigorously. In Sect. 2.3, the algorithm is adapted to solving the so-called Weight Match Problem, which is a problem that appears throughout this thesis. As a special case of this problem, we present an algorithm for the Nearest Neighbor Problem in Sect. 2.4 that is important for the Decoding algorithms in Chapter 6.

2.1 High-Level Idea

In this section, we want to present algorithmic ideas for both the NN Problem in Sect. 2.1.1 and the zeroAND Problem in Sect. 2.1.2. This leads to the best known algorithm for these and related problems as explained in a high-level manner in Sect. 2.1.3. The discussion in this section is mostly informal, whereas a rigorous analysis is presented in the subsequent section. Notice that throughout the thesis rounding issues for sizes relative to the vector size $n$ are ignored. That is, for any constant $c > 0$ we assume that $n$ is chosen such that $cn$ is a positive integer. This is motivated by the fact that we deal with asymptotic time complexities that are exponential in $n$ and rounding issues can usually be resolved at the cost of only a polynomial in $n$. Moreover, this simplification clearly improves the readability of the thesis.

2.1.1 NN Problem

In the NN problem, we are given two lists $L_1, L_2$ of identical size $|L_1| = |L_2|$ of uniformly random vectors from $\{0, 1\}^n$. Notice that most vectors derived like that have a weight of $\frac{n}{2}$, so we assume in the following that all vectors in the list are of this weight for the sake of simplicity. In Sect. 2.4 we present a re-randomization technique that ensures this simplified distribution.
In addition, we are given a distance parameter $0 \leq d \leq \frac{n}{2}$. The task is to output all pairs of one element from the first list and one element from the second list that have a Hamming distance of $d$. A problem instance with $n = 6$ and $d = 2$ could be $L_1 = \{010101, 100110, 011010, 100101\}$ and $L_2 = \{100011, 101100, 101100, 011001\}$. For instance, the vectors $x_1 = 010101$ and $x_2 = 011001$ have to be output. Informally, the problem can be defined as follows.

**Definition 1** (NN problem, informal). Let $n, d \in \mathbb{N}$ with $0 \leq d \leq \frac{n}{2}$. Given two lists $L_1, L_2 \subseteq \{0,1\}^n$ of equal size $|L_1| = |L_2|$ with uniformly random vectors of weight $\frac{n}{2}$, find all pairs $(x_1, x_2) \in L_1 \times L_2$ with a Hamming distance of $d$.

Let us analyze some ideas that lead to a better complexity than the naive search with worst case running time $|L_1| \cdot |L_2|$. Two inferior ideas that might come to mind first are the following brute-force and meet-in-the-middle approaches.

**Brute-Force**

If the distance $d$ is very small, i.e. $\binom{n}{d} \ll |L_1|$, one approach can be to guess the weight $d$ vector $x_1 + x_2$ of a solution $(x_1, x_2)$ to the problem. This brute-force approach enumerates a list $D$ of all possible $\binom{n}{d}$ vectors of size $n$ and weight $d$ and creates a list $L := L_1 + D$ of the sum of all elements in $L_1$ with each of the elements in $D$. Then it enumerates all elements in $L$ and binary-searches for each element in $L$ for a match in the sorted list $L_2$.

This approach clearly finds all pairs with a distance of $d$, because for each element in $L_1$ it checks all vectors with a distance of $d$. Since sorting and binary search have only a small logarithmic overhead, the time complexity is roughly $|L_1| \cdot \binom{n}{d}$, which can be better than naive, if $d$ is small. However, for large enough $d$ this method becomes worse than the naive approach.

**Meet-in-the-Middle**

The approach can be further improved with the help of the so-called meet-in-the-middle technique that was applied to many problems like the Knapsack Problem [HS74] or the Discrete Logarithm Problem [Sha71] and is also an important part of Chapter 3. The idea is to find a subset $I \subseteq [n]$ with $|I| = \frac{n}{2}$ such that the weight $d$ vector $x_1 + x_2$ of a solution pair $(x_1, x_2)$ of the problem splits such that it has a weight of $d/2$ on the components of $I$ and therefore the same weight on the remaining components. Since for a given $(x_1, x_2)$ there are many $I$ fulfilling this property, one such $I$ is found by checking a polynomial in $n$ number of $I$ that are chosen uniformly at random. Then, two lists $D_1, D_2$ are created such that $D_1$ contains all $\{0,1\}$-vectors of size $n$ and weight $d/2$ on the components of $I$ and weight 0 on the remaining components. The list $D_2$ is created the other way round. Finally, the lists $L_1 + D_1$ and $L_2 + D_2$ are searched for matching pairs with the help of binary search.

Now fix a solution $(x_1, x_2) \in L_1 \times L_2$ to the problem with distance $d$ and an $I$ that splits the weight in $x_1 + x_2$ into two equally large halves. Then there are $y_1, y_2$ both with weight $d/2$ such that $x_1 + x_2 = y_1 + y_2$. These can be found efficiently, because $x_1 + y_1 \in L_1 + D_1$ and $x_2 + y_2 \in L_2 + D_2$, which are equal and are therefore found by binary search. Due to only small overheads for finding $I$, sorting and binary search and since $|D_1| = |D_2| = \binom{n/2}{d/2}$, the algorithm runs in time roughly $|L_1| \cdot \binom{n/2}{d/2}$. Although we obtain a roughly square root improvement, the technique is still outperformed by our main algorithm in Sect. 2.2.
Hashing for Collisions

A step towards the algorithm in Sect. 2.2 is an observation by Indyk and Motwani [IM98] that vectors with a small distance of \( d < \frac{n}{2} \) have a relatively large number of \( n - d \) matching components. In contrast, two randomly chosen vectors of weight \( \frac{n}{2} \) have a rather large distance of about \( \frac{n}{2} \) and therefore only a small number of also about \( \frac{n}{2} \) matching components. Therefore, a possible algorithm is to search for a subset \( J \subseteq [n] \) on which the two vectors of distance \( d \) are identical. This locality sensitive hashing technique allows to find vectors with a small distance much faster than vectors with a large distance, since the number of good \( J \) (on which the vectors are identical) is much higher.

The idea of [IM98] is to define a family of hash functions \( h_J \) parameterized by a subset \( J \subseteq [n] \) with \( |J| = \ell \approx \log_2(|L_1|) \). Given a vector \( x_1 \in \{0,1\}^n \), the hash value \( h_J(x_1) \in \{0,1\}^\ell \) is defined as the restriction of \( x_1 \) to the components of \( J \).

For each of these \( J \), we start with \( |L_1| \) empty buckets. For each element \( x_1 \in L_1 \), we compute the hash value \( h_J(x_1) \) and put the element in the bucket corresponding to this hash value. Notice that due to the choice of \( \ell \), the number of possible hash values exactly matches the number of buckets. Due to the uniformity of the elements in the list \( L_1 \), there should therefore be about one element in each of the buckets. Then we apply the same hash function to all elements in \( L_2 \). For each element in \( L_2 \), we compute the corresponding hash value and check for collisions in one of the buckets. For each collision, we check if the colliding vectors have a distance of \( d \) on the whole vector. If \( J \) is such that two vectors \( x_1 \in L_1 \) and \( x_2 \in L_2 \) with a Hamming distance \( \Delta(x_1, x_2) = d \) have the same hash value \( h_J(x_1) = h_J(x_2) \), the vectors are found in time about \(|L_1| + |L_2| \) hash function evaluations.

The problem is therefore reduced to finding such a special \( J \) with \( h_J(x_1) = h_J(x_2) \). This is simply done by repeating the algorithm for many uniformly random choices of \( J \). Notice that the number of good \( J \) is \( \binom{n-d}{\ell} \), because we have to find a subset of \( \ell \) components out of the identical \( n-d \) components of the two vectors \( x_1, x_2 \) with distance \( d \). Since the total number of possible \( J \) is simply \( \binom{n}{\ell} \), we have to try about \( \binom{n}{\ell}/\binom{n-d}{\ell} \) of uniformly random \( J \), until we find one that leads to \( h_J(x_1) = h_J(x_2) \).

This locality sensitive hashing approach therefore requires about \(|L_1| \binom{n}{\ell}/\binom{n-d}{\ell} \) hash function evaluations, with a parameter \( \ell \approx \log_2(|L_1|) \). Notice that for sub-exponential list sizes as in [IM98], this complexity simplifies to \( |L_1|^{1+\log_2\left(\frac{1}{1-2/d}\right)} \).

The technique is very efficient in practice and on the first sight looks like the right answer to tackle this problem. Unfortunately, the given time complexity is not sufficient to improve algorithms for the Decoding Problem introduced in Chapter 6, which is one of the main targets of this thesis. In order to achieve this improvement, we have to study the following adaptation of this idea, which leads to an asymptotically faster algorithm.

Filtering for Weight

The idea was adapted by Dubiner [Dub10], who replaces the hashing by filtering. That is, given a subset \( J \subseteq [n] \), two sub-lists \( L_1' \subseteq L_1 \) and \( L_2' \subseteq L_2 \) are created, which only keep those of the initial elements that have a certain weight \( w \) on the components defined by \( J \). This degree of freedom from fixed to something small leads to an asymptotic speed-up.

The remaining elements in \( L_1' \) and \( L_2' \) that pass the filter \( J \) are then simply processed naively, by comparing each element in \( L_1' \) with each element in \( L_2' \). Due to the fact that we
want to minimize the costs for this final naive search, the parameter \( w \) is chosen such that there is expected only one element in each of these sub-lists.

Similarly to the *locality sensitive hashing* approach, we have to repeat this procedure many times for uniformly chosen subsets \( \mathcal{J} \subseteq [n] \). In Sect. 2.4 we derive the required number of repetitions, until we find a *good* \( \mathcal{J} \) that implies that two vectors with a distance of \( d \) are in the corresponding sub-lists \( L'_1, L'_2 \). Once again, the probability of finding such special \( \mathcal{J} \) increases, the closer the vectors are.

Notice that in the described algorithm, just as in the *locality sensitive hashing* approach, for each of the chosen \( \mathcal{J} \), the whole input lists \( L_1, L_2 \) have to be processed in order to compute the sub-lists \( L'_1, L'_2 \). However, the new approach requires that these sub-lists can effectively be built in a time that corresponds to the size of the sub-lists (which are each of expected only one element). Otherwise, we do not achieve the desired improvement in the asymptotic complexity. Unfortunately, this issue isn’t analyzed rigorously in [Dub10]. One of the contributions of this thesis is therefore to present a technique that effectively eliminates the cost for the creation of \( L'_1, L'_2 \), which is presented in a high-level manner in Sect. 2.1.3 and analyzed in Sect. 2.2.

Dubiner [Dub10] shows that the time complexity of the approach is \(|L_1|^{1/(1-d/n)}\), if the list sizes are sub-exponential in \( n \). A second contribution of this thesis is to analyze the time complexity for the case of list sizes that are exponential in \( n \), which is also done in Sect. 2.2. In Corollary 25 we show that the result of Dubiner follows from our analysis.

### 2.1.2 zeroAND Problem

Let us take a look at a related problem, which we want to call zeroAND Problem in the following. Once again, assume we are given two lists \( L_1, L_2 \) of identical size \(|L_1| = |L_2|\). In difference to the NN Problem, the elements in the lists are chosen uniformly at random among all elements in \( \{0, 1\}^n \) with a certain Hamming weight \( w \in \mathbb{N} \). The task is to find all pairs of a vector \( x_1 \) from the first list and a vector \( x_2 \) from the second list such that their bitwise AND is always zero, i.e. the vectors don’t have matching 1’s in any component. As an example with \( n = 6 \) and \( w = 2 \), a problem instance could be \( L_1 = \{010100, 100100, 001010, 000101\} \) and \( L_2 = \{100001, 100100, 001100, 001001\} \). Thus, the vectors \( x_1 = 010100 \) and \( x_2 = 001001 \) would solve the problem, whereas \( x_1 \) together with \( x'_2 = 001100 \) wouldn’t be a solution due to the matching 1’s at the 4th component. Informally, the problem can be defined as follows.

**Definition 2** (zeroAND problem, informal). Let \( n, w \in \mathbb{N} \) with \( 0 \leq w \leq \frac{n}{2} \). Given two lists \( L_1, L_2 \subseteq \{0, 1\}^n \) of equal size \(|L_1| = |L_2|\) with uniformly random vectors of weight \( w \), find all pairs \((x_1, x_2) \in L_1 \times L_2\) with a bitwise AND that is always zero.

The following ideas are very related to those for the Nearest Neighbor Problem and at some level of abstraction they can even be seen as the same. This finally leads to the joint solution presented in Sect. 2.1.3.

**One Filtering**

Since we know for sure that for each 1 in \( x_1 \) the corresponding \( x_2 \) can’t be 1 at the same component, one approach can be to use the 1-components to reduce the number of possible candidates for \( x_2 \). For instance, a vector \( x_1 = 010100 \) would lead to possible candidates \( x_2 \in \{101000, 100010, 100001, 001010, 001001, 000011\} \). It is possible to search for all these
candidates in the list \( \mathcal{L}_2 \) efficiently, by sorting the list \( \mathcal{L}_2 \) once and applying binary search. However, it can quickly happen that the time complexity gets worse than naive, whenever the list of candidates gets too large. Since the number of candidates is simply \( \binom{n-w}{w} \), the time complexity of this approach becomes \( |\mathcal{L}_1| \cdot \binom{n-w}{w} \). Thus, this idea is only applicable if \( |\mathcal{L}_2| \ll \binom{n-w}{w} \), i.e. for very small (or very large) values of \( w \). Compared to the NN Problem, this approach is therefore similar to the brute-force idea. The technique can also be easily extended to a meet-in-the-middle approach with time complexity \( |\mathcal{L}_1| \cdot \left( \frac{(n-w)/2}{w/2} \right) \).

**Zero Block Technique**

Let us describe an idea that leads to a better time complexity by searching for a zero block in *one* of the vectors, again very similarly to the approach in the NN problem case.

\[
\begin{array}{c|c|c}
\hline
x_1 & 0 & w \\
\hline
x_2 & w' & w - w' \\
\ell & n - \ell \\
\hline
\end{array}
\]

**Figure 2.1: Zero Block Splitting**

As illustrated in Fig. 2.1, the idea is realized by searching for a subset \( \mathcal{J} \subseteq [n] \) of a size \( |\mathcal{J}| = \ell \) such that one of the vectors of a fixed solution pair \((x_1, x_2)\) doesn’t have 1’s on this part of the vector. Concretely, we want the vector \( x_1 \) to be 0 on the components of \( \mathcal{J} \) and have the remaining weight of \( w \) on the remaining components. This allows the corresponding vector \( x_2 \) to have an arbitrary weight \( w' \) on the components of \( \mathcal{J} \) such that the remaining weight is \( w - w' \). Notice that this makes all vectors that pass the filter valid solutions on the subset \( \mathcal{J} \), but not all vectors pass that filter. Therefore, a certain number of subsets \( \mathcal{J} \) has to be chosen uniformly at random, until one is found that leads to the presented distribution. Once such a \( \mathcal{J} \) is found, two lists \( \mathcal{L}_1' \subseteq \mathcal{L}_1 \) and \( \mathcal{L}_2' \subseteq \mathcal{L}_2 \) can be computed that consist only of vectors with this distribution. These lists are finally simply searched naively on the remaining \( n - \ell \) components.

Let us analyze the number of subsets \( \mathcal{J} \) that have to be chosen. First of all, the total number of possible subsets is \( \binom{n}{\ell} \). Notice that a solution to the problem has the property that there are \( w \) matching \((0,1)\) components, \( w \) matching \((1,0)\) components, no \((1,1)\) components and therefore \( n - 2w \) matching \((0,0)\) components as illustrated in Fig. 2.2.

\[
\begin{array}{c|c|c|c|c}
(0,0) & (0,1) & (1,0) & (1,1) \\
\hline
n - 2w & w & w & 0 \\
\hline
\end{array}
\]

**Figure 2.2: vector pair distribution on the whole vector**

On the other hand, we want that on the chosen set \( \mathcal{J} \) there is a weight of 0 in the elements of the first list and a weight of \( w' \) in the elements of the second list. This automatically implies that there still have to be no \((1,1)\) components and also no \((1,0)\) components, since otherwise the first vector would have a non-zero weight. Furthermore, the number of \((0,1)\) components
2.1 High-Level Idea

has to be \( w' \) in order to achieve the weight and thus the number of \((0, 0)\) components is \( \ell - w' \) as illustrated in Fig. 2.3.

<table>
<thead>
<tr>
<th></th>
<th>(0,0)</th>
<th>(0,1)</th>
<th>(1,0)</th>
<th>(1,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ell - w' )</td>
<td>( w' )</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.3: vector pair distribution on the chosen \( J \)

The number of good subsets \( \mathcal{J} \) that fulfill this distribution is therefore \( N := \binom{n-2w}{\ell-w'} \cdot \binom{w}{w} \), because out of the \( n-2w \) components that are both zero there have to be \( \ell - w' \) in the chosen subset and out of the \( w \) components \((0, 1)\) there have to be \( w' \). Hence, the expected number of random choices until a good one is found is \( \left( \frac{n}{\ell} \right) / N \).

This choice leads to list sizes \( |\mathcal{L}'_1| \approx |\mathcal{L}_1| \cdot \binom{n-\ell}{w} / \binom{w}{w} \) and \( |\mathcal{L}'_2| \approx |\mathcal{L}_2| \cdot \binom{\ell}{w} \cdot \binom{n-\ell}{w-w'} / \binom{w}{w} \). Similarly as in the NN Problem approach, the parameters \( \ell \) and \( w' \) should be chosen such that these lists consist of expected only one element. Notice that just as in the NN Problem, the whole input lists \( \mathcal{L}_1, \mathcal{L}_2 \) have to be processed, in order to compute the lists \( \mathcal{L}'_1, \mathcal{L}'_2 \), if the algorithm is implemented in a straightforward manner. Once again, the technique of computing these lists efficiently of Sect. 2.1.3 can be used. That is, the construction can effectively be done in time \( |\mathcal{L}'_1|, |\mathcal{L}'_2| \) instead of \( |\mathcal{L}_1| \) and \( |\mathcal{L}_2| \), which implies a time complexity of simply \( \left( \frac{n}{\ell} \right) / N \).

Blocks of Small Weight

As already seen before for the Nearest Neighbor Problem, the presented filtering for a fixed value 0 is outperformed by an approach that filters for something small, as illustrated in Fig. 2.4. The reason is that the cost of finding such a restrictive subset that is all-zero outweighs the cost of having a small increasing number of pairs that is also erroneous on the considered subset.

<table>
<thead>
<tr>
<th></th>
<th>( w_1 \neq 0 )</th>
<th>( w-w_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( w_2 )</td>
<td>( w-w_2 )</td>
</tr>
<tr>
<td>( \ell )</td>
<td>( n-\ell )</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.4: Small Weight Block Splitting

The analysis of the algorithm has to be slightly generalized compared to the previous technique. That is, the number of repetitions has to be adapted such that subsets \( \mathcal{J} \) are found with the distribution presented in Fig. 2.5. This makes the number of good subsets exactly \( N := \binom{w}{w_1} \cdot \binom{w}{w_2} \cdot \binom{n-2w}{\ell-w_1-w_2} \) and the time complexity improves slightly to \( \left( \frac{n}{\ell} \right) / N \).

<table>
<thead>
<tr>
<th></th>
<th>(0,0)</th>
<th>(0,1)</th>
<th>(1,0)</th>
<th>(1,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ell - w_1 - w_2 )</td>
<td>( w_2 )</td>
<td>( w_1 )</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.5: generalized vector pair distribution on the chosen \( J \)
2. Consistency Problem

2.1.3 A Joint Solution

In Sect. 2.1.1 and in Sect. 2.1.2, two very similar algorithmic ideas for solving the NN Problem and the zeroAND Problem are presented. In both cases, we have to choose an exponential number of subsets $J$ of size $\ell$ that decide on which components we want to filter by weight. For each of these $J$, we create two sub-lists that only contain those of the initial vectors that have a certain weight distribution. The number of repetitions guarantees that with overwhelming probability there is at least one $J$ such that any fixed vector pair in the initial lists also appears in one of the sub-lists. This idea is summarized in Fig. 2.6.

![Figure 2.6: main idea of the weight filter algorithms](image)

For at least one sub-list pair we have $(x_1^*, x_2^*) \in L_1^* \times L_2^*$ w.o.p.

Ideally, we would like to choose the parameters such that each of these sub-lists contains only a constant number of elements. In this case, we can simply compare the remaining pairs of one element in the first list and one element in the second list to check if they fulfill the required criterion, effectively using a naive algorithm that runs in constant time. However, in the ideas of the last two sections, we have to process the whole input lists for the computation of each of the sub-lists. In the following, we want to show how to compute the sub-lists in time $|L'_1|$ and $|L'_2|$ instead of $|L_1|$ and $|L_2|$, which applies to the filtering technique in [Dub10]. The hashing approach in [IM98] simply maps the elements to certain buckets, without reducing the number of the elements. This seems to inherently require that all the input elements are hashed.

The observation of [MOL15] is that if the lists are filtered such that there is still a (smaller) exponential number of elements in the sub-list pairs, we essentially have copies of the initial problem. In each of the cases, we have two lists with still uniform elements and at least in one of them we have guaranteed to have our solution vectors with a certain criterion (dependent on the problem we want to solve). Since we know a superior algorithm for these problems – exactly the weight technique we have used to create these lists – we can simply apply this technique recursively on each of the created list pairs (of smaller size) as illustrated in Fig. 2.7.

This recursive application simply chooses for each of the sub-list pairs a new collection of sets $J$, until it is guaranteed with overwhelming probability that if there is a vector pair fulfilling the criterion in the input, the pair is also in one of the output lists. The idea is to repeat the technique for a total number of $t$ levels, creating a depth $t$ search tree, reducing the number of elements in the lists from level to level. The parameter $t$ has to be chosen such that on the final level, the number of elements in the lists is so small – ideally 1 – such that a naive technique can be applied to those lists. However, the weight distributions change from level to level and it is not apparent how to analyze the algorithm and more specifically how to choose the parameters.
This is why we want to stick to this general concept, but modify it slightly to obtain a provable result. The main idea of \cite{MO15} is a preprocessing of the initial lists such that the \( n \) components of the vectors are subdivided into \( t \) disjoint parts. Since the required \( t \) can be shown to be constant, there is an efficient algorithm that enforces the following. On each of the \( t \) parts, we want to guarantee that we have the same vector distribution \textit{locally} as we have \textit{globally} on the whole vector. As an example for the NN Problem, for any fixed vector that is a solution to the problem, we want to enforce that the relative weight of the vectors on each part is \( \frac{1}{2} \), whereas their relative distance is \( d/n \). Notice that in order to be able to balance the time complexity, the \( t \) parts aren’t chosen of the same size. Instead, starting with the largest size, it decreases from part to part. Since this computation can be done in time polynomially in \( n \) (with a degree that is related to \( t \)), we can ignore it in our analysis that suppresses factors that are logarithmic in the exponential time complexity.

### Algorithm 1 SPLIT (informal algorithm)

1. **Input:** \( n \in \mathbb{N}, \ell \in \mathbb{N}, \mathcal{L}_1 \subseteq \{0, 1\}^n, \mathcal{L}_2 \subseteq \{0, 1\}^n \) \( \triangleright \) called with \( \ell = 1 \) (level 1)
2. **Output:** A list of \((x_1, x_2) \in \mathcal{L}_1 \times \mathcal{L}_2\) fulfilling a certain criterion.
3:  
4:  **if** \( \ell = 1 \) **then** \( \triangleright \) pre-computation
5:  **if** \( \ell = t + 1 \) **then** \( \triangleright \) solve the problem naively
6:  **else**
7:  Choose uniform \( \mathcal{J} \subseteq \mathcal{I}_\ell \) and create sub-lists \( \mathcal{L}_1', \mathcal{L}_2' \) by filtering for a certain weight.
8:  **return** \textsc{Naive}(\( \mathcal{L}_1, \mathcal{L}_2 \))
9:  **for** an exponential in \( n \) number of times **do**
10:  \( \mathcal{L}_{out} \leftarrow \emptyset \)
11:  **for** an exponential in \( n \) number of times **do**
12:  \( \mathcal{L}_{out} \leftarrow \mathcal{L}_{out} \cup \text{SPLIT}(n, \ell + 1, \mathcal{L}_1', \mathcal{L}_2') \) \( \triangleright \) solve the remaining part recursively
13:  **return** \( \mathcal{L}_{out} \)
The idea is summarized informally in the algorithm \texttt{Split}, as well as in Fig. 2.8 and is presented rigorously in the subsequent Sect. 2.2 in application to a more general \textit{Consistency Problem}. The algorithm starts by choosing an optimal depth and sizes and splits the problem into \( t \) sub-problems. The sub-problems are chosen such that the global properties of both vectors of any fixed solution pair also hold locally, which allows to solve these sub-problems of smaller size individually, all of them with the same relative time complexity. Good sets \( I_1, \ldots, I_t \) can be found with overwhelming probability after a polynomial number of choices of a uniform collection of sets. Therefore, if the algorithm \texttt{Split} is called a polynomial in \( n \) number of times.
with independent uniform choices of the sets, we can be sure that in at least one of these calls we have a good collection.

The main part of the algorithm is such that on each level $1 \leq \ell \leq t$, the algorithm is called recursively with shorter sub-lists. These sub-lists are computed by restricting to certain weights. That is, only elements with certain weights $w_1$ resp. $w_2$ on a chosen subset $J$ are kept in the sub-lists $L'_1, L'_2$. This is repeated an exponential number of times such that it is guaranteed that in at least one of the calls the fixed solution pair is in the sub-lists. The sub-lists are finally processed recursively with the same technique. On the final level $t + 1$, the whole vector of size $n$ is already processed. By construction, the sizes of these final lists are small and can therefore be merged naively, which means that the time complexity, suppressing polynomial factors, is simply the overall number of repetitions.

The parameters can be chosen such that the time complexities on each level – which are dominated by traversing the input lists an exponential number of times – are approximately the same. The algorithm is analyzed rigorously in the subsequent section, where it is applied to a more general problem.

## 2.2 Consistency Problem

In this section, we want to generalize the zeroAND and the Nearest Neighbor to a more general Consistency Problem. The definition of this problem requires some notions that are introduced in Sect. 2.2.1. These are used to formally define the problem and to present an algorithm for the problem in Sect. 2.2.2. Eventually, the correctness and the time complexity of the algorithm is proven rigorously in Sect. 2.2.3.

### 2.2.1 Preliminaries

In order to simplify notation in the following sections and chapters and due to the mainly interest in asymptotical time complexities ignoring polynomial overheads, we assume that the vector size $n$ is arbitrary divisible. That is, for any constant $c > 0$, we always assume that $cn$ is a positive integer. This allows us to ignore rounding issues and more importantly to consider everything relatively to $n$. The following definition of a weight distribution (i.e. a probability distribution on $\mathbb{Z}$) is in the spirit of this restriction on $n$.

**Definition 3** (weight distribution). Let $w : \mathbb{Z} \to [0, 1]$ and denote $\sigma(w) := \{j \in \mathbb{Z} \mid w(j) \neq 0\}$ the support of $w$. The map $w$ is called a weight distribution, if $\sum_{j \in \mathbb{Z}} w(j) = 1$. We denote the set of all weight distributions as $\mathcal{W}$.

In the following, we use weight distributions in order to describe the relative number of times each $j \in \mathbb{Z}$ appears as a component of a vector $x \in \mathbb{Z}^n$. Moreover, we always assume that $|\sigma(w)|$ is constant, i.e. independent of $n$. Notice that in the zeroAND Problem introduced in Sect. 2.1.2, we have vectors in $\{0, 1\}^n$ with a Hamming weight (number of ones) of $w \in \mathbb{N}$. The weight distribution $w$ in this case is therefore $w(1) = w/n, w(0) = 1 - w/n$ and $w(j) = 0$ for all $j \in \mathbb{Z} \setminus \{0, 1\}$. Regarding the restriction on arbitrary divisible $n$, the following definition of sets of vectors with a certain weight distribution becomes meaningful and is useful in the following description and analysis of the algorithms.
Definition 4 (weighted (sub)set). Let \( n \in \mathbb{N} \) and \( \mathbf{w} \in \mathcal{W} \). Then denote
\[
Z^n[\mathbf{w}] := \{ \mathbf{x} \in \mathbb{Z}^n \mid \mathbf{x} \text{ has a weight distribution } \mathbf{w} \}
\]
and for all sets \( \mathcal{I} \subseteq [n] \) denote
\[
Z^n_\mathcal{I}[\mathbf{w}] := \{ \mathbf{x} \in \mathbb{Z}^n \mid \mathbf{x} \text{ has a weight distribution } \mathbf{w} \text{ on } \mathcal{I} \text{ and is } 0 \text{ on } [n] \setminus \mathcal{I} \}.
\]

As an example, assuming \( \mathbf{w} \in \mathcal{W} \) is such that \( \mathbf{w}(1) = \frac{1}{3} \) and \( \mathbf{w}(0) = \frac{2}{3} \), we have \( Z^6_{\{4,5,6\}}[\mathbf{w}] = \{000100, 000010, 000001\} \), i.e. the first three components are always 0, whereas the remaining three components have a weight of 1. If vectors consist of several parts with certain weights, the sets can therefore be combined by component-wise addition. As an example, assume \( \mathbf{w} \in \mathcal{W} \) is as defined above and \( \mathbf{w}' \in \mathcal{W} \) is such that \( \mathbf{w}'(1) = \frac{1}{2} \), then \( Z^5_{\{1,2\}}[\mathbf{w}'] + Z^5_{\{3,4,5\}}[\mathbf{w}] = \{10100, 10010, 10001, 01100, 01010, 01001\} \) with \( \mathcal{A} + \mathcal{B} := \{ a + b \mid a \in \mathcal{A}, b \in \mathcal{B} \} \) for sets \( \mathcal{A}, \mathcal{B} \).

In order to count the number of elements in sets \( Z^n[\mathbf{w}] \), we want to generalize the binomial coefficient, which is also called a multinomial coefficient. For the sake of simplicity we want to use the following notation.

Definition 5 (multinomial coefficient). Let \( n \in \mathbb{N} \) and \( \mathbf{w} \in \mathcal{W} \). We denote
\[
\binom{n}{\mathbf{w} \cdot n} := \frac{n!}{\prod_{j \in \mathbb{Z}} (\mathbf{w}(j) \cdot n)!}.
\]

If \( |\sigma(\mathbf{w})| = 2 \), we obtain the classical binomial coefficient, for \( |\sigma(\mathbf{w})| \geq 3 \) the object becomes a multinomial coefficient. Notice that slightly abusing notation, we also make use of binomial and trinomial coefficients. That is, for weight distributions \( \mathbf{w} \) with \( |\sigma(\mathbf{w})| = 2 \) such that \( \sigma(\mathbf{w}) = \{k_1, k_2\} \) and \( \mathbf{w}(k_1) = \delta \), we denote \( \binom{n}{\delta \cdot n} := \binom{n}{n-k_1} \), ignoring the second component due to the fact that \( \mathbf{w}(k_2) = 1 - \delta \). Analogously, for weight distribution \( \mathbf{w} \) with \( |\sigma(\mathbf{w})| = 3 \) such that \( \sigma(\mathbf{w}) = \{k_1, k_2, k_3\} \), \( \mathbf{w}(k_1) = \delta_1 \) and \( \mathbf{w}(k_2) = \delta_2 \), we denote \( \binom{n}{\delta_1 \cdot n, \delta_2 \cdot n} := \binom{n}{n-k_1} \), ignoring the third component due to the fact that \( \mathbf{w}(k_3) = 1 - \delta_1 - \delta_2 \) is determined by \( \delta_1, \delta_2 \).

The following lemma shows a relation of multinomial coefficients to weighted sets.

Lemma 6 (counting weight). Let \( n \in \mathbb{N}, |\mathcal{I}| \subseteq [n], \mathbf{w} \in \mathcal{W} \). Then \( |Z^n[\mathbf{w}]| = \binom{n}{\mathbf{w} \cdot n} \) and \( |Z^n_\mathcal{I}[\mathbf{w}]| = \binom{n}{\mathbf{w} \cdot \mathcal{I}} \).

Proof. Obviously, the second equality directly follows from the first, because \( |Z^n_\mathcal{I}[\mathbf{w}]| = |Z^{|\mathcal{I}|}[\mathbf{w}]| \).
This can be shown by giving a bijective map between the sets that removes the fixed zeros.

The first equality can be shown combinatorially as follows. Let w.l.o.g. \( \mathbf{w}(1), \ldots, \mathbf{w}(t) \neq 0 \) and \( \mathbf{w}(j) = 0 \) for all \( j \not\in [t] \). If all \( n \) components are distinguishable, the number of vectors of the set (choosing components without replacement) is simply \( n! \). If there are parts with identical components, for each of these parts of \( \mathbf{w}(j) \cdot n \) elements we double counted the part \( (\mathbf{w}(j) \cdot n)! \) times. Since each group is independent, the double count for all the groups is therefore the product of the double count of each individual group. The number without double counting is therefore \( n! \) divided by this product, which is exactly the definition of the multinomial coefficient.

The following definition of the entropy helps to describe the multinomial coefficient asymptotically.
2.2 Consistency Problem

Definition 7 (entropy). Let \(\mathbf{w} \in \mathcal{W}\). Then \(\mathcal{H}(\mathbf{w}) := - \sum_{j \in \mathbb{Z}} w(j) \cdot \log_2(w(j))\).

Once again, we abuse notation in the case of the binary entropy and ternary entropy function. That is, for a \(\mathbf{w}_2 \in \mathcal{W}\) with \(w_2(1) = \delta\) and \(w_2(0) = 1-\delta\), we denote \(\mathcal{H}(\delta) := \mathcal{H}(\mathbf{w}_2)\). Analogously, for a \(\mathbf{w}_3 \in \mathcal{W}\) with \(w_3(1) = \delta\), \(w_3(2) = \delta\) and \(w_3(0) = 1-\delta\), we denote \(\mathcal{H}(\delta,\delta) := \mathcal{H}(\mathbf{w}_3)\).

Furthermore, assume we have a \(v \neq 0\) such that \(\mathbf{w}_2' \in \mathcal{W}\) with \(w_2'(1) = \delta/v\) and \(w_2'(0) = 1-\delta/v\). Then we want to define \(\mathcal{H}_v(\delta) := v \cdot \mathcal{H}(\mathbf{w}_2')\). Analogously for \(\mathbf{w}_3' \in \mathcal{W}\) with \(w_3'(1) = \delta/v\), \(w_3'(2) = \delta/v\) and \(w_3'(0) = 1-\delta/v\) we want to define \(\mathcal{H}_v(\delta,\delta) := v \cdot \mathcal{H}(\mathbf{w}_3')\). This directly implies \(\mathcal{H}_v(\delta,\delta) = v \cdot \mathcal{H}(\mathbf{w}_3')\).

The idea behind this notation becomes apparent after we have established the following relation between the multinomial coefficient and the entropy.

Lemma 8 (multinomial vs. entropy). Let \(n \in \mathbb{N}\) and \(\mathbf{w} \in \mathcal{W}\). Then \(\binom{n}{\mathbf{w} \cdot n} \approx \tilde{\Theta}(2^{\mathcal{H}(\mathbf{w}) \cdot n})\).

Proof. The well-known approximation by Stirling states that the factorial behaves as

\[
n! = \tilde{\Theta} \left( \left( \frac{n}{e} \right)^n \right),
\]

where \(e\) is Euler’s constant. By the definition of the multinomial coefficient we therefore have

\[
\binom{n}{\mathbf{w} \cdot n} = \frac{n!}{\prod_{j \in \mathbb{Z}} (w(j) \cdot n)!} = \tilde{\Theta} \left( \frac{\left( \frac{n}{e} \right)^n}{\prod_{j \in \mathbb{Z}} \left( \frac{w(j) \cdot n}{e} \right)^{w(j) \cdot n}} \right) = \tilde{\Theta} \left( \frac{1}{\prod_{j \in \mathbb{Z}} \frac{w(j)^{w(j) \cdot n}}{w(j) \cdot n}} \right) = \tilde{\Theta}(2^{\mathcal{H}(\mathbf{w}) \cdot n}) .
\]

Notice that due to our notation from above, for any \(v \neq 0\) we therefore have \(\binom{v \cdot n}{\delta/n} = \tilde{\Theta}(2^{\mathcal{H}_v(\delta/n)})\) as well as \(\binom{v \cdot n}{\delta_1/n,\delta_2/n} = \tilde{\Theta}(2^{\mathcal{H}_v(\delta_1,n,\delta_2,n)})\), which can be verified by replacing \(\mathbf{w}_2\) by \(\mathbf{w}_2'\) and \(\mathbf{w}_3\) by \(\mathbf{w}_3'\).

In the following, we want to describe problems of finding pairs of vectors that have a certain number of matching components. This leads to the following definition of joint distributions.

Definition 9 (joint distribution). Let \(\Gamma : \mathbb{Z} \times \mathbb{Z} \to [0,1]\) and additionally denote \(\sigma(\Gamma) := \{(j_1,j_2) \in \mathbb{Z} \times \mathbb{Z} \mid \Gamma(j_1,j_2) \neq 0\}\) the support of \(\Gamma\). The map \(\Gamma\) is called a joint distribution if \(\sum_{j_1,j_2 \in \mathbb{Z}} \Gamma(j_1,j_2) = 1\). We denote the set of all joint distributions as \(\mathcal{S}\).

Similarly to the weight distribution, we use the joint distribution to describe the relative number of times each \((j_1,j_2)\) appears as a component of a vector pair \((\mathbf{x}_1,\mathbf{x}_2) \in \mathbb{Z}^n \times \mathbb{Z}^n\) and assume that \(|\sigma(\Gamma)|\) is constant, i.e. independent of \(n\). As an example, the zeroAND Problem can be described by a joint distribution \(\Gamma(1,1) = 0, \Gamma(0,1) = \Gamma(1,0) = \frac{w}{n}\) and \(\Gamma(0,0) = 1 - 2w/n\). Concretely, this means that a vector pair with joint distribution \(\Gamma\) has no matching \((1,1)\) components, which is as required for the zeroAND Problem. Notice that the corresponding vectors both have a relative Hamming weight of \(w/n\), which implies a weight distribution \(\mathbf{w}\) with \(w(1) = w/n\) and \(w(0) = 1 - w/n\). On the other hand, not all vector pairs with the individual weight distribution \(\mathbf{w}\) have the joint distribution \(\Gamma\). The zeroAND Problem is exactly, given two lists of vectors with weight distribution \(\mathbf{w}\), to identify a pair that has a joint distribution of \(\Gamma\). The following definition is useful to compute the individual weight distributions of vector pairs with a certain joint distribution. In contrast to the zeroAND Problem, in which both input lists have vectors of the same weight distribution \(\mathbf{w}\), we want to allow different weights \(\mathbf{w}_1,\mathbf{w}_2\) in the following generalization of the problem.
Definition 10 \((\psi_1, \psi_2)\). Let \(\Gamma \in \mathcal{G}\) be a joint distribution. Then \(w_1 = \psi_1(\Gamma)\) resp. \(w_2 = \psi_2(\Gamma)\) are the corresponding weight distributions of the first resp. the second vector, i.e. \(w_1(j_1) = \sum_{j_2 \in \mathbb{Z}} \Gamma(j_1, j_2)\) and \(w_2(j_2) = \sum_{j_1 \in \mathbb{Z}} \Gamma(j_1, j_2)\).

Analogously to \(\mathbb{Z}[w]\), it is useful to describe a set of vector pairs that have a certain joint distribution.

Definition 11 (joint (sub)set). Let \(n \in \mathbb{N}\) and \(\Gamma \in \mathcal{G}\). Then denote
\[
(\mathbb{Z}^2)^n[\Gamma] := \{(x_1, x_2) \in \mathbb{Z}^n \times \mathbb{Z}^n \mid (x_1, x_2) \text{ has a joint distribution } \Gamma \}
\]
and for all sets \(I \subseteq \{1, \ldots, n\}\) denote
\[
(\mathbb{Z}^2)^n[I] := \{(x_1, x_2) \in \mathbb{Z}^n \times \mathbb{Z}^n \mid (x_1, x_2) \text{ has joint distribution } \Gamma \text{ on } I \text{ and is } (0, 0) \text{ on } \{1, \ldots, n\} \setminus I\}.
\]

Finally, the following rules are useful.

Definition 12 \((\leq, -, \cdot)\). Let \(w, w'\) be two weight distributions. It is \(w \leq 2w'\), if for all \(j \in \mathbb{Z}\) we have \(w(j) \leq 2w'(j)\). Analogously, let \(\Gamma, \Gamma'\) be two joint distributions. It is \(\Gamma \leq 2\Gamma'\), if for all \(i, j \in \mathbb{Z}\) we have \(\Gamma(i, j) \leq 2\Gamma'(i, j)\). Similarly, the operations \(2w' - w\) and \(2\Gamma' - \Gamma\) are performed component-wise, always guaranteeing that the result is a weight/joint distribution.

2.2.2 Problem and Algorithm

With the help of the collected notation we are now able to describe the Consistency Problem, which is to identify a pair of vectors with a certain joint distribution \(\Gamma\), given two lists of vectors with certain weight distributions \(w_1\) resp. \(w_2\). For our analysis, we require that certain sub-list sizes are upper bounded with a certain probability that depends on an \(\varepsilon > 0\). This upper bound is simply (up to a factor of \(2^\varepsilon\)) the expected value that is obtained if the elements in the initial lists follow the uniform distribution. The requirement is described in the following definition.

Definition 13 (concentrated). Let \(n \in \mathbb{N}, t \in \mathbb{N}\) with \(t \leq n\) and fix weight distributions \(w\) and \(h \leq 2 \cdot w\). Fix pairwise disjoint \(\mathcal{I}_1, \ldots, \mathcal{I}_t \subseteq \{1, \ldots, n\}\) with \(\bigcup_{\ell = 1}^t \mathcal{I}_\ell = \{1, \ldots, n\}\) and \(\mathcal{J}_1, \ldots, \mathcal{J}_t\) with \(\mathcal{J}_\ell \subseteq \mathcal{I}_\ell\) and \(|\mathcal{J}_\ell| = |\mathcal{I}_\ell|/2\) for all \(1 \leq \ell \leq t\). For all \(0 \leq \ell \leq t\) define
\[
\mathcal{L}[\ell] := \mathcal{L} \cap \left( \sum_{k=1}^\ell \mathbb{Z}^n_{\mathcal{J}_k}[h] + \sum_{k=1}^\ell \mathbb{Z}^n_{\mathcal{J}_k \setminus \mathcal{J}_k}[2 \cdot w - h] + \sum_{k=t+1}^\ell \mathbb{Z}^n_{\mathcal{J}_k}[w] \right).
\]

Then a list (multi-set) \(\mathcal{L} \subseteq \mathbb{Z}^n[w]\) is called concentrated, if for any constant \(\varepsilon > 0\) with a probability of \(1 - \mathcal{O}(2^{-\varepsilon})\) (over the choice of the list \(\mathcal{L}\)), for all \(0 \leq \ell \leq t\), we have
\[
|\mathcal{L}[\ell]| \leq 2(1 + 2^\varepsilon) \left(1 + \frac{|\mathcal{L}|}{n \cdot w} \cdot \prod_{k=1}^\ell \frac{|\mathcal{I}_k|/2}{(h \cdot |\mathcal{J}_k|/2)} \cdot \prod_{k=1}^\ell \frac{|\mathcal{I}_k|/2}{(2 \cdot w - h) \cdot |\mathcal{J}_k|/2} \cdot \prod_{k=t+1}^\ell \frac{|\mathcal{I}_k|}{w \cdot |\mathcal{J}_k|} \right).
\]

As there are two models of distribution of the elements in the initial lists that interest us for the further applications, we want to keep the definition more general by requiring these upper bounds. For any fixed collection of \(\mathcal{I}_1, \ldots, \mathcal{I}_t\) and \(\mathcal{J}_1, \ldots, \mathcal{J}_t\) that are chosen in the algorithm, the \(2(t+1)\) sub-lists that need to be upper bounded are defined as all elements in the initial lists that have certain weights on the components defined by the fixed sets.

In Sect. 2.4 as well as in Chapter 3 we show that these upper bounds hold assuming a certain distribution of the elements in the initial lists.
**Definition 14** (Consistency Problem). Let \( n \in \mathbb{N} \) and \( \Gamma \) be a joint distribution with corresponding weight distributions \( \mathbf{w}_1 = \psi_1(\Gamma) \) and \( \mathbf{w}_2 = \psi_2(\Gamma) \). In the \((n, \Gamma)\) Consistency Problem we are given two concentrated lists \( \mathcal{L}_1 \subseteq \mathbb{Z}^n[\mathbf{w}_1] \), \( \mathcal{L}_2 \subseteq \mathbb{Z}^n[\mathbf{w}_2] \) of size at most exponential in \( n \). The problem is to output a list that contains any fixed pair \((x_1, x_2) \in \mathcal{L}_1 \times \mathcal{L}_2\) with joint distribution \( \Gamma \), or an empty list if no such pair exists.

The algorithm CONSISTENCYSIEVE – which was introduced in special cases in [Dub10, MO, MO15] – is divided into two parts. In the main algorithm, a set of subsets is selected that guarantees a certain weight distribution for any fixed solution. This allows to subdivide the problem into \( t \) disjoint parts that can be solved individually. In the second, recursive algorithm CONSISTENCYRECURSIVE, the problem is solved step by step on the \( t \) parts by choosing subsets of the parts that divide the parts into two halves that are checked for certain weight distributions.

**Algorithm 2 CONSISTENCYSIEVE**

1: **Input:** \( n \in \mathbb{N}, \Gamma \in \mathfrak{S}, \mathcal{L}_1 \subseteq \mathbb{Z}^n[\psi_1(\Gamma)], \mathcal{L}_2 \subseteq \mathbb{Z}^n[\psi_2(\Gamma)], \varepsilon > 0 \)

2: **Output:** A list of \((x_1, x_2) \in \mathcal{L}_1 \times \mathcal{L}_2\) with joint distribution \( \Gamma \).

3: 
4: \( \lambda_1 \leftarrow \frac{1}{n} \cdot \log_2(|\mathcal{L}_1|) \), \( \lambda_2 \leftarrow \frac{1}{n} \cdot \log_2(|\mathcal{L}_2|) \) and \( \lambda \leftarrow \max\{\lambda_1, \lambda_2\} \)
5: Pick optimized parameters \( h_1 \leq 2 \cdot \mathbf{w}_1 \) and \( h_2 \leq 2 \cdot \mathbf{w}_2 \) such that \( z := z_1 = z_2 \) with
6: \( z_1 \leftarrow \mathcal{H}(\mathbf{w}_1) - \frac{1}{\varepsilon} \mathcal{H}(h_1) - \frac{1}{\varepsilon} \mathcal{H}(2 \cdot \mathbf{w}_1 - h_1) \) and \( z_2 \leftarrow \mathcal{H}(\mathbf{w}_2) - \frac{1}{\varepsilon} \mathcal{H}(h_2) - \frac{1}{\varepsilon} \mathcal{H}(2 \cdot \mathbf{w}_2 - h_2) \)
7: \( \mathcal{C} \leftarrow \{C \in \mathfrak{S} \mid |C| \leq 2 \cdot \Gamma, (C(i, j) \cdot n) \in \mathbb{N}_0 \text{ for each } i, j \in \mathbb{Z}, \psi_1(C) = h_1 \text{ and } \psi_2(C) = h_2\} \)
8: if \( h_1 = \mathbf{w}_1 \) or \( h_2 = \mathbf{w}_2 \) or \(|\mathcal{C}| = 0\) then \( \triangleright \text{special case} \)
9: return Naive(\( \mathcal{L}_1, \mathcal{L}_2, \Gamma \)) \( \triangleright \text{compare each element in the 1st list to each in the 2nd list} \)
10: \( y \leftarrow \max\{\lambda, \frac{1}{2} \cdot \mathbf{w}_1, \frac{1}{2} \cdot \mathbf{w}_2\}, \min_{C \subseteq \mathcal{C}} \{1 - \sum_{i,j,\Gamma(i,j)\neq 0} \mathcal{H}(\Gamma(i,j))(C(i,j)/2)\}\)
11: \( t \leftarrow \frac{\log(y - \lambda + \max\{0, \lambda_1 - \varepsilon\}) + \log(\lambda + \max\{0, \lambda_2 - \varepsilon\})}{\log(\lambda + \max\{0, \lambda_1 - \varepsilon\})} \cdot \log(y - \lambda + \max\{0, \lambda_2 - \varepsilon\}) \cdot \log(y - \lambda + \max\{0, \lambda_2 - \varepsilon\}) \}
12: \( s(1) = \frac{y - \lambda + \max\{0, \lambda_1 - \varepsilon\}}{y}, s(\ell) = s(\varepsilon) \cdot s(\varepsilon - 1) \text{ for all } 2 \leq \ell \leq t. \)
13: \( \Lambda_b^{[\ell]} \leftarrow 2(1 + 2^{\eta t}) \cdot \left(1 + \frac{|\mathcal{C}|}{(m_{b \cdot s^n}) \cdot \prod_{k=1}^\ell \left(\left|\mathcal{I}_k\right|/2\right) \cdot \prod_{k=1}^\ell \left(\left|\mathcal{I}_k\right|/2\right) \cdot \prod_{k=\ell+1}^t \left(\left|\mathcal{I}_k\right|/2\right) \right) \forall b, \ell \)
14: \( \mathcal{L}_{\text{out}} \leftarrow \emptyset \)
15: for \( \varepsilon \cdot n \cdot \left(\prod_{i,j \in \mathbb{Z}} (\Gamma(i,j) \cdot n)\right) \text{ times do} \)
16: Choose uniformly random partition of \([n]\) into \( \mathcal{I}_1, \ldots, \mathcal{I}_t \) with \(|\mathcal{I}_\ell| = s(\ell) \cdot n \forall 1 \leq \ell \leq t. \)
17: \( \mathcal{L}_1 \leftarrow \text{all } x_1 \in \mathcal{L}_1 \text{ with weight } \mathbf{w}_1 \text{ on each individual of the } t \text{ parts} \)
18: \( \mathcal{L}_2 \leftarrow \text{all } x_2 \in \mathcal{L}_2 \text{ with weight } \mathbf{w}_2 \text{ on each individual of the } t \text{ parts} \)
19: \( \mathcal{L}_{\text{out}} \leftarrow \mathcal{L}_{\text{out}} \cup \text{CONSISTENCYRECURSIVE}[n, C, \Gamma, \varepsilon, t, \mathcal{I}_1, \ldots, \mathcal{I}_t, \Lambda_b^{[\ell]} \forall b, \ell](\mathcal{L}_1, \mathcal{L}_2, 1) \)
20: return \( \mathcal{L}_{\text{out}} \)

In the first part of CONSISTENCYSIEVE, optimal weight parameters \( h_1 \) and \( h_2 \) are chosen depending on the weight parameters \( \mathbf{w}_1 \) resp. \( \mathbf{w}_2 \) of the initial lists. The algorithm continues by creating a set \( \mathcal{C} \) of all joint distributions \( C \leq 2 \Gamma \) that have corresponding weight distributions \( h_1 \) and \( h_2 \). If \(|\mathcal{C}| = 0\) or for the special choice \( h_1 = \mathbf{w}_1 \) or \( h_2 = \mathbf{w}_2 \), the problem is solved with the naive algorithm. If \(|\mathcal{C}| \geq 1\), the set allows to compute the parameter \( y \), which determines the required number of repetitions in the \( t \) steps of the algorithm CONSISTENCYRECURSIVE. This parameter, together with the parameters \( z = z_1 = z_2 \), which define how much the lists are shortened during the process, determine the time complexity of the algorithm.
Depending on the computed parameters, the required number \( t \) of parts in which the \( n \) components of the vectors are split is determined. Notice that it is verified in the proof of correctness that \( t \) is always a positive real number. Also, due to the fact that all parameters that determine \( t \) are constant, also \( t \) is constant. Furthermore, a weight distribution \( s \) is computed that determines the sizes of each of these \( t \) parts. It is verified in the proof that these choices lead to \( t \) individual time complexities that are the same up to polynomial overheads and are the same as the final complexity of naively solving problems with small-sized list pairs. In the final step of the precomputation, the upper bounds on the lists are computed. Notice that it is guaranteed that for the good choice of \( \mathcal{I}_1, \ldots, \mathcal{I}_t \) and \( \mathcal{J}_1, \ldots, \mathcal{J}_t \) these upper bounds hold with overwhelming probability. Therefore, these upper bounds can be applied to all lists, even for bad choices. This allows to have a guaranteed upper bound on the time complexity of the algorithm that doesn’t depend on the distribution of the input lists or the coins of the algorithm.

The final step of the algorithm \( \text{CONSISTENCYSIEVE} \) is to repeatedly choose independent uniform subsets \( \mathcal{I}_1, \ldots, \mathcal{I}_t \), until one is found s.t. the fixed solution pair has a joint distribution of \( \Gamma \) on each of the \( t \) parts. We show in the \( \mathcal{I} \)-lemma that the required number of repetitions that guarantees to find at least one good collection of the \( \mathcal{I}_1, \ldots, \mathcal{I}_t \) is polynomial in \( n \). Finally, the algorithm \( \text{CONSISTENCYRECURSIVE} \) is called with lists \( \mathcal{L}_1', \mathcal{L}_2' \) that are reduced to elements with these special weights.

\begin{algorithm}
\caption{\textsc{ConsistencyRecursive}}
\begin{algorithmic}[1]
\STATE \textbf{Global Input:} \( n, \mathcal{C}, \Gamma, \varepsilon, t, \mathcal{I}_1, \ldots, \mathcal{I}_t, \Lambda_1^{[1]}, \ldots, \Lambda_1^{[\ell]}, \Lambda_2^{[1]}, \ldots, \Lambda_2^{[\ell]} \)
\STATE \textbf{Input:} \( \mathcal{L}_1, \mathcal{L}_2, \ell \)
\STATE \textbf{Output:} A list of \( (x_1, x_2) \in \mathcal{L}_1 \times \mathcal{L}_2 \) with joint distribution \( \Gamma \).
\IF {\( \ell = t + 1 \)}
\STATE \textbf{return} \( \text{NAIVE}(\mathcal{L}_1, \mathcal{L}_2, \Gamma) \)
\ELSE
\STATE \( \mathcal{L}_{\text{out}} \leftarrow \emptyset \)
\FOR {\( \varepsilon n \cdot \left( \frac{|\mathcal{I}_\ell|}{|\mathcal{I}_\ell|/2} \right) \left( \sum_{\mathcal{C} \in \mathcal{C}} \prod_{i,j \in \mathcal{Z}} \left( \frac{\Gamma(i,j) \cdot |\mathcal{I}_\ell|/2}{\mathcal{C}(i,j) \cdot |\mathcal{I}_\ell|/2} \right) \) times}
\STATE \( \mathcal{J}_\ell \leftarrow \text{partition}(\mathcal{I}_\ell) \quad \triangleright \quad \mathcal{J}_\ell \subseteq \mathcal{I}_\ell \) with \( |\mathcal{J}_\ell| = \frac{1}{2} \cdot |\mathcal{I}_\ell| \) is chosen uniformly at random
\STATE \( \mathcal{L}_1' \leftarrow \text{all } x_1 \in \mathcal{L}_1 \) with weight \( h_1 \) on \( \mathcal{J}_\ell \), but limited s.t. \( |\mathcal{L}_1'| \leq \Lambda_1^{[\ell]} \)
\STATE \( \mathcal{L}_2' \leftarrow \text{all } x_2 \in \mathcal{L}_2 \) with weight \( h_2 \) on \( \mathcal{J}_\ell \), but limited s.t. \( |\mathcal{L}_2'| \leq \Lambda_2^{[\ell]} \)
\STATE \( \mathcal{L}_{\text{out}} \leftarrow \mathcal{L}_{\text{out}} \cup \text{CONSISTENCYRECURSIVE}(\mathcal{L}_1', \mathcal{L}_2', \ell + 1) \)
\ENDFOR
\STATE \textbf{return} \( \mathcal{L}_{\text{out}} \)
\end{algorithmic}
\end{algorithm}

Once the precomputation of \( \text{CONSISTENCYSIEVE} \) is done, the algorithm \( \text{CONSISTENCYRECURSIVE} \) is straightforward. Starting with the first block of components \( \mathcal{I}_1 \), the choice of partitions \( \mathcal{J}_1 \) is repeated, until one is found such that the fixed solution pair has the required weight distribution on this partition. For each choice, two sub-lists \( \mathcal{L}_1', \mathcal{L}_2' \) of all elements with relative weights \( h_1 \) resp. \( h_2 \) on \( \mathcal{J}_1 \) are created, limited to the upper bound \( \Lambda_1^{[\ell]} \) resp. \( \Lambda_2^{[\ell]} \). For all of these sub-list pairs on the first part, the algorithm is called recursively on the second part, on which again an exponential number of sub-lists is created. Once all \( n \) components on all \( t \) parts are processed, the final lists – that have a small bounded size of \( 2^{\varepsilon n} \) – are processed naively, creating an arbitrary small \( \varepsilon \)-overhead in the exponential time complexity.
2.2 Consistency Problem

2.2.3 Analysis

In this section, we prove that the algorithm is correct and compute its time complexity. Before we present the main theorem, we prove two lemmas. In the \( I \)-lemma, we compute the number of repetitions in the algorithm \textsc{ConsistencySieve} that are required such that the joint distribution \( \Gamma \) splits evenly on the \( t \) parts of sizes \( s(1), \ldots, s(t) \). The number of repetitions is chosen such that the probability of an error is exponentially small in \( n \).

**Lemma 15 (\( I \)-lemma).** Let \( n \in \mathbb{N} \) and \( \Gamma \) be a joint distribution. Let \( t \in \mathbb{N} \) be constant and \( s \) be an arbitrary weight distribution with \( s(k) = 0 \) for all \( k \notin [t] \). Fix an arbitrary \( (x_1, x_2) \in (\mathbb{Z}^2)^n[\Gamma] \).

Then for any constant \( \varepsilon > 0 \), after choosing

\[
\varepsilon n \cdot \left( \frac{n}{\mathbf{s} \cdot n} \right) / \left( \prod_{i,j \in \mathbb{Z}} \frac{\Gamma(i, j) \cdot n}{\Gamma(i, j) \cdot \mathbf{s} \cdot n} \right)
\]

tuples of pairwise disjoint sets \( I_1, \ldots, I_t \subseteq [n] \) with \( |I_k| = s(k) \cdot n \) for all \( k \in [t] \) uniformly and independently at random, there is at least one choice such that

\[
(x_1, x_2) \in \sum_{k=1}^t (\mathbb{Z}^2)_I^{|I_k|} \Gamma
\]

with a probability of at least \( 1 - 2^{-\varepsilon n} \).

**Proof.** The number of possible choices of the \( I_1, \ldots, I_t \) is \( \binom{n}{\mathbf{s} \cdot n} \), since we choose one out of \( t \) different blocks, unordered (within each block), without replacement. On the other hand, a good choice is such that the already \( \Gamma \)-distributed pair \( (x_1, x_2) \) is also \( \Gamma \)-distributed on each of the \( t \) parts. This means that also each \( (i, j) \)-block has to be distributed on the \( t \) parts according to \( s \), with \( \frac{\Gamma(i, j) \cdot n}{\Gamma(i, j) \cdot \mathbf{s} \cdot n} \) possibilities for each part. The number of good choices is therefore the product of all individual choices and the probability for picking a good choice uniformly at random is \( p = \frac{1}{\mathbf{s} \cdot n} \) good choices/all choices. Thus after choosing \( \varepsilon n / p \) such \( I_1, \ldots, I_t \) independently and uniformly at random, the probability that none of the choices is good is less than \((1 - p)^{\varepsilon n / p} \leq 2^{-\varepsilon n} \).

Once it is guaranteed that at least for one of the choices \( I_1, \ldots, I_t \) there is a joint distribution of \( \Gamma \) on each individual of the \( t \) parts, we want to make sure with the \( J \)-lemma that in each of the parts the number of repetitions is high enough such that for at least one the choices of \( J_1, \ldots, J_t \) we have that there is a weight of \( h_1 \) resp. \( h_2 \) on that part.

