UFR des Sciences
École Doctorale STV

Laboratoire Parallélisme, Réseaux,
Systèmes, Modélisation (PRISM)

The representation technique
Applications to hard problems in cryptography

La technique de représentation
Application à des problèmes difficiles en cryptographie

Thèse présentée en vue de l'obtention du grade de
Docteur de l'Université de Versailles Saint-Quentin-en-Yvelines

Spécialité: Informatique

par

Anja Becker

Soutenue le 26 octobre 2012 devant le jury composé de:

Directeur de thèse:	Antoine Joux	Université de Versailles
Rapporteur:	Alexander May	Ruhr-Universität Bochum
	Nicolas Sendrier	INRIA
	Igor Shparlinski	Macquarie University
Examinateur:	Daniel Augot	INRIA
	Jean-Sébastien Coron	Université du Luxembourg
	David Naccache	Université Paris II
	Monica Nevins	University of Ottawa

The representation technique
Applications to hard problems in cryptography

Thanks

I am most grateful for the warm welcome that I received when I joined the crypto group and the great ambiance between the people I met in the last three years at UVSQ. My thanks goes to: Antoine Joux, Antoine Rojat, Bastien Vayssière, Bettina Kramer, Catherine Le-Quéré, Christophe Castro, Jaques Patarin, Joana Treger, Léonard Dallot, Luca de Feo, Louis Goubin, Malika Izabachène, María Naya, Mickael Quisquater, Naomi Benger, Peter Birkner, Sorina Ionica, Vanessa Vitse and William Jalby. My arrival and stay in France was as smooth as it can be (when settling down in Paris) due to their help.

I would like to thank Antoine for sharing his passion for research and for the continous exchange of ideas towards new directions in research. It is a great chance and pleasure to learn from Antoine and to look over his shoulder while he discovers a new trick. Our discussions about cryptography and German-French differences are many fond memories that I will take with me when I leave Versailles.

Special thanks also to Sorina who supported my application passionately and to Louis, William and Antoine who solved all financial and bureaucratic obstacles. I am also very grateful for Bettina's help who was a friend from the first moment on. She picked me up at the airport and offered me to stay at her place for the first weeks. I remember many amusing lunch breaks with the crypto group at Picardie, at a very nice restaurant at the market place or in the garden of the château de Versailles. The French simply know how to savour the lunch break. Bastien and me had many conversations about french everyday life, movies, books and very engaged political discussions. It was a pleasure. I would like to thank Bastien, Jérôme, María and Nicolas for many helpful discussions about crypto questions. Many thanks to Antoine, Bastien, Christian Rechberger, Christiane Peters, Léonard, María, Nicolas and Peter Schwabe who took the time to discuss and proof read my thesis although their schedules were quit full as well.

I thank my co-authors Alexander May, Alexander Meurer, Antoine and Jean-Sébastien Coron for the great teamwork. My special thanks to Alexander May who invited me to his group in Bochum and to the Dagstuhl seminar. To this respect, I would also like to thank the ambitious reviewer(s) of the papers whose comments and support lead to improved publications. I thank Nicolas Sendrier for inviting me several times to INRIA for discussing code based crypto with team members.

I would like to thank Alexander May, Igor Spharlinksi and Nicolas Sendrier for taking interest in my research and willing to accept being a 'rapporteur' of my thesis. I would also

like to express my gratitude to the other committee members of my PhD jury: Antoine, Daniel Augot, Jean-Sébastien, David Naccache and Monica Nevins.

I was very lucky to have Monica as a supervisor at the University of Ottawa and I am happy that we stayed in touch over the years. She helped me succeed the first year as a graduate student and was very understanding and supportive when I decided to change the program after Isabèlle Déchène left us. Isabèlle was such an encouraging and enthusiastic young woman. I am glad to have met her.

The 'Gottlieb Daimler- und Karl Benz-Stiftung' enabled me to fulfill my dream and study abroad by granting me a two-year PhD scholarship. I would like to thank all members of the selection committee for the trust they set into me. I also like to compliment the organizers of the conferences, summer schools and workshops that I joined for their accomplishment. I am grateful that I could participate in many events and would like to thank Louis for supporting my interest and for ensuring the financial basis.

My closest friends, Annika Gallistl, Jean-Christophe Fleury, Maike Bußmann, Maike Klingemann and Susanne Jasser, have always been a rich source of joy, guidance, calm and help. I am happy to be part of their lives and they play a very important part in mine. I am grateful to see that our friendship has lasted these many years and over long distances. I thank my parents and my family whose love and unconditional support has made it possible for me to be where I am today.

Merci,
Anja Becker

September 21, 2012

Contents

Introduction et motivation

Cette thèse porte sur les techniques algorithmiques pour résoudre des instances uniformes du problème du sac à dos exact (subset sum) et du décodage à distance d'un code linéaire aléatoire.

Étant donné un ensemble de nombres entiers a_i de taille maximale $M = \log_2(\max_i a_i)$, le subset sum pose la question de savoir s'il existe un sous-ensemble de nombres entiers dont la somme est égale à une valeur fixée. Il est prouvé NP-complet. Le ratio entre le nombre d'entiers donné et la taille maximale des éléments, M, est dénommé densité. Ce problème offre une alternative aux problèmes utilisés classiquement en cryptographie comme par exemple les problèmes basés sur la factorisation et le logarithme discret. Le problème admet une description simple et le calcul d'une somme de nombres entiers est facile à effectuer. De plus, contrairement à d'autres problèmes en théorie des nombres, aucun algorithme quantique n'est connu pour résoudre ce problème en temps polynomial.

À l'aide du subset sum, il est possible de construire des fonctions à sens unique, des générateurs de nombres pseudo aléatoires et des schémas de chiffrement à clé publique dont la sécurité est basé sur la difficulté du problème dans le cas moyen.

Dans les années 70 et 80, des chercheurs en cryptographie ont proposé des schémas de chiffrement à clé publiques dont la sécurité est basée directement sur le subset sum. Rapidement après la publication de ces résultats, des attaques ont été présentées. La principale faiblesse provient de la structure nécessaire à un déchiffrement efficace ou d'une faible densité, inférieure à un. Le cas le plus difficile apparaît pour un sac à dos uniforme de densité proche de un, ce qui est le cas pour nos études. Nous présentons une nouvelle technique algorithmique générique, supposant que la structure du problème est correctement masquée.

Outre les schémas de chiffrement, le subset sum permet des constructions efficaces de fonctions de hachage et de générateurs de nombres pseudo aléatoires. Ces derniers peuvent être utilisés pour implémenter des schémas de chiffrement à clé publique, des protocoles à divulgation nulle de connaissance, des schémas d'identification et des schémas de signature numérique. Leur sécurité repose sur la difficulté du subset sum dans le cas moyen. Il existe également des cryptosystèmes basés sur les réseaux euclidiens qui sont prouvés sûrs tant que le subset sum dans le cas moyen est difficile.

Le subset sum apparait également dans autres contextes en cryptographie. En effet, les problèmes de décodage peuvent être vus comme une version vectorielle du subset sum; les techniques utilisées pour la résolution du second peuvent donc s'appliquer aux premiers. Ces problèmes, et plus particulièrement le problème du décodage borné dans un code aléatoire, sont à la base de plusieurs schémas cryptographiques. Ce problème est prouvé NP-complet dans le cas d'un code binaire linéaire et aucun algorithme quantique n'est connu pour le résoudre en temps polynomial. Ce problème est l'un des candidats les plus prometteurs pour construire des systèmes cryptographiques sûr en présence d'un ordinateur quantique. Il admet des schémas de chiffrement à clé publique et de signature numérique et des fonctions de hachage.

La sécurité de presque tous les cryptosystèmes basés sur les codes correcteurs s'appuie sur la difficulté du décodage. La clé publique appartient à un code qui est indistinguable d'un code aléatoire. Pour permettre un décodage efficace pour le destinataire, la clé secrète est bien structurée. Par une transformation linéaire on obtient la clé publique qui déguise cette structure et rend le code d'apparence aléatoire. Un attaquant a deux options. Premièrement, il peut essayer de distinguer le code perturbé d'un code aléatoire et révéler la structure. Deuxièmement, il peut appliquer un algorithme générique qui résout le problème de décodage d'un code aléatoire.

Il existe des familles de codes dont la structure peut être masquée efficacement face aux attaques connues. Nous allons alors viser à améliorer les algorithmes génériques qui résolvent le problème de décodage d'un code aléatoire. La méthode la plus efficace est le décodage par ensemble d'information.

Un problème proche du décodage dans les codes linéaires aléatoires est le problème 'learning-parity-with-noise' (LPN) qui est fréquemment utilisé en cryptographie [Ale03, HB01, KKC$^+$01]. Le problème de recherche LPN est un problème de décodage dans un code donné. Regev a montré [Reg05] que la version décisionnelle, un outil utile pour des constructions cryptographiques, est équivalent au problème de recherche LPN. Il est alors équivalent au problème de décodage d'un code linéaire aléatoire.

Le problème 'learning-with-error' (LWE) introduit par Regev [Reg05] est une généralisation du problème LPN au cas de codes définis sur un plus grand corps. Les algorithmes présentés pourraient être également adaptés à ce problème (de façon similaire à [CG90, Pet10]).

Le problème du décodage borné dans un code linéaire aléatoire est donc à la base de l'ensemble de la cryptographie fondée sur les codes correcteurs d'erreurs et sur les problèmes LPN et LWE. Étudier la complexité de ce problème permet donc de déterminer les paramètres qui assurent la sécurité de ces constructions cryptographiques.

Résultats et organisation de la thèse

Ce mémoire se divise en trois parties. La première partie est un préliminaire algorithmique qui présente les techniques de recherche de collisions dont nous avons besoin dans la suite. Elle introduit le problème des k-sommes qui est lié au problème de la somme de sous-ensembles. Les algorithmes bien connus pour sa résolution constituent des exemples d'application des techniques de recherche de collisions.

La deuxième partie est dédiée au subset sum. La technique générique la plus efficace pour résoudre ce problème pour un ensemble de n éléments était un algorithme par Shamir and Schroeppel qui nécessite un temps $2^{\frac{n}{2}}$ et de la mémoire $2^{\frac{n}{4}}$. En 2010, Howgrave-Graham et Joux ont présenté une nouvelle technique, nommé la technique de représentation. Le nouvel algorithme probabiliste tourne on temps $2^{0.337\,n}$ en utilisant $2^{0.311\,n}$ de mémoire en moyen pour les instances aléatoires de densité un. Nous présentons une analyse plus détaillée de l'algorithme et proposons un nombre variable de niveaux dans le cas du sac à dos déséquilibré ou de passer au sac à dos complémentaire. Des améliorations par rapport à la mémoire nous permettons de proposer un algorithme qui nécessite une mémoire heuristique de $2^{0.272\,n}$ et garde le même temps asymptotique.

La technique de représentation peut être généralisée en permettant une petite fraction de coefficients négatifs dans les éléments intermédiaires de l'algorithme. Suite à cette généralisation nous proposons un algorithme probabiliste avec un temps de calcul réduit. L'algorithme tourne en temps asymptotique $2^{0.291\,n}$ et utilise $2^{0.291\,n}$ de mémoire. Le résultat a mené à la publication [BCJ11]. La mémoire peut être réduite par la même observation que précédemment et descend à $2^{0.279\,n}$ sous hypothèses heuristiques.

La partie II est organisée comme suit. Le chapitre 2 présente le subset sum et définit le cas intéressant pour nos études. Nous offrons un tour d'horizon des primitives cryptographiques basées sur le subset sum et des schémas dont la sécurité dépend de ce problème. Les chapitres suivants présentent les techniques algorithmiques génériques pour résoudre le subset sum pour les instances aléatoire dans le pire des cas, de densité un. Nous distinguons entre les attaques classiques par paradoxe d'anniversaire comme l'algorithme par Shamir et Schroeppel (chapitre 3) et les algorithmes qui utilisent la méthode récente des représentations dans sa forme basique (chapter 4) et généralisée (chapter 5). Le chapitre 6 présente un algorithme qui utilise une mémoire constante qui fait partie de la publication [BCJ11]. Les chapitres 2 et 3 sont inspirés en partie du chapitre 8 dans [Jou09]. Les chapitres 4 et 5 sont basés sur les publications [HGJ10] et [BCJ11], respectivement.

La troisième partie présente une amélioration du décodage par ensemble d'information qui est la méthode la plus efficace pour résoudre le problème de décodage par syndrome d'un code aléatoire. Une méthode classique dans le pire cas a un temps asymptotique de $2^{0.05564\,n}$ pour le décodage à distance moitié et un code a longueur n. Le pire cas est un code d'un ratio d'information qui maximise le temps de calcul pour résoudre le problème de décodage. L'algorithme a été amélioré récemment par 'ball-collision decoding'. L'algorithme tourne en temps $2^{0.05559\,n}$ et utilise $2^{0.0148\,n}$ de mémoire.

L'application de la technique de représentation permet de trouver la solution en temps $2^{0.05363\,n}$ avec une petite augmentation en mémoire qui est de l'ordre de $2^{0.0215\,n}$. Nous proposons une méthode qui permet de diminuer la mémoire heuristiquement à $2^{0.0139\,n}$.

L'algorithme n'exploite pas au total la puissance de la technique basée sur les représentations. En utilisant notre technique de représentation généralisée nous pouvons nettement baisser le temps asymptotique. Nous permettons des intersections de positions de bits analogue au cas du subset sum sur les nombres entiers. Le nouvel algorithme permet de résoudre le problème de décodage par syndrome d'un code aléatoire linéaire en temps $2^{0.04933\,n}$ en utilisant $2^{0.0307\,n}$ de mémoire. Le résultat est publié dans [BJMM12]. La mémoire peut être réduite à $2^{0.0306\,n}$ heuristiquement.

La partie III est organisée comme suit. Le chapitre 8 fait une introduction au codes correcteurs d'erreurs linéaires et présente des schémas cryptographiques populaires basés sur les codes. Le chapitre 9 introduit la méthode du décodage par ensemble d'information générale et explique les techniques précédentes. Nous présentons ensuite (chapitre 10) comment on peut améliorer le décodage par ensemble d'information en utilisant le technique de représentations généralisée.

Le premier chapitre est inspiré par [Bar98, Pet11] et les chapitres 9 et 10 sont basés sur la publication [BJMM12].

Nous présentons des approximations asymptotiques de termes binomiaux qui nous permettent d'évaluer la complexité asymptotiques des algorithmes dans l'appendice A. Le code octave qui nous permet d'évaluer la complexité se trouve dans l'appendice B. Les deux sections suivantes résument les définitions, algorithmes et résultats importants du mémoire.

Comment résoudre le subset sum uniforme de densité un

Nous définissons le problème subset-sum qui est aussi appelé le problème de la somme des sous-ensembles comme suit.

Definition 0.1 (Problème subset-sum)
Étant donnés n nombres entiers a_i et un entier S, trouver une solution $\mathbf{x} = (x_1, .., x_n) \in \{0,1\}^n$ tel que

$$\mathbf{a} \cdot \mathbf{x} := \sum_{i=1}^{n} a_i\, x_i = S \tag{0.1}$$

ou montre qu'aucune solution n'existe.

Le problème est NP-complet [Kar72] dans sa forme décisionnelle et NP-difficile dans sa forme calculatoire. Nous dénotons par M la taille maximale des nombres entiers a_i, $M = \log_2(\max_i a_i)$. Le problème est aussi nommé problème du sac à dos (exact) en cryptographie. Les a_i sont les poids ou les éléments du sac à dos.
La difficulté pour résoudre un subset sum dépend d'une propriété du sac à dos qui est la densité.

Definition 0.2 (Densité)
La proportion entre les éléments du sac à dos $\mathbf{a} = (a_1, .., a_n)$ et leur taille maximale M est la densité:

$$d := \frac{n}{\log_2(\max_i a_i)} \ . \tag{0.2}$$

Nous nous intéressons spécialement aux sacs à dos sans structure. Ces sacs à dos sont nommés des sacs à dos uniformes.

Definition 0.3 (Sac à dos uniforme)
Nous choisissons les poids a_i uniformément et aléatoirement dans l'intervalle $[1, \lfloor 2^{\frac{n}{d}} \rfloor]$. La solution $\mathbf{x}$ est choisie uniformément et aléatoirement dans $\{0,1\}^n$. La sous-somme $S = \sum_{i=1}^{n} a_i x_i$ est un sac à dos uniforme avec solution.

Impagliazzo et Naor ont montré [IN96, IN89] que le problème subset sum pour une instance uniforme est le plus difficile si la densité est un. Pour un sac à dos uniforme nous pouvons compter les nombres de sous-sommes possibles de taille ℓ. Il y en a $\binom{n}{\ell}$ ce qui est maximal pour $\ell = \lceil \frac{n}{2} \rceil$. Nous supposons alors que la densité est un et que la solution a un poids $\ell \approx \lceil \frac{n}{2} \rceil$. Pour résoudre le subset sum nous allons passer au sac à dos modulaire en calculant

$$\mathbf{a} \cdot \mathbf{x} := \sum_{i=1}^{n} a_i \, x_i \equiv S \mod M$$

pour un grand nombre entier M où les a_i et S sont dans $\mathbb{Z}_M$.

Algorithmes simples. Pour trouver la solution du subset sum, nous pouvons énumérer tous vecteurs $x \in \{0,1\}^n$ et calculer la sous-somme correspondant à chaque vecteur. L'algorithme nécessite 2^n calculs de sous-sommes dans le pire cas et garde qu'un élément en mémoire.

On peut réduire le temps de calcul si on partage les éléments en deux ensembles de même taille [HS74]. Nous calculons toutes les $2^{\lfloor \frac{n}{2} \rfloor}$ sommes $\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} a_i \, x_i$ et les mettons dans une liste $\mathcal{L}_1$. Une recherche de collision avec les sommes $S - \sum_{\lfloor \frac{n}{2} \rfloor + 1}^{n} a_i \, x_i$ dans une deuxième liste $\mathcal{L}_2$ permet de trouver la solution. Car une collision correspond à l'équation

$$\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} a_i \, x_i = S - \sum_{\lfloor \frac{n}{2} \rfloor + 1}^{n} a_i \, x_i$$

chaque collision est une solution du subset sum. Les listes contiennent à peu près $2^{\frac{n}{2}}$ éléments. Le coût pour la création des listes et la recherche de collisions entre deux listes triées se fait en temps $\mathcal{O}\left(n \, 2^{\frac{n}{2}}\right)$ (détails dans la section 1.1).

Algorithme par Schroeppel et Shamir sous forme heuristique. On peut réduire la mémoire comme observé par Shamir et Schroeppel [SS81]. Leur algorithme déterministe résout toutes instances en temps $\mathcal{O}\left(n \, 2^{\frac{n}{2}}\right)$en utilisant $\mathcal{O}\left(n \, 2^{\frac{n}{4}}\right)$ de mémoire (section 3.1). Nous présentons une version heuristique qui a la même complexité. Pour des rares cas, comme un sac à dos dont tous éléments sont zero, la complexité peut être supérieure. L'algorithme donne une bonne introduction à la technique que nous allons utiliser pour améliorer le temps.

Nous supposons que la solution a un poids $\frac{n}{2}$ et que $4 \mid n$. La solution se découpe en quatre morceaux tel que

$$\sum_{i=0}^{\lfloor \frac{n}{4} \rfloor} a_i x_i + \sum_{i=\lfloor \frac{n}{4} \rfloor + 1}^{\lfloor \frac{n}{2} \rfloor} a_i x_i = S - \Big(\sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{\lfloor \frac{3n}{4} \rfloor} a_i x_i + \sum_{i=\lfloor \frac{3n}{4} \rfloor \rfloor + 1}^{n} a_i x_i \Big) \; .$$

Si nous énumérons toutes les sous-sommes possibles d'un quart des éléments, nous allons trouver la solution. Cette idée est par contre trop couteuse. Pour un nombre entier M, nous observons que

$$\sum_{i=0}^{\lfloor \frac{n}{4} \rfloor} a_i x_i + \sum_{i=\lfloor \frac{n}{4} \rfloor+1}^{\lfloor \frac{n}{2} \rfloor} a_i x_i \equiv S - (\sum_{i=\lfloor \frac{n}{2} \rfloor+1}^{\lfloor \frac{3n}{4} \rfloor} a_i x_i + \sum_{i=\lfloor \frac{3n}{4} \rfloor\rfloor+1}^{n} a_i x_i) \equiv R \mod M$$

pour une solution et un entier $R \in \mathbb{Z}_M$. Car nous ne connaissons pas R, il nous faut les essayer tous.

L'algorithme consiste en les étapes suivantes. Nous choisissons quatre sous-ensembles disjoints d'éléments du sac à dos chaque contenant un quart d'éléments. Pour chaque ensemble nous calculons toutes sous-sommes σ_i possibles et les mettons dans des listes $\mathcal{Y}_1, .., \mathcal{Y}_4$. Les listes ont une longueur $2^{\frac{n}{4}}$. Nous choisissons un nombre R aléatoire. La prochaine étape unit les listes $\mathcal{Y}_1$ et $\mathcal{Y}_2$ par recherche de collision. Elle crée une liste $\mathcal{L}_1$ d'éléments $\sigma_1 + \sigma_2$ où $\sigma_1 + \sigma_2 \equiv R \mod M$: Nous trions $\mathcal{Y}_1$ par rapport à ses valeurs modulo M et cherchons pour chaque élément $\sigma_2 \in \mathcal{Y}_2$ un élément de valeur $R - \sigma_2$ dans la liste triée $\mathcal{Y}_1$.

Nous faisons de même pour les listes $\mathcal{Y}_3$ et $\mathcal{Y}_4$ et créons des éléments $\sigma_3 + \sigma_4$ dans une liste $\mathcal{L}_2$. Étant donné un sac à dos uniforme et un M qui est inférieur ou égal à la taille des éléments nous pouvons supposer que les sous-sommes sont uniformément distribuées dans $\mathbb{Z}_M$. Les listes $\mathcal{L}_1$ et $\mathcal{L}_2$ ont une longueur $2^{\frac{n}{2}}/M$. Pour minimiser la mémoire et le temps de calcul, nous choisissons M de tailles $2^{\frac{n}{4}}$ à peu près. La dernière étape cherche des collisions entre $\mathcal{L}_1$ et $\mathcal{L}_2$ sur les entiers. Pour cela nous trions les deux listes en ordre croissante. A chaque tour nous comparons deux éléments commençant avec le premier de la liste $\mathcal{L}_1$ et le dernier le la liste $\mathcal{L}_2$. Il y a trois possibilités: Si $\sigma_1 + \sigma_2 \in \mathcal{L}_1$ est inférieur à $S - (\sigma_3 + \sigma_4) \in \mathcal{L}_2$, nous choisissons le prochain élément de la liste $\mathcal{L}_1$. Si $\sigma_1 + \sigma_2 \in \mathcal{L}_1 > S - (\sigma_3 + \sigma_4)$, nous changeons pour le prochain élément dans la liste $\mathcal{L}_2$. Dans le cas d'égalité nous avons trouvé une collision qui est une solution du subset sum.

L'algorithme trie et mémorise des listes de taille $2^{\frac{n}{4}}$. La première étape trouve un nombre attendue de $2^{\frac{n}{4}}$ éléments qui sont mis dans deux listes. L'algorithme nécessite alors $\mathcal{O}\left(\frac{n}{4}\, 2^{\frac{n}{4}}\right)$ de mémoire. La dernière étape trie deux listes à $2^{\frac{n}{4}}$ éléments et exécute $2 \cdot 2^{\frac{n}{4}}$ calculs au maximum pour trouver une solution. Il nous faut répéter avec un R différent si la solution n'est pas trouvée. Le nombre des répétitions dans le pire cas est $2^{\frac{n}{4}}$. Le temps total de l'algorithme devient $\mathcal{O}\left(n\, 2^{\frac{n}{2}}\right)$.