**Lemma 16 (\( J \)-lemma).** Let \( n \in \mathbb{N} \) and let \( \Gamma \) be a joint distribution with corresponding weight distributions \( w_1 = \psi_1(\Gamma) \) and \( w_2 = \psi_2(\Gamma) \). Let \( n \in \mathbb{N} \), \( I \subseteq [n] \), \( x_1 \in \mathbb{Z}^n_I[w_1] \) and \( x_2 \in \mathbb{Z}^n_I[w_2] \). Let \( h_1 \leq 2 \cdot w_1 \), \( h_2 \leq 2 \cdot w_2 \) be arbitrary weight distributions and denote \( C \) the set of all joint distributions \( C \leq 2 \Gamma \) with \((C(i, j) \cdot n) \in \mathbb{N}_0\) for each \( i, j \in \mathbb{Z} \) and \( \psi_1(C) = h_1 \), \( \psi_2(C) = h_2 \).

Then for any constant \( \varepsilon > 0 \), after choosing

\[
\varepsilon n \cdot \left( |I| / |I|/2 \right) / \left( \prod_{C \in C} \prod_{i,j \in \mathbb{Z}} \frac{\Gamma(i, j) \cdot |I|}{C(i, j) \cdot |I|/2} \right)
\]

sets \( J \subseteq I \) with \( |J| = |I|/2 \) uniformly and independently at random, for at least one of these chosen \( J \) we have

\[
x_1 \in \mathbb{Z}^n_J[h_1] + \mathbb{Z}^n_{I \setminus J}[2 \cdot w_1 - h_1] \quad \text{and} \quad x_2 \in \mathbb{Z}^n_J[h_2] + \mathbb{Z}^n_{I \setminus J}[2 \cdot w_2 - h_2]
\]

with a probability of at least \( 1 - 2^{-\varepsilon n} \).
Proof. The overall number of subsets \( J \subseteq I \) with \( |J| = |I|/2 \) is \( \left( \frac{|I|}{2} \right) \). Let us continue to determine the number of good subsets \( J \) that lead to \( x_1 \) and \( x_2 \) of that special form. That is, we simultaneously need a weight of \( h_1 \) in \( x_1 \) and a weight of \( h_2 \) in \( x_2 \) on the chosen subset \( J \). This means, given the joint distribution \( \Gamma \), we first of all have to consider the set \( C \) of all possible weight distributions \( C \) that can be built by choosing exactly half the weight of \( \Gamma \) with the additional property that the weight distribution of the first vector is \( h_1 \) and the weight distribution of the second vector is \( h_2 \). For each of these \( C \in \mathcal{C} \), for all \( \Gamma(i,j) \cdot |I|/2 \) components \((i,j)\), we are now free to choose \( C(i,j) \cdot |I|/2 \) components. Combining all these chosen components to a set \( J \), we obtain a joint distribution of \( C \) on this set and therefore also the required individual weight distributions \( h_1 \) and \( h_2 \). For each \( C \), the number of possible choices for \( J \) is \( \prod_{i,j \in \mathbb{Z}} (\Gamma(i,j) \cdot |I|/2) \). Since sets \( J \) are not double counted for different elements from \( \mathcal{C} \), the overall number of good sets \( J \) is simply the sum over the numbers for each individual \( C \). Denoting \( p \) the probability of finding a good \( J \), choosing it uniformly at random, we see that after choosing \( \varepsilon n/p \) such \( J \) independently, the probability that none of the \( J \) is good is less than \((1 - p)^{\varepsilon n/p} \leq 2^{\varepsilon n}\). \( \square \)

These two lemmas are now used to prove the following Consistency Theorem, which proves the correctness and the time complexity of the algorithm CONSISTENCY-SIEVE. The theorem is a generalization of results presented in [Dub10, MOI, MO15].

**Theorem 17** (Consistency Theorem). Let \( (L_1, L_2) \) be an instance of a \((n, \Gamma)\) Consistency Problem with corresponding weight distributions \( w_1 = \psi_1(\Gamma) \) and \( w_2 = \psi_2(\Gamma) \) such that \( L_1 \subseteq \mathbb{Z}^n[w_1] \) and \( L_2 \subseteq \mathbb{Z}^n[w_2] \). Then for any constant \( \varepsilon > 0 \) and any weight distributions \( h_1 \leq 2 \cdot w_1 \) and \( h_2 \leq 2 \cdot w_2 \) such that \( z := z_1 = z_2 \) and \( |C| \geq 1 \), the algorithm CONSISTENCY-SIEVE with input \((n, \Gamma, L_1, L_2, \varepsilon)\) solves the instance with a probability of \( 1 - O(2^{-\varepsilon n}) \) in time

\[
\tilde{O}\left((2^n)^{y+\max\{0, \lambda_1-\varepsilon\}+\max\{0, \lambda_2-\varepsilon\}+2\varepsilon}\right)
\]

with \( y := \max\{\lambda, z+\varepsilon, \min_{C \subseteq \mathcal{C}} \left\{ 1 - \sum_{i,j, \Gamma(i,j) \neq 0} H(\Gamma(i,j))(C(i,j)/2) \right\} \} \), \( \lambda := \max\{\lambda_1, \lambda_2\} \) with \( \lambda_1 := \frac{1}{2} \cdot \log_2(|L_1|) \), \( \lambda_2 := \frac{1}{2} \cdot \log_2(|L_2|) \), reduction values \( z_1 := H(w_1) - \frac{1}{2} H(h_1) - \frac{1}{2} H(2 \cdot w_1 - h_1) \), \( z_2 := H(w_2) - \frac{1}{2} H(h_2) - \frac{1}{2} H(2 \cdot w_2 - h_2) \) and \( C \) being the set of all joint distributions \( C \leq 2 \cdot \Gamma \) with \( \psi_1(C) = h_1 \) and \( \psi_2(C) = h_2 \).

Proof. It is easy to see that we have \( y > 0 \), \( z_1 \geq 0 \) and \( z_2 \geq 0 \) by definition, whereas \( z_1 = 0 \) or \( z_2 = 0 \) is only possible if \( h_1 = w_1 \) or \( h_2 = w_2 \), which leads to the naive algorithm. We therefore can assume that we have \( y, z_1, z_2 > 0 \) with \( z := z_1 = z_2 \) in the main algorithm. It chooses \( t = \frac{\log(y) - \log(z)}{\log(y) - \log(z)} \), which is always a positive real number due to \( y > z \), \( y \geq \lambda \) per definition of \( y \) and \( z = \lambda + \max\{0, \lambda - z\} \geq 0 \). Also, notice that \( t \) is constant, since all the parameters that \( t \) depends on are at most constant. We have \( |I_1| = \frac{y \cdot \lambda \cdot (\lambda + \max\{0, \lambda - z\} + \max\{0, \lambda - z\} + z)}{y} \cdot n \) and \( |I_2| = \frac{z}{y} \cdot |I_{t-1}| \) for all \( 2 \leq \ell \leq t \). Notice that it can be easily verified that this choice of the parameters leads to \( \sum_{t=1}^{t} |I_\ell| = n \), which means it is a valid choice to partition \([n]\) into \( t \) parts and also shows that \( s \) is indeed a weight distribution.

Let us begin the main part of the proof by computing an upper bound on the time complexity of the algorithm, which can be done independently of the correctness of the algorithm due to the given upper bounds on all lists. The loop in the main algorithm CONSISTENCY-SIEVE is
repeated $\varepsilon n \cdot \binom{n}{\mathfrak{s}_n}/\left(\prod_{i,j \in \mathbb{Z}} (\Gamma(i,j)-n)\right)$ times for a weight distribution $\mathfrak{s}$. Since $\binom{n}{\mathfrak{s}_n} = \widetilde{\Theta}(2^{H(\mathfrak{s})-n})$ and $(\Gamma(i,j)-n) = \widetilde{\Theta}(2^{H(i,j)-n})$, it follows that the number of repetitions is polynomial in $n$ due to the fact that $|\sigma(\Gamma)|$ is constant and $\sum_{i,j \in \mathbb{Z}} \Gamma(i,j) = 1$. Notice that

$$\prod_{k=1}^{\ell} \left( \frac{|I_k|/2}{b_k \cdot |I_k|/2} \right) \prod_{k=1}^{\ell} \left( 2 \cdot w_{b_k} - b_k \cdot |I_k|/2 \right) \prod_{k=\ell+1}^{n} \left( \frac{|I_k|}{w_{b_k} \cdot |I_k|} \right) = \widetilde{O} \left( 2^{-(\varepsilon n \cdot \mathfrak{s}_n)} \right),$$

since $(|I_k|/2)/b_k = \widetilde{O}(2^{H(b_k)-|I_k|/2})$, $(|I_k|/2)/{(2w_{b_k} - b_k \cdot |I_k|)/2} = \widetilde{O}(2^{H(2w_{b_k} - b_k \cdot |I_k|)/2})$, $(|I_k|/w_{b_k} \cdot |I_k|) = \widetilde{O}(2^{H(w_{b_k} \cdot |I_k|)})$ and $(n|I_k|/w_{b_k} \cdot |I_k|) = \widetilde{O}(2^{H(w_{b_k} \cdot |I_k|)})$, which directly implies that the upper bounds in the algorithm are $\Lambda^{[\ell]}_b = \widetilde{O} \left( 2^{\varepsilon n + \max \{0, \lambda(n-z \cdot |I_1|+\ldots+|I_{\ell}|)\}} \right)$ for each $b \in \{1, 2\}$ and all $1 \leq \ell \leq t$. Thus, in the recursive part of the algorithm, on each level $1 \leq \ell \leq t$, we create a total number of $\varepsilon n \cdot N_{\ell}$ sub-list pairs with $N_{\ell} := (\prod_{|I_\ell|/2}/\left( \sum_{C \in \mathcal{P}_{i,j \in \mathbb{Z}} \left( \left| I_{C(i,j)} \right| / |I_{\ell}| \right) \right)$ such that the first list is of size $\widetilde{O} \left( 2^{\varepsilon n + \max \{0, \lambda(n-z \cdot |I_1|+\ldots+|I_{\ell}|)\}} \right)$ and the second one of size $\widetilde{O} \left( 2^{\varepsilon n + \max \{0, \lambda(n-z \cdot |I_1|+\ldots+|I_{\ell-1}|)\}} \right)$. We have $N_{\ell} = \widetilde{O}(2^{\varepsilon n |I_\ell|})$, since $N_{\ell} \leq (|I_\ell|/2)\left( \max_{C \in \mathcal{P}_{i,j \in \mathbb{Z}} \left( \left| I_{C(i,j)} \right| / |I_{\ell}| \right) \right) \left( |I_\ell|/2 \right) = \widetilde{O}(2^{|I_\ell|})$ and $(\prod_{i,j \in \mathbb{Z}} \left( \left| I_{C(i,j)} \right| / |I_{\ell}| \right) = \widetilde{O}(2^{|I_\ell|}).$

The complexity on each level $1 \leq \ell \leq t$ is $\widetilde{O}(2^{\varepsilon n + \max \{0, \lambda(n-z \cdot |I_1|+\ldots+|I_{\ell}|)\}})$ for each $b \in \{1, 2\}$. Since $y \cdot |I_\ell| = z \cdot |I_{\ell-1}|$ for all $2 \leq \ell \leq t$, all these upper bounds can be once more upper bounded by $\widetilde{O}(2^{\varepsilon n + \max \{0, \lambda(n-z \cdot |I_1|+\ldots+|I_{\ell-1}|)\}})$, which is $\widetilde{O}(2^{|I_\ell|})$, resp. $\widetilde{O}(2^{\varepsilon n + \max \{0, \lambda(n-z \cdot |I_1|+\ldots+|I_{\ell-1}|)\}})$ for each $b \in \{1, 2\}$ due to the choice of $|I_\ell|$. In the last step, we have $\widetilde{O}(2^{|I_{\ell}|})$ list pairs of size at most $\widetilde{O}(2^{\varepsilon n + \max \{0, \lambda(n-z \cdot |I_1|+\ldots+|I_{\ell-1}|)\}})$ resp. $\widetilde{O}(2^{\varepsilon n + \max \{0, \lambda(n-z \cdot |I_1|+\ldots+|I_{\ell-1}|)\}})$. The final naive check for consistency has therefore also a time complexity of $\widetilde{O}(2^{|I_\ell|})$.

Let us move to the correctness of the algorithm and fix, if existent, an arbitrary $(\mathfrak{x}_1, \mathfrak{x}_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ that is also $\Gamma$-distributed. Notice that otherwise the algorithm correctly outputs an empty list, since no vector pair passes the final naive check. It follows directly from the $I$-lemma that for at least one of the chosen sets $I_1, \ldots, I_t$ we have $(\mathfrak{x}_1, \mathfrak{x}_2) \in \sum_{\ell=1}^{t} (2^{\mathfrak{s}_n})^{\Gamma(i,j)-n}[\Gamma]$ with a probability of at least $1 - 2^{-\mathfrak{s}_n}$. Now, if existent, fix a call of CONSISTENCYRECURSIVE with such good chosen sets. This algorithm creates a tree with an exponential number of children at each node. The idea is that we want to guarantee that at each level there is at least one list pair that contains $(\mathfrak{x}_1, \mathfrak{x}_2)$, which is shown with the help of the $J$-lemma. Fix a level $1 \leq \ell \leq t$. If $(\mathfrak{x}_1, \mathfrak{x}_2)$ is in one of the lists on level $\ell-1$ (with level 0 consisting only of the original input lists), we want to show that with a probability of at least $1 - 2^{-\mathfrak{s}_n}$ there is one $J_\ell$ chosen such that $\mathfrak{x}_1$ has a weight of $h_1$ on $J_\ell$ and $\mathfrak{x}_2$ has a weight of $h_2$ on $J_\ell$. This would mean that the pair is also in one of the lists on level $\ell$, unless some elements are removed from one of the lists due to exceeding of the upper bound.

Denote $(\mathfrak{x}_1, \mathfrak{x}_2)$ the $I_\ell$-part of $(\mathfrak{x}_1, \mathfrak{x}_2)$. Then due to the $J$-lemma we know that after choosing at least $\mathfrak{s}_n \cdot N_{\ell}$ uniform subsets $J_\ell$, with a probability of at least $1 - 2^{-\mathfrak{s}_n}$ there is at least one $J_\ell$ such that $\mathfrak{x}_1$ has a weight of $h_1$ on $J_\ell$ and $\mathfrak{x}_2$ has a weight of $h_2$ on $J_\ell$, which consequently also holds for $(\mathfrak{x}_1, \mathfrak{x}_2)$.

This means that with a probability of at least $1 - (t+1) \cdot 2^{-\mathfrak{s}_n}$, for at least one call of CONSISTENCYRECURSIVE, the pair $(\mathfrak{x}_1, \mathfrak{x}_2)$ is contained in the input list and in at least one list pair on each level, unless the lists grow too large and exceed the upper bound. Fixing the good $I_1, \ldots, I_t, J_1, \ldots, J_t$ that guarantee $(\mathfrak{x}_1, \mathfrak{x}_2)$ to be in one of the list pairs on the last
level, we know due to the concentration property that with a probability of \(1 - \mathcal{O}(2^{-en})\) all the corresponding lists are bounded with the upper bounds

\[
\Lambda_b^{[\ell]} = 2 \cdot (1 + 2^{en}) \cdot \left(1 + \frac{\mathcal{L}_b}{(w_b \cdot n)} \cdot \prod_{k=1}^\ell \left(\frac{|\mathcal{I}_k|/2}{\ell_b \cdot |\mathcal{I}_k|/2}\right) \cdot \prod_{k=1}^\ell \left((2 \cdot w_b - \ell_b) \cdot |\mathcal{I}_k|/2\right) \cdot \prod_{k=t+1}^t \left(\frac{|\mathcal{I}_k|}{w_b \cdot |\mathcal{I}_k|}\right)\right)
\]

of the algorithm. Therefore, the algorithm succeeds with a probability of \(1 - \mathcal{O}(2^{-en})\). \(\square\)

The algorithm CONSISTENCIESIEVE has a time complexity which depends on the concrete choice of the joint distribution \(\Gamma\) as well as on the parameters \(\ell_1\) and \(\ell_2\) chosen by the algorithm. Unfortunately, there is no apparent way of choosing these parameters in full generality. Therefore, we consider interesting special cases like the zeroAND Problem or the Nearest Neighbor Problem and compute optimized parameters for each of them. However, even for some of the special cases we already have to rely on numerical optimization, since we obtain multidimensional nonlinear functions that have to be minimized. Throughout the thesis, we solve these optimization problems with the help of the Mathematica 10.2 computer algebra program \([\text{Res15}]\). This tool can never guarantee to find optimal results, but the choices of \(\ell_1\) and \(\ell_2\) output by the tool are certainly valid solutions to the problem, since the result is proven for any choice. It is left as an open problem to find optimal choices that do not rely on numerics.

The question of optimality also arises in the specific choice of the algorithm. Although the presented algorithm is the best known for all the considered special cases, we can never guarantee that there isn’t a better one. The algorithm follows a weight restriction idea that fits very well into the Consistency Problem framework. However, simply checking for a certain weight on a certain part of the vectors surely doesn’t have to be the best strategy, especially in full generality. An open problem is therefore to find algorithms with a better asymptotic complexity or to prove a lower bound that matches the presented complexity.

Another open problem is the question of the practicality of the algorithm. In order to be able to analyze the algorithm, factors that are polynomial in \(n\) are suppressed. Unfortunately, the degree of these polynomial can grow very large, leading to impractical algorithms especially for relatively small values of \(n\). A very promising approach in rendering the algorithm practical is presented in the context of an algorithm for the Shortest Vector Problem by Becker, Gama and Joux \([\text{BGJ15}]\). In this work, a variant of the algorithm is used in a more direct manner. That is, the precomputation of the \(\mathcal{I}\)-lemma, which is the main source of the polynomial overheads, is removed. This leads to a heuristic algorithm that operates on the whole vector instead on \(t\) disjoint parts of \([n]\), but still has a recursion depth of \(t\). Now, instead of searching for subsets \(\mathcal{J}\) in each individual of the \(t\) parts, the subsets are chosen on the whole part. Once again, the sizes of the lists are therefore reduced step by step until the final lists consist of only a small number of elements. The issue of this approach is that the \(t\) different steps are not independent any more. That is, the subsets \(\mathcal{J}\) of two different steps are clearly not disjoint. These dependencies lead to the fact that the list shortening probabilities are not as expected and the lists may stay larger as hoped. An extreme example would be that the sets \(\mathcal{J}\) of two different steps are chosen to be the same, in which case the lists stay the same size. However, on the other hand the success probabilities are higher from step to step. That is, there are more choices that lead to a good \(\mathcal{J}\). If the increase in the probability and the slightly higher list sizes would compensate, this algorithm may become a practical alternative. It is left as an open problem to analyze the behavior of this modified algorithm. Another approach that makes use of the presented ideas in application to lattices is by Becker, Ducas, Gama and Laarhoven \([\text{BDGL15}]\).
2.3 Weight Match Problem

2.3.1 General Case

In this section, we want to use the algorithm for the Consistency Problem as a building block for an algorithm for the so-called Weight Match Problem. In this problem, we are once again given lists of vectors of size $n$ that have certain weight distributions $w_1$ resp. $w_2$. Given these lists of vectors, the Weight Match Problem asks to find pairs of vectors, one of each list, such that if added component-wise modulo $g$ for some $g \in \mathbb{N}$, have a weight distribution of $w'$.  

It can be easily seen that the zeroAND Problem can also be described as a Weight Match Problem by defining $w = w_1 = w_2$ such that $w(1) = w/n, w(0) = 1 - w/n$. Now, instead of describing the problem with a joint distribution $\Gamma$, we define $g$ to be some integer greater than 2, such that $0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1$ and $1 + 1 = 2$ and define $w'$ such that $w'(1) = 2w/n$ and $w'(0) = 1 - 2w/n$. The problem can therefore be seen as finding all vector pairs that, if added together (effectively over the integers), don’t have any 2-components, which makes them have $2w$ 1-components.  

The Nearest Neighbor Problem can also be described as a Weight Match Problem by defining $w = w_1 = w_2$ such that $w(1) = w(0) = 1/2$ and $g = 2$. In this case we have the addition modulo 2 such that $0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1$ and $1 + 1 = 2$, which means we define $w'$ such that $w'(1) = d/n$ and $w'(0) = 1 - d/n$. The problem is therefore to find all vector pairs that, if added modulo 2, have exactly $d$ ones. This directly matches the problem, because in the Nearest Neighbor Problem we are looking for vector pairs with a Hamming distance of $d$.  

The Weight Match Problem can therefore be seen as a more intuitive way to model the discussed problems. Throughout the thesis, this is also the way the problems appear, like in Chapter 3 in the context of Subset Sum or in Chapter 6 in the case of Decoding. In the remaining part of the thesis, we therefore formulate the problems as special cases of the Weight Match Problem and use the Consistency Problem framework as a building block.  

The zeroAND Problem and the Nearest Neighbor Problem can be described as both a Consistency Problem or a Weight Match Problem uniquely. For this special problem description, the Weight Match Problem can therefore be solved with the algorithm for the Consistency Problem in a straightforward manner. The following definition of a so-called output weight distribution is useful to describe this special class.

**Definition 18** (output weight distribution). Let $\Gamma \in \mathcal{G}$ be a joint distribution and $g \in \mathbb{N}$. Then $\psi_{out}^g(\Gamma)$ is the weight distribution of the addition modulo $g$ of each component in $\Gamma$, i.e.

$$\psi_{out}^g(\Gamma)(k) = \sum_{(i,j) \in \mathbb{Z}^2, i+j = k \mod g} \Gamma(i,j) \text{ for each } k \in \mathbb{Z}_g.$$  

As an example, assume $\sigma(\Gamma) = \{(0,0), (0,1), (1,0), (1,1)\}$. Then $\psi_{out}^2(\Gamma)$ describes the weight distribution of all components of $\Gamma$ with the same parity, i.e. $\psi_{out}^2(\Gamma)(0) = \Gamma(0,0) + \Gamma(1,1)$ and $\psi_{out}^2(\Gamma)(1) = \Gamma(0,1) + \Gamma(1,0)$. Analogously, $\psi_{out}^3(\Gamma)$ is such that $\psi_{out}^3(\Gamma)(0) = \Gamma(0,0), \psi_{out}^3(\Gamma)(1) = \Gamma(0,1) + \Gamma(1,0)$ and $\psi_{out}^3(\Gamma)(2) = \Gamma(1,1)$.  

With the help of this definition, we want to define the Weight Match Problem such that there is a unique joint distribution $\Gamma$ with $\psi_1(\Gamma) = w_1$, $\psi_2(\Gamma) = w_2$ and $\psi_{out}^g(\Gamma) = w'$ for some $g \in \mathbb{N}$, which is always fulfilled for the special cases considered in this thesis. This restriction has the nice advantage that it allows a direct application of the algorithm CONSISTENCYSIEVE, because the problem is simply transformed into a Consistency Problem.
Notice that in general there might be polynomially in n many joint distributions \( \Gamma \) (with integer components \( \Gamma(i,j) \cdot n \)) for a given Weight Match Problem instance. In this case, the problem can simply be solved by applying the algorithm CONSISTENCYSIEVE to each of these \( \Gamma \). The time complexity (ignoring polynomial overheads) is then simply the maximum of the running times for each \( \Gamma \).

However, this unfortunately leads to a much more cumbersome notation, due to the fact that each of the \( \Gamma \) has a corresponding set \( \mathcal{C}[\Gamma] \) and the time complexity becomes the sum over all these individual components. Therefore, due to the fact that all relevant problems are covered by this simplified definition, we want to concentrate on problems with a unique \( \Gamma \) and leave the study of the more general class for future research.

**Definition 19** (Weight Match Problem). Let \( n, g \in \mathbb{N} \) and \( w_1, w_2, w' \) be weight distributions such that there is exactly one joint distribution \( \Gamma \) with \( \psi_1(\Gamma) = w_1 \), \( \psi_2(\Gamma) = w_2 \) and \( \psi_{\text{out}}(\Gamma) = w' \). In the \((n, g, w_1, w_2, w')\) Weight Match Problem we are given two concentrated lists \( L_1 \subseteq \mathbb{Z}^n[w_1] \) and \( L_2 \subseteq \mathbb{Z}^n[w_2] \) of size at most exponential in \( n \). The problem is to output a list that contains any fixed pair \((x_1, x_2) \in L_1 \times L_2\) with the property that \((x_1 + x_2 \mod g) \in \mathbb{Z}^n[w']\), or an empty list if no such pair exists.

In the following, we want to solve any Weight Match Problem by transforming it into a Consistency Problem by making use of the fact that there is a unique joint distribution \( \Gamma \).

**Algorithm 4** \textsc{WeightSieve}

1. **Input:** \( n, g \in \mathbb{N}, w_1, w_2, w' \in \mathbb{N}, L_1 \subseteq \mathbb{Z}^n[w_1], L_2 \subseteq \mathbb{Z}^n[w_2], \varepsilon > 0 \)
2. **Output:** A list of \((x_1, x_2) \in L_1 \times L_2\) with \((x_1 + x_2 \mod g) \in \mathbb{Z}^n[w']\).
3. Compute the unique joint distribution \( \Gamma \) with \( \psi_1(\Gamma) = w_1 \), \( \psi_2(\Gamma) = w_2 \) and \( \psi_{\text{out}}(\Gamma) = w' \).
4. **return** CONSISTENCYSIEVE\((n, \Gamma, L_1, L_2, \varepsilon)\)

Since the algorithm \textsc{WeightSieve} simply calls the algorithm CONSISTENCYSIEVE and the complexity of the precomputation in line 4 of \textsc{WeightSieve} can be neglected, the following Corollary follows directly from Theorem \[17\]

**Corollary 20** (Weight Match). Let \((L_1, L_2)\) be an instance of a \((n, g, w_1, w_2, w')\) Weight Match Problem with \( L_1 \subseteq \mathbb{Z}^n[w_1] \) and \( L_2 \subseteq \mathbb{Z}^n[w_2] \). Let \( \Gamma \) be the unique joint distribution with \( \psi_1(\Gamma) = w_1 \), \( \psi_2(\Gamma) = w_2 \) and \( \psi_{\text{out}}(\Gamma) = w' \). Then for any constant \( \varepsilon > 0 \) and any weight distributions \( h_1 \leq 2 \cdot w_1 \) and \( h_2 \leq 2 \cdot w_2 \) such that \( z := z_1 = z_2 \) and \( |C| \geq 1 \), the algorithm \textsc{WeightSieve} solves the instance with a probability of \( 1 - \mathcal{O}(2^{-\varepsilon n}) \) in time

\[
\tilde{O}\left((2^n)^{y + \max\{0, \lambda_1 - z\} + \max\{0, \lambda_2 - z\} + 2\varepsilon}\right)
\]

with \( y := \max\left\{\lambda, z + \varepsilon, \min_{C \in \mathcal{C}} \left\{1 - \sum_{i,j : \Gamma(i,j) \neq 0} \mathcal{H}^{\Gamma(i,j)}(C(i,j)/2)\right\}\right\}, \lambda := \max\{\lambda_1, \lambda_2\} \) with \( \lambda_1 := \frac{1}{2} \log_2(|L_1|) \), \( \lambda_2 := \frac{1}{2} \log_2(|L_2|) \), reduction values \( z_1 := \mathcal{H}(w_1) - \frac{1}{2} \mathcal{H}(h_1) - \frac{1}{2} \mathcal{H}(2 \cdot w_1 - h_1) \), \( z_2 := \mathcal{H}(w_2) - \frac{1}{2} \mathcal{H}(h_2) - \frac{1}{2} \mathcal{H}(2 \cdot w_2 - h_2) \) and \( \mathcal{C} \) being the set of all joint distributions \( C \leq 2 \cdot \Gamma \) with \( \psi_1(C) = h_1 \) and \( \psi_2(C) = h_2 \).
2.3.2 Random Weight Match Problem

Both the Consistency Problem and the Weight Match Problem require concentrated input lists, i.e. inputs that achieve upper bounds on sub-list sizes with overwhelming probability. In this section, we want to show that if the elements in the initial lists are chosen uniformly and pairwise independently at random, then this condition is fulfilled. The special choice of the lists is described in the following definition. Notice that in this section we replace the parameter \( n \) with \( m \) in order to avoid confusion in Chapter 6.

**Definition 21** (Random Weight Match Problem). Let \( m, g \in \mathbb{N} \) and \( w_1, w_2, w' \) be valid weight distributions such that there is exactly one joint distribution \( \Gamma \) with \( \psi_1(\Gamma) = w_1 \), \( \psi_2(\Gamma) = w_2 \) and \( \psi_{\text{out}}(\Gamma) = w' \). In the \((m, g, w_1, w_2, w')\) Random Weight Match Problem, we are given two lists \( L_1 \subseteq \mathbb{Z}^m[w_1] \) and \( L_2 \subseteq \mathbb{Z}^m[w_2] \) of size at most exponential in \( m \) with uniformly random and pairwise independent elements. The problem is to output a list that contains any fixed pair \((x_1, x_2)\) \( \in L_1 \times L_2 \) with \((x_1 + x_2 \mod g) \in \mathbb{Z}^m[w']\), or an empty list if no such pair exists.

**Lemma 22** (Random Weight Match). Let \((L_1, L_2)\) be an instance of a \((m, g, w_1, w_2, w')\) Random Weight Match Problem with \( L_1 \subseteq \mathbb{Z}^m[w_1] \) and \( L_2 \subseteq \mathbb{Z}^m[w_2] \). Let \( \Gamma \) be the unique joint distribution with \( \psi_1(\Gamma) = w_1 \), \( \psi_2(\Gamma) = w_2 \) and \( \psi_{\text{out}}(\Gamma) = w' \). Then for any constant \( \varepsilon > 0 \) and any weight distributions \( h_1 \leq 2 \cdot w_1 \) and \( h_2 \leq 2 \cdot w_2 \) such that \( z := z_1 = z_2 \) and \(|C| \geq 1\), the algorithm \textsc{WeightSieve} with input \((m, g, w_1, w_2, w', L_1, L_2, z, \varepsilon)\) solves the instance with a probability of \( 1 - O(2^{-\varepsilon m}) \) in time

\[
\tilde{O}\left((2^m)^y + \max(0, \lambda_1 - z) + \max(0, \lambda_2 - z) + 2\varepsilon\right)
\]

with \( y := \max\left\{ \lambda, z + \varepsilon, \min_{C \subseteq C} \left\{ 1 - \sum_{i,j,i,j \neq 0} (\mathcal{H}_i(i,j)(C(i,j)/2)) \right\} \right\} \), \( \lambda := \max\{\lambda_1, \lambda_2\} \) with \( \lambda_1 := \frac{1}{4} \cdot \log_2(|L_1|) \), \( \lambda_2 := \frac{1}{4} \cdot \log_2(|L_2|) \), reduction values \( z_1 := \mathcal{H}(w_1) - \frac{1}{2} \mathcal{H}(h_1) - \frac{1}{2} \mathcal{H}(2 \cdot w_1 - h_1) \), \( z_2 := \mathcal{H}(w_2) - \frac{1}{2} \mathcal{H}(h_2) - \frac{1}{2} \mathcal{H}(2 \cdot w_2 - h_2) \) and \( C \) being the set of all joint distributions \( C \leq 2 \cdot \Gamma \) with \( \psi_1(C) = h_1 \) and \( \psi_2(C) = h_2 \).

**Proof.** Due to the fact that the Random Weight Match Problem is simply a Weight Match Problem, the only thing we have to show is the property on the intermediate lists by making use of the uniformity of the input. Afterwards, we can simply use Corollary 20.

Fix an arbitrary constant \( t \in \mathbb{N} \), arbitrary weights \( h_1 \leq 2 \cdot w_1 \), \( h_2 \leq 2 \cdot w_2 \), arbitrary pairwise disjoint sets \( T_1, \ldots, T_t \subseteq [m] \) with \( \bigcup_{\ell=1}^t T_\ell = [m] \) and arbitrary \( J_1, \ldots, J_t \) with \( J_\ell \subseteq T_\ell \) and \( |J_\ell| = |T_\ell|/2 \) for all \( 1 \leq \ell \leq t \). Define \( S_{b}^{[\ell]} := \sum_{k=1}^t \mathbb{Z}^m_{J_\ell}[b] + \sum_{k=1}^\ell \mathbb{Z}^m_{J_{k+1}\setminus J_k}[2 \cdot w_b - b] + \sum_{k=\ell+1}^t \mathbb{Z}^m_{J_k}[w_b] \) with \( |S_{b}^{[\ell]}| = \prod_{k=1}^\ell \left( \frac{|J_k|/2}{(2 \cdot w_b - b)/2} \right) \prod_{k=1}^\ell \left( \frac{|J_k|/2}{(2 \cdot w_b - b)/2} \right) \prod_{k=\ell+1}^t \left( \frac{|J_k|/2}{m} \right) \). We want to show that for any constant \( \varepsilon > 0 \), any fixed \( 0 \leq \ell \leq t \) and any fixed \( b \in \{1, 2\} \), we have

\[ |L_b \cap S_{b}^{[\ell]}| \leq 2 \cdot (1 + 2^m) \cdot (1 + |L_b| \cdot |S_{b}^{[\ell]}|/(m \cdot m)) \]

with a probability of \( 1 - O(2^{-\varepsilon m}) \). Notice that this shows the lemma by making use of the union bound, because \( 1 - O(2(t+1) \cdot 2^{-\varepsilon m}) = 1 - O(2^{-\varepsilon m}) \).

For each element \( x_k \in L_b \) we want to define random variables

\[
X_k = \begin{cases} 
1 & \text{if } x_k \in S_{b}^{[\ell]} \\
0 & \text{otherwise}
\end{cases}
\]

and \( X := \sum_{k=1}^{|L_b|} X_k \). Thus the random variable \( X \) counts the number of elements in \( L_b \cap S_{b}^{[\ell]} \). Since each of the elements \( x_k \in L_b \) is uniform in \( \mathbb{Z}^m[w_b] \) and \( S_{b}^{[\ell]} \subseteq \mathbb{Z}^m[w_b] \), we know that the
probability $\mathbb{P}[X_k = 1] = |S_b^{[\ell]}|/|Z^m|_b[w_0]$ and therefore we know that the expected value is $\mathbb{E}[X] = |S_b^{[\ell]}|/|Z^m|_b[w_0]$. Notice that the variance $\mathbb{V}[X] = \mathbb{V}[\sum_k X_k] = \sum_k \mathbb{V}[X_k] = \sum_k (\mathbb{E}[X_k^2] - \mathbb{E}[X_k]^2) \leq \sum_k \mathbb{E}[X_k] = \mathbb{E}[X]$. We have $\mathbb{V}[\sum_k X_k] = \sum_k \mathbb{V}[X_k]$, because the $X_k$ are pairwise independent. Applying Chebyshev’s inequality in the case $\mathbb{E}[X] > 1$, we obtain
\[
\mathbb{P}[|X - \mathbb{E}[X]| \geq 2\varepsilon \mathbb{E}[X]] \leq \frac{\mathbb{V}[X]}{2\varepsilon^2 \mathbb{E}[X]^2} \leq \frac{1}{2\varepsilon^2 \mathbb{E}[X]} \leq 2^{-\varepsilon m},
\]
whereas in the case $\mathbb{E}[X] \leq 1$ we get
\[
\mathbb{P}[|X - \mathbb{E}[X]| \geq 2\varepsilon \mathbb{E}[X]] \leq \frac{\mathbb{V}[X]}{2\varepsilon \mathbb{E}[X]} \leq \frac{2^\varepsilon}{2 \varepsilon},
\]
which means that indeed with a probability of at least $1 - O(2^{-\varepsilon m})$ we have
\[
|L_b \cap S_b^{[\ell]}| \leq (1 + 2^\varepsilon) \cdot \max\{1, |L_b| \cdot |S_b^{[\ell]}|/|Z^m|_b[w_0]|\} \leq 2 \cdot (1 + 2^\varepsilon) \cdot \left(1 + |L_b| \cdot |S_b^{[\ell]}|/|Z^m|_b[w_0]|\right).
\]

Another way to choose the lists, which requires a little more theory, is discussed in Chapter 3.

This leads to algorithms for variants of the zeroAND Problem. The presented lemma, however, can be used to solve the Nearest Neighbor Problem of the subsequent section.

## 2.4 Nearest Neighbor Problem

The Nearest Neighbor Problem of the following definition was introduced in [MO15] as a tool for decoding of random linear codes and is already introduced informally at the beginning of this chapter. The results of that paper together with the application of the subsequent analysis are presented in Chapter 3. The formal definition of the problem is as follows.

**Definition 23 (NN problem).** Let $m \in \mathbb{N}$, $0 < \gamma < \frac{1}{2}$. In the $(m, \gamma)$-Nearest Neighbor (NN) problem, we are given $\gamma$ and two lists $L_1, L_2 \subseteq \mathbb{F}_2^m$, with uniform and pairwise independent vectors of size at most exponential in $m$. The problem is to output a list that contains any fixed pair $(x_1, x_2) \in L_1 \times L_2$ with Hamming distance $\gamma m$, or an empty list if no such pair exists.

In the subsequent section, we want to analyze the following algorithm **NEARESTNEIGHBOR**.

**Algorithm 5 NEARESTNEIGHBOR**

1. **Input:** $m \in \mathbb{N}, 0 < \gamma \leq \frac{1}{2}, L_1, L_2 \subseteq \mathbb{F}_2^m, \varepsilon > 0$
2. **Output:** A list of $(x_1, x_2) \in L_1 \times L_2$ with Hamming distance $\Delta(x_1, x_2) = \gamma m$.
3. 
4. Choose $w$ s.t. $w(1) = w(0) = \frac{1}{2}$ and $w'$ s.t. $w'(1) = \gamma$, $w'(0) = 1 - \gamma$ and set $L_{\text{out}} \leftarrow \emptyset$.
5. **for** all $0 \leq \alpha_0 m \leq m$ and all $0 \leq \alpha_1 m \leq m$ **do** $\triangleright$ poly($m$) many
6. **for** $\varepsilon m \cdot \left(\frac{m}{2}\right) / \left(\frac{\alpha_0 m}{2}, \frac{\alpha_1 m}{2}, \frac{\gamma - \alpha_0 m}{2}, \frac{\gamma - \alpha_1 m}{2}, \frac{1 - \gamma - \alpha_0 m}{2}, \frac{1 - \gamma - \alpha_1 m}{2}\right)$ times **do** $\triangleright$ poly($m$) many
7. $y \leftarrow \mathbb{F}_2^m \triangleright$ choose a uniformly random vector
8. $L_1' \leftarrow$ the vector $y$ added to all vectors in $L_1$ restricted to all vectors with weight $\frac{m}{2}$
9. $L_2' \leftarrow$ the vector $y$ added to all vectors in $L_2$ restricted to all vectors with weight $\frac{m}{2}$
10. $L_{\text{out}}' \leftarrow$ call WEIGHTSIEVE($m, 2, w, w', L_1', L_2', \varepsilon$) and add $(y, y)$ to all pairs in list
11. $L_{\text{out}} \leftarrow L_{\text{out}}' \cup L_{\text{out}}'$
12. **return** $L_{\text{out}}$
2.4 Nearest Neighbor Problem

2.4.1 Analysis

The algorithm NearestNeighbor solves the problem with the help of the algorithm Weight-Match, mainly by deriving the required weight distributions. Additionally, we use a re-randomization technique to guarantee a certain weight distribution on any fixed solution pair.

We want to concentrate on the case that \( L_1 \) and \( L_2 \) are roughly of the same size and therefore upper bound both lists with the size of the larger one. Notice that the algorithm requires an additional upper bound \( \lambda \leq 1 - \mathcal{H}(\frac{2}{3}) \) on the list sizes, where \( \mathcal{H}(\cdot) \) denotes the binary entropy function. However, if it is violated, the lists can be split into parts of a size that corresponds to the upper bound. Each pair of sub-problems can then be solved naively. In our application to decoding of random linear codes in Chapter 6 though, the lists are always bounded as required. In the following theorem, we show that the problem can be solved with the algorithm NearestNeighbor. Notice that in the following theorem, \( \mathcal{H}^{-1}(\cdot) \) denotes the inverse of the binary entropy function mapping to values between 0 and \( \frac{1}{2} \).

**Theorem 24** (Nearest Neighbor). Let \( (L_1, L_2) \) be an instance of a \((m, \gamma)\) Nearest Neighbor Problem with \( \lambda := \frac{1}{m} \cdot \log_2(\max(|L_1|, |L_2|)) \leq 1 - \mathcal{H}(\frac{2}{3}) \). Then for any constant \( \varepsilon > 0 \), the algorithm NearestNeighbor with input \((m, \gamma, L_1, L_2, \varepsilon)\) solves the instance with a probability of \( 1 - \mathcal{O}(2^{-\varepsilon m}) \) in time

\[
\tilde{O}\left(2^{(\lambda+3\varepsilon)m} + (2^m)^{(1-\gamma)}\left(1-\mathcal{H}\left(\frac{\mathcal{H}^{-1}(1-\lambda) - \frac{2}{3}}{1-\gamma}\right)\right)+2\varepsilon\right).
\]

**Proof.** The main idea of the proof is to transform the problem into a Random Weight Match Problem. Therefore, we need to define the parameters \( g, \mathbf{w}_1, \mathbf{w}_2 \) and \( \mathbf{w}' \) and to show that there is a unique joint distribution \( \Gamma \) corresponding to this choice. First of all, notice that the input lists in the NN problem have the same distribution, which means that \( \mathbf{w}_1 \) and \( \mathbf{w}_2 \) also have to be chosen the same. We choose \( \mathbf{w} = \mathbf{w}_1 = \mathbf{w}_2 \) s.t. \( \mathbf{w}(1) = \mathbf{w}(0) = \frac{1}{2} \). Notice that this is not yet the distribution of the elements in the input lists of the NN problem. However, we make use of the fact that due to the uniform choice of the vectors, it is most likely for them to have a Hamming weight of \( \frac{m}{2} \). Since we are looking for elements with a Hamming distance of \( \gamma m \), the distribution \( \mathbf{w}' \) is chosen such that \( \mathbf{w}'(1) = \gamma \) and \( \mathbf{w}'(0) = 1 - \gamma \). In order to match the addition in \( \mathbb{F}_2 \), we finally choose \( g = 2 \) in order to have \( 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1 \) and \( 1 + 1 = 0 \). Notice that this choice makes the output indeed be all vectors with Hamming distance \( \gamma m \), since \( \Delta(x_1, x_2) = \text{wt}(x_1 + x_2) \). This choice also leads to the unique joint distribution \( \Gamma \) of Fig. 2.9, which can be easily derived from the fact that the sum of the \((0,1)\) and \((1,1)\) components, as well as the sum of the \((1,0)\) and \((1,1)\) components have to be \( \frac{1}{2} \), whereas the sum of the \((0,1)\) and \((1,0)\) components have to be \( \gamma \). Finally, the sum of all components has to be 1.

\[
\begin{array}{c|c|c|c|c}
\Gamma & (0,0) & (0,1) & (1,0) & (1,1) \\
& \frac{1-\gamma}{2} & \frac{\gamma}{2} & \frac{\gamma}{2} & \frac{1-\gamma}{2} \\
\end{array}
\]

**Figure 2.9:** joint distribution \( \Gamma \) for the NN Problem

Let us assume for the moment that the vectors of a fixed solution pair both have a Hamming weight of \( \frac{m}{2} \). Then the original lists restricted to all elements with Hamming weight \( \frac{m}{2} \) are lists in \( \mathbb{Z}^m[\mathbf{w}] \) and have elements that are chosen pairwise independently and uniformly at random
and therefore are an instance of a Random Weight Match Problem. Denoting these modified lists $L'_1, L'_2$, the output of the call WeightSieve($m, 2, w, w', L'_1, L'_2, \varepsilon$) is therefore a direct solution to the NN problem.

Hence, due to Lemma 22, with these parameters the problem is solved with a probability of $1 - \mathcal{O}(2^{-\varepsilon m})$. Let us show the time complexity that we obtain with these parameters. It is computed by choosing $h = h_1 = h_2$ such that $h(1) = H^{-1}(1 - \lambda)$ and $h(0) = 1 - h(1)$. First of all, we notice that this is a valid choice, because $h \leq 2 \cdot w$ is fulfilled due to the fact that $0 \leq h(1) \leq 1$. Secondly, this concrete choice seems to be optimal for most interesting cases, as numerical numerical optimization indicates, though there doesn’t seem to be an apparent proof for that fact. Notice, however, that Lemma 22 is applicable to any choice, which means the result with this particular choice might not be optimal, but is surely correct. The choice of $h$ leads to parameters $z = z_1 = z_2 = \lambda$.

We want to continue by computing the set of joint distributions $C$. All the joint distributions $C$ in this set have to fulfill that both the sum of the $(0, 1)$ and the $(1, 1)$ components, as well as the sum of the $(1, 0)$ and the $(1, 1)$ components are $h(1)$. Also, the sum of all components is 1, which leaves one degree of freedom. Moreover, we choose the number of $(0, 1)$ and $(1, 0)$ components as $\tilde{\gamma}$. Again, this restrictive choice is motivated by numerical optimization that indicates its optimality.

\[
C = \left\{ \begin{array}{c|c|c|c}
(0, 0) & (0, 1) & (1, 0) & (1, 1) \\
1 - H^{-1}(1 - \lambda) - \tilde{\gamma} & 2 & 2 & 2 - H^{-1}(1 - \lambda) - \frac{\tilde{\gamma}}{2}
\end{array} \right\}
\]

Figure 2.10: joint distribution $C$ for the NN Problem

Notice that this is a valid choice due to the restriction $\lambda \leq 1 - H(\frac{\tilde{\gamma}}{2})$, i.e. all the components of $C$ are non-negative. This choice of $C$ then leads to a value $y = \{\lambda, z + \varepsilon, \tilde{\gamma}\}$ with

\[
\tilde{y} = 1 - \frac{1 - \gamma}{2} \cdot H\left(\frac{1 - H^{-1}(1 - \lambda) - \tilde{\gamma} - 2}{1 - \gamma}\right) - 2 \cdot \frac{\gamma}{2} \cdot H(1/2) - \frac{1 - \gamma}{2} \cdot H\left(\frac{H^{-1}(1 - \lambda) - \tilde{\gamma}}{1 - \gamma}\right),
\]

which simplifies to $\tilde{y} = (1 - \gamma) \left(1 - H\left(\frac{H^{-1}(1 - \lambda) - \tilde{\gamma}}{1 - \gamma}\right)\right)$. Due to the fact that $z = \lambda$, the presented time complexity therefore directly follows from Lemma 22.

What is left to show is that the fixed solution pair is indeed of the described weight or can be transformed into one. This can be done by creating polynomially many in $m$ copies of the problem with the hope that in at least one of these copies we have the desired distribution. Each of these copies is created by adding a uniformly chosen vector $y \in \{0, 1\}^m$ to both lists and afterwards restricting both lists to vectors with a weight $\frac{m}{2}$. It remains to show that the required number of copies is indeed polynomial such that this overhead can be neglected in the $\mathcal{O}$ notation and we indeed get the claimed time complexity.

In general we don’t know much about the distribution of the components in the fixed solution pair $(x_1, x_2)$ with distance $\gamma m$. However, we can parameterize it with parameters $\alpha_0, \alpha_1$ and have the joint distribution $\tilde{\Gamma}$ of Fig. 2.11

We want to show that any vector $y \in \{0, 1\}^m$ that has the joint distribution $Y$ in Fig. 2.12 of 1’s on the components of the solution pair transforms the solution pair into a pair with joint distribution $\Gamma$, if it is added to both vectors of the solution pair.
2.4 Nearest Neighbor Problem

\[ \tilde{\Gamma} \quad (0, 0) \quad (0, 1) \quad (1, 0) \quad (1, 1) \]
\[ \alpha_0 \quad \alpha_1 \quad \gamma - \alpha_1 \quad 1 - \gamma - \alpha_0 \]

Figure 2.11: actual joint distribution \( \tilde{\Gamma} \) for the NN Problem

\[ Y \quad (0, 0) \quad (0, 1) \quad (1, 0) \quad (1, 1) \]
\[ \frac{\alpha_0}{2} \quad \frac{\alpha_1}{2} \quad \frac{\gamma - \alpha_1}{2} \quad \frac{1 - \gamma - \alpha_0}{2} \]

Figure 2.12: required joint distribution \( Y \) for the 1’s of \( y \)

This can be shown as follows. The number of (0, 0) components that stay (0, 0) is \( \frac{\alpha_0}{2} \), whereas the number of (1, 1) components that get transformed to (0, 0) components is \( \frac{1 - \gamma - \alpha_0}{2} \). Therefore, the number of (0, 0) components of the transformed vector pair is \( \frac{1 - \gamma - \alpha_0}{2} \). Analogously, this can be shown for the other components.

It remains to show how many vectors \( y \in \{0,1\}^m \) exist such that out of \( \alpha_0 \) of the (0, 0) components of \( \tilde{\Gamma} \) there are \( \frac{\alpha_0}{2} \) ones in this vector, out of \( \alpha_1 \) of the (0, 1) components there are \( \frac{\alpha_1}{2} \) and so on. This number is simply \( \left(\frac{\alpha_0}{2}\right)^m \cdot \left(\frac{\alpha_1}{2}\right)^m \cdot \left(\frac{\gamma - \alpha_1}{2}\right)^m \cdot \left(\frac{1 - \gamma - \alpha_0}{2}\right)^m = \tilde{\Theta}(2^m) \). Since the number of vectors in \( \{0,1\}^m \) is simply \( 2^m \), the probability to find a vector that fulfills this distribution choosing a vector uniformly at random is the inverse of a polynomial in \( m \).

Notice that multiplying the number of repetitions by an extra \( \varepsilon m \) factor, it can be shown to find a vector fulfilling the distribution \( Y \) with a probability of at least \( 1 - 2^{-\varepsilon m} \) such that the probability of success of the overall algorithm doesn’t change qualitatively. □

We want to conclude this section with a corollary that leads to a result in \([Dub10]\) for list sizes that are sub-exponentially in \( m \). In our applications in the subsequent chapters, we are mainly interested in list sizes \( |L_1| \approx |L_2| \approx 2^\lambda m \) with a constant value of \( \lambda \). However, if the list sizes are sub-exponential, we get a \( \lambda \) that tends to 0. This implies the following corollary.

**Corollary 25.** In the case of a list sizes \( |L_1| \approx |L_2| \) that are sub-exponential in \( m \), we obtain a complexity exponent \( \lim_{\lambda \to 0} \frac{y}{\lambda} = \frac{1}{1 - \gamma} \), i.e. our complexity is \( \tilde{\Theta}(|L_1|^{\frac{1}{1 - \gamma}}) \).

**Proof.** Notice that we defined the inverse of the binary entropy function as \( H^{-1}(\cdot) \) and that \( H^{-1}(1) = \frac{1}{2} \). The derivative of the inverse of the binary entropy function is \( \frac{d}{dx} H^{-1}(x) = \log_2 \left( \frac{1}{x} - 1 \right) \) and the derivative of the inverse of the binary entropy function is \( (H^{-1}(1 - \lambda))' = \frac{-1}{\log_2 \left( \frac{1}{H^{-1}(1 - \lambda)} - 1 \right)} \). We obtain the result by the following calculation, using L’Hospital’s rule twice.

\[
\lim_{\lambda \to 0} \frac{y}{\lambda} = \lim_{\lambda \to 0} \frac{(1 - \gamma) \log_2 \left( \frac{1 - \gamma}{H^{-1}(1 - \lambda) - \frac{\gamma}{2}} - 1 \right)}{1 - \gamma \log_2 \left( \frac{1 - \gamma}{H^{-1}(1 - \lambda) - 1} \right)}
\]
\[
= \lim_{\lambda \to 0} \left( \frac{1 - \gamma}{H^{-1}(1 - \lambda) - \frac{\gamma}{2}} - 1 \right) / \left( \frac{1 - \gamma}{H^{-1}(1 - \lambda) - 1} \right)
\]
\[
= \lim_{\lambda \to 0} \left( \frac{1 - \gamma}{H^{-1}(1 - \lambda) - \frac{\gamma}{2}} - 1 \right)^{-1} \left( \frac{-1}{H^{-1}(1 - \lambda) - \frac{\gamma}{2}} \right) (H^{-1}(1 - \lambda))' = \lim_{\lambda \to 0} \left( \frac{-1}{H^{-1}(1 - \lambda) - \frac{\gamma}{2}} \right) (H^{-1}(1 - \lambda))' = \lim_{\lambda \to 0} \left( \frac{-1}{H^{-1}(1 - \lambda)^2} \right) = \frac{1}{1 - \gamma}
\]

□
Chapter 3

Subset Sum Problem

The Subset Sum Problem is best known in the special case of the Knapsack Problem [Kar72]: given \( n \) integers \( a_1, \ldots, a_n \) and a target integer \( s \), the problem is to find a subset of the integers that sums to that target. However, in this thesis we want to concentrate on combinatorial algorithms that work in any group. A first combinatorial non-trivial algorithm for the problem was given by Horowitz and Sahni [HS74]. More recent algorithms were proposed by Howgrave-Graham, Joux [HJ10] and Becker, Coron, Joux [BCJ11]. All these algorithms have in common that they don’t make use of the special structure of the underlying group, but only use the fact that there are \( n \) elements that have to sum up to a target element.

In fact, the same ideas can be applied to various other problems. For example, the NTRU Problem [HPS98] is to find a polynomial \( f \) of degree \( n \) with binary coefficients such that \( hf = g \) for some known polynomial \( h \) and implicitly known \( g \). This problem can thus be easily redefined as given \( h_1, \ldots, h_n \), to find a linear combination that sums to \( g \), learning \( f \) bit by bit. The Discrete Logarithm Problem [DH76] asks to find an \( n \)-bit integer \( x \) such that \( \alpha^x = \beta \) for known \( \alpha \) and \( \beta \). Once again, this problem can be easily redefined into finding a subset of \( a_1 := \alpha^0, \ldots, a_n := \alpha^{2^n-1} \) such that their product is \( \beta \), which allows to learn the binary representation of \( x \). The Decoding Problem [McE78] for binary linear codes asks to find a binary \( n \)-dimensional error vector \( e \) such that \( He = s \), where \( H \) is a parity check matrix and \( s \) is a syndrome. Also here, the problem can be redefined by splitting the matrix \( H \) into \( n \) parts and searching for a subset that sums to \( s \).

In the literature, all these problems that are very close to the Knapsack Problem were discussed individually [BCJ11, BGLS14, BJLM13, BJMM12, HJ10, MNT11, MO, MO14, MO15, Oze12] and all known combinatorial attacks for the Knapsack Problem were adapted to these problems. At some level of abstraction, however, all these problems and corresponding attacks are identical, as already pointed out in [HJ10]. The idea of this chapter is to establish a level of abstraction, by defining the problem in a more general manner that has all the presented applications as special cases, similarly to the framework in [BGLS14], but in a more rigorous manner. In this generalized Subset Sum Problem, we receive \( n \) group elements \( a_1, \ldots, a_n \) from an arbitrary finite abelian group. The problem is to find a linear combination of these group elements that sums to another known target group element. Then, we describe the best known algorithm for this whole class of problems for this generalization. Since some variations of the problems don’t ask for a binary linear combination – which means that the individual group elements are either part of the sum or not – but also allow to pick elements multiple times, the problem is additionally generalized in this manner.
3. Generalized Problem

The multiple choice is characterized by a weight distribution $w$, which is already introduced in Definition 3 of the previous chapter. This $w$ describes the relative number of how often each component is chosen. As already discussed in the previous chapter, for any fixed weight $w$ we assume that $n$ is chosen such that the absolute weight is always a non-negative integer. The reason is that we are mostly interested in asymptotical time complexities and don’t care about polynomial overheads. This simplifies the description and analysis of the algorithms a lot. In practical applications, this rounding issues can always be easily resolved at the cost of at most polynomial overheads and a much more obscure description of the algorithms. The definition of the following problem and the subsequent analysis is therefore meant to be in this asymptotical spirit. The general problem is defined as follows, where $G$ is a finite, abelian group with $|G| \geq 2$ throughout the thesis.

**Definition 26 (Subset Sum Problem).** Let $G$ be a group, $n \in \mathbb{N}$ and $w$ be a weight distribution. In the $(G, n, w)$ Subset Sum Problem we are given arbitrary $a \in G^n$ and $s \in G$. The problem is to output an $x \in \mathbb{Z}^n[w]$ with $\sum_{i=1}^{n} a_i \cdot x_i = s$, or $\bot$ if no such $x$ exists.

In order to simplify the notation throughout the thesis, we denote $f_a(x) := \sum_{i=1}^{n} a_i \cdot x_i$. One task of this thesis is to find the best known algorithms for the Subset Sum Problem both in general and for interesting special cases. In the remaining part of this section, we want to discuss some algorithms for solving the Subset Sum Problem which are basic and work in any group $G$. In the subsequent section, we study more advanced algorithms that impose restrictions on the group $G$ and the choice of the parameters $a$ and $s$.

3.1.1 Brute Force

The most basic algorithm for solving the problem is a brute-force technique. This deterministic algorithm simply enumerates all possible vectors $x$ with the given weight and of the given size and checks for each of these candidates, if its target value $f_a(x)$ equals the given target $s$. Obviously, the algorithm succeeds for any Subset Sum Problem instance. Since in the worst case the whole search space $\mathbb{Z}^n[w]$ with $|\mathbb{Z}^n[w]| = \tilde{O}(2^{H(w)n})$ has to be searched, the time complexity of the algorithm is $\tilde{O}(2^{H(w)n})$, whereas the space complexity is polynomial in the input length.

```
Algorithm 6 BruteForce
1: Input: $G$, $n \in \mathbb{N}$, $w \in \mathcal{W}$, $a \in G^n$, $s \in G$
2: Output: An $x \in \mathbb{Z}^n[w]$ with $f_a(x) = s$.
3: for each $x \in \mathbb{Z}^n[w]$ do
4: if $f_a(x) = s$ then
5: return $x$
6: return $\bot$
```

Notice that this algorithm can be easily extended to output all solutions to the problem, instead of only the first solution that appears without increasing the time, but possibly the space complexity. As can be seen in the following analysis, it is usually harder to output all solutions instead of only one.
3.1.2 Meet-in-the-Middle

The meet-in-the-middle approach was introduced by Horowitz and Sahni [HS74] as an algorithm for the Knapsack Problem. The same technique, a so-called time-memory trade-off, can be applied in the more general setting of our Subset Sum Problem. The idea is that by storing (and reusing) certain elements, we obtain a reduced time complexity compared to the brute-force approach. A major technique in handling these stored lists of elements is binary search, which allows to find any fixed element in time the logarithm of the size of the lists. The application of binary search requires sorting the list, which also only introduces a logarithmic overhead.

In order to describe the main idea, let's assume there is a solution \( \bar{x} \in \mathbb{Z}^n[w] \) to the Subset Sum Problem such that also \( \bar{x} \in \mathbb{Z}^{n/2}[w] \times \mathbb{Z}^{n/2}[w] \) holds, i.e. it can be split into a left and a right hand part with the same relative weight as the original vector as illustrated in Fig. 3.1.

![Figure 3.1: required splitting for meet-in-the-middle](image)

Thus we know that there is an \( \bar{x}_1 \in \mathbb{Z}^{n/2}[w] \times \{0\}^{n/2} \) and an \( \bar{x}_2 \in \{0\}^{n/2} \times \mathbb{Z}^{n/2}[w] \) such that \( \bar{x}_1 + \bar{x}_2 = \bar{x} \). Due to the linearity of \( f_a(x) \), we also know that \( f_a(\bar{x}_1) + f_a(\bar{x}_2) = f_a(\bar{x}_1 + \bar{x}_2) = f_a(\bar{x}) = s \). The idea of the algorithm is therefore to store the value \( f_a(\bar{x}_1) \) for all possible \( \bar{x}_1 \in \mathbb{Z}^{n/2}[w] \times \{0\}^{n/2} \) in a list \( \mathcal{L} \), which makes sure that also \( f_a(\bar{x}_1) \in \mathcal{L} \). Afterwards, the list is sorted to be efficiently searchable via binary search. Then for all possible \( \bar{x}_2 \in \{0\}^{n/2} \times \mathbb{Z}^{n/2}[w] \) we check if \( s - f_a(\bar{x}_2) \) is in the list \( \mathcal{L} \), which guarantees that the correct \( \bar{x}_2 \) is found. One issue is that it is not guaranteed that \( \bar{x} \) splits into two parts like seen above, since the weight might distribute arbitrary over the vector. Therefore, the algorithm has to find a certain subset \( I \subseteq [n] \) with \( |I| = \frac{n}{2} \) such that the relative weight on that subset is \( w \), which makes the relative weight on \( [n] \setminus I \) also \( w \). Once such an \( I \) is found, we can simply apply the algorithm from above by creating a list of \( x_1 \in \mathbb{Z}^n_{I}[w] \) and afterwards enumerating all \( x_2 \in \mathbb{Z}^n_{[n] \setminus I}[w] \). These ideas lead to the following algorithm Mitm.