La technique simple des représentations. En 2010, Howgrave-Graham et Joux ont proposé un nouvel algorithme [HGJ10] pour résoudre des instances uniformes du problème subset sum de densité un. La densité un implique qu'il y a une seule (ou très peu de) solutions. La nouvelle idée est de choisir des sous-ensembles des éléments du sac à dos non disjoints. Nous définissons une représentation d'une solution de poids ℓ comme un couple de vecteurs $(\mathbf{y}, \mathbf{z}) \in \{0,1\}^n \times \{0,1\}^n$ de poids $\ell/2$.

La figure 0.1 montre une telle représentation.

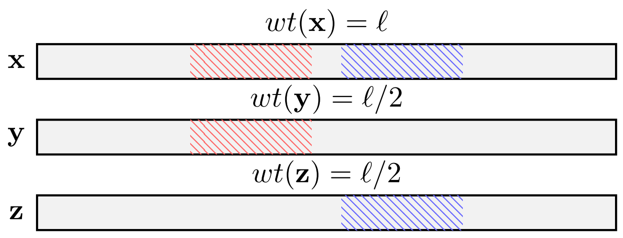


Figure 0.1.: *Une représentation* $(\mathbf{y}, \mathbf{z})$ *d'une solution* $\mathbf{x}$*. Les zones rayées montrent les positions non-nulles.*

Les vecteurs $\mathbf{y}, \mathbf{z}$ sont une solution partielle et le nombre des représentations dépend du nombre de façon de choisir $\lfloor \ell/2 \rfloor$ positions de valeur un dans ℓ:

$$N_{HGJ} = \begin{cases} 2 \cdot \binom{\ell}{(\ell-1)/2} & \text{pour } \ell \text{ impair} \\ \binom{\ell}{\frac{\ell}{2}} & \text{pour } \ell \text{ pair} \end{cases}$$

qui est de l'ordre $\mathcal{O}\left(2^{\ell}\right)$ pour grand n et ℓ.

Nous simplifions la présentation en supposant que $2|\ell$. L'algorithme cherche une représentation qui satisfait la contrainte suivante:

$$\begin{aligned} \mathbf{a} \cdot \mathbf{y} &\equiv R &\mod M \quad \text{et} \\ \mathbf{a} \cdot \mathbf{z} &\equiv S - R &\mod M \end{aligned} \tag{0.3}$$

pour un nombre entier M et un élément R aléatoirement choisi dans $\mathbb{Z}_M$.

Supposons que nous avons deux listes $\mathcal{L}_1, \mathcal{L}_2$ de sommes $\mathbf{a} \cdot \mathbf{y} = \sum_{i=1}^{n} a_i y_i$ et $\mathbf{a} \cdot \mathbf{z} = \sum_{i=1}^{n} a_i z_i$ satisfaisant (0.3) tel que $\mathbf{y}, \mathbf{z}$ ont un poids $\ell/2$ et sont choisis indépendamment et aléatoirement dans $\{0,1\}^n$. Chaque couple d'éléments des listes $\mathcal{L}_1$ et $\mathcal{L}_2$ satisfait

$$\mathbf{a} \cdot \mathbf{y} + \mathbf{a} \cdot \mathbf{z} \equiv S \mod M \ .$$

Si nous en trouvons un pour lequel l'égalité est vraie sur les entiers et pour lequel le poids de $\mathbf{y} + \mathbf{z}$ est ℓ, nous avons trouvé une représentation de la solution. La solution est donnée par $\mathbf{y} + \mathbf{z}$.

Nous avons transformé le problème d'origine de deux manières. Premièrement, nous ne cherchons plus directement la solution unique mais nous cherchons une représentation parmi un nombre exponentiel. Nous ajoutons des degrés de liberté et augmentons l'espace de recherche. Pour réduire le coût, nous ajoutons les contraintes modulaires. Deuxièmement, les éléments dans $\mathcal{L}_1, \mathcal{L}_2$ sont eux-mêmes des solutions de poids $\ell/2$ de deux problèmes du sac à dos uniforme donnés dans (0.3). Ces solutions peuvent être trouvées par une technique classique comme l'algorithme par Shamir et Schroeppel ou en appliquant la technique des représentations encore une fois.

L'algorithme par Howgrave-Graham et Joux a deux niveaux de représentation et tourne en temps $\tilde{\mathcal{O}}\left(2^{0.337\,n}\right)$ en utilisant $\tilde{\mathcal{O}}\left(2^{0.311\,n}\right)$ de mémoire. Le chapitre 4 explique l'idée et l'algorithme en détail.

La technique des représentations généralisée. L'idée de représenter la solution d'un sac à dos par deux vecteurs binaires de même taille comme la solution peut être généralisée. Nous proposons de décomposer la solution en deux vecteurs en coefficients dans $\{-1, 0, 1\}$. En ajoutant quelques coefficients -1, nous cherchons des solutions partielles de poids légèrement plus élevées. Nous gagnons plus de degrés de liberté car le nombre des représentations augmente. Nous pouvons alors imposer des contraintes modulaires plus fortes et diminuer la taille des listes.

Nous choisissons un paramètre α tel que αn est le nombre des positions à valeur -1 par solution partielle. Une représentation d'une solution $\mathbf{x}$ est un couple $(\mathbf{y}, \mathbf{z}) \in \{-1, 0, 1\}^n \times \{-1, 0, 1\}^n$ tel que $\mathbf{x} = \mathbf{y} + \mathbf{z}$. Les vecteurs $\mathbf{y}, \mathbf{z}$ ont le même poids. Ils contiennent $\ell/2 + \alpha n$ 1 et αn -1 chacun.

La figure 0.2 montre une représentation possible. Le nombre des représentations se calcule par le nombre de façon de décomposer chaque $x_i \in \{1, 0\}$ de la solution en un couple (y_i, z_i) tel que $y_i + z_i = x_i$. Nous pouvons décomposer les ℓ 1 en couples $(0, 1)$ ou $(1, 0)$ se qui peut être fait en $\binom{\ell}{\ell/2}$ manières différentes. Les $n - \ell$ 0 sont représentés par $(0, 0)$, $(1, -1)$ ou $(-1, 1)$. Nous choisissons αn couples $(1, -1)$ et $(-1, 1)$. Le coefficient multinomial $\binom{n-\ell}{\alpha n, \alpha n, n-\ell-2\alpha n}$ compte le nombre des décompositions possibles des zéros. Le nombre total des représentation est:

$$N_{BCJ} = \binom{\ell}{\ell/2}\binom{n-\ell}{\alpha n, \alpha n, n-\ell-2\alpha n} . \tag{0.4}$$

Par rapport à la technique des représentations basique, le nombre augmente pour $\alpha > 0$ du deuxième facteur dans (0.4). Les contraintes modulaires pour les solutions partielles sont choisies de l'ordre du nombre des représentations. Cela nous permet de réduire les listes intermédiaires en garantissant une représentation en moyenne. Comme le nombre des représentations augmente, nous pouvons alors augmenter les contraintes et réduire encore plus les listes.

Notre algorithme a un temps de calcul asymptotique inférieur aux autres algorithmes. Il a une complexité $\tilde{\mathcal{O}}\left(2^{0.291\,n}\right)$ en temps et mémoire comme détaillé dans le chapitre 5.

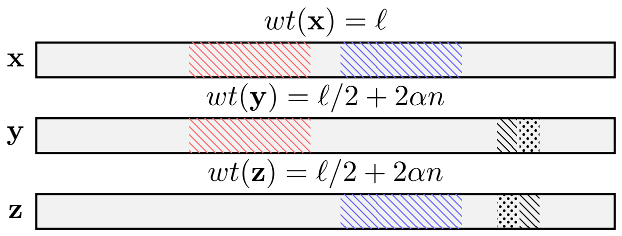

$wt(\mathbf{x}) = \ell$
$\mathbf{x}$
$wt(\mathbf{y}) = \ell/2 + 2\alpha n$
$\mathbf{y}$
$wt(\mathbf{z}) = \ell/2 + 2\alpha n$
$\mathbf{z}$

Figure 0.2.: *Une représentation de la solution $\mathbf{x} = \mathbf{y} + \mathbf{z}$. Les zones rayées montrent les positions des 1 et les zones pointées représentent les positions des -1.*

Application de la technique des représentations généralisée

Les technique utilisées pour le problème de subset sum sur les nombres entiers peuvent être appliquées dans le domaine des codes linéaires. La sécurité des schémas cryptographiques basés sur les codes linéaires dépend de la difficulté du problème suivant.

Definition 0.4 (Problème de décodage par syndrome)
Étant donné une matrice de parité $H \in \mathbb{F}_2^{(n-k)\times n}$ d'un code linéaire, un syndrome $s \in \mathbb{F}_2^{n-k}$ et un entier positive ω, le problème consiste en trouvant un vecteur $\mathbf{e} \in \mathbb{F}_2^n$ de poids ω tel que $H\mathbf{e}^t = s$.

La version décisionnelle est NP-complète [BMvT78] pour des codes binaires linéaires. Il est équivalent [LDW94] au problème de décodage borné.

Definition 0.5 (Problème de décodage borné)
Étant donné une matrice génératrice $G \in \mathbb{F}_2^{k\times n}$ d'un code binaire linéaire C, un vecteur aléatoire $y \in \mathbb{F}_2^n$ et un entier positive ω, le problème consiste à trouver un vecteur $\mathbf{e} \in \mathbb{F}_2^n$ de poids au plus ω tel que $y + e \in C$.

La méthode la plus efficace pour résoudre ces problèmes pour un code aléatoire est le décodage par ensemble d'information (information-set decoding, ISD). Nous utilisons le cadre du ISD généralisé comme introduit par Finiasz et Sendrier [FS09] en 2009. Les données en entrée d'un algorithme ISD sont une matrice de parité $H \in \mathbb{F}_2^{(n-k)\times n}$ d'un code binaire linéaire en longueur n, dimension k et distance minimale d et un syndrome $\mathbf{s} = H\mathbf{e}$. Le vecteur $\mathbf{e}$ est inconnu et a un poids $\omega := \mathrm{wt}(\mathbf{e}) = \lfloor \frac{d-1}{2} \rfloor$.

ISD est un algorithme randomisé avec deux étapes principales qui sont itérées jusqu'à ce que le vecteur $\mathbf{e}$ soit trouvé. La première partie consiste en une transformation linéaire de la matrice de parité qui dépend d'une permutation aléatoire des colonnes de la matrice. La deuxième partie de l'algorithme exécute une recherche.

La matrice de sortie de la première partie est en forme systématique, comme montré dans la figure 0.3. Elle est obtenue par multiplication avec une matrice de permutation $P \in \mathbb{F}_2^{n\times n}$ et une matrice de transformation inversible $T \in \mathbb{F}_2^{(n-k)\times(n-k)}$: $\tilde{H} = THP$.

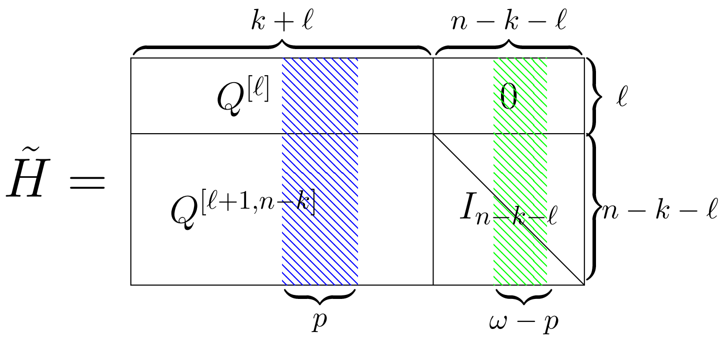


Figure 0.3.: *Matrice de parité $\tilde{H}$ sous forme systématique.*

Nous utilisons les notations suivantes: Q^I dénote la projection de Q aux lignes définie par l'ensemble des indices $I \subset \{1, \ldots, n-k\}$. Analoguement, Q_I dénote la projection de Q sur les colonnes. Nous définissons $[\ell] := \{1, \ldots, \ell\}$ et $[\ell, n-k] := \{\ell, \ldots, n-k\}$.

Avec $\tilde{\mathbf{s}} := T\mathbf{s}$ le nouveau problème est de trouver un vecteur $\tilde{\mathbf{e}}$ de poids ω qui satisfait $\tilde{H}\tilde{\mathbf{e}} = \tilde{\mathbf{s}}$. En appliquant l'inverse des transformations nous retrouvons la solution $\mathbf{e} = P\tilde{\mathbf{e}}$ au problème de départ.

Pendant la phase de recherche, nous énumérons tous les vecteurs avec une distribution de poids spécifique. Les vecteurs peuvent être décomposés: $\tilde{\mathbf{e}} = (\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2) \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{n-k-\ell}$ ou $\mathrm{wt}(\tilde{\mathbf{e}}_1) = p$ et $\mathrm{wt}(\tilde{\mathbf{e}}_2) = \omega - p$. La figure 0.4 montre la distribution du poids. La permutation

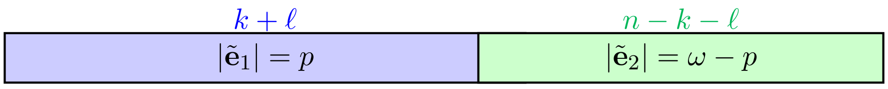

$k+\ell$
$n-k-\ell$
$|\tilde{\mathbf{e}}_1| = p$
$|\tilde{\mathbf{e}}_2| = \omega - p$

Figure 0.4.: *Distribution du poids de* $\tilde{\mathbf{e}}$.

disperse les coordonnées non-nulles aléatoirement. La probabilité qu'un vecteur $\tilde{\mathbf{e}}$ possède la structure recherchée est

$$\mathcal{P} = \frac{\binom{k+l}{p}\binom{n-k-l}{\omega-p}}{\binom{n}{\omega}} \quad . \tag{0.5}$$

L'inverse de la probabilité $\mathcal{P}^{-1}$ correspond au nombre d'itérations attendu pour trouver le bon vecteur $\tilde{\mathbf{e}}$.

Grâce à la forme systématique de $\tilde{H}$, nous voyons que

$$\tilde{H}\tilde{\mathbf{e}} = \begin{bmatrix} Q^{[\ell]}\tilde{\mathbf{e}}_1 \\ Q^{[\ell+1,n-k]}\tilde{\mathbf{e}}_1 + \tilde{\mathbf{e}}_2 \end{bmatrix} = \tilde{\mathbf{s}} \ .$$

Cela nous permet de chercher des candidats $\tilde{\mathbf{e}}_1$ et de choisir $\tilde{\mathbf{e}}_2$ en cohérence avec le syndrome.

Premièrement, nous recherchons le vecteur tronqué $\tilde{\mathbf{e}}_1 \in \mathbb{F}_2^{k+\ell}$ de poids p. Nous énumérons tous les $\tilde{\mathbf{e}}_1$ tels que $Q^{[\ell]}\tilde{\mathbf{e}}_1 = s + [\ell]$ avec $Q^{[\ell]} \in \mathbb{F}_2^{\ell \times (k+\ell)}$. Pour trouver $\tilde{\mathbf{e}} = (\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2)$, nous choisissons $\tilde{\mathbf{e}}_2$ égal aux derniers $n-k-\ell$ coordonnées de $Q\tilde{\mathbf{e}}_1 + \tilde{\mathbf{s}}$. Le but de la phase de recherche est de trouver les vecteurs $\tilde{\mathbf{e}}_1$ efficacement. Nous présentons dans la suite une technique classique, l'algorithme de Stern, et notre nouvel algorithme qui utilise le technique des représentations généralisée. Les chapitres 9 et 10 donnent plus de détails sur les algorithmes d'ISD précédents et récents.

Algorithme de Stern – ISD classique Stern [Ste89] propose de chercher des vecteurs $\tilde{e}$ avec p positions non-nulles dans ses premières k positions, des zéros dans les ℓ positions suivantes et qui ont le plus de leur poids dans les derniers $n-k-\ell$ coordonnées. Il exécute une recherche de collisions pour trouver les p colonnes de $\tilde{H}$ telles que leur sommes est égale au syndrome sur les premières ℓ coordonnées (représenté par $\tilde{e}_1$ dans le paragraphe précédent).

La figure 0.5 présente la distribution de poids des vecteur $\tilde{e}$. Les entiers p et ℓ sont des paramètres d'optimisation. Nous partageons les colonnes de $Q_{[\ell]}$, dénotées par q_i, en deux

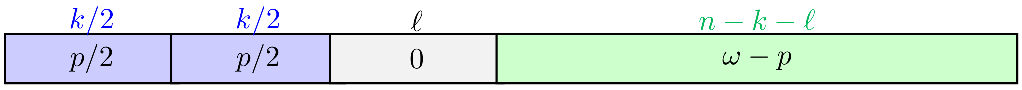


Figure 0.5.: *Distribution de poids de $\tilde{e}$ dans l'algorithme de Stern.*

ensembles de taille $k/2$:

$$Q_1 = \{\mathbf{q}_i \,|\, i \in [1, \frac{k}{2}]\} \text{ and } Q_2 = \{\mathbf{q}_i \,|\, i \in [\frac{k}{2}+1, k]\} \ .$$

Les ensembles des indices I_1 et I_2 correspondent aux colonnes dans Q_1 et Q_2, respectivement. Le problème de collisions est le suivant:

$$\sum_{i \in I_1} \mathbf{q}_i = \tilde{\mathbf{s}}_{[\ell]} + \sum_{i \in I_2} \mathbf{q}_i \tag{0.6}$$

avec $I_1 \subset \left[1, \frac{k}{2}\right]$, $I_2 \subset \left[\frac{k}{2}+1, k\right]$ et $|I_1| = |I_2| = \frac{p}{2}$.

L'algorithme crée des listes $\mathcal{L}_1, \mathcal{L}_2$ contenant toutes les sommes possibles des cotés gauche et droit de (0.6). Les listes contiennent $L = \binom{(k+\ell)/2}{p/2}$ éléments chacunes. Une collisions entre les listes est équivalent à un ensemble $I = I_1 \cup I_2$ de p colonnes dont la somme est égale au syndrome sur les premières ℓ coordonnées. Nous attendons $L^2/2^\ell$ collisions en moyenne. Si $Q^{[\ell+1,n-k]}\tilde{\mathbf{e}}_1 + \tilde{s}_{[\ell]}$ a un poids au moins de $\omega - p$ nous choisissons les bonnes colonnes de $\tilde{H}$ parmi les $n-k-\ell$ dernières pour trouver $\tilde{e}$.

Le temps de calcul de chaque itération est donné par la taille des listes et le nombre de collisions:

$$\mathcal{T}_{Stern} = \max(\binom{k/2}{p/2}, \frac{\binom{k/2}{p/2}^2}{2^\ell}) \ .$$

La probabilité que nous ayons choisi la bonne permutation dans la première partie de l'algorithme est

$$\mathcal{P}_{Stern} = \frac{\binom{k/2}{p/2}^2 \binom{n-k-\ell}{\omega-p}}{\binom{n}{\omega}} \ .$$

Le temps de calcul total attendu est

$$\mathcal{T}_{Stern} \cdot \mathcal{P}_{Stern}^{-1} \ .$$

La complexité peut être minimisée sous les contraintes que $0 \le p \le \omega$ et $0 \le \ell \le n-k-\omega+p$. Nous obtenons un temps de calcul du pire cas pour le décodage borné de $2^{0.05562\,n}$ en utilisant $2^{0.0134\,n}$ de mémoire.

Application de la technique des représentations généralisée Nous désirons résoudre le problème de collision (0.6):

$$\sum_{i\in I_1}\mathbf{q}_i = \tilde{\mathbf{s}}_{[\ell]} + \sum_{i\in I_2}\mathbf{q}_i \ .$$

Dans l'algorithme de Stern, les ensembles des indices I_1, I_2 sont choisis disjoints. Chaque solution I a une représentation unique (I_1, I_2) et est donnée par $I_1 \cup I_2$. Nous proposons de choisir les indices dans I_1 et I_2 parmi l'intervalle complet et de permettre quelques même colonnes dans I_1 et I_2. Les ensembles ont une taille $|I_1| = |I_2| = \frac{p}{2} + \varepsilon$ pour un nouveau paramètre $\varepsilon > 0$ et nous énumérons ceux qui ont exactement ε colonnes au commun. Chaque solution peut être écrite de plusieurs manières comme $I = I_1 \Delta I_2 := I_1 \cup I_2 \setminus (I_1 \cap I_2)$ comme montré dans la figure 0.6.

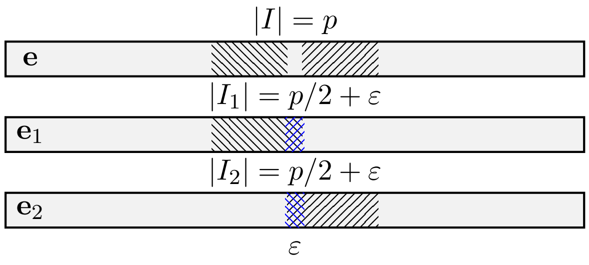


Figure 0.6.: *Décomposition d'un ensemble des indices I en deux.*

Le nombre des représentations est

$$N = \binom{p}{p/2}\binom{k+\ell-p}{\varepsilon} \ . \tag{0.7}$$

Le vecteur $\mathbf{e}$ est creux pour des codes aléatoires et p est petit en comparaison de $k+\ell$ ce qui veut dire que le nombre des représentations N augmente vite pour un petit ε à cause du deuxième facteur dans (0.7).

Comme plein de couples (I_1, I_2) produisent la même solution, nous pouvons ajouter une contrainte aux ensembles d'indices. Cela réduit le nombre d'éléments parmi lesquels nous cherchons la solution et donc réduit la taille de listes construites dans l'algorithme. Nous supposons que les sommes des colonnes sont des valeurs uniformément distribuées dans $\mathbb{F}_2^\ell$ car la matrice donnée est indistinguable à une matrice aléatoire.
La probabilité que

$$(\sum_{i\in I_1}\mathbf{q}_i)_{[r]} = \mathbf{t} \tag{0.8}$$

pour un vecteur aléatoire $\mathbf{t} \in \mathbb{F}_2^r$ est 2^{-r} pour $0 \le r \le \ell$. Si nous choisissons

$$r \approx \log_2 N \ ,$$

nous pouvons nous attendre à ce qu'un ensemble des indices I_1, qui correspond à une représentation (I_1, I_2), satisfasse (0.8).

L'ensemble $I \setminus I_1$ satisfait alors

$$(\sum_{i \in I \setminus I_1} \mathbf{q}_i)_{[r]} = \mathbf{s}_{[r]} + \mathbf{t} \quad . \tag{0.9}$$

Inversement, pour des ensembles d'indices arbitraires $(J_1, J_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ satisfaisant (0.8) et (0.9), nous savons que

$$(\sum_{i \in J_1 \cup J_2} \mathbf{q}_i)_{[r]} = \mathbf{s}_{[r]} \quad .$$

Il nous reste à tester si l'égalité est satisfaite sur toutes les ℓ coordonnées, ce qui est vrai avec une probabilité de 2^{-r}, et si $|J_1 \cap J_2| = \varepsilon$. Dans ce cas nous avons trouvé une solution $I = J_1 \cup J_2 \setminus (J_1 \cap J_2)$ et (J_1, J_2) est une des représentations. Pour quelques cibles $\mathbf{t}$ il peut arriver qu'aucune représentation ne satisfasse (0.8) et (0.9). Dans ces cas, il nous faut choisir un nouveau vecteur de cible et répéter l'algorithme.