**Algorithm 7 Mitm**

1. **Input:** \( G, n \in \mathbb{N}, w \in \mathbb{W}, a \in \mathbb{G}^n, s \in \mathbb{G} \)
2. **Output:** An arbitrary \( x \in \mathbb{Z}^n[w] \) with \( f_a(x) = s \).
3. **for** \( n \cdot |\mathbb{Z}^n[w]|/|\mathbb{Z}^{n/2}[w]|^2 \) times **do**
4. Choose \( I \subseteq [n] \) with \( |I| = \frac{n}{2} \) and enumerate all elements from \( \mathbb{Z}^n_I[w] \) in a list \( \mathcal{L} \)
5. **Sort** \( \mathcal{L} \) by target values \( f_a(x_1) \) for each element in \( x_1 \in \mathcal{L} \).
6. **for** each \( x_2 \in \mathbb{Z}^n_{[n] \setminus I}[w] \) **do**
7. **if** there is an \( x_1 \in \mathcal{L} \) with \( f_a(x_1) = s - f_a(x_2) \) **then** \( \triangleright \text{binary search} \)
8. **return** \( x_1 + x_2 \)
9. **return** \( \perp \)
Theorem 27 (MITM). Let \( (a,s) \) be an instance of a \((G,n,w)\) Subset Sum Problem. Then MITM with input \((G,n,w,a,s)\) solves the instance with a probability of at least \(1 - 2^{-n}\) (over the coins of the algorithm) in time \(\widetilde{O}(2^{H(w)n}/2)\).

Proof. The algorithm does \(n \cdot |Z^n[w]|/|Z^{n/2}[w]|^2\) repetitions of creating lists of size \(|Z^{n/2}[w]|\). Due to the fact that \(|Z^n[w]| = \Theta(2^{H(w)n}), \quad |Z^{n/2}[w]|^2 = \Theta(2^{H(w)n})\) and \(|Z^{n/2}[w]| = \Theta(2^{H(w)n}/2)\), this results in a complexity of \(\widetilde{O}(2^{H(w)n}/2)\). Since sorting and binary search only give an overhead linear in \(n\) – which is suppressed due to the \(\widetilde{O}\) notation – this is also the overall time complexity of the algorithm.

Now assume there is an \(x \in Z^n[w]\) with \(f_a(x) = s\) and fix that \(x\). Notice that if there is no such solution, then the output is clearly \(\bot\). This can be seen easily, because if \(x_1 + x_2\) is output, then \(f_a(x_1 + x_2) = s\), due to the linearity of \(f\). Moreover, \(x_1 + x_2 \in Z^n[w]\), because \(Z^n[w] + Z_{n/2}[w] \subseteq Z^n[w]\) for any \(I\) chosen as in the algorithm.

The algorithm chooses \(n \cdot |Z^n[w]|/|Z^{n/2}[w]|^2\) uniformly random sets \(I \subseteq [n]\) with \(|I| = \frac{n}{2}\). We want to show that with a probability of at least \(1 - 2^{-n}\) there is at least one \(I\) such that there is a decomposition \(\bar{x} = \bar{x}_1 + \bar{x}_2\) with \(\bar{x}_1 \in Z^n[w]\) and \(\bar{x}_2 \in Z_{n/2}[w]\). It is easy to see that the probability of finding such an \(I\) that splits well is \(p := |Z^{n/2}[w]/|Z^n[w]|\). This is due to the fact that fixing a vector \(\bar{x}\) and choosing the partition \(I\) uniformly at random is the same as fixing the partition \(I\) and choosing the vector \(\bar{x}\) uniformly at random. Since \(|Z^n[w]| = |Z_{n/2}[w]| = |Z^{n/2}[w]|\), we get the probability \(p\), because \(|Z^n[w]/|Z_{n/2}[w]|\) of \(|Z^n[w]/|Z_{n/2}[w]|\) vectors are good. This means that the probability of not having a good \(I\) after \(n/p\) independent and uniformly random choices is \((1 - p)^{n/p} \leq 2^{-n}\), which proves the bound on the success probability.

Now fix such an \(I\) that splits the vector as desired. Then the algorithm stores all corresponding values of \(x_1 \in Z^n[w]\) in a list \(L\), which means that also \(x_1 \in L\). Then, \(x_2\) is part of the enumerated elements from \(Z_{n/2}[w]\). Since \(f_a(x_1) + f_a(x_2) = s\) due to the linearity of \(f\), it is for sure output in this situation. We have no control about the actual \(x\) that is output. Since we are only interested in an arbitrary \(x \in Z^n[w]\) with \(f_a(x) = s\), this \(x = x_1 + x_2\) also solves the problem, since its weight is clearly \(w\) and its target value is \(f_a(x_1) + f_a(x_2) = f_a(x_1 + x_2) = f_a(x) = s\). This means that the algorithm outputs a correct solution with a probability of at least \(1 - 2^{-n}\). 

### 3.2 Random Subset Sum Problem

The algorithm MITM has the issue that it doesn’t guarantee a fixed \(\bar{x} \in Z^n[w]\) with \(f_a(\bar{x}) = s\) to be output. Instead, an arbitrary \(x \in Z^n[w]\) is output, i.e. the first \(x\) that fulfills the equation. In some applications it might be sufficient to output an arbitrary \(x\), but in the following applications it isn’t. In these *representation algorithms*, we require a certain fixed golden solution \(\bar{x}\) to be constructed. The natural adaption to finding fixed instead of arbitrary solutions is to simply output all solutions. Unfortunately, in some cases this might lead to much worse than expected running times. One extreme example is the choice of \(n\) identical group elements in \(a\). This would lead to the case that simply all possible vectors are solutions and require the brute-force complexity to output all vectors.

In order to overcome this problem, we have to restrict ourselves to parameter choices, where \(a\) and \(s\) are chosen uniformly at random. Compared to the naive approach, it doesn’t seem to be possible to show the correctness of our algorithms for arbitrary weight distributions \(w\).
Therefore, we introduce the following notion of a valid weight distribution. In some applications like decoding of linear codes, this notion of valid isn’t a problem, since it is always fulfilled. In other scenarios like the Knapsack Problem, this requirement leads to some limitations in the choice of the weight distributions \( w \), but is fulfilled for the the choices discussed throughout this thesis. The definition is as follows.

**Definition 28 (valid weight distribution).** Let \( N \in \mathbb{N} \geq 2 \). The weight distribution \( w \) is called \( N \)-valid, if \( \sigma(w) \geq 2 \) and if there is an integer \( k \in \{1, \ldots, p\} \) such that we have \( w(z) = 0 \) for all \( z \notin \{1 - k, 2 - k, \ldots, p - k\} \), where \( p \) is the smallest prime divisor of \( N \).

This means that we restrict the weights \( w \) such that all the non-zero weights are in an integer interval of size \( p \) and all elements in \( \sigma(w) \) (except for the zero) are invertible modulo \( N \), which helps to show Lemma 30. This restriction is also tailored such that the so-called Counting Lemma – that can be used to show upper bounds on lists – can be applied. This is guaranteed by the fact that the difference of two different elements from \( \sigma(w) \) is invertible modulo \( N \). The Random Subset Sum Problem is defined as follows and has to output a list that contains any fixed solution. Notice that in this definition \( f_a(x) := \sum_{i=1}^{n} a_i \cdot x_i \), where \( n \) is the size of the vectors \( a \) and \( x \).

**Definition 29 (Random Subset Sum Problem).** Let \( G \) be a group, \( n \in \mathbb{N} \) and \( w \) a \(|G|\)-valid weight distribution. In the \((G, n, w)\) Random Subset Sum Problem we are given \( a \in G^n \) and \( s \in G \) chosen uniformly at random. The problem is to output a list that contains any fixed \( x \in \mathbb{Z}_n[w] \) with \( f_a(x) = s \), or an empty list if no \( x \) with \( f_a(x) = s \) exists.

Notice that \( a \) and \( s \) are not necessarily independent, whereas each individual component of \( a \) is. That is, the target \( s \) is usually chosen as \( f_a(x) = s \) for a value of \( x \) that is chosen independently of \( a \). In the following lemma, we want to show that having a \(|G|\)-valid \( w \), the value \( f_a(x) \) is uniform in \( G \).

**Lemma 30 (uniform target).** Let \( G \) be a group, \( n \in \mathbb{N} \) and \( w \) a \(|G|\)-valid weight distribution. Then if \( a \in G^n \) is chosen uniformly at random and for an arbitrary \( x \in \mathbb{Z}_n[w] \) that is chosen independently of \( a \), we have that \( f_a(x) \) is uniformly distributed in \( G \).

**Proof.** Since \( w \) is \(|G|\)-valid, we know that there is at least one component in \( x \) that is invertible modulo \(|G|\). Due to the fact that the corresponding component of \( a \) is uniform, also the whole sum \( f_a(x) \) is uniform in \( G \). \( \square \)

Notice that this lemma is also important for the proof of the correctness of the representation algorithms in Sect. 3.3.

### 3.2.1 Tools

Let us introduce some tools that are required in the analysis of our more advanced algorithms. A very important tool to measure the deviation from the expectation is the Counting Lemma, which was already seen in Nguyen et al. [NSS00] and Meurer [Meu13] and is proven for arbitrary groups in this thesis. With the help of this theorem, we are able to prove certain upper bounds on numbers of solutions and list sizes that appear in the algorithms. An important definition from [NSS00] to measure list sizes is the following number of elements of a set \( B \) that leads to a certain target value.
Definition 31 (preimage cardinality). Let $G$ be a group and $n \in \mathbb{N}$. For any $a \in G^n$, $s \in G$ and any set $B \subseteq \mathbb{Z}^n$ we define

$$N_a(B, s) = |\{ x \in B \mid f_a(x) = s \}|.$$

In our application, the set $B$ that appears in this definition is instantiated with sets of the form $\mathbb{Z}^n[w]$ for valid weight distributions $w$. The following definition of a valid set restricts the choice of $B$, which allows to prove the Counting Lemma.

Definition 32 (valid set). Let $n \in \mathbb{N}$ and $N \in \mathbb{N}_{\geq 2}$. A set $B \subseteq \mathbb{Z}^n$ is called $N$-valid, if for each $x \neq x' \in B$ there is at least one $j \in [n]$ with $\gcd(x_j - x'_j, N) = 1$.

We have already seen a definition of a valid weight distribution in Definition 28. The following lemma shows that any valid weight distribution implies a valid set.

Lemma 33 (valid weight $\implies$ valid set). Let $n \in \mathbb{N}$, $N \in \mathbb{N}_{\geq 2}$ and $w \in \mathcal{W}$ be $N$-valid. Then also the set $\mathbb{Z}^n[w]$ is $N$-valid.

Proof. Notice that we always have $|\mathbb{Z}^n[w]| \geq 2$ and let $x \neq x'$ be two arbitrary elements from $\mathbb{Z}^n[w]$. Then there is at least one $j \in [n]$ such that $x_j \neq x'_j$. Since $w$ is $N$-valid, we therefore have $0 < |x_j - x'_j| < p$, where $p \geq 2$ is the smallest prime divisor of $N$. This directly implies $\gcd(x_j - x'_j, N) = 1$, which makes $\mathbb{Z}^n[w]$ $N$-valid.

Now we have everything to state and prove the following Counting Lemma for arbitrary finite abelian groups $G$ with $|G| \geq 2$. It measures how much $N_a$ deviates from its expected value. Notice that the following lemma can be proven similarly as in [Men13] by induction. As a part of this proof, one has to show that $\sum_{a \in G^n} N_a(B, s) = |G|^{n-1} \cdot |B|$, using that $B$ is $|G|$-valid. In the following proof, however, we want to use an argument from [Her15], which also explains the statement of the lemma.

Lemma 34 (Counting Lemma). Let $G$ be a group and $n \in \mathbb{N}$. For any $|G|$-valid set $B \subseteq \mathbb{Z}^n$ the equation

$$\sum_{a \in G^n} \sum_{s \in G} \left( N_a(B, s) - \frac{|B|}{|G|} \right)^2 = |G|^n \cdot \frac{|G| - 1}{|G|} \cdot |B|$$

holds.

Proof. Notice that the equation obviously holds if $|B| = 0$. Now assume $|B| \geq 1$ and let $B$ be an arbitrary $|G|$-valid set. For each $a \in G^n$, $s \in G$ and $x \in B$ define random variables

$$Z_{a,s}(x) := \begin{cases} 1 & \text{if } f_a(x) = s \\ 0 & \text{otherwise} \end{cases}$$

and corresponding shifted random variables $Y_{a,s}(x) := Z_{a,s}(x) - \frac{1}{|G|}$. Now assume that $a$ and $s$ are chosen uniformly at random. Then for arbitrary $x \in B$ we have

$$\mathbb{E}_{a,s}[Z_{a,s}(x)] = \frac{1}{|G|},$$

which implies

$$\mathbb{E}_{a,s}[Y_{a,s}(x)] = 0.$$
Since \( Z_{a,s}(x) \) is Bernoulli distributed and \( Y_{a,s}(x) \) is a shifted Bernoulli distributed random variable, we have

\[
\mathbb{E}_{a,s}[(Y_{a,s}(x))^2] = \mathbb{V}_{a,s}[Y_{a,s}(x)] = \mathbb{V}_{a,s}[Z_{a,s}(x)] = \frac{1}{|G|} \cdot \left( 1 - \frac{1}{|G|} \right) = \frac{|G| - 1}{|G|^2}.
\]

Due to the fact that \( B \) is \(|G|-valid, we know that for each \( x, x' \in B \) with \( x \neq x' \) there is at least one \( j \in [n] \) such that \( \gcd(x_j - x'_j, |G|) = 1 \). This directly implies that the random variables \( Z_{a,s}(x) \) (and therefore also the random variables \( Y_{a,s}(x) \)) are pairwise independent. For uniform \( a \) and \( s \) and for the fixed \( B \), we want to compute the expected value

\[
\mathbb{E}_{a,s} \left[ \left( N_a(B, s) - \frac{|B|}{|G|} \right)^2 \right].
\]

Notice that the summation over all \( a \in G^n \) and \( s \in G \) is then the left hand side of the lemma statement. With the help of the random variable \( Z_{a,s}(x) \), we can replace \( N_a(B, s) \) by the sum over all individual elements \( x \in B \). This directly implies an unshifted sum over all random variables \( Y_{a,s}(x) \).

\[
= \mathbb{E}_{a,s} \left[ \left( \sum_{x \in B} Z_{a,s}(x) - \frac{|B|}{|G|} \right)^2 \right] = \mathbb{E}_{a,s} \left[ \left( \sum_{x \in B} \left( Z_{a,s}(x) - \frac{1}{|G|} \right) \right)^2 \right] = \mathbb{E}_{a,s} \left[ \sum_{x \in B} Y_{a,s}(x)^2 \right]
\]

The squared sum can be split into two parts, one covering all identical vectors in \( B \), the other covering all pairwise different ones.

\[
= \mathbb{E}_{a,s} \left[ \sum_{x \in B} (Y_{a,s}(x))^2 \right] + \mathbb{E}_{a,s} \left[ \sum_{x, x' \in B, x \neq x'} Y_{a,s}(x) \cdot Y_{a,s}(x') \right]
\]

In both cases we exchange summation and expected value. On the right hand part, we also make use of the pairwise independence of the random variable \( Y \), which allows to exchange the product and the expected value. Therefore, the second sum becomes zero, whereas the first sum is simply the sum over the variance of \( Y \).

\[
= \sum_{x \in B} \mathbb{E}_{a,s} [(Y_{a,s}(x))^2] + \sum_{x, x' \in B, x \neq x'} \mathbb{E}_{a,s}[Y_{a,s}(x)] \cdot \mathbb{E}_{a,s}[Y_{a,s}(x')] = \sum_{x \in B} \frac{|G| - 1}{|G|^2} = \frac{|G| - 1}{|G|^2} \cdot |B|
\]

Now, the summation over all \( a \in G^n \) and \( s \in G \) directly shows the statement of the lemma, due to the fact that the right hand side is multiplied by \(|G|^{n+1}\).

The following \textit{Upper Bound Lemma} shows an upper bound on the number of solutions and sizes of lists that appear in our algorithms. It can be shown that the number of \textit{bad} choices of \( a \) and \( s \) that violate this upper bound is always exponentially large in \( n \). This is the reason why we have to require a uniformly random choice of \( a \) and \( s \), in which case – although the number of bad instances is exponentially large – we have a good instance with overwhelming probability. It is possible to prove that if \( a \) and \( s \) are chosen uniformly at random, the list sizes don’t exceed a certain factor \( 2^{en} \) with a probability of at least \( 1 - 2^{-en} \), where \( e > 0 \) is some constant. The technique of cutting lists that exceed this bound guarantees on the one hand a fixed upper bound on the running time independent of the input and, on the other hand, an overwhelming success probability. The lemma can be found for special cases in [HJ10, BCJ11, Meu13].
Lemma 35 (Upper Bound Lemma). Let \( \mathcal{G} \) be a group and \( n \in \mathbb{N} \). For any \(|\mathcal{G}|\)-valid set \( \mathcal{B} \subseteq \mathbb{Z}^n \), for any constant \( \varepsilon > 0 \), for a fraction of at least \( 1 - 2^{-\varepsilon n} \) of choices \( \mathbf{a} \in \mathbb{G}^n \), there is at least a fraction of \( 1 - 2^{-\varepsilon n} \) values \( s \in \mathcal{G} \) with

\[
N_a(\mathcal{B}, s) \leq (1 + 2^{\varepsilon n}) \cdot (1 + |\mathcal{B}|/|\mathcal{G}|).
\]

Proof. In the following we want to show \( N_a(\mathcal{B}, s) \leq (1 + 2^{\varepsilon n}) \cdot \max\{1, |\mathcal{B}|/|\mathcal{G}|\} \), which implies the above statement. For any instance \( \mathbf{a} \) and any set \( \mathcal{B} \) we want to define

\[
\mathcal{S}_{a, \mathcal{B}} := \left\{ s \in \mathcal{G} \mid N_a(\mathcal{B}, s) \geq (1 + 2^{\varepsilon n}) \cdot \max\left\{1, \frac{|\mathcal{B}|}{|\mathcal{G}|}\right\} \right\},
\]

the set of all \( s \) for which there are at least \( (1 + 2^{\varepsilon n}) \cdot \max\{1, |\mathcal{B}|/|\mathcal{G}|\} \) values \( \mathbf{x} \in \mathcal{B} \) with \( f_a(\mathbf{x}) = s \). We want to call an instance \( \mathbf{a} \) bad, if \( |\mathcal{S}_{a, \mathcal{B}}| \geq 2^{-\varepsilon n} \cdot |\mathcal{G}| \). For any bad \( \mathbf{a} \) we therefore get

\[
\sum_{s \in \mathcal{G}} \left( N_a(\mathcal{B}, s) - \frac{|\mathcal{B}|}{|\mathcal{G}|} \right)^2 \geq \sum_{s \in \mathcal{S}_{a, \mathcal{B}}} \left( (1 + 2^{\varepsilon n}) \cdot \max\left\{1, \frac{|\mathcal{B}|}{|\mathcal{G}|}\right\} - \frac{|\mathcal{B}|}{|\mathcal{G}|} \right)^2 \geq \frac{|\mathcal{G}|}{2^{\varepsilon n}} \cdot (2^{\varepsilon n})^2 \cdot \max\left\{1, \frac{|\mathcal{B}|^2}{|\mathcal{G}|^2}\right\}.
\]

For the first inequality we make use of the fact that \( N_a(\mathcal{B}, s) > \frac{|\mathcal{B}|}{|\mathcal{G}|} \) for all \( s \in \mathcal{S}_{a, \mathcal{B}} \), which makes the expression inside the parenthesis always positive. The second inequality can be shown by considering the two cases \( |\mathcal{B}| \leq |\mathcal{G}| \) and \( |\mathcal{G}| \leq |\mathcal{B}| \) separately. Let \( k \) denote the number of bad instances. Applying the Counting Lemma, we obtain

\[
|\mathcal{G}|^n \cdot \frac{|\mathcal{G}| - 1}{|\mathcal{G}|} \cdot |\mathcal{B}| = \sum_{\mathbf{a} \in \mathcal{G}^n} \sum_{s \in \mathcal{G}} \left( N_a(\mathcal{B}, s) - \frac{|\mathcal{B}|}{|\mathcal{G}|} \right)^2 \geq k \cdot |\mathcal{G}| \cdot 2^{\varepsilon n} \cdot \max\left\{1, \frac{|\mathcal{B}|^2}{|\mathcal{G}|^2}\right\},
\]

which means that the fraction of bad instances is

\[
\frac{k}{|\mathcal{G}|^n} \leq \frac{|\mathcal{B}|}{|\mathcal{G}|} \cdot \max\left\{1, \frac{|\mathcal{B}|^2}{|\mathcal{G}|^2}\right\} \cdot 2^{-\varepsilon n} = \frac{|\mathcal{B}|}{|\mathcal{G}|} \cdot \min\left\{1, \frac{|\mathcal{G}|^2}{|\mathcal{B}|^2}\right\} \cdot 2^{-\varepsilon n} = \min\left\{\frac{|\mathcal{B}|}{|\mathcal{G}|}, \frac{|\mathcal{G}|}{|\mathcal{B}|}\right\} \cdot 2^{-\varepsilon n} \leq 2^{-\varepsilon n}.
\]

Therefore the fraction of good instances \( \mathbf{a} \) with \( |\mathcal{S}_{a, \mathcal{B}}| < 2^{-\varepsilon n} \cdot |\mathcal{G}| \) is at least \( 1 - 2^{-\varepsilon n} \). For these good instances we have at least a fraction of \( 1 - 2^{-\varepsilon n} \) values \( s \in \mathcal{G} \) with \( N_a(\mathcal{B}, s) \leq (1 + 2^{\varepsilon n}) \cdot \max\{1, \frac{|\mathcal{B}|}{|\mathcal{G}|}\} \).

Let us present a first application of this lemma for the computation of the number of solutions to the Random Subset Sum Problem. Obviously, the expected number is simply \(|\mathbb{Z}^n[\mathbf{w}]|/|\mathcal{G}|\), i.e. each target \( s \in \mathcal{G} \) should have approximately the same number of \( \mathbf{x} \in \mathbb{Z}^n[\mathbf{w}] \) such that \( f_a(\mathbf{x}) = s \). The lemma allows us to show that over the uniform choice of \( \mathbf{a} \) and \( s \) the expected bound on the number of solutions is met up to a factor of \( 2^{\varepsilon n} \) with a probability of \( 1 - \mathcal{O}(2^{-\varepsilon n}) \).

Lemma 36 (number of solutions). Let \((\mathbf{a}, s)\) be an instance of a \((\mathcal{G}, n, \mathbf{w})\) Random Subset Sum Problem. Then for any constant \( \varepsilon > 0 \) with probability of at least \( 1 - \mathcal{O}(2^{-\varepsilon n}) \) (over the random choice of the instance), the number of solutions \( \mathbf{x} \in \mathbb{Z}^n[\mathbf{w}] \) with \( f_a(\mathbf{x}) = s \) is \( \mathcal{O}(2^{\varepsilon n} + 2^{\varepsilon n} \cdot 2^{H(\mathbf{w}) n}/|\mathcal{G}|) \).

Proof. Since \( \mathbf{a} \) and \( s \) are chosen uniformly at random, \( \mathbf{w} \) is a \(|\mathcal{G}|\)-valid weight distribution and therefore \( \mathbb{Z}^n[\mathbf{w}] \) a \(|\mathcal{G}|\)-valid set, due the Upper Bound Lemma we have \( N_a(\mathbb{Z}^n[\mathbf{w}], s) \leq (1 + 2^{\varepsilon n}) \cdot (1 + |\mathbb{Z}^n[\mathbf{w}]|/|\mathcal{G}|) \) solutions with a probability of at least \( 1 - 2 \cdot 2^{-\varepsilon n} = 1 - \mathcal{O}(2^{-\varepsilon n}) \).

Since \(|\mathbb{Z}^n[\mathbf{w}]| = \mathcal{O}(2^{H(\mathbf{w}) n})\), this proves the lemma.
3.3 Known Results

In this section, we want to present some known results for the Random Subset Sum Problem for arbitrary finite abelian non-empty groups. The first part is an adaption of the meet-in-the-middle approach on the standard Subset Sum Problem to the new problem. In the second part, a newer technique based on so-called representations is shown, which is a generalization of the technique introduced by Howgrave-Graham and Joux [HJ10] as well as Becker, Coron and Joux [BCJ11] in the special case of the Knapsack Problem, as well as the application to Decoding by May, Meurer, Thomae [MMT11] and Becker, Joux, May, Meurer [BJMM12].

3.3.1 Meet-in-the-Middle Resivited

The presented bound on the number of solutions of a Random Subset Sum Problem helps us to upper bound the time complexity of the following algorithm MitmList. In this algorithm, we want to output a list that contains a fixed solution instead of just outputting an arbitrary solution in the previous algorithm Mitm. The main difference is a technique of cutting lists that exceed the computed upper bound, which allows to bound the running time.

Algorithm 8 MitmList

1: Input: $G$, $n \in \mathbb{N}$, $w \in \mathbb{W}$, $a \in G^n$, $s \in G$, $\varepsilon > 0$
2: Output: A list of $x \in \mathbb{Z}^n[w]$ with $f_a(x) = s$.
3: $L_{out} \leftarrow \emptyset$
4: for $\varepsilon n \cdot |\mathbb{Z}^n[w]|/|\mathbb{Z}^n[w]|^2$ times do
5: $I \subseteq [n]$ with $|I| = \frac{n}{2}$
6: Enumerate all elements from $\mathbb{Z}^n[I][w]$ and store them in a list $L$.
7: Sort $L$ by the target values $f_a(x_1)$ for each $x_1 \in L$.
8: for each $x_2 \in \mathbb{Z}^n[I]\setminus[w]$ do
9: for all $x_1 \in L$ with $f_a(x_1) = s - f_a(x_2)$ do
10: $L_{out} \leftarrow L_{out} \cup \{x_1 + x_2\}$ \hspace{1cm} \text{▷ binary search}
11: if $|L_{out}| > (1 + 2^{|n|}) \cdot \left(1 + |\mathbb{Z}^n[w]|/|G|\right)$ then
12: return $L_{out}$ \hspace{1cm} \text{▷ stop, if output list exceeds upper bound}
13: return $L_{out}$

Theorem 37 (MitmList). Let $(a, s)$ be an instance of a $(G, n, w)$ Random Subset Sum Problem. Then for any constant $\varepsilon > 0$, the algorithm MitmList with input $(G, n, w, a, s, \varepsilon)$ solves the instance with a probability of $1 - O(2^{-cn})$ (over both the choice of the input and the coins of the algorithm) in time $\tilde{O}(2^{H(w)n}/2 + 2^cn + 2^cn \cdot 2^{H(w)n}/|G|)$.

Proof. Notice that due to the upper bounds on the list sizes, the time complexity is independent of the chosen $a$ and $s$. Therefore we have a fixed upper bound on the time complexity in any case. Notice that it is known from the analysis of the algorithm Mitm that the number of repetitions in line 5 of the algorithm is polynomial in $n$. The creation of the lists and the sorting in lines 6, 7 and 8 have a time complexity of $\tilde{O}(|\mathbb{Z}^n[w]|) = \tilde{O}(2^{H(w)n}/2)$. Due to the fact that in the lines 9 to 13, the number of steps is upper bounded by $(1 + 2^cn) \cdot \left(1 + |\mathbb{Z}^n[w]|/|G|\right)$,
the time complexity of this part is \( \tilde{O}(2^{en} + 2^{en} \cdot 2^{H(n)/|G|}) \). Thus, the time complexity of the whole algorithm is as claimed.

If there is no \( x \in \mathbb{Z}^n[w] \) with \( f_a(x) = s \), then the output list is empty, because there is no \( I \) s.t. there is an \( x_1 \in \mathbb{Z}^n[w] \) and an \( x_2 \in \mathbb{Z}^n[w] \backslash I \) with \( f_a(x_1) + f_a(x_2) = s \) and therefore no element is added to the list. Otherwise, let \( x \in \mathbb{Z}^n[w] \) with \( f_a(x) = s \) be any fixed solution. After repeating \( \varepsilon n \cdot |Z^n[w]|/|Z^n/2[w]|^2 \) times, we can be sure with a probability of at least \( 1 - 2^{-en} \) that for at least one of the chosen \( I \), there are \( x_1 \in \mathbb{Z}^n[w] \) and \( x_2 \in \mathbb{Z}^n[w] \backslash I \) with \( x_1 + x_2 = x \), which can be proven as follows. It is easy to see that the probability of finding such \( I \) that splits well is \( p := |Z^n/2[w]|^2/|Z^n[w]| \). This is due to the fact that fixing a vector \( x \) and choosing the partition \( I \) uniformly at random is the same as fixing the partition \( I \) and choosing the vector \( x \) uniformly at random. Since \( |Z^n[w]| = |Z^n[w]/I[w]| = |Z^n/2[w]| \), we get the probability \( p \), because \( |Z^n/2[w]| \cdot |Z^n/w| \) out of \( |Z^n/w| \) vectors are good. This means that the probability of not having a good \( I \) after \( \varepsilon n/p \) independent and uniformly random choices is \( (1 - p)^{\varepsilon n/p} \leq 2^{-en} \), which proves the probability.

For this special \( I \), we know that \( \bar{x}_1 + \bar{x}_2 \) is added to \( L_{out} \), if the size of the list doesn’t exceed the upper bound of the Upper Bound Lemma. This happens with a probability of \( 1 - 2 \cdot 2^{-en} \) over the uniform choice of the instance, making the overall success probability at least \( 1 - 3 \cdot 2^{-en} = 1 - \tilde{O}(2^{-en}) \).

### 3.3.2 Classical Representations

In this section, we want to present the representation technique as it was introduced by Joux and Howgrave-Graham for the Knapsack Problem [HJ10] and later used to improve a subroutine for Decoding in May, Meurer and Thomae [MMT11]. It is shown in this section that both cases can be generalized to a general algorithm CLASSICALREP that solves the Random Subset Sum Problem in arbitrary groups \( G \).

However, we have some additional requirement on \( G \) compared to the definition of the Random Subset Sum Problem. The most important requirement is that the group order \( |G| \) is composite. Moreover, we want to concentrate on special group structures in the following. However, in the most interesting cases the group \( G \) either already fulfills all requirements or the group can be transformed efficiently into a group that fulfills the requirements.

One of these requirements is that the group is either of the form \( G = G_0 \times \ldots \times G_u \) for some \( u \in \mathbb{N} \) that has to be optimized or that there is an efficient isomorphism to a group of this form. In the following, we always assume w.l.o.g. that \( G \) is already of this form.

The algorithm additionally requires certain bounds on the sizes of the groups \( G_1, \ldots, G_u \). These sizes are derived from additional weight distributions \( w_1, \ldots, w_u \) that need to be optimized in order to minimize the time complexity of our algorithm. Notice that these weight distributions are part of the input of the algorithm, additionally to the weight distribution \( w \) of the actual Random Subset Sum Problem, which is denoted as \( w_0 \) in the following.

Let \( n \in \mathbb{N} \). The choice of the weight distributions defines so-called search spaces \( \mathbb{Z}^n[w_i] \) for all \( 0 \leq i \leq u \). For the initial weight distribution \( w_0 \), this is the set of possible solution vectors of the Random Subset Sum Problem. The idea of the technique is to solve the problem recursively by dividing the initial Random Subset Sum Problem with weight distribution \( w_i \) into two instances of an easier Random Subset Sum Problem with weight distribution \( w_{i+1} \) such that \( |\mathbb{Z}^n[w_{i+1}]| < |\mathbb{Z}^n[w_i]| \), i.e. the search spaces get smaller. In the last step \( u \), the problem is
once again solved in a meet-in-the-middle manner with square root complexity, which is now more efficient due to the decreasing size of the search spaces.

The recursive approach is realized in a similar manner as in the meet-in-the-middle approach. The idea is to represent a vector \( x \in \mathbb{Z}^n[\mathbf{w}] \) as a sum of two vectors \( x_1, x_2 \in \mathbb{Z}^n[\mathbf{w}_{t+1}] \). The most important notion in this new approach is that of a representation, firstly introduced in \cite{HJ10}.

**Definition 38** (g-
representation). Let \( n, g \in \mathbb{N} \), \( \mathbf{w} \) and \( \mathbf{w}' \) be weight distributions and \( \bar{x} \in \mathbb{Z}^n[\mathbf{w}'] \) be arbitrary. Then each \( x_1 \in \mathbb{Z}^n[\mathbf{w}] \) s.t. there is an \( x_2 \in \mathbb{Z}^n[\mathbf{w}] \) with \( x_1 + x_2 = \bar{x} \mod g \) is called a g-
representation of \( \bar{x} \) and

\[
\mathcal{R}^{n,g}_{\mathbf{w},\mathbf{w}'}(\bar{x}) := \{x_1 \in \mathbb{Z}^n[\mathbf{w}] \mid x_1 \text{ is a g-
representation of } \bar{x}\}
\]

is denoted the set of g-
representations of an \( \bar{x} \in \mathbb{Z}^n[\mathbf{w}'] \).

Notice that the vectors are used to define a linear combination of group elements of a group \( \mathbb{G} \). Thus, the components of the vectors are integers in \( \mathbb{Z}^{|\mathbb{G}|} \) without loss of generality. In the special case that the order of the group elements is not identical to the group order, i.e. it divides the group order, it makes additionally sense to restrict to integers in \( \mathbb{Z}_p \), where \( p \) is the order of the elements. Therefore, in most cases the only possible choice is \( g = |\mathbb{G}| \) and the value \( g \) can only be chosen as a divisor of \( |\mathbb{G}| \) if \( g \) matches the element order for each element in \( \mathbb{G} \) (or is a multiple of it). For example, we study the case of the group \( \mathbb{G} = \mathbb{F}_2^n \) in Chapter \ref{chp:meet-in-the-middle_approach} in which all elements have an order of two and therefore a choice \( g = 2 \) is possible.

The idea of this technique is that we only need one of these representations in order to find our \( \bar{x} \), a solution to the Random Subset Sum Problem. Concretely, it would be enough to have two lists \( L_1, L_2 \subseteq \mathbb{Z}^n[\mathbf{w}] \) such that only one of these representations \( \bar{x}_1 \) is in \( L_1 \) and the corresponding \( \bar{x}_2 = \bar{x} - \bar{x}_1 \mod g \) is in \( L_2 \). Ideally, the lists \( L_1, L_2 \) would contain only one element. On the other hand, since \( \bar{x} \) is unknown beforehand, this is too restrictive. Therefore, we have to settle for lists \( L_1, L_2 \) such that for all (or most) possible \( \bar{x} \in \mathbb{Z}^n[\mathbf{w}] \), there is a representation \( \bar{x}_1 \in L_1 \) and a corresponding \( \bar{x}_2 \in L_2 \). One important observation is that the number of representations is identical for each \( x \).

**Lemma 39** (identical number). Let \( n, g \in \mathbb{N} \) and \( \mathbf{w}, \mathbf{w}' \) be weight distributions. Then for each \( x \in \mathbb{Z}^n[\mathbf{w}'] \) the number of representations \( |\mathcal{R}^{n,g}_{\mathbf{w},\mathbf{w}'}(x)| \) is identical.

**Proof.** Let \( x, x' \in \mathbb{Z}^n[\mathbf{w}'] \) and \( g \) be arbitrary. Then due to the fixed weight, there is a permutation \( \pi : [n] \rightarrow [n] \) s.t. \( \pi(x) = x' \). Due to the component-wise addition, this also means that the set of representations of \( x' \) is \( \mathcal{R}^{n,g}_{\mathbf{w},\mathbf{w}'}(x') = \{\pi(x_1) \mid x_1 \in \mathcal{R}^{n,g}_{\mathbf{w},\mathbf{w}'}(x)\} \), which directly implies \( |\mathcal{R}^{n,g}_{\mathbf{w},\mathbf{w}'}(x)| = |\mathcal{R}^{n,g}_{\mathbf{w},\mathbf{w}'}(x')| \). \( \square \)

Notice that this observation makes use of the fact that the addition modulo \( g \) is performed component-wise. Although one might also consider cases with a more general definition, we want to restrict to component-wise addition in order to ease the notation. The restriction to component-wise addition makes the following definition of a number of representations (independent of the vector that is represented) useful.

**Definition 40** (number of representations). Let \( n, g \in \mathbb{N} \) and \( \mathbf{w}, \mathbf{w}' \) be weight distributions. Then for any \( x \in \mathbb{Z}^n[\mathbf{w}'] \) we denote \( r^{n,g}_{\mathbf{w},\mathbf{w}'} := |\mathcal{R}^{n,g}_{\mathbf{w},\mathbf{w}'}(x)| \).
With the help of the notion of joint distributions of Definition 9, the notion of corresponding weight of Definition 10 and the notion of output weight of Definition 18 as well as the notion of a multinomial coefficient of Chapter 2 we are able to explicitly compute this number of representations.

**Lemma 41** (number of representations). Let \( n, g \in \mathbb{N} \) and \( \mathbf{w, w}' \) be weight distributions. Define \( S \) to be the set of all joint distributions \( \Gamma \) such that \( \psi_1(\Gamma) = \psi_2(\Gamma) = \mathbf{w} \) and \( \psi^\text{out}_1(\Gamma) = \mathbf{w}' \) and such that all components \( \Gamma(i, j) \) multiplied by \( n \) are integers. Let \( T_k \) define the set of all \( (i, j) \in \sigma(\mathbf{w}) \times \sigma(\mathbf{w}) \) with \( i + j = k \mod g \) and let \( \tau : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z} \) be an arbitrary bijective map. Then we have

\[
\frac{r^w \cdot r^w'_{n, g}}{r^w_{n, g}} = \sum_{i \in S} \prod_{k \in \sigma(\mathbf{w})} \binom{w'(k) \cdot n}{w'_k(w'(k) \cdot n)}
\]

with \( w'_k \) being a weight distribution such that \( w'_k(\tau(i, j)) = \Gamma(i, j)/w'(k) \) for all \( (i, j) \in T_k \).

**Proof.** Each element in \( k \in \sigma(\mathbf{w}') \) is constructed by a sum modulo \( g \) of two elements from \( \sigma(\mathbf{w}) \). Notice that \( T_k \) is the set of all pairs in \( (\sigma(\mathbf{w}) \times \sigma(\mathbf{w})) \) that lead to this value \( k \). Therefore, the joint distributions \( \Gamma \) with \( \psi_1(\Gamma) = \psi_2(\Gamma) = \mathbf{w} \) and \( \psi^\text{out}_1 = \mathbf{w}' \) describe the only constellations that allow the sum of two weight \( \mathbf{w} \) vectors modulo \( g \) be of weight \( \mathbf{w}' \). If there is no such \( \Gamma \), clearly the number of representations is zero. Otherwise, we can simply sum up the individual numbers of representations that arise from certain joint distributions \( \Gamma \), since clearly none of the combinations is double counted. For each fixed \( \Gamma \), we compute the individual number of representations as follows. We are able to represent each element \( k \in \sigma(\mathbf{w}') \) in each of the possible combinations in the set \( T_k \). Clearly, the sum of the \( \Gamma(i, j) \) values for each of the \( (i, j) \in T_k \) is \( \mathbf{w}'(k) \), making \( \mathbf{w}'^\Gamma_k \) a weight distribution. For each \( k \) the number of possibilities is therefore the described binomial coefficient. The number of representations is the product over all \( k \).

This means that having a list \( L_1 \) with \( |L_1| \approx |\mathbb{Z}^n[\mathbf{w}]|/r^w_{n, g}' \) would leave an expected number of one vector \( \mathbf{x}_1 \in \mathbb{Z}^n[\mathbf{w}] \) representing an \( \mathbf{x} \in \mathbb{Z}^n[\mathbf{w}'] \). Since we additionally need that the corresponding \( \mathbf{x}_2 \) with \( \mathbf{x}_1 + \mathbf{x}_2 = \mathbf{x} \mod g \) is in the list \( L_2 \) (and not only an arbitrary representation) choosing \( L_1 \) and \( L_2 \) simply uniformly at random would not be enough. Instead, we need to keep a link between these two lists that guarantees the corresponding element to be in \( L_2 \), if there is a representation in \( L_1 \).

At this point we make use of the fact that the group \( G \) is of the form \( G_0 \times \ldots \times G_u \) for some \( u \in \mathbb{N} \) and assume \( u = 1 \) for the moment. The task in the Random Subset Sum Problem is to find an \( \mathbf{x} \in \mathbb{Z}^n[\mathbf{w}_0] \) such that \( f_\mathbf{a}(\mathbf{x}) = s = (s_0, s_1) \in G_0 \times G_1 \). This link between the two lists is now established by choosing \( L_1 \) and \( L_2 \) to be all elements of a certain subgroup of \( G \). Concretely, a \( s'_1 \in G_1 \) is chosen uniformly at random and \( L_1 \) is constructed to be all elements from the subgroup \( G_0 \times \{s'_1\} \). Since we know that \( \mathbf{x} \) has a target value of \( s_1 \), we construct \( L_2 \) such that it contains all elements from the subgroup \( G_0 \times \{s'_1 - s_1\} \). This makes sure that if there is an \( \mathbf{x}_1 \in L_1 \) that is a representation of \( \mathbf{x} \), the corresponding \( \mathbf{x}_2 \) with \( \mathbf{x}_1 + \mathbf{x}_2 = \mathbf{x} \mod g \) is in the list \( L_2 \).

The expected sizes of these lists become \( |\mathbb{Z}^n[\mathbf{w}_1]|/|G_1| \). Since we are interested in lists that contain approximately one representation, the size of the group \( G_1 \) should be approximately the number of representations \( r^w_{n, g} \). This would indeed lead to lists of an expected size of
For any bad $x$ which there is no probability. In the middle section, we are able to show that this list size is obtained with overwhelming probability. Therefore have at least a fraction of 1 with the help of the Complexity of course, the whole approach would only make sense, if this running time outperforms the time complexity of $O(|Z^n[w_1]|/r_{m_0,m_1})$.

As already mentioned, we have to guarantee that there is at least one representation in the list $L_1$, which would automatically lead to the corresponding counterpart being in $L_2$. That’s why we have to study the probability that there is a representation in the subgroup $G_0 \times \{s_i\}$ for a uniformly chosen $s'_i \in G_1$. This leads to the following lemma, which can also be proven with the help of the Counting Lemma.

**Lemma 42** (Golden Solution Lemma). Let $G$ be a group and $n \in \mathbb{N}$. For any constant $\varepsilon > 0$ and any $|G|$-valid set $B \subseteq \mathbb{Z}^n$ with $|G|/|B| \leq (2^{-\varepsilon n})^2$, for a fraction of at least $1 - 2^{-\varepsilon n}$ of choices $a \in G^n$, there is a fraction of at least $1 - 2^{-\varepsilon n}$ values $s \in G$ such that there is at least one $x \in B$ with $f_a(x) = s$.

Proof. For any instance $a$ and any set $B$ let $S_{a,B} := \{s \in G \mid N_a(B, s) = 0\}$, the set of all $s$ for which there is no $x \in B$ with $f_a(x) = s$. We want to call an instance $a$ bad, if $|S_{a,B}| \geq 2^{-\varepsilon n} \cdot |G|$. For any bad $a$ we therefore get

$$
\sum_{s \in G} \left( N_a(B, s) - \frac{|B|}{|G|} \right)^2 \geq \sum_{s \in S_{a,B}} \left( N_a(B, s) - \frac{|B|}{|G|} \right)^2 = \sum_{s \in S_{a,B}} \left( 0 - \frac{|B|}{|G|} \right)^2 = \frac{|G|}{2^\varepsilon n} \cdot |B|^2 \cdot \frac{|B|^2}{2^\varepsilon n |G|}.
$$

Let $k$ denote the number of bad instances. Applying the Counting Lemma, we obtain

$$
|G|^n \cdot \frac{|G| - 1}{|G|} \cdot |B| = \sum_{a \in G^n} \sum_{s \in G} \left( N_a(B, s) - \frac{|B|}{|G|} \right)^2 \geq k \cdot \frac{|B|^2}{2^\varepsilon n \cdot |G|},
$$

which means that the fraction of bad instances is $k/|G|^n \leq 2^{-\varepsilon n} \cdot |G|^n / |B|^2 \leq 2^{-\varepsilon n}$. Therefore the fraction of good instances $a$ with $|S_{a,B}| < 2^{-\varepsilon n} \cdot |G|$ is at least $1 - 2^{-\varepsilon n}$. For these good instances we therefore have at least a fraction of $1 - 2^{-\varepsilon n}$ values $s \in G$ such such there is at least one $x \in B$ with $f_a(x) = s$.

Since $a$ is chosen uniformly at random and we decide to choose $s'_i$ uniformly at random from $G_1$, applying this lemma to the group $G_1$ guarantees a representation in $L_1$ with overwhelming probability. Finally, $L_1$ and $L_2$ can be merged with a standard meet-in-the-middle technique that finds a match in the whole group and keeps the running time in $\tilde{O}(|Z^n[w_1]|/r_{m_0,m_1})$. Of course, the whole approach would only make sense, if this running time outperforms the time complexity of $\tilde{O}(\sqrt{|Z^n[w_0]|})$ of the meet-in-the-middle approach. Indeed, we present concrete instantiations that lead to better algorithms. The computation tree of the algorithm is illustrated in Fig. 3.2.

![Figure 3.2: Computation tree for $u = 1$](image-url)
One more thing we have to consider is the construction of the lists $L_1$ and $L_2$, since this should ideally be done in time roughly the list size. Unfortunately, there doesn’t seem a way to do so in general, i.e. there are initial weight distribution $\mathbf{w}_0$ and corresponding (optimal) $\mathbf{w}_1$ such that the construction time of the lists $L_1, L_2$ is larger than the list sizes itself. The reason is that the only generic way for constructing the lists seems to be a meet-in-the-middle approach. Concretely, the vectors are split into two halves, which makes the algorithm run in time $\tilde{O}(\sqrt{|Z^n[\mathbf{w}_1]|})$, the square root of the newly considered search space. Notice that $\mathbf{w}_1$ is usually chosen such that the corresponding search space is smaller than the initial search space, which results in an overall running time that is still below the standard meet-in-the-middle complexity of $\tilde{O}(\sqrt{|Z^n[\mathbf{w}_0]|})$.

The fact that this construction time might indeed be larger than the actual size of the constructed list is the reason for considering choices $u > 1$, which allows to apply the representation technique recursively. The extension to $u = 2$, which is illustrated in Fig. 3.3, has a similar idea to construct $L_1$ and $L_2$ as all elements from the subgroup $G_0 \times \{s'_1\} \times \{s'_2\}$ resp. $G_0 \times \{s_1 - s'_1\} \times \{s_2 - s'_2\}$ for uniformly chosen $s'_1 \in G_1$ and $s'_2 \in G_2$. This leads to lists $L_1$ and $L_2$ of size $\tilde{O}(|Z^n[\mathbf{w}_1]|/|G_1 \times G_2|)$ and we once again choose $|G_1 \times G_2| \approx r_{n,0,\mathbf{w}_1}^{n,1}$ to obtain the same list size as before. In difference to the case $u = 1$, this construction is done in two steps, instead of a direct application of the meet-in-the-middle approach. Let us describe only the construction for $L_1$, since the construction of $L_2$ is done analogously.

Before building $L_1$, we start by building two lists $L_{11}$ and $L_{12}$. The list $L_{11}$ is constructed as a subset of $Z^n[\mathbf{w}_2]$ with all elements from the subgroup $G_0 \times G_1 \times \{s'_1\}$ for $s'_2 \in G_2$ chosen uniformly at random. Analogously, $L_{12} \subseteq Z^n[\mathbf{w}_2]$ contains all elements from the subgroup $G_0 \times G_1 \times \{s'_1 - s''_1\}$. The construction is therefore in a direct recursive manner and the choice $|G_2| \approx r_{n,1,\mathbf{w}_2}^{n,1}$ allows to construct these lists in time $\tilde{O}(|Z^n[\mathbf{w}_2]|/r_{n,1,\mathbf{w}_1}^{n,1})$. Once again, it can be shown that this time complexity is smaller than the time complexity of $\tilde{O}(\sqrt{|Z^n[\mathbf{w}_1]|})$ from the $u = 1$ case. The sub-lists $L_{11}$ and $L_{12}$ can finally be once again constructed with a meet-in-the-middle approach in time $\tilde{O}(\sqrt{|Z^n[\mathbf{w}_2]|})$. In the case that this meet-in-the-middle construction time once again dominates the size of the lists, one might consider to choose $u > 2$, which adds more levels to the algorithm. The recursive application of the algorithm creates a binary computation tree of depth $u$. However, in the most interesting special cases $u$ is a small constant of not larger than 3.

In the description of the construction of the lists in the case $u = 2$, the construction of $L_1$ given $L_{11}$ and $L_{12}$ isn’t described yet. The issue is that compared to the meet-in-the-middle approach in the last step, not all sums of an element in $L_{11}$ and an element in $L_{12}$ have to lead to an element in $Z^n[\mathbf{w}_1]$, because the weight is distributed over the whole vector instead of being split into two halves. This leads to an increasing running time due to possible inconsistent solutions, which was actually missed in the original work of [HJJ10].
One possible way to deal with these inconsistencies is to simply brute-force all pairs and keep only those that are consistent, which is done in the classical representation algorithm. In order to be able to bound the time complexity for filtering these inconsistencies in this naive way, we can simply bound the number of elements in $L_{11}$ that lead to the chosen target value with the help of the Upper Bound Lemma. This allows to bound the time complexity as $\tilde{O}(|\mathbb{Z}^n|w_2|^2/(r_{n,w_1}^{n,q} \cdot r_{n,w_2}^{n,g}))$. More efficient techniques for dealing with inconsistencies are discussed in Sect. 3.4 in which we use the algorithm for the Consistency Problem analyzed in Chapter 2. The main idea is to solve a Weight Match Problem, i.e. to find a better algorithm for the problem given two lists vectors in $\mathbb{Z}^n[w_2]$, to find the vectors that have a sum modulo $g$ that is in the set $\mathbb{Z}^n[w_1]$.

Notice that this problem also arises in the last step of the computation, in which the final two lists are merged to find the solution. Solving this step naively has $\tilde{O}(|\mathbb{Z}^n[w_1]|^2/(r_{n,w_0,w_1}^{n,q} \cdot |G|))$ complexity, which doesn’t fall into account if $|G|$ is large enough. However, if $|G|$ is small, one might also consider algorithms that find consistent vectors more efficiently by searching for vectors in $\mathbb{Z}^n[w_1]$ that sum to a vector in $\mathbb{Z}^n[w_0]$. The following algorithm is from [HJ10].

**Algorithm 9 ClassicalRep**

1. **Global:** $u \in \mathbb{N}_0$, $G = G_0 \times \ldots \times G_u$, $w_0, \ldots, w_u, n, g \in \mathbb{N}$, $a \in \mathbb{G}^n$, $\varepsilon > 0$
2. **Input:** $\ell \in \{0, \ldots, u\}$, $\mathbb{G} = \mathbb{G}_\ell \times \ldots \times \mathbb{G}_u$
3. **Output:** A list of $x \in \mathbb{Z}^n[w_\ell]$ with $f_{a[G_\ell \times \ldots \times G_u]}(x) = s$.

4: if $\ell = u$ then
5: return $\text{MITMLIST}(G_u, w_\ell, n, a[G_u], s, \epsilon)$ \>$\triangleright$ $a[G_u]$ is the projection to the group $G_u$
6: else
7: $s' \leftarrow G_{\ell+1} \times \ldots \times G_u$
8: $L_1 \leftarrow \text{CLASSICALREP}(\ell + 1, s')$
9: $L_2 \leftarrow \text{CLASSICALREP}(\ell + 1, s[G_\ell+1 \times \ldots \times G_u] - s')$
10: Sort $L_1$ by the target values $f_{a[G_\ell \times \ldots \times G_u]}(x_1)$ for each $x_1 \in L_1$.
11: $L_{\text{out}} \leftarrow \emptyset$
12: for each $x_2 \in L_2$ do
13: for all $x_1 \in L_1$ with $f_{a[G_\ell \times \ldots \times G_u]}(x_1) = s - f_{a[G_\ell \times \ldots \times G_u]}(x_2)$ do \>$\triangleright$ binary search
14: if $(x_1 + x_2 \mod g) \in \mathbb{Z}^n[w_\ell]$ then
15: $L_{\text{out}} \leftarrow L_{\text{out}} \cup \{x_1 + x_2 \mod g\}$
16: if $|L_{\text{out}}| > (1 + 2^n) \cdot (1 + |\mathbb{Z}^n[w_\ell]| / |G_\ell \times \ldots \times G_u|)$ then
17: return $L_{\text{out}}$ \>$\triangleright$ stop, if output list exceeds upper bound
18: return $L_{\text{out}}$

In the recursively defined algorithm ClassicalRep, we want to differentiate between global and local input values. The global parameters are valid for all recursive calls of the algorithms, whereas the local input values $\ell$ and $s$ change in each call and are initiated with $\ell = 0$ and the input target value $s$. The global parameter $u \in \mathbb{N}_0$ describes the number of levels of representations. As a special case $u = 0$, we get the classical meet-in-the-middle algorithm. The parameters $w_0, \ldots, w_u$ have to be valid weight distributions, such that $w_0$ is defined by the Random Subset Sum Problem, whereas $w_1, \ldots, w_u$ have to be chosen in order to optimize the performance of the algorithm. A further restriction on the choice of the $w_1, \ldots, w_u$ is that
there is always at least one representation on each level, i.e. $r_{v,t}^{n,g} \geq 1$ for all $0 \leq \ell \leq u - 1$. The parameter $g$ has to be chosen as a divisor of $|G|$ such that each group element in $G$ has an order that divides $g$.

The group $G$ should be decomposable into $u + 1$ parts $G_0, \ldots, G_u$. As already mentioned before, it would be enough if there is an efficiently computable isomorphism between the groups $G$ and $G_0 \times \ldots \times G_u$, but for the ease of description we assume that $G = G_0 \times \ldots \times G_u$. Additionally, we need certain restrictions on the group sizes dependent on the chosen weight distributions that are described in Theorem 43. The restriction on the size of the lists guarantees at least one representation in the lists with overwhelming probability, which can be shown with the help of the Golden Solution Lemma. On the first sight, the existence of such an isomorphism with certain sizes seems to be a hard restriction, but we can show that for interesting problems this decomposition is always fulfilled.

On each level $0 \leq \ell \leq u$ (starting with $\ell = 0$) we want to have the promise that there is a linear combination with weights in $Z^n[\mu_\ell]$ in the group $G_\ell \times \ldots \times G_u$ that sums up to an input value $s$. Notice that for $\ell = 0$, this defines the Random Subset Sum Problem itself and for $\ell > 0$ the problem gets reduced to sub-problems in which this is true with overwhelming probability. At the end, the algorithm has to output a list of vectors $x \in Z^n[\mu_\ell]$ with $f_a[G_\ell \times \ldots \times G_u](x) = s$, which has to contain any fixed solution $x \in Z^n[\mu_\ell]$ with overwhelming probability.

The recursive algorithm can be described as follows. On each level $\ell \neq u$ the problem which is stated in a group $G_\ell \times \ldots \times G_u$ is solved recursively in the group $G_{\ell + 1} \times \ldots \times G_u$ by choosing a uniformly random target value $s'$. The splitting into two lists $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathcal{L}_1$ is a list of elements with target value $s'$ and $\mathcal{L}_2$ a list with target value $s[G_{\ell + 1} \times \ldots \times G_u] - s'$ guarantees that for each sum of an element in $\mathcal{L}_1$ and an element in $\mathcal{L}_2$, we have that the target value $s$ is already fixed in the group $G_{\ell + 1} \times \ldots \times G_u$. We can guarantee with the help of the Golden Solution Lemma that the choice of only one of such $s'$ makes sure that the promised decomposition (the unknown $x$) of $s$ has at least one representation with this target value $s'$. If this is the case, the promise on level $\ell$ implies a promise on both sub-problems on level $\ell + 1$. In this manner, the algorithm creates a binary recursion tree of depth $u$. On the last level, the problem is finally solved in a meet-in-the-middle manner.

In the conquer step of the algorithm, the two lists $\mathcal{L}_1, \mathcal{L}_2$ that are either constructed recursively or in a meet-in-the-middle manner are combined to a list $\mathcal{L}_{\text{out}}$ that contains all sums of an element in $\mathcal{L}_1$ and an element in $\mathcal{L}_2$ with a target value of $s$ in the group $G_\ell \times \ldots \times G_u$. This is done by firstly sorting the list $\mathcal{L}_1$ by their corresponding target values. The sorted list can then easily be shortened such that there is a certain upper bound on the number of elements of each target value. This can for example be done by removing arbitrary elements in groups that violate that upper bound, until this upper bound holds for all target values. An application of the Upper Bound Lemma guarantees that the upper bound holds for the group of elements with the uniformly random target value of the golden solution.

In the loop, we simply enumerate all elements from $\mathcal{L}_2$ and search for all corresponding elements from $\mathcal{L}_1$ such that their sum has a target value of $s$. Notice that due to the shortened $\mathcal{L}_1$, this number is always bounded. Then, we check if the output list is not too large yet, with an upper bound that once again holds with overwhelming probability. Finally, we apply a consistency check, since we are only interested in elements from $Z^n[\mu_\ell]$ and most of the elements that are built as a sum of two vectors from $Z^n[\mu_{\ell + 1}]$ are not in $Z^n[\mu_\ell]$. All these outputs are merged to an output list $\mathcal{L}_{\text{out}}$. The following theorem proves correctness and running time of the algorithm and is a generalization of similar theorems from [HJ10] [BCJ11] [MMT11] [Meu13].
Lemma 43 (CLASSICALREP). Let \((a, s)\) be an instance of a \((G, n, w_0)\) Random Subset Sum Problem. Let \(u \in \mathbb{N}_0\) be a constant independent of \(n\) such that \(G = G_0 \times \ldots \times G_u\) and let \(g \in \mathbb{N}\) be a divisor of \(|G|\) such that all elements in \(G\) have an order that divides \(g\). Let \(w_0, \ldots, w_u\) be valid weight distributions with \(r^{n,g}_{w_0,w_0+1} \geq 1\) for all \(0 < \ell < u - 1\). Then for any constant \(\varepsilon > 0\), if \(\frac{1}{2} \leq |G_\ell| \cdot \ldots \cdot |G_u|/r^{n,g}_{w_0,w_0+1} \cdot (2^{2\varepsilon n})^2 \leq 1\) for all \(0 \leq \ell \leq u\), the algorithm CLASSICALREP with global input \((u, G, w_0, \ldots, w_u, n, g, a, s)\) and local input \((0, s)\) solves the instance with a probability (over both the coins of the algorithm and the choice of \(a\) and \(s\)) of \(1 - \mathcal{O}(2^{-2\varepsilon n})\) in time

\[
\tilde{O}
\left(2^{2\varepsilon n} + 2H(w_u)n/2 + 2^{2\varepsilon n} \cdot \frac{2^{H(w_u)n}}{r^{n,g}_{w_{u-1},w_u}} + 2^{6\varepsilon n} \cdot \sum_{\ell=0}^{u-1} \frac{2^{2H(w_{\ell+1})n}}{r^{n,g}_{w_{\ell-1},w_\ell} \cdot r^{n,g}_{w_\ell,w_{\ell+1}}} \right),
\]

with \(r^{n,g}_{w_0,w_0} := |G|\).

Proof. Due to the valid choice of the weight distributions \(w_0, \ldots, w_u\), it is possible to apply both the Upper Bound Lemma and the Golden Solution Lemma. Moreover, the choice of the sizes of the subgroups of \(G\) guarantees that \(\frac{1}{2} \leq |G_{\ell+1}| \cdot \ldots \cdot |G_u|/r^{n,g}_{w_0,w_0+1} \cdot (2^{2\varepsilon n})^2 \leq 1\) for all \(0 \leq \ell \leq u - 1\). In the following, we also want to assume that there is an \(\bar{x} \in \mathbb{Z}^n[w_0]\) with \(f_a(\bar{x}) = s\) (in the whole group \(G\)). Otherwise the output list is obviously empty. Notice that due to the fact that the order of each individual group element is a divisor of \(g\), it also holds that \(f_a(\bar{x} + y) = s\) for any \(y \in (g\mathbb{Z})^n\). This allows to consider each intermediate vector mod \(g\).

The algorithm creates a binary computation tree of depth \(u\). Notice that in each node of the tree, the input parameter \(s\) is a uniformly random element from the respective group \(G_\ell \times \ldots \times G_u\). This is the case, since the initial \(s\) is uniform, \(s'\) is chosen uniformly at random by the algorithm, which makes also \(s[G_{\ell+1} \times \ldots \times G_u] - s'\) uniform. In each call for \(\ell \neq u\) (a total of \(2^u - 1\) nodes), we have to guarantee a representation in \(L_1\) with target value \(s'\). For each of these nodes, the Golden Solution Lemma guarantees this with a probability of at least \(1 - 2 \cdot 2^{-2\varepsilon n}\). This can be shown by choosing the group as \(G_{\ell+1} \times \ldots \times G_u\) and the set \(B = R^{n,g}_{w_0,w_{\ell+1}}(\bar{x})\), with \(\bar{x}\) being the golden solution of the previous step for each node with \(0 \leq \ell \leq u - 1\), fulfilling the restriction of the lemma and guaranteeing at least one \(\bar{x}_1 \in R^{n,g}_{w_0,w_{\ell+1}}(\bar{x})\) with \(f_a[G_{\ell+1} \times \ldots \times G_u](\bar{x}_1) = s'\).

Let us now assume we have golden solutions in all nodes, i.e. a promise that there is at least one \(\bar{x} \in \mathbb{Z}^n[w_\ell]\) with \(f_a[G_{\ell+1} \times \ldots \times G_u](\bar{x}) = s\). Then we first of all have to make sure that MITMLIST works correctly on all \(2^u\) leaf nodes. As can be seen from Theorem 37 with a probability of \(1 - \mathcal{O}(2^{-2\varepsilon n})\) for each node, the fixed solution \(\bar{x} \in \mathbb{Z}^n[w_u]\) with \(f_a[G_u](\bar{x}) = s\) in \(G_u\) is found in time \(\tilde{O}(2^{2\varepsilon n}n/2 + 2^{2\varepsilon n} + 2^{2\varepsilon n} \cdot 2^{H(w_u)n}/|G_u|)\).

For \(\ell \neq u\) we have to guarantee two upper bounds for each node. The first bound is one on the output list \(L_{\text{out}}\). With the help of the Upper Bound Lemma instantiated with the group \(G_\ell \times \ldots \times G_u\) and the choice \(B = \mathbb{Z}^n[w_\ell]\), due to the uniform choice of \(a\) and the input \(s\), the probability that \(|L_{\text{out}}| \leq 1 + 2^{2\varepsilon n}\) \cdot \left(1 + |\mathbb{Z}^n[w_{\ell+1}]/|G_\ell \times \ldots \times G_u|\right)\) is at least \(1 - 2 \cdot 2^{-2\varepsilon n}\) for each of the \(2^u - 1\) nodes.

It is left to show the second upper bound on the number of elements in \(L_1\) with a target value of \(f_a[G_{\ell+1} \times \ldots \times G_u](\bar{x}_1), \bar{x}_1\) being the representation guaranteed by the Golden Solution Lemma with the uniformly random target value \(f_a[G_{\ell+1} \times \ldots \times G_u](\bar{x}_1) = s'\). With the help of Lemma 31, we can use the uniformity of \(a[G_\ell]\) to show the uniformity of \(f_a[G_\ell](\bar{x}_1)\). This means that also \(f_a[G_{\ell+1} \times \ldots \times G_u](\bar{x}_1)\) is uniformly distributed, which makes the Upper Bound Lemma applicable. The lemma tells us that there are at most \((1 + 2^{2\varepsilon n}) \cdot (1 + |\mathbb{Z}^n[w_{\ell+1}]/|G_\ell \times \ldots \times G_u|)\) elements with this particular target value with a probability of at least \(1 - 2 \cdot 2^{-2\varepsilon n}\).
This means that with the union bound, the probability that the algorithm fails in any of these steps is $1 - \mathcal{O}(2^{-\varepsilon n})$ due to the fact that $u$ is constant. Thus, with this probability, the golden solution is constructed in the case $\ell = u$ by the correctness of MitmList as well as in the case $\ell < u$, which is shown as follows. The algorithm receives two lists $\mathcal{L}_1, \mathcal{L}_2$ with $x_1 \in \mathcal{L}_1$, $x_2 \in \mathcal{L}_2$, with $\bar{x}_1, \bar{x}_2$ being the golden solutions of the previous level. These vectors remain in the list after the shortening, due to the bounds shown above. Due to the fact that there is a golden solution $\bar{x} = \bar{x}_1 + \bar{x}_2 \mod q$ with $\bar{x} \in \mathbb{Z}^n[w]_\ell$ and $f_{\varepsilon \mathbb{Z}[\ell] \times \ldots \times \mathbb{Z}[u]}(\bar{x}) = s$ on the next level, it is added to the output list by enumerating all elements in $\mathcal{L}_2$ and binary searching for all elements with a matching target value in $\mathcal{L}_1$. Afterwards, the weight filter accepts the correct solution. The upper bound at the end isn’t exceeded, as shown above, such that the golden solution is part of the output in every node.

The time complexity is first of all that of the MitmList algorithm that is called $2^u$ times. Each of these calls have a complexity of

$$\tilde{\mathcal{O}}(2^H(w_u)n/2)$$

for creating and processing the two sub-lists and an additional complexity of

$$\tilde{\mathcal{O}}(2^{\varepsilon n} + 2^{\varepsilon n} \cdot 2^H(w_u)/|\mathbb{G}_u|)$$

for creating the output list, which means that the second complexity becomes

$$\tilde{\mathcal{O}}(2^{\varepsilon n} + (2^{\varepsilon n})^3 \cdot 2^H(w_u)/r_{n,g}^{m_u-1,m_u}),$$

due to the lower bound on $|\mathbb{G}_u|$. For $\ell \neq u$ we always have lists $\mathcal{L}_1, \mathcal{L}_2$ of size less than

$$(1 + 2^{\varepsilon n}) \cdot (1 + |\mathbb{Z}^n[w]_{\ell+1}|/|\mathbb{G}_{\ell+1} \times \ldots \times \mathbb{G}_u|)$$

elements. These are processed with negligible overhead, after which for each element of $\mathcal{L}_2$ there are at most

$$(1 + 2^{\varepsilon n}) \cdot (1 + |\mathbb{Z}^n[w]_{\ell+1}|/|\mathbb{G}_{\ell} \times \ldots \times \mathbb{G}_u|)$$

possible candidates in $\mathcal{L}_1$, due to its shortening to a maximal number of candidates of a certain target value. Therefore, the time complexity is

$$\tilde{\mathcal{O}}\left((2^{\varepsilon n})^2 \cdot \left(1 + \frac{|\mathbb{Z}^n[w]_{\ell+1}|}{|\mathbb{G}_{\ell+1} \times \ldots \times \mathbb{G}_u|} + \frac{|\mathbb{Z}^n[w]_{\ell+1}|^2}{|\mathbb{G}_{\ell} \times \ldots \times \mathbb{G}_u| \cdot |\mathbb{G}_{\ell+1} \times \ldots \times \mathbb{G}_u|}\right)\right),$$

which can be upper bounded by

$$\tilde{\mathcal{O}}\left((2^{\varepsilon n})^2 + (2^{\varepsilon n})^4 \cdot \frac{|\mathbb{Z}^n[w]_{\ell+1}|}{r_{n,g}^{m_{\ell-1},m_{\ell}} / r_{n,g}^{m_{\ell},m_{\ell+1}}} + (2^{\varepsilon n})^6 \cdot \frac{|\mathbb{Z}^n[w]_{\ell+1}|^2}{r_{n,g}^{m_{\ell-1},m_{\ell}} / r_{n,g}^{m_{\ell},m_{\ell+1}}}\right)$$

for $\ell \geq 1$. Notice that this bound also holds for $\ell = 0$, if we denote $r_{n,g}^{m_0,0} := |\mathbb{G}|$. On each level $\ell$ there are $2^\ell$ such nodes such that the time complexity is multiplied by this amount. Since the size of the output list on each level is always at most this complexity, it can be left out of the overall time complexity that suppresses constant factors. \[\square\]

These known results are used to construct algorithms for the Binary Subset Sum Problem in Chapter 4, the Knapsack Problem in Chapter 5, as well as the Decoding Problem in Chapter 6. In the following section we want to introduce novel results that build on the presented classical representation algorithm and create a consistent representation algorithm.
3.4 Consistent Representations

In this section, we present an improvement over the previously best known algorithm CLASSICALREP for solving the Random Subset Sum Problem. Parts of this section are already presented in \[\text{MO}\].

An issue of the previous approach is that we get a certain overhead due to inconsistent solutions. That is, we have to check all possible pairs that agree on the target value, if they are also distributed correctly, i.e. their sum is in \(\mathbb{Z}^n[\mathbf{w}]\). In the main part of the algorithm CLASSICALREP, we sort one of the lists \(L_1\) that is the output of the recursive call. Then each element \(x_2 \in L_2\) is enumerated and for each element \(x_1 \in L_1\) that has a fitting target value in the group \(G_\ell \times \ldots \times G_u\), it is checked whether \(x_1 + x_2 \mod g\) has the desired weight \(w_\ell\).

Notice that since the target values are already fixed in \(G_{\ell+1} \times \ldots \times G_u\), there are only \(|G_\ell|\) target value candidates left in each of the lists. The two lists therefore consist of sub-lists \(L_1^1, \ldots, L_1^{|G_\ell|}\) and \(L_2^1, \ldots, L_2^{|G_\ell|}\) of elements with one of the \(|G_\ell|\) fixed target values. Due to the fact that the target value of the output is fixed as \(s\) with \(G_\ell\)-component \(s[G_\ell]\), for each of the sub-lists of the first list, there is exactly one sub-list in the second list that matches this \(s[G_\ell]\). Since we don’t know in which of these \(|G_\ell|\) pairs of lists our fixed solution \((\bar{x}_1, \bar{x}_2)\) is, we have to search for it in all of the pairs.

\[
\text{List sizes on each level: } |\mathbb{Z}^n[\mathbf{w}]|/r_{w_0, \mathbf{w}_1}^{n,g} \quad |\mathbb{Z}^n[\mathbf{w}]|/r_{w_1, \mathbf{w}_2}^{n,g}
\]

In the algorithm CLASSICALREP this search is performed naively. Let us describe the improvement on the two level case \(u = 2\), as illustrated in Fig. 3.4. For each element \(x_{12} \in L_{12}\) of weight \(\mathbf{w}_2\) in a certain sub-list \(L_{12}^j\) characterized by its target value and for each element \(x_{11} \in L_{11}\) with weight \(\mathbf{w}_2\) in the corresponding list \(L_{11}^i\), it is checked whether \((x_{11} + x_{12} \mod g)\) is in \(\mathbb{Z}^n[\mathbf{w}_1]\). All those vector pairs that pass this test are part of the output. The idea of the new approach CONSISTENTREP is to speed up this search. Concretely, we want to replace a naive search with an algorithm for the following problem. Given two sub-lists \(L_{11}^i, L_{12}^j\) (each of size roughly \(|\mathbb{Z}^n[\mathbf{w}_1]|/|G_{11} \times G_{12}|\) of vectors such that each vector in the first list agrees with each vector in the second list on the given target value \(s\). The problem is to find all the pairs that are additionally \emph{consistent}, i.e. their sum is in \(\mathbb{Z}^n[\mathbf{w}_1]\). The problem that appears is exactly a Weight Match Problem that is studied in Chapter 2 since we receive two lists of elements in \(\mathbb{Z}^n[\mathbf{w}_2]\) and have to find sums modulo \(g\) that are in \(\mathbb{Z}^n[\mathbf{w}_1]\). In the following, we want to adapt the Weight Match Problem to the group setting, in order to show upper bounds on intermediate lists. With the help of the algorithm WEIGHTSIEVE, the complexity exponent \(\rho\) drops from \(\rho = 2\) (naive algorithm) to some value \(\rho < 2\). We use Corollary 20 to show the time complexity and correctness of the algorithm CONSISTENTREP. In the following section, we adapt this corollary to the new setting.
### 3.4.1 Group Weight Match Problem

In this section, we want to combine the results on the Weight Match Problem of the previous chapter with the Upper Bound Lemma of the previous section. Very similarly to the Random Weight Match Problem, in which we have assumed the input lists to have elements that are chosen uniformly at random, we want to introduce a Group Weight Match Problem. In this new problem, we choose the input lists as all vectors \( \mathbf{x} \in \mathbb{Z}^n[\mathbf{w}] \) with \( f_a(\mathbf{x}) = s_1 \) resp. \( f_a(\mathbf{x}) = s_2 \) for some target values \( s_1, s_2 \in G \) that arise in our representation algorithm. Once again, we have to show that all intermediate lists achieve their expected value up to factors of \( 2^n \), which can be done with the help of the Upper Bound Lemma. Notice that in contrast to the general Weight Match Problem, the elements in the two input lists have an identical weight \( \mathbf{w} \).

**Definition 44** (Group Weight Match Problem). Let \( n \in \mathbb{N}, G \) be a group, \( \mathbf{w}, \mathbf{w}' \) be valid weight distributions and let \( g \in \mathbb{N} \) such that there is exactly one joint distribution \( \Gamma \) with \( \psi_1(\Gamma) = \psi_2(\Gamma) = \mathbf{w} \) and \( \psi_{out}(\Gamma) = \mathbf{w}' \). In the \((G, n, g, \mathbf{w}, \mathbf{w}')\) Group Weight Match Problem we are given two lists \( L_1 := \{ \mathbf{x} \in \mathbb{Z}^n[\mathbf{w}] \mid f_a(\mathbf{x}) = s_1 \} \) and \( L_2 := \{ \mathbf{x} \in \mathbb{Z}^n[\mathbf{w}] \mid f_a(\mathbf{x}) = s_2 \} \) with \( a \in \mathbb{G}^n \), \( s_1, s_2 \in G \) chosen uniformly at random. The problem is to output a list that contains any fixed pair \((\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2 \) with \((\mathbf{x}_1 + \mathbf{x}_2 \mod g) \in \mathbb{Z}^n[\mathbf{w}']\), or an empty list if no such pair exists.

Notice that the restriction to weights \( \mathbf{w}, \mathbf{w}' \) and the \( g \) that lead to exactly one joint distribution \( \Gamma \) is a restriction solely to simplify the presentation. As already discussed in the previous chapter, it can be easily removed with only a polynomial overhead in \( n \).

In the following, we also need a Lower Bound Lemma, which compared to the Upper Bound Lemma requires some bound on the relation of \(|B|\) and \(|G|\), which also has to be fulfilled in the application. This bound is very similar to the bound in the Golden Solution Lemma, which is between the group sizes and the number of representations.

**Lemma 45** (Lower Bound Lemma). Let \( G \) be a group and \( n \in \mathbb{N} \). Then for any constant \( \varepsilon > 0 \), any \(|G|\)-valid set \( B \subseteq \mathbb{Z}^n \) with \(|B|/|G| \geq 2^{en}\), for a fraction of at least \( 1 - 4 \cdot 2^{-\varepsilon n} \) of choices \( a \in \mathbb{G}^n \), there is at least a fraction of \( 1 - 2^{-\varepsilon n} \) values \( s \in G \) with \( N_a(B, s) \geq \frac{1}{2} \cdot \frac{|B|}{|G|} \).

**Proof.** For any instance \( a \) and any set \( B \) we want to define \( \mathcal{S}_{a,B} := \left\{ s \in G \mid N_a(B, s) \leq \frac{1}{2} \cdot \frac{|B|}{|G|} \right\} \), the set of all \( s \) for which there are at most \( \frac{1}{2} \cdot \frac{|B|}{|G|} \) values \( \mathbf{x} \in B \) with \( f_a(\mathbf{x}) = s \). We want to call an instance \( a \) bad, if \( |\mathcal{S}_{a,B}| \geq 2^{-\varepsilon n} \cdot |G| \). For any bad \( a \) we therefore get

\[
\sum_{s \in G} \left( N_a(B, s) - \frac{|B|}{|G|} \right)^2 \geq \sum_{s \in \mathcal{S}_{a,B}} \left( N_a(B, s) - \frac{|B|}{|G|} \right)^2 \geq \sum_{s \in \mathcal{S}_{a,B}} \left( \frac{1}{2} \cdot \frac{|B|}{|G|} - \frac{|B|}{|G|} \right)^2 \geq \frac{|G|}{2^{\varepsilon n}} \cdot \frac{1}{4} \cdot \frac{|B|^2}{|G|^2}.
\]

Notice that at (*) we make use of the fact that \( N_a(B, s) < \frac{|B|}{|G|} \), which makes the expression inside the parenthesis always negative. Let \( k \) denote the number of bad instances. Applying the Counting Lemma, we obtain

\[
|G|^n \cdot \frac{|G| - 1}{|G|} \cdot |B| = \sum_{a \in \mathbb{G}^n} \sum_{s \in G} \left( N_a(B, s) - \frac{|B|}{|G|} \right)^2 \geq k \cdot \frac{|B|^2}{4 \cdot |G|^2} \cdot 2^{-\varepsilon n},
\]

which means that the fraction of bad instances is \( \frac{k}{|G|^n} \leq \frac{4 \cdot |G|}{|B|} \cdot 2^{\varepsilon n} \leq 4 \cdot 2^{-\varepsilon n} \). Therefore the fraction of good instances \( a \) with \( |\mathcal{S}_{a,B}| < 2^{-\varepsilon n} \cdot |G| \) is at least \( 1 - 4 \cdot 2^{-\varepsilon n} \). For these good instances we therefore have at least a fraction of \( 1 - 2^{-\varepsilon n} \) values \( s \in G \) with \( N_a(B, s) \geq \frac{1}{2} \cdot \frac{|B|}{|G|} \). \( \square \)
With the help of this lemma, we are able to prove the following lemma.

**Lemma 46 (Group Weight Match Lemma).** Let $(\mathcal{L}_1, \mathcal{L}_2)$ be an instance of a $(G, n, g, w, w')$ Group Weight Match Problem with $\mathcal{L}_1, \mathcal{L}_2 \subseteq \mathbb{Z}^n[w]$. Let $\Gamma$ be the unique joint distribution with $\psi_1(\Gamma) = \psi_2(\Gamma) = w$ and $\psi_{\text{out}}^\text{\Gamma}(\Gamma) = w'$. Then for any constant $\varepsilon > 0$, if $|\mathbb{Z}^n[w]|/|\Gamma| \geq 2^{2\varepsilon n}$, for any weight distributions $h_1, h_2 \leq 2 \cdot w$ such that $z := z_1 = z_2$ and $|\mathcal{C}| \geq 1$, the algorithm WeightSieve with input $(n, g, w, w', \mathcal{L}_1, \mathcal{L}_2, \varepsilon)$ solves the given instance with a probability of $1 - \mathcal{O}(2^{-\varepsilon n})$ in time

\[
\tilde{O} \left( (2^n)^y + \max \{ 0, \lambda_1 - z_1 \} + \max \{ 0, \lambda_2 - z_2 \} + 2\varepsilon \right)
\]

with $y := \max \{ \lambda, z + \varepsilon, \min_{C \in \mathcal{C}} \left\{ 1 - \sum_{i,j, \Gamma(i,j) \neq 0} \mathcal{H}_{\Gamma(i,j)}(C(i,j)/2) \right\} \}$, $\lambda := \max \{ \lambda_1, \lambda_2 \}$ with $\lambda_1 := \frac{1}{n} \log_2(|\mathcal{L}_1|)$, $\lambda_2 := \frac{1}{n} \log_2(|\mathcal{L}_2|)$, reduction values $z_1 := \mathcal{H}(w) - \frac{1}{2} \mathcal{H}(h_1) - \frac{1}{2} \mathcal{H}(2 \cdot w - h_1)$, $z_2 := \mathcal{H}(w) - \frac{1}{2} \mathcal{H}(h_2) - \frac{1}{2} \mathcal{H}(2 \cdot w - h_2)$ and $\mathcal{C}$ being the set of all joint distributions $C \leq 2 \cdot \Gamma$ with $\psi_1(C) = h_1$ and $\psi_2(C) = h_2$.

**Proof.** The Group Weight Match Problem is a Weight Match Problem, except for the concentration property on the lists. Therefore, the only thing we have to show are the bounds on the lists that achieve this property. Afterwards, we can simply use Corollary 20, which uses the Consistency Theorem.

Fix an arbitrary constant $t \in \mathbb{N}$, arbitrary weights $h_1, h_2 \leq 2 \cdot w$, arbitrary pairwise disjoint sets $\mathcal{I}_1, \ldots, \mathcal{I}_t \subseteq [n]$ with $\bigcup_{\ell=1}^t \mathcal{I}_\ell = [n]$ and arbitrary $\mathcal{J}_1, \ldots, \mathcal{J}_t$ with $\mathcal{J}_\ell \subseteq \mathcal{I}_\ell$ and $|\mathcal{J}_\ell| = |\mathcal{I}_\ell|/2$ for all $1 \leq \ell \leq t$. Define a collection of sets

\[
S_b^{[\ell]} := \sum_{k=1}^\ell \mathbb{Z}^n_{\mathcal{I}_k \setminus \mathcal{J}_b}[h_b] + \sum_{k=1}^\ell \mathbb{Z}^n_{\mathcal{I}_k \setminus \mathcal{J}_b}[2 \cdot w - h_b] + \sum_{k=\ell+1}^t \mathbb{Z}^n_{\mathcal{I}_k \setminus \mathcal{J}_b}[w]
\]

with sizes

\[
|S_b^{[\ell]}| = \prod_{k=1}^\ell \left( \frac{|\mathcal{I}_k|/2}{h_b \cdot |\mathcal{I}_k|/2} \right) \cdot \prod_{k=1}^\ell \left( \frac{|\mathcal{I}_k|/2}{(2 \cdot w - h_b) \cdot |\mathcal{I}_k|/2} \right) \cdot \prod_{k=\ell+1}^t \left( \frac{|\mathcal{I}_k|}{w \cdot |\mathcal{I}_k|} \right).
\]

We want to show that for any constant $\varepsilon > 0$, any fixed $0 \leq \ell \leq t$ and any fixed $b \in \{1, 2\}$ we have

\[
|\mathcal{L}_b \cap S_b^{[\ell]}| \leq 2 \cdot (1 + 2^{\varepsilon n}) \cdot (1 + |\mathcal{L}_b| \cdot |S_b^{[\ell]}/|w^n|)) \]

with a probability of $1 - \mathcal{O}(2^{-\varepsilon n})$. Notice that this shows the lemma by making use of the union bound, because $1 - \mathcal{O}(2(t+1) \cdot 2^{-\varepsilon n}) = 1 - \mathcal{O}(2^{-\varepsilon n})$.

Notice that $\mathcal{L}_b \cap S_b^{[\ell]} = \{ x \in S_b^{[\ell]} \mid f_a(x) = s_b \}$, since $S_b^{[\ell]} \subseteq \mathbb{Z}^n[w]$. The cardinality of these kinds of sets is studied in the Upper Bound Lemma, which states that

\[
|\mathcal{L}_b \cap S_b^{[\ell]}| \leq (1 + 2^{\varepsilon n}) \cdot (1 + |S_b^{[\ell]}|/|\mathcal{G}|)
\]

with a probability of $1 - \mathcal{O}(2^{-\varepsilon n})$. Moreover, it follows from the Lower Bound Lemma that

\[
|\mathcal{L}_b| \geq \frac{1}{2} \cdot |\mathbb{Z}^n[w]|/|\mathcal{G}|
\]

with a probability of $1 - \mathcal{O}(2^{-\varepsilon n})$, due to the fact that $|\mathbb{Z}^n[w]|/|\mathcal{G}| \geq 2^{2\varepsilon n}$. This implies

\[
|\mathcal{L}_b \cap S_b^{[\ell]}| \leq (1 + 2^{\varepsilon n}) \left( 1 + \frac{|S_b^{[\ell]}|}{|\mathcal{G}|} \right) \leq (1 + 2^{\varepsilon n}) \left( 1 + 2 \cdot \frac{|\mathcal{L}_b| \cdot |S_b^{[\ell]}|}{|\mathbb{Z}^n[w]|} \right) \leq 2(1 + 2^{\varepsilon n}) \left( 1 + \frac{|\mathcal{L}_b|}{|\mathbb{Z}^n[w]|} \right)
\]

with a probability of $1 - \mathcal{O}(2^{-\varepsilon n})$, which shows the lemma. 

\[
\square
\]
3.4.2 Algorithm

With this tool at hand, we are now able to handle inconsistencies more efficiently. Whereas in the algorithm CLASSICALREP the problem is solved naively, by simply comparing all pairs of elements with matching target values, the new idea is to create two exponential size lists that contain all elements with identical target value. The search for all vectors in these two lists with a certain output weight distribution is then exactly the Group Weight Match Problem. In the algorithm CONSISTENTREP, the two lists $L_1$, $L_2$ are either constructed recursively or in a meet-in-the-middle manner. The algorithm then combines these two lists to a list $L_{\text{out}}$ that contains all sums of an element in $L_1$ and an element in $L_2$ with a target value of $s$ in the group $G_\ell \times \ldots \times G_u$. This is done by firstly sorting the lists by their corresponding target values. Having these two lists, it is easily possible to shorten these lists by removing elements that violate a certain upper bound on the number of elements with a certain target value. Then, two sets $B_1, B_2$ are constructed that contain all target values of the elements in $L_1$ resp. $L_2$.

Algorithm 10 CONSISTENTREP

```plaintext
1: Global: $u \in \mathbb{N}_{\geq 2}$, $G = G_0 \times \ldots \times G_u$, $w_0, \ldots, w_u$, $n, g \in \mathbb{N}$, $a \in G^n$, $\varepsilon > 0$
2: Input: $\ell \in \{0, \ldots, u\}$, $s \in G_\ell \times \ldots \times G_u$ \quad \triangleright \text{init with } \ell = 0
3: Output: A list of $x \in \mathbb{Z}^n[w_\ell]$ with $f_{a[G_\ell \times \ldots \times G_u]}(x) = s$.
4: if $\ell = u$ then
5: \quad return MITMLIST($G_u, w_u, n, a[G_u], s, \varepsilon$)
6: else
7: \quad $s' \leftarrow G_{\ell+1} \times \ldots \times G_2$
8: \quad $L_1 \leftarrow$ CONSISTENTREP($\ell + 1, s'$)
9: \quad $L_2 \leftarrow$ CONSISTENTREP($\ell + 1, s[G_{\ell+1} \times \ldots \times G_u] - s'$)
10: \quad Sort $L_1$ by the target values $f_{a[G_\ell \times \ldots \times G_u]}(x_1)$ for each $x_1 \in L_1$.
11: \quad Sort $L_2$ by the target values $f_{a[G_\ell \times \ldots \times G_u]}(x_2)$ for each $x_2 \in L_2$.
12: \quad Shorten $L_1, L_2$ s.t. $(1 + 2^{\varepsilon n}) \cdot (1 + |\mathbb{Z}^n[w_{\ell+1}]| / |G_\ell \times \ldots \times G_u|)$ elements of each target.
13: \quad Construct two sets $B_1, B_2$ with all target values of $L_1$ resp. $L_2$.
14: $L_{\text{out}} \leftarrow \{\}$
15: for each $b \in B_1$ do
16: \quad if $s - b \in B_2$ then
17: \quad \quad $L_1 \leftarrow \{x_1 \in L_1 \text{ with } f_{a[G_\ell \times \ldots \times G_u]}(x_1) = b\}$
18: \quad \quad $L_2 \leftarrow \{x_2 \in L_2 \text{ with } f_{a[G_\ell \times \ldots \times G_u]}(x_2) = s - b\}$
19: \quad $L_{\text{out}} \leftarrow L_{\text{out}} \cup \text{WEIGHTSIEVE}(n, g, w_{\ell+1}, w_{\ell+1}, w_\ell, L_1, L_2, \varepsilon)$
20: \quad if $|L_{\text{out}}| > (1 + 2^{\varepsilon n}) \cdot (1 + |\mathbb{Z}^n[w_\ell]| / |G_\ell \times \ldots \times G_u|)$ then
21: \quad \quad return $L_{\text{out}}$ \quad \triangleright \text{stop, if output list exceeds upper bound}
22: return $L_{\text{out}}$
```

In the loop, we consider each target value $b$ and search for a possible corresponding target value $s - b$ in the second list. If there is a match, two sub-lists $L_1 \subseteq L_1$ and $L_2 \subseteq L_2$ are created, with target values $b$ resp. $s - b$. For these two lists, we now apply the algorithm WEIGHTSIEVE, which is filtering to only those pairs of a vector $x_1$ from the first list and a vector $x_2$ from the second list such that $x_1 + x_2 \mod g$ has the desired weight distribution $w_\ell$. As a special case,
the algorithm WeightSieve can still run the naive algorithm, which simply compares each element of $\mathcal{L}_1$ with each element of $\mathcal{L}_2$.

All these outputs are merged to an output list $\mathcal{L}_{out}$ of vectors with a weight of $w_\ell$. Notice that the output list gets cut if it exceeds a certain upper bound, which guarantees that there is a provable running time, independent of the algorithm’s success or fail.

In the following theorem, we prove the time complexity and correctness of the algorithm ConsistentRep, restricting the application of the non-naive algorithm WeightSieve to the second-highest level. The reason for this restriction is on the one hand to simplify the notation and readability of the theorem and on the other hand, because for all applications the improvement in applying the algorithm on additional levels is marginal. A similar theorem can be proven for algorithms that use the improved technique on more than one level.

**Theorem 47 (ConsistentRep).** Let $(a, s)$ be an instance of a $(G, w_0, n)$ Random Subset Sum Problem. Let $u \in \mathbb{N}_{\geq 2}$ be a constant independent of $n$ such that $G = G_0 \times \cdots \times G_u$ and let $g \in \mathbb{N}$ be a divisor of $|G|$ such that all elements in $G$ have an order that divides $g$. Let $w_0, \ldots, w_u$ be valid weight distributions with $r_{w_0, w_{\ell+1}}^{n, g} \geq 1$ and $|Z^n[w_{\ell+1}]| \geq |G_\ell| \cdots |G_u|$ for all $0 \leq \ell \leq u - 1$.

Then for any constant $\varepsilon > 0$, if $\frac{1}{2} \leq |G_\ell| \cdots |G_u|/r_{w_0, w_{\ell+1}}^{n, g} \cdot (2^\varepsilon n)^2 \leq 1$ for all $1 \leq \ell \leq u$, if $|Z^n[w_\ell]|/(|G_1| \cdots |G_u|) \geq 2^{\varepsilon n}$, if there is a unique joint distribution $\Gamma$ with $\psi_1(\Gamma) = \psi_2(\Gamma) = \psi_{\ell+1}(\Gamma)$ and $\psi_{\ell}^{\text{out}}(\Gamma) = w_1$, for any weight distributions $h_1, h_2 \leq 2 \cdot w_\ell$ such that $z := z_1 = z_2 = \lambda$ and $|C| \geq 1$, the algorithm ConsistentRep with global input $(u, G, w_0, \ldots, w_u, n, g, a, \varepsilon)$ and local input $(0, s)$ solves the instance with a probability (over both the coins of the algorithm and the choice of $a$ and $s$) of $1 - O(2^{-\varepsilon n})$ in time

$$
\tilde{O}\left(2^{2\varepsilon n} + 2^{H(w_u)n/2} + 2^{4\varepsilon n} \cdot \frac{2H(w_u)n}{r_{w_0, w_{\ell+1}}, w_\ell} + 2^{6\varepsilon n} \cdot \sum_{\ell=0,\ell\neq 1,}^{u-1} \frac{2^{2H(m_{\ell+1})n}}{r_{w_0, w_{\ell+1}}, w_\ell} + 2(y+2\varepsilon)n\right),
$$

with $r_{w_0, w_0}^{n, g} := |G|$, $y := \max\left\{\lambda + \varepsilon, \min_{i,j \in C} \left\{1 - \sum_{i,j,\Gamma(i,j)\neq 0} H_\Gamma(i,j)(C(i,j)/2)\right\}\right\}$, upper bound on the list sizes $\lambda := \frac{1}{n} \cdot \log_2\{(1 + 2^n) \cdot |Z^n[w_2]|/(G_1 \times \cdots \times G_u)|\}$, the two list reduction values $z_1 := H(w_2) - \frac{1}{2}H(h_1) - \frac{1}{2}H(2 \cdot w_2 - h_1)$ and $z_2 := H(w_2) - \frac{1}{2}H(h_2) - \frac{1}{2}H(2 \cdot w_2 - h_2)$ and $C$ being the set of all joint distributions $C \leq 2 \cdot \Gamma$ with $\psi_1(C) = h_1$ and $\psi_2(C) = h_2$.

**Proof.** Due to the valid choice of the weight distributions $w_0, \ldots, w_u$, it is possible to apply both the Upper Bound Lemma and the Golden Solution Lemma. Moreover, the choice of the sizes of the subgroups of $G$ guarantees that $\frac{1}{2} \leq |G_\ell| \cdots |G_u|/r_{w_0, w_{\ell+1}}^{n, g} \cdot (2^\varepsilon n)^2 \leq 1$ for all $0 \leq \ell \leq u - 1$.

In the following, we also want to assume that there is an $\bar{x} \in \mathbb{Z}^n[w_0]$ with $f_a(\bar{x}) = s$ (in the whole group $G$). Otherwise the output list is obviously empty. Notice that due to the fact that the order of each individual group element is a divisor of $g$, also $f_a(\bar{x} + y) = s$ for any $y \in (g\mathbb{Z})^n$.

This enables us to consider each intermediate solution vector modulo $g$.

The algorithm creates a binary computation tree of depth $u$. Notice that in each node of the tree, the input parameter $s$ is a uniformly random element from the respective group $G_\ell \times \cdots \times G_u$. This is the case, since the initial $s$ is uniform, $s'$ is chosen uniformly at random by the algorithm, which makes also $s[G_\ell \times \cdots \times G_u] - s'$ uniform. In each call for $\ell \neq u$ (a total of $2^u - 1$ nodes), we have to guarantee a representation in $\mathcal{L}_1$ with target value $s'$. For each of these nodes, the Golden Solution Lemma guarantees this with a probability of at least $1 - 2 \cdot 2^{-\varepsilon n}$. This can be shown by choosing the group as $G_\ell \times \cdots \times G_u$ and the set $\mathcal{B} = \mathcal{R}_{w_0, w_{\ell+1}}^{n, g}(\bar{x})$, with $\bar{x}$ being the golden solution of the previous step for each node with $0 \leq \ell \leq u - 1$, fulfilling the restriction of the lemma and guaranteeing at least one $\bar{x}_1 \in \mathcal{R}_{w_0, w_{\ell+1}}^{n, g}(\bar{x})$ with $f_a(G_\ell \times \cdots \times G_u)(\bar{x}_1) = s'$.
Let us assume we have golden solutions in all nodes, i.e. a promise that there is at least one \( \bar{x} \in \mathbb{Z}^n_u \) with \( f_{a[G_x \times \ldots \times G_u]}(\bar{x}) = s \). Then we first of all have to take care that MitmList works correctly on all \( 2^u \) leaf nodes. As can be seen in Theorem 37 with a probability of \( 1 - \mathcal{O}(2^{-\varepsilon n}) \) for each node, the fixed solution \( \bar{x} \in \mathbb{Z}^n_u \) with \( f_{a[G_u]}(\bar{x}) = s \) in \( G_u \) is found in time \( \tilde{O}(2^H(w_u)n/2 + 2^\varepsilon n + 2^{\varepsilon n} \cdot 2^H(w_u)n/|G_u|) \).

For \( \ell \neq u \) we have to guarantee two upper bounds for each node. The first bound is one on the output list \( L_{\text{out}} \). With the help of the Upper Bound Lemma instantiated with the group \( G_{\ell} \times \ldots \times G_u \) and the choice \( B = \mathbb{Z}^n_w \), due to the uniform choice of \( a \) and the input \( s \), the probability that \( |L_{\text{out}}| \leq (1 + 2^{\varepsilon n}) \cdot (1 + |\mathbb{Z}^n_w|/|G_{\ell} \times \ldots \times G_u|) \) is at least \( 1 - 2 \cdot 2^{-\varepsilon n} \) for each of the \( 2^u - 1 \) nodes.

It is left to show the second upper bound on the number of elements in \( L_1 \) resp. \( L_2 \) with a target value of \( f_{a[G_x \times \ldots \times G_u]}(\bar{x}_1) \) resp. \( f_{a[G_x \times \ldots \times G_u]}(\bar{x}_2) \), \( \bar{x}_1 \) being the representation guaranteed by the Golden Solution Lemma with the uniformly random target value \( f_{a[G_{\ell+1} \times \ldots \times G_2]}(\bar{x}_1) = s' \) and \( \bar{x}_2 \) being the corresponding value such that \( \bar{x}_1 + \bar{x}_2 = \bar{x} \). With the help of Lemma 30 we can use the uniformity of \( a[G_{\ell}] \) to show the uniformity of \( f_{a[G_{\ell}]}(\bar{x}_1) \). This means that also \( f_{a[G_x \times \ldots \times G_u]}(\bar{x}_1) \) is uniformly distributed, which makes the Upper Bound Lemma applicable. Analogously, it can be shown for \( \bar{x}_2 \). The Upper Bound Lemma tells us in each case that there are at most \((1 + 2^{\varepsilon n}) \cdot (1 + |\mathbb{Z}^n_w|/|G_{\ell} \times \ldots \times G_u|)\) elements with each of these particular target values with a probability of each at least \( 1 - 2 \cdot 2^{-\varepsilon n} \).

The algorithm receives two lists \( L_1, L_2 \) with \( \bar{x}_1 \in L_1, \bar{x}_2 \in L_2 \), with \( \bar{x}_1, \bar{x}_2 \) being the golden solutions of the previous level. These vectors remain in the list after the shortening, due to the bounds shown above. Notice that there is a golden solution \( \bar{x} = \bar{x}_1 + \bar{x}_2 \) mod \( g \) with \( \bar{x} \in \mathbb{Z}^n_w \) and \( f_{a[G_x \times \ldots \times G_u]}(\bar{x}) = s \) on the next level. The algorithm Weightsieve can now be implemented such that it runs the naive algorithm on all levels \( \ell \neq 1 \) and the improved algorithm on level \( \ell = 1 \). In the naive case, the golden solution \( \bar{x} \) is therefore clearly a part of the output. In the non-naive case, we want to apply Lemma 46. This lemma requires \( w_2 \) and \( w_1 \) to be valid weight distributions and the existence of a unique joint distribution \( \Gamma \) such that \( \psi_1(\Gamma) = \psi_2(\Gamma) = w_2 \) and \( \psi_{\text{out}}(\Gamma) = w_1 \), which is the case. Due to the fact that \( a, f_{a[G_x \times \ldots \times G_u]}(\bar{x}_1) \) and \( f_{a[G_x \times \ldots \times G_u]}(\bar{x}_2) \) are uniformly random, we therefore get a Group Weight Match Problem with input lists \( L_1 \) and \( L_2 \). Thus, the algorithm constructs the golden solution \( \bar{x} \in \mathbb{Z}^n_w \) with \( \bar{x}_1 + \bar{x}_2 \) mod \( g \) with a probability of \( 1 - \mathcal{O}(2^{-\varepsilon n}) \). The upper bound at the end isn’t exceeded, as shown above, such that the golden solution is part of the output in every node. This means that with the union bound, the probability that the algorithm fails in any of these steps is \( 1 - \mathcal{O}(2^{-\varepsilon n}) \) due to the fact that \( u \) is constant.

The time complexity is first of all that of the MitmList algorithm that is called \( 2^u \) times. Each of these calls have a complexity of

\[
\tilde{O}(2^H(w_u)n/2)
\]

for creating and processing the two sub-lists and an additional complexity of

\[
\tilde{O}(2^\varepsilon n + 2^{\varepsilon n} \cdot 2^H(w_u)n/|G_u|)
\]

for creating the output list, which means that the second complexity becomes

\[
\tilde{O}(2^\varepsilon n + (2^\varepsilon n)^3 \cdot 2^H(w_u)n/|G_u|^{n/2})
\]

due to the lower bound on \( |G_u| \).
For $\ell \neq u$ we always have lists $\mathcal{L}_1, \mathcal{L}_2$ of sizes less than

$$(1 + 2^{cn}) \cdot (1 + |\mathbb{Z}^n[w_{\ell+1}]|/|G_{\ell+1} \times \ldots \times G_u|)$$

elements. These are processed with negligible overhead. Notice that $|B_1| \leq |G_\ell|$ and that for each element in $B_1$ there is at most one element in $B_2$ that matches the target value. In the worst case, for each element in $B_1$, two lists $\tilde{\mathcal{L}}_1, \tilde{\mathcal{L}}_2$ with each having an upper bound

$$(1 + 2^{cn}) \cdot (1 + |\mathbb{Z}^n[w_{\ell+1}]|/|G_{\ell} \times \ldots \times G_u|)$$

are created. If $\ell \neq 1$, the algorithm WeightSieve is implemented naively, which means due to the upper bounds that we have a time complexity

$$\tilde{\mathcal{O}} \left( (2^{cn})^2 \cdot \left( 1 + \frac{|\mathbb{Z}^n[w_{\ell+1}]|}{|G_{\ell+1} \times \ldots \times G_u|} + \frac{|\mathbb{Z}^n[w_{\ell+1}]|^2}{|G_{\ell} \times \ldots \times G_u| \cdot |G_{\ell+1} \times \ldots \times G_u|} \right) \right),$$

which can be upper bounded by

$$\tilde{\mathcal{O}} \left( (2^{cn})^2 + (2^{cn})^4 \cdot \frac{|\mathbb{Z}^n[w_{\ell+1}]|}{r_{w_0,\ell,\ell+1}^{n,g}} + (2^{cn})^6 \cdot \frac{|\mathbb{Z}^n[w_{\ell+1}]|^2}{r_{w_0,\ell-1,\ell}^{n,g} \cdot r_{w_0,\ell+1}^{n,g}} \right)$$

for $\ell \geq 2$. Notice that this bound also holds for $\ell = 0$, if we denote $r_{w_0,\ell,\ell}^{n,g} := |G|$. In the special case $\ell = 1$, in which we use Lemma 46, the merging is more efficient. Due to the fact that we choose $z_1 = z_2 = \lambda$ with $\lambda$ being the logarithm of an upper bound on the list sizes that is fulfilled as shown above, the time complexity of this step simplifies to $2^{(y+2\epsilon)n}$ with the value of $y$ defined in the statement of the theorem.

Since the size of the output list is always at most the complexity of the merging, it can be left out of the overall time complexity that suppresses constant factors.

This result from [MO] can be used to improve algorithms for the Binary Subset Sum Problem, as well as the Knapsack Problem as shown in Chapter 4 and Chapter 5.
3. Subset Sum Problem
Chapter 4

Binary Subset Sum Problem

In this chapter, we want to analyze a special case of the Random Subset Sum Problem, in which the weight distribution \( w \) is such that only the 0 and 1 components are non-zero. This problem is surely the most common instantiation and is also applicable in several groups \( G \). Possible choices for the groups are the Knapsack Problem [Kar72] with \( G = \mathbb{Z}_N \), the Decoding Problem [McE78] with \( G = \mathbb{F}_2^{n-k} \) and the NTRU Problem [HPS98] with \( G = \mathbb{F}_q^n \). Impagliazzo and Naor [IN96] showed that there are provably secure schemes based on the Binary Subset Sum Problem, starting a fruitful chain of research. Ajtai [Ajt96] introduced a link to lattices and identified a class of hard instances. Variants of the Binary Subset Sum Problem appear e.g. as a hardness assumption in works by Ajtai and Dwork [AD97], Regev [Reg03, Reg05], Peikert [Pei09] and Lyubashevsky et al. [LMPR08, LPS10]. We derive a generic result for arbitrary groups \( G \) in this chapter, which is applied to the Knapsack Problem in Chapter 5 and the Decoding Problem in Chapter 6. Notice that similar results can be obtained for the Ternary Subset Sum Problem, which isn’t covered by this thesis, but is, for example, interesting for the NTRU Problem and was analyzed in [Oze12]. The definition of the Random Binary Subset Sum Problem is as follows.

**Definition 48** (Random Binary Subset Sum Problem). Let \( G \) be a group, \( n \in \mathbb{N} \), \( 0 \leq \delta \leq \frac{1}{2} \) and \( w \) a weight distribution with \( w(1) = \delta \) and \( w(0) = 1 - \delta \). In the \((G, n, \delta)\) Random Binary Subset Sum Problem we are given \( a \in G^n \) and \( s \in G \) chosen uniformly at random. The problem is to output a list that contains any fixed \( \bar{x} \in \mathbb{Z}^n[w] \) with \( f_a(\bar{x}) = s \), or an empty list if no \( x \in \mathbb{Z}^n[w] \) with \( f_a(x) = s \) exists.

Notice that this problem can be easily extended to \( 0 \leq \delta \leq 1 \), i.e. allowing \( \delta > \frac{1}{2} \). However, each problem with a \( \delta > \frac{1}{2} \) can be solved with an algorithm that is built for \( \delta < \frac{1}{2} \) by replacing the target \( s \) with the value \( s' = \sum_{i=1}^n a_i - s \). Due to the fact that this is only a shift, it preserves the fact that the target value is uniformly chosen. Additionally for each \( \delta \geq \frac{1}{2} \), there is a \( I \subseteq [n] \) with \( |I| = (1 - \delta) \cdot n \leq \frac{1}{2} \cdot n \) with \( \sum_{i \in I} a_i = s' \) iff. there is a \( J = [n] \setminus I \) with \( |J| = \delta \cdot n \) with \( \sum_{i \in J} a_i = s \). Due to the existence of this transformation, we restrict to \( \delta \leq \frac{1}{2} \) in the following study of algorithms.

The Random Binary Subset Sum Problem can be solved with the brute-force technique in time \( (\delta^n)^{O(2^{H(\delta)n})} \), by checking for each possible \( \{0, 1\}\)-vector \( x \in \mathbb{Z}^n[w] \) if it leads to the target value \( s \). In the following, we want to present several algorithms for this problem that are derived from the general analysis of the previous chapter. Notice that in most cases we obtain non-linear optimization problems in many unknowns and therefore rely on numerical methods.
4. Binary Subset Sum Problem

We solve the problems with the help of Mathematica 10.2 [Res15] and compare the obtained results at certain values of $\delta$.

In Sect. 4.1 we present known results, in Sect. 4.2 some new results that are based on [MO]. These are important for the Knapsack Problem in Chapter 5 as well as the Decoding Problem in Chapter 6. The results for special groups in Sect. 4.3 are mainly important for Chapter 6.

4.1 Known Results

4.1.1 Meet-in-the-Middle

For a long time, the fastest known algorithm to solve the problem was the meet-in-the-middle technique by Horowitz and Sahni [HS74] with time complexity $\tilde{O}(2^{H(\delta)n/2})$. The main idea is that there is a unique disjoint decomposition of $x$ into two vectors $x^{[1]}, x^{[2]} \in \{0, 1\}^{n/2}$ with relative weight $\delta$ such that $x = (x^{[1]}, x^{[2]})$. This allows us to restate the problem as

$$\sum_{i=1}^{n/2} a_i x_i^{[1]} = s - \sum_{i=1}^{n/2} a_i x_i^{[2]}.$$ 

Now, enumerating all possible $x^{[1]}$ and storing the left hand side value in a list, sorting this list and searching for collisions with the right hand part solves the problem. Thus, both list creation and collision finding takes time only $\tilde{O}(2^{H(\delta)n/2})$. The following corollary follows directly from the results of Chapter 3.

**Corollary 49** (meet-in-the-middle). Let $(a, s)$ be an instance of a $(G, n, \delta)$ Random Binary Subset Sum Problem with corresponding weight distribution $w$ such that $w(1) = \delta$ and $w(0) = 1 - \delta$. Then for any $\varepsilon > 0$, the algorithm MitmList with input $(G, n, w, a, s, \varepsilon)$ solves the instance with a probability of $1 - \tilde{O}(2^{-\varepsilon n})$ (over the choice of the input and the coins of the algorithm) in time $\tilde{O}(2^{H(\delta)n/2} + 2^{\varepsilon n} + 2^{\varepsilon n} \cdot 2^{H(\delta)n/|G|})$.

The correctness and time complexity follows directly from Theorem 37. In order to be able to compare the results quantitatively, we want to present the complexity exponents for several values of $\delta$ assuming an arbitrary small $\varepsilon$, i.e. $\varepsilon = 0$ for the sake of simplicity. The complexity exponent for this algorithm is therefore $C = \max\{\tilde{C}, C_0\}$ with $C_0 = H(\delta)/2$ and

$$\tilde{C} = H(\delta) - \frac{1}{n} \log_2(|G|).$$

Notice that the latter complexity depends on the size of the group $G$ and if $|G|$ is small, this complexity might dominate the running time of the whole algorithm. However, $|G| = \tilde{\Omega}(2^{H(\delta)n/2})$ is usually chosen such that this special complexity is dominated by the remaining running time of the algorithm. In the following, we want to assume the latter and exclude this complexity from the analysis. The remaining complexity $C_0$ has the following numerical values.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0$</td>
<td>0.144</td>
<td>0.235</td>
<td>0.305</td>
<td>0.361</td>
<td>0.406</td>
<td>0.441</td>
<td>0.468</td>
<td>0.486</td>
<td>0.497</td>
<td>0.500</td>
</tr>
</tbody>
</table>

**Table 4.1:** numerical values for meet-in-the-middle

4.1.2 Representations I

In this section, we study two algorithms presented by Howgrave-Graham and Joux [HJ10] that follow directly from Theorem 43 using the algorithm ClassicalRep. In the algorithm, one
has to choose a recursion depth \( u \in \mathbb{N} \). We only consider the cases \( u = 1 \) in Corollary 50 and \( u = 2 \) in Corollary 51. Notice that \( u > 2 \) doesn’t lead to better results due to the fact that the meet-in-the-middle sub-complexity in the algorithm is already dominated by the remaining part of the complexity. Let us begin with the result for one level from [HJ10].

**Corollary 50** (rep1.1). Let \((a, s)\) be an instance of a \((G, n, \delta)\) Random Binary Subset Sum Problem. Let \( u = 1, \ g = |G|, \ G = G_0 \times G_1 \) with \(|G| \geq 2, \ \omega_0 \) be such that \( \omega_0(1) = \delta, \ \omega_0(0) = 1 - \delta \) and let \( \omega_1 \) be such that \( \omega_1(1) = \frac{\delta}{2} \) and \( \omega_1(0) = 1 - \frac{\delta}{2} \). Then for any \( \varepsilon > 0 \), if \( \frac{1}{2} \leq (2^n)^2 \cdot |G_1|/(\delta \cdot n/2) \leq 1 \), the algorithm \textsc{ClassicalRep} with global input \((u, G, \omega_0, \omega_1, n, g, a, \varepsilon)\) and local input \((0, s)\) solves the instance with a probability of \( 1 - \mathcal{O}(2^{-\varepsilon n}) \) (over the choice of the input and the coins of the algorithm) in time

\[
\tilde{O}\left(2^{cn} + 2^{C_1 n} + 2^{6cn} + 2^{6cn} + 2^{(2H(\delta/2) - \delta)n}/|G|\right),
\]

with \( C_0 = H(\delta/2) - \delta \) and \( C_1 = H(\delta/2)/2 \).

Let us explain the parameter choice that leads to this result. As already mentioned, we choose \( u = 1 \), which means only one level of representations. We choose \( g = |G| \), which means we don’t have any better upper bound on the maximal element order of the group elements, i.e. do not impose any further restriction on the group \( G \). The weight \( \omega_0 \) corresponds to the given Random Binary Subset Sum Problem, whereas the weight \( \omega_1 \) is chosen accordingly. Notice that both weight distributions are valid, due to the fact that all non-zero parts lie in an interval of size 2 and \(|G| \geq 2\). The number of representations is \( r_{\omega_0, \omega_1}^{n, |G|} = (\delta \cdot n) \), since out of the \( \delta \cdot n \) ones of a vector of weight \( \omega_0 \), one is allowed to choose \( \delta \cdot n/2 \) ones for the first representing vector, which determines the second representing vector of the same weight completely. In other words, for each \( x \in \mathbb{Z}^n[\omega_0] \) there are \( (\delta \cdot n/2) \) possible \( x_1 \in \mathbb{Z}^n[\omega_1] \) such that there is a \( x_2 \in \mathbb{Z}^n[\omega_1] \) with \( x_1 + x_2 = x \mod |G| \). This directly implies that the number of representations is at least 1. Notice that the choice of \( \omega_1 \) is unique, since any other number of ones would lead to sums of vectors with weight \( \omega_1 \) that are not of weight \( \omega_0 \). Thus, since all requirements for Theorem 43 are fulfilled, we obtain the presented time complexity. In its computation, we make use of the fact that \( (\delta \cdot n/2) \approx \Theta(2^{\delta n}) \). We denote \( R_{0,1} = \delta \) the exponent of the number of representations.

\[
\begin{align*}
\text{construction time:} & \\
\binom{n/2}{\delta \cdot n/2} & \approx 2^{C_1 n} & \quad \text{final list size:} & \\
1 & \quad \quad & \binom{n}{\delta \cdot n/2}/\binom{n}{\delta \cdot n/2} & \approx 2^{C_0 n}
\end{align*}
\]

Figure 4.1: complexities of the classical representation algorithm for \( u = 1 \)

Let us quickly recall the idea of the algorithm \textsc{ClassicalRep} on one level, which is summarized in Fig. 4.1. For the sake of simplicity, we want to ignore the \( \varepsilon \)-overhead in the following analysis of the time complexity. Moreover, we want to ignore the part of the complexity that depends on the group size \(|G|\), assuming \(|G|\) is large enough that this complexity doesn’t dominate. In this case, the complexity exponent is therefore simply \( \max\{C_0, C_1\} \).
The main idea is to initially create two lists that are finally merged into one. These two lists both consist of vectors with half the initial weight that are fixed to a certain element in $G_1$. Due to choice of the group size fitting the number of representations, the number of elements in these lists is therefore roughly $\left(\frac{n}{\delta \cdot n/2}\right) / \left(\frac{\delta \cdot n/2}{\delta \cdot n/4}\right) \approx 2^{C_0 n}$. The construction of these lists is performed in a meet-in-the-middle manner with vectors of half the size and quarter of the initial weight and can therefore be done in time $\left(\frac{n/2}{\delta \cdot n/4}\right) \approx 2^{C_1 n}$.

In the following table, we want to present numerical complexity values for certain relative weights $\delta$. The maximal complexity is always underlined. The $\delta$ with the maximal overall complexity of $2^{0.406 \delta}$ can be computed numerically at $\delta = \frac{1}{2}$.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.313</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0$</td>
<td>0.119</td>
<td>0.187</td>
<td>0.235</td>
<td>0.269</td>
<td>0.294</td>
<td>0.310</td>
<td>0.313</td>
<td>0.320</td>
<td>0.322</td>
<td>0.320</td>
<td>0.312</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.085</td>
<td>0.144</td>
<td>0.193</td>
<td>0.235</td>
<td>0.272</td>
<td>0.305</td>
<td>0.313</td>
<td>0.335</td>
<td>0.361</td>
<td>0.385</td>
<td>0.406</td>
</tr>
<tr>
<td>$R_{0.1}$</td>
<td>0.050</td>
<td>0.100</td>
<td>0.150</td>
<td>0.200</td>
<td>0.250</td>
<td>0.300</td>
<td>0.313</td>
<td>0.350</td>
<td>0.400</td>
<td>0.450</td>
<td>0.500</td>
</tr>
</tbody>
</table>

Table 4.2: numerical values for classical representations with $u = 1$

Notice that for some $\Delta \approx 0.313$, the maximum of these two partial complexities is the top-level complexity $C_0$ for $\delta < \Delta$, whereas for $\delta > \Delta$ the construction on the bottom level $C_1$ dominates the time complexity. Though compared to the meet-in-the-middle approach the maximum of both complexities – which is the overall time complexity of this $u = 1$ approach – is better for each $\delta$, one might consider to analyze the $u = 2$ case for any $\delta > \Delta$, since increasing $u$ might lead to a decrease of the time complexity on the bottom level of the algorithm. However, using $u = 2$ for $\delta < \Delta$ can’t improve the result due to the fact that the top-level complexity remains identical. The result for $u = 2$ (also from [HJJ10]) is as follows. We can see that the complexity even gets worse in the range $\delta < \Delta$ due to the dealing with inconsistencies.

**Corollary 51 (rep1.2).** Let $(a, s)$ be an instance of a $(G, n, \delta)$ Random Binary Subset Sum Problem. Let $u = 2$, $g = |G|$, $G = G_0 \times G_1 \times G_2$ with $|G| \geq 2$, $w_0$ be such that $w_0(1) = \delta$, $w_0(0) = 1 - \delta$, $w_1$ be such that $w_1(1) = \frac{\delta}{2}$ and $w_1(0) = 1 - \frac{\delta}{2}$ and let $w_2$ be such that $w_2(1) = \frac{\delta}{4}$ and $w_2(0) = 1 - 4 \cdot \frac{\delta}{4}$. Then for any $\varepsilon > 0$, if $\frac{1}{2} \leq \left(2^{\varepsilon n} \cdot 2^{\frac{\delta}{2}} \cdot |G| \times |G_2| / \left(\frac{\delta \cdot n/2}{\delta \cdot n/4}\right) \leq 1 \text{ and } \frac{1}{2} \leq \left(2^{\varepsilon n} \cdot 2^{\frac{\delta}{4}} \cdot |G| \times |G_2| / \left(\frac{\delta \cdot n/2}{\delta \cdot n/4}\right) \leq 1\right)$, the algorithm ClassicalRep with global input $(u, G, w_0, w_1, w_2, n, g, a, \varepsilon)$ and local input $(0, s)$ solves the instance with a probability of $1 - \mathcal{O}(2^{-\varepsilon n})$ (over the choice of the input and the coins of the algorithm) in time

$$\tilde{O} \left(2^{\varepsilon n} + 2^{C_2 n} + 2^{\frac{\delta}{2}} \cdot 2^{C_1 n} + 2^{\frac{\delta}{2}} \cdot 2^{C_0 n} + 2^{\frac{\delta}{2}} \cdot 2^{C_0 n} \cdot 2^{2(\mathcal{H}(\delta/2) - \frac{\delta}{2}) n / |G|} \right),$$

with $C_0 = 2 \mathcal{H}(\delta/4) - \frac{3}{2} \delta$, $C_1 = \mathcal{H}(\delta/4) - \frac{\delta}{2}$ and $C_2 = \mathcal{H}(\delta/4) / 2$.

In this case, the choice $u = 2$ leads to two levels of representations. The weight $w_0$ corresponds to the given Random Binary Subset Sum Problem, whereas the weights $w_1, w_2$ are chosen accordingly. Notice that the weight distributions are valid, due to the fact that $\sigma(w_0) = \{0, 1\}$ and $|G| \geq 2$. The number of representations is $r_{w_0,w_1}^{n} = \left(\frac{\delta \cdot n}{\delta \cdot n/2}\right)$ as above and $r_{w_0,w_2}^{n} = \left(\frac{\delta \cdot n}{\delta \cdot n/4}\right)$ since out of the $\delta \cdot n/2$ ones of a vector of weight $w_1$, one is allowed to choose $\delta \cdot n/4$ ones for the first representing vector, which determines the second representing vector of the same weight completely. In other words, for each $x \in \mathbb{Z}^n[w_1]$ there are $\left(\frac{\delta \cdot n}{\delta \cdot n/4}\right)$ possible $x_1 \in \mathbb{Z}^n[w_2]$ such
that there is a $\mathbf{x}_2 \in \mathbb{Z}^n[\mathbf{w}_2]$ with $\mathbf{x}_1 + \mathbf{x}_2 = \mathbf{x} \mod |\mathcal{G}|$. This directly implies that the number of representations is at least 1 in both cases. Notice that the choice of $\mathbf{w}_1$ and $\mathbf{w}_2$ is unique, as any other choice would lead to sums of the wrong weight. Thus, since all requirements for Theorem 43 are fulfilled, we obtain the presented time complexity. In its computation, we make use of the fact that $(\frac{\delta}{\delta-n/2}) = \tilde{\Theta}(2^{\delta n})$ and $(\frac{\delta-n/2}{\delta-n/4}) = \tilde{\Theta}(2^{\delta n/2})$. We further denote $R_{0,1} = \delta$ and $R_{1,2} = \frac{\delta}{2}$ the exponents of the number of representations.