La section 10.2 explique la nouvelle technique et l'algorithme en détail. Notre algorithme atteint un temps de calcul pour le décodage borné de $2^{0.04933\,n}$ et utilise $2^{0.0307\,n}$ de mémoire. Le temps de calcul est toujours plus court comme nous le montrons dans la section 10.3.

Introduction and motivation

The focus of this thesis are algorithmic techniques to solve the random subset-sum problem over the integers and the syndrome-decoding problem in a random linear code.

Given a large set of n integers a_i of maximal size $M = \log_2(\max_i a_i)$, the subset-sum problem asks whether there exists a subset that sums up to a fixed value. It is proven to be NP-complete. The ratio between the number of elements and their maximal size M is called density. This mathematical problem provides an alternative to other hard problems used in cryptography such as factoring or the discrete logarithms. Its description is simple and the computation of sums of integers is an easy task. Furthermore, in contrast to widely used number-theoretic problems no polynomial-time quantum algorithm is known.

One can construct one-way functions, pseudo-random generators and secure private-key cryptography schemes from the hardness assumption of an average-case subset-sum problem. Especially, in the 1970s and 1980s cryptographers showed an interest in designing public-key encryption schemes where security was directly based on this problem. However, extensive research was undertaken and the security of the proposed schemes was undermined quickly after the proposal. The weakness was due to a badly hidden trapdoor needed for an efficient decryption or due to a low density, that is, a density less than one. The provably hardest case occurs for a random subset sum problem of density close to one which is the case we study. We present a new generic algorithmic technique that assumes no underlying structure in the set of elements.

Aside from the public-key encryption, the subset-sum problem allows efficient constructions of pseudo-random generators and universal one-way hash-functions that can be used to implement private-key encryption, zero-knowledge protocols, identification schemes and digital-signature schemes. Their intractability is given by the hardness of the subset-sum problem. There are also cryptosystems based on lattice problems which are provably as secure as the hardness of the subset-sum problem in the average case.

Apart from its form over the integers, the subset-sum problem appears in other places in cryptography. Decoding problems can be seen as a vectorial subset-sum problem. Especially the bounded-distance-decoding problem in a random code permits public-key encryption, McEliece's and Niederreiter's encryption scheme, as well as digital signatures, namely the CFS and the parallel-CFS signature scheme. The generic tools to solve the integer subset sum problem apply and lead to advanced algorithms for solving the underlying decoding problem as we present in part III.

Decoding a random linear code is one of the most promising problems for the design of cryptosystems that are secure even in the presence of quantum computers. The problem is proven to be NP-complete for binary linear codes and no polynomial time quantum algorithm is known. The security of almost all code-based cryptosystems, for example, the McEliece cryptosystem, relies on the fact that random linear codes are hard to decode. The public key belongs to a code that is supposed to be indistinguishable from a random code. To permit an efficient decoding on the receivers end, one needs to embed a trapdoor. Starting with a well-structured secret code C, we transform it linearly into a code C' that shows no structure.

An attacker has two choices. First, she can try to distinguish the scrambled code C', derived from the well-structured code C, from a random code by revealing the underlying structure. Second, she directly tries to run a generic decoding algorithm on the scrambled code C' thus attacking the random decoding problem. Research shows that there are code families for which the structure can be efficiently disguised. We thus focus on decoding random linear codes to estimate the difficulty of the adversary.

Closely related to decoding random linear codes is the learning-parity-with-noise problem (LPN) that is frequently used in cryptography [Ale03, HB01, KKC+01]. In LPN, one directly starts with a random linear code C and the LPN search problem is a decoding problem in C. It was shown in [Reg05] that the popular LPN decision variant, a very useful tool for many cryptographic constructions, is equivalent to the LPN search problem. This means that is equivalent to decoding a random linear code.

The learning-with-error problem (LWE) by Regev [Reg05] is a generalization of LPN to codes over a larger field. The presented decoding algorithms could be adjusted to work for these larger fields (similar to what was done in [CG90, Pet10]). Since the decoding problem lies at the the heart of coding-based and LPN/LWE-based cryptography it is necessary to study its complexity in order to define proper security parameters for cryptographic constructions.

Results and organization

This thesis is divided into three parts. The first part presents algorithmic techniques for collision search which is an important tool that we will use later. We review the k-sum problem which is related to the subset-sum problem. The well-known algorithms for its solution represent a first example of an application of the collisions search technique. The second and third part present recent research to solve the subset-sum problem and the bounded decoding problem.

Until recently, the most efficient generic algorithm to solve the subset-sum problem with n elements was proposed by Shamir and Schroeppel in 1981 and requires $2^{\frac{n}{2}}$ time and $2^{\frac{n}{4}}$ memory. In 2010, Howgrave-Graham and Joux presented a new algorithmic technique that we call representation technique. The resulting probabilistic algorithm runs in expected time $2^{0.337\,n}$ and needs $2^{0.311\,n}$ of memory on average on random, hard instances. We provide a more extensive complexity analysis. Due to our detailed study in the unbalanced case, we propose a variable number of levels of the algorithm or to change to the complementary problem to achieve a minimal running time. Further improvements on the memory requirement allow us to propose an algorithm of heuristically same asymptotic time using less memory $2^{0.272\,n}$.

The representation technique can be extended by allowing negative coefficients in intermediate solutions. This generalization improves the search for a solution of a random, hard subset-sum problem. The algorithm has a heuristic running time and memory requirement of $2^{0.291\,n}$ and lead to the publication [BCJ11]. The memory requirement can be reduced by heuristic assumptions to $2^{0.279\,n}$ while the asymptotic running time stays the same.

Part II is organized as follows. Chapter 2 introduces the subset-sum problem and defines the setting of interest. We also give a short overview of proposed cryptographic primitives based on the subset-sum problem and related schemes whose security depends on the subset-sum problem. The following chapters present generic algorithmic techniques to solve the subset-sum problem on a random, hard instance. We distinguish between classical birthday-paradox attacks as the algorithm by Shamir and Schroeppel (chapter 3) and algorithms that use the more recent representation technique in its basic (chapter 4) and extended form (chapter 5). Chapter 6 presents an algorithm of constant memory requirement which is part of the publication [BCJ11]. Chapters 2 and 3 follow in part chapter 8 from [Jou09] while chapter 4 and 5 and are based on the publications [HGJ10] and [BCJ11], respectively.

The most efficient method to solve the bounded-distance-decoding problem for a random linear code is information-set decoding. A classical approach leads to an asymptotic worst-case running time $2^{0.05564\,n}$. The worst case is attained for an information rate that maximizes the time to solve the decoding problem. Ball-collision decoding, recently proposed, obtains a better result of running time $2^{0.05559\,n}$. An application of the representation technique proves to be very effective. A simple representation technique that permits no intersections allows to recover the error vector in time $2^{0.05363\,n}$ with a small increase in memory. We propose a variant that reduced the memory requirement to $2^{0.0139\,n}$ under heuristic assumptions.

The algorithm does not use the full power of the representation technique. Permitting intersections in some bit positions, analogously to the extended representation technique, allows to solve the syndrome-decoding problem over a random linear code to be solved in time $2^{0.04933\,n}$ using $2^{0.0307\,n}$ space. The result is published in [BJMM12]. Heuristically, this memory requirement can be reduced to $2^{0.0306\,n}$.

Part III is organized as follows. Chapter 8 gives an introduction to linear error-correcting codes and presents popular code-based cryptography: McEliece's and Niederreiter's public-key-encryption schemes and the CFS signature scheme. Chapter 9 introduces the general setting for information-set decoding and explains previous ideas. We present our improved information-set-decoding algorithm in chapter 10. The first chapter is inspired by [Bar98, Pet11] while the chapters 9 and 10 are based on the publication [BJMM12].

Appendix A gives an introduction to asymptotic approximations that we use later to estimate the asymptotic complexity of the algorithms. We present the octave code used to evaluate the complexity of the different algorithms in appendix B.

Part I

Algorithmic prerequisites

Chapter 1

Collision search: A valuable tool

An important algorithmic tool that has many applications in cryptology, in particular to solve the knapsack problem and the syndrome decoding problem, is the efficient search for collisions within two lists $\mathcal{L}_1, \mathcal{L}_2$ of elements. We denote the list of collisions by $\mathcal{L}_1 \bowtie \mathcal{L}_2$ and the number of elements in $\mathcal{L}_j$ by L_j. A collision is the appearance of the same element in each of the lists. A naive approach, checks all possible tuples of elements in the two lists leading to a long running time of $L_1 \cdot L_2$. More efficient algorithms use hash tables or sorting methods. Assume that $L_1 \leq L_2$ and that all elements within each list are different. One classic method is the *hash-join* that computes and stores hash values of the elements of the shorter list, $\mathcal{L}_2$, and then checks for each element of the second list, $\mathcal{L}_2$, if its hash value appears in the stored hash table. The table look-up is done in constant time. The algorithm requires $L_1 + L_2$ simple computations and requires to store the smaller of the two lists in a hash table.

A *merge-join* algorithm sorts the elements of the lists before and then searches for matching elements. Sorting the lists costs $L_1 \log L_1 + L_2 \log L_2$. The merge-join reads simultaneously one entry in each list and compares their values starting from the smallest elements. If no match is found, the smaller entry is discarded and the next element is considered. Since both lists are sorted this value can not match to any other entry in the other list. The algorithm stops if one of the lists is scanned completely. As both lists can potentially be read until the end, the maximal cost is $L_1 + L_2$. The total time complexity is $L_1 \log L_1 + L_2 \log L_2 + L_1 + L_2$ with a memory requirement of $L_1 \log L_1 + L_2 \log L_2$. We can also sort only one of the lists resulting in a total time complexity of $L_1 \log L_1 + L_2 \log L_1$.

According to the above complexity analysis, we see that a hash-join is in favor of a merge-join as the dominating space requirement is the shorter of the two lists and no sorting routine is needed which reduces the time. Depending on the size of the lists and the accessible memory, the sequential access to the lists may however be faster than a random access to a hash table. Sorting can also be performed by an external machine leading to a fast and memory efficient merge-join.

For later use in this work, we will often assume that both lists are of same size. Also, the developed algorithms that use a join routine are of exponential complexity such that logarithmic factors represent a minor cost. For the sake of simpler presentation, we therefore stick to a merge-join routine keeping in mind that an implementation can be realized by use of an efficient hash-join.

Section 1.1 describes a merge-join algorithm for integers that we later use as a subroutine to solve the knapsack problem (part II). The merge-join algorithm in section 1.2 deals with binary vectors and proves to be useful to solve the syndrome decoding problem (part III). Both join routines are based on a classical algorithm such as presented in [Knu98, Wag02].

1.1. Searching collisions between integers

We are given two lists of integers $\mathcal{L}_1$ and $\mathcal{L}_2$ together with two integers M and $R \in \mathbb{Z}_M$ and want to compute the list of elements $\mathcal{L}_R$ where

$$\mathcal{L}_R = \{x + y \mid x \in \mathcal{L}_1, y \in \mathcal{L}_2 \text{ such that } x + y \equiv R \mod M\} \text{ of size } L_R.$$

To find the elements of $\mathcal{L}_R = \mathcal{L}_1 \bowtie \mathcal{L}_2$, we follow the pseudo-code as given in algorithm 1.1 and described below.

Algorithm 1.1: MERGE-JOIN – Find collisions between lists $\mathcal{L}_1$ and $\mathcal{L}_2$ of integers.

```
Input:   $\mathcal{L}_1$, $\mathcal{L}_2$, $M$, $R \in \mathbb{Z}_M$, (additional parameters for filtering)
Output:  $\mathcal{L}_R = \mathcal{L}_1 \bowtie \mathcal{L}_2$,C
Sort the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ (by increasing order of the values modulo $M$)
Set counter $C \leftarrow 0$
For each Target $\in \{R, R+M\}$
    Set $i \leftarrow 0$ and $j \leftarrow L_2 - 1$
    While $i < L_1$ and $j \geq 0$
        Set Sum $\leftarrow (\mathcal{L}_1[i] \mod M) + (\mathcal{L}_2[j] \mod M)$
        If Sum < Target then Increment $i$
        Else if Sum > Target then Decrement $j$
        Else
            Set $i_0, i_1 \leftarrow i$
            While $i_1 < L_1$ and $\mathcal{L}_1[i_1] \equiv \mathcal{L}_1[i_0] \pmod M$ Increment $i_1$
            Set $j_0, j_1 \leftarrow j$
            While $j_1 \geq 0$ and $\mathcal{L}_2[j_1] \equiv \mathcal{L}_2[j_0] \pmod M$ Decrement $j_1$
            For $i \leftarrow i_0$ to $i_1 - 1$
                For $j \leftarrow j_1 + 1$ to $j_0$
                    Increment $C$
                    Append $\mathcal{L}_1[i] + \mathcal{L}_2[j]$ to $\mathcal{L}_R$ (unless filtered out, e.g., duplicates)
            Set $i \leftarrow i_1$ and $j \leftarrow j_1$
```

We first sort the two input lists in increasing order with respect to the values modulo M. Two pointers i and j index the current element of each list and are initially set to the first and last element of the list $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. We then compute consecutively the sum of the modulo values of the two current elements: $(\mathcal{L}_1[i] \mod M) + (\mathcal{L}_2[j] \mod M)$. The value is bounded by $2M$ such that all elements whose sum equals to R or $R + M$ need to be found. We perform the search twice, once for each target R and $R + M$. Whenever the sum exceeds the target, we go to the next smaller element in the list $\mathcal{L}_2$ by decrementing j. If the sum is smaller than the target, we move to the next larger element in the list $\mathcal{L}_1$ by incrementing i. If equality holds, we have found a collision. The following elements in both

lists might be the same such that all possible combinations lead to a collision. We fix in turn the pointer for the first and second list and check the consecutive elements of the other list for collisions. An optional filtering process may test the colliding elements for consistency to additional conditions. We may for example want to exclude all duplicates. The algorithm stops when the pointer has reached the end of $\mathcal{L}_1$ or the first element of $\mathcal{L}_2$ while searching with the target $R + M$.
Using a slight variation of algorithm 1.1, it is also possible to find collision over the integers, that is, given $\mathcal{L}_1$ and $\mathcal{L}_2$ together with a target integer R to construct the set

$$\mathcal{L}_R = \{x + y \mid x \in \mathcal{L}_1, y \in \mathcal{L}_2 \text{ s.t. } x + y = R\} \quad .$$

The only differences are that we sort the lists by value (not by modular values) and then run the loop with a single target value R.

Complexity. One needs to store the lists $\mathcal{L}_1, \mathcal{L}_2$ and $\mathcal{L}_R$ leading to a memory requirement

$$\mathcal{M}_{\text{Merge-Join}} = L_1 \log L_1 + L_2 \log L_2 + L_R \log L_R \quad .$$

The initial sorting can be performed in time $L_1 \log L_1 + L_2 \log L_2$. The algorithm computes the sum of two elements in each step and discards one until one list is scanned completely. If no collision exists, both lists may be read until there end resulting in $L_1 + L_2$ computations. If a match is found, all consecutive elements of the other lists may produce a match as well which we will store in $\mathcal{L}_R$. The algorithm 1.1 requires $L_1 \log L_1 + L_2 \log L_2 + L_1 + L_2 + L_R$ computations.

If we can assume that the values of the initial lists modulo M are uniformly distributed, the expected value of L_R is $L_1 \cdot L_2 / M$. Depending on additional conditions, we can also discard collisions reducing the size of $\mathcal{L}_R$ and thus reducing the memory requirement. We introduced a counter C in algorithm 1.1 that counts the collisions. It may be larger than L_R if not all collisions satisfy the filtering condition. We express the time complexity in this case as

$$\mathcal{T}_{\text{Merge-Join}} = L_1 \log L_1 + L_2 \log L_2 + \max(L_1 + L_2, C) \quad .$$

For uniformly distributed elements, we approximate C by $L_1 \cdot L_2 / M$. In the following we neglect logarithmic factors and write

$$\mathcal{T}_{\text{Merge-Join}} = \tilde{\mathcal{O}}\left(\max(L_1, L_2, C)\right)$$

and

$$\mathcal{M}_{\text{Merge-Join}} = \tilde{\mathcal{O}}\left(\max(L_1, L_2, L_R)\right) \quad .$$

Variants for implementation. We already mentioned in the introduction to this section that a hash-join is a good alternative. Using the here presented merge-join algorithm, we can also save memory in practice if we can produce the elements of the starting lists on the fly. We store only the smaller of the lists in sorted form, say $\mathcal{L}_1$. We then generate each element of $\mathcal{L}_2$ on the fly and search for an collision. The memory requirement is thus changed to $L_1 \log L_1 + L_R \log L_R$. Using hash tables, we gain a logarithmic factor in time.

It is also possible to store the positions where the value modulo M changes in the sorted list. In this way, we omit to search the corresponding indices for the second iteration with target $R + M$.

1.2. Searching collisions in the vectorial case

Given a matrix $Q \in \mathbb{F}_2^{\ell \times (m)}$, a target $\mathbf{t} \in \mathbb{F}_2^{\ell}$ and two lists $\mathcal{L}_1$ and $\mathcal{L}_2$ containing binary vectors of length m and weight $\frac{p}{2} + \varepsilon$. Let x and y be two elements from $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. We furthermore know that the sum of columns indexed by $x \oplus y$ equals the target $\mathbf{t}$ on its $r \leq \ell$ right most coordinates: $Q(x \oplus y) = \mathbf{t}$. We search all elements for which the above equality holds on all ℓ bits and whose sum has weight p. The resulting list $\mathcal{L}$ is denoted by $\mathcal{L}_1 \bowtie \mathcal{L}_2$. The join process is illustrated in figure 1.1.

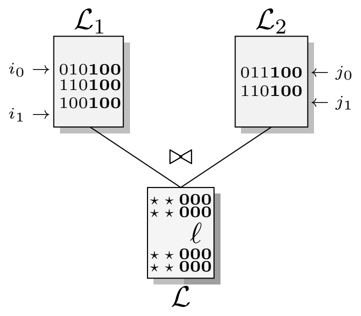


Figure 1.1.: *Illustration of the vectorial merge-join algorithm to obtain* $\mathcal{L} = \mathcal{L}_1 \bowtie \mathcal{L}_2$.

Note that the weight of a sum of two vectors can differ from p depending on the number of ones at the same coordinates. Also, we can obtain several times the same elements. We filter out all such inconsistent collisions. The complete MERGE-JOIN-DECODE algorithm is given as algorithm 1.2 and performs the following steps. Sort the first list in lexicographical order according to the labels $\mathcal{L}_1(x) := (Qx)$ and the second list according to the labels $\mathcal{L}_2(y) := (Qy) + t$. A match of same labels is a collision for which $Q(x \oplus y)_{[r]} = \mathbf{t}$. It remains to filter out the inconsistent and double elements.

To detect all collisions, one initializes two pointers i and j starting at the beginning of the list $\mathcal{L}_1$ and the end of list $\mathcal{L}_2$, respectively. As long as the current elements do not collide, as they differ in the label, we either increase i or j depending on the relative order of the labels. If $\mathcal{L}_1(x_i) < \mathcal{L}_2(y_j)$, we increase i else j. Once a collision occurs, that is, $\mathcal{L}_1(x_i) = \mathcal{L}_2(y_j)$, we need to check if the consecutive elements have the same labels. All possible tuples are collisions. We initialize four auxiliary counters i_0, i_1 and j_0, j_1 with i and j, respectively. The indices i_0 and j_0 mark the first collision. Then i_1 and j_1 can further be incremented as long as the list elements retain the same labels. This procedure defines two sets $C_1 = \{x_{i_0}, \ldots, x_{i_1}\}$ and $C_2 = \{y_{j_0}, \ldots, y_{j_1}\}$ for which all possible combinations yield a collision.

Algorithm 1.2: Merge-Join-Decode – Find collisions between lists $\mathcal{L}_1$ and $\mathcal{L}_2$ of binary vectors.

```
**Input:**   $\mathcal{L}_1, \mathcal{L}_2, r, p$ and $t \in \mathbb{F}_2^r$
**Output:**  $\mathcal{L} = \mathcal{L}_1 \bowtie \mathcal{L}_2, C$
Lexicographically sort $\mathcal{L}_1$ and $\mathcal{L}_2$ according to the labels $\mathcal{L}_1(x_i) := (Qx_i)$ and
$\mathcal{L}_2(y_j) := (Qy_j) + t$.
Set collision counter $C \leftarrow 0$.
Let $i \leftarrow 0$ and $j \leftarrow (L_2 - 1)$
While $i < L_1$ and $j < L_2$
    If $\mathcal{L}_1(x_i) <_{lex} \mathcal{L}_2(y_j)$ then Increment $i$
    Else if $\mathcal{L}_1(x_i) >_{lex} \mathcal{L}_2(y_j)$ then Increment $j$
    Else
        Let $i_0, i_1 \leftarrow i$ and $j_0, j_1 \leftarrow j$
        While $i_1 < L_1$ and $\mathcal{L}_1(x_{i_1}) = \mathcal{L}_1(x_{i_0})$ Increment $i_1$
        While $j_1 < L_2$ and $\mathcal{L}_2(y_{j_1}) = \mathcal{L}_2(y_{j_0})$ Increment $j_1$
        For $i \leftarrow i_0$ to $i_1 - 1$
            For $j \leftarrow j_0$ to $j_1 - 1$
                Increment $C$
                Insert collision $x_i + y_j$ into list $\mathcal{L}$ (unless filtered out)
        Let $i \leftarrow i_1, j \leftarrow j_1$
```

We remove on the fly solutions with incorrect weight, $\mathrm{wt}(x_i+y_j) \neq p$, and duplicate elements $x_i + y_j = x_k + y_\ell$ and add all remaining elements to the list $\mathcal{L}$. The procedure continues at the positions $i \leftarrow i_1$ and $j \leftarrow j_1$ until the end of one list is reached. The collision counter C in the algorithm allows us to measure the time spent for removing inconsistent solutions and duplicates. We denote the length of the input lists by L_1, L_2 and the number of returned elements by L. The total running time of Merge-Join-Decode (algorithm 1.2) is

$$\mathcal{T}_{\text{Merge-Join-Decode}} = L_1 \log L_1 + L_2 \log L_2 + \max(L_1 + L_2, C)$$

using space

$$\mathcal{M}_{\text{Merge-Join-Decode}} = L_1 \log L_1 + L_2 \log L_2 + L \log L \ .$$

In the asymptotic setting, we will neglect logarithmic factors and simplify the time and memory cost to

$$\mathcal{T}_{\text{Merge-Join-Decode}} = \tilde{\mathcal{O}}\left(\max(L_1, L_2, C)\right)$$

and

$$\mathcal{M}_{\text{Merge-Join-Decode}} = \tilde{\mathcal{O}}\left(\max(L_1, L_2, L)\right) \ .$$

Assuming uniformly distributed labels, the probability of a collisions in $\mathcal{L}_1 \times \mathcal{L}_2$ is $2^{\ell - r}$. We can then estimate the number of collisions: $\mathbb{E}\left[C\right] = \frac{L_1 \cdot L_2}{2^{\ell - r}}$.

1.3. Birthday problem and k-sum problem

The collision search algorithms can be applied to solve the birthday problem. It is a well known combinatorial tool that we state in its binary form.

Definition 1.1 (Birthday problem or collision problem)
Given two lists $\mathcal{L}_1, \mathcal{L}_2$ of elements drawn independently and uniformly at random in $\{0,1\}^n$, find $x_1 \in \mathcal{L}_1$ and $x_2 \in \mathcal{L}_2$ such that $x_1 = x_2$.