![Figure 4.2: complexities of the classical representation algorithm for $u = 2$](image)

Let us analyze the extension of the algorithm CLASSICALREP to two levels, which is summarized in Fig. 4.2. Once again, we want to assume an arbitrary small $\varepsilon$ for the sake of simplicity as well as a sufficiently large group $\mathcal{G}$ in order to be able to ignore the part of the complexity that depends on the size of the group $\mathcal{G}$. First of all, four lists with roughly $(\frac{n}{\delta-n/4})/(\frac{\delta}{\delta-n/4}) \approx 2^{C_1 n}$ elements are constructed, where $(\frac{n}{\delta-n/4})$ is the set of all vectors with relative weight $\delta/4$ and $(\frac{\delta}{\delta-n/4})$ is the number of ways to represent a vector of relative weight $\delta/2$ as the sum of two vectors with relative weight $\delta/4$. In order to construct these lists, a polynomial number of lists with vectors that are zero on half the vector and have a weight of $\delta/8$ on the remaining part, which are of roughly size $(\frac{n}{\delta-n/8}) \approx 2^{C_2 n}$ is built.

These four lists are merged to two lists of size $(\frac{n}{\delta-n/2})/(\frac{\delta}{\delta-n/2})$ which are less than $2^{C_1 n}$ due to the fact that its construction – that is described as follows – is at least as costly. In order to compute these lists, we obtain roughly $(\frac{n}{\delta-n/2})/(\frac{\delta}{\delta-n/4})$ smaller list pairs that are checked for consistent vector pairs naively. Due to the fact that each individual of these list pairs is roughly of size $(\frac{n}{\delta-n/4})/(\frac{\delta}{\delta-n/2})$, the time complexity is the product of number times size, i.e.\[
(\frac{(\frac{n}{\delta-n/4})}{(\frac{\delta}{\delta-n/2})})^2 (\frac{(\frac{n}{\delta-n/2})}{(\frac{\delta}{\delta-n/4})}) \approx 2^{C_0 n}.
\]

We present numerical complexity values for several $\delta$ in Table 4.3. Interestingly, the worst-case complexity isn’t at $\delta = \frac{1}{2}$ any more, which is the case with the initially largest number of possible solutions. Instead, under the assumption that $C_0$ is always the dominating complexity, which can be verified numerically, it can be shown rigorously that the worst-case of $2^{0.540 n}$ is obtained at $\delta = \frac{4}{9}$. Moreover, since for small values of $\delta$ the complexity of the $u = 1$ approach is smaller, one should also consider the previous algorithm depending on the given $\delta$. Concretely, for $\delta < \Delta$ for some $\Delta \approx 0.313$ the $u = 1$ approach should be preferred.

Since the top level complexity $C_0$ is always dominating, an increase of the number of levels also doesn’t help to improve the complexity, since it would only further decrease the complexity $C_2$ and doesn’t influence $C_0$. Instead, one has to look for ways to decrease the top-level complexity $C_0$, which is done in the subsequent section and restores the more natural worst-case at $\delta = \frac{1}{2}$. 


4. Binary Subset Sum Problem

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.35</th>
<th>0.40</th>
<th>( \frac{1}{2} )</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_0 )</td>
<td>0.119</td>
<td>0.188</td>
<td>0.237</td>
<td>0.273</td>
<td>0.300</td>
<td>0.319</td>
<td>0.332</td>
<td>0.338</td>
<td>0.340</td>
<td>0.340</td>
<td>0.338</td>
</tr>
<tr>
<td>( C_1 )</td>
<td>0.072</td>
<td>0.119</td>
<td>0.156</td>
<td>0.187</td>
<td>0.213</td>
<td>0.235</td>
<td>0.254</td>
<td>0.269</td>
<td>0.282</td>
<td>0.283</td>
<td>0.294</td>
</tr>
<tr>
<td>( C_2 )</td>
<td>0.049</td>
<td>0.085</td>
<td>0.116</td>
<td>0.144</td>
<td>0.169</td>
<td>0.193</td>
<td>0.215</td>
<td>0.235</td>
<td>0.252</td>
<td>0.254</td>
<td>0.272</td>
</tr>
<tr>
<td>( R_{0,1} )</td>
<td>0.050</td>
<td>0.100</td>
<td>0.150</td>
<td>0.200</td>
<td>0.250</td>
<td>0.300</td>
<td>0.350</td>
<td>0.400</td>
<td>0.445</td>
<td>0.450</td>
<td>0.500</td>
</tr>
<tr>
<td>( R_{1,2} )</td>
<td>0.025</td>
<td>0.050</td>
<td>0.075</td>
<td>0.100</td>
<td>0.125</td>
<td>0.150</td>
<td>0.175</td>
<td>0.200</td>
<td>0.223</td>
<td>0.225</td>
<td>0.250</td>
</tr>
</tbody>
</table>

Table 4.3: numerical values for classical representations with \( u = 2 \)

4.1.3 Representations II

In this section, we want to require a group \( G \) that is of odd order, which means that the weight distributions chosen in the algorithms are allowed to have a support of cardinality \( 3 \), in order to be \( |G| \)-valid. This extension was presented by Becker, Coron and Joux [BCJ11] and is also implied by our algorithm ClassicalRep. The main idea is that instead of decompositions into binary vectors, one might also split into ternary vectors. For example, \( x = 1110 \) could also be represented as \( 110-1 + 0011 \) such that the last component sums to zero. This idea allows to increase the representations on the top level, decreasing its running time. Once again, we want to start with a result for only one level of representations from [BCJ11].

**Corollary 52** (rep2.1). Let \((a, s)\) be an instance of a \((G, n, \delta)\) Random Binary Subset Sum Problem. Let \( u = 1, g = |G|, G = G_0 \times G_1 \) with \( \gcd(|G|, 2) = 1, w_0 \) be such that \( w_0(1) = \delta, w_0(0) = 1 - \delta \) and let \( w_1 \) be such that \( w_1(1) = \frac{\delta}{2} + \frac{\tau}{2}, w_1(-1) = \frac{\tau}{2} \) and \( w_1(0) = 1 - \frac{\delta}{2} - \tau \).

Then for any \( 0 \leq \tau \leq 1 - \delta \) and any \( \varepsilon > 0 \), if \( \frac{1}{2} \leq (2^{\varepsilon n})^2 \cdot |G_1|/ \left( (\frac{\delta}{n}) \cdot (\tau_{n/2}, \tau_{n/2}) \right) \leq 1 \), the algorithm ClassicalRep with global input \((u, G, w_0, w_1, n, g, a, \varepsilon)\) and local input \((0, s)\) solves the instance with a probability of \( 1 - \mathcal{O}(2^{-\varepsilon n}) \) (over the choice of the input and the coins of the algorithm) in time

\[
\tilde{O} \left( 2^{\varepsilon n} + 2^{C_1 n} + 2^{6\varepsilon n} \cdot 2^{C_0 n} + 2^{6\varepsilon n} \cdot 2^{(2H(w_1) - R_{0,1})n}/|G| \right),
\]

with \( C_0 = H(w_1) - R_{0,1} \) and \( C_1 = H(w_1)/2 \), where \( H(w_1) = H(\frac{\delta}{2}, \tau_{n/2}) \) and \( R_{0,1} = \delta + H_{1-\delta}(\frac{\tau}{2}, \frac{\tau}{2}) \).

We choose \( u = 1 \), which means only one level of representations. The weight \( w_0 \) corresponds to the given Random Subset Sum Problem, whereas the weight \( w_1 \) is now chosen introducing -1 components. Each -1 component needs a corresponding 1 component in order to cancel to 0 and represent a vector with weight distribution \( w_0 \) that doesn’t have any -1 components. Therefore, the chosen \( \tau_{n/2} \) of the -1 components determines the number of ones and zeros in \( w_1 \). Notice that both weight distributions are \(|G| \)-valid. It is obviously true for \( w_0 \), because all non-zero parts lie in an interval of size 2 and \( |G| \geq 2 \). However, for \( w_1 \) to be \(|G| \)-valid, we need \( \gcd(|G|, 2) = 1 \), which implies that the smallest prime divisor of \(|G|\) is at least 3. The number of representations is \( \tau_{n/2}^{\frac{|G|}{n}} \cdot \tau_{n/2}^{\frac{n}{2}} \cdot (\tau_{n/2}, \tau_{n/2}) \), since out of the \( \delta \cdot n \) ones of a vector of weight \( w_0 \), one is allowed to choose \( \delta \cdot n/2 \) ones for the first representing vector, whereas out of the \((1 - \delta) \cdot n \) zeros of the \( w_0 \) vector, one can either choose \( \tau_{n/2} \cdot n \) negative ones (and the same number of ones in the second vector), \( \tau_{n/2} \cdot n \) ones (and the same number of negative ones in the second vector) or \((1 - \tau) \cdot n \) zeros (and the same number of zeros in the second vector). In other words, for each \( x \in \mathbb{Z}^n[w_0] \) there are \((\delta/2)^n \cdot (1-\delta)^n \cdot \tau_{n/2} \cdot \tau_{n/2} \) possible \( x_1 \in \mathbb{Z}^n[w_1] \) such that there is a \( x_2 \in \mathbb{Z}^n[w_1] \).
with \( x_1 + x_2 = x \mod |G| \). This directly implies that the number of representations is at least 1. Thus, since all requirements for Theorem 43 are fulfilled, we obtain the presented time complexity. In its computation, we make use of the fact that \((\delta^{n/2}, \tau_{n/2}) = \Theta(2^{n\cdot u})\).

\[
\text{construction time: } (\frac{\delta^{n/2}}{2}, \frac{\tau_{n/2}}{2}) \approx 2^{C_{1n}} \\
\text{final list size: } (\frac{\delta^{n/2}}{2}, \frac{\tau_{n/2}}{2}) \approx 2^{C_0 n}
\]

Figure 4.3: complexities of the classical ternary representation algorithm for \( u = 1 \)

Compared to the previous result with one level, we therefore add a certain number of ones and negative ones on the lower levels. This allows to have a smaller final list of size \( 2^{C_0 n} \) at the cost of a slightly larger construction time of \( 2^{C_{1n}} \). The optimal choice of the parameter \( \tau \) should therefore be a value that leads to \( C_0 = C_1 \), whenever this is possible. In the following table, we see improvements of this kind at small values of \( \delta \), with a \( \tau \) that leads to \( C_0 = C_1 \). However, for large values of \( \delta \) the bottom complexity \( C_1 \) is already dominating in the result of Corollary 50, which means that \( \tau = 0 \) is the optimal choice and we get no improvement compared to the previous result. The \( \delta \) with the maximal overall complexity \( 2^{0.406n} \) therefore stays at \( \frac{1}{2} \).

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.313</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_0 )</td>
<td>0.106</td>
<td>0.168</td>
<td>0.215</td>
<td>0.252</td>
<td>0.283</td>
<td>0.308</td>
<td>0.313</td>
<td>0.320</td>
<td>0.322</td>
<td>0.320</td>
<td>0.312</td>
</tr>
<tr>
<td>( C_1 )</td>
<td>0.106</td>
<td>0.168</td>
<td>0.215</td>
<td>0.252</td>
<td>0.283</td>
<td>0.308</td>
<td>0.313</td>
<td>0.335</td>
<td>0.361</td>
<td>0.385</td>
<td>0.406</td>
</tr>
<tr>
<td>( R_{0,1} )</td>
<td>0.106</td>
<td>0.168</td>
<td>0.215</td>
<td>0.252</td>
<td>0.283</td>
<td>0.308</td>
<td>0.313</td>
<td>0.350</td>
<td>0.400</td>
<td>0.450</td>
<td>0.500</td>
</tr>
<tr>
<td>( 10^5 \tau )</td>
<td>5.664</td>
<td>7.223</td>
<td>6.901</td>
<td>5.381</td>
<td>3.107</td>
<td>0.556</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.4: numerical values for classical ternary representations with \( u = 1 \)

Hence, it makes sense to increase the number of levels of the representation algorithm in the ternary case, i.e. analyze \( u = 2 \), introducing two parameters \( \tau_1 \) and \( \tau_2 \) as done in [BCJ1].

**Corollary 53** (rep2.2). Let \((a, s)\) be an instance of a \((G, n, \delta)\) Random Binary Subset Sum Problem. Let \( u = 2, g = |G|, G = G_0 \times G_1 \times G_2 \) with \( \gcd(|G|, 2) = 1 \), \( w_0 \) be such that \( w_0(1) = \delta, w_0(0) = 1 - \delta \), \( w_1 \) be such that \( w_1(1) = \frac{\delta + n}{2}, w_1(-1) = \frac{n}{2} \), and \( w_1(0) = 1 - \frac{\delta}{2} - \tau_1 \) and let \( w_2 \) be such that \( w_2(1) = \frac{\delta + n + \tau_2}{2}, w_2(-1) = \frac{\tau_1 + \tau_2}{2} \), and \( w_2(0) = 1 - \frac{\delta}{2} - \tau_1 \). Then for any \( 0 \leq \tau_1 \leq 1 \), any \( 0 \leq \tau_2 \leq 2 - \delta - 2 \cdot \tau_1 \), any \( \varepsilon > 0 \), if \( \frac{1}{2} \leq (2^{\varepsilon n})^2 \cdot |G_1 \times G_2|/\left(\left((\delta + n)^2)/(\delta + n + \tau_2)^2\right),\left((\delta + n + \tau_2)^2)/(\tau_1 + \tau_2)^2\right)\right) \) \( \leq 1 \), and \( \frac{1}{2} \leq (2^{\varepsilon n})^2 \cdot |G_2|/\left(\left((\delta + n)^2)/(\delta + n + \tau_2)^2\right),\left((\delta + n + \tau_2)^2)/(\tau_1 + \tau_2)^2\right)\right) \) \( \leq 1 \), the algorithm **CLASSICALRep** with global input \((u, G, w_0, w_1, w_2, n, g, a, \varepsilon)\) and local input \((0, s)\) solves the instance with a probability of \( 1 - O(2^{-\varepsilon n}) \) (over the choice of the input and the coins of the algorithm) in time

\[
\tilde{O} \left( 2^{C_0} + 2^{C_1} + 2^{C_0 n} \cdot 2^{C_1 n} + 2^{6n} \cdot 2^{2\mathcal{H}(w_0) - R_{0,1}} n/|G| \right) 
\]

with \( C_0 = 2 \mathcal{H}(w_2) - R_{0,1} - R_{1,2}, C_1 = \mathcal{H}(w_2) - R_{1,2}, C_2 = \mathcal{H}(w_2)/2, R_{0,1} = \delta + \mathcal{H}_{1,2}(\frac{n}{2}), R_{1,2} = \delta + \mathcal{H}_{1,2}(\tau_1, \frac{n}{2}) \) and \( R_{1,2} = \delta + \mathcal{H}(\frac{n}{2}, \tau_1, \frac{n}{2}), \mathcal{H}(w_1) = \mathcal{H}(\frac{n}{2}, \tau_1, \frac{n}{2}) \) and \( \mathcal{H}(w_2) = \mathcal{H}(\frac{n}{2}, \tau_1, \frac{n}{2}). \)
Once again, this is an extension of the previous $u = 2$ result by introducing parameters $\tau_1$ and $\tau_2$ that denote how many ones and negative ones are added on the respective levels. The weight distributions are valid and are unique for each fixed choice of $\tau_1$ and $\tau_2$. The positive number of representations on the first level is $r_{m_0,m_1}^{n,|G|} = (\delta_n/2) \cdot (1-\delta_n) = \Theta(2^{R_{0,1} n})$ as in the one level case. On the second level also the negative ones have to be represented. This leads to the number $r_{m_0,m_2}^{n,|G|} = (\delta_n/2 \cdot \tau_1/2) \cdot (1-\delta_n/2 \cdot \tau_1/2) \cdot (1-\delta_n/2 \cdot \tau_2/2) = \Theta(2^{R_{0,2} n})$. Thus, since the number of representations is always non-zero and since all requirements for Theorem 43 are fulfilled, we obtain the presented time complexity.

![Figure 4.4](image.png)

**Figure 4.4:** complexities of the classical ternary algorithm for $u = 2$

The algorithm is extended in the same manner as in the previous two level approach and is summarized in Fig. 4.4. Four lists of size roughly $|Z^n[\mathbf{w}_2]|/r_{m_0,m_2}^{n,|G|} \approx 2^{C_2 n}$ are built by creating a polynomial number of lists of size $|Z^n[\mathbf{w}_2]| \approx 2^{C_2 n}$. Out of the four lists, two lists of size $|Z^n[\mathbf{w}_1]|/r_{m_0,m_1}^{n,|G|}$ are constructed that are bounded by $2^{C_1 n}$. In order to construct these lists, $r_{m_0,m_1}^{n,|G|}/r_{m_0,m_2}^{n,|G|}$ list pairs of size $|Z^n[\mathbf{w}_2]|/r_{m_0,m_1}^{n,|G|}$ of vectors in $Z^n[\mathbf{w}_2]$ are constructed that have to be merged to one list that contains all vectors in $Z^n[\mathbf{w}_1]$, which takes quadratic time with a naive approach and leads to a time complexity of roughly $2^{C_1 n}$ for this step. This leads to the following numerical values for the time complexities.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0$</td>
<td>0.101</td>
<td>0.156</td>
<td>0.195</td>
<td>0.224</td>
<td>0.246</td>
<td>0.263</td>
<td>0.277</td>
<td>0.289</td>
<td>0.297</td>
<td>0.304</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.101</td>
<td>0.156</td>
<td>0.195</td>
<td>0.224</td>
<td>0.246</td>
<td>0.263</td>
<td>0.277</td>
<td>0.289</td>
<td>0.297</td>
<td>0.304</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.071</td>
<td>0.122</td>
<td>0.170</td>
<td>0.221</td>
<td>0.246</td>
<td>0.263</td>
<td>0.277</td>
<td>0.289</td>
<td>0.297</td>
<td>0.304</td>
</tr>
<tr>
<td>$R_{0,1}$</td>
<td>0.142</td>
<td>0.244</td>
<td>0.340</td>
<td>0.441</td>
<td>0.491</td>
<td>0.526</td>
<td>0.554</td>
<td>0.577</td>
<td>0.594</td>
<td>0.607</td>
</tr>
<tr>
<td>$R_{1,2}$</td>
<td>0.042</td>
<td>0.089</td>
<td>0.145</td>
<td>0.218</td>
<td>0.246</td>
<td>0.263</td>
<td>0.277</td>
<td>0.289</td>
<td>0.297</td>
<td>0.304</td>
</tr>
<tr>
<td>$100 \cdot \tau_1$</td>
<td>1.017</td>
<td>1.779</td>
<td>2.530</td>
<td>3.475</td>
<td>3.527</td>
<td>3.311</td>
<td>2.967</td>
<td>2.521</td>
<td>1.994</td>
<td>1.413</td>
</tr>
<tr>
<td>$100 \cdot \tau_2$</td>
<td>0.088</td>
<td>0.358</td>
<td>0.880</td>
<td>1.841</td>
<td>1.911</td>
<td>1.778</td>
<td>1.589</td>
<td>1.355</td>
<td>1.085</td>
<td>0.788</td>
</tr>
</tbody>
</table>

**Table 4.5:** numerical values for classical ternary representations with $u = 2$

Once again we can see a threshold in the range $0.20 \leq \delta \leq 0.30$ at which the complexity of the bottom level is already dominated by the top level such that there is no chance for further improvements. However, above that threshold all three complexities become identical. That’s why we also want to consider the three level approach $u = 3$, which leads to better results for larger $\delta$ and is the main result of [BCJII].
Corollary 54 (rep2.3). Let \((a, s)\) be an instance of a \((G, n, \delta)\) Random Binary Subset Sum Problem. Let \(u = 3, g = |G|, G = G_0 \times G_1 \times G_2 \times G_3\) with \(\text{gcd}(|G|, 2) = 1\), \(w_0\) be such that \(w_0(1) = \delta, w_0(0) = 1 - \delta\), \(w_1\) be such that \(w_1(1) = \frac{\delta + 1}{2}, w_1(0) = \frac{1}{2}\) and \(w_1(0) = 1 - \delta - \tau_1\), \(w_2\) be such that \(w_2(1) = \frac{\delta + \tau_1 + \tau_2}{4}, w_2(-1) = \frac{\tau_1 + \tau_2}{2}\) and \(w_2(0) = 1 - \delta - \tau_1\) and let \(w_3\) be such that \(w_3(1) = \frac{\delta + \tau_1 + \tau_2 + \tau_3}{3}, w_3(-1) = \frac{\tau_1 + \tau_2 + \tau_3}{3}\) and \(w_3(0) = 1 - \delta - \frac{\tau_1 + \tau_2 + \tau_3}{3}\). Then for any \(0 \leq \tau_1 \leq 1 - \delta\), any \(0 \leq \tau_2 \leq 2 - \delta - \tau_1\), any \(0 \leq \tau_3 \leq 4 - 2\delta - 2\tau_1 - 2\tau_2\), any \(\varepsilon > 0\), if \(\frac{1}{2} \leq (2^{|G|})^2 \cdot |G_1 \times G_2 \times G_3| = (\delta - \frac{n}{2}) \cdot \left(\frac{(\tau_2 + \tau_3) - n}{(\tau_2 + \tau_3 + n) / 8}, \frac{1}{2}\right)\), and if also \(\frac{1}{2} = (2^{|G|})^2 \cdot |G_3| = \left(\frac{(\tau_2 + \tau_3 + n) / 4, (\tau_2 + \tau_3 + n) / 4, (\tau_2 + \tau_3 + n) / 8, (\tau_2 + \tau_3 + n) / 8}\right) \leq 1\) the algorithm CLASSICALREP with global input \((u, G, w_0, w_1, w_2, w_3, n, g, a, \varepsilon)\) and local input \((0, s)\) solves the instance with a probability of \(1 - O(2^{-\varepsilon n})\) (over the choice of the input and the coins of the algorithm) in time
\[
\tilde{O}\left(2^{cn} + 2^{2cn} + 2^{|G|} \cdot 2^{cn} + 2^{|G|} \cdot 2^{cn} + 2^{6cn} \cdot 2^{|G|} \cdot 2^{6cn} \cdot 2^{|G|} \cdot 2^{6cn} \cdot 2^{|G|} \cdot 2^{6cn} \cdot 2^{|G|} \cdot 2^{6cn}\right),
\]
with \(C_0 = 2\mathcal{H}(w_2) - R_{0, 1} - R_{1, 2}, C_1 = 2\mathcal{H}(w_3) - R_{1, 2} - R_{2, 3}, C_2 = \mathcal{H}(w_3) - R_{2, 3}, C_3 = \mathcal{H}(w_3) / 2, R_{0, 1} = \delta + H_1 - \frac{(\tau_2 + \tau_3)}{8}, R_{1, 2} = \frac{\delta + \tau_1 + H_1 - 4 - (\tau_2 + \tau_3)}{8}, R_{2, 3} = \frac{\delta + 2\tau_1 + 2\tau_2 + \tau_3}{8}, \frac{\tau_2 + \tau_3}{8}, \frac{\tau_2 + \tau_3}{8}\),
where \(\mathcal{H}(w_1) = \mathcal{H}(\delta + \frac{\tau_1 + \tau_2}{8}, \frac{\tau_1 + \tau_2}{8}), \mathcal{H}(w_2) = \mathcal{H}(\frac{\delta + \tau_1 + \tau_2}{8}, \frac{\tau_1 + \tau_2}{8})\) and \(\mathcal{H}(w_3) = \mathcal{H}(\frac{\delta + \tau_1 + \tau_2 + \tau_3}{8}, \frac{\tau_1 + \tau_2 + \tau_3}{8})\).

This is a direct extension of the previous result to \(u = 3\) with an additional number of representations \(r_{w_2, w_3} = (\frac{\tau_2 + \tau_3}{8}, \frac{\tau_2 + \tau_3}{8}, \frac{\tau_2 + \tau_3}{8})\). Therefore, we obtain the presented time complexities.

![Figure 4.5: complexities of the classical ternary representation algorithm for u = 3](image-url)

The algorithm is extended by another level as illustrated in Fig. 4.5. Eight lists of size roughly \(\frac{|Z_n|}{|w_3|} \approx 2^{2cn}\) are built by creating a polynomial number of lists of size \(|Z_n/2|/w_3| \approx 2^{2cn}\). Out of the eight lists, four lists of size \(\frac{|Z_n|}{|w_2|} \approx 2^{2cn}\) are constructed that are bounded by \(2^{2cn}\). In order to construct these lists, \(\frac{|Z_n|}{|w_2|} \approx 2^{2cn}\) list pairs of size \(\frac{|Z_n|}{|w_3|} \approx 2^{2cn}\) of vectors in \(Z_n\) are constructed that have to be merged to one list that contains all vectors in \(Z_n\), which takes quadratic time with a naive approach and leads to a time complexity of roughly \(2^{2cn}\) for this step. The remaining part is as in the previous approach. This leads to the following numerical values for the time complexities.
Table 4.6: numerical values for classical ternary representations with \( u = 3 \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_0 )</td>
<td>0.101</td>
<td>0.156</td>
<td>0.196</td>
<td>0.225</td>
<td>0.247</td>
<td>0.261</td>
<td>0.273</td>
<td>0.281</td>
<td>0.287</td>
<td>0.291</td>
</tr>
<tr>
<td>( C_1 )</td>
<td>0.101</td>
<td>0.156</td>
<td>0.196</td>
<td>0.225</td>
<td>0.247</td>
<td>0.261</td>
<td>0.273</td>
<td>0.281</td>
<td>0.287</td>
<td>0.291</td>
</tr>
<tr>
<td>( C_2 )</td>
<td>0.062</td>
<td>0.102</td>
<td>0.141</td>
<td>0.178</td>
<td>0.247</td>
<td>0.261</td>
<td>0.273</td>
<td>0.281</td>
<td>0.287</td>
<td>0.291</td>
</tr>
<tr>
<td>( C_3 )</td>
<td>0.042</td>
<td>0.071</td>
<td>0.099</td>
<td>0.130</td>
<td>0.195</td>
<td>0.208</td>
<td>0.231</td>
<td>0.241</td>
<td>0.255</td>
<td>0.266</td>
</tr>
<tr>
<td>( R_{0.1} )</td>
<td>0.144</td>
<td>0.243</td>
<td>0.337</td>
<td>0.433</td>
<td>0.628</td>
<td>0.668</td>
<td>0.720</td>
<td>0.747</td>
<td>0.777</td>
<td>0.798</td>
</tr>
<tr>
<td>( R_{1.2} )</td>
<td>0.044</td>
<td>0.088</td>
<td>0.143</td>
<td>0.212</td>
<td>0.389</td>
<td>0.416</td>
<td>0.461</td>
<td>0.482</td>
<td>0.510</td>
<td>0.531</td>
</tr>
<tr>
<td>( R_{2.3} )</td>
<td>0.021</td>
<td>0.040</td>
<td>0.058</td>
<td>0.081</td>
<td>0.142</td>
<td>0.155</td>
<td>0.188</td>
<td>0.202</td>
<td>0.223</td>
<td>0.240</td>
</tr>
</tbody>
</table>

It can be seen that with \( u = 3 \) the complexity of the computation on the bottom is always smaller than the other complexities, which means that further adaptations by increasing the number of levels doesn’t lead to improvements.

One way of improving the time complexity even more could be to increase the degree of freedom in the representations by adding for example 2 or -2 components to the chosen weight distributions. However, these kinds of improvements are out of scope for this thesis and are left for future research. Instead, we want to propose an application of the techniques developed in Sect. 3.4 which allows to filter inconsistent vectors more efficiently.

4.2 Novel Results

4.2.1 Consistent Representations I

In this section, we want to apply the algorithm \textsc{ConsistentRep} to the Random Binary Subset Sum Problem. This algorithm uses the algorithm \textsc{WeightSieve} that allows to solve the Group Weight Match Problem more efficiently. Since there is no such problem in variants of the representation algorithm with \( u = 1 \), we want to apply this improvement to the \( u = 2 \) case of the original Howgrave-Graham and Joux [HJ10] approach. The observation is that in the special case of Corollary 55, we have exactly the weights described in the definition of the zeroAND Problem introduced in Chapter 2. The following corollary was firstly presented in [MO].

**Corollary 55** (con1.2). Let \((a, s)\) be an instance of a \((G, n, \delta)\) Random Binary Subset Sum Problem. Let \( u = 2, g = |G|, G = G_0 \times G_1 \times G_2 \) with \( |G| \geq 3, w_0 \) be such that \( w_0(1) = \delta, w_0(0) = 1 - \delta, w_1 \) be such that \( w_1(1) = \frac{\delta}{2} \) and \( w_1(0) = 1 - \frac{\delta}{2} \) and let \( w_2 \) be such that \( w_2(1) = \frac{\delta}{4} \) and \( w_2(0) = 1 - \frac{\delta}{4} \). Then for any \( \varepsilon > 0 \), if \( \frac{1}{2} \leq (2^n)^2 \cdot |G_1 \times G_2|/(\delta^{-n}/2) \leq 1, \frac{1}{2} \leq (2^\varepsilon n)^2 \cdot |G_2|/|\delta^{-n}/4| \leq 1 \) and \( |Z^n[w_2]|/|G_1 \times G_2| \geq 2^{2\varepsilon n} \), the algorithm \textsc{ConsistentRep} with global input \((u, G, w_0, w_1, w_2, n, g, a, \varepsilon)\) and local input \((0, s)\) solves the instance with a probability of \( 1 - O(2^{-\varepsilon n}) \) (over the choice of the input and the coins of the algorithm) in time

\[
\tilde{O} \left( 2^\varepsilon n + 2^{C_2} n + 2^6 \varepsilon n \cdot 2^C n + 2^2 \varepsilon n \cdot 2^C n + 2^6 \varepsilon n \cdot 2^{(2H(\delta/2)-\delta)n/|G|} \right),
\]
with \( C_0 = \frac{\delta}{2} + \max \left\{ \lambda + \varepsilon, \frac{\delta}{2} \cdot \left(1 - \mathcal{H}(\frac{2n}{\lambda})\right) \right\} \), \( C_1 = \mathcal{H}(\delta/2) - \delta \) and \( C_2 = \mathcal{H}(\delta/4)/2 \) with list length \( \lambda := \frac{1}{n} \cdot \log_2((1 + 2^{\varepsilon n}) \cdot \left(\frac{n}{\delta n/4}\right) / |G_1 \times G_2|) \) and \( \mu \) chosen such that \( \mathcal{H}(\frac{\delta}{2}) - \frac{1}{2} \mathcal{H}(\mu) - \frac{1}{2} \mathcal{H}(\frac{\delta}{2} - \mu) = \lambda \).

The algorithm \textsc{ConsistentRep} has basically the same requirements as the algorithm \textsc{ClassicalRep} that is already analyzed in the setting with \( u = 2 \) and without adding any negative ones. The additional requirement \(|\mathbb{Z}^n[w_2]| / |G_1 \times G_2| \geq 2^{2\varepsilon n}\) is fulfilled for small enough \( \varepsilon \), which can be seen in the numerical results.

The only difference to the previous approach is that the consistency check of finding vectors in \( \mathbb{Z}^n[w_1] \) as a sum of vectors from two lists of vectors in \( \mathbb{Z}^n[w_2] \) is now done in sub-quadratic time. Concretely, we obtain \( \left(\frac{n}{\delta n/2}\right) / \left(\frac{\delta n}{\delta n/4}\right) \) copies of a \textit{Group Weight Match Problem} with two lists of size roughly \( \left(\frac{n}{\delta n/4}\right) / \left(\frac{\delta n}{\delta n/4}\right) \), which are merged with an exponent \( \rho < 2 \) instead of naively with \( \rho = 2 \) as seen before.

The problem that appears is exactly the \textit{zeroAND Problem} that is introduced in the beginning of Chapter 2. In this problem, we receive two lists of vectors in \( \{0,1\}^n \) with Hamming weight \( \delta \cdot n/4 \). The problem is to find all pairs of one vector from the first list and one vector from the second list that have a sum (component-wise over \( \mathbb{Z}_{|G|} \) with \( |G| \geq 3 \) that is also a vector in \( \{0,1\}^n \), i.e. the original vectors don’t have components with a matching one, which would lead to 2-components in the sum. This \textit{zeroAND Problem} is a special case of the \textit{Group Weight Match Problem}, which in turn is a special case of the \textit{Weight Match Problem}, for which we have an algorithm \textsc{WeightSieve}. Let us take a closer look at this special case with given weight distributions \( w_1 \) and \( w_2 \) that lead to a unique joint distribution \( \Gamma \) which is illustrated in Fig. 4.7:

\[
\begin{array}{c|c|c|c|c}
\Gamma & (0,0) & (0,1) & (1,0) & (1,1) \\
\hline
1 - \delta/2 & \delta/4 & \delta/4 & 0 \\
\end{array}
\]

Figure 4.7: joint distribution \( \Gamma \) for the \textit{zeroAND Problem}

This choice characterizes the problem of having two lists of vectors with relative weight \( \delta/4 \) and it has to be found a pair of one vector from the first list and one vector of the second list that doesn’t have any matching ones. That is, the sum of the (0,1) and (1,1) as well as the (1,0) and (1,1) components has to be \( \delta/4 \), whereas the sum of the (0,1) and (1,0) components has to be \( \delta/2 \), which means there are no (1,1) components, which sum would be 2.
In order to compute the optimal time complexity in Theorem 47, we have to find a choice of $h_1$ and $h_2$ that minimizes the time complexity. Due to the fact that $h_1, h_2 \leq 2 \cdot m$, we have that $h_1(j) = 0$ and $h_2(j) = 0$ for all $j \not\in \{0, 1\}$. Numerical optimization suggests that setting $h_1(1) = \mu$ and $h_2(1) = \frac{\delta}{2} - \mu$ for some $0 \leq \mu \leq \frac{\delta}{2}$ is optimal. Notice that this also determines the weights $h_1, h_2$ and the joint distribution $C$ uniquely.

$C$  
<table>
<thead>
<tr>
<th></th>
<th>(0, 0)</th>
<th>(0, 1)</th>
<th>(1, 0)</th>
<th>(1, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 - \delta/2$</td>
<td>$\delta/2 - \mu$</td>
<td>$\mu$</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.8: joint distribution $C$ for the zeroAND Problem

The choice leads to a repetition parameter $\tilde{y} = \frac{\delta}{2} \cdot (1 - \mathcal{H}(\frac{2\mu}{\delta}))$ such that $z_1 = z_2 = \lambda$. The expression for $\tilde{y}$ can be verified by plugging in the concrete joint distributions $\Gamma$ and $C$. Notice that both $z_1$ and $z_2$ are $\mathcal{H}(\delta/4) - \frac{1}{2} \cdot \mathcal{H}(\mu) - \frac{1}{2} \cdot \mathcal{H}(\delta/2 - \mu)$ due to the concrete choice of $h_1$ and $h_2$. Thus, since all requirements for Theorem 47 are fulfilled, we obtain the presented time complexity. In its computation, we make use of the fact that $(\frac{\delta \cdot n}{\delta \cdot n/2}) = \tilde{\Theta}(2^{\delta n})$ and $(\frac{\delta \cdot n/2}{\delta \cdot n/4}) = \tilde{\Theta}(2^{\delta n/4})$.

Ignoring the $\varepsilon$-parts and the part that depends on the size of the group $G$, the complexity exponent is thus $\max\{C_0, C_1, C_2\}$, which leads to the following numerical values. Notice that also the exponent of the number of representations is added, where $R_{0,1} = \delta$ determines the size of $G_1 \times G_2$ and $R_{1,2} = \frac{\delta}{2}$ determines the size of $G_2$ up to $\varepsilon$-factors. There is no solution $\mu$ for any $\delta < \Delta$, where $\Delta \approx 0.313$ is defined such that $\mathcal{H}(\delta/2)/2 = \Delta$. However, in this case the $u = 1$ approach outperforms the $u = 2$ approach anyway.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$\Delta$</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0$</td>
<td>0.313</td>
<td>0.321</td>
<td>0.327</td>
<td>0.329</td>
<td>0.327</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.313</td>
<td>0.320</td>
<td>0.322</td>
<td>0.320</td>
<td>0.312</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.198</td>
<td>0.215</td>
<td>0.235</td>
<td>0.254</td>
<td>0.272</td>
</tr>
<tr>
<td>$R_{0,1}$</td>
<td>0.300</td>
<td>0.350</td>
<td>0.400</td>
<td>0.450</td>
<td>0.500</td>
</tr>
<tr>
<td>$R_{1,2}$</td>
<td>0.150</td>
<td>0.175</td>
<td>0.200</td>
<td>0.225</td>
<td>0.250</td>
</tr>
<tr>
<td>$\mu$</td>
<td>0</td>
<td>0.171</td>
<td>0.186</td>
<td>0.198</td>
<td>0.204</td>
</tr>
<tr>
<td>$\rho$</td>
<td>1.886</td>
<td>1.860</td>
<td>1.829</td>
<td>1.798</td>
<td>1.766</td>
</tr>
</tbody>
</table>

Table 4.7: numerical values for consistent representations with $u = 2$

Notice that the worst case is once again not at $\delta = \frac{1}{2}$, but at $\delta \approx 0.45$. However, the top-level complexity $C_0$ always dominates the overall time complexity. An increase of the number of levels therefore doesn’t help to improve the complexity, since it would only further decrease the complexity $C_2$. Instead, one has to look for ways to decrease the top-level complexity $C_0$ in combination with the faster filtering of inconsistent solutions, which is done in the subsequent section.
4.2 Novel Results

4.2.2 Consistent Representations II

In this section, we want to improve the results by Becker, Coron and Joux [BCJ11] by applying the better filter for inconsistent vectors. Although it would be already applicable to the two level case \( u = 2 \), it doesn’t seem to lead to improved results. That is why we want to present an application to the \( u = 3 \) case directly, which is the main result of [MO].

**Corollary 56 (con2.3).** Let \( (a, s) \) be an instance of a \((G, n, \delta)\) Random Binary Subset Sum Problem. Let \( u = 3 \), \( g = |G| \), \( G = G_0 \times G_1 \times G_2 \times G_3 \) with \( \text{gcd}(|G|, 2) = 1 \) and \( |G| \geq 5 \), \( w_0 \) be such that \( w_0(1) = \delta, w_0(0) = 1 - \delta \), \( w_1 \) be such that \( w_1(1) = \frac{\delta + \tau_1}{2}, w_1(-1) = \frac{\tau_1}{2} \) and \( w_1(0) = 1 - \frac{\delta}{2} - \tau_1 \), \( w_2 \) be such that \( w_2(1) = \frac{\delta + \tau_1 + \tau_2}{4}, w_2(-1) = \frac{\tau_1 + \tau_2}{4} \) and \( w_2(0) = 1 - \frac{\delta}{4} - \frac{\tau_1 + \tau_2}{4} \) and let \( w_3 \) be such that \( w_3(1) = \frac{\delta + \tau_1 + \tau_2 + \tau_3}{8}, w_3(-1) = \frac{\tau_1 + \tau_2 + \tau_3}{8} \) and \( w_3(0) = 1 - \frac{\delta}{8} - \frac{\tau_1 + \tau_2 + \tau_3}{8} \). Then for any \( 0 \leq \tau_1 \leq 1 - \delta \), any \( 0 \leq \tau_2 \leq 2 - \delta - 2 \cdot \tau_1 \), any \( 0 \leq \tau_3 \leq 4 - \delta - 2 \tau_1 - 2 \tau_2 \), any \( \varepsilon > 0 \), any \( 0 \leq \mu,-,\mu_0,\mu_1 \leq 1 \), if

\[
\mathcal{H}(w_2) - \frac{1}{2} \cdot \mathcal{H}(\frac{\delta + \tau_1}{2} \cdot (1 - \mu_1) + \frac{\tau_2}{2} \cdot (1 - \mu_0), \frac{\tau_1}{2} \cdot \mu_1 + \frac{\tau_2}{2} \cdot \mu_0) \\
- \frac{1}{2} \cdot \mathcal{H}(\frac{\delta + \tau_1}{2} \cdot \mu_1 + \frac{\tau_2}{2} \cdot \mu_0, \frac{\tau_1}{2} \cdot (1 - \mu_1) + \frac{\tau_2}{2} \cdot (1 - \mu_0)) = \lambda,
\]

\( \lambda := \frac{1}{n} \log_2((1+2^n)|Z^n[w_2]|/|G_1 \times G_2 \times G_3|) \), if \( \frac{1}{2} \leq (2^n)^2|G_1 \times G_2 \times G_3|/ \left( (\frac{\delta}{n}, \frac{n}{2}) \cdot (\frac{1 - \delta}{n}, \frac{n}{2}) \right) \leq 1 \) and if \( \frac{1}{2} \leq (2^n)^2|G_2 \times G_3|/ \left( (\frac{\delta + \tau_1 + \tau_2}{n}, \frac{n}{2}) \cdot (\frac{1 - \delta - \frac{\tau_1 + \tau_2}{2}}{n}, \frac{n}{2}) \right) \leq 1 \) and if for the last level also

\( \frac{1}{2} \leq (2^n)^2|G_3|/ \left( (\frac{\delta + \tau_1 + \tau_2 + \tau_3}{n}, \frac{n}{2}) \cdot (\frac{1 - \delta - \frac{\tau_1 + \tau_2 + \tau_3}{2}}{n}, \frac{n}{2}) \right) \leq 1 \), \( |Z^n[w_2]|/|G_1 \times G_2 \times G_3| \geq 2^{2^n} \), the algorithm \textsc{ConsistentRep} with global input \((u, G, w_0, w_1, w_2, w_3, n, g, a, \varepsilon)\) and local input \((0, s)\) solves the instance with a probability of \( 1 - \mathcal{O}(2^{-\varepsilon n}) \) (over the choice of the input and the coins of the algorithm) in time

\[
\mathcal{O} \left( 2^{2n} + 2C_2n + 2^{6n} \cdot 2C_1n + 2^{6n} \cdot 2C_0n + 2^{6n} \cdot 2(\mathcal{H}(w_1) - R_0,1)n / |G| \right),
\]

with \( C_0 = R_{0,1} - R_{1,2} + \max \{ \lambda + \varepsilon, \frac{\delta + \tau_1}{2} \cdot (1 - \mathcal{H}(\mu_1)) + \frac{\tau_2}{2} \cdot (1 - \mathcal{H}(\mu_0)) \} \),

\( C_1 = 2\mathcal{H}(w_3) - R_{1,2} - R_{2,3}, C_2 = \mathcal{H}(w_3) - R_{2,3}, C_3 = \mathcal{H}(w_3)/2, R_{0,1} = \delta + H_{1-\delta}(\frac{\tau_2}{2}, \frac{\tau_3}{2}) \) and

\( R_{1,2} = \frac{\delta}{2} + \tau_1 + H_{1-\frac{\delta}{2} - \tau_1}(\frac{\tau_2}{4}, \frac{\tau_3}{4}) \) and \( R_{2,3} = \frac{\delta}{2} + \tau_1 + \tau_2 + H_{1-\frac{\delta}{2} - \tau_1 - \tau_2}(\frac{\tau_3}{8}, \frac{\tau_4}{8}) \), where \( \mathcal{H}(w_1) = \mathcal{H}(\frac{\delta + \tau_1}{2}, \frac{\tau_2}{4}) \), \( \mathcal{H}(w_2) = \mathcal{H}(\frac{\delta + \tau_1 + \tau_2}{4}, \frac{\tau_1 + \tau_2}{4}) \) and \( \mathcal{H}(w_3) = \mathcal{H}(\frac{\delta + \tau_1 + \tau_2 + \tau_3}{8}, \frac{\tau_1 + \tau_2 + \tau_3}{8}) \).

In this main result, we once again apply the improved algorithm \textsc{WeightSieve} for the Group Weight Match Problem to the top level computation of the algorithm \textsc{ConsistentRep}. The improved technique could also be applied to the similar computation on the middle level, which however doesn’t seem to lead to notably better results asymptotically. It is unclear if this is really the case, because the results at this point fully rely on numerics. Even an application to solely the top level already requires to solve a non-linear optimization problem in six variables. Extending the problem to an even higher number of variables usually leads to very unreliable results. It remains as an open problem to try to obtain better techniques by presenting better techniques for solving these kinds of optimization problems.
4. Binary Subset Sum Problem

The new problem on the top level is a generalization of the previously discussed zeroAND Problem. In this problem, we receive two lists of vectors in \{-1, 0, 1\}^n weight distribution \(\mathbf{w}_2\). The problem is to find all pairs of one vector from the first list and one vector from the second list that have a sum (component-wise over \(\mathbb{Z}\)) with \(|G| \geq 5\) that is once again a vector in \{-1, 0, 1\}^n, but now with a weight distribution \(\mathbf{w}_1\). This problem is a special case of the Group Weight Match Problem, which in turn is a special case of the Weight Match Problem, for which we have an algorithm \textsc{WeightSieve}. Let us take a closer look at this special case with given weight distributions \(\mathbf{w}_1\) and \(\mathbf{w}_2\) that lead to a unique joint distribution \(\Gamma\) which is as follows.

<table>
<thead>
<tr>
<th>(\Gamma)</th>
<th>(-1, -1)</th>
<th>(-1, 0)</th>
<th>(-1, 1)</th>
<th>(0, -1)</th>
<th>(0, 0)</th>
<th>(0, 1)</th>
<th>(1, -1)</th>
<th>(1, 0)</th>
<th>(1, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mu)</td>
<td>(\frac{\tau_1}{4})</td>
<td>(\frac{\tau_2}{4})</td>
<td>(\frac{\tau_1}{4})</td>
<td>(1 - \frac{\delta + \tau_1 + \tau_2}{4})</td>
<td>(\frac{\delta + \tau_1}{4})</td>
<td>(\frac{\delta + \tau_1}{4})</td>
<td>(\frac{\delta + \tau_1}{4})</td>
<td>(\frac{\delta + \tau_1}{4})</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4.10: joint distribution \(\Gamma\) for the Generalized zeroAND Problem

In the problem, we have given two lists of vectors with weight distribution \(\mathbf{w}_2\), which means a relative number of \(\frac{\delta + \tau_1 + \tau_2}{4}\) ones and \(\frac{\tau_1}{4}\) negative ones. One has to find a pair of each list such that their sum (over the integers) has a weight distribution \(\mathbf{w}_1\), i.e. \(\frac{\delta + \tau_1}{2}\) ones and \(\frac{\tau_1}{2}\) negative ones. That is, the sum of the (-1, 1), (0, 1) and (1, 1) as well as the (1, -1), (1, 0) and (1, 1) components has to be \(\frac{\delta + \tau_1 + \tau_2}{4}\), whereas the sum of the (-1, -1), (0, -1) and (-1, -1) components as well as the (-1, 1), (-1, 0) and (-1, -1) components has to be \(\frac{\tau_1 + \tau_2}{4}\). Both the (1, 1) component and the (-1, -1) component have to be 0, due to the fact that the sum is ternary.

The sum has to have a weight distribution of \(\mathbf{w}_1\), i.e. a relative number of \(\frac{\delta + \tau_1}{2}\) ones and \(\frac{\tau_1}{2}\) negative ones. This means that the number of (0, 1) and (1, 0) components has to be \(\frac{\delta + \tau_1}{2}\) ones, whereas the number of (0, -1) and (-1, 0) components has to be \(\frac{\tau_1}{2}\). All of this leads to the above unique joint distribution \(\Gamma\).

In order to compute the optimal time complexity in Theorem 47, we have to find a choice of \(\mathbf{h}_1\) and \(\mathbf{h}_2\) that minimizes the time complexity. Due to the fact that \(\mathbf{h}_1, \mathbf{h}_2 \leq 2 \cdot \mathbf{w}\), we have that \(\mathbf{h}_1(j) = 0\) and \(\mathbf{h}_2(j) = 0\) for all \(j \notin \{-1, 0, 1\}\). Numerical optimization suggests that the sum of the (0, 1) and (1, 0) components should be \(\frac{\delta + \tau_1}{2}\), the sum of the (0, -1) and (-1, 0) components should be \(\frac{\tau_1}{2}\) and the sum of the (1, -1) and (-1, 1) components should be \(\frac{\tau_1}{2}\). Notice that this also determines the weights \(\mathbf{h}_1, \mathbf{h}_2\) and the joint distribution \(C \leq 2 \cdot \Gamma\) up to three variables \(0 \leq \mu_1, \mu_0, \mu_1 \leq 1\) uniquely.
4.2 Novel Results

<table>
<thead>
<tr>
<th>C</th>
<th>(-1, 0)</th>
<th>(-1, 1)</th>
<th>(0, -1)</th>
<th>(0, 0)</th>
<th>(0, 1)</th>
<th>(1, -1)</th>
<th>(1, 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\mu_0 \cdot \frac{\tau_1}{2}</td>
<td>\mu_0 \cdot \frac{\tau_2}{2}</td>
<td>(1 - \mu_1) \cdot \frac{\tau_1}{2}</td>
<td>1 - \frac{\tau_1 + \tau_2}{2}</td>
<td>\mu_1 \cdot \frac{\tau_1 + \tau_2}{2}</td>
<td>(1 - \mu_0) \cdot \frac{\tau_2}{2}</td>
<td>(1 - \mu_1) \cdot \frac{\tau_1 + \tau_2}{2}</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.11: joint distribution $C$ for the Generalized zeroAND Problem

The choice leads to a repetition parameter

$$
\tilde{y} = \frac{\delta + \tau_1}{2} \cdot (1 - \mathcal{H}(\mu_1)) + \frac{\tau_1}{2} \cdot (1 - \mathcal{H}(\mu_1)) + \frac{\tau_2}{2} \cdot (1 - \mathcal{H}(\mu_0)).
$$

One can verify that the chosen constraint

$$
\mathcal{H}(w) - \frac{1}{2} \cdot \mathcal{H}(\delta + \tau_1) \cdot (1 - \mu_1) + \frac{\tau_1}{2} \cdot (1 - \mu_0), \frac{\tau_1}{2} \cdot \mu_1 + \frac{\tau_2}{2} \cdot \mu_0
$$

leads to the fact that $z_1 = z_2 = \lambda$ with

$$
z_1 := \mathcal{H}(w) - \frac{1}{2} \mathcal{H}(h_1) - \frac{1}{2} \mathcal{H}(2 \cdot w - h_1)
$$

and

$$
z_2 := \mathcal{H}(w) - \frac{1}{2} \mathcal{H}(h_2) - \frac{1}{2} \mathcal{H}(2 \cdot w - h_2),
$$

with $h_1$ and $h_2$ that can be derived from $C$. The expression for $\tilde{y}$ can be verified by plugging in the concrete joint distributions $\Gamma$ and $C$.

Ignoring the $\varepsilon$-parts and the part that depends on the size of the group $\mathcal{G}$, the complexity exponent is thus $\max\{C_0, C_1, C_2, C_3\}$, which leads to the following numerical values.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0$</td>
<td>0.101</td>
<td>0.156</td>
<td>0.195</td>
<td>0.224</td>
<td>0.246</td>
<td>0.260</td>
<td>0.271</td>
<td>0.278</td>
<td>0.283</td>
<td>0.287</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.101</td>
<td>0.156</td>
<td>0.195</td>
<td>0.224</td>
<td>0.246</td>
<td>0.260</td>
<td>0.270</td>
<td>0.277</td>
<td>0.283</td>
<td>0.287</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.062</td>
<td>0.104</td>
<td>0.140</td>
<td>0.178</td>
<td>0.228</td>
<td>0.260</td>
<td>0.271</td>
<td>0.278</td>
<td>0.283</td>
<td>0.287</td>
</tr>
<tr>
<td>$C_3$</td>
<td>0.040</td>
<td>0.071</td>
<td>0.099</td>
<td>0.131</td>
<td>0.174</td>
<td>0.216</td>
<td>0.258</td>
<td>0.269</td>
<td>0.278</td>
<td>0.283</td>
</tr>
<tr>
<td>$R_{0,1}$</td>
<td>0.141</td>
<td>0.244</td>
<td>0.335</td>
<td>0.436</td>
<td>0.566</td>
<td>0.676</td>
<td>0.748</td>
<td>0.772</td>
<td>0.793</td>
<td>0.804</td>
</tr>
<tr>
<td>$R_{1,2}$</td>
<td>0.041</td>
<td>0.089</td>
<td>0.142</td>
<td>0.216</td>
<td>0.329</td>
<td>0.431</td>
<td>0.515</td>
<td>0.538</td>
<td>0.556</td>
<td>0.566</td>
</tr>
<tr>
<td>$R_{2,3}$</td>
<td>0.019</td>
<td>0.038</td>
<td>0.057</td>
<td>0.084</td>
<td>0.119</td>
<td>0.171</td>
<td>0.245</td>
<td>0.259</td>
<td>0.273</td>
<td>0.279</td>
</tr>
</tbody>
</table>

| $100 \cdot \tau_1$ | 1.107 | 1.771 | 2.446 | 3.370 | 5.021 | 6.453 | 7.178 | 6.751 | 6.290 | 5.455 |
| $100 \cdot \tau_2$ | 0.800 | 0.363 | 0.817 | 1.806 | 3.927 | 6.073 | 8.091 | 8.224 | 8.218 | 7.989 |
| $100 \cdot \tau_3$ | 0.008 | 0.048 | 0.074 | 0.245 | 0.377 | 1.258 | 3.581 | 3.765 | 3.984 | 3.930 |

| $\mu_1$ | 0.414 | 0.372 | 0.362 | 0.343 | 0.331 | 0.361 | 0.278 | 0.277 | 0.275 | 0.272 |
| $\mu_0$ | 0.379 | 0.355 | 0.347 | 0.278 | 0.276 | 0.264 | 0.157 | 0.146 | 0.145 | 0.141 |
| $\mu_1$ | 0.477 | 0.463 | 0.444 | 0.439 | 0.432 | 0.380 | 0.336 | 0.325 | 0.322 | 0.317 |
| $\rho$  | 1.946 | 1.875 | 1.810 | 1.724 | 1.627 | 1.574 | 1.542 | 1.542 | 1.539 | 1.537 |

Table 4.8: numerical values for consistent ternary representations with $u = 3$

Due to the fact that the bottom level complexity $C_3$ is already dominated by the remaining complexities, an increasing of the number of levels doesn’t help to improve the result.
4.3 Results in Special Groups

In this section, we want to present algorithms for the case of a special group structure. We want to assume that this group $G$ has the property that all elements in $G$ have an order of 2. One important candidate is $G = \mathbb{F}_2^n$ for $m \in \mathbb{N}$, with exclusive-or addition. These results are important for the Decoding application in Chapter 6 and are based on [BJMM12].

### 4.3.1 Algorithms

Once we have this special group structure, we can make use of the fact that a group element added to itself is always the neutral element. This means that the \{0,1\}-vectors in the Subset Sum Problem can be added modulo 2 such that not only ones can be represented in two ways as 1 + 0 or 0 + 1, but also zeros can be represented in two ways as 0 + 0 or 1 + 1. With one level of representations, this leads to the following result from [BJMM12].

**Corollary 57 (sg1).** Let $(a,s)$ be an instance of a $(G,n,\delta)$ Random Binary Subset Sum Problem with $G$ being a group such that each group element has an element order of at most 2. Let $u = 1$, $g = 2$, $G = G_0 \times G_1$, $w_0$ be such that $w_0(1) = \delta, w_0(0) = 1 - \delta$ and let $w_1$ be such that $w_1(1) = \frac{\delta + \tau}{2}$ and $w_1(0) = 1 - \frac{\delta + \tau}{2}$. Then for any $0 \leq \tau \leq 2 - 2\delta$ and any $\varepsilon > 0$, if \( \frac{1}{2} \leq (2^m)^2 \cdot |G| / \left( \left( \frac{\delta}{\delta-n/2} \right) \cdot \left( \frac{1-\delta}{\tau-n/2} \right) \right) \leq 1 \), the algorithm CLASSICALRep with global input $(u,G,w_0,w_1,n,g,a,\varepsilon)$ and local input $(0,s)$ solves the instance with a probability of $1 - \Theta(2^{-\varepsilon n})$ (over the choice of the input and the coins of the algorithm) in time

$$\tilde{O} \left( 2^{2\varepsilon n} + 2^{C_1n} + 2^{6\varepsilon n} \cdot 2^{6\varepsilon n} \cdot 2^{6\varepsilon n} \cdot 2^{(2H(\frac{\delta+\tau}{2})-R_{0,1})n} / |G| \right),$$

with $C_0 = H(\frac{\delta+\tau}{2}) - R_{0,1}$ and $C_1 = H(\frac{\delta+\tau}{2}) / 2$, where $R_{0,1} = \delta + H(1-\delta)(\frac{\tau}{2})$. We choose $u = 1$ such that we have only one level of representations. The weight $w_0$ corresponds to the given Random Subset Sum Problem, whereas the weight $w_1$ introduces additional 1 components that represent the zero. Notice that both weight distributions are $|G|$-valid, because all non-zero parts lie in an interval of size 2 and $|G| \geq 2$. The number of representations is $r_{w_0,w_1}^2 = \frac{\delta}{\delta-n/2} \cdot \frac{(1-\delta)}{\tau-n/2}$, since out of the $\delta \cdot n$ ones of a vector of weight $w_0$, one is allowed to choose $\delta \cdot n/2$ ones for the first representing vector, whereas out of the $(1-\delta) \cdot n$ zeros of the vector, we can choose $\frac{\tau}{2} \cdot n$ ones or the remaining number of zeros. In other words, for each $x \in \mathbb{Z}^n[w_0]$ there are $\frac{\delta}{\delta-n/2} \cdot \frac{(1-\delta)}{\tau-n/2}$ possible $x_1 \in \mathbb{Z}^n[w_1]$ such that there is a $x_2 \in \mathbb{Z}^n[w_1]$ with $x_1 + x_2 = x \text{ mod } 2$. This directly implies that the number of representations is at least 1. Thus, since all requirements for Theorem 43 are fulfilled, we obtain the presented time complexity. In its computation, we make use of the fact that $\left( \frac{\delta}{\delta-n/2} \right) \cdot \left( \frac{(1-\delta)}{\tau-n/2} \right) = \Theta(2^{R_{0,1} \cdot n})$.

![Diagram](image)

**Figure 4.12:** complexities of the classical special group representation algorithm for $u = 1$.
4.3 Results in Special Groups

Compared to the classical result with one level, we therefore add a certain number of 1’s on the lower levels. This allows to have a smaller final list of size $2^{C_0 n}$ at the cost of a slightly larger construction time of $2^{C_1 n}$. The optimal choice of the parameter $\tau$ should therefore be a value that leads to $C_0 = C_1$, whenever this is possible. In the following table we see improvements of this kind for small values of $\delta$, with a $\delta$ that leads to $C_0 = C_1$. However, for large values of $\delta$ the bottom complexity $C_1$ is already dominating, which means that $\tau = 0$ is the optimal choice and we get no improvement compared to the previous result. The $\delta$ with the maximal overall complexity $2^{0.406n}$ therefore stays at $\frac{1}{2}$.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.313</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0$</td>
<td>0.099</td>
<td>0.158</td>
<td>0.204</td>
<td>0.242</td>
<td>0.276</td>
<td>0.306</td>
<td>0.313</td>
<td>0.320</td>
<td>0.322</td>
<td>0.320</td>
<td>0.312</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.099</td>
<td>0.158</td>
<td>0.204</td>
<td>0.242</td>
<td>0.276</td>
<td>0.306</td>
<td>0.313</td>
<td>0.335</td>
<td>0.361</td>
<td>0.385</td>
<td>0.406</td>
</tr>
<tr>
<td>$R_{0,1}$</td>
<td>0.099</td>
<td>0.158</td>
<td>0.204</td>
<td>0.242</td>
<td>0.276</td>
<td>0.306</td>
<td>0.313</td>
<td>0.350</td>
<td>0.400</td>
<td>0.450</td>
<td>0.500</td>
</tr>
<tr>
<td>$100\tau$</td>
<td>1.087</td>
<td>1.345</td>
<td>1.251</td>
<td>0.950</td>
<td>0.533</td>
<td>0.092</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.9: numerical values for classical special group representations with $u = 1$

Therefore, it makes sense to increase the number of levels of the representation algorithm in the ternary case, i.e. analyze $u = 2$, introducing two parameters $\tau_1$ and $\tau_2$ as done in [BJMM12].