We can make use of a join algorithm that finds collisions to find a single solution similar to the algorithms in section 1. Allowing more lists, generalizes the combinatorial problem to the k-sum problem:

Definition 1.2 (k-sum problem)
Given k lists $\mathcal{L}_1, .., \mathcal{L}_k$ of elements drawn independently and uniformly at random in $\{0,1\}^n$, find $x_1 \in \mathcal{L}_1, .., x_k \in \mathcal{L}_k$ such that $x_1 \oplus .. \oplus x_k = 0^n$.

To solve a more general equality with a target $s \in \{0,1\}^n$, i.e., $x_1 \oplus .. \oplus x_k = s$, we can modify the last list to elements $x_k \oplus s$ for $x_k \in \mathcal{L}_k$. We can hence restrict ourselves to explain how to find one solution for the case $s = 0^n$.

1.3.1. How to solve the k-sum problem

Consider a 4-sum problem. We are given four lists $\mathcal{L}_1, .., \mathcal{L}_4$ of independently and equally distributed bitstrings of length n and we want to find four elements (each in one of the lists) whose XOR is zero. Let L_i denote the size of list $\mathcal{L}_i$ for $i = 1, .., 4$. If $L_1 \cdot L_2 \cdot L_3 \cdot L_4 \gg 2^n$, we expect to find a solution with good probability. Observe that for a solution where

$$x_1 + x_2 + x_3 + x_4 = 0^n, \tag{1.1}$$

necessarily $x_1 + x_2 = x_3 + x_4$. It is hence natural to create sums of elements $x_1 + x_2 \in \mathcal{L}_1 \times \mathcal{L}_2$ and $x_3 + x_4 \in \mathcal{L}_3 \times \mathcal{L}_4$ in a first step. A collision between these two lists then detects a solution to (1.1).

To reduce the size of the intermediate lists, we can impose a constraint on the elements in $\mathcal{L}_1 \bowtie \mathcal{L}_2$ and $\mathcal{L}_3 \bowtie \mathcal{L}_4$: We denote by $[x]_t$ the lower t bits of an element x in $\{0,1\}^n$. If $x_1 + x_2 = x_3 + x_4$, then $[x_1 + x_2]_t = [x_3 + x_4]_t$ for all $1 \leq t \leq n$. We can guess the intermediate value $M_t = [x_1 + x_2]_t$ of a solution and create all elements $(x_1, x_2) \in \mathcal{L}_1 \times \mathcal{L}_2$, $(x_3, x_4) \in \mathcal{L}_3 \times \mathcal{L}_4$ which correspond to $M_t \lessapprox 2^t$. Again, a collision leads to the solution. As the correct value of M_t is not known, we have to try all 2^t possible values. For a random knapsack, we expect that the values of $[x_1 + x_2]_t, [x_3 + x_4]_t$ are equally distributed and independent. The probability that $[x_1 + x_2]_t = [x_3 + x_4]_t = M_t$ is then 2^{-t}.

Algorithm. Based on these observations, we can develop the following algorithm. We are given four lists $\mathcal{L}_j$ of same size $L = 2^\ell$ containing elements x_i drawn independently and uniformly at random in $\{0,1\}^n$. Two merges create elements $x_1 + x_2$ and $x_3 + x_4$ that correspond to the target M_t on the lower t coordinates. We expect to find about $L^2/2^t = 2^{2\ell - t}$

elements for the lists $\mathcal{L}_1 \bowtie \mathcal{L}_2$ and $\mathcal{L}_3 \bowtie \mathcal{L}_4$. In a second step, we search collisions between $x_1 + x_2$ and $x_3 + x_4$. For each M_t, this happens with probability 2^{t-n} and we expect to find $2^{4\ell-t-n}$ vectors in the final join per M_t.

Using a hash-join, the required memory is

$$L \log L + min(|\mathcal{L}_1 \bowtie \mathcal{L}_2| \log(|\mathcal{L}_1 \bowtie \mathcal{L}_2|), |\mathcal{L}_3 \bowtie \mathcal{L}_4| \log(|\mathcal{L}_3 \bowtie \mathcal{L}_4|) + N_{sol} \log N_{sol})$$

where N_{sol} denotes the final number of solutions. In the worst case, we need to loop over all 2^t values for M_t. The running time becomes

$$2^t \cdot (L + |\mathcal{L}_1 \bowtie \mathcal{L}_2| + |\mathcal{L}_3 \bowtie \mathcal{L}_4| + N_{sol}) \ .$$

If we assume that the elements in the base lists are uniformly distributed, we approximate $|\mathcal{L}_1 \bowtie \mathcal{L}_2|$ and $|\mathcal{L}_3 \bowtie \mathcal{L}_4|$ by $2^{2\ell-t}$. The expected running time becomes $\mathcal{O}\left(max(2^{\ell+t}, 2^{2\ell}, N_{sol})\right)$ using $\mathcal{O}\left(M \log M\right)$ space where $M = max(2^\ell, 2^{2\ell-t})$.

How to choose ℓ and t. Depending on the choice of ℓ, we have to adapt t such that the last join finds at least one solution which means that $t \leq 4\ell - n$.

A typical choice for the starting lists is a size $2^{\frac{n}{4}}$ and $t \approx \frac{n}{4}$ to minimize the memory requirement as done by Chose, Joux and Mitton [CJM02] to speed-up parity checks. The algorithm finds one solution on average in time $\mathcal{O}\left(n\, 2^{\frac{n}{2}}\right)$ using memory $\mathcal{O}\left(n\, 2^{\frac{n}{4}}\right)$.

If we can create more elements in the base lists $\mathcal{L}_1, .., \mathcal{L}_4$, we can reduce the running time of the above presented algorithm at cost of an increased memory requirement. Wagner showed [Wag02] in 2002 that the k-sum problem can be solved with a time and memory complexity of at most $\mathcal{O}\left(n\, 2^{n/3}\right)$, given that the base lists can be extended freely with elements drawn independently and uniformly at random to about size $2^{n/3}$. This in return means that there are sufficiently many solutions to the problem.

Consider $k = 4$ and lists of size $L = 2^\ell$. To balance the memory, we choose $t = \ell$ and obtain about $2^{2\ell-(n-l)} = 2^{3\ell-n}$ colliding elements on average after the final join. Hence, if we choose $\ell \geq n/3$, we can expect to find at least one solution on average. The minimal time and memory complexity is then $\mathcal{O}\left(n\, 2^{n/3}\right)$ which occurs for base lists of size $2^{n/3}$ where $\ell = n/3$.

The idea can be applied to the general case of k lists. A solution can be expected if $L \geq 2^{n/(log\, k+1)}$. This k-tree algorithm [Wag02] requires at most time and space $\tilde{\mathcal{O}}\left(k\, 2^{n/(1+\log k)}\right)$ with lists of size $\tilde{\mathcal{O}}\left(2^{n/\log k}\right)$.

In 2009, Minder and Sinclair [MS09] extended the k-tree algorithm by Wagner in that they permit smaller starting lists. For $2^{n/k} \leq L \leq 2^{n/(log\, k+1)}$, they present an algorithm that trades memory against time. The basic idea is to allow a flexible number of bits that are eliminated in each step.

Further related work. Already in 1991, Camion and Patarin presented a k-tree algorithm to break a knapsack-based hash-function. Blum, Kalai and Wasserman constructed a k-tree scheme for a proof in learning theory. Coron and Joux [CJ04] applied Wagner's algorithm to break a hash function based on error correcting codes. The attack was later refined in [AFS05].

Lyubashevsky [Lyu05] and Shallue [Sha08] presented a variant of Wagner's algorithm to solve the subset sum problem for random instances of high density.

In the case that $k \geq n$, the k-sum problem can be solved by Gaussian elimination in time $\mathcal{O}\left(n^3 + kn\right)$ as for example shown in [BM97].

Part II

Improving the search for a solution to the subset-sum problem

Chapter 2

The subset sum problem in cryptography

2.1. Definition, characteristics and assumptions

The binary knapsack problem is a general case of the subset-sum problem. It is an optimization problem and can be stated as follows:

Definition 2.1 (Binary knapsack problem)
Given a set of n positive integer values v_i, positive integer weights w_i and a maximal positive integer weight S, find binary x_i that maximize

$$\sum_{i=1}^{n} x_i v_i \;\; s.t.$$

$$\sum_{i=1}^{n} x_i w_i \leq S \;\; .$$

In the special case where $w_i = v_i$ and we ask for equality with S, we obtain the subset-sum problem.

Definition 2.2 (Subset-sum problem – SS)
Given a set of n integers a_i and an integer S, find all solutions $\mathbf{x} = (x_1, .., x_n) \in \{0,1\}^n$ such that

$$\mathbf{a} \cdot \mathbf{x} := \sum_{i=1}^{n} a_i \, x_i = S \tag{2.1}$$

or show that no solution exists.

The problem is NP-complete [Kar72] in its decisional form and its optimization form above is NP-hard. Let M denote the maximal size of the integers a_i, $M = \log_2(\max_i a_i)$. We then denote a subset-sum problem as defined above by $SS(n, M)$. We omit n, M if it is clear from the context. The subset-sum problem is also referred to as knapsack problem in cryptography. The integers a_i are called weights or knapsack elements. The set of weights is the knapsack. We will concentrate on the SS for usage in cryptography. The corresponding decision problem to SS is in NP as we can efficiently check a solution for validity and it is NP-complete [Ajt98, Kar72]. It can be reduced to the computational problem, SS, which is NP-hard. Also the computational problem can be reduced to the decisional problem such that is NP-hard as well. Under the conjecture that $P \neq NP$, we do not expect to find a polynomial-time algorithm that solves the problem exactly in all cases.

The hardness and the way how to solve an instance of the subset-sum problem depends on a property of the underlying knapsack which is called density.

Definition 2.3 (Density)
The ratio between the number of elements in the knapsack $\mathbf{a} = (a_1, .., a_n)$ *and the size of the largest element is called density,*

$$d := \frac{n}{\log_2(\max_i a_i)} \quad . \tag{2.2}$$

We are furthermore interested in knapsacks that have no apparent structure. For this purpose, we define a random knapsack or an average-case knapsack as follows.

Definition 2.4 (Random knapsack or average-case subset-sum problem)
Let the weights a_i *be chosen uniformly at random in the interval* $[1, \lfloor 2^{\frac{n}{d}} \rfloor]$. *Let the solution* $\mathbf{x}$ *be chosen uniformly at random in* $\{0,1\}^n$. *The subset sum* $S = \sum_{i=1}^n a_i x_i$ *is called a random knapsack with solution.*

Given a random knapsack problem, we can count the number of binary vectors of weight ℓ and length n. There are $\binom{n}{\ell}$ many which is maximal for $\ell = \lceil \frac{n}{2} \rceil$. We thus expect that the solution vector $\mathbf{x}$ has a Hamming weight, number of non-zero coordinates, $\ell \approx \lceil \frac{n}{2} \rceil$ and call the knapsack *equibalanced*. Consequently, if we split the solution of a random knapsack in k parts of same size, where $k \mid \ell$, we expect that each part has weight close to $\frac{\ell}{k}$. If ℓ differs considerably from $\frac{n}{2}$, we call the knapsack *unbalanced*.

Modular knapsack. A modular knapsack problem is the following problem:

$$\mathbf{a} \cdot \mathbf{x} := \sum_{i=1}^{n} a_i \, x_i \equiv S \mod M \tag{2.3}$$

where the knapsack elements are elements in $\mathbb{Z}_M$ for an integer M. Solving modular knapsacks or knapsacks over the integers are equivalent tasks if we neglect polynomial factors. Having to solve a knapsack problem over the integers and given an algorithm that solves any modular knapsack, we simply set $M = \sum_{i=1}^n a_i + 1$ to obtain a solution to the integer knapsack. Conversely, if we have to solve problem (2.3) and are given an algorithm that solves non-modular knapsacks, we apply the algorithm several times with different targets. The n weights a_i of the modular knapsack lie in $[0, M-1]$ and all possible sums are values in the range $[0, n(M-1)]$. The target $S \in \mathbb{Z}_M$ implies that we search a solutions corresponding to integer values in $\{S, S+M, .., S+(n-1)M\}$ which are the $n-1$ different targets we have to test. (We have done the same in section 1.1 when we searched for a collision in $\mathbb{Z}_M$ and changed the target to R and $R+M$.)

Assumptions. We turn our attention especially to the case of the average subset-sum problem as it is of interest for designing and cryptanalysis of cryptographic systems. We assume that no information is leaking about the hidden structure (e.g., a super increasing sequence) of the underlying knapsack used as trapdoor in a cryptographic system and thus study generic algorithms for the hard knapsack problem, i.e., density equal to one. For cryptographic purposes, we assume that a solution always exists. For security reasons the number of knapsack elements is large which is why we analyse the cost in the asymptotic case. The weight of the solution vector is expected to be $n/2$. It is also at most $n/2$ as we can always consider the inverse problem where we search for $\bar{\mathbf{x}} := \mathbf{1} - \mathbf{x}$ such that

$$\mathbf{a} \cdot \bar{\mathbf{x}} = \sum_{i=1}^{n} a_i - S \ .$$

Relation to the k-sum problem. The subset-sum problem is related to the k-sum problem, we presented in section 1.3. We can split the sets of n knapsack elements into k disjoint sets and create k lists of size $2^{n/k}$ containing all possible subsums. An algorithm that solves the k-sum problem can hence be used to find the solution to the subset-sum problem. Section 3 presents algorithms based on this idea.

Otherwise, consider we are given k lists of positive integers for which we desire to solve the k-sum problem with target S. For simplicity of presentation, let us assume that all lists are of same length m. We can write all elements in a row of length km and thus create a subset-sum problem. An additional constraint is that within every successive block of m elements, only one is chosen.

Take for example the classical birthday problem, $k = 2$. Let a_i be the elements of the first list and b_i be the elements of the second list. Compute the integer $K = \sum_i (a_i + b_i) + 1$. We create a knapsack of elements $a_i + K$ and $b_i + (m+1)K$ with target $S + K + (m+1)K$. The target is strictly smaller than $2K + (m+1)K = (m+3)K$ and larger than $(m+1)K$. Note that summing up all elements $a_i + K$ leads to a value that is strictly smaller than $(m+1)K$. We thus need to include one element $b_j + (m+1)K$ to the solution. If we include two such elements, we obtain a value of at least size $2(m+1)K$ which exceeds the target. As the target is larger than $(m+1)K$, it contains an additional element of the form $a_i + K$. If we would include a third element to the solution of the form $a_i + K$, we would obtain a sum that is at least $(m+1)K + 2K = (m+3)K$ which also exceeds the target.

The method can be extended to the general case of k lists. In this way we can solve the k-sum problem by applying an algorithm that solves the subset-sum problem. The reduction is not tight. The dimension of the knapsack is mk and a classical algorithm runs in time $\tilde{\mathcal{O}}\left(2^{\frac{mk}{2}}\right)$ (section 3). Already an exhaustive search on the k-list problem runs in time m^k which is much more efficient.

2.2. Proof of NP-completeness: Reduction to the vertex cover problem

Let $G = G(V, E)$ be a graph with a set of n vertices V and m edges E. A vertex cover for G is a set $C \subset V$ of vertices such that every edge of the graph has at least one incident vertex in C. We define the vertex cover problem as:

Problem 2.1 (Vertex cover problem (VC))
Given a graph $G(V, E)$ and an integer k, is there a vertex cover of at most k vertices?

To find a minimal vertex cover (VC) is a classical optimization problem that is NP-hard. The decisional version is NP-complete [Kar72]. We can reduce the decisional subset-sum problem (SS) to VC and conclude that the subset-sum problem is at least as hard as VC and thus NP-complete. Given an oracle that solves SS one could solve VC.

Reduction: Let $G(V, E)$ be a graph with a set of n vertices V and m edges E and let k be a parameter. We aim to define a subset-sum problem depending on the graph and show that there exists a k-vertex cover (a vertex cover of at most k vertices) if and only if there exists a solution to the subset-sum problem. We define a set of $m + n$ integers and a target S for a subset-sum problem in the following way: For every vertex v and every edge (u, v), we define integers a_v and $b_{u,v}$, respectively, represented as coefficient vectors of length $m + 1$ in base four: For the j-th edge (u, v), $j = 0, .., m - 1$, we set the j-th digit of a_u and a_v to one while all others are set to zero except for the m-th position of a_u, a_v which is one. The i-t vertex is associated to the integer

$$a_{v_i} = \sum_{j=0}^{m-1} t_{i,j} \cdot 4^j + 1 \cdot 4^m$$

where $t_{i,j} = 1$ if e_j is incident to a_{v_i} and $t_{i,j} = 0$ else. The digits of $b_{(u,v)} = b_{e_j}$ are zero except for the j-th position that equals 1. We note that a sum of such integers can only lead to coefficients in $\{0, 1, 2, 3\}$ at the positions corresponding to 4^i for $i = 0, .., m - 1$. This means that no carry bits occur. We can know define the target S of the subset-sum problem:

$$S = k \cdot 4^m + \sum_{j=0}^{m-1} 2 \cdot 4^j \quad .$$

Table 2.1 visualizes the coefficients of the elements of the set and the target. Consider a sum of some integers a_v and $b_{u,v}$, i.e., a subsum of lines in table 2.1. We want to find a subset of integers a_v and $b_{u,v}$ that satisfies the target. Note that the first term of S can only be obtained by summing up exactly k integers a_v as they have a one at the m-th position. The second term is true in two cases: Either for every edge e_j, that we add, exactly one incident vertex is present in the sum or an edge is not included while both its endpoints are. The coefficient two is thus equivalent to the fact that the set of vertices is a vertex cover.
Suppose that G has a vertex cover C' of at most k vertices. It then also has a vertex cover C of exactly k vertices as adding vertices preserves the property of being a vertex cover. We

	$\times 4^0$	$\times 4^1$	$\dots$	$\times 4^m$
a_{v_0}	$t_{0,0}$	$\dots$	$t_{0,m-1}$	1
$\vdots$	$\vdots$	$\dots$	$\vdots$	$\vdots$
$a_{v_{n-1}}$	$t_{n-1,0}$	$\dots$	$t_{n-1,m-1}$	1
b_{e_0}	1			0
		$\ddots$		$\vdots$
$b_{e_{m-1}}$			1	0
S	2	$\dots$	2	k

Table 2.1.: *Subset-sum problem* $(a_v, b_{(u,v)}, S)$ *in base four.*

compute the sum of integers a_v for all $v \in C$ and $b_{u,v}$ for which exactly one out of u, v is in C. By the previous observation, we obtain the target sum S.

Suppose now that there exist subsets $M \subseteq V$ and $N \subseteq E$ such that

$$\sum_{v \in M} a_v + \sum_{(u,v) \in N} = S \ .$$

Since $a_v \geq 4^m$ and $S < (k+1)4^m$, we know that $|M| \leq k$. As no carry bits occur and every of the first m positions of S is two, we can conclude that for every edge at least one incident vertex is in M. The set M is hence a vertex cover of G of at most k vertices. The parameter k is not fixed and the above reasoning is hence true also for a minimal vertex cover.

2.3. The knapsack problem in cryptography

A public key encryption scheme based on the knapsack problem (2.2) could be as follows. We provide a public sequence $\mathbf{a} = (a_1, .., a_n)$ of weights and encrypt a message $\mathbf{x} \in \{0,1\}^n$ as $S = a \cdot x$. For a random hard knapsack, the receiver of the encrypted message and an attacker are faced with the same problem. To allow for a more efficient decryption on the receiver's end, one needs a trapdoor. The knapsack problem is easy to solve if the sequence $\mathbf{a}$ is super-increasing, which means that each element is larger than the sum of the previous elements:

$$a_j > \sum_{i=1}^{j-1} a_i \text{ for } j = 2, .., n \ .$$

Starting from the last coordinate x_n, we can then deduce one after another that $x_j = 1$ if and only if $S > \sum_{i=1}^{j-1} a_i$. The sequence $\mathbf{a}$ is kept secret and is used for an easy decryption, while a scrambled sequence $\mathbf{b}$, derived from $\mathbf{a}$, is made public. The security relies on the quality of the disguise. The attacker has to solve a random knapsack problem if the structure of the knapsack is well hidden. Merkle and Hellman proposed the first public key encryption scheme based on the knapsack problem over the integers [MH78] following this approach. The structure is hidden by a strong modular multiplication but attacks [Sha82] showed that this disguise is insufficient. In the following many systems were proposed and broken. A good survey of proposed systems and successful attacks is provided in [BO88, Des88, Odl90, Lai01].

In 1997, Ajtai and Dwork [AD97] constructed a provably-secure cryptosystem based on the the subset-sum problem. The scheme is as hard as solving the worst-case unique shortest vector problem. The difficulty to break the scheme is then related to the reduction from average subset sum to the unique vector problem [LO85, FG84]. Other lattice based cryptosystems [Reg04, Reg05, Pei09] are also as hard to attack as the average-case subset-sum problem. While the reduction is not tight, the scheme by Lyubashevsky, Palacio and Segev [LPS10] is polynomial-time equivalent to the average subset-sum problem.

Beside public-key schemes other cryptographic primitives can be constructed. Impagliazzo and Naor proposed [IN89] efficient pseudo-random generators, universal one-way function and bit-comitment schemes that can be proven secure based on random instances.

2.4. Solving the subset-sum problem in short

Impagliazzo and Naor showed [IN96, IN89] that the subset-sum problem for average-case instances is hardest for a density close to one. We present algorithms for average-case instances of density one in chapter 3, 4 and 5. The average running time of a generic algorithm for random instances using no structural information on the group is exponential.

The knapsack problem is of low density if $d < 1$. For a density smaller than 0.94, the knapsack problem can be solved by lattice reduction [LO85, CJL+92] with exception of an exponentially small fraction of random knapsacks. The technique reduces the SS to a shortest vector problem in an integer lattice that can be solved given a lattice oracle. For small dimensions the oracle is replaced in practice by a reduction algorithm (e.g., LLL-algorithm [LLL82] or BKZ [Sch87]) that solves the problem in polynomial-time.

For knapsacks of high density, i.e., $d > 1$, polynomial-time algorithms exist as well. The SS can be solved for $a_i = \mathcal{O}(n)$ by a dynamic programming algorithm [TS86] or an algorithm based on elementary number theory [GM91] in time at most $\mathcal{O}(n^2)$. The technique based on a reduction to lattice problems can also be applied to high density knapsacks [JG94] leading to an exponential algorithm of time $\mathcal{O}(2^{n/1000})$. On top of these generic attacks, a cryptosystem based on the subset-sum problem may suffer from a badly disguised trapdoor that allows an efficient attacker (see previous section 2.3).

The ideal public key cryptosystem based on the knapsack problem would therefore have high density (close to one) and a good mechanism to hide the structure needed for an efficient decryption.

Chapter 3

Classical birthday paradox algorithms

A brute force attack on the subset-sum problem enumerates all coefficient vectors $x \in \{0,1\}^n$ and evaluates 2^n sums; it takes time 2^n storing only one element at a time. It is however more efficient to split the set of weights a_i into two of approximate same size [HS74]. We then store all $2^{\lfloor \frac{n}{2} \rfloor}$ sums $\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} a_i\, x_i$ in a list $\mathcal{L}_1$ and search for collisions with the sums $S - \sum_{\lfloor \frac{n}{2} \rfloor+1}^{n} a_i\, x_i$ in a second list $\mathcal{L}_2$. A collision corresponds to the equation

$$\sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} a_i\, x_i = S - \sum_{\lfloor \frac{n}{2} \rfloor+1}^{n} a_i\, x_i$$

which clearly gives a solution $\mathbf{x}$ to the original problem (2.1). The lists contain each about $2^{\frac{n}{2}}$ elements. Creation of the lists and the collisions search between sorted lists has a running time of order $\mathcal{O}\left(n\, 2^{\frac{n}{2}}\right)$ as elaborated in Section 1.1.

3.1. Shamir-Schroeppel algorithm

The memory requirement of the above presented algorithm can be reduced as proposed by Schroeppel and Shamir [SS81]. We assume that the solution has weight $\frac{n}{2}$ and that $4|n$. We split the set of weights into four parts and compute $\mathcal{L}_1, \mathcal{L}_2$ on the fly as sums of twice $\frac{n}{4}$ knapsack elements. The elements of the fictive lists $\mathcal{L}_1$ are sums $\sigma_1 + \sigma_2$ where

$$\sigma_1 \in \mathcal{Y}_1 := \left\{ \sum_{i=1}^{\lfloor \frac{n}{4} \rfloor} a_i\, x_i \,|\, x_i \in \{0,1\} \right\} \text{ and } \sigma_2 \in \mathcal{Y}_2 := \left\{ \sum_{i=\lfloor \frac{n}{4} \rfloor+1}^{\lfloor \frac{n}{2} \rfloor} a_i\, x_i \,|\, x_i \in \{0,1\} \right\}.$$

The elements of the fictive list $\mathcal{L}_2$, sums $\sigma_3 + \sigma_4$, are built in the same way using the other half of the knapsack elements. The lists $\mathcal{Y}_i$, for $i = 1,2,3,4$), are called base lists and their size is denoted by Y_i which is equal to $2^{\frac{n}{4}}$. We found a solution if a collision occurs, that is, if

$$\sigma_1 + \sigma_2 = S - (\sigma_3 + \sigma_4)\ . \tag{3.1}$$

The elements of $\mathcal{L}_1$ are created in increasing order while we need $\mathcal{L}_2$ to be produced in decreasing order. To compute $\mathcal{L}_1$, we create the lists $\mathcal{Y}_1, \mathcal{Y}_2$ and sort $\mathcal{Y}_2$ in increasing order.

We then compute the sums of the current (at the beginning the smallest) element in $\mathcal{Y}_2$ with all of $\mathcal{Y}_1$ and store these values in a binary search tree. Note that the smallest element of $\mathcal{L}_1$ is necessarily a sum of the smallest element of $\mathcal{Y}_2$ and some element in $\mathcal{Y}_1$; it is equal to the smallest element in the tree. Also, if we update the values of the tree by replacing $\mathcal{Y}_2[i]$ (i initially by 0) by a following element of the ordered list $\mathcal{Y}_2$, we augment the value. The algorithm repeatedly outputs the smallest element of the tree, thus creating L_1 on the fly, and updates the node. The new value is the sum of to the same element of $\mathcal{Y}_1$ and the next element of $\mathcal{Y}_2$ (until the end of the list is reached).
The elements of $\mathcal{L}_2$ are created by sorting $\mathcal{Y}_4$ in decreasing order and outputting the largest elements of the tree. In order to find the solution to the original problem (2.1), we produce the first element $\sigma_1 + \sigma_2$ of $\mathcal{L}_1$ and $\sigma_3 + \sigma_4$ of $\mathcal{L}_2$ and compare the value. As long as $\sigma_1 + \sigma_2 < S - (\sigma_3 + \sigma_4)$, we create a new element of $\mathcal{L}_1$; otherwise we compute the next element of $\mathcal{L}_2$. The size of the self-balancing trees is $Y_1 \approx Y_4 = 2^{\frac{n}{4}}$ and determines the memory complexity of size $\mathcal{O}\left(n\, 2^{\frac{n}{4}}\right)$. The cost for search and update are of order $\log Y_1$. The overall time complexity lies unchanged by $\mathcal{O}\left(n\, 2^{\frac{n}{2}}\right)$ as we still need to compare $2^{\frac{n}{2}}$ elements. If n is not divisible by 4, the complexity changes in its constant terms that are neglected in the $\mathcal{O}$-notation.

3.1.1. Unbalanced case

If the knapsack is unbalanced and has weight $\ell = \alpha n$, for real α in the interval $(0, 0.5)$, we modify the base lists to sums of $\frac{l}{4}$ knapsack elements out of $\frac{n}{4}$ elements. The lists $\mathcal{Y}_i$ contain each $Y_i = \binom{\frac{n}{4}}{\frac{\ell}{4}} = \mathcal{O}\left(2^{\frac{n}{4}\, h(\alpha)}\right)$ elements. The memory requirement is $\max_i(Y_i \log Y_i, N_{Sol})$. To find the solution, the algorithm joins the lists Y_i and creates twice $Y_1^2 = \binom{n/4}{\ell/4}^2$ elements for the fictive lists $\mathcal{L}_1, \mathcal{L}_2$.

So far, the algorithm finds the solution if exactly $\ell/4$ knapsack elements are indexed in each of the lists $\mathcal{Y}_1, .., \mathcal{Y}_4$. For example, the solution will be missed if is has weight $\ell/4 + 1$ within the first $\frac{n}{4}$ knapsack elements. To guarantee a solution, we have to permute the knapsack elements. The choice of the sets can be done randomly or follow a systematic approach as described in the following.

Deterministic variant. We line up the knapsack elements and start with a window containing the first half of the elements for the list $\mathcal{L}_1$. The second half is associated to $\mathcal{L}_2$. If the solution vector indexes more than $\frac{n}{4}$ ones in the first half, it indexes less than $\frac{n}{4}$ in the second half. Moving the window to the right by one changes the partial weight until the window of $\mathcal{L}_1$ marks the last half. As we start with a window containing more than $\frac{n}{4}$ ones and end with one having less than $\frac{n}{4}$, we know that necessarily one intermediate window contained exactly $\frac{n}{4}$ active knapsack elements. Applying the algorithm on each window, we have found the right subsets for $\mathcal{L}_1, \mathcal{L}_2$ after at most $\frac{n}{2}$ steps,. Assume that the solution is well distributed for one window. We then have to find a right choice of $\frac{n}{4}$ elements for $\mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Y}_3$ and $\mathcal{Y}_4$. We proceed in the same way: Within each half, we choose inner windows of length $\frac{n}{4}$ starting with the first quarter of elements for $\mathcal{Y}_1$ and the last quarter for $\mathcal{Y}_4$. If no solution is found, we slide the inner windows by one to the right. After at most $(\frac{n}{4})^2$ steps, we have found the right subsets $\mathcal{Y}_1, .., \mathcal{Y}_4$ [Sti02]. In total, we have to try $\mathcal{O}\left(n^3\right)$ different sets. The deterministic algorithm is of time complexity $\mathcal{O}\left(n^4 2^{\frac{n}{2}h(\alpha)}\right)$.

Probabilistic variant. A randomized choice of the sets of knapsack elements for $\mathcal{L}_1$ as proposed by Coppersmith [Sti02] permits to reduce the number of iterations. We assume that the choice of $\frac{n}{2}$ elements is done independently at random for each iteration. The probability to pick a good set that contains $\frac{n}{4}$ knapsack elements that are indexed by the solution vector is

$$\binom{\frac{n}{2}}{\frac{n}{4}}^2 \Big/ \binom{n}{\frac{n}{2}} \approx 2\sqrt{\frac{2}{\pi n}} = \mathcal{O}\left(1/\sqrt{n}\right)$$

which reduces the iterations to an expected number of about $\sqrt{n}$ on average for large n. Using this randomized approach for the algorithm of Schroeppel-Shamir leaves us with $n^{3/2}$ random choices on average for the subsets $\mathcal{Y}_1, .., \mathcal{Y}_4$. The probabilistic algorithm is of time complexity $\mathcal{O}\left(n^{5/2}\, 2^{\frac{n}{2}h(\alpha)}\right)$.

3.2. Heuristic Schroeppel-Shamir

The approach of Shamir and Schroeppel to reduce the memory requirement works if an ordering on $\mathbb{Z}$ exists which is compatible to the group operation. This might not always be the case. Depending on the group, we can impose a constraint on the elements in the intermediate lists of a birthday-paradox algorithm. This is a common technique used to solve the k-sum problem as presented in Section 1.3. We have to take care that the property that we impose is shared by at least one solution in the case that many exist. If there exists only one solution, we change the condition until we find the solution. In the binary case, we impose a constraint on t higher bits while for an integer knapsack, we can work with a modular condition. The two ways are very similar as integers can be represented as binary vectors and a modular condition can be realized as a bit condition. The idea leads to a heuristic version [HGJ10] of the algorithm by Schroeppel-Shamir to solve integer knapsacks. It avoids the use of priority queues and has the same heuristic complexity as the original algorithm. For some degenerated cases however the running time might be increased.

We assume that the solution has weight $\frac{n}{2}$ and that $4\,|\,n$. A solution is represented by subsums $\sigma_1, .., \sigma_4 \in \mathcal{Y}_1 \times .. \times \mathcal{Y}_4$ that fulfil equation (3.1):

$$\sigma_1 + \sigma_2 = S - (\sigma_3 + \sigma_4)\ .$$

So they also satisfy the congruence

$$\sigma_1 + \sigma_2 \equiv S - (\sigma_3 + \sigma_4) \equiv R \mod M$$

for some modulus M and a random integer in $\mathbb{Z}_M$. As we do not know the value R, we will loop over all possible M values.

We choose a random integer R and four non-overlapping subsets of knapsack elements, each containing one quarter of the n elements, and create lists $\mathcal{Y}_1, .., \mathcal{Y}_4$ of all possible subsums σ_i. The lists are of size $2^{\frac{n}{4}}$. The next step joins the list $\mathcal{Y}_1$ with $\mathcal{Y}_2$ to obtain the list $\mathcal{L}_1$ of elements $\sigma_1 + \sigma_2$ where $\sigma_1 + \sigma_2 \equiv R \mod M$: We sort $\mathcal{Y}_1$ according to the value mod M and search for each $\sigma_2 \in \mathcal{Y}_2$ for the element $R - \sigma_2$ in $\mathcal{Y}_1$. The lists $\mathcal{Y}_3$ and $\mathcal{Y}_4$ are joined likewise to create elements $\sigma_3 + \sigma_4$ in a list $\mathcal{L}_2$. For a random knapsack and M smaller or equal to the size of the largest knapsack element, we can assume that the values are independently and equally distributed values in $\mathbb{Z}_M$ (see further details in section 4.1). The size of the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ is then about $2^{\frac{n}{2}}/M$. To minimize the memory requirement and the time, we choose M of size close to $2^{\frac{n}{4}}$ (a bit smaller). The last step searches a collision between $\mathcal{L}_1$ and $\mathcal{L}_2$ over the integers. We can realize this in an efficient way by sorting $\mathcal{L}_1$ and $\mathcal{L}_2$ in increasing order. We then start with the first element of $\mathcal{L}_1$ and the last one of $\mathcal{L}_2$. Three cases are possible: If $\sigma_1 + \sigma_2 \in \mathcal{L}_1$ is smaller than $S - (\sigma_3 + \sigma_4)$, we move the pointer of $\mathcal{L}_1$ one element up. If $\sigma_1 + \sigma_2 \in \mathcal{L}_1 > S - (\sigma_3 + \sigma_4)$, the pointer of the second list is updated. We have found a collision if equality holds.

Complexity. The algorithm needs to sort and store base lists of $2^{\frac{n}{4}}$ elements. The first join then finds an expected number of $2^{\frac{n}{4}}$ colliding elements that it stores in lists. The memory requirement is thus of order $\mathcal{O}\left(\frac{n}{4}\, 2^{\frac{n}{4}}\right)$. The last step sorts twice about $2^{\frac{n}{4}}$ elements and searches a collision in at most $2 \cdot 2^{\frac{n}{4}}$ steps. We need to repeat the previously described steps choosing each time a new R which leads to a running time of $\mathcal{O}\left(n\, 2^{\frac{n}{2}}\right)$.

Unbalanced case. For a solution vector of weight $\ell = \alpha n$, $\alpha \in (0, 0.5)$, we change the base lists to contain all subsums of $\ell/4$ knapsack elements. For a right choice of four sets of knapsack elements for the base lists, we then find the solution. However, the correct split is not known in advance and we can apply the window method or the probabilistic way to change the base lists as explained in section 3.1.1.

Degenerated case. For some knapsacks, the running time can considerably differ from the expected one. This is the case when the number of modulo sums is not well distributed over all possible values in $\mathbb{Z}_M$. Some target R will then lead to many solutions and the size of the lists $\mathcal{L}_1, \mathcal{L}_2$ will exceed the expected size $2^{\frac{n}{4}}$. For an average knapsack, we can assume that the values are well distributed as shown in section 4.1.

Many solutions. For a large number N_{Sol} of solutions, we can stop the algorithm as soon as we have found enough solutions. The *maximal* running time to search only one solution is then divided by N_{Sol}. It is lower bounded by $2^{\frac{n}{4}}$, the size of the lists.

3.3. A time-memory tradeoff on Schroeppel-Shamir

The original Schroeppel-Shamir algorithm works in time $\tilde{\mathcal{O}}\left(2^{n/2}\right)$ and memory $\tilde{\mathcal{O}}\left(2^{n/4}\right)$. In this section we describe a continuous time-memory tradeoff down to $\tilde{\mathcal{O}}\left(2^{n/16}\right)$ memory. That is we describe a variant of Schroeppel-Shamir that runs in time $\tilde{\mathcal{O}}\left(2^{(11/16-\varepsilon)n}\right)$ requiring space $\tilde{\mathcal{O}}\left(2^{(1/16+\varepsilon)n}\right)$ for any $0 \leq \epsilon \leq 3/16$. For simplicity we first describe the algorithm with exactly $\tilde{\mathcal{O}}\left(2^{n/16}\right)$ memory.

We write the knapsack as $\sigma_1 + \sigma_2 = S - (\sigma_3 + \sigma_4)$ as in (3.1) where each σ_i is a knapsack of $n/4$ elements:

$$\sigma_1 = \sum_{i=1}^{n/4} x_i a_i, \quad \sigma_2 = \sum_{i=n/4+1}^{n/2} x_i a_i, \quad \sigma_3 = \sum_{i=n/2+1}^{3n/4} x_i a_i, \quad \sigma_4 = \sum_{i=3n/4+1}^{n} x_i a_i \ .$$

We guess three values R_1, R_2 and R_3 of $3n/16$-bit each and we set R_4 such that $R_1 + R_2 + R_3 + R_4 = S \mod 2^{3n/16}$. We consider the four subknapsack equations

$$\sigma_i = R_i \mod 2^{3n/16} \ . \tag{3.2}$$

We solve the four knapsack problems, w.r.t. (3.2), independently by using the original Schroeppel-Shamir algorithm. Therefore in time $\tilde{\mathcal{O}}\left(2^{n/8}\right)$ and memory $\tilde{\mathcal{O}}\left(2^{n/16}\right)$ we obtain four lists $\{\sigma_1\}$, $\{\sigma_2\}$, $\{\sigma_3\}$ and $\{\sigma_4\}$ of solutions satisfying the four equations (3.2). To recover the knapsack solution we merge these four lists using the same merging procedure as in the original Schroeppel-Shamir algorithm; since each list has size $\tilde{\mathcal{O}}\left(2^{n/16}\right)$, the merging procedure runs in time $\tilde{\mathcal{O}}\left(2^{n/8}\right)$ and uses $\tilde{\mathcal{O}}\left(2^{n/16}\right)$ memory. Since the partial target sums, R_i of $3n/16$-bit each, are not known, we have we have to repeat $2^{3n/16}$ times. The total running time becomes

$$\tilde{\mathcal{O}}\left(2^{3n/16}\right)^3 \cdot \left(\tilde{\mathcal{O}}\left(2^{n/8}\right) + \tilde{\mathcal{O}}\left(2^{n/8}\right)\right) = \tilde{\mathcal{O}}\left(2^{11n/16}\right)$$

and the memory consumption is $\tilde{\mathcal{O}}\left(2^{n/16}\right)$.

It is easy to generalize the idea if we can devote more memory, say $\tilde{\mathcal{O}}\left(2^{(1/16+\varepsilon)n}\right)$ for any $0 \leq \epsilon < 3/16$. We choose the R_is of size $(3/16 - \varepsilon)n$-bit each. We can still build the four lists $\{\sigma_i\}$ in time $\tilde{\mathcal{O}}\left(2^{n/8}\right)$ using Schroeppel-Shamir, but this time the size of the lists is $\tilde{\mathcal{O}}\left(2^{(1/16+\varepsilon)n}\right)$. The merging-join runs in time $\tilde{\mathcal{O}}\left(2^{(1/8+2\varepsilon)n}\right)$, still with memory $\tilde{\mathcal{O}}\left(2^{(1/16+\varepsilon)n}\right)$. We obtain a total running time of

$$\tilde{\mathcal{O}}\left(2^{(3/16-\varepsilon)n}\right)^3 \cdot \left(\tilde{\mathcal{O}}\left(2^{n/8}\right) + \tilde{\mathcal{O}}\left(2^{(1/8+2\varepsilon)n}\right)\right) = \tilde{\mathcal{O}}\left(2^{(11/16-\varepsilon)n}\right)$$

and a memory consumption $\tilde{\mathcal{O}}\left(2^{(1/16+\varepsilon)n}\right)$.

Chapter 4

Representation-technique algorithm

In 2010, Howgrave-Graham and Joux proposed a new generic algorithm [HGJ10] to solve the knapsack problem (2.2) for random instances of density 1. A density close to 1 implies that the problem has a unique solution (or few solutions). The new idea is to allow for an overlap between the vectors that we later combine to a solution, i.e., all vectors are of same length as the solution and no longer of half its length as in the previously proposed algorithms.

We define a representation of the solution of weight ℓ as a tuple $(\mathbf{y}, \mathbf{z}) \in \{0,1\}^n \times \{0,1\}^n$ such that $\mathbf{x} = \mathbf{y}+\mathbf{z}$ where $\mathbf{y}$ and $\mathbf{z}$ are almost of same weight $\sim \ell/2$. Figure 4.1 shows one such representation. The vectors $\mathbf{y}, \mathbf{z}$ are each a partial solution. The number of representations depends on the number of possibilities to choose $\lfloor \ell/2 \rfloor$ out of ℓ ones in $\mathbf{x}$:

$$N_{HGJ} = \begin{cases} 2 \cdot \binom{\ell}{(\ell-1)/2} & \text{for odd } \ell \\ \binom{\ell}{\frac{\ell}{2}} & \text{for even } \ell \end{cases} \tag{4.1}$$

which is of order $\mathcal{O}\left(2^\ell\right)$ for large n and ℓ.

For simplicity, we will assume that $2|\ell$ from now on. The new idea is to search for one representations of the solution for which the following modular constraint holds:

$$\begin{aligned} \mathbf{a} \cdot \mathbf{y} &\equiv R &&\mod M \quad \text{and} \\ \mathbf{a} \cdot \mathbf{z} &\equiv S - R &&\mod M \end{aligned} \tag{4.2}$$

for a large integer M and a random element R of $\mathbb{Z}_M$.

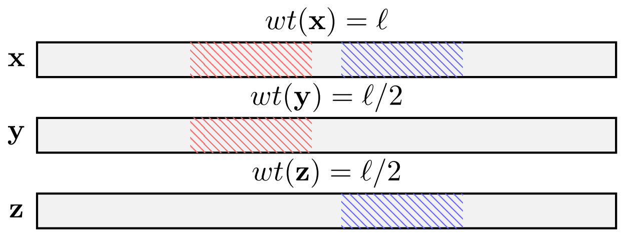


Figure 4.1.: *One representation* $(\mathbf{y}, \mathbf{z})$ *of a solution* $\mathbf{x}$. *The shaded area indicates non-zero bit positions.*

Suppose we are given two lists $\mathcal{L}_1, \mathcal{L}_2$ of sums $\mathbf{a} \cdot \mathbf{y} = \sum_{i=1}^n a_i y_i$ and $\mathbf{a} \cdot \mathbf{z} = \sum_{i=1}^n a_i z_i$ that satisfy (4.2) where $\mathbf{y}, \mathbf{z}$ are drawn independently and uniformly at random from $\{0,1\}^n$ and have weight $\ell/2$. It follows that

$$\mathbf{a} \cdot \mathbf{y} + \mathbf{a} \cdot \mathbf{z} \equiv S \mod M$$

for all pairs of elements in $\mathcal{L}_1, \mathcal{L}_2$. If we find a pair for which the equation holds over the integers and where $\mathbf{y} + \mathbf{z}$ is of weight ℓ, we have found a representation and the solution. We see that a join between the two given lists may lead to a solution. In order to develop an algorithm along these lines, it is of importance to study the probability that partial knapsack sums as $\mathbf{a} \cdot \mathbf{y}$ take values in $\mathbb{Z}_M$ which we will do in section 4.1.

The ideas behind. We have transformed the original problem in two ways: First, we no longer search directly for a unique solution but we perform a search for one-out-of-many representations. We have added degrees of freedom and increased our search space exponentially. In return, we need to decrease the search space to obtain an efficient algorithm. Second, the elements in $\mathcal{L}_1, \mathcal{L}_2$ are solutions of weight $\ell/2$ of two modulo knapsacks given in (4.2). They can be found by a classical Shamir-Schroeppel algorithm or by applying the representation technique again to obtain an efficient (but still exponential) algorithm. The problem is transformed into several smaller problems that can be solved by classical means.

How to determine the number of levels of the algorithm. The number of times we apply the representation technique is motivated by the goal to obtain an algorithm of minimal running time. We assume that $4|\ell$ and $\ell = \frac{n}{2}$ and choose M as an integer of size $\mathcal{O}\left(2^{\ell}\right)$. In order to create the lists $\mathcal{L}_1, \mathcal{L}_2$ that correspond to vectors of length n and weight $\ell/2$ satisfying (4.2), we may follow the classical approach: We split the knapsack in two and enumerate vectors of half length and half weight which means that we create bottom lists of $B = \binom{n/2}{\ell/4} = \mathcal{O}\left(2^{0.406n}\right)$ elements. We then apply a merge-join algorithm to obtain an expected number of $\binom{n}{\ell/2}/M \approx \mathcal{O}\left(2^{0.311n}\right)$ elements for the lists $\mathcal{L}_1, \mathcal{L}_2$. The cost is then $\tilde{\mathcal{O}}\left(2^{0.406n}\right)$ in time and memory given by the size of the bottom lists. However, if we apply the representation technique again, we can diminish these cost as we see in the next section.

Remark on the complexity notation. We aim to study and improve time and memory requirements in the asymptotic case and hence apply two simplifications when analysing the complexity. While the actual complexity may be given as a sum of partial cost we can compute the overall cost by the maximal term. In the paragraph above, for example, the overall memory requirement is given by $|\mathcal{L}_1| + |\mathcal{L}_2| + 2 \cdot B = \mathcal{O}(\max(|\mathcal{L}_1|, |\mathcal{L}_2|, B))$. We do also neglect polynomial and logarithmic factors by use of the soft-$\mathcal{O}$ notation, denoted by $\tilde{\mathcal{O}}(.)$. It conceals constants as well as logarithmic and polynomial factors in n. Logarithmic and polynomial factors appear in our algorithms due to sorting, addition and storage of the elements as well when we need to repeat to create the bottom lists. The constants change slightly if the assumptions such as $4 \mid n$ or $8 \mid n$ are not satisfied.

The presented complexity analysis provides thereby a comparison in the asymptotic case and a theoretical recommendation for very large n. In practice, the polynomial and logarithmic factors due to sorting, the number of lists, the storage of elements etc. have to be taken into account and influence the memory cost and time consumption considerably.