**Corollary 58 (sg2).** Let $(a, s)$ be an instance of a $(G, n, \delta)$ Random Binary Subset Sum Problem with $G$ being a group such that each group element has an element order of at most 2. Let $u = 2$, $g = 2$, $G = G_0 \times G_1 \times G_2$, $w_0$ be such that $w_0(1) = \delta, w_0(0) = 1 - \delta$, $w_1$ be such that $w_1(1) = \frac{\delta + \tau_1}{2}$ and $w_1(0) = 1 - \frac{\delta + \tau_1}{2}$ and let $w_2$ be such that $w_2(1) = \frac{\delta + \tau_1 + \tau_2}{4}$ and $w_2(0) = 1 - \frac{\delta + \tau_1 + \tau_2}{4}$. Then for any $0 \leq \tau_1 \leq 2 - 2\delta$, any $0 \leq \tau_2 \leq 4 - 2\delta - 2\tau_1$, any $\varepsilon > 0$, if $\frac{1}{2} \leq (2^{\varepsilon n})^2 \cdot |G_1 \times G_2| / \left( \left( \frac{\delta - \varepsilon n}{\delta - n/2} \right) \cdot \left( \frac{1 - \varepsilon n}{\tau_1/2} \right) \right) \leq 1$ and $\frac{1}{2} \leq (2^{\varepsilon n})^2 \cdot |G_2| / \left( \left( \frac{\delta + \tau_1}{\delta + \tau_1 + \tau_2/4} \right) \cdot \left( \frac{1 - (\delta + \tau_1/2)^n}{\tau_2 n/4} \right) \right) \leq 1$, the algorithm CLASSICALREP with global input $(u, G, w_0, w_1, w_2, n, g, a, \varepsilon)$ and local input $(0, s)$ solves the instance with a probability of $1 - O(2^{-\varepsilon n})$ (over the choice of the input and the coins of the algorithm) in time

$$\tilde{O}\left( 2^{\varepsilon n} + 2^{C_2 n} + 2^{6\varepsilon n} \cdot 2^{C_1 n} + 2^{6\varepsilon n} \cdot 2^{C_0 n} + 2^{6\varepsilon n} \cdot 2^{(2H(\frac{\delta + \tau_1}{4}) - R_{0,1})n} / |G| \right),$$

with $C_0 = 2H(\frac{\delta + \tau_1 + \tau_2}{4}) - R_{0,1} - R_{1,2}$, $C_1 = H(\frac{\delta + \tau_1 + \tau_2}{4}) - R_{1,2}$, $C_2 = H(\frac{\delta + \tau_1 + \tau_2}{4}) / 2$, $R_{0,1} = \delta + H_{1 - \delta}(\frac{\tau_2}{4})$ and $R_{1,2} = \frac{\delta + \tau_1}{2} + H_{1 - (\delta + \tau_1)}(\frac{\tau_2}{4})$.

Once again, this is an extension of the previous $u = 2$ result by introducing parameters $\tau_1$ and $\tau_2$ that denote how many ones are added on the respective levels. The weight distributions are valid due to the same reasons as before and are unique for each fixed choice of $\tau_1$ and $\tau_2$. The positive number of representations on the first level is $r^{n/2}_{0,1,0,1} = \left( \frac{\delta - n/2}{\delta - n/2} \right) \cdot \left( \frac{1 - \varepsilon n}{\tau_1/4} \right) = \tilde{O}(2^{R_{0,1}n})$ as in the one level case. On the second level we obtain analogously the number $r^{n/2}_{1,0,1,1} = \left( \frac{(\delta + \tau_1)/n}{(\delta + \tau_1)/n} \right) \cdot \left( \frac{(1 - (\delta + \tau_1/2)^n)}{\tau_2 n/4} \right) = \tilde{O}(2^{R_{1,2}n})$. Thus, since the number of representations is always non-zero and since all requirements for Theorem 43 are fulfilled, we obtain the presented time complexity.
Figure 4.13: complexities of the classical special group representation algorithm for \( u = 2 \)

The algorithm is built analogously to the previous two level algorithms and leads to the following numerical values for the time complexities.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_0 )</td>
<td>0.087</td>
<td>0.128</td>
<td>0.158</td>
<td>0.183</td>
<td>0.203</td>
<td>0.222</td>
<td>0.238</td>
<td>0.254</td>
<td>0.268</td>
<td>0.281</td>
</tr>
<tr>
<td>( C_1 )</td>
<td>0.087</td>
<td>0.128</td>
<td>0.158</td>
<td>0.183</td>
<td>0.203</td>
<td>0.222</td>
<td>0.238</td>
<td>0.254</td>
<td>0.268</td>
<td>0.281</td>
</tr>
<tr>
<td>( C_2 )</td>
<td>0.079</td>
<td>0.128</td>
<td>0.158</td>
<td>0.183</td>
<td>0.203</td>
<td>0.222</td>
<td>0.238</td>
<td>0.254</td>
<td>0.268</td>
<td>0.281</td>
</tr>
<tr>
<td>( R_{0,1} )</td>
<td>0.158</td>
<td>0.256</td>
<td>0.316</td>
<td>0.365</td>
<td>0.406</td>
<td>0.443</td>
<td>0.476</td>
<td>0.507</td>
<td>0.535</td>
<td>0.562</td>
</tr>
<tr>
<td>( R_{1,2} )</td>
<td>0.071</td>
<td>0.128</td>
<td>0.158</td>
<td>0.183</td>
<td>0.203</td>
<td>0.222</td>
<td>0.238</td>
<td>0.254</td>
<td>0.268</td>
<td>0.281</td>
</tr>
<tr>
<td>100 ( \cdot \tau_1 )</td>
<td>2.882</td>
<td>4.643</td>
<td>5.106</td>
<td>5.145</td>
<td>4.903</td>
<td>4.461</td>
<td>3.876</td>
<td>3.193</td>
<td>2.448</td>
<td>1.679</td>
</tr>
<tr>
<td>100 ( \cdot \tau_2 )</td>
<td>1.298</td>
<td>2.528</td>
<td>2.690</td>
<td>2.658</td>
<td>2.505</td>
<td>2.271</td>
<td>1.981</td>
<td>1.650</td>
<td>1.291</td>
<td>0.914</td>
</tr>
</tbody>
</table>

Table 4.10: numerical values for classical special group representations with \( u = 2 \)

Once again we can see a threshold in the range \( 0 \leq \delta \leq 0.10 \) at which the complexity of the bottom level is already dominated by the top level such that there is no chance for further improvements. However, above that threshold all three complexities become identical. That's why we also want to consider the three level approach \( u = 3 \), which leads to better results for larger \( \delta \).

**Corollary 59 (sg3).** Let \((a, s)\) be an instance of a \((G, n, \delta)\) Random Binary Subset Sum Problem with \( G \) being a group such that each group element has an element order of at most 2. Let \( u = 3 \), \( g = 2 \), \( G = G_0 \times G_1 \times G_2 \times G_3 \), \( w_0 \) be such that \( w_0(1) = \delta, w_0(0) = 1 - \delta \), \( \omega_1 \) be such that \( \omega_1(1) = \delta + 2 \tau_2 \) and \( \omega_1(0) = 1 - \frac{\delta + 2 \tau_2}{2} \), \( \omega_2 \) be such that \( \omega_2(1) = \frac{\delta + 2 \tau_2 + \tau}{4} \) and \( \omega_2(0) = 1 - \frac{\delta + 2 \tau_2 + \tau}{4} \) and let \( \omega_3 \) be such that \( \omega_3(1) = \frac{\delta + 2 \tau_2 + \tau_3 + \tau_4}{2} \) and \( \omega_3(0) = 1 - \frac{\delta + 2 \tau_2 + \tau_3 + \tau_4}{2} \). Then for any \( 0 \leq \tau_1 \leq 2 - 2 \delta \), any \( 0 \leq \tau_2 \leq 4 - 2 \delta - 2 \tau_1 \), any \( 0 \leq \tau_3 \leq 8 - 2 \delta - 2 \tau_1 - 2 \tau_2 \), any \( \epsilon > 0 \), if \( \frac{1}{2} \leq (2^{\epsilon n})^{2} \cdot |G_1 \times G_2 \times G_3|/(\delta_{n/2} \cdot (1-\delta_{n/2})^{2n}) \leq 1 \), \( \frac{1}{2} \leq (2^{\epsilon n})^{2} \cdot |G_2 \times G_3|/(\delta_{n/4} \cdot (1-\delta_{n/4})^{8n}) \leq 1 \), \( \frac{1}{2} \leq (2^{\epsilon n})^{2} \cdot |G_3|/(\delta_{n/8} \cdot (1-\delta_{n/8})^{16n}) \leq 1 \), the algorithm **ClassicalRep** with global input \((G, w_0, \omega_1, \omega_2, \omega_3, n, g, a, \varepsilon)\) and local input \((0, s)\) solves the instance with a probability of \( 1 - \mathcal{O}(2^{-\varepsilon n}) \) (over the choice of the input and the coins of the algorithm) in time

\[
\tilde{\mathcal{O}} \left( 2^{2n} + 2^{C_2 n} + 2^{C_1 n} + 2^{C_0 n} + 2^{C_1 n} + 2^{C_0 n} + 2^{C_1 n} + 2^{C_0 n} \right),
\]

with \( C_0 = 2H(\frac{\delta + 2 \tau_2 + \tau}{4}) - R_{0,1} - R_{1,2}, C_1 = 2H(\frac{\delta + 2 \tau_2 + \tau_3 + \tau_4}{4}) - R_{1,2} - R_{2,3}, C_2 = H(\frac{\delta + 2 \tau_2 + \tau_3 + \tau_4}{4}) - R_{2,3}, C_3 = H(\frac{\delta + 2 \tau_2 + \tau_3 + \tau_4}{4})/2, R_{0,1} = \delta + H(\frac{\tau_1}{2}), R_{1,2} = \frac{\delta + \tau_1}{2} + H(\frac{\tau_1}{2}), \) and \( R_{2,3} = \frac{\delta + \tau_1}{4} + H(\frac{\tau_1}{4}). \)
4.3 Results in Special Groups

This is a direct extension of the previous result to \( u = 3 \) with an additional number of representations \( r_{w_2,w_3}^{n,2} = \left( \frac{\delta + v_1 + r_2}{\delta + v_1 + r_2} \right)^n \cdot \left( 1 - \frac{\delta + v_1 + r_2}{r_3} \right)^n \). Therefore, we obtain the presented time complexities.

- **time on each level:**
  \[
  \left( \frac{\left| Z^n[w_3] \right|}{\left| Z^n[w_2] \right|} \right)^2 \cdot \frac{r_{w_2,w_3}^{n,2}}{r_{w_3}^{n,2}} \approx 2 C_{0n}
  \]
- **list sizes on each level:**
  \[
  \left| Z^n[w_3] \right| / r_{w_2,w_3}^{n,2} \approx 2 C_{1n}
  \]

Figure 4.14: complexities of the classical special group representation algorithm for \( u = 3 \)

This leads to the following numerical values for the time complexities.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.35</th>
<th>0.40</th>
<th>0.45</th>
<th>0.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_0 )</td>
<td>0.087</td>
<td>0.124</td>
<td>0.144</td>
<td>0.158</td>
<td>0.170</td>
<td>0.180</td>
<td>0.189</td>
<td>0.197</td>
<td>0.205</td>
<td>0.212</td>
</tr>
<tr>
<td>( C_1 )</td>
<td>0.087</td>
<td>0.124</td>
<td>0.144</td>
<td>0.158</td>
<td>0.170</td>
<td>0.180</td>
<td>0.189</td>
<td>0.197</td>
<td>0.205</td>
<td>0.212</td>
</tr>
<tr>
<td>( C_2 )</td>
<td>0.067</td>
<td>0.124</td>
<td>0.144</td>
<td>0.158</td>
<td>0.170</td>
<td>0.180</td>
<td>0.189</td>
<td>0.197</td>
<td>0.205</td>
<td>0.212</td>
</tr>
<tr>
<td>( C_3 )</td>
<td>0.046</td>
<td>0.101</td>
<td>0.144</td>
<td>0.158</td>
<td>0.170</td>
<td>0.180</td>
<td>0.189</td>
<td>0.197</td>
<td>0.205</td>
<td>0.212</td>
</tr>
<tr>
<td>( R_{0,1} )</td>
<td>0.158</td>
<td>0.324</td>
<td>0.423</td>
<td>0.468</td>
<td>0.505</td>
<td>0.537</td>
<td>0.565</td>
<td>0.590</td>
<td>0.613</td>
<td>0.635</td>
</tr>
<tr>
<td>( R_{1,2} )</td>
<td>0.071</td>
<td>0.201</td>
<td>0.287</td>
<td>0.315</td>
<td>0.339</td>
<td>0.359</td>
<td>0.377</td>
<td>0.394</td>
<td>0.409</td>
<td>0.424</td>
</tr>
<tr>
<td>( R_{2,3} )</td>
<td>0.025</td>
<td>0.078</td>
<td>0.144</td>
<td>0.158</td>
<td>0.170</td>
<td>0.180</td>
<td>0.189</td>
<td>0.197</td>
<td>0.205</td>
<td>0.212</td>
</tr>
<tr>
<td>( 100 \cdot \tau_1 )</td>
<td>2.880</td>
<td>7.458</td>
<td>9.895</td>
<td>9.860</td>
<td>9.446</td>
<td>8.766</td>
<td>7.891</td>
<td>6.875</td>
<td>5.760</td>
<td>4.584</td>
</tr>
<tr>
<td>( 100 \cdot \tau_2 )</td>
<td>1.297</td>
<td>6.226</td>
<td>9.870</td>
<td>10.21</td>
<td>10.33</td>
<td>10.32</td>
<td>10.20</td>
<td>10.01</td>
<td>9.750</td>
<td>9.442</td>
</tr>
<tr>
<td>( 100 \cdot \tau_3 )</td>
<td>0.108</td>
<td>1.373</td>
<td>5.277</td>
<td>5.378</td>
<td>5.387</td>
<td>5.338</td>
<td>5.247</td>
<td>5.124</td>
<td>4.979</td>
<td>4.812</td>
</tr>
</tbody>
</table>

Table 4.11: numerical values for classical special group representations with \( u = 3 \)

Although it can be seen that for values \( \delta > 0.1 \) it would make sense to increase the number of levels even more, we want to stop at this point. The reason is that we want to use these kinds of algorithms for values \( \delta \leq 0.1 \) for the Decoding Problem of Chapter 6, at which point the \( u = 2 \) or \( u = 3 \) algorithms are already optimal, i.e. an increasing of \( u \) wouldn’t help in this range.

Notice that it is theoretically possible to combine the consistency results of Sect. 4.2 with the results for \( g = 2 \) for the cases \( u = 2 \) or \( u = 3 \). However, unfortunately this doesn’t seem lead to notably better results. The reason is that the \( \tau \)-values are always chosen such that there are as good as no inconsistent vectors.

This can for example be verified in Table 4.10 of the two level case at the case of \( \delta = 0.3 \). In this case, we have \( H(w_1) - R_{0,1} = H(\frac{\delta + \tau_1}{2}) - \delta - H_{1-\delta}(\frac{\tau_1}{2}) \approx 0.221 \), which more or less equals \( C_0 \approx 0.222 \), the time to compute this list. Similar results can be seen in Table 4.11 for example at \( \delta = 0.05 \) on the top level. Here we have \( H(w_1) - R_{0,1} \approx 0.082 \), which more or less equals
$C_0 \approx 0.087$. We obtain similar results on the second level. Possible improvements are therefore at most minor, especially in the application to the algorithms in Chapter 6. Although the gap in the latter case seems to be large, the issue is that the consistency algorithms usually don’t achieve this optimal bound, which is also the case here. It is left as an open problem to study possible improvements in more detail. This question is also discussed at the end of Chapter 6 for the Decoding application.
Chapter 5

Knapsack Problem

In the Knapsack Problem – which can also be seen as a Subset Sum Problem over the integers – one is given a vector \( \mathbf{a} \in \mathbb{Z}^n \) of integers and an integer target \( s \in \mathbb{Z} \). The problem is to find a vector \( \mathbf{x} \in \{0, 1\}^n \) such that \( \sum_{i=1}^n a_i x_i = s \). As one of the 21 problems of Karp [Kar72], its decision version was shown to be NP-complete. The first knapsack based cryptosystem was proposed by Merkle and Hellman [MH78] and subsequently broken by Shamir [Sha82]. An important quantity to define the hardness of a Knapsack Problem is the density, which links \( n \) and the size of the \( a_i \). Impagliazzo and Naor [IN96] showed that hard instances are at densities close to 1. In the case of a small density, the problem can be solved in polynomial time by an approach of Lagarias and Odlyzko [LO85] and an improvement by Coster et al. [CJL+92] using lattice based techniques. There are also efficient algorithms for high densities, e.g. [JG94].

We want to formally define the problem in the hard setting of density close to 1 with uniformly random \( \mathbf{a} \) and \( s \) as follows. The problem is defined over a group \( \mathbb{Z}_M \) for an \( M \in \mathbb{N} \), i.e. the addition is modulo \( M \). The hard instances are assumed at \( M \approx 2^n \), so we assume \( M \) to be close to that value.

**Definition 60** (Knapsack Problem). Let \( M, n \in \mathbb{N} \) with \( 2^n - 1 \leq M \leq 2^{n+1} \) and \( 0 \leq \delta \leq \frac{1}{2} \). In the \((M, n, \delta)\) Knapsack Problem we are given \( \mathbf{a} \in \mathbb{Z}_M^n \) and \( s \in \mathbb{Z}_M \) chosen uniformly at random. The problem is to output a list that contains any fixed subset \( D \subseteq [n] \) with \( |D| = \delta \cdot n \) such that \( \sum_{i \in D} a_i = s \mod M \), or an empty list if no such \( D \) exists.

The problem can therefore be seen as a special case of the Random Binary Subset Sum Problem with \( G = \mathbb{Z}_M \). In the following, we want to present complexity results for this special problem. Notice that some of the algorithms that are considered in the previous chapter only work in groups with a certain structure, i.e. for values of \( M \) with a special factorization. Therefore, we want to present a technique to adapt the group, whenever it doesn’t have the desired structure.

The heuristic argument is that if we have an \( M' \) with a desired factorization such that \( M' = 2^{-\tau n} \cdot M \) for a constant \( \tau > 0 \), we can solve the \((M, n, \delta)\) problem by solving a \((M', n, \delta)\) problem. Indeed, we can reduce each of the \( a_i \in \mathbb{Z}_M \) modulo \( M' \) to obtain \( a_i' \in \mathbb{Z}_{M'} \). Moreover, we have that \( \sum_{i \in D} a_i = s \mod M \). Therefore, there is a \( k \in \{0, \ldots, n - 1\} \) with \( \sum_{i \in D} a_i = s + kM \) over the integers and we can define \( s' \in \mathbb{Z}_{M'} \) such that \( s' = s + kM \mod M' \). This directly implies that \( \sum_{i \in D} a_i' = s' \mod M' \). However, the values \( a_i' \) and \( s' \) are clearly not uniform in \( \mathbb{Z}_{M'} \) and we have a probability between \( \frac{1}{M'} \) and \( \frac{1}{M'} \) for each group element. The argument is that our algorithms are robust against this exponentially small bias. That is, we slightly
modify the algorithms that require a group structure \( G = G_0 \times G_1 \) to solve the problem in the group \( G_1 = \mathbb{Z}_{M'} \) (with close-to-uniform parameters) up to the point we have two final lists. In the final step, where the two lists have to be merged, we simply check for matches modulo \( M \). During this process, we need to guess the value of \( k \), which can be done by simply running the algorithm \( n \) times.

Therefore, we can assume in the following that the \( M \) can be arbitrarily split into \( u + 1 \) pairwise different prime factors \( M_0, \ldots, M_u \) with sizes that are required by the algorithm such that the group is \( G = \mathbb{Z}_{M_0} \times \mathbb{Z}_{M_1} \times \ldots \times \mathbb{Z}_{M_u} \).

5.1 Results

Up to 2010, the best known result was the original result by Horowitz and Sahni [HS74] which follows directly from Corollary 49. Notice that \(|G| \geq 2^{n-1} \) implies \( 2^{H(\delta)n}/|G| \leq 2 \).

**Corollary 61 (HS).** Let \((a, s)\) be an instance of a \((M, n, \delta)\) Knapsack Problem. Then the instance can be solved with overwhelming probability in time \( 2^{0.5n} \).

Howgrave-Graham and Joux [HJ10] introduced the representation technique. The following result follows directly from Corollaries 50 and 51, due to the fact that \((2^n)2^{H(\delta/2) - \delta - 1}\) is bounded by \(2^{0.340n}\).

**Corollary 62 (HJ).** Let \((a, s)\) be an instance of a \((M, n, \delta)\) Knapsack Problem. Then the instance can be solved with overwhelming probability in time \( 2^{0.340n} \).

This result was then improved by Becker, Coron and Joux [BCJ11] by introducing -1 components. The following result follows directly from Corollaries 52, 53 and 54, due to the fact that it is numerically verified that \(2^{H(\frac{\delta+n}{2}, \frac{n}{2}) - R_{0,1} - 1 \leq 0.291}\) for various \(\delta\) and the corresponding optimized \(\tau_1\).

**Corollary 63 (BCJ).** Let \((a, s)\) be an instance of a \((M, n, \delta)\) Knapsack Problem. Then the instance can be solved with overwhelming probability in time \(2^{0.291n}\).

The first novel result of [MO] improves upon [HJ10] by finding inconsistent vectors more efficiently by solving the zeroAND Problem. The following result follows directly from Corollary 55 due to the fact that \(2^{H(\delta/2) - \delta - 1 \leq 0.329}\) for all \(\delta\).

**Corollary 64 (MO.1).** Let \((a, s)\) be an instance of a \((M, n, \delta)\) Knapsack Problem. Then the instance can be solved with overwhelming probability in time \(2^{0.329n}\).

The final novel result of [MO] improves upon [BCJ11] by solving a generalized version of the zeroAND Problem. The following result follows directly from Corollary 56 due to the fact that \(2^{H(\frac{\delta+n}{2}, \frac{n}{2}) - R_{0,1} - 1 \leq 0.287}\) for all \(\delta\) and the corresponding \(\tau_1\).

**Corollary 65 (MO.2).** Let \((a, s)\) be an instance of a \((M, n, \delta)\) Knapsack Problem. Then the instance can be solved with overwhelming probability in time \(2^{0.287n}\).

An open question that remains is if these results can be further improved. One possibility could be to introduce new components like 2 or -2 in order to increase the number of representations on the top level. Another possibility could be to try to improve the underlying inconsistency algorithm for the zeroAND Problem and its generalizations.
Chapter 6

Decoding Problem

The Decoding Problem for linear codes is a well-known NP-hard problem in complexity theory. The hardness of decoding of random codes is used as an assumption in the cryptographic encryption scheme by McEliece [McE78]. The idea of this public key scheme is to start with an efficiently decodable Goppa [Gop70] code \( C \). With the help of a random transformation, this code is transformed into a different, publicly known code \( C' \) with the same parameters, which is designed to be indistinguishable from a random code. In the encryption function, a message is encoded with \( C' \) and a uniformly random error is added to the corresponding code word such that the erroneous word is within the error correction distance. Due to the fact that the transformation preserves the weight of the error, the decryption function can undo the random transformation and decode the received word efficiently with the decoding algorithm for \( C \). Therefore, the security of this schemes relies on the fact that there is no efficient algorithm for decoding the random code \( C' \).

The Learning Parity with Noise Problem (LPN) and the Learning with Errors Problem (LWE) can also be seen as Decoding Problems. In both cases, we receive a certain number of erroneous samples that can be used to build a random code. Both the LPN [HB01] [KPC+11] and LWE [Reg05] [Reg09] [BV11] [GPV08] [PW08] problem can be used to build cryptographic schemes with special functionalities. Whereas LWE is built over \( \mathbb{F}_q \), LPN is defined over binary fields, which we want to concentrate on in this thesis. However, the general framework that is developed in the previous chapters easily allows to extend the results to arbitrary fields, as it was for example done in [Pet10].

A binary linear code \( C \) is a \( k \)-dimensional subspace of \( \mathbb{F}_2^n \). Thus, there is a full rank parity check matrix \( H \in \mathbb{F}_2^{(n-k)\times n} \) with \( Hc = 0 \) iff. \( c \in C \). The distance \( d \) of \( C \) is defined by the minimal Hamming distance of two different codewords in \( C \). We denote \( C \) a random code, if \( H \) is chosen by picking each column vector independently and uniformly at random from \( \mathbb{F}_2^{n-k} \) such that \( H \) has full rank. The latter happens with a probability of at least \( \prod_{i=1}^{n-k} (1 - 2^{-i}) \geq \frac{1}{4} \).

For the sake of simplicity, we therefore define our Decoding Problem for matrices \( H \) that are chosen as described, but without the restriction on the rank.

The task of this chapter is to develop algorithms for decoding this kind of described codes. Notice that for every erroneous codeword \( \mathbf{x} = \mathbf{c} + \mathbf{e} \) with error vector \( \mathbf{e} \), we obtain \( \mathbf{Hx} = \mathbf{He} \) by linearity. We call \( \mathbf{s} := \mathbf{Hx} \in \mathbb{F}_2^{n-k} \) the syndrome of a message \( \mathbf{x} \). In order to decode \( \mathbf{x} \), it suffices to find a low weight vector \( \mathbf{e} \) such that \( \mathbf{He} = \mathbf{s} \). Once \( \mathbf{e} \) is found, we can simply recover \( \mathbf{c} \) from \( \mathbf{x} \). Therefore, we assume that the input of our algorithms is \( \mathbf{s} \) and the problem is to find a low-weight \( \mathbf{e} \).
In the case of **Full Distance Decoding** (FDD), we receive an arbitrary point \( x \in \mathbb{F}_2^n \) and want to decode to a closest codeword in the Hamming metric. We want to argue that there is always a codeword within (roughly) Hamming distance \( d \). Therefore, we observe that the syndrome equation \( H e = s \) is solvable as long as the search space for \( e \) roughly equals \( 2^{n-k} \). Hence, for weight-\( d \) vectors \( e \in \mathbb{F}_2^n \) we obtain \( \binom{n}{d} \approx 2^{H(\frac{d}{n})} \approx 2^{n-k} \), where \( H(\cdot) \) is the binary entropy function. This implies \( H(\frac{\delta}{n}) \approx 1 - \frac{k}{n} \), a relation that is known as the Gilbert-Varshamov bound. Moreover, it is well-known that random codes asymptotically reach the Gilbert-Varshamov bound [Sud]. This implies that for every \( x \) we can always expect to find a closest codeword within distance \( d \). In the other case of **Half Distance Decoding** (HDD), we obtain the promise that the error vector is within the error correction distance, i.e. \( \text{wt}(e) \leq \lfloor \frac{d-1}{2} \rfloor \), which is for example the case in cryptographic settings, where an error is added artificially by e.g. the encryption of a message. This leads to the following problem definition with \( H^{-1}(\cdot) \) being the inverse of the binary entropy function mapping to values between 0 and \( \frac{1}{2} \).

**Definition 66** (Decoding Problem). Let \( n, k \in \mathbb{N} \), define \( \kappa := k/n \) and \( \delta := H^{-1}(1 - \kappa) \). In the \((n, k)\) Decoding Problem we are given a matrix \( H \in \mathbb{F}_2^{(n-k) \times n} \) with column vectors that are chosen uniformly and independently at random and a vector \( s \in \mathbb{F}_2^{n-k} \) that is chosen uniformly at random. The FDD Problem is to output a vector \( e \in \mathbb{F}_2^n \) with relative Hamming weight at most \( \delta \) such that \( H e = s \), or \( \perp \) if no such \( e \) exists. The BDD Problem is to output a vector \( e \in \mathbb{F}_2^n \) with relative Hamming weight at most \( \delta/2 \) such that \( H e = s \), or \( \perp \) if no such \( e \) exists.

The running time \( T(n, k) \) of algorithms for the Decoding Problem therefore depends on two parameters \( n \) and \( k \), whereas an upper bound on the weight of the error vector \( e \) is defined by one of the two cases FDD or BDD. Since the actual weight of \( e \) is unknown, we can simply check all possible weights \( 0 \leq \omega \leq \delta n \) in the FDD case or \( 0 \leq \omega \leq \delta n/2 \) in the BDD case, losing only a polynomial factor, since all our algorithms attain their maximum running time for their maximal weight \( \omega = \delta n \), respectively \( \omega = \delta n/2 \), which is verified numerically. When we speak of worst-case running time, we maximize \( T(n, k) \) for all \( k \) where the maximum is obtained for code rates \( \kappa := \frac{k}{n} \) near \( \frac{1}{2} \). Usually, it suffices to compare worst-case complexities since all known decoding algorithms with running time \( T, T' \) and worst-case complexity \( T(n) < T'(n) \) satisfy \( T(n, k) < T'(n, k) \) for all \( k \). The complexities for the different algorithms is displayed for certain rates \( \kappa \). Similarly to the previous chapters, we rely on numerical optimization with Mathematica 10.2 [Res15] to obtain the results.

Naively, one can solve both problems by simply enumerating all weight-\( \omega \) vectors \( e \in \mathbb{F}_2^n \) in time \( \binom{n}{\omega} \). However, it was already noticed by Prange [Pra62] that the search space for \( e \) can be considerably lowered by applying simple linear algebra. Prange’s algorithm consists of an enumeration step with exponential complexity and some Gaussian elimination step with only polynomial complexity. The worst-case complexity of Prange’s algorithm is \( 2^{0.121n} \) in the full distance decoding case and \( 2^{0.0576n} \) with half distance decoding.

Stern [Ste88] uses a meet-in-the-middle technique to improve upon this result. The technique is realized in two steps. In the first step, a part of the error is enumerated in two lists that are merged into one list with all candidates that agree on a small part of the syndrome. In the second part, all these candidates are checked if they also agree on the full syndrome. This technique leads to a running time improvement to \( 2^{0.117n} \) (FDD) and \( 2^{0.0557n} \) (HDD).

In [MO15], the results of Chapter 2 are used to build an algorithm for the Decoding Problem that has to solve a Nearest Neighbor Problem as a sub-routine. Using this algorithm in the full distance decoding setting directly leads to an improved decoding algorithm for random binary
6.1 Classical Algorithms

linear codes with complexity $2^{0.114n}$, whereas in the half distance decoding setting, a complexity of $2^{0.0550n}$ is obtained.

The classical result by Stern was also improved by using the *representation* technique introduced in [HJ10] and explained in Chapter 3 in full generality and in Chapter 4 for the binary case. In a straightforward application, May, Meurer and Thomae [MMTI] obtain a result of $2^{0.114n}$ (FDD) and $2^{0.0550n}$ (HDD). This is improved by Becker, Joux, May and Meurer [BJMM12] making use of the special structure of $F_2^n$ and obtaining a result of $2^{0.102n}$ (FDD) and $2^{0.0494n}$ (HDD).

The techniques of Chapter 2 as well as Chapters 3 and 4 are finally combined in [MO15] to obtain the currently best known decoding algorithm with time complexity $2^{0.097n}$ with full distance decoding and $2^{0.0473n}$ in half distance decoding. The mentioned algorithms are summarized in Fig. 6.1 and Fig. 6.2.

Figure 6.1: History of Information Set Decoding: full distance decoding (FDD)

Figure 6.2: History of Information Set Decoding: half distance decoding (HDD)

### 6.1 Classical Algorithms

#### 6.1.1 Prange’s Information Set Decoding

Prange [Pra62] shows that the use of linear algebra provides a significant speedup compared to a simple *brute-force* technique. Notice that we can simply reorder the positions of the error vector $e$ with $wt(e) = \omega$ by permuting the columns of $H$. For some column permutation $\pi$ let $Q \in F_2^{(n-k) \times (n-k)}$ denote the quadratic matrix at the right hand side of $\pi(H) = (\cdot || Q)$. Assume that $Q$ has full rank, which happens with constant probability. Define $\bar{s} = Q^{-1} \cdot s$ and $\bar{H} = Q^{-1} \cdot \pi(H) = (\cdot || I)$, of which the left hand part doesn’t matter and the right hand part is an identity matrix $I$, as illustrated in Fig. 6.3. Let $\pi(e) = (x||y)$ with $x \in F_2^k$ and $y \in F_2^{n-k}$ be the permuted error vector and assume that $x = 0^k$. In this case, we call the first $k$ error-free coordinates an *information set*. Having an information set, we can rewrite our equation as

$$\bar{H}\pi(e) = \bar{H}x + y = \bar{s},$$

where $wt(x) = 0$ and $wt(y) = \omega$.

Since $x$ is the zero vector, we can simplify to $y = \bar{s}$. Thus we only have to check whether $\bar{s}$ has the correct weight $wt(\bar{s}) = \omega$. 
Let us analyze this idea more rigorously, since this basic concept reappears in each of the algorithms for the problem we want to study. In an abstract view, we want the columns of $H$ to be redistributed such that the corresponding error vector is in a certain set $\mathcal{S}$ and such that the corresponding matrix $Q$ of the permuted matrix $\pi(H)$ is invertible.

**Lemma 67** (Redistribution Lemma). Let $n, k, \omega \in \mathbb{N}$ with $0 \leq k, \omega \leq n$, $\mathcal{S} \subseteq \mathbb{F}_2^n$ be a non-empty set with vectors of Hamming weight $\omega$ and $H \in \mathbb{F}_2^{(n-k)\times n}$ be a fixed vector with Hamming weight $\omega$ and $\mathbf{x} \in \mathbb{F}_2^n$ be chosen by choosing each column uniformly and independently at random. Then with a probability of at least $1 - 2^{-\varepsilon n}$, after choosing $4\varepsilon n \cdot \binom{n}{\omega}/|\mathcal{S}|$ permutations $\pi: [n] \rightarrow [n]$ uniformly and independently at random, there is at least one such that $\pi(\mathbf{x}) \in \mathcal{S}$ and $Q \in \mathbb{F}_2^{(n-k)\times(n-k)}$ is invertible, with $Q$ being the right hand part of the permuted $H$, i.e. $\pi(H) = (\cdot || Q)$.

**Proof.** Fix a $\mathbf{y} \in \mathcal{S}$. Then there are $\omega! \cdot (n - \omega)!$ permutations $\pi$ such that $\mathbf{y} = \pi(\mathbf{x})$. Thus, the transformation from $\mathbf{x}$ to $\mathbf{y}$ has a probability of $\omega! \cdot (n - \omega)!/n! = 1/\binom{n}{\omega}$. Hence, the probability to obtain one of the elements in $\mathcal{S}$ is $|\mathcal{S}|/\binom{n}{\omega}$. Notice that any fixed subset of $n - k$ columns of $H$ is invertible with a probability of at least $\frac{1}{4}$ over the choice of $H$. Setting $p := 4 \cdot |\mathcal{S}|/\binom{n}{\omega}$, this means that after choosing $\varepsilon n/p$ such $\pi$ independently and uniformly at random, the probability that none of the $\pi$ lead to both $\pi(\mathbf{x}) \in \mathcal{S}$ and an invertible $Q$ is less than $(1 - p)^{\varepsilon n/p} \leq 2^{-\varepsilon n}$. \hfill \square

This description of the Information Set Decoding technique together with the Redistribution Lemma leads to the following algorithm.

**Algorithm 11** PRANGEDECODING

1: **Input:** $n, k, H \in \mathbb{F}_2^{(n-k)\times n}, s \in \mathbb{F}_2^n, \varepsilon > 0$
2: **Output:** $e \in \mathbb{F}_2^n$ with $He = s$ and $\text{wt}(e)$ upper bounded as defined in the FDD/HDD case
3: for $\omega \leftarrow 0 \ldots \mathcal{H}^{-1}(1 - \frac{k}{n}) \cdot n \text{ do }$ (FDD) or $\omega \leftarrow 0 \ldots \frac{1}{2} \cdot \mathcal{H}^{-1}(1 - \frac{k}{n}) \cdot n \text{ in the HDD case}$
4: for $4\varepsilon n \cdot \binom{n}{\omega}/\binom{n-k}{\omega}$ times do
5: $\pi \leftarrow$ uniformly random permutation on $[n]$
6: $(\cdot || Q) \leftarrow \pi(H)$ (column permutation) with $Q \in \mathbb{F}_2^{(n-k)\times(n-k)}$
7: if $Q$ is invertible then
8: $s \leftarrow Q^{-1}s$
9: if $\text{wt}(s) = \omega$ then
10: return $\pi^{-1}(0^k || s)$ \hfill \triangleright invert permutation on vector $s$ with $k$ preceding zeros
11: return $\perp$

Figure 6.3: Information Set Decoding
Lemma \( \pi \) at there is a solution and any solution that is output is correct.

\( \kappa \)

\[ \kappa \in S \subseteq \mathbb{F}_2^n \] be the set of all vectors of Hamming weight \( \omega \) that are zero on the first \( k \) components and have the whole weight on the remaining \( n-k \) components. Since \( |S| = \binom{n}{\omega}^k \), the Redistribution Lemma guarantees with a probability of at least \( 1 - 2^{-\varepsilon n} \) that after \( 4\varepsilon n \cdot (\omega^*)/(n-k) \) chosen permutations there is at least one such that \( \pi(e^*) \in S \) and \( Q \) with \( \pi(H) = (\cdot||Q) \) is invertible with \( Q \in \mathbb{F}_2^{(n-k)\times(n-k)} \). Fix that permutation \( \pi \), which implies that \( \pi(H)\pi(e^*) = s \). Due to the fact that \( \pi(e^*) = (0^k||y) \) for some \( y \in \mathbb{F}_2^{n-k} \), we therefore obtain \( s = (\cdot||Q)(0^k||y) = Qy \) and therefore \( y = s = Q^{-1}s \), which is also illustrated in Fig. 6.3. Conversely, each \( s = Q^{-1}s \) with \( \text{wt}(s) = \omega \) leads to a correct solution, since \( H\pi^{-1}(0^k||s) = s \). Therefore, we have shown that there is a solution and any solution that is output is correct.

Proof. Let \( \kappa := k/n, \delta := \mathcal{H}^{-1}(1 - \kappa) \) and let \( \omega_{\text{max}} := \delta n \) in the FDD setting or \( \omega_{\text{max}} := \delta n/2 \) in the HDD setting. Assume there is a solution \( e^* \) with \( \omega^* := \text{wt}(e^*) \leq \omega_{\text{max}} \). Notice that otherwise the algorithm correctly outputs \( \perp \).

Let us continue by showing the correctness in the case such an \( e^* \) exists. Let \( S \subseteq \mathbb{F}_2^n \) be the set of all vectors of Hamming weight \( \omega \) that are zero on the first \( k \) components and have the whole weight on the remaining \( n-k \) components. Since \( |S| = \binom{n}{\omega}^k \), the Redistribution Lemma guarantees with a probability of at least \( 1 - 2^{-\varepsilon n} \) that after \( 4\varepsilon n \cdot (\omega^*)/(n-k) \) chosen permutations there is at least one such that \( \pi(e^*) \in S \) and \( Q \) with \( \pi(H) = (\cdot||Q) \) is invertible with \( Q \in \mathbb{F}_2^{(n-k)\times(n-k)} \). Fix that permutation \( \pi \), which implies that \( \pi(H)\pi(e^*) = s \). Due to the fact that \( \pi(e^*) = (0^k||y) \) for some \( y \in \mathbb{F}_2^{n-k} \), we therefore obtain \( s = (\cdot||Q)(0^k||y) = Qy \) and therefore \( y = s = Q^{-1}s \), which is also illustrated in Fig. 6.3. Conversely, each \( s = Q^{-1}s \) with \( \text{wt}(s) = \omega \) leads to a correct solution, since \( H\pi^{-1}(0^k||s) = s \). Therefore, we have shown that there is a solution and any solution that is output is correct.

For the computation of the time complexities, we assume that the time complexity is maximal at \( \omega^* = \omega_{\text{max}} \), which can be verified numerically. This makes the time complexity of the for-loops \( \tilde{O}(\omega_{\text{max}}/(n-k)) \), whereas the computation in each step of the for-loops is polynomial in \( n \). In the FDD case (Table 6.1), we obtain the following time complexities \( 2^{\varphi n} \) for several rates \( \kappa \), with a maximal value at \( \kappa \approx 0.454 \). At this value, the complexity becomes \( 2^{0.121n} \).

<table>
<thead>
<tr>
<th>( \kappa )</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.454</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varphi )</td>
<td>0.059</td>
<td>0.092</td>
<td>0.111</td>
<td>0.120</td>
<td>0.121</td>
<td>0.120</td>
<td>0.113</td>
<td>0.098</td>
<td>0.076</td>
<td>0.045</td>
</tr>
</tbody>
</table>

Table 6.1: time complexity of \texttt{PrangeDecoding} for FDD

In Table 6.2, we see similar results for the HDD setting with a maximum of \( 2^{0.0576n} \) at \( \kappa \approx 0.469 \).

<table>
<thead>
<tr>
<th>( \kappa )</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.469</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varphi )</td>
<td>0.0263</td>
<td>0.0421</td>
<td>0.0518</td>
<td>0.0566</td>
<td>0.0576</td>
<td>0.0574</td>
<td>0.0543</td>
<td>0.0475</td>
<td>0.0369</td>
<td>0.0219</td>
</tr>
</tbody>
</table>

Table 6.2: time complexity of \texttt{PrangeDecoding} for HDD

Notice that all complexity in Prange’s algorithm is moved to the initial permutation \( \pi \) of \( H \)'s columns, whereas the remaining step has polynomial complexity. To improve upon the running time, it is reasonable to lower the restriction that the information set has no 1-entries in \( x \), which was done in the work of Lee and Brickell [LBSS]. Assume that the information set carries exactly \( p \) 1-positions. Then we enumerate over all \( \binom{k}{p} \) possible \( x \in \mathbb{F}_2^k \) with weight \( p \). Therefore, we can test whether \( \text{wt}(y) = \text{wt}(s - Hx) = \omega - p \). On the downside, this trade-off between lowering the complexity for finding a good \( \pi \) and enumerating weight-\( p \) vectors does not pay.
off. Namely, asymptotically (in $n$) the trade-off achieves its optimum for $p = 0$. However, this idea leads to the following algorithm by Stern.

### 6.1.2 Stern’s Decoding

Instead of enumerating full weight-$p$ vectors, one can also enumerate two lists of weight-$\frac{k}{2}$ vectors. This *meet-in-the-middle* approach allows to reduce the enumeration time from $\binom{k}{p}$ to $\binom{k/2}{p/2}$ and was introduced by Stern [Ste88] as an extension of both the idea by Prange [Pra62] as well as Lee and Brickell [LB88]. Concretely, the idea is to transfer the problem into a *Subset Sum Problem* in the group $G = \mathbb{F}_2^{n-k}$. However, due to the $y$-part that appears in the decoding equation, one isn’t able to obtain an *exact* equation only in the two parts of size $k/2$ and weight $p/2$, which would be necessary to obtain a *Subset Sum Problem*.

Therefore, the idea of Stern and an extension of Finiasz and Sendrier [FS09] is to introduce an additional parameter $\ell \in \mathbb{N}$ and permute the vector $e$ such that $\pi(e) = (x|y)$ splits into an $x \in \mathbb{F}_2^{k+\ell}$ and a $y \in \mathbb{F}_2^{n-k-\ell}$ with $\text{wt}(x) = p$ and $\text{wt}(y) = \omega - p$ for some properly chosen weight $p$ and a certain number of columns $\ell$.

Now fix some column permutation $\pi$ and let $Q \in \mathbb{F}_2^{(n-k)\times(n-k)}$ denote the quadratic matrix at the right hand side of $\pi(H) = (\cdot||Q)$. Assume that $Q$ has full rank, which happens with constant probability. Then there is a transformation matrix $T \in \mathbb{F}_2^{(n-k)\times(n-k)}$ that performs a partial Gaussian elimination on $\pi(H)$ and creates a matrix $\bar{H} = T \cdot \pi(H)$ such that $\bar{H} = (\bar{H}^{(k)} \ 0)$ with uniformly random matrices $A \in \mathbb{F}_2^{k\times(k+\ell)}$ and $B \in \mathbb{F}_2^{(n-k-\ell)\times(k+\ell)}$ as well as a zero matrix $O \in \mathbb{F}_2^{(n-k-\ell)\times(n-k-\ell)}$ and the identity matrix $I \in \mathbb{F}_2^{(n-k-\ell)\times(n-k-\ell)}$. Additionally, denote $s = Ts$.

Having introduced the parameter $\ell$, one is now able to identify a *Subset Sum Problem* in the group $G = \mathbb{F}_2^k$ as follows. Notice that we have $\bar{H} \cdot (x|y) = s$ on the whole $n-k$ components. However, on the top $\ell$ components, the vector $y$ is multiplied with the zero part of $H$. Denoting $[\cdot]^\ell$ the first $\ell$ rows of a row-vector, we therefore have $Ax = [\bar{s}]^\ell$ independently of $y$, with known $A$ and $s$. Splitting $A$ into $k + \ell$ individual group elements $a_1, \ldots, a_{k+\ell} \in G$, the problem of finding $x$ therefore becomes a *Random Subset Sum Problem* $\sum_{i=1}^{k+\ell} a_i x_i = [\bar{s}]^\ell$, i.e. $f_a(x) = [\bar{s}]^\ell$.

In Stern’s algorithm, this problem is solved with the *meet-in-the-middle* algorithm MitmList introduced in Chapter 3. The vector $x$ is therefore split into two equally large halves of half the weight. Then two lists, enumerating both halves individually, are created and merged to obtain a list of solutions. Clearly, this simple technique can also be replaced by the more efficient techniques developed in Chapter 3. We discuss the more efficient methods for solving this problem that were introduced in [MM11] and [BJ12] in Sect. 6.1.3 and Sect. 6.1.4.

Once we have a set $L_{\text{out}}$ of solutions $x$ for the equation on the first $\ell$ rows, we need to verify whether it matches on the whole $s$. This is done by checking whether $\text{wt}(Bx + [\bar{s}]_{n-k-\ell}) = \omega - p$, where $[\bar{s}]_{n-k-\ell}$ are the last $n - k - \ell$ components of $\bar{s}$. The permutation $\pi$ is finally inverted to obtain the error $e$ that can be built from $x$ and $y$.

The algorithm *ExactDecoding* is used several times with different instantiations of the so-called algorithm ColumnMatch (as denoted in [MM11]). This algorithm has essentially to solve the *Random Subset Sum Problem* in the group $\mathbb{F}_2^\ell$. We denote this framework the *exact* framework, because we obtain a *Random Subset Sum Problem* with exactly known target (on a certain part of size $\ell$). In Sect. 6.2, we compare this framework to an *approximate* one, in which only an approximation of the target is known. This enables us to use the methods developed in Chapter 2 to improve the algorithms.
### Algorithm 12 ExactDecoding

1. **Input:** $n, k, \mathbf{H} \in \mathbb{F}_2^{(n-k) \times n}, \mathbf{s} \in \mathbb{F}_2^n, \varepsilon > 0$
2. **Output:** $\mathbf{e} \in \mathbb{F}_2^n$ with $\mathbf{He} = \mathbf{s}$ and $\text{wt}(\mathbf{e})$ upper bounded as defined in the FDD/HDD case
3. \begin{algorithmic}[1]
   \State $\omega \leftarrow 0 \ldots \mathcal{H}^{-1}(1 - \frac{k}{n}) \cdot n$
   \For{$\omega \leftarrow 0 \ldots \mathcal{H}^{-1}(1 - \frac{k}{n}) \cdot n$} \Comment{(FDD) or $\omega \leftarrow 0 \ldots \frac{1}{2} \cdot \mathcal{H}^{-1}(1 - \frac{k}{n}) \cdot n$ in the HDD case}
   \State Choose optimized parameters $p$ and $\ell$ that minimize the time complexity.
   \State for $4 \in n \cdot \binom{n}{\omega}/ \left( \left( \frac{k+\ell}{p} \right) \cdot \frac{n-k-\ell}{\omega-p} \right)$ times do
   \State $\pi \leftarrow$ uniformly random permutation on $[n]$
   \State $(\cdot||Q) \leftarrow \pi(\mathbf{H})$ (column permutation) with $Q \in \mathbb{F}_2^{(n-k) \times (n-k)}$
   \If{$Q$ is invertible then}
   \State Choose invertible $\mathbf{T}$ such that $\mathbf{H} = \mathbf{T} \cdot \pi(\mathbf{H}) = (\begin{smallmatrix} A & 0 \end{smallmatrix})$
   \State $\mathcal{L}_{\text{out}} \leftarrow \text{COLUMNMATCH}(k, \ell, p, \mathbf{A}, [\bar{s}]^{\ell}, \varepsilon)$
   \For{all $\mathbf{x} \in \mathcal{L}_{\text{out}}$} do
   \State if $\text{wt}(\mathbf{x} || \mathbf{Bx} + [\bar{s}]_{n-k-\ell}) = \omega$ then
   \State \quad $\pi^{-1}(\mathbf{x} || \mathbf{Bx} + [\bar{s}]_{n-k-\ell})$ \Comment{invert the permutation}
   \EndIf
   \EndFor
   \EndIf
   \EndFor
   \EndFor
   \EndFor
   \Return $\bot$
\end{algorithmic}

### Theorem 69 (ExactDecoding). Let $(\mathbf{H}, \mathbf{s})$ be an instance of a $(n, k)$ Decoding Problem and let $\mathbf{w}$ be a weight distribution with $\mathbf{w}(1) = p/(k+\ell)$ and $\mathbf{w}(0) = 1 - \mathbf{w}(1)$. Let ColumnMatch be an algorithm that solves a $(\mathbb{F}_2^d, \mathbf{w}, k+\ell)$ Random Subset Sum Problem instance $(\mathbf{A}, [s]^{\ell})$ in time $T(k, \ell, p, \varepsilon)$ with a probability of $1 - \mathcal{O}(2^{-\varepsilon n})$ (over both the random choice of the input and the coins of the algorithm). Then for any $\varepsilon > 0$ and any $p, \ell \in \mathbb{N}$, the algorithm ExactDecoding solves the decoding instance with a probability of $1 - \mathcal{O}(2^{-\varepsilon n})$ (over both the uniform choice of the input and the coins of the algorithm) in time

$$\tilde{O} \left( (2^n)^{\mathcal{H}(\bar{\omega}) - \mathcal{H}_{\kappa + \zeta}(\rho) - \mathcal{H}_{1 - \kappa - \zeta}(\bar{\omega} - \rho)} \cdot T(k, \ell, p, \varepsilon) \right)$$

with $\kappa := k/n$, $\delta := \mathcal{H}^{-1}(1 - \kappa)$, $\rho := p/n$, $\zeta := \ell/n$ and such that $\bar{\omega} := \delta$ in the FDD and $\bar{\omega} := \delta/2$ in the HDD setting.

**Proof.** Let $\omega_{\text{max}} := \delta n$ in the FDD setting and $\omega_{\text{max}} := \delta n/2$ in the HDD setting. Assume there is a solution $\mathbf{e}^*$ with $\omega^* := \text{wt}(\mathbf{e}^*) \leq \omega_{\text{max}}$ of the Decoding Problem. Notice that otherwise the algorithm correctly outputs $\bot$.

Now fix a choice of $p$ and $\ell$. Let us continue by showing the correctness in the case such an $\mathbf{e}^*$ exists. Let $\mathcal{S} \subseteq \mathbb{F}_2^n$ be the set of all vectors of Hamming weight $\omega^*$ that are $p$ on the first $k+\ell$ components and have the remaining weight of $\omega - p$ on the remaining $n - k - \ell$ components. Since $|\mathcal{S}| = \binom{k+\ell}{p} \cdot \binom{n-k-\ell}{\omega_p}$, the Redistribution Lemma guarantees with a probability of at least $1 - 2^{-\varepsilon n}$ that after $4 \in n \cdot \binom{n}{\omega}/|\mathcal{S}|$ chosen permutations there is at least one such that $\pi(\mathbf{e}^*) \in \mathcal{S}$ and invertible $Q \in \mathbb{F}_2^{(n-k) \times (n-k)}$ with $\pi(\mathbf{H}) = (\cdot||Q)$. Fix that permutation $\pi$, which implies that $\pi(\mathbf{H})\pi(\mathbf{e}^*) = \mathbf{s}$.

Then it is possible to apply a partial Gaussian elimination that corresponds to multiplying by an invertible matrix $\mathbf{T}$ from the left to obtain $\mathbf{H} = \mathbf{T} \cdot \pi(\mathbf{H})$ and $\bar{s} = \mathbf{Ts}$ such that $\mathbf{H} \cdot \pi(\mathbf{e}^*) = \bar{s}$ with $\mathbf{H} = (\begin{smallmatrix} A & 0 \end{smallmatrix})$. Due to the fact that $\pi(\mathbf{e}^*) = (\mathbf{x}^* || \mathbf{y}^*)$ for some $\mathbf{x}^* \in \mathbb{F}_2^{k+\ell}$ and some $\mathbf{y}^* \in \mathbb{F}_2^{n-k-\ell}$, we therefore have the two equations $\mathbf{A}\mathbf{x}^* = [\bar{s}]^{\ell}$ and $\mathbf{Bx}^* + \mathbf{y}^* = [\bar{s}]_{n-k-\ell}$.

In the equation $\mathbf{A}\mathbf{x}^* = [\bar{s}]^{\ell}$ the only unknown is $\mathbf{x}^*$. Moreover, this is exactly a Random Subset Sum Problem in the group $\mathbb{F}_2^l$ with $n_{\text{RSSP}} := k + \ell$. In the statement of the theorem, we
assume that this problem can be solved in time $T(k, \ell, p, \varepsilon)$ with a probability of $1 - O(2^{-\varepsilon n})$. Thus we obtain a list $\mathcal{L}_{\text{out}}$ that contains $x^*$ with the given probability.

For the given $x^*$ we can therefore simply compute the corresponding $y^* = Bx^* + [\bar{s}]_{n-k-\ell}$. It passes the final weight test, because the overall weight of $\pi(e^*)$ is indeed $\omega^*$. Notice that for any $x$ that passes the test in line 13, the corresponding weight is $\omega^*$ and therefore the output – which is obtained after inverting the permutation – is correct.

Finally, the complexity is simply the complexity of the repetitions in the for loop times the time complexity for solving the Random Subset Sum Problem.

The result of [FS09] (originally by [Ste88]) follows directly from the presented algorithm, if the sub-algorithm COLUMNMATCH is instantiated with the algorithm MitmList. The algorithm subdivides the matrix as illustrated in Fig. 6.4. The idea is to enumerate two lists with the help of $A$ as shown in Corollary 49, which are merged to one list, using the exact information on the syndrome (independently of $y$) on $\ell$ components. The elements of this candidate list are then verified with the help of the matrix $B$, by checking if they match the syndrome on all components.

\[ x \begin{cases} \ell \{ & A \\ n-k-\ell \} & O \\ k+\ell \} & B \\ n-k-\ell \} & I \end{cases} = \begin{cases} \bar{s} \end{cases} \]

**complexity:**
- repetitions (outer loop): $2^{\varepsilon n}$
- list enumeration with $A$: $2^{C_0 n}$
- merge with $A$ and verify with $B$: $2^{C_1 n}$

This leads to the following corollary.

**Corollary 70 (SternDecoding).** Let $(H, s)$ be an instance of a $(n, k)$ Decoding Problem. Then for any $\varepsilon > 0$, any $p, \ell \in \mathbb{N}$, the algorithm ExactDecoding instantiated with the algorithm MitmList($F_2^\ell, w, k+\ell, A, [\bar{s}]^\ell, \varepsilon \cdot \frac{n}{k+\ell}$) with a weight distribution $w$ with $w(1) = p/(k+\ell)$ and $w(0) = 1 - w(1)$ solves the instance with overwhelming probability (over both the uniform choice of the input and the coins of the algorithm) in time $2^{0.117n}$ in the FDD setting and time $2^{0.0557n}$ in the HDD setting.

Let $\kappa := k/n$, $\rho := p/n$ and $\zeta := \ell/n$ and $\delta := H^{-1}(1 - \kappa)$ such that $\bar{\omega} := \delta$ in the FDD and $\tilde{\omega} := \delta/2$ in the HDD setting. Then the time complexity for the repetitions is $\tilde{O}(2^{\varepsilon n})$ with

\[ \varphi = H(\bar{\omega}) - H_{\kappa+\zeta}(\rho) - H_{1-\kappa-\zeta}(\bar{\omega} - \rho). \]

Due to Corollary [49] the algorithm MitmList solves the problem in $F_2^\ell$ with probability $1 - O(2^{-\varepsilon n}) = 1 - O(2^{-\varepsilon n})$ in time

\[ \tilde{O}(2^{\varepsilon n} + 2^{C_0 n} + 2^{(C_1+\varepsilon)n}) \]

with $C_0 = (\kappa + \zeta) \cdot H(\frac{\rho}{\kappa+\zeta})/2$ and $C_1 = (\kappa + \zeta) \cdot H(\frac{\rho}{\kappa+\zeta}) - \zeta$, where $C_0$ is the construction time for the two lists and $C_1$ is the time to merge and verify, as illustrated in Fig. 6.4.
Let us assume for the sake of simplicity that the chosen $\varepsilon$ is arbitrary small, i.e. $\varepsilon = 0$, in order to be able to compare the time complexities more easily. In the FDD case (Table 6.3), we obtain the following time complexity $\tilde{O}(2^{Cn})$ with $C = \varphi + \max\{C_0, C_1\}$ for several rates $\kappa$, with a maximal value at $\kappa \approx 0.446$. At this value, the complexity becomes $2^{0.117n}$. The time complexity is always optimized such that $C_0 = C_1$.

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.446</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.058</td>
<td>0.089</td>
<td>0.108</td>
<td>0.116</td>
<td>0.117</td>
<td>0.116</td>
<td>0.108</td>
<td>0.093</td>
<td>0.071</td>
<td>0.041</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.044</td>
<td>0.066</td>
<td>0.078</td>
<td>0.082</td>
<td>0.081</td>
<td>0.079</td>
<td>0.070</td>
<td>0.056</td>
<td>0.038</td>
<td>0.017</td>
</tr>
<tr>
<td>$C_0$</td>
<td>0.014</td>
<td>0.024</td>
<td>0.030</td>
<td>0.035</td>
<td>0.036</td>
<td>0.038</td>
<td>0.038</td>
<td>0.037</td>
<td>0.033</td>
<td>0.024</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.014</td>
<td>0.024</td>
<td>0.030</td>
<td>0.035</td>
<td>0.036</td>
<td>0.038</td>
<td>0.038</td>
<td>0.037</td>
<td>0.033</td>
<td>0.024</td>
</tr>
<tr>
<td>$100\rho$</td>
<td>0.462</td>
<td>0.726</td>
<td>0.898</td>
<td>0.999</td>
<td>1.026</td>
<td>1.042</td>
<td>1.028</td>
<td>0.955</td>
<td>0.804</td>
<td>0.531</td>
</tr>
<tr>
<td>$100\zeta$</td>
<td>1.395</td>
<td>2.310</td>
<td>2.973</td>
<td>3.431</td>
<td>3.580</td>
<td>3.708</td>
<td>3.798</td>
<td>3.679</td>
<td>3.269</td>
<td>2.358</td>
</tr>
</tbody>
</table>

Table 6.3: time complexity of $\text{ExactDecoding}$ with $\text{MitmList}$ for FDD

In Table 6.4 we see similar results for the HDD setting with a maximum of $2^{0.0556n}$ at $\kappa \approx 0.466$.