4.1. Distribution of modular sums

Given a large integer M and a set of n integers $\mathbf{a} = (a_1, .., a_n) \in \mathbb{Z}_M^n$ where a_i are chosen uniformly at random. We denote by $\mathcal{B} \subset \mathbb{Z}_M^n$ the set of all vectors $\mathbf{y}$ and study the distribution of the values $\mathbf{a} \cdot \mathbf{y} = \sum_{i=1}^n a_i y_i \mod M$. The present section shows that the values $\mathbf{a} \cdot \mathbf{y} \mod M$ behave well for almost all $\mathbf{a}$ in the sense that: For fixed $R \in \mathbb{Z}_M$, there are $\mathcal{B}/M$ sums $\sum_{i=1}^n a_i y_i \pmod M$ on average that are congruent to $R \mod M$. The proportion of knapsacks $\mathbf{a} = (a_1, .., a_n) \in \mathbb{Z}_M^n$ for which this is not the case is exponentially small. We derive these results from the following observations.

Let $N_{\mathbf{a}}(\mathcal{B}, R)$ denote the number of solutions $\mathbf{y} \in \mathcal{B}$ of $\mathbf{a} \cdot \mathbf{y} \equiv R \mod M$,

$$N_{\mathbf{a}}(\mathcal{B}, R) = \left| \left\{ \mathbf{y} \in \mathcal{B} \text{ such that } \sum_{i=1}^n a_i y_i \equiv R \mod M \right\} \right| \ .$$

Let $P_{\mathbf{a}}(\mathcal{B}, R)$ denote the probability that a knapsack of elements $\mathbf{a}$ results in the value c modulo M for a uniformly at random chosen partial solution $\mathbf{y}$ from $\mathcal{B}$,

$$P_{\mathbf{a}}(\mathcal{B}, R) = \frac{N_{\mathbf{a}}(\mathcal{B}, R)}{|\mathcal{B}|} \ .$$

Our main tool to theoretically study the distribution of the scalar products is the following theorem [NSS01]:

Theorem 4.1
For any set $\mathcal{B} \subset \mathbb{Z}_M^n$, the equality:

$$\frac{1}{M^n} \sum_{\mathbf{a} \in \mathbb{Z}_M^n} \sum_{R \in \mathbb{Z}_M} \left(P_{a_1, \cdots, a_n}(\mathcal{B}, R) - \frac{1}{M} \right)^2 = \frac{M-1}{M|\mathcal{B}|} \tag{4.3}$$

holds.

The theorem shows that for almost all random knapsack $\mathbf{a} \in \mathbb{Z}_M^n$, the number of subsolutions $\mathbf{y}$ that lead to a value R, $N_{\mathbf{a}}(\mathcal{B}, R)$, takes its expected value $\frac{|\mathcal{B}|}{M}$. Choosing M close to $\mathcal{B}$ (but a bit smaller), we can thus expect to find one solution on average for almost all knapsacks. We will now study the fraction of knapsacks for which this is not the case.

Number of bad knapsacks. We fix an integer $\lambda > 0$ and define a bad knapsack as a random modular knapsack that attains less than M/λ values in $\mathbb{Z}_M$. We are interested in the number, $F(\lambda)$, of such bad random modular knapsacks within all M^n possible choices for $\mathbf{a}$. If we count only the bad knapsacks on the left side of (4.3), we derive that

$$F(\lambda) \sum_{R \in \mathbb{Z}_M} \left(P_{a_1,..,a_n}(\mathcal{B}, R) - \frac{1}{M} \right)^2 \leq \frac{M-1}{M|\mathcal{B}|} M^n \quad . \tag{4.4}$$

We can lower bound the sum using the following observations. Denote by N_0 the number of values R which are never attained for a bad knapsack and define $P_R = P_{a_1,..,a_n}(\mathcal{B}, R)$. We split the sum into M/λ values for which $P_R \neq 0$ and $N_0 := \frac{\lambda-1}{\lambda} M$ summands for which $P_R = 0$ which leads to:

$$\begin{aligned}
\sum_{R \in \mathbb{Z}_M} \left(P_R - \frac{1}{M} \right)^2 &= \sum_{R \in \mathbb{Z}_M, P_R \neq 0} \left(P_R - \frac{1}{M} \right)^2 + \frac{N_0}{M^2} \\
&= \sum_{R \in \mathbb{Z}_M, P_R \neq 0} \left(P_R^2 - \frac{2P_R}{M} \right) + \frac{M - N_0}{M^2} + \frac{N_0}{M^2} \\
&= \sum_{R \in \mathbb{Z}_M, P_R \neq 0} P_R^2 - \frac{2}{M} \underbrace{\sum_{R \in \mathbb{Z}_M P_R \neq 0} P_R}_{1} + \frac{1}{M} \\
&= \sum_{R \in \mathbb{Z}_M, P_R \neq 0} P_R^2 - \frac{1}{M} \quad .
\end{aligned}$$

As $\sum_{R \in \mathbb{Z}_M, P_R \neq 0} P_R = 1$, we have that $\sum_{R \in \mathbb{Z}_M, P_R \neq 0} P_R^2 \geq M/\lambda \cdot (\lambda/M)^2 = \lambda/M$. It follows that

$$\sum_{R \in \mathbb{Z}_M} \left(P_R - \frac{1}{M} \right)^2 \geq \frac{\lambda - 1}{M}$$

and together with (4.4) we obtain that

$$F(\lambda) \leq \frac{M-1}{(\lambda-1)|\mathcal{B}|} M^n \quad . \tag{4.5}$$

The fraction becomes arbitrarily small choosing M slightly smaller than $|\mathcal{B}|$ and λ large enough.

4.2. Two-level representation-technique algorithm

In order to find the solution $\mathbf{x} \in \{0,1\}^n$ of weight $\ell = \alpha n$ to the problem

$$\sum_{i=1}^{n} a_i x_i = S \quad , \tag{4.6}$$

Howgrave-Graham and Joux [HGJ10] propose to apply the representation technique twice: We suppose that $\ell = \frac{n}{2}$ and that $4|\ell$. At first the solution is represented by tuples of vectors of weight $\ell/2$ and a modular constraint with respect to a modulus M_1. Each of the partial solutions is then again represented by tuples of vectors of weight $\ell/4$ that satisfy a modular constraint with respect to a second modulus M_2. We therefore search a representation of the solution as a set of four vectors $\mathbf{x}^{(j)}$ each of length n and weight $\ell/4$ such that $\mathbf{x} = \mathbf{x}^{(1)} + \mathbf{x}^{(2)} + \mathbf{x}^{(3)} + \mathbf{x}^{(4)}$. The six subproblems are:

$$\mathbf{a} \cdot \mathbf{x}^{(j)} \equiv R_j \mod M_2 \tag{4.7}$$

for $j = 1,2,3,4$ where $\mathbf{x}^{(j)} \in \{0,1\}^n$ is of weight $\ell/4$ and

$$\begin{aligned} \mathbf{a} \cdot (\mathbf{x}^{(1)} + \mathbf{x}^{(2)}) &\equiv R && \mod M_1M_2 \text{ and} \\ \mathbf{a} \cdot (\mathbf{x}^{(3)} + \mathbf{x}^{(4)}) &\equiv S - R && \mod M_1M_2 \end{aligned} \tag{4.8}$$

where $\mathbf{x}^{(1)}+\mathbf{x}^{(2)}$ and $\mathbf{x}^{(3)}+\mathbf{x}^{(4)}$ have weight $\ell/2$. The targets R and R_1, R_3 are chosen uniformly at random in $\mathbb{Z}_{M_1M_2}$ and $\mathbb{Z}_{M_2}$, respectively. The other targets are set to: $R_2 = R - R_1 \mod M_2$ and $R_4 = S - R - R_3 \mod M_2$. Figure 4.2 illustrates the tree of subproblems we have to solve in order to find the solution to the original knapsack problem.

The new algorithm performs the following steps: Each subproblem (4.7) at the bottom of the tree is solved by a classical algorithm, e.g., the Schroeppel-Shamir algorithm, creating four lists $\mathcal{L}_j^{(2)}$. As we want to solve modular knapsacks, the algorithm has to run several times for each list with modified targets as explained in Section 2 and demonstrated by Algorithm 4.2.

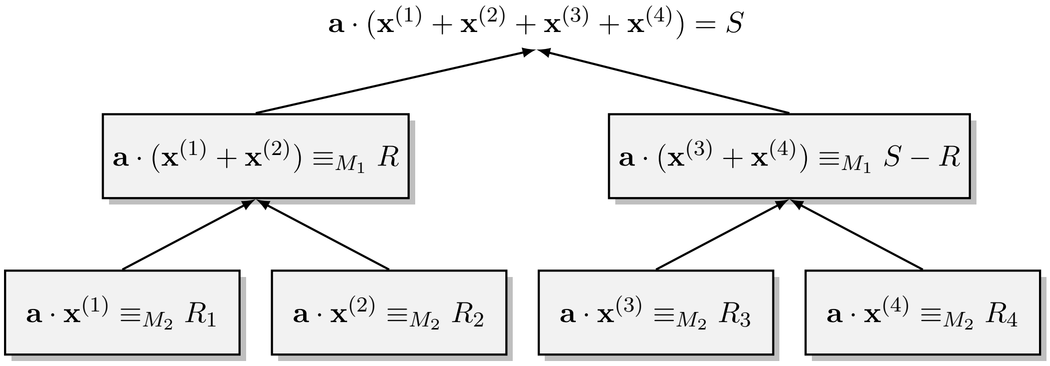

$\mathbf{a} \cdot (\mathbf{x}^{(1)} + \mathbf{x}^{(2)} + \mathbf{x}^{(3)} + \mathbf{x}^{(4)}) = S$
$\mathbf{a} \cdot (\mathbf{x}^{(1)} + \mathbf{x}^{(2)}) \equiv_{M_1} R$
$\mathbf{a} \cdot (\mathbf{x}^{(3)} + \mathbf{x}^{(4)}) \equiv_{M_1} S - R$
$\mathbf{a} \cdot \mathbf{x}^{(1)} \equiv_{M_2} R_1$
$\mathbf{a} \cdot \mathbf{x}^{(2)} \equiv_{M_2} R_2$
$\mathbf{a} \cdot \mathbf{x}^{(3)} \equiv_{M_2} R_3$
$\mathbf{a} \cdot \mathbf{x}^{(4)} \equiv_{M_2} R_4$

Figure 4.2.: *Tree of subknapsack problems.*

The next step is to join the lists and to keep elements

$$\begin{aligned}\mathbf{x}^{(1)} + \mathbf{x}^{(2)} &\in \mathcal{L}_1^{(1)} = \mathcal{L}_1^{(2)} \bowtie \mathcal{L}_2^{(2)} \text{ and} \\ \mathbf{x}^{(3)} + \mathbf{x}^{(4)} &\in \mathcal{L}_2^{(1)} = \mathcal{L}_3^{(2)} \bowtie \mathcal{L}_4^{(2)}\end{aligned}$$

of weight $\ell/2$ that fulfil (4.8). A final join between $\mathcal{L}_1^{(1)}$ and $\mathcal{L}_2^{(1)}$ creates elements $\mathbf{x} = \mathbf{x}^{(1)} + \mathbf{x}^{(2)} + \mathbf{x}^{(3)} + \mathbf{x}^{(4)}$ for which

$$\mathbf{a} \cdot (\mathbf{x}^{(1)} + \mathbf{x}^{(2)} + \mathbf{x}^{(3)} + \mathbf{x}^{(4)}) \equiv S \mod M_1 M_2$$

by construction. If equality holds over the integers and $\mathbf{x}$ is of weight ℓ, the algorithm has found the solution to (4.6). If no solution is found, new targets are chosen and the above steps are repeated. Algorithm 4.1 presents the technique. It is a variation of [HGJ10, Algorithm 5] for the case $\ell = \frac{n}{2}$.

Algorithm 4.1: Two-level representation technique

```
Input:   $\mathbf{a} = (a_1, .., a_n)$, $S$, $M_1$, $M_2$, weight of solution $\ell$
Output:  $\mathbf{x}$ such that $\mathbf{a} \cdot \mathbf{x} = S$

Split knapsack $\mathbf{a}$ into two disjoint sets $\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$.
List $\mathcal{B}_1 \leftarrow$ Compute all sums of $\ell/8$ elements w.r.t $\mathbf{a}^{(1)}$.
List $\mathcal{B}_2 \leftarrow$ Compute all sums of $\ell/8$ elements w.r.t $\mathbf{a}^{(2)}$.

Repeat
    Choose random elements $R \in \mathbb{Z}_{M_1 M_2}, R_1, R_3 \in \mathbb{Z}_{M_2}$
    Set $R_2 = R - R_1$, $R_4 = S - R - R_3 \mod M_2$

    List $\mathcal{L}_1^{(2)} \leftarrow$ create-base-list($\mathcal{B}_1, \mathcal{B}_2, R_1, M_2, \ell/4$) (see Alg. 4.2)
    List $\mathcal{L}_2^{(2)} \leftarrow$ create-base-list($\mathcal{B}_1, \mathcal{B}_2, R_2, M_2, \ell/4$)
    List $\mathcal{L}_3^{(2)} \leftarrow$ create-base-list($\mathcal{B}_1, \mathcal{B}_2, R_3, M_2, \ell/4$)
    List $\mathcal{L}_4^{(2)} \leftarrow$ create-base-list($\mathcal{B}_1, \mathcal{B}_2, R_4, M_2, \ell/4$)
     List $\mathcal{L}_1^{(1)}$  $\leftarrow$  merge-join($\mathcal{L}_1^{(2)}$, $\mathcal{L}_2^{(2)}$, $M_1$, $R$, $\ell/2$) (filter for vectors of length $\ell/2$)
     List $\mathcal{L}_2^{(1)}$  $\leftarrow$  merge-join($\mathcal{L}_3^{(2)}$, $\mathcal{L}_4^{(2)}$, $M_1$, $S - R \mod M_1$, $\ell/2$)
     $\mathbf{x}$               $\leftarrow$  merge-join($\mathcal{L}_1^{(1)}$, $\mathcal{L}_2^{(1)}$, $S$, $\ell$) (filter for vectors of length $\ell$)

  done about $\log_2(M_1 M_2^2)$ times ;
```

Choice of the moduli To ensure a good probability of success, we are interested in choosing the applying modular constraints not too high such that we can still find representations. At the same time, a larger modular constraint keeps the lists small. We choose M_2 as a prime close to the number of representations at the second level, N_2, i.e., $|M_2| \lessapprox N_2 = \binom{\ell/2}{\ell/4} = \tilde{\mathcal{O}}\left(2^{\frac{\ell}{2}}\right)$.

Algorithm 4.2: Create base list

```
Input:   Lists $\mathcal{B}_j$ of disjoint sums of $\ell/8$ knapsack elements, target $R$, module $M$,
         weight of solution $\ell/4$
Output:  List $\mathcal{L}$ of elements $\mathbf{x}$ of weight $\ell/4$ such that $\mathbf{a}\cdot\mathbf{x} \equiv R \mod M$

For each $Target \in \{R + k(M-1) \mid k = 1, .., (n/4-1)\}$
    $\mathcal{L} \leftarrow$merge-join$(\mathcal{B}_1, \mathcal{B}_2, M, \text{Target}, \ell/4)$ (compute concatenation of vectors)
```

The elements in the middle level satisfy

$$\begin{array}{lllll} \mathbf{a}\cdot(\mathbf{x}^{(1)}+\mathbf{x}^{(2)}) & \equiv & R_1+R_2 & \equiv & R & \mod M_2 \text{ and} \\ \mathbf{a}\cdot(\mathbf{x}^{(3)}+\mathbf{x}^{(4)}) & \equiv & R_3+R_4 & \equiv & S-R & \mod M_2 \ . \end{array}$$

If M_2 is co-prime to M_1, it is sufficient to check if

$$\begin{array}{llll} \mathbf{a}\cdot(\mathbf{x}^{(1)}+\mathbf{x}^{(2)}) & \equiv & R & \mod M_1 \text{ and} \\ \mathbf{a}\cdot(\mathbf{x}^{(3)}+\mathbf{x}^{(4)}) & \equiv & S-R & \mod M_1 \end{array}$$

to satisfy (4.8) due to the Chinese reminder theorem. We choose the applying modular constraints on the first level of order of the number of representations $N_1 = \binom{\ell}{\ell/2}$. The partial modulus M_1 is a prime such that $|M_1M_2| \lessapprox N_1 = \binom{\ell}{\ell/2} = \tilde{\mathcal{O}}\left(2^{\ell}\right)$ and we obtain $|M_1| = \tilde{\mathcal{O}}\left(2^{\frac{\ell}{2}}\right)$. In this way, the expected size of the intermediate lists is kept small while we can still expect one representation on average for almost all targets. We also want to ensure that the intermediate knapsacks are random which is why we need $M_1M_2 < 2^n$. The reduced knapsack weights $a_i \pmod{M_i}$ can then be assumed to be equally distributed values in $\mathbb{Z}_{M_1M_2}$.

4.2.1. Minimal time and memory complexity in the balanced case

We assume that $\ell = \frac{n}{2}$ and that $8|\ell$. The first step is to give an estimation for the complexity of algorithm 4.2 that creates the baselists. It searches elements of length n and weight $\frac{\ell}{4}$ that satisfy (4.7). To this purpose it creates lists of size

$$B_{2l} = \binom{n/2}{\ell/8} \approx \tilde{\mathcal{O}}\left(2^{0.272\,n}\right)$$

containing all possible vectors of length $\frac{n}{2}$ and weight $\ell/8$. The number of collisions C_B between those lists is of size $B_{2l}^2/N_2 \approx 2^{0.294\,n}$ on average. We expect that

$$L^{(2)} = \binom{n}{\ell/4}/N_2 \approx \binom{n}{\ell/4}/2^{\frac{\ell}{2}} = \tilde{\mathcal{O}}\left(2^{0.294n}\right)$$

elements are found for each list $\mathcal{L}_j^{(2)}$. We estimate the running time of algorithm 4.2 as

$$\tilde{\mathcal{O}}\left(\max(L^{(2)}, B_{2l}, C_B)\right) = \tilde{\mathcal{O}}\left(2^{0.294n}\right) \ .$$

The algorithm does also perform a repeated call of a merge-join routine that uses a sorting algorithm. These steps introduce logarithmic and polynomial factors in n which we omit in the asymptotic analysis. As we solve modular knapsacks, we have to repeat these steps with changing targets as explained in section 2. This adds a polynomial factor of order n to the time which is neglected if we use the $\tilde{\mathcal{O}}$-notation.

The time to create lists $\mathcal{L}_1^{(1)} = \mathcal{L}_1^{(2)} \bowtie \mathcal{L}_2^{(2)}$ and $\mathcal{L}_2^{(1)} = \mathcal{L}_3^{(2)} \bowtie \mathcal{L}_4^{(2)}$ of the next level is of order $\max(L^{(2)}, L^{(1)}, C_2)$ where

$$C_2 \approx (L^{(2)})^2/N_1 = \tilde{\mathcal{O}}\left(2^{0.337n}\right)$$

is the expected number of collisions and $L^{(1)}$ the expected number of colliding elements of correct weight. We expect to find

$$L^{(1)} = \binom{n}{\ell/2}/N_1 = \tilde{\mathcal{O}}\left(2^{0.311n}\right)$$

elements for $\mathcal{L}_1^{(1)}, \mathcal{L}_2^{(1)}$ of correct weight. A last merge then takes time $\tilde{\mathcal{O}}\left(\max(L^{(1)}, C_1)\right)$ where

$$C_1 \approx (L^{(1)})^2/(2^n/N_1) = \tilde{\mathcal{O}}\left(2^{0.123n}\right)$$

is the expected number of collisions that we have to check in order to find the solution.

The algorithm is of asymptotic running time

$$\mathcal{T}_{\text{2levels}} = \tilde{\mathcal{O}}\left(\max(B_{2l}, C_B, \max_j |\mathcal{L}_j^{(2)}|, C_2, \max_j |\mathcal{L}_j^{(1)}, C_1)\right)$$

using space

$$\mathcal{M}_{\text{2levels}} = \tilde{\mathcal{O}}\left(\max(B_{2l}, L^{(2)}, L^{(1)})\right) .$$

We could also use a Shamir Schroeppel algorithm to create the elements at the second level. The term B_{2l} then stands for the time needed by Schroeppel-Shamir to create intermediate lists (see Sect. 3.2). Assuming that each list has a size close to its expected value, the expected running time is:

$$\mathcal{T}_{\text{2levels}} = \tilde{\mathcal{O}}\left(\max(B_{2l}, \frac{B_{2l}^2}{M_2}, L^{(2)}, \frac{(L^{(2)})^2}{M_1 M_2}, L^{(1)}, (L^{(1)})^2 \cdot \frac{M_1 M_2}{2^n})\right) . \tag{4.9}$$

We see that the overall memory requirement is $\tilde{\mathcal{O}}\left(2^{0.311n}\right)$ dominated by the number of elements of the first level. The time complexity is dominated by the join to enumerate them and is of order[1] $\tilde{\mathcal{O}}\left(2^{0.337n}\right)$.

A third application of the representation technique does not change these cost and does hence not ameliorate the time complexity. It can however reduce the size of the lists. A more

[1]The cost to merge the lists were neglected in [HGJ10] and corrected in [MM11, BCJ11]

efficient way to reduce the memory requirement, at the cost of longer intermediate calculations, is to choose larger moduli. The total running time still stays the same in the asymptotic sense. Above we chose them as large as possible while still expecting one representation per random target. Section 4.2.2 proposes a memory efficient algorithm of same running time comparing a two-level with a three-level algorithm.

Three levels of representations. For a third level of representations, we introduce a third modulus M_3 and set the constraints at the second and first level to M_2M_3 and $M_1M_2M_3$, respectively. The number of representations at the third level is

$$N_3 = \binom{\ell/2}{\ell/4} = \tilde{\mathcal{O}}\left(2^{\frac{\ell}{4}}\right) \quad .$$

The expected number of elements at the third level is:

$$L^{(3)} = \binom{n}{\ell/8}/N_3 \approx \binom{n}{\ell/8}/2^{\frac{\ell}{4}} = \tilde{\mathcal{O}}\left(2^{0.212\,n}\right),$$

which are obtained from base lists of size

$$B_{3l} = \binom{n/2}{n/16} = \tilde{\mathcal{O}}\left(2^{0.169\,n}\right) \quad .$$

The running time of algorithm 4.2 is of order

$$\max\left(L^{(3)}, B_{3l}, C_B\right) = \tilde{\mathcal{O}}\left(2^{0.212\,n}\right)$$

where $C_B \approx \frac{(B_{3l})^2}{M_3}$. A join of lists at level three treats about

$$C_3 \approx \frac{(L^{(3)})^2}{M_2} = \tilde{\mathcal{O}}\left(2^{0.30\,n}\right)$$

elements. Let us assume that each list has a size close to its expected value. The average running time is then:

$$\begin{aligned}
&\mathcal{T}_{3\text{levels}} \\
&= \tilde{\mathcal{O}}\left(\max(B_{3l}, \frac{B_{3l}^2}{M_3}, L^{(3)}, \frac{(L^{(3)})^2}{M_2}, L^{(2)}, \frac{(L^{(2)})^2}{M_1}, L^{(1)}, (L^{(1)})^2 \cdot \frac{M_1M_2M_3}{2^n})\right) \\
&= \tilde{\mathcal{O}}\left(C_2\right) \quad .
\end{aligned} \tag{4.10}$$

The expected memory requirement is

$$\mathcal{M}_{3\text{levels}} = \tilde{\mathcal{O}}\left(\max(B_{3l}, L^{(3)}, L^{(2)}, L^{(1)})\right) = \tilde{\mathcal{O}}\left(L^{(1)}\right) \quad .$$

Note that we do not gain in time nor in memory as the number of collisions C_2 and the size of the lists at level one, $L^{(1)}$, are unchanged. We can change the memory requirement as explained in section 4.2.2.

4.2.2. Improvements on the memory requirement

We so far assumed that memory is plentiful. To spare memory in practice, we propose two techniques. An efficient join operation, used as subroutine in the above two-level algorithm, needs only to store one of the starting lists while the elements of the second list are created and joined on the fly. This has no impact on our asymptotic analysis.

Additionally, we can choose a slightly larger modulus per level thus reducing the expected size of the starting lists. More precisely, we choose an integer m co-prime to the modulus M. Beside the normal constraint w.r.t. M, we take successively every value in $\mathbb{Z}/m\mathbb{Z}$ and keep only elements in a list at each run that satisfy both constraints. By Chinese remainder theorem, all elements fulfil a constraint modulo Mm. Heuristically, we can assume that the lists are smaller by a factor m. A join between these shortened lists creates a list in the upper level that is smaller by exactly the same factor than the starting lists. To recover all elements in the upper level, we repeat the process with the next value in $\mathbb{Z}/m\mathbb{Z}$. The number of repetitions is m.

The total running time $\mathcal{T}_{\text{2levels}} = \tilde{\mathcal{O}}\left(C_2\right)$ derived in section 4.2.1 stays the same. We can apply this technique in both levels of the two-level algorithm and can reduce the values $L^{(i)}$. The size of the bottom lists determines the minimal memory requirement.

Reduce lists. We start with the lists at the first level and increase the modular constraints: $M = M_1M_2 \approx N_1$ and $m \approx \Lambda$ for real parameter $\Lambda \geq 1$ (exponential in n) such that $\gcd(M, m) = 1$. The parameter is not completely free as we require a same overall running time $\mathcal{T}_{\text{2levels}}$. The solution vector is searched within two lists of size $\frac{L^{(1)}}{\Lambda}$ which reduces the expected number of collisions by a factor Λ. We repeat Λ times with changed target modulo m to find the same number of collisions C_1 than before. Let $\mathcal{T}_i$ denote the time to create the lists of level i. The time to find the solution can then be computed as

$$\mathcal{T}_0 = \Lambda \cdot max(\frac{C_1}{\Lambda}, \mathcal{T}_1)$$

where

$$\Lambda \cdot \mathcal{T}_1 = \Lambda \cdot max(\frac{L^{(1)}}{\Lambda}, \frac{C_2}{\Lambda}, \mathcal{T}_2) = \max(L^{(1)}, C_2, \Lambda \cdot \mathcal{T}_2) \ .$$

The memory in the first level is now reduced but the overall memory requirement is still determined by the lists at the second level, we apply the same idea again. Let $\Delta \geq 1$ be a real parameter. We choose a second small modulus $m' \approx \Delta$ co-prime to $M_2 \approx N_2$. This reduces the expected size of the lists to $L'^{(2)} = L^{(2)}/\Delta$. The drawback is that we can only expect to find C_2/Δ collisions for the lists in the level above. However, if we repeatedly create lists of size $L'^{(2)}$ with changed target modulo m', we create all $L^{(1)}$ elements in the upper level after Δ iterations.

The time to create all lists of level two of size $L^{(2)}$ is given by the time to construct Δ times lists of expected size $L'^{(2)}$:

$$\mathcal{T}_2 = \Delta \cdot max(\frac{L^{(2)}}{\Delta}, \frac{C_B}{\Delta}, B_{2l}) = max(L^{(2)}, C_B, \Delta \cdot B_{2l}) \ .$$

The running time is unchanged if $\mathcal{T}_0 = \mathcal{T}_{\text{2levels}}$, that is, if

$$\mathcal{T}_0 = \max(C_1, L^{(1)}, C_2, \Lambda \cdot L^{(2)}, \Lambda \cdot C_B, \Lambda \cdot \Delta \cdot B_{2l})$$

equals

$$\mathcal{T}_{\text{2levels}} = \tilde{\mathcal{O}}\left(C_2\right) \ .$$

As $L^{(2)} \leq C_B$, we need to ensure that

$$C_2 \geq \Lambda \cdot \max(C_B, \Delta \cdot B_{2l}) = \Lambda \cdot B_{2l} \max(\frac{B_{2l}}{N_2}, \Delta) \ .$$

We can thus choose a maximal $\Delta = \frac{B_{2l}}{N_2}$ leading to a maximal modulus $M_2 = B_{2l}$. The memory requirement is down to

$$\max(L'^{(1)}, L'^{(2)}, B_{2l}) = \max(\frac{L^{(1)}}{\Lambda}, \frac{L^{(2)}}{\Delta}, B_{2l}) \ .$$

which is minimal for $\frac{L^{(1)}}{\Lambda} = \frac{L^{(2)}}{\Delta}$. We obtain a larger modulus for the first level by a factor of $\Lambda = \frac{L^{(1)}}{L^{(2)}} \frac{B_{2l}}{N_2}$ such that all stored lists are of same size B_{2l}. We remark that in this way the cost to create the elements at the second level is asymptotically the same than to store them. The difference in practice lies in logarithmic factors caused by storage and sorting.

Minimal running time with reduced memory requirement. For slightly larger modular constraints,

$$|mM_1M_2| \approx 2^{0.540\,n} \text{ and } |m'M_2| \approx 2^{0.272\,n},$$

the memory requirement drops to $B_{2l} \approx L'^{(1)} \approx L'^{(2)}) = \tilde{\mathcal{O}}\left(2^{0.272\,n}\right)$. We need to repeat the join routines with changed targets about $m'M_2/2^{\frac{n}{4}} = 2^{0.22\,n}$ times for the second level and about $mM_1M_2/2^{\frac{n}{2}} = 2^{0.04\,n}$ times for the last join. As the minimal memory is blocked by the bottom lists, the gain in memory is optimal. The running time is unchanged at $\tilde{\mathcal{O}}\left(2^{0.337\,n}\right)$. In practice, we need to store only one of the lists that we join. This does however not change the asymptotic memory requirement. As the minimal size of the lists is determined by the bottom lists, we can try to add a third level of decomposition.

Reduce memory for three levels of representations. We want to permit slightly larger moduli than the number of representation per level. To this end we choose three moduli m, m' and m'' of size $\kappa, \Delta \geq 1$ and $\Lambda \geq 1$, respectively, that are co-prime to the moduli. The applying modulo per level three, two and one are:

$$m''M_3 \approx \kappa \cdot N_3, \quad m''M_2M_3 \approx \Delta \cdot N_2, \quad mM_1M_2M_3 \approx \Lambda \cdot N_1 \ .$$

The reasoning to bound the parameters is similar to the analysis above. We want to minimize the expected memory requirement per iteration given by

$$\max(L'^{(1)}, L'^{(2)}, L'^{(3)}, B_{3l}) = \max(\frac{L^{(1)}}{\Lambda}, \frac{L^{(2)}}{\Delta}, \frac{L^{(3)}}{\kappa}, B_{3l})$$

under the constraint that the overall asymptotic running time stays the same as $\mathcal{T}_{3\text{levels}}$. The time to create the lists at level two becomes

$$\mathcal{T}_2 = \Delta \cdot max(\frac{L^{(2)}}{\Delta}, \frac{C_3}{\Delta}, \mathcal{T}_3) \text{ where}$$

$$\mathcal{T}_3 = \kappa \cdot max(\frac{L^{(3)}}{\kappa}, \frac{C_B}{\kappa}, B_{3l}) \ .$$

To obtain the same asymptotic running time, we require that

$$\mathcal{T}_0 = \max(C_1, L^{(1)}, C_2, \Lambda \cdot L^{(2)}, \Lambda \cdot C_3, \Lambda \cdot \Delta \cdot L^{(3)}, \Lambda \cdot \Delta \cdot C_B, \Lambda \cdot \Delta \cdot \kappa \cdot B_{3l})$$

equals

$$\mathcal{T}_{2\text{levels}} = \tilde{\mathcal{O}}\left(C_2\right) \ .$$

As the number of colliding elements is always larger than or equal to the number of consistent elements, we can simplify the constraint to

$$C_2 \geq \max(\Lambda \cdot C_3, \Lambda \cdot \Delta \cdot C_B, \Lambda \cdot \Delta \cdot \kappa \cdot B_{3l}) \ .$$

The average value $L^{(1)}$ is dictating the asymptotic space complexity so far. By the equation above, we can reduce the memory at the first level by a maximal Λ for which holds that $C_2 \approx \Lambda \cdot C_3$. In this way, $\frac{L^{(1)}}{\Lambda} = \tilde{\mathcal{O}}\left(2^{0.274\,n}\right)$. Comparing the value to the result for a two-level algorithm, where all lists are of size $\tilde{\mathcal{O}}\left(2^{0.272\,n}\right)$, with larger moduli, we see that we do not attain a better result here.

4.2.3. Minimize time complexity in the unbalanced case

To describe the asymptotic complexity in the more general case, that is, where $\ell = \alpha n$, we first write the various partial cost in terms of α and n using the binary entropy function of section A.1 in the appendix:

$$h(\alpha) = -\alpha \log_2(\alpha) - (1-\alpha)\log_2(1-\alpha) \ .$$

Let $L^{(2)}$ be the size of the maximal list of the bottom level, that is, $L^{(2)} = \max(|\mathcal{L}_j^{(2)}|)$ for $j = 1, 2, 3, 4$. The expected value of $L^{(2)}$ in the asymptotic case is $\binom{n}{\ell/4}/M_2 \approx 2^{n(h(\alpha/4)-\alpha/2)}$. The partial cost B_{2l} of a classical algorithm that creates the bottom lists becomes $B_{2l} = \binom{n/2}{\ell/8} \approx 2^{\frac{n}{2}(h(\alpha/4)}$. Let $L^{(1)}$ denote the size of the maximal list of $\{\mathcal{L}_1^{(1)}, \mathcal{L}_2^{(1)}\}$ at the middle level. Its expected value is $\binom{n}{\ell/2}/(M_1 M_2) \approx 2^{n(h(\alpha/2)-\alpha)}$. The expected number of collisions during the join of lists $\mathcal{L}_j^{(2)}$ are $C_2 \approx 2^{n(2h(\alpha/4)-3/2\alpha)}$. We expect $C_1 \approx 2^{n(2h(\alpha/2)-\alpha-1)}$ colliding elements between the lists $\mathcal{L}_1^{(1)}$ and $\mathcal{L}_2^{(1)}$. The overall memory requirement is then given by

$$\mathcal{M}_{\text{2levels}} = \tilde{\mathcal{O}}\left(\max(L^{(2)}, L^{(1)})\right) = \tilde{\mathcal{O}}\left(2^{m(\alpha)n}\right) \text{ where}$$

$$m(\alpha) = \max(h(\alpha/4) - \alpha/2, h(\alpha/2) - \alpha) \ .$$

The algorithm is of asymptotic running time

$$\mathcal{T}_{\text{2levels}} = \tilde{\mathcal{O}}\left(\max(L^{(2)}, L^{(1)}, B_{2l}, C_1, C_2)\right) = \tilde{\mathcal{O}}\left(2^{t(\alpha)n}\right) \text{ where}$$

$$t(\alpha) = \max(h(\alpha/4) - \alpha/2, h(\alpha/2) - \alpha, h(\alpha/4)/2, 2h(\alpha/2) - \alpha - 1, 2h(\alpha/4) - 3/2\alpha) \ .$$

Figure 4.3 shows the individual cost (the terms of $t(\alpha)$) of the two-level representation algorithm for α varying between 0 and 1.

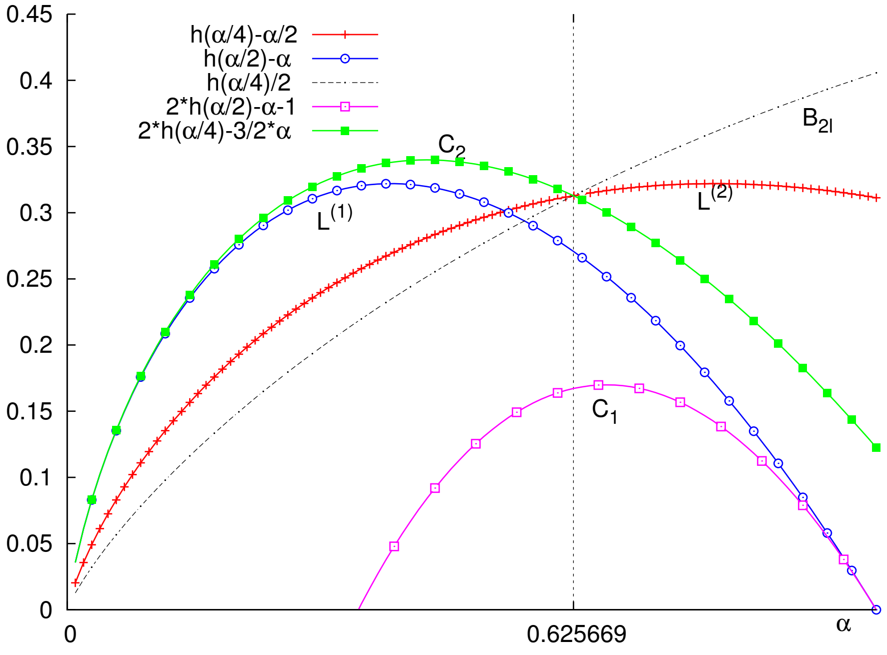


Figure 4.3.: *Asympt. time complexity for 2-level algorithm where* $h(\alpha) = -\alpha\log_2(\alpha) - (1-\alpha)\log_2(1-\alpha)$ *is the binary entropy function.*

For $\alpha = 0.5$, we rediscover the result of the previous section. The curve for C_2 is dominating the time complexity. We will now determine the algorithm that achieves an optimal running time for $\alpha \neq 0.5$. Looking at the graph, we see that for $\alpha \leq 0.626$, $C_2(\alpha)$ is dominating while for greater α the cost to create the bottom lists, B_{2l}, determines the complexity. In this case, we have three options. First, we can change to the complementary knapsack $a \cdot (1-x)$ whose solution has weight $(1-\alpha)n$. Second, we can add an additional level of representations to our algorithm to lower the cost at the bottom. Third, for small α (or large α due to the complement trick), it might even be sufficient to use a one-level algorithm. The best choice is given by the minimal complexity.

An algorithm that uses three levels of representation technique starts to enumerate vectors of length n and weight $\ell/8$ by Schroeppel-Shamir compliant to a modular constraint. We call the respective modulus M_3 and choose it of a size close to the number of representations: $|M_3| \approx \binom{\ell/4}{\ell/8} = \tilde{\mathcal{O}}\left(2^{\ell/4}\right)$. The resulting lists are $\mathcal{L}_j^{(3)}$ for $j = 1, .., 16$ and are of expected size $L^{(3)} = \max\{\mathcal{L}_j^{(3)}\} \approx \binom{n}{\ell/8}/M_3 = \tilde{\mathcal{O}}\left(2^{(h(\alpha/8)-\alpha/4)n}\right)$. The cost to enumerate the bottom lists is given by $\max(B_{3l}, L^{(3)})$ where $B_{3l} \approx \binom{n/2}{\ell/16} = \tilde{\mathcal{O}}\left(2^{(h(\alpha/8)\,\frac{n}{2})}\right)$. We then obtain the elements of the above level by a merge-join in expected time $C_3 \approx (L^{(3)})^2/M_2 = \tilde{\mathcal{O}}\left(2^{(2h(\alpha/8)-3/4\alpha)n}\right)$.

A simple one-level algorithm creates the lists $\mathcal{L}_j^{(1)}$, $j = 1, 2$, by means of a Shroeppel-Shamir in time $\tilde{\mathcal{O}}\left(\max(B_{1l}, L^{(1)})\right)$ where $B_{1l} \approx \binom{n/2}{\ell/4} = \tilde{\mathcal{O}}\left(2^{(h(\alpha/2))\,\frac{n}{2}}\right)$.

In figure 4.4 we added the new terms $B_{1l}, B_{3l}, L^{(3)}, C_3$. If we compare the cost $L^{(1)}, C_2$ and B_{2l} for small α ($\alpha \leq 0.341$), we see that a one-level algorithm is in favour as the two blue curves for $L^{(1)}, B_{1l}$ lie below the green line for C_2. This has to be compared to large α as we could also solve the complementary knapsack by a two-level or three-level algorithm. For $\alpha \geq 0.681$, we see that the curve of B_{2l} is above C_3 which is the maximum cost for a three-level algorithm.

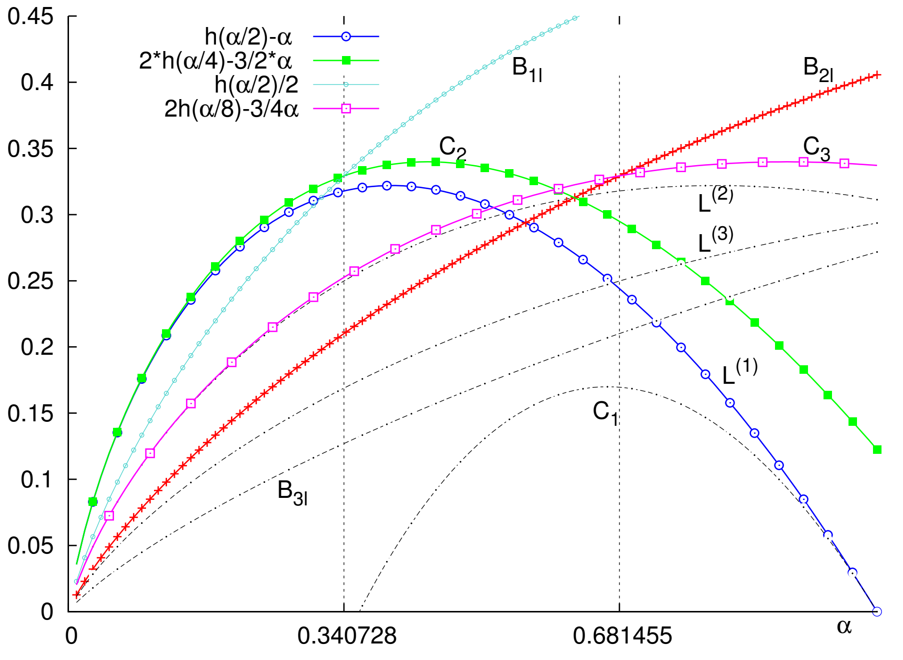


Figure 4.4.: *Asympt. time complexity for 1-, 2- and 3-level algorithm where* $h(\alpha) = -\alpha \log_2(\alpha) - (1-\alpha)\log_2(1-\alpha)$ *is the binary entropy function.*

In general, if we compute the minimum over

$$\{\mathcal{T}_{\text{2levels}}(\alpha), \mathcal{T}_{\text{1level}}(\alpha), \mathcal{T}_{\text{3levels}}(\alpha), \mathcal{T}_{\text{2levels}}(1-\alpha), \mathcal{T}_{\text{1level}}(1-\alpha), \mathcal{T}_{\text{3levels}}(1-\alpha\})$$

for each α, we obtain the minimal complexity

$$\mathcal{T}(n) = 2^{t(\alpha)+o(n)}$$

that one can reach by representation technique. The result is depicted in figure 4.5.

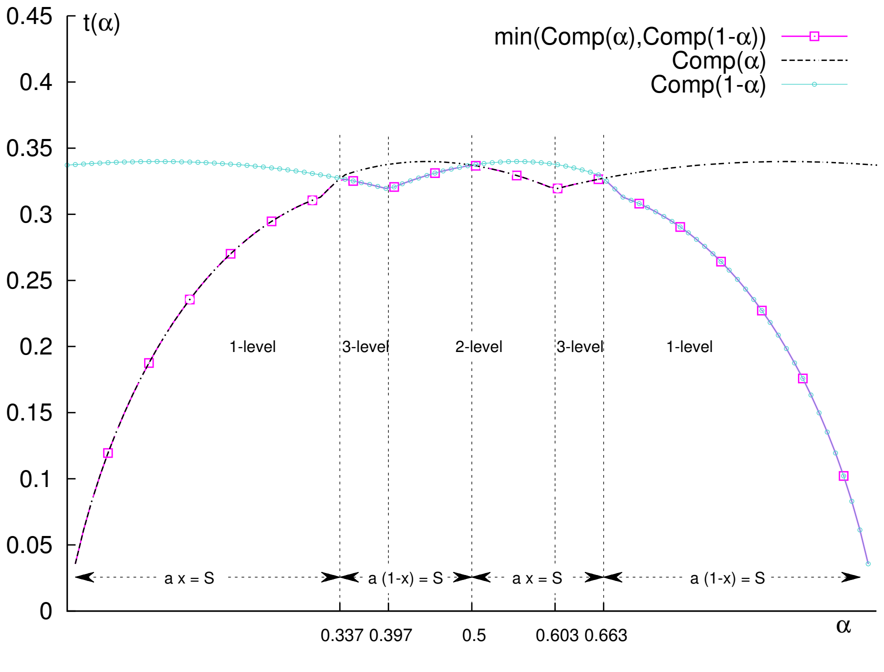


Figure 4.5.: *Minimal time complexity to solve the knapsack problem by representation-technique algorithm of varying number of levels;* $Comp(.) = min(\mathcal{T}_{2levels}(.), \mathcal{T}_{1level}(.), \mathcal{T}_{3levels}(.))$.

The largest coefficient for the time complexity occurs for $\alpha = 0.5$. Due to the possibility to switch to the complementary knapsack, the complexity to find a solution of weight α is the same as for a weight $1-\alpha$ which leads to a symmetric graph.

We see that a change of the number of levels as well solving the complementary knapsack may be advantageous if we simply compare the exponents of the exponential in the asymptotic time complexity. The difference may then vary up to 0.025 as is the case for $\alpha = 0.397$.

We observe a peak in the complexity at $\alpha = 0.337$ which is the value for which a 1-level algorithm on the given knapsack problem becomes less efficient than a three-level algorithm on the inverse knapsack. The analogue happens for $\alpha = 0.663$. This is not consistent to the understanding and suggests that the algorithms are not optimal. Also for a practical implementation slightly varied limits for α are probable. In the following chapter, we will extend the representation technique. The running time of the new algorithm lies always below the curve in 4.5 as shown in figure 5.3.

Chapter 5

Extended-representation-technique algorithm

The idea of representing a solution of a knapsack problem by two binary vectors of full length and of half weight can be extended. Instead of decomposing the original solution into two *binary* coefficient vectors, we allow coefficients $\{-1, 0, 1\}$ for the partial solutions. By adding (a few) -1 coefficients, we search for partial solutions of slightly increased weight. This provides us with an additional degree of freedom as it increases the number of representations per solution and allows in return to reduce the size of the intermediate lists of our new algorithm leading to a better running time in the end.

Increased number of representations per solution. We choose a parameter α where αn is the number of -1 positions per partial solution. A representation of a solution $\mathbf{x}$ is now a tuple $(\mathbf{y}, \mathbf{z}) \in \{-1, 0, 1\}^n \times \{-1, 0, 1\}^n$ such that $\mathbf{x} = \mathbf{y} + \mathbf{z}$ and where $\mathbf{y}, \mathbf{z}$ are of same weight containing $\ell/2 + \alpha n$ 1's and αn -1s each. Their Hamming weight, wt(.), equals $\ell/2 + 2\alpha n$. Figure 5.1 shows one possible representation.

To compute the number of representations, we count the number of ways to represent each $x_i \in \{1, 0\}$ of the solution as a tuple (y_i, z_i) such that $y_i + z_i = x_i$. We can split the ℓ 1s into pairs $(0, 1)$ or $(1, 0)$ leading to $\binom{\ell}{\ell/2}$ possibilities. The $n - \ell$ 0s can be written as pairs $(0, 0)$, $(1, -1)$ or $(-1, 1)$. We choose each αn tuples $(1, -1)$ and $(-1, 1)$. The multinomial $\binom{n-\ell}{\alpha n, \alpha n, n-\ell-2\alpha n}$ counts the choices how to decompose the zeros of a solution.