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.466</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.0256</td>
<td>0.0408</td>
<td>0.0502</td>
<td>0.0548</td>
<td>0.0556</td>
<td>0.0554</td>
<td>0.0523</td>
<td>0.0456</td>
<td>0.0352</td>
<td>0.0206</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.0199</td>
<td>0.0316</td>
<td>0.0385</td>
<td>0.0415</td>
<td>0.0416</td>
<td>0.0412</td>
<td>0.0379</td>
<td>0.0317</td>
<td>0.0229</td>
<td>0.0115</td>
</tr>
<tr>
<td>$C_0$</td>
<td>0.0057</td>
<td>0.0093</td>
<td>0.0117</td>
<td>0.0134</td>
<td>0.0141</td>
<td>0.0143</td>
<td>0.0145</td>
<td>0.0139</td>
<td>0.0123</td>
<td>0.0091</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.0057</td>
<td>0.0093</td>
<td>0.0117</td>
<td>0.0134</td>
<td>0.0141</td>
<td>0.0143</td>
<td>0.0145</td>
<td>0.0139</td>
<td>0.0123</td>
<td>0.0091</td>
</tr>
<tr>
<td>$100\rho$</td>
<td>0.1487</td>
<td>0.2326</td>
<td>0.2848</td>
<td>0.3147</td>
<td>0.3243</td>
<td>0.3258</td>
<td>0.3200</td>
<td>0.2963</td>
<td>0.2508</td>
<td>0.1721</td>
</tr>
<tr>
<td>$100\zeta$</td>
<td>0.5639</td>
<td>0.9219</td>
<td>1.1693</td>
<td>1.3336</td>
<td>1.4020</td>
<td>1.4240</td>
<td>1.4438</td>
<td>1.3856</td>
<td>1.2264</td>
<td>0.9022</td>
</tr>
</tbody>
</table>

Table 6.4: time complexity of $\text{ExactDecoding}$ with $\text{MitmList}$ for HDD

### 6.1.3 MMT

The approach was improved by May, Meurer and Thomae [MMT11] by replacing the $\text{MitmList}$ component of the algorithm with the better approach by Howgrave-Graham and Joux [HJ10] of Chapter 4.

Figure 6.5: Decoding with MMT / one level BJMM
As can be seen in Corollary 50 and Fig. 6.5, two lists are constructed with the help of $A_1$, by enumerating lists of size $2^{C_1 n}$, using the exact knowledge (independently of $y$) of the syndrome on $\ell_1$ components. The size of these final lists is then $2^{C_0 n}$. With the help of $A_2$, these two lists are then merged in time $2^{C_2 n}$, using the exact knowledge of the syndrome on the first $\ell$ components, which leads to a list of candidates that already match the syndrome $\tilde{s}$ on the first $\ell$ components. In the same time $2^{C_2 n}$, all elements in this list are finally checked with $B$, whether they match the syndrome on all components. The result is as follows.

Corollary 71 (MMTDecoding). Let $(H, s)$ be an instance of a $(n, k)$ Decoding Problem. Then for any $\varepsilon > 0$, any $p, \ell \in \mathbb{N}$, the algorithm \texttt{ExactDecoding} instantiated with the sub-algorithm \texttt{ClassicalRep} and one level can be used to solve the instance with overwhelming probability (over both the uniform choice of the input and the coins of the algorithm) in time $2^{0.112 n}$ in the FDD setting and time $2^{0.0537 n}$ in the HDD setting.

Let $\kappa := k/n$, $\rho := p/n$ and $\zeta := \ell/n$ and $\delta := H^{-1}(1 - \kappa)$ such that $\tilde{\omega} := \delta$ in the FDD and $\omega := \delta/2$ in the HDD setting. Then the time complexity for the repetitions is $\tilde{O}(2^{\varepsilon n})$ with $\varphi = H(\omega) - H_{\kappa + \zeta}(\rho) - H_{1 - \kappa - \zeta}(\omega - \rho)$.

Assume the parameter choice of the algorithm \texttt{ClassicalRep} presented in Corollary 50. Notice that the group $\mathbb{F}_2^\ell$ can be arbitrary split component-wise into groups $G := \mathbb{F}_2^{\ell_1} \times \mathbb{F}_2^{\ell - \ell_1}$ for any $0 \leq \ell_1 \leq \ell$ such that the sizes of the individual components of $G$ can be arbitrary chosen, which is required by the corollary. Due to the fact that we have a $(G, k + \ell, \frac{p}{\kappa + \zeta})$ Random Binary Subset Sum Problem, we obtain a time complexity

$$\tilde{O}(2^{\varepsilon n} + 2^{C_1 n} + 2^{\varepsilon n} \cdot 2^{C_0 n} + 2^{\varepsilon n} \cdot 2^{C_2 n})$$

with $C_0 = H_{\kappa + \zeta}(\frac{\rho}{2}) - \rho$, $C_1 = H_{\kappa + \zeta}(\frac{\rho}{2})/2$ and $C_2 = 2 \cdot H_{\kappa + \zeta}(\frac{\rho}{2}) - \rho - \zeta$. For arbitrary small $\varepsilon$ in the FDD case (Table 6.5), we obtain the following time complexity $\tilde{O}(2^{C n})$ with $C = \varphi + \max\{C_0, C_1, C_2\}$ for several rates $\kappa$, with a maximal value at $\kappa \approx 0.439$. At this value, the complexity becomes $2^{0.112 n}$.

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.439</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C'$</td>
<td>0.057</td>
<td>0.087</td>
<td>0.104</td>
<td>0.111</td>
<td>0.112</td>
<td>0.111</td>
<td>0.103</td>
<td>0.088</td>
<td>0.067</td>
<td>0.038</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.037</td>
<td>0.052</td>
<td>0.059</td>
<td>0.059</td>
<td>0.058</td>
<td>0.055</td>
<td>0.047</td>
<td>0.036</td>
<td>0.023</td>
<td>0.010</td>
</tr>
<tr>
<td>$C_0$</td>
<td>0.020</td>
<td>0.035</td>
<td>0.045</td>
<td>0.052</td>
<td>0.054</td>
<td>0.056</td>
<td>0.056</td>
<td>0.056</td>
<td>0.044</td>
<td>0.029</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.015</td>
<td>0.026</td>
<td>0.033</td>
<td>0.037</td>
<td>0.038</td>
<td>0.039</td>
<td>0.039</td>
<td>0.036</td>
<td>0.029</td>
<td>0.019</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.020</td>
<td>0.035</td>
<td>0.045</td>
<td>0.052</td>
<td>0.054</td>
<td>0.056</td>
<td>0.056</td>
<td>0.056</td>
<td>0.056</td>
<td>0.044</td>
</tr>
<tr>
<td>$100\rho$</td>
<td>0.954</td>
<td>1.559</td>
<td>1.928</td>
<td>2.116</td>
<td>2.147</td>
<td>2.150</td>
<td>2.041</td>
<td>1.788</td>
<td>1.373</td>
<td>0.774</td>
</tr>
<tr>
<td>$100\zeta$</td>
<td>2.948</td>
<td>5.008</td>
<td>6.424</td>
<td>7.309</td>
<td>7.524</td>
<td>7.714</td>
<td>7.637</td>
<td>7.029</td>
<td>5.768</td>
<td>3.622</td>
</tr>
</tbody>
</table>

Table 6.5: time complexity of \texttt{ExactDecoding} with \texttt{ClassicalRep} ($u = 1$) for FDD

The time complexity is always optimized as $C_0 = C_2$, whereas the bottom complexity $C_1$ is dominated. In Table 6.6 we see similar results for the HDD setting with a maximum of $2^{0.0537 n}$ at $\kappa \approx 0.463$. 


6.1 Classical Algorithms

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.463</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.0248</td>
<td>0.0395</td>
<td>0.0485</td>
<td>0.0529</td>
<td>0.0537</td>
<td>0.0534</td>
<td>0.0504</td>
<td>0.0438</td>
<td>0.0336</td>
<td>0.0195</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.0160</td>
<td>0.0251</td>
<td>0.0303</td>
<td>0.0323</td>
<td>0.0322</td>
<td>0.0317</td>
<td>0.0287</td>
<td>0.0233</td>
<td>0.0164</td>
<td>0.0078</td>
</tr>
<tr>
<td>$C_0$</td>
<td>0.0089</td>
<td>0.0144</td>
<td>0.0182</td>
<td>0.0206</td>
<td>0.0215</td>
<td>0.0218</td>
<td>0.0217</td>
<td>0.0205</td>
<td>0.0173</td>
<td>0.0118</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.0060</td>
<td>0.0097</td>
<td>0.0121</td>
<td>0.0135</td>
<td>0.0140</td>
<td>0.0141</td>
<td>0.0139</td>
<td>0.0130</td>
<td>0.0108</td>
<td>0.0073</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.0089</td>
<td>0.0144</td>
<td>0.0182</td>
<td>0.0206</td>
<td>0.0215</td>
<td>0.0218</td>
<td>0.0217</td>
<td>0.0205</td>
<td>0.0173</td>
<td>0.0118</td>
</tr>
<tr>
<td>100$\rho$</td>
<td>0.3157</td>
<td>0.4851</td>
<td>0.5837</td>
<td>0.6318</td>
<td>0.6406</td>
<td>0.6388</td>
<td>0.6087</td>
<td>0.5457</td>
<td>0.4313</td>
<td>0.2649</td>
</tr>
<tr>
<td>100$\zeta$</td>
<td>1.1967</td>
<td>1.9242</td>
<td>2.4021</td>
<td>2.6901</td>
<td>2.7859</td>
<td>2.8131</td>
<td>2.7781</td>
<td>2.591</td>
<td>2.1598</td>
<td>1.4402</td>
</tr>
</tbody>
</table>

Table 6.6: time complexity of ExactDecoding with ClassicalRep ($u = 1$) for HDD

In both cases it doesn’t help to add another level of representations, because the weight $\frac{\rho}{\kappa+\zeta}$ is much smaller than the bound $\Delta \approx 0.313$ which means that the bottom complexity $C_1$ is also smaller than the other two complexities. Since adding another level would only improve $C_1$, it doesn’t improve the overall complexity. Notice that the number of representations is simply $\tilde{O}(2^m)$.

6.1.4 BJMM

The group $F_2^\ell$ has the special property that each group element has an order of at most 2. This means, we can apply the improved representation algorithm that makes use of $1 + 1 = 0$ and is the main contribution of [BJMM12]. Whereas the MMT approach doesn’t extend to more than one level, the BJMM approach gets better with a higher number of levels. However, we want to begin by presenting the one level results and extend the levels step by step as done in [Men13]. The following approach uses the same construction as illustrated in Fig. 6.5.

**Corollary 72 (BJMMDecoding1).** Let $(H, s)$ be an instance of a $(n, k)$ Decoding Problem. Then for any $\varepsilon > 0$, any $p, \ell \in \mathbb{N}$, the algorithm ExactDecoding instantiated with the sub-algorithm ClassicalRep, with one level and making use of the special group structure of $F_2^\ell$, can be used to solve the instance with overwhelming probability (over both the uniform choice of the input and the coins of the algorithm) in time $2^{0.106n}$ in the FDD setting and time $2^{0.0505n}$ in the HDD setting.

Let $\kappa := k/n$, $\rho := p/n$ and $\zeta := \ell/n$ and $\delta := \mathcal{H}^{-1}(1 - \kappa)$ such that $\tilde{\omega} := \delta$ in the FDD and $\tilde{\omega} := \delta/2$ in the HDD setting. Then the time complexity for the repetitions is $\tilde{O}(2^m)$ with $\varphi = \mathcal{H}(\tilde{\omega}) - \mathcal{H}_{\kappa+\zeta}(\rho) - \mathcal{H}_{1-\kappa-\zeta}(\tilde{\omega} - \rho)$.

Assume the parameter choice of the algorithm ClassicalRep presented in Corollary 57. Notice that the group $F_2^\ell$ can be arbitrary split component-wise into groups $G := F_2^{\ell_1} \times F_2^{\ell_2}$ for any $0 \leq \ell_1 \leq \ell$ such that the sizes of the individual components of $G$ can be arbitrary chosen, which is required by the corollary. Due to the fact that we have a $(G, k+\ell, \frac{\rho}{\kappa+\zeta})$ Random Binary Subset Sum Problem, we obtain a time complexity

$$\tilde{O}(2^m + 2^{C_1 n} + 2^{6\varepsilon n} \cdot 2^{C_0 n} + 2^{6\varepsilon n} \cdot 2^{C_2 n}),$$

with $C_0 = \mathcal{H}_{\kappa+\zeta}(\frac{\tau+\varepsilon}{2}) - R_{0,1}$, $C_1 = \mathcal{H}_{\kappa+\zeta}(\frac{\tau+\varepsilon}{2})/2$, $C_2 = 2\mathcal{H}_{\kappa+\zeta}(\frac{\tau+\varepsilon}{2}) - R_{0,1} - \zeta$ and $R_{0,1} = \rho + \mathcal{H}_{\kappa+\zeta-\rho}(\frac{\tau}{2})$ for some $0 \leq \tau \leq 2(\kappa + \zeta - \rho)$. 

For arbitrary small $\varepsilon$ in the FDD case (Table 6.7), we obtain the following time complexity $\tilde{O}(2^{Cn})$ with

$$C = \varphi + \max\{C_0, C_1, C_2\}$$

for several rates $\kappa$, with a maximal value at $\kappa \approx 0.428$. At this value, the complexity becomes $2^{0.106n}$.

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.428</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.055</td>
<td>0.084</td>
<td>0.099</td>
<td>0.105</td>
<td>0.106</td>
<td>0.104</td>
<td>0.096</td>
<td>0.081</td>
<td>0.061</td>
<td>0.034</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.026</td>
<td>0.040</td>
<td>0.043</td>
<td>0.041</td>
<td>0.036</td>
<td>0.036</td>
<td>0.028</td>
<td>0.019</td>
<td>0.011</td>
<td>0.004</td>
</tr>
<tr>
<td>$C_0$</td>
<td>0.029</td>
<td>0.044</td>
<td>0.056</td>
<td>0.065</td>
<td>0.066</td>
<td>0.069</td>
<td>0.068</td>
<td>0.062</td>
<td>0.050</td>
<td>0.031</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.029</td>
<td>0.044</td>
<td>0.056</td>
<td>0.065</td>
<td>0.066</td>
<td>0.069</td>
<td>0.068</td>
<td>0.062</td>
<td>0.050</td>
<td>0.031</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.029</td>
<td>0.044</td>
<td>0.056</td>
<td>0.065</td>
<td>0.066</td>
<td>0.069</td>
<td>0.068</td>
<td>0.062</td>
<td>0.050</td>
<td>0.031</td>
</tr>
<tr>
<td>$R_{0.1}$</td>
<td>0.029</td>
<td>0.044</td>
<td>0.056</td>
<td>0.066</td>
<td>0.066</td>
<td>0.069</td>
<td>0.068</td>
<td>0.062</td>
<td>0.050</td>
<td>0.031</td>
</tr>
<tr>
<td>100$\rho$</td>
<td>1.982</td>
<td>2.696</td>
<td>3.302</td>
<td>3.593</td>
<td>3.624</td>
<td>3.601</td>
<td>3.348</td>
<td>2.828</td>
<td>2.045</td>
<td>1.040</td>
</tr>
<tr>
<td>100$\tau$</td>
<td>0.208</td>
<td>0.383</td>
<td>0.534</td>
<td>0.652</td>
<td>0.679</td>
<td>0.731</td>
<td>0.762</td>
<td>0.731</td>
<td>0.615</td>
<td>0.382</td>
</tr>
</tbody>
</table>

Table 6.7: time complexity of ExactDecoding with ClassicalRep ($g = 2, u = 1$) for FDD

In Table 6.8 we see similar results for the HDD setting with a maximum of $2^{0.0505n}$ at $\kappa \approx 0.458$.

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.458</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.0236</td>
<td>0.0375</td>
<td>0.0459</td>
<td>0.0499</td>
<td>0.0505</td>
<td>0.0502</td>
<td>0.0470</td>
<td>0.0407</td>
<td>0.0310</td>
<td>0.0178</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.0123</td>
<td>0.0189</td>
<td>0.0222</td>
<td>0.0230</td>
<td>0.0226</td>
<td>0.0219</td>
<td>0.0185</td>
<td>0.0151</td>
<td>0.0096</td>
<td>0.0041</td>
</tr>
<tr>
<td>$C_0$</td>
<td>0.0114</td>
<td>0.0187</td>
<td>0.0237</td>
<td>0.0269</td>
<td>0.0280</td>
<td>0.0283</td>
<td>0.0286</td>
<td>0.0256</td>
<td>0.0215</td>
<td>0.0137</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.0114</td>
<td>0.0187</td>
<td>0.0237</td>
<td>0.0269</td>
<td>0.0280</td>
<td>0.0283</td>
<td>0.0286</td>
<td>0.0256</td>
<td>0.0215</td>
<td>0.0137</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.0114</td>
<td>0.0187</td>
<td>0.0237</td>
<td>0.0269</td>
<td>0.0280</td>
<td>0.0283</td>
<td>0.0286</td>
<td>0.0256</td>
<td>0.0215</td>
<td>0.0137</td>
</tr>
<tr>
<td>$R_{0.1}$</td>
<td>0.0114</td>
<td>0.0187</td>
<td>0.0237</td>
<td>0.0269</td>
<td>0.0280</td>
<td>0.0283</td>
<td>0.0286</td>
<td>0.0256</td>
<td>0.0215</td>
<td>0.0137</td>
</tr>
<tr>
<td>100$\rho$</td>
<td>0.5600</td>
<td>0.8651</td>
<td>1.0424</td>
<td>1.1254</td>
<td>1.1348</td>
<td>1.1250</td>
<td>1.0795</td>
<td>0.8972</td>
<td>0.6929</td>
<td>0.3874</td>
</tr>
<tr>
<td>100$\zeta$</td>
<td>2.2610</td>
<td>3.7256</td>
<td>4.7354</td>
<td>5.3758</td>
<td>5.5837</td>
<td>5.6598</td>
<td>5.7113</td>
<td>5.1106</td>
<td>4.2808</td>
<td>2.7377</td>
</tr>
<tr>
<td>100$\tau$</td>
<td>0.1273</td>
<td>0.2180</td>
<td>0.2841</td>
<td>0.3285</td>
<td>0.3442</td>
<td>0.3508</td>
<td>0.3573</td>
<td>0.3223</td>
<td>0.2702</td>
<td>0.1703</td>
</tr>
</tbody>
</table>

Table 6.8: time complexity of ExactDecoding with ClassicalRep ($g = 2, u = 1$) for HDD

The time complexity is always optimized such that $C_0 = C_1 = C_2$, which means adding one level in the algorithm helps to improve the result. This leads to the main result in [BJMM12], which again begins by enumerating lists in time $2^{C_1n} + 2^{C_2n}$ with the help of $A_1$, knowing the syndrome exactly on $\ell_1$ components. These lists are merged with $A_2$ in time $2^{C_0n}$, knowing the syndrome on additional $\ell_2$ components. Finally, the lists are merged to one list of candidates in time $2^{C_3n}$, using $A_3$ and knowing the syndrome exactly (independently of $y$) on a total number of $\ell$ components. The final list can be searched for matches on the whole syndrome with the help of $B$ in the same time complexity. This is illustrated in Fig. 6.6 and Fig. 6.7.
time on each level: \[2^{C_0n}\]
\[2^{C_2n}\]

\[\begin{array}{ccc}
& 1 & \text{with } A_3 \\
2^{C_3n} & \text{list sizes on each level:} & \text{with } A_2 \\
& 2^{C_0n} & \text{with } A_1 \\
\end{array}\]

Figure 6.6: tree structure of the two level representation algorithm

complexity:
repetitions (outer loop): \[2^{\omega n}\]
list enumeration with \(A_1\): \[2^{C_2n}\]
final size of these lists: \[2^{C_1n}\]
merge with \(A_2\): \[2^{C_0n}\]
merge with \(A_3\), verify with \(B\): \[2^{C_3n}\]

Figure 6.7: Decoding with two level BJMM

**Corollary 73** (BJMMDecoding2). Let \((H,s)\) be an instance of a \((n,k)\) Decoding Problem. Then for any \(\varepsilon > 0\), any \(p, \ell \in \mathbb{N}\), the algorithm \textsc{ExactDecoding} instantiated with the subalgorithm \textsc{ClassicalRep}, with two levels and making use of the special group structure of \(\mathbb{F}_2\), can be used to solve the instance with overwhelming probability (over both the uniform choice of the input and the coins of the algorithm) in time \(2^{0.102n}\) in the FDD setting and time \(2^{0.0494n}\) in the HDD setting.

Let \(\kappa := k/n\), \(\rho := p/n\) and \(\zeta := \ell/n\) and \(\delta := \mathcal{H}^{-1}(1 - \kappa)\) such that \(\tilde{\omega} := \delta\) in the FDD and \(\bar{\omega} := \delta/2\) in the HDD setting. Then the time complexity for the repetitions is \(\tilde{O}(2^{\omega n})\) with \(\varphi = \mathcal{H}(\bar{\omega}) - \mathcal{H}_{\kappa + \zeta}(\rho) - \mathcal{H}_{1 - \kappa - \zeta}(\bar{\omega} - \rho)\).

Assume the parameter choice of the algorithm \textsc{ClassicalRep} presented in Corollary 58. Notice that the group \(\mathbb{F}_2^\ell\) can be arbitrary split component-wise into \(G := \mathbb{F}_{2^{\ell_1}} \times \mathbb{F}_{2^{\ell_2}} \times \mathbb{F}_{2^{\ell_1 - \ell_2}}\) for any \(0 \leq \ell_1, \ell_2, \ell_1 + \ell_2 \leq \ell\) such that the sizes of the individual components of \(G\) can be arbitrary chosen, which is required by the corollary. Due to the fact that we have a \((G, k + \ell, \frac{p}{k+\ell})\) Random Binary Subset Sum Problem, we obtain a time complexity

\[\tilde{O}(2^{\varepsilon n} + 2^{C_3n} + 2^{6\varepsilon n} \cdot 2^{C_1n} + 2^{6\varepsilon n} \cdot 2^{C_0n} + 2^{6\varepsilon n} \cdot 2^{C_2n})\]

with \(C_0 = 2\mathcal{H}_{\kappa + \zeta}(\frac{\rho + \tau_1 + \tau_2}{4}) - R_{0,1} - R_{1,2}, C_1 = \mathcal{H}_{\kappa + \zeta}(\frac{\rho + \tau_1 + \tau_2}{4}) - R_{1,2}, C_2 = \mathcal{H}_{\kappa + \zeta}(\frac{\rho + \tau_1 + \tau_2}{4})/2, C_3 = 2\mathcal{H}_{\kappa + \zeta}(\frac{\rho + \tau_1}{2}) - R_{0,1} - \zeta, R_{0,1} = \rho + \mathcal{H}_{\kappa + \zeta - \rho}(\frac{\tau_2}{4})\) and \(R_{1,2} = \frac{\rho + \tau_2}{4} + \mathcal{H}_{\kappa + \zeta - \rho + \tau_1}(\frac{\tau_2}{4})\) for some \(0 \leq \tau_1 \leq 2(\kappa + \zeta - \rho)\) and some \(0 \leq \tau_2 \leq 4(\kappa + \zeta) - 2(\rho + \tau_1)\).

For arbitrary small \(\varepsilon\) in the FDD case (Table 6.9), we obtain the following time complexity \(\tilde{O}(2^{C n})\) with

\[C = \varphi + \max\{C_0, C_1, C_2, C_3\}\]

for several rates \(\kappa\), with a maximal value at \(\kappa \approx 0.427\). At this value, the complexity becomes \(2^{0.102n}\).
6. Decoding Problem

Table 6.9: time complexity of ExactDecoding with ClassicalRep \((g = 2, u = 2)\) for FDD

<table>
<thead>
<tr>
<th>(\kappa)</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.427</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C)</td>
<td>0.054</td>
<td>0.081</td>
<td>0.096</td>
<td>0.102</td>
<td>0.102</td>
<td>0.100</td>
<td>0.092</td>
<td>0.079</td>
<td>0.059</td>
<td>0.033</td>
</tr>
<tr>
<td>(C_0)</td>
<td>0.034</td>
<td>0.055</td>
<td>0.070</td>
<td>0.076</td>
<td>0.078</td>
<td>0.079</td>
<td>0.076</td>
<td>0.070</td>
<td>0.053</td>
<td>0.031</td>
</tr>
<tr>
<td>(C_1)</td>
<td>0.034</td>
<td>0.055</td>
<td>0.070</td>
<td>0.076</td>
<td>0.078</td>
<td>0.079</td>
<td>0.076</td>
<td>0.070</td>
<td>0.053</td>
<td>0.031</td>
</tr>
<tr>
<td>(C_2)</td>
<td>0.034</td>
<td>0.055</td>
<td>0.070</td>
<td>0.076</td>
<td>0.078</td>
<td>0.079</td>
<td>0.076</td>
<td>0.070</td>
<td>0.053</td>
<td>0.031</td>
</tr>
<tr>
<td>(C_3)</td>
<td>0.034</td>
<td>0.055</td>
<td>0.070</td>
<td>0.076</td>
<td>0.078</td>
<td>0.079</td>
<td>0.076</td>
<td>0.070</td>
<td>0.053</td>
<td>0.031</td>
</tr>
<tr>
<td>(R_{0,1})</td>
<td>0.068</td>
<td>0.103</td>
<td>0.126</td>
<td>0.130</td>
<td>0.132</td>
<td>0.129</td>
<td>0.118</td>
<td>0.111</td>
<td>0.075</td>
<td>0.041</td>
</tr>
<tr>
<td>(R_{1,2})</td>
<td>0.034</td>
<td>0.048</td>
<td>0.057</td>
<td>0.054</td>
<td>0.054</td>
<td>0.050</td>
<td>0.043</td>
<td>0.042</td>
<td>0.022</td>
<td>0.011</td>
</tr>
</tbody>
</table>

In Table 6.10 we see similar results for the HDD setting with a maximum of \(2^{0.0494n}\) at \(\kappa \approx 0.453\).

Table 6.10: time complexity of ExactDecoding with ClassicalRep \((g = 2, u = 2)\) for HDD

<table>
<thead>
<tr>
<th>(\kappa)</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.453</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C)</td>
<td>0.0231</td>
<td>0.0367</td>
<td>0.0449</td>
<td>0.0488</td>
<td>0.0494</td>
<td>0.0491</td>
<td>0.0460</td>
<td>0.0398</td>
<td>0.0304</td>
<td>0.0175</td>
</tr>
<tr>
<td>(\varphi)</td>
<td>0.0099</td>
<td>0.0154</td>
<td>0.0183</td>
<td>0.0189</td>
<td>0.0203</td>
<td>0.0180</td>
<td>0.0157</td>
<td>0.0122</td>
<td>0.0080</td>
<td>0.0035</td>
</tr>
<tr>
<td>(C_0)</td>
<td>0.0132</td>
<td>0.0213</td>
<td>0.0266</td>
<td>0.0299</td>
<td>0.0292</td>
<td>0.0311</td>
<td>0.0304</td>
<td>0.0277</td>
<td>0.0224</td>
<td>0.0141</td>
</tr>
<tr>
<td>(C_1)</td>
<td>0.0132</td>
<td>0.0213</td>
<td>0.0266</td>
<td>0.0299</td>
<td>0.0292</td>
<td>0.0311</td>
<td>0.0304</td>
<td>0.0277</td>
<td>0.0224</td>
<td>0.0141</td>
</tr>
<tr>
<td>(C_2)</td>
<td>0.0105</td>
<td>0.0162</td>
<td>0.0195</td>
<td>0.0215</td>
<td>0.0201</td>
<td>0.0217</td>
<td>0.0205</td>
<td>0.0183</td>
<td>0.0145</td>
<td>0.0088</td>
</tr>
<tr>
<td>(C_3)</td>
<td>0.0132</td>
<td>0.0213</td>
<td>0.0266</td>
<td>0.0299</td>
<td>0.0292</td>
<td>0.0311</td>
<td>0.0304</td>
<td>0.0277</td>
<td>0.0224</td>
<td>0.0141</td>
</tr>
<tr>
<td>(R_{0,1})</td>
<td>0.0209</td>
<td>0.0323</td>
<td>0.0390</td>
<td>0.0429</td>
<td>0.0402</td>
<td>0.0434</td>
<td>0.0410</td>
<td>0.0366</td>
<td>0.0290</td>
<td>0.0175</td>
</tr>
<tr>
<td>(R_{1,2})</td>
<td>0.0077</td>
<td>0.0110</td>
<td>0.0125</td>
<td>0.0130</td>
<td>0.0110</td>
<td>0.0124</td>
<td>0.0107</td>
<td>0.0089</td>
<td>0.0067</td>
<td>0.0035</td>
</tr>
<tr>
<td>(10^2 \rho)</td>
<td>3.8018</td>
<td>1.1764</td>
<td>1.3560</td>
<td>1.4270</td>
<td>1.3128</td>
<td>1.3851</td>
<td>1.2602</td>
<td>1.0578</td>
<td>0.7733</td>
<td>0.4148</td>
</tr>
<tr>
<td>(10^2 \zeta)</td>
<td>3.3805</td>
<td>5.3271</td>
<td>6.5215</td>
<td>7.2383</td>
<td>6.9185</td>
<td>7.4194</td>
<td>7.1289</td>
<td>6.4093</td>
<td>5.1293</td>
<td>3.1486</td>
</tr>
<tr>
<td>(10^2 \tau_1)</td>
<td>0.3340</td>
<td>0.5118</td>
<td>0.6135</td>
<td>0.6685</td>
<td>0.6125</td>
<td>0.6690</td>
<td>0.6191</td>
<td>0.5429</td>
<td>0.4214</td>
<td>0.2422</td>
</tr>
<tr>
<td>(10^2 \tau_2)</td>
<td>0.0721</td>
<td>0.0859</td>
<td>0.0847</td>
<td>0.0782</td>
<td>0.0397</td>
<td>0.0624</td>
<td>0.0341</td>
<td>0.0234</td>
<td>0.0160</td>
<td>0.0045</td>
</tr>
</tbody>
</table>

Whereas in the FDD case it might make sense to add another level for very small values of \(\kappa\), in the HDD setting it doesn’t help. This is due to the fact that in both cases for most of the \(\kappa\) the bottom level complexity \(C_2\) is dominated by the other three complexities \(C_0, C_1\) and \(C_3\), which are chosen to be identical.

Until 2015, this was the best known algorithm for the Decoding Problem. In [MO15], the novel idea based on the Nearest Neighbor Problem of Chapter 2 was presented that in combination with the results of Chapter 3 leads to a faster algorithm for the Decoding Problem and is explained in the following section.
6.2 Application of the Nearest Neighbor Technique

6.2.1 Basic Approach

In the algorithms of the Random Binary Subset Sum Problem in general groups $G$, we always have a final step that involves merging two lists. Since the complexity of this step is always of the form $T >= \lvert G \rvert$, for some complexity $T$, this part can be neglected whenever $\lvert G \rvert$ is large enough. In the previous approaches, the parameter $\ell$ that regulates the size of the group $G = \mathbb{F}_2^\ell$ is chosen such that the time of this step matches the worst case time of the other parts of the complexity. Generally, we want to choose $\ell$ as small as possible, due to the fact that it additionally increases the vector size of our algorithms as well as the complexity for the outer loop that searches for a good permutation $\pi$.

Assume there is a $e^* \in \mathbb{F}_2^n$ with Hamming weight $\omega^*$ such that $He^* = s$. The idea of the new approach is that we can get rid of $\ell$ at all by making use of the fact that the correct solution has an unusually small Hamming weight. That is, similarly to Stern’s approach, we first of all denote $Q \in \mathbb{F}_2^{(n-k)\times(n-k)}$ the right hand part of $\pi(H)$, where $\pi$ is a permutation such that $\pi(e^*) = (x^* || y^*)$ with $x^* \in \mathbb{F}_2^n$ and $y^* \in \mathbb{F}_2^n$ such that $wt(x^*) = p$ and $wt(y^*) = \omega^* - p$ for some $p$. Notice that we moreover want to assume that $x^* = x^*_1 + x^*_2$ with $x^*_1 \in \mathbb{F}_2^{k/2} \times 0^{k/2}$ and $x^*_2 \in 0^{k/2} \times \mathbb{F}_2^{k/2}$ such that $wt(x^*_1) = wt(x^*_2) = p/2$.

If $Q$ is invertible, one obtains the equation $H(x^* || y^*) = s$ with $H = Q^{-1} \cdot \pi(H)$ and $s = Q^{-1} \cdot s$. Due to the fact that $H = (A || I)$ for some matrix $A \in \mathbb{F}_2^{(n-k)\times k}$ and the identity matrix $I$ of dimension $n - k$, the equation simplifies to $Ax^* + y^* = s$. This directly implies that the Hamming weight of $Ax^*_1 + \bar{s}$ is the Hamming weight of $y^*$, which is $\omega^* - p$, and therefore small. On the other hand, the Hamming weight of $Ax + s$ for most of the other $x$ with the same size and the same weight should be close to $\omega_{2}^* / 2$, due to the fact that $A$ is uniform. Moreover, any $x \in \mathbb{F}^k_2$ with $wt(x) = p$ such that the Hamming distance of $Ax + \bar{s}$ is $\omega^* - p$ leads to a solution to our problem, since $H(x || Ax + \bar{s}) = s$.

Therefore, the problem simplifies to solving an approximate version of a Random Binary Subset Sum Problem, i.e. $Ax \approx \bar{s}$ with known $A$ and $\bar{s}$ and unknown $x \in \{0, 1\}^k$ of a certain weight. Notice that checking for each possible $x$ if the corresponding $y$ has the desired weight is exactly the algorithm proposed by Lee and Brickell [LB88] that doesn’t lead to a better asymptotical complexity compared to PRANGEDECODING, which is the special case $p = 0$, i.e. $x$ is the zero vector.

An improved time complexity is obtained by a meet-in-the-middle approach very similarly to that of Stern. It makes use of the fact that $Ax^*_1 + Ax^*_2 + y^* = s$ and therefore the Hamming distance between $Ax^*_1$ and $Ax^*_2 + \bar{s}$ is $\omega^* - p$. Let $L_1$ be the list of $Ax_1$ for all $x_1 \in \mathbb{F}_2^{k/2} \times 0^{k}$ with weight $p/2$ and analogously $L_2$ the list of all $Ax_2 + \bar{s}$ for all $x_2 \in 0^{k} \times \mathbb{F}_2^{k/2}$ with weight $p/2$. Then one can be sure that there is a pair $(Ax^*_1, Ax^*_2) \in L_1 \times L_2$ with a Hamming distance of $\omega^* - p$. Of course, searching for this pair naively – by comparing each element of the first list with each element of the second list – would lead to the same time complexity as for brute-forcing the $x$ directly. However, since the target pair has a particularly small Hamming weight and all elements in both lists are pairwise independent due to uniform $A$, we obtain exactly a Nearest Neighbor Problem of Chapter 2 for which we know a sub-quadratic algorithm. This idea leads to the algorithm nnBASICDECODING, firstly introduced in [MO15].
In Fig. 6.8 we see the construction of the algorithm. Compared to the previous approach by Stern, we get rid of the \( \ell \) and enumerate two lists directly, using the full matrix \( A \). These two lists contain elements that are not exactly the same, but have a small Hamming distance. Vectors with a fitting distance are then found with a nearest neighbor approach.

\[
\begin{array}{c|c|c}
\text{n} & \text{k} & \text{n} - k \\
\hline
A & I & \bar{s}
\end{array}
\]

**complexity:**
repetitions (outer loop): \( 2^{\psi n} \)
list sizes: \( 2^{\mu n} \)
nearest neighbor: \( 2^{\psi n} \)

**Theorem 74** (nnBasicDecoding \([\text{MO15}]\)). \textsc{nnBasicDecoding} solves the decoding problem with overwhelming probability in time \( \mathcal{O}(2^{0.114n}) \) in the full and time \( \mathcal{O}(2^{0.0550n}) \) in the half distance decoding setting.

**Proof.** Let \( \kappa := k/n, \delta := \mathcal{H}^{-1}(1 - \kappa) \) and let \( \omega_{\text{max}} := \delta n \) in the FDD setting and \( \omega_{\text{max}} := \delta n/2 \) in the HDD setting. Assume there is a solution \( \mathbf{e}^* \) with \( \omega^* := \text{wt}(\mathbf{e}^*) \leq \omega_{\text{max}} \). Notice that otherwise the algorithm correctly outputs \( \perp \).

Now fix some \( p \) and let us continue by showing the correctness in the case such an \( \mathbf{e}^* \) exists. Let \( S \subseteq \mathbb{F}_2^n \) be the set of all vectors of Hamming weight \( \omega^* \) that are \( p/2 \) on the first \( k/2 \)
components, $p/2$ on the middle $k/2$ components and have the remaining weight of $\omega - p$ on the last $n - k - \ell$ components. Since $|S| = \binom{k/2}{p/2}^2 \cdot \binom{n-k-\ell}{\omega-p}$, the Redistribution Lemma guarantees with a probability of at least $1 - 2^{-\varepsilon n}$ that after $4\varepsilon n \cdot \binom{n}{\omega}/|S|$ chosen permutations there is at least one such that $\pi(e^*) \in S$ and $Q \in \mathbb{F}_2^{(n-k) \times (n-k)}$ with $\pi(H) = (\cdot ||Q)$ is invertible. Fix that permutation $\pi$, which implies that $\pi(H)\pi(e^*) = s$.

Let $\pi(e^*) = ((x^*_i + x^*_j)||y^*)$ with $x^*_i \in \mathbb{F}_2^{k/2} \times 0^{k/2}$ and $x^*_j \in 0^{k/2} \times \mathbb{F}_2^{k/2}$. This means that $(Ax^*_i, Ax^*_j) \in L_1 \times L_2$ and $wt(y^*) = \omega^* - p$. Moreover, the lists $L_1, L_2$ consist of uniform and pairwise independent elements, due to the fact that $H$ and therefore $A$ as well as $s$ and therefore $\mathsf{Red}$ are uniform. Denote $\gamma := \frac{\omega - p}{n-k}$ and $m := n - k$ and notice that $\lambda := \frac{1}{m} \cdot \log_2(\text{max}\{|L_1|, |L_2|\}) \leq 1 - \mathcal{H}(\frac{\gamma}{2})$, which can be verified numerically. Thus due to Theorem~24 the list $L_{\mathsf{out}}$ contains the vector pair $(Ax^*_i, Ax^*_j + \bar{s})$ with a probability of at least $1 - \mathcal{O}(2^{-\varepsilon n})$. Due to the fact that it passes the Hamming distance check, we can easily compute the corresponding vector pair $(x^*_i, x^*_j)$. Now $\pi^{-1}((x^*_i + x^*_j)||(Ax^*_i + Ax^*_j + \bar{s}))$ solves the problem, since we have

$$H\left(\pi^{-1}((x^*_i + x^*_j)||(Ax^*_i + Ax^*_j + \bar{s}))\right) = Q \cdot (A||I) \cdot ((x^*_i + x^*_j)||(Ax^*_i + Ax^*_j + \bar{s})) = Q \cdot \bar{s} = s.$$}

Moreover, notice that any $(y_1, y_2)$ with Hamming distance $\omega - p$ is a valid solution to the problem, since $\text{wt}((x_1 + x_2)||(y_1 + y_2)) = \omega$.

Let us move to the time complexity of the algorithm. The time complexity for creating the two lists $L_1, L_2$ is $\tilde{O}(2^{\mu n})$ with $\mu := \frac{n}{2} \cdot \mathcal{H}(\rho/\kappa)$. Moreover, the algorithm NEARESTNEIGHBOR has a time complexity of

$$\tilde{O}\left(2^{(\mu + 3\varepsilon)n} + 2^{(\psi + 2\varepsilon)n}\right) \text{ with } \psi := (1 - \kappa)(1 - \gamma) \left(1 - \mathcal{H}\left(\frac{H^{-1}(1 - \frac{\mu}{1-\kappa}) - \frac{\gamma}{2}}{1 - \gamma}\right)\right),$$

since $\lambda := \frac{\mu}{1-\kappa}$. In addition, the for-loops have a complexity of $\tilde{O}(2^{\varepsilon n})$ with a repetition number $\varphi = \mathcal{H}(\bar{\omega}) - \mathcal{H}_\kappa(\rho) - \mathcal{H}_{1-\kappa}(\bar{\omega} - \rho)$, where $\rho := p/n$ and $\bar{\omega} := \delta$ in the FDD and $\bar{\omega} := \delta/2$ in the HDD setting. The overall time complexity is therefore $\tilde{O}(2^{(C + 3\varepsilon)n})$ with $C := \varphi + \max\{\mu, \psi\}$. This leads in the FDD setting to results in Table~6.11 with a maximum of $2^{0.114n}$ at $\kappa \approx 0.447$.

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.447</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C^\prime$</td>
<td>0.057</td>
<td>0.088</td>
<td>0.105</td>
<td>0.113</td>
<td>0.114</td>
<td>0.113</td>
<td>0.106</td>
<td>0.091</td>
<td>0.069</td>
<td>0.040</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.030</td>
<td>0.048</td>
<td>0.057</td>
<td>0.061</td>
<td>0.060</td>
<td>0.059</td>
<td>0.053</td>
<td>0.043</td>
<td>0.030</td>
<td>0.014</td>
</tr>
<tr>
<td>$\mu$</td>
<td>0.018</td>
<td>0.029</td>
<td>0.036</td>
<td>0.041</td>
<td>0.043</td>
<td>0.044</td>
<td>0.044</td>
<td>0.041</td>
<td>0.035</td>
<td>0.024</td>
</tr>
<tr>
<td>$\psi$</td>
<td>0.027</td>
<td>0.041</td>
<td>0.049</td>
<td>0.053</td>
<td>0.054</td>
<td>0.055</td>
<td>0.053</td>
<td>0.048</td>
<td>0.040</td>
<td>0.027</td>
</tr>
<tr>
<td>$100\rho$</td>
<td>0.638</td>
<td>0.978</td>
<td>1.171</td>
<td>1.273</td>
<td>1.290</td>
<td>1.289</td>
<td>1.230</td>
<td>1.097</td>
<td>0.877</td>
<td>0.541</td>
</tr>
</tbody>
</table>

Table 6.11: time complexity of nnBasicDecoding for FDD

In Table~6.12 we see similar results for the HDD setting with a maximum of $2^{0.0556n}$ at $\kappa \approx 0.466$. 


Similarly to the approach of [MMT11], we want to combine this new approach with the representation technique. This can be done without changing the underlying algorithm by reintroducing the parameter $\ell$, but only for the representation part. We discuss possible adaptions of the technique to an algorithm without $\ell$ at the end of this chapter. The last step, which is the merging of the final two lists, is done in the approximate manner. We want to show the correctness and time complexity of the algorithm \textsc{ApproxD}ecoding once again in a general manner, by making use of an algorithm \textsc{ColumnMatch} that implements one of the previously presented representation techniques, but slightly modified. The idea is to abort the algorithm before computing

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.466</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.0252</td>
<td>0.0403</td>
<td>0.0496</td>
<td>0.0542</td>
<td>0.0550</td>
<td>0.0548</td>
<td>0.0518</td>
<td>0.0451</td>
<td>0.0348</td>
<td>0.0204</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.0170</td>
<td>0.0275</td>
<td>0.0338</td>
<td>0.0367</td>
<td>0.0370</td>
<td>0.0367</td>
<td>0.0339</td>
<td>0.0285</td>
<td>0.0207</td>
<td>0.0105</td>
</tr>
<tr>
<td>$\mu$</td>
<td>0.0068</td>
<td>0.0110</td>
<td>0.0138</td>
<td>0.0155</td>
<td>0.0161</td>
<td>0.0163</td>
<td>0.0163</td>
<td>0.0153</td>
<td>0.0133</td>
<td>0.0094</td>
</tr>
<tr>
<td>$\psi$</td>
<td>0.0083</td>
<td>0.0129</td>
<td>0.0158</td>
<td>0.0175</td>
<td>0.0181</td>
<td>0.0182</td>
<td>0.0179</td>
<td>0.0167</td>
<td>0.0142</td>
<td>0.0099</td>
</tr>
<tr>
<td>$100\rho$</td>
<td>0.1903</td>
<td>0.2908</td>
<td>0.3491</td>
<td>0.3784</td>
<td>0.3848</td>
<td>0.3844</td>
<td>0.3696</td>
<td>0.3340</td>
<td>0.2747</td>
<td>0.1805</td>
</tr>
</tbody>
</table>

Table 6.12: time complexity of \textsc{nnBasicDecoding} for HDD
the final merge of the two top lists. Instead, we perform the presented NEARESTNEIGHBOR algorithm on these two final lists.

In the algorithm APPROXDECODING, we call the algorithm COLUMNMATCH as it is done in EXACTDECODING. However, we abort the algorithm before the last step of merging the final two lists and instead output these two final lists. Notice that this implies that for each pair of an \(x_1 \in L_1\) and an \(x_2 \in L_2\), we have \(A(x_1 + x_2) = [s]^{\ell}\). We want to prove the following theorem in general and then derive some corollaries that instantiate the algorithm COLUMNMATCH.

**Theorem 75 (ApproxDecoding).** Let \((H, s)\) be an instance of a \((n, k)\) Decoding Problem. Let COLUMNMATCH be an algorithm that for any \(\varepsilon > 0, p, \ell \in \mathbb{N}\) runs in time \(T(k, \ell, p, \varepsilon)\) and computes two lists \(L_1, L_2\) with the following properties. \(L_1\) is a list of vectors \(x_1\) with \(Ax_1 = s_1\) chosen uniformly at random, whereas \(L_2\) is a list of vectors \(x_2\) with \(Ax_2 = x_2\) with \(s_2 = [s]^{\ell} - s_1\). Moreover, with a probability of \(1 - O(2^{-\varepsilon n})\) (over both the random choice of the input and the coins of the algorithm), the lists have to property that for any fixed \(x^*\) with \(\text{wt}(x^*) = p\), there is at least one \(x_1^* \in L_1\) and one \(x_2^* \in L_2\) such that \(x_1^* + x_2^* = x^*\). Then for any \(\varepsilon > 0\) and any \(p, \ell \in \mathbb{N}\), the algorithm APPROXDECODING solves the decoding instance with a probability of \(1 - O(2^{-\varepsilon n})\) (over both the uniform choice of the input and the coins of the algorithm) in time

\[
\tilde{O}\left((2^\kappa)^{\mathcal{H}(\bar{\omega}) - \mathcal{H}(\kappa\cdot\zeta) - \mathcal{H}(\kappa - \zeta)} \cdot \max\{T(k, \ell, p, \varepsilon), 2^{(\mu + 3\varepsilon)n}, 2^{(\psi + 2\varepsilon)n}\}\right)
\]

with \(\kappa := k/n\), \(\delta := \mathcal{H}^{-1}(1 - \kappa)\), \(\rho := p/n\), \(\zeta := \ell/n\) and such that \(\bar{\omega} := \delta\) in the FDD and \(\bar{\omega} := \delta/2\) in the HDD setting. With \(\gamma := \frac{\omega - \rho}{n - k - \ell}\), \(\lambda := \frac{1}{n - k - \ell} \cdot \log_2(\max\{|L_1|, |L_2|\})\) such that \(\lambda \leq 1 - \mathcal{H}^{-1}(\frac{\gamma}{2})\), \(\mu := \lambda \cdot (1 - \kappa - \zeta)\) and \(\psi := (1 - \kappa - \zeta)(1 - \gamma) (1 - \mathcal{H}\left(\frac{\mathcal{H}^{-1}(1 - \frac{\rho}{n - k - \ell})}{1 - \gamma}\right))\).

**Proof.** Let \(\omega_{\max} := \delta n\) in the FDD setting and \(\omega_{\max} := \delta n/2\) in the HDD setting. Assume there is a solution \(e^*\) with \(\omega^* := \text{wt}(e^*) \leq \omega_{\max}\) of the Decoding Problem. Notice that otherwise the algorithm correctly outputs \(\perp\).

Now fix a choice of \(p\) and \(\ell\). Let us continue by showing the correctness in the case such an \(e^*\) exists. Let \(S \subseteq \mathbb{F}_2^n\) be the set of all vectors of Hamming weight \(\omega^*\) that are \(p\) on the first \(k + \ell\) components and have the remaining weight of \(\omega - p\) on the remaining \(n - k - \ell\) components. Since \(|S| = \left(\begin{array}{c} k + \ell \\ p \end{array}\right) \cdot \left(\begin{array}{c} 2^{n-k-\ell} \\ \omega^* - p \end{array}\right)\), the Redistribution Lemma guarantees with a probability of at least \(1 - 2^{-\varepsilon n}\) that after \(4\varepsilon n \cdot \left(\begin{array}{c} n \\ n-k-\ell \end{array}\right)\) chosen permutations there is at least one such that \(\pi(e^*) \in S\) and \(Q \in \mathbb{F}_2^{(n-k) \times (n-k)}\) with \(\pi(H) = (\|Q\) is invertible. Fix that permutation \(\pi\), which implies that \(\pi(H)\pi(e^*) = s\).

Then it is possible to apply a partial Gaussian elimination that corresponds to multiplying with an invertible matrix \(T\) from the left to obtain \(\tilde{H} = H \cdot \pi(H)\) and \(\tilde{s} = Ts\) such that \(\tilde{H} \cdot \pi(e^*) = \tilde{s}\) with \(\tilde{H} = (A_0 \ 0)\). Due to the fact that \(\pi(e^*) = (x^* || y^*)\) for some \(x^* \in \mathbb{F}_2^{k+\ell}\) with \(\text{wt}(x^*) = p\) and some \(y^* \in \mathbb{F}_2^{n-k-\ell}\) with \(\text{wt}(y^*) = \omega^* - p\), we therefore have the two equations \(Ax^* = [s]^{\ell}\) and \(Bx^* + y^* = [s]^{n-k-\ell}\).

In the equation \(Ax^* = [s]^{\ell}\) the only unknown is \(x^*\). Moreover, this is exactly a Random Subset Sum Problem in the group \(\mathbb{F}_2^n\) with \(n_{\text{RSSP}} := k + \ell\). In the statement of the theorem we assume that in time \(T(k, \ell, p, \varepsilon)\) with a probability of \(1 - O(2^{-\varepsilon n})\) we obtain the following lists. The list \(L_1\) consists of elements \(x_1\) with \(Ax_1 = s_1\) and \(s_1\) chosen uniformly at random by the algorithm. The list \(L_2\) consists of elements \(x_2\) with \(Ax_2 = [s]^{\ell} - s_1\). Moreover, with the given probability, the lists have the property that there is at least one \(x_1^* \in L_1\) and one \(x_2^* \in L_2\) such that \(x^* = x_1^* + x_2^*\) for the \(x^*\) fixed above.
From the elements in \( L_1 \) resp. \( L_2 \), two lists \( L'_1 \) resp. \( L'_2 \) with elements \( Bx_1 \) resp. \( Bx_2 + [s]_{n-k-\ell} \) are built. This means the lists consist of uniform and pairwise independent elements, due to the fact that \( H \) and therefore \( B \), as well as \( s \) and therefore \( \bar{s} \) are uniform.

Thus due to Theorem \ref{theo:subset_sum}, the list \( L_{\text{out}} \) contains the vector pair \((y_1^*, y_2^*) := (Bx_1^*, Bx_2^* + [s]_{n-k-\ell})\) with a probability of at least \( 1 - \Omega(2^{-\varepsilon n}) \). Due to the fact that it passes the Hamming distance check, we can easily compute the corresponding vector pair \((x_1^*, x_2^*)\). Now the vector \( \pi^{-1}((x_1^* + x_2^*)||(y_1^* + y_2^*)) \) solves the problem, since we have

\[
H(\pi^{-1}((x_1^* + x_2^*)||(y_1^* + y_2^*))) = T^{-1} \cdot (\frac{A}{B} \circ \vartheta) \cdot ((x_1^* + x_2^*)||(B(x_1^* + x_2^*) + [s]_{n-k-\ell})) = T^{-1}s = s.
\]

Moreover, notice that any \((y_1, y_2)\) with Hamming distance \( \omega - p \) such that the corresponding \((x_1, x_2)\) have a weight of \( p \) is a valid solution to the problem, since \( wt((x_1 + x_2)||(y_1 + y_2)) = \omega \).

The time complexity can be derived as follows. The creation of the two lists \( L_1, L_2 \) has a time complexity of \( T(k, \ell, p, \varepsilon) \), whereas the lists themselves have a size of \( \tilde{O}(2^{\mu n}) \) by definition of \( \mu \). Moreover, the Nearest Neighbor algorithm has a time complexity of \( \tilde{O}(2^{(\mu + 3\varepsilon)n} + 2^{(\psi + 2\varepsilon)n}) \), whereas the for-loops have a complexity of \( \tilde{O}(2^{\mu n}) \) with \( \varphi = H(\bar{\omega}) - H_{k+\varepsilon}(\rho) - H_{k-\varepsilon}(\bar{\omega} - \rho) \).

The overall time complexity is therefore as presented.

In the following, we want to instantiate the algorithm ColumnMatch with a slight adaption of the algorithm ClassicalRep applied to the group \( G = G_0 \times \ldots \times G_u = \{0\} \times \mathbb{F}_2^{\ell_1} \times \ldots \times \mathbb{F}_2^{\ell_u} \) for some \( u \in \mathbb{N} \) and \( \ell_1 + \ldots + \ell_u = \ell \) chosen accordingly to the number of representations. The adaption is that instead of performing the last step of the algorithm that consists of merging the two final lists to one output list, we instead output these two lists. Notice that the requirements of Theorem \ref{theo:subset_sum} directly follow from the correctness of the algorithm ClassicalRep. That is, for each pair \((x_1, x_2)\) of one element of the first list and one element of the second list, we have \( A(x_1 + x_2) = [s]^l \) and for each fixed \( x^* \) with \( wt(x^*) = p \) there is at least one \( x_1^* \in L_1 \) and one \( x_2^* \in L_2 \) such that \( x_1^* + x_2^* = x^* \). Since the last step is removed, the corresponding part of the time complexity can also be ignored. Notice that this part of the computation is replaced by the algorithm of the Nearest Neighbor algorithm.

### 6.2.3 Nearest Neighbor with MMT

In this section we want to use the derived results to combine the Nearest Neighbor technique with the ideas of the MMT decoding. That is, we use raw representations without introducing any additional ones.

The basic structure of the algorithm is illustrated in Fig. \ref{fig:columnmatch}. That is, a part of size \( \ell \) is reintroduced, which allows an exact matching on the first \( \ell \) components of \( \bar{s} \). This part is used to shorten the lists such that only expected one representation of a certain vector remains in these lists, following the ideas of Corollary \ref{cor:subset_sum}. For each of the two lists, this is done by enumerating two sub-lists of about \( 2^{C_0 n} \) elements, with a value of \( C_0 \) defined below.

The sub-lists are merged to two lists of size about \( 2^{C_1 n} \) elements, by picking only those of the additive combinations of the elements of the sub-lists that have a certain fixed value on the first \( \ell \) components of \( \bar{s} \).

This leaves us with two lists of elements, which already agree with the syndrome on the first \( \ell \) components. Using the \( B \)-part of the matrix, the approximate Subset Sum Problem is solved by using the algorithm for the Nearest Neighbor Problem developed in Chapter \ref{chap:mmt} that runs in time \( 2^{\psi n} \) for a value of \( \psi \) defined below.
6.2 Application of the Nearest Neighbor Technique

<table>
<thead>
<tr>
<th>n−k−ℓ</th>
<th>k+ℓ</th>
<th>n−k−ℓ</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>I</td>
</tr>
</tbody>
</table>

**Complexity:**
- Repetitions (outer loop): $2^{\varepsilon n}$
- List enumeration with $A$: $2^{C_0 n}$
- Merge with $A$: $2^{C_1 n}$
- Nearest neighbor with $B$: $2^{\psi n}$

Figure 6.9: MMT Nearest Neighbor Decoding

This leads to the following results for one level of representations.

**Corollary 76 (nnMMTDecoding).** Let $(H, s)$ be an instance of a $(n, k)$ Decoding Problem. Then for any $\varepsilon > 0$, any $p, \ell \in \mathbb{N}$, the algorithm APPROXDECODING instantiated with the sub-algorithm CLASSICALREP and one level can be used to solve the instance with overwhelming probability (over both the uniform choice of the input and the coins of the algorithm) in time $2^{0.109n}$ in the FDD setting and time $2^{0.0529n}$ in the HDD setting.

Let $\kappa := k/n$, $\rho := p/n$ and $\zeta := \ell/n$ and $\delta := \mathcal{H}^{-1}(1 - \kappa)$ such that $\tilde{\omega} := \delta$ in the FDD and $\tilde{\omega} := \delta/2$ in the HDD setting. Then the time complexity for the repetitions is $\tilde{O}(2^{\varepsilon n})$ with $\varphi = \mathcal{H}(\tilde{\omega}) - \mathcal{H}_{\kappa+\zeta}(\rho) - \mathcal{H}_{1-k-\zeta}(\tilde{\omega} - \rho)$.

Assume the parameter choice of the algorithm CLASSICALREP presented in Corollary 50, i.e. one level of representations. Due to the fact that we apply a special case of the algorithm CLASSICALREP that doesn’t perform the last step of computation, the group is simply $G = \{0\} \times \mathbb{F}_2^\ell$, where $\ell$ corresponds to the required number of representations. Due to the fact that we have a $(G, k+\ell, \frac{\rho}{\kappa+\zeta})$ Random Binary Subset Sum Problem, we obtain a time complexity of

$$\tilde{O}(2^{\varepsilon n} + 2^{C_1 n} + 2^{6\varepsilon n} \cdot 2^{C_0 n})$$

with $C_0 = \mathcal{H}_{\kappa+\zeta}(\frac{\rho}{2}) - \rho$ and $C_1 = \mathcal{H}_{\kappa+\zeta}(\frac{\rho}{2})/2$ such that the complexity of the removed last step doesn’t appear any more and is replaced by a nearest neighbor technique, which is illustrated in Fig. 6.9. Notice that the size of the output lists is $\tilde{O}(2^{2\mu n})$ with $\mu = C_0 + 6\varepsilon$. The Nearest Neighbor algorithm has a time complexity of

$$\tilde{O}(2^{(\mu+3\varepsilon)n} + 2^{(\psi+2\varepsilon)n})$$

with $\psi := (1 - \kappa - \zeta)(1 - \gamma)\left(1 - \mathcal{H}\left(\frac{\mathcal{H}^{-1}(1 - \frac{\mu}{1-k-\zeta}) - \frac{\rho}{2}}{1-\gamma}\right)\right)$,

with $\gamma := \frac{\tilde{\omega} - \rho}{1-k-\zeta}$. For arbitrary small $\varepsilon$ in the FDD case (Table 6.13), we obtain the following time complexity $\tilde{O}(2^{Cn})$ with $C = \varphi + \max\{C_0, C_1, \psi\}$ for several rates $\kappa$, with a maximal value at $\kappa \approx 0.439$. At this value, the complexity becomes $2^{0.109n}$. Notice that in this case the number of representations, the parameter $\ell$, as well as the parameter $p$ are identical. In Table 6.14, we see similar results for the HDD setting with a maximum of $2^{0.0529n}$ at $\kappa \approx 0.465$. 
6. Decoding Problem

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.439</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.056</td>
<td>0.085</td>
<td>0.102</td>
<td>0.109</td>
<td>0.108</td>
<td>0.100</td>
<td>0.086</td>
<td>0.065</td>
<td>0.037</td>
<td></td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.026</td>
<td>0.037</td>
<td>0.042</td>
<td>0.042</td>
<td>0.041</td>
<td>0.039</td>
<td>0.034</td>
<td>0.026</td>
<td>0.017</td>
<td>0.008</td>
</tr>
<tr>
<td>$C_0$</td>
<td>0.020</td>
<td>0.034</td>
<td>0.045</td>
<td>0.052</td>
<td>0.054</td>
<td>0.056</td>
<td>0.056</td>
<td>0.052</td>
<td>0.043</td>
<td>0.028</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.015</td>
<td>0.026</td>
<td>0.033</td>
<td>0.037</td>
<td>0.038</td>
<td>0.039</td>
<td>0.039</td>
<td>0.035</td>
<td>0.029</td>
<td>0.018</td>
</tr>
<tr>
<td>$\psi$</td>
<td>0.030</td>
<td>0.049</td>
<td>0.060</td>
<td>0.067</td>
<td>0.068</td>
<td>0.069</td>
<td>0.067</td>
<td>0.060</td>
<td>0.048</td>
<td>0.030</td>
</tr>
<tr>
<td>$100\rho$</td>
<td>0.995</td>
<td>1.640</td>
<td>2.017</td>
<td>2.204</td>
<td>2.229</td>
<td>2.212</td>
<td>2.078</td>
<td>1.787</td>
<td>1.348</td>
<td>0.747</td>
</tr>
</tbody>
</table>

Table 6.13: time complexity of \texttt{ApproxDecoding} with \texttt{ClassicalReps} ($u = 1$) for FDD

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.465</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.0243</td>
<td>0.0389</td>
<td>0.0478</td>
<td>0.0522</td>
<td>0.0529</td>
<td>0.0527</td>
<td>0.0497</td>
<td>0.0432</td>
<td>0.0333</td>
<td>0.0194</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.0132</td>
<td>0.0211</td>
<td>0.0258</td>
<td>0.0278</td>
<td>0.0278</td>
<td>0.0274</td>
<td>0.0250</td>
<td>0.0206</td>
<td>0.0146</td>
<td>0.0071</td>
</tr>
<tr>
<td>$C_0$</td>
<td>0.0093</td>
<td>0.0152</td>
<td>0.0192</td>
<td>0.0217</td>
<td>0.0226</td>
<td>0.0226</td>
<td>0.0226</td>
<td>0.0210</td>
<td>0.0176</td>
<td>0.0118</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.0064</td>
<td>0.0103</td>
<td>0.0128</td>
<td>0.0143</td>
<td>0.0147</td>
<td>0.0148</td>
<td>0.0145</td>
<td>0.0133</td>
<td>0.0111</td>
<td>0.0073</td>
</tr>
<tr>
<td>$\psi$</td>
<td>0.0112</td>
<td>0.0178</td>
<td>0.0220</td>
<td>0.0244</td>
<td>0.0252</td>
<td>0.0253</td>
<td>0.0247</td>
<td>0.0226</td>
<td>0.0188</td>
<td>0.0123</td>
</tr>
<tr>
<td>$100\rho$</td>
<td>0.3466</td>
<td>0.5309</td>
<td>0.6351</td>
<td>0.6826</td>
<td>0.6886</td>
<td>0.6841</td>
<td>0.6447</td>
<td>0.5654</td>
<td>0.4428</td>
<td>0.2656</td>
</tr>
</tbody>
</table>

Table 6.14: time complexity of \texttt{ApproxDecoding} with \texttt{ClassicalReps} ($u = 1$) for HDD

Once again, the extension to a two level algorithm doesn’t help to improve the result due to the fact that the bottom list sizes $C_1$ are already dominated.

6.2.4 Nearest Neighbor with BJMM

The one level result with additional ones, which is also based on the construction in Fig. 6.9, is as follows.

**Corollary 77** (nnBJMMDecoding1). Let $(H, s)$ be an instance of a $(n, k)$ Decoding Problem. Then for any $\varepsilon > 0$, any $p, \ell \in \mathbb{N}$, the algorithm \texttt{ApproxDecoding} instantiated with the subalgorithm \texttt{ClassicalRep}, with one level and making use of the special group structure of $\mathbb{F}_2^\ell$, can be used to solve the instance with overwhelming probability (over both the uniform choice of the input and the coins of the algorithm) in time $2^{\varphi n}$ in the FDD setting and time $2^{\varphi/2 n}$ in the HDD setting.

Let $\kappa := k/n$, $\rho := p/n$ and $\zeta := \ell/n$ and $\delta := \mathcal{H}^{-1}(1 - \kappa)$ such that $\tilde{\omega} := \delta$ in the FDD and $\tilde{\omega} := \delta/2$ in the HDD setting. Then the time complexity for the repetitions is $\mathcal{O}(2^{\varphi n})$ with $\varphi = \mathcal{H}(\tilde{\omega}) - \mathcal{H}_{k+\zeta}(\rho) - \mathcal{H}_{1-k-\zeta}(\tilde{\omega} - \rho)$.

Assume the parameter choice of the algorithm \texttt{ClassicalRep} presented in Corollary 57.

Notice that the group $\mathbb{F}_2^\ell$ is used as part of the group $G := \{0\} \times \mathbb{F}_2^\ell$ with the idea to abort the algorithm \texttt{ClassicalRep} before processing the final two lists. Due to the fact that we have a $(G, k+\ell, \rho_{n, \zeta})$ Random Binary Subset Sum Problem, we obtain a time complexity

$$\mathcal{O}(2^n + 2^{C_1 n} + 2^{6\varepsilon n} \cdot 2^{C_0 n} + 2^{6\varepsilon n} \cdot 2^{C_2 n})$$
6.2 Application of the Nearest Neighbor Technique

with \( C_0 = \mathcal{H}_{\kappa+\zeta}(\frac{\rho+\tau}{2}) - R_{0,1}, C_1 = \mathcal{H}_{\kappa+\zeta}(\frac{\rho+\tau}{2})/2 \) and \( R_{0,1} = \rho + \mathcal{H}_{\kappa+\zeta-\rho}(\frac{\tau}{2}) \) for some \( 0 \leq \tau \leq 2(\kappa + \zeta - \rho) \), with parameters explained in Fig. 6.9. Notice that once again the complexity of the merging of the top lists doesn’t appear any more, due to the fact that it is replaced by the Nearest Neighbor technique. Notice that the size of the output lists is once again \( \mathcal{O}(2^{\mu n}) \) with \( \mu = C_0 + 6\varepsilon. \) The time complexity of the final merging of the two top lists is

\[
\tilde{O}(2^{(\kappa+3\varepsilon)n} + 2^{(\psi+2\varepsilon)n}) \text{ with } \psi := (1 - \kappa - \zeta)(1 - \gamma) \left(1 - \mathcal{H} \left( \frac{\mathcal{H}^{-1}(1-\frac{\rho}{1-\gamma}) - \frac{\tau}{2}}{1-\gamma} \right) \right),
\]

with \( \gamma := \frac{\mu-\rho}{1-\kappa-\zeta}. \) For arbitrary small \( \varepsilon \) in the FDD case (Table 6.15), we obtain the following time complexity \( \tilde{O}(2^{Cn}) \) with \( C = \varphi + \max\{C_0, C_1, \psi\} \) for several rates \( \kappa \), with a maximal value at \( \kappa \approx 0.428. \) At this value, the complexity becomes \( 2^{0.101n}. \) Notice that the parameter \( \ell \) is chosen that it matches the group sizes \( |G_1| \), which matches the number of representations.

<table>
<thead>
<tr>
<th>( \kappa )</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.428</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C )</td>
<td>0.053</td>
<td>0.08</td>
<td>0.095</td>
<td>0.100</td>
<td>0.101</td>
<td>0.099</td>
<td>0.091</td>
<td>0.078</td>
<td>0.059</td>
<td>0.034</td>
</tr>
<tr>
<td>( \varphi )</td>
<td>0.013</td>
<td>0.020</td>
<td>0.023</td>
<td>0.022</td>
<td>0.021</td>
<td>0.020</td>
<td>0.019</td>
<td>0.014</td>
<td>0.012</td>
<td>0.011</td>
</tr>
<tr>
<td>( C_0 )</td>
<td>0.026</td>
<td>0.042</td>
<td>0.054</td>
<td>0.062</td>
<td>0.063</td>
<td>0.065</td>
<td>0.066</td>
<td>0.058</td>
<td>0.044</td>
<td>0.026</td>
</tr>
<tr>
<td>( C_1 )</td>
<td>0.040</td>
<td>0.060</td>
<td>0.073</td>
<td>0.079</td>
<td>0.079</td>
<td>0.080</td>
<td>0.078</td>
<td>0.066</td>
<td>0.049</td>
<td>0.028</td>
</tr>
<tr>
<td>( \psi )</td>
<td>0.040</td>
<td>0.060</td>
<td>0.073</td>
<td>0.079</td>
<td>0.079</td>
<td>0.080</td>
<td>0.078</td>
<td>0.076</td>
<td>0.049</td>
<td>0.028</td>
</tr>
<tr>
<td>( R_{0,1} )</td>
<td>0.053</td>
<td>0.077</td>
<td>0.091</td>
<td>0.097</td>
<td>0.096</td>
<td>0.095</td>
<td>0.089</td>
<td>0.089</td>
<td>0.075</td>
<td>0.054</td>
</tr>
<tr>
<td>100( \rho )</td>
<td>2.690</td>
<td>3.674</td>
<td>4.150</td>
<td>4.275</td>
<td>4.158</td>
<td>4.042</td>
<td>3.668</td>
<td>2.852</td>
<td>1.809</td>
<td>0.889</td>
</tr>
<tr>
<td>100( \tau )</td>
<td>0.820</td>
<td>1.189</td>
<td>1.378</td>
<td>1.445</td>
<td>1.434</td>
<td>1.405</td>
<td>1.290</td>
<td>1.070</td>
<td>0.760</td>
<td>0.412</td>
</tr>
</tbody>
</table>

Table 6.15: complexity of \textsc{ApproxDecoding} with \textsc{ClassicalReps} \((g = 2, u = 1)\) for FDD

In Table 6.16 we see results for the HDD setting with a maximum of \( 2^{0.0491n} \) at \( \kappa \approx 0.458. \)

<table>
<thead>
<tr>
<th>( \kappa )</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.458</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C )</td>
<td>0.0228</td>
<td>0.0363</td>
<td>0.0445</td>
<td>0.0485</td>
<td>0.0491</td>
<td>0.0489</td>
<td>0.0465</td>
<td>0.0399</td>
<td>0.0304</td>
<td>0.0176</td>
</tr>
<tr>
<td>( \varphi )</td>
<td>0.0104</td>
<td>0.0155</td>
<td>0.0177</td>
<td>0.0210</td>
<td>0.0178</td>
<td>0.0209</td>
<td>0.0238</td>
<td>0.0154</td>
<td>0.0088</td>
<td>0.0048</td>
</tr>
<tr>
<td>( C_0 )</td>
<td>0.0103</td>
<td>0.0178</td>
<td>0.0235</td>
<td>0.0245</td>
<td>0.0253</td>
<td>0.0253</td>
<td>0.0207</td>
<td>0.0227</td>
<td>0.0206</td>
<td>0.0124</td>
</tr>
<tr>
<td>( C_1 )</td>
<td>0.0125</td>
<td>0.0208</td>
<td>0.0268</td>
<td>0.0276</td>
<td>0.0314</td>
<td>0.0281</td>
<td>0.0227</td>
<td>0.0245</td>
<td>0.0217</td>
<td>0.0129</td>
</tr>
<tr>
<td>( \psi )</td>
<td>0.0125</td>
<td>0.0208</td>
<td>0.0268</td>
<td>0.0276</td>
<td>0.0314</td>
<td>0.0281</td>
<td>0.0227</td>
<td>0.0245</td>
<td>0.0217</td>
<td>0.0129</td>
</tr>
<tr>
<td>( R_{0,1} )</td>
<td>0.0146</td>
<td>0.0239</td>
<td>0.0302</td>
<td>0.0306</td>
<td>0.0345</td>
<td>0.0308</td>
<td>0.0247</td>
<td>0.0262</td>
<td>0.0228</td>
<td>0.0134</td>
</tr>
<tr>
<td>100( \rho )</td>
<td>0.5735</td>
<td>0.9153</td>
<td>1.1355</td>
<td>1.0762</td>
<td>1.2357</td>
<td>1.0367</td>
<td>0.7507</td>
<td>0.8034</td>
<td>0.6752</td>
<td>0.3471</td>
</tr>
<tr>
<td>100( \zeta )</td>
<td>1.4572</td>
<td>2.3817</td>
<td>3.0160</td>
<td>3.0576</td>
<td>3.4487</td>
<td>3.0736</td>
<td>2.4663</td>
<td>2.6147</td>
<td>2.2780</td>
<td>1.3309</td>
</tr>
<tr>
<td>100( \tau )</td>
<td>0.2190</td>
<td>0.3502</td>
<td>0.4358</td>
<td>0.4395</td>
<td>0.4887</td>
<td>0.4363</td>
<td>0.3459</td>
<td>0.3592</td>
<td>0.3052</td>
<td>0.1712</td>
</tr>
</tbody>
</table>

Table 6.16: complexity of \textsc{ApproxDecoding} with \textsc{ClassicalReps} \((g = 2, u = 1)\) for HDD

The time complexity is always optimized such that \( C_1 = \psi \), which means adding one level in the algorithm can help to improve the result.
This leads to the following main result of [MO15], which is based on Fig. 6.10. That is, the exact part of size \( \ell \) is subdivided into two parts. On the bottom level of representations, only the first part of size \( \ell_1 \) is fixed to a certain value. On the top level, also the remaining part with matrix \( A_2 \) is fixed. In the final step, once again the nearest neighbor technique is applied.

**Corollary 78** (nnBJMMDecoding2). Let \( (H, s) \) be an instance of a \((n, k)\) Decoding Problem. Then for any \( \varepsilon > 0 \), any \( p, \ell \in \mathbb{N} \), the algorithm APPROXDECODING instantiated with the subalgorithm CLASSICALREP, with two levels and making use of the special group structure of \( \mathbb{F}_2^{\ell} \), can be used to solve the instance with overwhelming probability (over both the uniform choice of the input and the coins of the algorithm) in time \( 2^{0.097n} \) in the FDD setting and time \( 2^{0.0473n} \) in the HDD setting.

Let \( \kappa := k/n \), \( \rho := p/n \) and \( \zeta := \ell/n \) and \( \delta := H^{-1}(1 - \kappa) \) such that \( \tilde{\omega} := \delta \) in the FDD and \( \tilde{\omega} := \delta/2 \) in the HDD setting. Then the time complexity for the repetitions is \( \tilde{O}(2^\varphi n) \) with \( \varphi = H(\tilde{\omega}) - H_{\kappa+\zeta}(\rho) - H_{1-\kappa-\zeta}(\tilde{\omega} - \rho) \).

Assume the parameter choice of the algorithm CLASSICALREP presented in Corollary 58. Notice that the group \( \mathbb{F}_2^{\ell} \) can be arbitrary split component-wise into groups \( G := \{0\} \times \mathbb{F}_2^{\ell_1} \times \mathbb{F}_2^{\ell_2} \) for any \( 0 \leq \ell_1 \leq \ell \) such that the sizes of the individual components of \( G \) can be arbitrary chosen, which is required by the corollary. Due to the fact that we have a \((G, k+\ell, \frac{\varphi}{\kappa+\zeta})\) Random Binary Subset Sum Problem, we obtain a time complexity

\[
\tilde{O}(2^{\varphi n} + 2^{C_2 n} + 2^{\varepsilon n} \cdot 2^{C_1 n} + 2^{\varepsilon n} \cdot 2^{C_2 n})
\]

with \( C_0 = 2H_{\kappa+\zeta}(\frac{\rho + \tau_0 + \tau_2}{4}) - R_{0,1} - R_{1,2}, C_1 = H_{\kappa+\zeta}(\frac{\rho + \tau_1 + \tau_2}{4}) - R_{1,2}, C_2 = H_{\kappa+\zeta}(\frac{\rho + \tau_1 + \tau_2}{4})/2 \), which are explained in Fig. 6.10. Additionally, we have representation exponents \( R_{0,1} = \rho + H_{\kappa+\zeta-\rho}(\frac{\varphi}{2}) \) and \( R_{1,2} = \frac{\rho + \tau_1}{2} + H_{\kappa+\zeta-\rho+\tau_1}(\frac{\tau_2}{4}) \) for some \( 0 \leq \tau_1 \leq 2(\kappa + \zeta - \rho) \) and some \( 0 \leq \tau_2 \leq 4(\kappa + \zeta - 2(\rho + \tau_1)) \) that determine the size of the \( \ell_1 \) and \( \ell_2 \) parts. Notice that once again the complexity of the merging of the top lists doesn’t appear any more, due to the fact that it is replaced by the Nearest Neighbor technique. The size of the output lists is once again \( \tilde{O}(2^\mu n) \) with \( \mu = H_{\kappa+\zeta}(\frac{\rho + \tau_2}{2}) - R_{0,1} + 6\varepsilon \). Therefore, the time complexity of the final merging of the two top lists is

\[
\tilde{O}(2^{(\mu+\varepsilon)n} + 2^{(\psi+2\varepsilon)n}) \text{ with } \psi := (1 - \kappa - \zeta)(1 - \gamma) \left(1 - H\left(H^{-1}(1 - \frac{\mu}{1-\gamma}) - \frac{\tilde{\omega}}{1-\gamma}\right)\right),
\]

with \( \gamma := \frac{\tilde{\omega} - \rho}{1-\kappa-\zeta} \). For arbitrary small \( \varepsilon \) in the FDD case (Table 6.17), we obtain the following time complexity \( \tilde{O}(2^{C n}) \) with \( C = \varphi + \max\{C_0, C_1, C_2, \mu, \psi\} \) for several rates \( \kappa \), with a maximal value at \( \kappa \approx 0.428 \). At this value, the complexity becomes \( 2^{0.097n} \).
6.2 Application of the Nearest Neighbor Technique

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.428</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.052</td>
<td>0.078</td>
<td>0.092</td>
<td>0.097</td>
<td>0.097</td>
<td>0.095</td>
<td>0.087</td>
<td>0.075</td>
<td>0.056</td>
<td>0.032</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.008</td>
<td>0.011</td>
<td>0.013</td>
<td>0.012</td>
<td>0.010</td>
<td>0.010</td>
<td>0.006</td>
<td>0.006</td>
<td>0.005</td>
<td>0.003</td>
</tr>
<tr>
<td>$C_0$</td>
<td>0.029</td>
<td>0.048</td>
<td>0.061</td>
<td>0.068</td>
<td>0.072</td>
<td>0.072</td>
<td>0.076</td>
<td>0.062</td>
<td>0.052</td>
<td>0.029</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.045</td>
<td>0.067</td>
<td>0.080</td>
<td>0.085</td>
<td>0.087</td>
<td>0.086</td>
<td>0.082</td>
<td>0.069</td>
<td>0.052</td>
<td>0.030</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.045</td>
<td>0.067</td>
<td>0.080</td>
<td>0.082</td>
<td>0.086</td>
<td>0.081</td>
<td>0.081</td>
<td>0.056</td>
<td>0.046</td>
<td>0.022</td>
</tr>
<tr>
<td>$\mu$</td>
<td>0.029</td>
<td>0.048</td>
<td>0.060</td>
<td>0.067</td>
<td>0.069</td>
<td>0.070</td>
<td>0.070</td>
<td>0.061</td>
<td>0.047</td>
<td>0.028</td>
</tr>
<tr>
<td>$\psi$</td>
<td>0.045</td>
<td>0.067</td>
<td>0.080</td>
<td>0.085</td>
<td>0.087</td>
<td>0.086</td>
<td>0.082</td>
<td>0.069</td>
<td>0.052</td>
<td>0.030</td>
</tr>
<tr>
<td>$R_{0,1}$</td>
<td>0.104</td>
<td>0.154</td>
<td>0.179</td>
<td>0.180</td>
<td>0.187</td>
<td>0.174</td>
<td>0.169</td>
<td>0.118</td>
<td>0.092</td>
<td>0.045</td>
</tr>
<tr>
<td>$R_{1,2}$</td>
<td>0.045</td>
<td>0.067</td>
<td>0.080</td>
<td>0.078</td>
<td>0.085</td>
<td>0.085</td>
<td>0.075</td>
<td>0.081</td>
<td>0.043</td>
<td>0.040</td>
</tr>
</tbody>
</table>

| $100\rho$ | 4.592 | 6.121 | 6.550 | 6.218 | 6.352 | 5.678 | 5.075 | 3.520 | 2.312 | 1.050 |
| $100\zeta$ | 10.36 | 15.33 | 17.83 | 17.98 | 18.67 | 17.36 | 16.80 | 11.80 | 9.131 | 4.430 |
| $100\tau_1$ | 2.203 | 3.324 | 3.875 | 3.803 | 3.964 | 3.578 | 3.446 | 2.178 | 1.677 | 0.714 |
| $100\tau_2$ | 0.485 | 0.924 | 1.290 | 1.246 | 1.496 | 1.252 | 1.697 | 0.523 | 0.771 | 0.165 |

Table 6.17: complexity of ApproxDecoding with ClassicalReps ($g = 2, u = 2$) for FDD

In Table 6.18 we see results for the HDD setting with a maximum of $2^{0.0473\kappa}$ at $\kappa \approx 0.451$.

<table>
<thead>
<tr>
<th>$\kappa$</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.451</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>0.0220</td>
<td>0.0350</td>
<td>0.0430</td>
<td>0.0470</td>
<td>0.0473</td>
<td>0.0472</td>
<td>0.0443</td>
<td>0.0385</td>
<td>0.0295</td>
<td>0.0172</td>
</tr>
<tr>
<td>$\varphi$</td>
<td>0.0064</td>
<td>0.0106</td>
<td>0.0128</td>
<td>0.0085</td>
<td>0.0144</td>
<td>0.0105</td>
<td>0.0118</td>
<td>0.0100</td>
<td>0.0065</td>
<td>0.0041</td>
</tr>
<tr>
<td>$C_0$</td>
<td>0.0139</td>
<td>0.0228</td>
<td>0.0284</td>
<td>0.0379</td>
<td>0.0330</td>
<td>0.0364</td>
<td>0.0324</td>
<td>0.0282</td>
<td>0.0230</td>
<td>0.0128</td>
</tr>
<tr>
<td>$C_1$</td>
<td>0.0157</td>
<td>0.0245</td>
<td>0.0302</td>
<td>0.0385</td>
<td>0.0330</td>
<td>0.0367</td>
<td>0.0326</td>
<td>0.0286</td>
<td>0.0230</td>
<td>0.0131</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.0157</td>
<td>0.0240</td>
<td>0.0274</td>
<td>0.0363</td>
<td>0.0303</td>
<td>0.0325</td>
<td>0.0271</td>
<td>0.0220</td>
<td>0.0171</td>
<td>0.0084</td>
</tr>
<tr>
<td>$\mu$</td>
<td>0.0130</td>
<td>0.0209</td>
<td>0.0265</td>
<td>0.0346</td>
<td>0.0296</td>
<td>0.0335</td>
<td>0.0301</td>
<td>0.0268</td>
<td>0.0219</td>
<td>0.0127</td>
</tr>
<tr>
<td>$\psi$</td>
<td>0.0157</td>
<td>0.0245</td>
<td>0.0302</td>
<td>0.0385</td>
<td>0.0330</td>
<td>0.0367</td>
<td>0.0326</td>
<td>0.0286</td>
<td>0.0230</td>
<td>0.0131</td>
</tr>
<tr>
<td>$R_{0,1}$</td>
<td>0.0331</td>
<td>0.0495</td>
<td>0.0567</td>
<td>0.0732</td>
<td>0.0606</td>
<td>0.0654</td>
<td>0.0543</td>
<td>0.0444</td>
<td>0.0342</td>
<td>0.0172</td>
</tr>
<tr>
<td>$R_{1,2}$</td>
<td>0.0157</td>
<td>0.0235</td>
<td>0.0246</td>
<td>0.0341</td>
<td>0.0276</td>
<td>0.0283</td>
<td>0.0215</td>
<td>0.0153</td>
<td>0.0112</td>
<td>0.0037</td>
</tr>
<tr>
<td>$10^2\rho$</td>
<td>1.0283</td>
<td>1.4366</td>
<td>1.6119</td>
<td>2.0783</td>
<td>1.5646</td>
<td>1.7516</td>
<td>1.3747</td>
<td>1.0885</td>
<td>0.7911</td>
<td>0.3718</td>
</tr>
<tr>
<td>$10^2\tau_1$</td>
<td>0.6944</td>
<td>1.0079</td>
<td>1.0979</td>
<td>1.4094</td>
<td>1.1394</td>
<td>1.2036</td>
<td>0.9531</td>
<td>0.7366</td>
<td>0.5405</td>
<td>0.2444</td>
</tr>
<tr>
<td>$10^2\tau_2$</td>
<td>0.3227</td>
<td>0.4969</td>
<td>0.4558</td>
<td>0.7000</td>
<td>0.5689</td>
<td>0.5319</td>
<td>0.3604</td>
<td>0.2066</td>
<td>0.1416</td>
<td>0.0152</td>
</tr>
</tbody>
</table>

Table 6.18: complexity of ApproxDecoding with ClassicalReps ($g = 2, u = 2$) for HDD

Whereas in the FDD case it might make sense to add another level for very small values of $\kappa$, in the HDD setting it doesn’t help. This is due to the fact that in both cases for most of the $\kappa$ the bottom level complexity $C_2$ is dominated by the other three complexities $C_1$ and $\psi$, which are chosen to be identical.
6.3 Conclusion

In this chapter, a novel technique for decoding random linear codes is introduced, making use of an algorithm for approximate merging introduced in Chapter 2. In its basic form `nnBasicDecoding`, the algorithm gets rid of the parameter $\ell$, which was introduced by Stern to allow an exact merging.

One question that arises is if it is really optimal to get rid of the whole $\ell$ at once. That is, maybe a hybrid of the approach by Stern and the new approach might lead to better results. However, numerical optimization suggests that $\ell = 0$ is indeed optimal. It remains an open question to study this more rigorously.

The basic technique is combined with the representation technique introduced in Chapter 3. The algorithm is modified such that the Nearest Neighbor algorithm is only used on the top level, merging the final two lists. On the remaining levels, the parameter $\ell$ is reintroduced. This enables to use the exact representation technique without modification. However, one might also try to adapt the representation technique such that the syndrome doesn’t have to be known exactly, but allows a small additive error. This might allow to get rid of the parameter $\ell$ once and for all, reducing the cost for a good permutation $\pi$. It remains an open question to find such a modification of the representation technique and to study if the removing of $\ell$ leads to better results.

In Chapter 3, the main improvement for the Subset Sum Problem is an improved algorithm for the Consistency Problem introduced in Chapter 2. As already shortly discussed at the end of Chapter 4, a combination of the Nearest Neighbor Problem and the zeroAND Problem might lead to better results. Already interesting is the question in how far a faster solution of the zeroAND Problem without the application of the Nearest Neighbor Problem algorithm might lead to better algorithms for the Decoding Problem.

Surprisingly, a sole application of the zeroAND Problem in algorithms with at least two levels doesn’t seem to lead to notably better results. The reason is that in the optimization, the parameters are chosen such that the problem effectively doesn’t appear, i.e. there are as good as no inconsistent vectors. The last question is that of a combination of these techniques. Very similarly, this doesn’t seem to lead to notable improvements. The reason is once again that the small number of inconsistent vectors doesn’t seem to lead to any advantage filtering on this part, making it more advantageous to filter on the part of the Nearest Neighbor Problem. An open question is if this is really inherent or maybe a problem of the numerical optimization that might only output a local minimum, missing a global one.
Chapter 7

Discrete Logarithm Problem

After roughly forty years of academic research, the Discrete Logarithm Problem in cyclic groups \( G \) is still one of the most central sources for building cryptographic schemes: given a generator \( \alpha \in G \) and a \( \beta \in G \), the problem asks to find an integer \( x \) such that \( \alpha^x = \beta \). Although there are polynomial time quantum algorithms \( [\text{Sho94}] \) as well as sub-exponential classical algorithms \( [\text{Adl79}] \) known for many interesting groups, the problem stays hard in general. An important result by Shoup \( [\text{Sho97}] \) shows the hardness in a generic group, i.e. a group in which the group elements are modeled as uniformly random elements. Concretely, if \(| G | \) is prime, the problem can’t be solved faster than in \( \sqrt{|G|} \) steps, matching upper bounds by Shanks \( [\text{Sha71}] \) and Pollard \( [\text{Pol78}] \), who present algorithms that achieve this time complexity. The former simply implements the algorithm Mitm that we have seen in Chapter 3 for the Subset Sum Problem, whereas the latter is a cycle-finding based technique that achieves the same time complexity, but can solve the problem with only polynomial space complexity.

However, if \(| G | = N \) is composite with prime factorization \( N = \prod_{i=1}^{k} p_i^{e_i} \), there is a more efficient algorithm proposed by Silver, Pohlig and Hellman \( [\text{PH78}] \), denoted SPH in the following. The idea is that the discrete logarithm \( x \) can be computed modulo each of the \( p_i \) individually with one of the proposed meet-in-the-middle techniques. Each of these individual logarithms can then be lifted efficiently to discrete logarithms modulo \( p_i^{e_i} \). Finally, due to the fact that the moduli are pairwise coprime, the \( k \) parts can be composed to a discrete logarithm modulo \( N \) with the Chinese Remainder Theorem (CRT). Due to the efficiency of the lifting and the CRT, the complexity of this approach is \( \max_i \{ \sqrt{p_i} \} \).

The running time can also be improved, if the discrete logarithm \( x \) itself is of a special form. With the help of Pollard’s kangaroo method \( [\text{Pol78}] \), the problem can be solved in time \( \sqrt{|G|^\gamma} \), whenever the \( 0 \leq x \leq |G|^\gamma \) for some \( 0 \leq \gamma \leq 1 \). There are also algorithms for the case when the bit representation of \( x \) has a small Hamming weight. If \( x \) is an \( n \)-bit integer with \( \delta n \) ones for some \( 0 \leq \delta \leq 1 \), the meet-in-the-middle based algorithms of Heiman-Odlyzko \( [\text{Hei92}] \), Coppersmith \( [\text{Cop79}] \) and Stinson \( [\text{Sti02}] \) solve the problem in time \( \sqrt{|G|^\gamma H(\delta)} \).

If the group order \( N = p \cdot q \) with primes \( p < q \) is composite and the \( 0 \leq x \leq |G|^\gamma \) is small, an algorithm of van Oorschot and Wiener \( [\text{vOW96}] \) solves the problem in time \( \sqrt{p} + \sqrt{|G|^\gamma} / \sqrt{p} \). That is, the discrete logarithm is firstly computed modulo \( p \) with a standard meet-in-the-middle method in time \( \sqrt{p} \). This allows us to learn \( x_0 \) such that \( x = x_1 \cdot p + x_0 \) for some unknown \( 0 \leq x_1 \leq x/p \). The remaining \( x_1 \) can then be easily learned with Pollard’s kangaroo method in time \( \sqrt{x/p} \). Notice that this technique always outperforms the SPH technique, since in the time complexity \( \sqrt{q} \) gets improved to \( \sqrt{x/p} \). However, if \( p \) is too large, i.e. \( p > x \), the method...
is clearly worse than applying Pollard’s kangaroo method directly, which has a time complexity of $\sqrt{|G|}$. This method can be easily extended to groups that split into more than two primes, allowing additional improvements.

All possible methods of combining prime/composite order groups with standard/small/small weight $x$ are illustrated in Fig. 7.1, one of which isn’t explained yet. The combination of a composite order group with a small weight $x$ was firstly considered in [PH78] and is the main contribution of this chapter. This algorithm can be seen as a generalization of both the meet-in-the-middle algorithms of Heiman-Odlyzko and Coppersmith as well as the Silver-Pohlig-Hellman algorithm, similarly as it was done for small $x$ in [vOW96].

The novel algorithm also makes use of the Silver-Pohlig-Hellman algorithm as a subroutine. However, our computation of the second part is way more challenging than in the algorithm of van Oorschot and Wiener. Notice that the property of a small Hamming weight discrete logarithm does not transfer to its Chinese Remainder representation and vice versa. Nevertheless, we are able to show that parts of the Chinese Remainder representation automatically reduce the search space for small weight discrete logarithms.

In general, finding algorithms for small Hamming weight appears to be a harder problem than finding algorithms for small size, e.g. for polynomial equations there is an efficient algorithm that finds all small size integer roots due to Coppersmith [Cop97], but there is no analogue known for small Hamming weight roots.

Let us describe the algorithm in the special case of a group $G$ with $|G| = p \cdot q$ for primes $p < q$. In the proof of our main theorem, we then show how to extend the result to groups with arbitrary factorization. Let $n, n_p, n_q$ denote the bit sizes of $p \cdot q$, $p$, $q$ respectively, such that roughly $n = n_p + n_q$. In the first step, the discrete logarithm is learned modulo $p$ with a standard meet-in-the-middle method in time $\sqrt{p}$. This technique roughly allows to learn the $n_p$ least significant bits of the bit representation of $x$. However, the remaining $n_q$ bits remain unknown and have an expected Hamming weight (over the uniformly random choice of $x$) of $\delta \cdot n_q$. Therefore, these remaining bits can be learned with a meet-in-the-middle technique in time $\sqrt{\delta \cdot n_q} \approx \sqrt{q}^{\mathcal{H}(\delta)}$.

This method is inspired by the representation technique [HJ10] for the Subset Sum Problem in Chapter 4. In this algorithm, the unknown $x \in \{0, 1\}^n$ is represented as a sum (in $\mathbb{Z}^n$) of vectors $x_1, x_2 \in \{0, 1\}^n$. However in this case the addition $x_1 + x_2$ is in $\mathbb{Z}$, i.e. we allow carry bits that allow for new kinds of representations. Our algorithm is also in the spirit of Stern’s algorithm [Ste88] for the Decoding Problem in Chapter 6. In this algorithm, one part of the unknown error vector is obtained combinatorially, whereas the remaining bits are computed efficiently with simple linear algebra. In our case, the $n_q$ bits are obtained combinatorially, whereas the remaining $n_p$ bits are computed by learning the discrete logarithm modulo $p$. 

<table>
<thead>
<tr>
<th>prime order</th>
<th>composite order</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sqrt{</td>
<td>G</td>
</tr>
<tr>
<td>Sha71</td>
<td>PH78</td>
</tr>
<tr>
<td>$\sqrt{</td>
<td>G</td>
</tr>
<tr>
<td>Pol78</td>
<td>$\sqrt{p} + \sqrt{q}^{\mathcal{H}(\delta)}$</td>
</tr>
</tbody>
</table>

Figure 7.1: Overview

<table>
<thead>
<tr>
<th>standard $x$</th>
<th>small $x$</th>
<th>small weight $x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>prime order</td>
<td>composite order</td>
<td></td>
</tr>
<tr>
<td>$\sqrt{</td>
<td>G</td>
<td>}$</td>
</tr>
<tr>
<td>Sha71</td>
<td>PH78</td>
<td></td>
</tr>
<tr>
<td>$\sqrt{</td>
<td>G</td>
<td>}^{\gamma}$</td>
</tr>
<tr>
<td>Pol78</td>
<td>$\sqrt{p} + \sqrt{q}^{\mathcal{H}(\delta)}$</td>
<td>MO14</td>
</tr>
</tbody>
</table>
7.1 Known Generic Algorithm

In this section, we want to present a result by Heiman and Odlyzko [Hei92] that allows to solve the Discrete Logarithm Problem in a group $G$ with arbitrary order. That is, given a generator $\alpha \in G$ and an arbitrary $\beta \in G$, find an $x \in \mathbb{Z}_{|G|}$ s.t. $\alpha^x = \beta$. Furthermore, assume that $|G|$ is an $n$-bit number.

This problem can be easily solved by reformulating it as a Subset Sum Problem of Chapter 3. Then, the algorithm Mitm can be used to solve it in time roughly $\sqrt{|G|}^{H(\delta)}$. The problem is simply defined over the same group $G$ and with the $n$ defined as above. The target is $s = \beta$ and the $a \in G^n$ is chosen such that $a_i = \alpha^{2^{i-1}}$ for all $1 \leq i \leq n$. The weight distribution $w$ is chosen such that $w(1) = \delta$ and $w(0) = 1 - \delta$.

![Figure 7.2: Splitting](image)

The algorithm Mitm searches for a subset $\mathcal{I} \subseteq [n]$ such that the weight of $x$ splits evenly on both the bit positions of $\mathcal{I}$ and the bit positions of $[n] \setminus \mathcal{I}$, as illustrated in Fig. 7.2. Then it enumerates all (vectors) $x_1 \in \mathbb{Z}_{\mathcal{I}}^{n/2}[w] \times 0^{n/2}$ in a list $\mathcal{L}$ and searches for a match with all $x_2 \in 0^{n/2} \times \mathbb{Z}_{[n] \setminus \mathcal{I}}^{n/2}[w]$. Due to the choice of $a$ with the powers of two in the exponent, once these $x_1, x_2$ are found, they can be easily combined to the discrete logarithm $x$. Due to Theorem 27, the algorithm Mitm with input $(G, n, w, a, s)$ solves the Subset Sum Problem in time $2^{H(w)n/2}$, which is exactly $\sqrt{|G|}^{H(\delta)}$.

7.2 New Generic Algorithm

![Figure 7.3: New splitting](image)
In this section, we want to describe our new algorithm that requires a composite order group. Notice that the following is mostly taken from [MO14].

Let \( \alpha \) generate a composite order group \( G \simeq G_1 \times G_2 \) with \( |G| = N \), \( |G_1| = N^\tau \) and \( |G_2| = N^{1-\tau} \) for some \( 0 < \tau < 1 \). In general, there might be several ways to decompose \( G \) as \( G_1 \times G_2 \). Firstly, we describe our algorithm for a fixed decomposition. In Sect. 7.3, we minimize the running time by adjusting the decomposition accordingly.

Our new algorithm combines the Silver-Pohlig-Hellman idea with a subsequent \textit{meet-in-the-middle} approach of the previous section. However, in comparison to the previous technique, we consider the splitting of Fig. 7.3. Besides the fact that we have a splitting into three parts, the main difference is that the \( t \) bits of the right hand parts have to the consecutive, i.e. represent a \( t \)-bit number. This means, we can’t simply search for a subset \( I \) that leads to a good distribution of the weight.

Instead, we have to assume that \( x \) is chosen uniformly at random and have to compute the probability that the relative Hamming weight on the \( n-t \) most significant bits of \( x \) is in the interval \( (\delta - \varepsilon, \delta + \varepsilon) \) for some small \( \varepsilon > 0 \). In this way, we can ensure that \( x \) splits in \( x_1 + x_2 \) with appropriate Hamming weights in this interval with a probability that is exponentially close to 1, while only slightly increasing the running time.

In the following, we always assume w.l.o.g. that we know the factorization of the group order \( N \). Notice that this does not limit the applicability of our generic algorithm, since its running time is exponential in the bit-length of \( N \) anyway, whereas the factorization of \( N \) can be computed in sub-exponential time [LP31] [MB75].

\begin{algorithm}
\caption{DLOG}
\begin{algorithmic}[1]
\State \textbf{Input:} \(|G_1|, |G_2|, \alpha, \beta, \delta, \varepsilon\)
\State \textbf{Output:} \(x\)
\State \(n \leftarrow \lceil \log_2(|G_1|) \rceil\)
\State \(t \leftarrow \lceil \log_2(|G_2|) \rceil\)
\State \(x' \leftarrow \text{SPH}(|G_1|, \alpha^{\lfloor |G_2| \rfloor}, \beta^{|G_2|}) \quad \triangleright \text{Use Silver-Pohlig-Hellman to get } x \mod |G_1|\)
\State \(M \leftarrow |G_1| \quad \triangleright \lfloor \cdot \rfloor_M \text{ denotes the smallest non-negative representative mod } M\)
\State \(S_1 \leftarrow \{\}\)
\ForAll{0 \leq v_1 < 2^{(n-t)/2} \text{ with } (\delta - \varepsilon)\frac{n-t}{2} \leq \text{wt}(v_1) \leq (\delta + \varepsilon)\frac{n-t}{2} \text{ do}}
\State \(S_1 \leftarrow S_1 \cup \{v_1 \cdot 2^{(n+t)/2} + [-v_1 \cdot 2^{(n+t)/2}]_M\}\)
\EndFor
\State \(S_2 \leftarrow \{\}\)
\ForAll{0 \leq v_2 < 2^{(n-t)/2} \text{ with } (\delta - \varepsilon)\frac{n-t}{2} \leq \text{wt}(v_2) \leq (\delta + \varepsilon)\frac{n-t}{2} \text{ do}}
\State \(S_2 \leftarrow S_2 \cup \{v_2 \cdot 2^t + [x' - v_2 \cdot 2^t]_M\}\)
\EndFor
\State \(S_2 \leftarrow S_2 \cup \{v_2 \cdot 2^t + [x' - v_2 \cdot 2^t]_M - M\}\)
\State \(S_2 \leftarrow S_2 \cup \{v_2 \cdot 2^t + [x' - v_2 \cdot 2^t]_M + M\}\)
\State Create a list \( L \) with entries \((\alpha^{x_1}, x_1)\) for all \( x_1 \in S_1 \), sort by its first component
\ForAll{\( x_2 \in S_2 \) do}
\State Binary search for a \((\alpha^{x_1}, x_1) \in L \) such that \( \alpha^{x_1} = \beta/\alpha^{x_2} \)
\State \Return\( x_1 + x_2 \) if there is a match
\EndFor
\Return\ no match
\end{algorithmic}
\end{algorithm}
Let us first define \( \lceil x' \rceil_M = \lfloor x \rfloor_M \), where \( \lfloor \cdot \rfloor_M \) describes the smallest non-negative representative of some number modulo \( M := |G_1| \). Afterwards, we apply a *meet-in-the-middle* technique on the bigger subgroup \( G_2 \), where we cut down the search space by the amount of information that is provided by \( x' \). More precisely, knowing only \( n - t \) bits of the discrete logarithm \( x \) (i.e. \( v_1 \) and \( v_2 \) in Fig. 7.3), it is possible to compute the remaining \( t \) consecutive bits \( w \) in polynomial time with the help of \( x' = x \mod M \).

Let us fix some useful notation. We denote \( x' = \lfloor x' \rfloor_M = \lfloor x \rfloor_M \), where \( \lfloor \cdot \rfloor_M \) describes the smallest non-negative representative of some number modulo \( M \), i.e. in \( [0, M) \). Let us choose \( t \) such that \( 2^{t-1} < M \leq 2^t \). Then \( w \) is either \( \lfloor w \rfloor_M := \lfloor x' - v_1 \cdot 2^{(n-t)/2} - v_2 \cdot 2^t \rfloor_M \) or \( \lceil w \rceil_M + M \). For any \( x \in \mathbb{N} \) we denote by \( \text{wt}(x) \) the Hamming weight of the binary representation of \( x \).

In lines 8 through 10, a list \( S_1 \) is computed by enumerating all values of \( v_1 \) (see Fig. 7.3). We show that the remaining \( t \)-bit value \( w_1 \) can be uniquely obtained from \( v_1 \). Similarly, in lines 11 through 15 we obtain only three possible values for \( w_2 \) for each value of \( v_2 \). In total, it is sufficient to perform a *meet-in-the-middle* algorithm on only \( n - t \) bits instead of the full \( n \) bits of the binary representation of \( x \). Eventually, the two sets \( S_1, S_2 \) are searched for a match, which gives the discrete logarithm \( x \), since we show that with overwhelming probability there is always some \( x_1 \in S_1 \) and \( x_2 \in S_2 \) that sum up to \( x \).

### Theorem 79

Let \( \alpha \) be a generator of a cyclic group \( G \simeq G_1 \times G_2 \) of known order \( N \), where \( N \) has bit-size \( n \). Let \( \delta \in (0, \frac{1}{2}) \) and let \( x \) be sampled uniformly at random from all elements of \( \mathbb{Z}_N \) with Hamming weight \( \delta n \). Let \( \beta := \alpha^x \) and \( p \) be the largest prime factor of \( |G_1| \). Then for any \( \varepsilon > 0 \) with \( \delta + \varepsilon \leq \frac{1}{2} \) on input \( (|G_1|, |G_2|, \alpha, \beta, \delta) \) algorithm DLOG outputs \( x \) with probability at least \( 1 - \frac{4(n+1)}{|G_2|^2} \) in time \( \tilde{O} \left( \sqrt{p} + \sqrt{|G_2|}^{H(\delta+\varepsilon)} \right) \) and space \( \tilde{O} \left( \sqrt{|G_2|}^{H(\delta+\varepsilon)} \right) \).

**Proof.** Let us first define \( M := |G_1|, n := \lfloor \log_2(N) \rfloor, t := \lceil \log_2(M) \rceil \) and \( x' := \lfloor x \rfloor_M \) as in DLOG. Recall that \( \lceil \cdot \rceil_M \) denotes the least non-negative representative modulo \( M \), and \( \text{wt}(\cdot) \) denotes the Hamming weight of the binary representation. For simplicity, we ignore rounding problems like with \( (n-t)/2 \), since they can easily be resolved without affecting the asymptotic running time.

Similarly to the standard *meet-in-the-middle* approach, in DLOG we decompose the unknown \( x = v_1 \cdot 2^{(n+1)/2} + v_2 \cdot 2^t + w \) for some \( 0 \leq v_1, v_2 < 2^{(n-t)/2} \) and \( 0 \leq w < 2^t \), as illustrated in Fig. 7.3. Moreover, we require that both \( v_1, v_2 \) have some Hamming weight in the interval \( [(\delta - \varepsilon)(n-t)/2, (\delta + \varepsilon)(n-t)/2] \). The proof is organized as follows. Firstly, we show that any random \( x \) possesses the correct weights for \( v_1, v_2 \) with overwhelming probability. Secondly, we show that for the correct weights, DLOG always outputs \( x \). Thus, DLOG is of Las Vegas type. Its output is always correct, but DLOG fails on an exponentially small fraction of all input instances.

Let \( x \in \mathbb{Z}_N \) be chosen uniformly at random with Hamming weight \( \text{wt}(x) = \delta n \). We show that \( (\delta - \varepsilon)(n-t)/2 \leq \text{wt}(v_1), \text{wt}(v_2) \leq (\delta + \varepsilon)(n-t)/2 \) holds with a probability that is at least \( 1 - 4(n+1)/|G_2|^2 \). Let \( (x_{n-1}, \ldots, x_0) \) denote the binary representation of \( x \), and let \( X_i \) be a random variable for \( x_i \). Simplifying our proof, we assume that \( x \) is sampled by \( n \) independent Bernoulli trials with \( \mathbb{P}[X_i = 1] = \delta \) for all bits \( i = 0, \ldots, n-1 \). Notice that sampling \( x \) in this manner and rejecting all \( x \) that have an incorrect Hamming weight gives the same distribution as sampling \( x \) uniformly at random from all \( x \) with Hamming weight \( \delta n \).

Let \( I \subseteq \{0, \ldots, n-1\} \) with \( |I| = (n-t)/2 \) be some index set. Let \( X = \sum_{i=0}^{n-1} X_i \) be the Hamming weight of \( x \), and let \( Y = \sum_{i \in I} X_i \) be the Hamming weight of coordinates \( I \). In order
to estimate DLOG’s failure probability, we compute
\[ P \left[ |Y - \delta(n-t)/2| > \varepsilon(n-t)/2 \mid X = \delta n \right], \]
which is the probability that the Hamming weight on the I-bits of \( x \) is not in the range between \((\delta - \varepsilon) \cdot (n-t)/2\) and \((\delta + \varepsilon) \cdot (n-t)/2\), under the condition that \( x \) has the correct Hamming weight. Notice that \( P[X = \delta n] \geq P[X = i] \) for any \( i \neq \delta n \). Since \( 0 \leq X \leq n \), we get \( 1 = \sum_{i=0}^{n} P[X = i] \leq (n+1) \cdot P[X = \delta n] \) and thus \( P[X = \delta n] \geq \frac{1}{n+1} \). This implies
\[ P \left[ |Y - \delta(n-t)/2| > \varepsilon(n-t)/2 \mid X = \delta n \right] \leq (n+1) \cdot P \left[ |Y - \delta(n-t)/2| > \varepsilon(n-t)/2 \right]. \]

An application of Hoeffding’s inequality yields
\[ (n+1) \cdot P \left[ |Y - \delta(n-t)/2| > \varepsilon(n-t)/2 \right] \leq 2(n+1)2^{-\varepsilon^2(n-t)} \leq \frac{2(n+1)}{|G_2|^2}. \]

Hence, we obtain a probability of at most \( 2(n+1)/|G_2|^2 \) that the relative Hamming weight for one of \( v_1, v_2 \) is incorrect. By the union bound, the probability that \( v_1 \) or \( v_2 \) have incorrect weight is bounded by \( 4(n+1)/|G_2|^2 \).

It remains to show that for correct Hamming weight of \( v_1, v_2 \) DLOG always succeeds in computing \( x \). For that, it suffices to show the existence of \( (x_1, x_2) \in S_1 \times S_2 \) with \( x_1 + x_2 = x \).

We split \( x \) in three parts \( v_1, v_2, w \) with \( x = v_1 \cdot 2^{(n+t)/2} + v_2 \cdot 2^t + w \) (see Fig. 7.3). Denote
\[
x_1 := v_1 \cdot 2^{(n+t)/2} + w_1, \quad x_2 := v_2 \cdot 2^t + w_2 \quad 0 \leq v_1, v_2 < 2^{(n-t)/2} \quad 0 \leq w_1, w_2 < 2^t.
\]

In DLOG, we enumerate a list \( S_1 \) of all possible \( v_1 \) and compute for each \( v_1 \) a corresponding \( w_1 \). We proceed with \( v_2 \) and their corresponding \( v_2 \) analogously.

In \( S_1 \) we choose to fix \( x_1 = 0 \mod M \) — the value 0 could be any constant in \( \mathbb{Z}_M \). Therefore, we compute \( w_1 = -v_1 \cdot 2^{(n+t)/2} \mod M \) and store the corresponding integer
\[
x_1 := v_1 \cdot 2^{(n+t)/2} + [-v_1 \cdot 2^{(n+t)/2}]_M.
\]

Notice that there always exists a \( 0 \leq w_1 < 2^t \) with \( w_1 = -v_1 \cdot 2^{(n+t)/2} \mod M \), since \( M \leq 2^t \) by the choice of \( t \).

Since we have to ensure \( x_1 + x_2 = x \), we require \( x_1 + x_2 = x' \mod M \) and thus \( x_2 = x' \mod M \) by our choice of \( x_1 \). This in turn implies \( w_2 = x' - v_2 \cdot 2^t \mod M \). By construction, we obtain
\[
x_1 + x_2 = v_1 \cdot 2^{(n+t)/2} + [-v_1 \cdot 2^{(n+t)/2}]_M + v_2 \cdot 2^t + [x' - v_2 \cdot 2^t]_M = x \mod M.
\]

However, this does not necessarily imply \( x = x_1 + x_2 \) over \( \mathbb{Z} \). Especially, we have to guarantee \( w_1 + w_2 = w \). Notice that by definition \( w < 2^t \) and \( M \leq 2^t < 2M \). Since \( 0 \leq [w_1]_M, [w_2]_M < M \) we have \( 0 \leq [w_1]_M + [w_2]_M < 2M \).

If either \( 0 \leq [w_1]_M + [w_2]_M < M \) and \( w < M \) (case I+I in Fig. 7.4) or \( M \leq [w_1]_M + [w_2]_M < 2M \) and \( M \leq w \) (case II+II in Fig. 7.4), we are done. If \( 0 \leq [w_1]_M + [w_2]_M < M \) and \( M \leq w \) (case I+II in Fig. 7.4), we have to add \( M \) to \( [w_1]_M + [w_2]_M \). In the remaining case II+I, we have to subtract \( M \) from \( [w_1]_M + [w_2]_M \).

Thus, \([w_1]_M + [w_2]_M + kM = w \) holds for some \( k \in \{-1, 0, 1\} \). In DLOG we choose \( x_1 = v_1 \cdot 2^{(n+t)/2} + [w_1]_M \in S_1 \) and \( x_2 = v_2 \cdot 2^t + [w_2]_M + kM \in S_2 \) for all \( k \in \{-1, 0, 1\} \). For the correct \( k \), we obtain \( x_1 + x_2 = x \), as desired. Thus DLOG outputs the discrete logarithm \( x \).
7.2 New Generic Algorithm

![Figure 7.4: ±M](image)

It remains to show the time and space complexities. SPH takes time $\tilde{O}(\sqrt{p})$ with only polynomial memory consumption, when using Pollard’s Rho Method as a subroutine. Notice that the complexities of the for-loops in steps 9 and 12 of DLOG are dominated by the time to enumerate and store those $v_i$ with largest weight $(\delta + \varepsilon)\frac{2^n-t}{2}$. Thus, due to the fact that binary search and sorting is efficient, our algorithm has time and space complexity $\tilde{O}\left(\sqrt{|G|^{H(\delta+\varepsilon)}}\right)$.

Let us finish this section with some remarks. The algorithm DLOG is by definition restricted to small Hamming weight $0 < \delta \leq \frac{1}{2}$. Symmetrically, DLOG can be applied to large Hamming weight $\frac{1}{2} \leq \delta < 1$ by transforming the discrete logarithm instance to $\tilde{\beta} := \alpha^{2^n-1}/\beta$. This transforms $x$ to $\tilde{x} = (2^n - 1) - x$ with Hamming weight $(1 - \delta) \cdot n$.

We introduce $\varepsilon$ to ensure that the Hamming weight of $v_1$ and $v_2$ lies within some $\varepsilon$-strip around its expectation with overwhelming probability. If we set $\varepsilon = 0$, then DLOG finds the discrete logarithm $x$ only for a polynomial fraction of all $x$ that exactly match the expected Hamming weight on $v_1, v_2$.

One might be tempted to use cyclic rotations of the binary representation of $x$ to find a strip of $t$ bits that has the desired distribution. However, some problems arise here. If we fully rotate, we obtain a bit vector for which the Hamming weight of $v_1$ and $v_2$ matches its expectation. In this case, it might however happen that $w$ gets split into two parts by the cyclic rotations. In this case, we were not able to bound the number of $w$’s by a polynomial. We could also consider the case where we do not fully rotate $x$, but restrict to $n - t$ left rotations only such that the $w$-part does not split. We conjecture that the number of pathological instances where DLOG does not succeed for at least one of the $n - t$ rotations is exponentially small in this case, but are not able to prove that.

Our algorithm can be interpreted in terms of the representation technique introduced by Howgrave-Graham and Joux [HJ10] for solving the subset sum problem. Notice that we split $w = w_1 + w_2$ with $w_1, w_2 \in \mathbb{Z}_M$. Thus, we obtain exactly $M$ representations $(w_1, w_2)$ of $w$ as a sum. In our case, we use the fact that exactly one representation $(w_1, w_2)$ ensures that $x_1 = 0 \mod M$ and $x_2 = x \mod M$, simultaneously. We can directly compute this representation in polynomial time – once $x'$ is known – without any further assumption on the problem instance. This differs from [HJ10], where the lists have to be constructed in exponential time. Notice that like in [MMT11, BJMM12] it is possible to combine our technique with the classical technique of [HJ10]. We leave as an open problem whether this leads to even better results. Notice that in comparison to the small $x$ algorithm that runs in time $\sqrt{p} + \sqrt{|G|^7}/\sqrt{p}$, one could also expect the small weight $x$ algorithm to run in $\sqrt{p} + \sqrt{|G|^{H(\delta)}}/\sqrt{p}$ instead of $\sqrt{p} + \sqrt{q}^{H(\delta)}$. Exactly this gap could be closed by applying the representation technique.
7.3 Optimal Splitting

It remains to show how to optimally choose the subgroups $G_1$ and $G_2$ dependent on the factorization of $|G|$ and on the Hamming weight $\delta \cdot \log |G|$ of $x$. Since we apply Silver-Pohlig-Hellman on $G_1$, the group $G_1$ should contain all prime subgroups of $G$ that are smaller or as large as the largest prime subgroup of $G_1$. In other words, if $N = \prod_{i=1}^{k} p_i$ is the factorization of the order of $G$ and $p_1 \leq \ldots \leq p_k$, the only useful choices are $|G_1| = \prod_{i=1}^{\ell} p_i$ and $|G_2| = \prod_{i=\ell+1}^{k} p_i$ for $1 \leq \ell \leq k-1$. This is because we have to spend time $\sqrt{p}$ for the maximal prime divisor $p$ of $|G_1|$ anyway. Thus, our ordering of the $p_i$ minimizes $|G_2|$ and thus the overall running time.

Among the remaining $k-1$ choices, we have to find the best choice for $\ell$. Fix an $\ell$, $1 \leq \ell \leq k-1$, and define $\tau_i := \log_N p_i$ for each $1 \leq i \leq k$. From Theorem 1, the time complexity of DLOG is

$$\widetilde{O} \left( \sqrt{p_{\ell}} + \sqrt{p_{\ell+1}} \cdots p_k^{H(\delta+\varepsilon)} \right) = \widetilde{O} \left( (2^{n/2})^{\tau_{\ell}} + (2^{n/2})^{(\tau_{\ell+1} + \ldots + \tau_k) \cdot H(\delta+\varepsilon)} \right).$$

Let us define $p_0 := 1$, and thus $\tau_0 = 0$. In the case $\ell = 0$ we obtain the time complexity of the standard small weight Meet-in-the-Middle algorithm without using SPH. In the case $\ell = k$ we obtain the standard SPH without any Meet-in-the-Middle approach. Thus, our algorithm perfectly interpolates between both cases and there exist $\tau_i$ such that our algorithm improves upon SPH and standard Meet-in-the-Middle for any $0 < \ell < k$ with $\delta < \frac{1}{2}$ (cf. Fig. 7.5).

**Theorem 80.** Let $\tau_0 := 0$. Given $\delta, \varepsilon$ with $\delta + \varepsilon \in (0, \frac{1}{2}]$ and $\tau_1, \ldots, \tau_k$ as defined above, an optimal choice for DLOG is to pick $0 \leq \ell \leq k-1$ such that

$$\frac{\tau_\ell}{\tau_\ell + \ldots + \tau_k} < H(\delta + \varepsilon) \leq \frac{\tau_{\ell+1}}{\tau_{\ell+1} + \ldots + \tau_k}$$

and to choose $|G_1| = \prod_{i=1}^{\ell} p_i$ and $|G_2| = \prod_{i=\ell+1}^{k} p_i$.

**Proof.** First notice that

$$\bigcup_{\ell=0}^{k-1} \left( \frac{\tau_\ell}{\tau_\ell + \ldots + \tau_k}, \frac{\tau_{\ell+1}}{\tau_{\ell+1} + \ldots + \tau_k} \right)$$

defines a disjoint partition of $(0,1]$, due to the fact that $\tau_1 + \ldots + \tau_k = 1$. Thus each $\delta + \varepsilon$ with $0 < H(\delta + \varepsilon) \leq 1$ leads to a unique choice of $\ell$.

Fix $\delta, \varepsilon$ and choose $\ell$ as defined above. We want to show that for each choice of $\ell' \neq \ell$, DLOG’s complexity does not improve. Notice that there may be other choices for $\ell$ that achieve the same complexity.

If $\ell' < \ell$, then it is easy to see that $(\tau_{\ell+1} + \ldots + \tau_k) \cdot H(\delta + \varepsilon) \leq (\tau_{\ell+1} + \ldots + \tau_k) \cdot H(\delta + \varepsilon)$. Since $\frac{\tau_\ell}{\tau_\ell + \ldots + \tau_k} < H(\delta + \varepsilon)$, we also have $\tau_\ell < (\tau_\ell + \ldots + \tau_k) \cdot H(\delta + \varepsilon) \leq (\tau_{\ell+1} + \ldots + \tau_k) \cdot H(\delta + \varepsilon)$. Thus the complexity does not improve for $\ell' < \ell$.

If $\ell' > \ell$, obviously we have $\tau_\ell \leq \tau_{\ell'}$. Additionally, we have that $(\tau_{\ell+1} + \ldots + \tau_k) \cdot H(\delta + \varepsilon) \leq \tau_{\ell+1} \leq \tau_{\ell'}$, since $H(\delta + \varepsilon) \leq \frac{\tau_{\ell+1}}{\tau_{\ell+1} + \ldots + \tau_k}$. Therefore, the complexity also does not improve for any $\ell' > \ell$.

\[\square\]
Fig. 7.5 shows the time complexity for a fixed group of size $N$ that is a product of five primes with sizes $N^{0.1}$, $N^{0.15}$, $N^{0.2}$, $N^{0.25}$, and $N^{0.3}$. In this case, Silver-Pohlig-Hellman (dashed line in Fig. 7.5, corresponding to $\ell = k$) has a time complexity of $N^{0.15}$. As we can see, our algorithm’s improvement (solid line) is defined piecewise for $1 \leq \ell \leq k - 1$. For small values of $\delta$, our algorithm uses a standard meet-in-the-middle approach (dotted line, corresponding to $\ell = 0$), which yields the optimal complexity.
Bibliography


[MO] Alexander May and Ilya Ozerov. Solving knapsacks in $2^{0.287n}$.


[Sud] Madhu Sudan. Algorithmic Introduction to Coding Theory. Lecture notes, MIT.