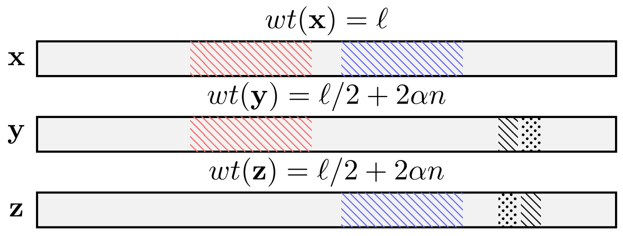

$wt(\mathbf{x}) = \ell$
$\mathbf{x}$
$wt(\mathbf{y}) = \ell/2 + 2\alpha n$
$\mathbf{y}$
$wt(\mathbf{z}) = \ell/2 + 2\alpha n$
$\mathbf{z}$

Figure 5.1.: *One representation of a solution $\mathbf{x} = \mathbf{y} + \mathbf{z}$. The striped area indicates positions of 1s and the dotted area indicates positions of -1s.*

The total number of representations is the product of these possibilities:

$$N_{BCJ} = \binom{\ell}{\ell/2}\binom{n-\ell}{\alpha n, \alpha n, n-\ell-2\alpha n} \, . \tag{5.1}$$

We see that the number of representations is increased for $\alpha > 0$ by the second factor in (5.1) compared to (4.1). Remember that, in the representation-technique algorithm, the modular constraints allowed us to reduce the size of the intermediate lists and that they were chosen close to the number of representations. A larger number of representations has therefore the potential to reduce the lists further and to gain an improved complexity.

A first intuition. Consider an algorithm that receives two lists, $\mathcal{L}_1, \mathcal{L}_2$, of elements $\mathbf{y}, \mathbf{z}$ that contain each $(\ell/2 + \alpha n)$ 1s and αn -1s and satisfy

$$\mathbf{a} \cdot \mathbf{y} \equiv R \text{ and } \mathbf{a} \cdot \mathbf{z} \equiv S - R \mod M \text{ where} \tag{5.2}$$

$|M| \approx \log_2 N_{BCJ}$. The lists are of expected size

$$L \approx \frac{\binom{n}{\ell/2+\alpha n, \alpha n, n-\ell-2\alpha n}}{M}$$

if we assume that $\mathbf{a} \cdot \mathbf{y}$ and $\mathbf{a} \cdot \mathbf{z}$ are uniformly distributed values in Z_M. A unique solution $\mathbf{x} = \mathbf{y} + \mathbf{z}$ such that $\mathbf{a} \cdot \mathbf{x} = S$ over the integers can then be found by sorting the lists and searching for collisions in time $L \log L$. As we want to ensure that $\mathbf{a} \cdot \mathbf{y}$ and $\mathbf{a} \cdot \mathbf{z}$ are random knapsacks, we constrain M to be smaller than the size of the knapsack elements, $M < 2^n$. The optimal value for α in the asymptotic case is then 0.05677 for which the lists are of minimal size, $L \approx 2^{0.173n}$.

Unfortunately, we forgot to take into account the cost to create the starting lists $\mathcal{L}_1$ and $\mathcal{L}_2$. Classic means realize this task by creating base lists $\mathcal{B}$ of all elements of length $\frac{n}{2}$ containing $(\ell/2 + \alpha n)/2$ 1s and $\alpha n/2$ -1s and searching for collisions. The cost is then determined by $\max(|\mathcal{B}|, \mathbb{E}[|\mathcal{L}_1|], \mathbb{E}[|\mathcal{L}_2|])$ where

$$|\mathcal{B}| = \binom{n/2}{(\ell/2+\alpha n)/2, \alpha n/2, (n-\ell-2\alpha n)/2} \, .$$

For the above given α, the base lists are of large size $|\mathcal{B}| = 2^{0.586n}$. The size of $\mathcal{B}$ is dominating the cost and is minimal for $\alpha = 0$ which is the standard representation technique. We do not achieve a faster algorithm than proposed by Howgrave-Graham and Joux [HGJ10] in this way. In order to obtain a more efficient algorithm, we need to render the creation of the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ more efficient. We propose to proceed as described in the following section.

5.1. Three-level extended-representation-technique algorithm

For simplicity, we assume that $8|\ell$. The asymptotic analysis of the complexity is not affected by this restriction. The first algorithm, we propose, introduces three levels in which we apply the representation technique meaning that we represent the solution by two vectors in the first level. As the cost to create those elements is too high, we apply the same idea again and represent the partial solutions in the second level, each by two vectors. Splitting up again, introduces the third level of representations which we find by classical means. In each of the three levels, we allow a different number of -1s and impose modular constraints using moduli M_1, M_2, M_3. In each level, the number of ones of the intermediate solution of the previous level is halved and a number of -1s and 1s is added given by the parameters α, β, γ. Starting from a solution of ℓ ones, the first level representations have

$$\begin{aligned} n_1^{'} &= \ell/2 + \alpha n \quad \text{1s and we add} \\ n_{-1}^{'} &= \alpha n \quad \text{-1s.} \end{aligned}$$

At the second level, we are interested in vectors having

$$\begin{aligned} n_1^{''} &= n_1'/2 + \beta n &&= \ell/4 + \alpha n/2 + \beta n \quad \text{1s and} \\ n_{-1}^{''} &= n_{-1}'/2 + \beta n &&= \alpha n/2 + \beta n \quad \text{-1s.} \end{aligned}$$

These vectors are represented by vectors of

$$\begin{aligned} n_1^{'''} &= n_1^{''}/2 + \gamma n &&= \ell/8 + \alpha n/4 + \beta n/2 + \gamma n \quad \text{1s and} \\ n_{-1}^{'''} &= n_{-1}^{''}/2 + \gamma n &&= \alpha n/4 + \beta n/2 + \gamma n \quad \text{-1s.} \end{aligned}$$

Corresponding to the partial solutions, we have to solve a knapsack problem. Figure 5.2 shows the successive decomposition of the original knapsack into 2, 4 and finally 8 subproblems. For each of the 7 decompositions, we choose one random target for the left subproblem and set the constraint of the right subproblem to fulfil the constraint of the upper level modulo the current modulus similar to (5.2). For the first level, we draw a random value $R_1^{(1)} \in \mathbb{Z}_{M_1M_2M_3}$ for the left side and set the right target to $R_2^{(1)} = S - R_1^{(1)} \mod M_1M_2M_3$. The two problems of the first level are: We search vectors $\mathbf{z}^{(1)}$ on the left side and vectors $\mathbf{z}^{(2)}$ on the right side of weight $(n_1^{'}, n_{-1}^{'})$ for which

$$\mathbf{a} \cdot \mathbf{z}^{(j)} \equiv R_j^{(1)} \mod M_1M_2M_3 \tag{5.3}$$

for $j = 1, 2$. In this way we know that

$$\mathbf{a} \cdot (\mathbf{z}^{(1)} + \mathbf{z}^{(2)}) \equiv S \mod M_1M_2M_3 \ .$$

The probability that equality holds over the integers is close to $(M_1M_2M_3)/2^n$.

We proceed in an analogue manner for the second level by picking two random values $R_1^{(2)} \in \mathbb{Z}_{M_2M_3}$ and $R_3^{(2)} \in \mathbb{Z}_{M_2M_3}$ and setting $R_2^{(2)} = R_1^{(1)} - R_1^{(2)} \in \mathbb{Z}_{M_2M_3}$ and $R_4^{(2)} = R_2^{(1)} - R_3^{(2)} \in \mathbb{Z}_{M_2M_3}$.

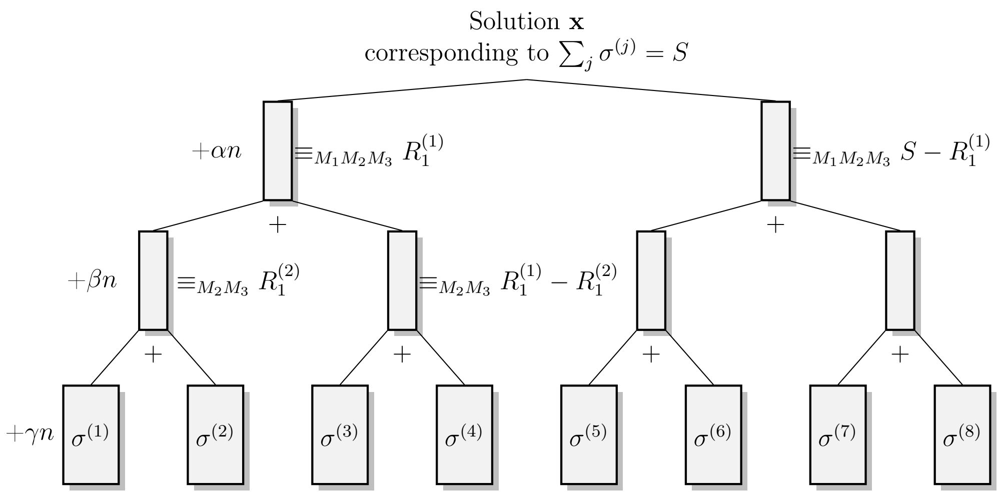

Solution $\mathbf{x}$
corresponding to $\sum_j \sigma^{(j)} = S$
$+\alpha n$
$\equiv_{M_1M_2M_3} R_1^{(1)}$
$\equiv_{M_1M_2M_3} S - R_1^{(1)}$
$+$
$+\beta n$
$\equiv_{M_2M_3} R_1^{(2)}$
$\equiv_{M_2M_3} R_1^{(1)} - R_1^{(2)}$
$+\gamma n$
$\sigma^{(1)}$
$\sigma^{(2)}$
$\sigma^{(3)}$
$\sigma^{(4)}$
$\sigma^{(5)}$
$\sigma^{(6)}$
$\sigma^{(7)}$
$\sigma^{(8)}$

Figure 5.2.: *Tree of subproblems of a three-level algorithm applying the extended representation technique.*

This determines the four subproblems of the second level: We search vectors $\mathbf{y}^{(j)}$ of weight $(n_1^{''}, n_{-1}^{''})$ that satisfy:

$$\mathbf{a} \cdot \mathbf{y}^{(j)} \equiv R_j^{(2)} \mod M_2M_3 \tag{5.4}$$

for $j = 1, 2, 3, 4$. The joined elements $\mathbf{y}^{(1)} + \mathbf{y}^{(2)}$ and $\mathbf{y}^{(3)} + \mathbf{y}^{(4)}$ satisfy the modular constraint of the above level with probability $M_2M_3/(M_1M_2M_3) = 1/M_1$.

Finally, the third level constructs elements $\mathbf{x}^{(j)} \in \{0,1\}^n$ of weight $(n_1^{'''}, n_{-1}^{'''})$ with their respective constraint:

$$\sigma^{(j)} := \mathbf{a} \cdot \mathbf{x}^{(j)} \equiv R_j^{(3)} \mod M_3 \tag{5.5}$$

for $j = 1, .., 8$ where $R_j^{(3)}$ are random elements in $\mathbb{Z}_{M_3}$ created as before.

The elements at the third level are built from the base lists $\mathcal{B}$ that contain vectors of half length:

$$\mathcal{B} = \{\varepsilon \in \{0, 1, -1\}^{\frac{n}{2}} \text{ with } n_1^{'''}/2 \text{ 1s and } n_{-1}^{'''}/2 \text{ -1s}\} \ .$$

Three levels of consecutive merge-joins reveal the solution to the original knapsack if it discovers an element $\mathbf{x} = \mathbf{x}^{(1)} + .. + \mathbf{x}^{(8)}$ of weight ℓ such that $\mathbf{a} \cdot \mathbf{x} = S$ over the integers.

Modular constraints. We follow the previous strategy of section 4.2 and choose the applying modular constraint of a size close to the number of representations, N_i for level $i = 1, 2, 3$: $M_3 \approx N_3$, $M_2M_3 \approx N_2$, $M_1M_2M_3 \approx N_1$. This ensures that each solution appearing at a given level is represented by a single partial solution at the previous level on average. A larger modular constraint would lead to a loss of solutions from one level to the next while for smaller constraints each solution would be constructed many times thus increasing the overall cost unnecessarily.

The number of representations at each level are computed analogously to (5.1) as follows:

$$\begin{aligned} N_3 &\approx \binom{n''_1}{n''_1/2} \cdot \binom{n''_0}{\gamma n, \gamma n, \star} \cdot \binom{n''_{-1}}{n''_{-1}/2} \ , \\ N_2 &\approx \binom{n'_1}{n'_1/2} \cdot \binom{n'_0}{\beta n, \beta n, \star} \cdot \binom{n'_{-1}}{n'_{-1}/2} \text{ and} \\ N_1 &\approx \binom{\ell}{n/4} \cdot \binom{n-\ell}{\alpha n, \alpha n \ \star} \ . \end{aligned} \tag{5.6}$$

In the multinomials above $\star$ counts the number of zeros that are represented at zeros in the lower level. E.g., for a term $\binom{a}{b,c,\star}$, we replace $\star$ by $a-b-c$.

Estimation of list size and number of collisions. We denote by $\mathcal{L}_j^{(i)}$ the j-th list in level i and by $L^{(i)}$ the estimated size of the lists of level i. We assume that the elements in the lists of the respective lower level are distributed in the same way for neighbouring lists. For independently at random chosen targets in level i, we can then expect that all lists are of same size in level i. The eight lists $\mathcal{L}_j^{(3)}$ on the bottom level can be constructed using a straightforward adaptation of the simple birthday paradox algorithm (or by use of Schroeppel-Shamir). It suffices to split the n knapsack elements into two random subsets of size $n/2$ and to assume that the 1s and -1s are (almost) evenly distributed between the two halves. The probability of this event is the inverse of a polynomial in n:

$$\frac{\binom{n/2}{n'''_1/2, n'''_{-1}/2, n'''_0/2}^2}{\binom{n}{n'''_1, n'''_{-1}, n'''_0}} \approx \frac{2}{\pi n \sqrt{st(1-s-t)}} = \mathcal{O}\left(1/n\right)$$

where $s := n'''_1/n$ and $t := n'''_{-1}/n$.

Thus by repeating polynomially many times, we recover all of $\mathcal{L}_j^{(3)}$ with overwhelming probability. The running time of the construction of each list $\mathcal{L}_j^{(3)}$ is given by $\max(L^{(3)}, B_{3l})$ where $B_{3l} = \binom{n/2}{n'''_1/2, n'''_{-1}/2, n'''_0/2}$ is the size of the lists that have to be joined in a Schroeppel-Shamir algorithm. Assuming that the sums $\mathbf{a} \cdot x^{(j)}$ for the elements $x^{(j)}$ in the lists are random w.r.t the corresponding moduli $M_3 \approx N_3, M_3M_2 \approx N_2$ and $M_3M_2M_1 \approx N_1$, the expected sizes of the lists are:

$$L^{(3)} \approx \frac{\binom{n}{n'''_1, n'''_{-1}, n'''_0}}{N_3}, \quad L^{(2)} \approx \frac{\binom{n}{n''_1, n''_{-1}, n''_0}}{N_2}, \quad L^{(1)} \approx \frac{\binom{n}{n'_1, n'_{-1}, n'_0}}{N_1} \ .$$

These are only upper bounds since we ignore the fact that we discard solutions that cannot be decomposed with the modular constraints of the lower level. The number of collisions, we have to go through, in order to build the lists $\mathcal{L}_j^{(2)}$ and $\mathcal{L}_j^{(1)}$ can be estimated as

$$C_3 \approx (L^{(3)})^2/(N_2/N_3) \quad \text{and} \quad C_2 \approx (L^{(2)})^2/(N_1/N_2) \ ,$$

respectively. The solution can then be found with an additional number of collisions

$$C_1 \approx (L^{(1)})^2/(2^n/N_1) \ .$$

Complexity for three levels. The overall running time of the algorithm is the maximum of the individual costs and the construction of the eight lists, which gives:

$$\tilde{\mathcal{O}}\left(\max(B_{3l}, \max_j |\mathcal{L}_j^{(3)}|, C_3, \max_j |\mathcal{L}_j^{(2)}|, C_2, \max_j |\mathcal{L}_j^{(1)}|, C_1)\right) \ .$$

Assuming that each list has a size close to its expected value, the expected running time is:

$$\mathcal{T}_{\text{3levels-x}} = \tilde{\mathcal{O}}\left(\max(B_{3l}, L^{(3)}, \frac{(L^{(3)})^2}{M_2}, L^{(2)}, \frac{(L^{(2)})^2}{M_1}, L^{(1)}, (L^{(1)})^2 \cdot \frac{M_1 \cdot M_2 \cdot M_3}{2^n})\right) \ . \tag{5.7}$$

Since not all collisions need to be stored, the amount of memory required is:

$$\mathcal{M}_{\text{3levels-x}} = \tilde{\mathcal{O}}\left(\max(B_{3l}, L^{(3)}, L^{(2)}, L^{(1)})\right) \ . \tag{5.8}$$

Using Shamir-Schroeppel for the first step does reduce the first term in (5.8) to $\sqrt{B_{3l}}$. As the cost do not play an important role, we neglect this improvement in an asymptotic study. The asymptotic complexity depends on the optimization parameters (α, β, γ) that determine the optimal number of -1 coefficients per level. Section 5.1.1 presents numerical results which show that the used memory can be decreased by adding a fourth level.

Complexity for four levels. We add a forth level of decomposition in the same way as done for three levels. A new parameter δ manages the proportion of -1 coefficients in the forth level that creates vectors of weight $(n_1^{''''}, n_{-1}^{'''})$ where $n_1^{''''} = n_1^{'''}/2 + \delta n$ and $n_{-1}^{''''} = n_{-1}^{'''}/2 + \delta n$. The value $B_{4l} = \binom{n/2}{n_1^{''''}/2, n_{-1}^{''''}/2, n_0^{''''}/2}$ counts the elements in the bottom lists from which we create the lists $\mathcal{L}_j^{(4)}$ for $j = 1, .., 16$. The elements fulfil a modular constraint modulo M_4 of size of the number of representations N_4 where

$$N_4 \approx \binom{n_1^{'''}}{n_1^{'''}/2} \cdot \binom{n_{-1}^{'''}}{n_{-1}^{'''}/2} \cdot \binom{n_0^{'''}}{\delta n, \delta n, \ \star} \ .$$

We expect that $|\mathcal{L}_j^{(4)}|$ is of size $L^{(4)} \approx \frac{\binom{n}{n_1^{''''}, n_{-1}^{''''}, n_0^{''''}}}{N_4}$ for all j and estimate the number of collisions between the forth and third level as $C_4 \approx (L^{(4)})^2/(N_3/N_4)$. The overall running time of a four-level algorithm is:

$$\tilde{\mathcal{O}}\left(\max(B_{4l}, \max_j |\mathcal{L}_j^{(4)}|, C_4, \ \max_j |\mathcal{L}_j^{(3)}|, C_3, \max_j |\mathcal{L}_j^{(2)}|, C_2, \max_j |\mathcal{L}_j^{(1)}|, C_1)\right) \ .$$

The expected running time is:

$$\mathcal{T}_{\text{4levels-x}} = \tilde{\mathcal{O}}\left((B_{4l}, L^{(4)}, \frac{(L^{(4)})^2}{M_3}, L^{(3)}, \frac{(L^{(3)})^2}{M_2}, L^{(2)}, \frac{(L^{(2)})^2}{M_1}, L^{(1)}, (L^{(1)})^2 \cdot \frac{M_1 \cdot M_2 \cdot M_3 \cdot M_4}{2^n}) \right) . \tag{5.9}$$

The amount of memory required is:

$$\mathcal{M}_{\text{4levels-x}} = \tilde{\mathcal{O}}\left(\max(B_{4l}, L^{(4)}, L^{(3)}, L^{(2)}, L^{(1)}) \right) . \tag{5.10}$$

As observed for three levels, a Shamir-Schroeppel algorithm does need to store only lists of size $\sqrt{B_{4l}}$ thus reducing the first term in the above memory cost. The term B_{4l} is not relevant in the asymptotic case which is why we consider a birthday paradox split with lists of size B_{4l}. The next section provides a numerical evaluation of the complexity for a three- and four-level algorithm.

5.1.1. Parameters for a minimal running time

In this section, we analyse the asymptotic cost of the previously presented algorithms that use the extended representation technique. We hence allow a small number of negative coefficients for the partial solutions and obtain a smaller expected running time compared to a birthday paradox algorithm or the simple representation technique.

Choosing $\alpha = \beta = \gamma = 0$ in (5.7) and (5.8), we recover the simple representation technique of section 4.2.1 of minimal time complexity $\tilde{\mathcal{O}}\left(2^{0.377n}\right)$ using memory $\tilde{\mathcal{O}}\left(2^{0.311n}\right)$ which we care to ameliorate. Optimal parameters (α, β, γ) for our algorithms, based on the extended representation technique, aim to minimize the time. We have already seen for the simple case that increased constraints on the elements can reduce the memory requirement additionally. Previous analysis [BCJ11] did only optimize the time component while we perform a more extensive investigation. We discover that adding a fourth level does lead to an algorithm of same running time and reduced space $\tilde{\mathcal{O}}\left(2^{0.289n}\right)$ already.It is however more effective to increase the size of the moduli at intermediate levels over the bound of the number of representations as we have proposed in section 4.2.2. In this way, we derive a minimal space requirement of $\tilde{\mathcal{O}}\left(2^{0.279\,n}\right)$ and same running time for the three-level algorithm.

Asymptotic approximations. In order to find parameters $\alpha, \beta, \gamma, \delta$ that minimize the time and memory, we approximate the binomials that appear in the terms of the individual cost. We summarize the usefull results of section A.2. For large n:

$$\log_2 \binom{n}{k} \approx n \cdot h(k/n)$$

where $h(x) = -x\log_2(x) - (1-x)\log_2(1-x)$ is the binary entropy function of section A.1. Also

$$\begin{array}{rcl} \log_2 \binom{un}{sn, sn, (1-2s)n} & \approx & n \cdot a(u,s) \text{ and} \\ \log_2 \binom{n}{sn, tn, (1-s-t)n} & \approx & n \cdot \tilde{a}(s,t) \end{array}$$

where we define

$$\begin{array}{rcl} a(u,s) & = & u\log_2(u) - 2s\log_2(s) - (1-2s)\log_2(1-2s) \text{ and} \\ \tilde{a}(s,t) & = & -s\log_2(s) - t\log_2(t) - (1-s-t)\log_2(1-s-t) \ . \end{array}$$

Using these asymptotic approximations, we can express all cost in terms of $\alpha, \beta, \gamma, \delta, \ell/n$ and n.

The number of representations in level i, N_i, can be approximated as $2^{n_1+n_{-1}+n \cdot a(n_0/n, \chi)}$ where n_1, n_{-1} and n_0 denote the numbers of 1s, 0s and -1s in the previous level and χ is the proportion of additional -1s for level i. We obtain

$$\begin{array}{rcllll} \log_2(N_4) & \approx & n \cdot (\frac{\ell/n}{8} + \alpha/2 + \beta + 2\gamma & + & a(1 - \frac{\ell/n}{8} - \alpha/2 - \beta - 2\gamma, \delta)) \ , \\ \log_2(N_3) & \approx & n \cdot (\frac{\ell/n}{4} + \alpha + 2\beta & + & a(1 - \frac{\ell/n}{4} - \alpha - 2\beta, \gamma)) \ , \\ \log_2(N_2) & \approx & n \cdot (\frac{\ell/n}{2} + 2\alpha & + & a(1 - \frac{\ell/n}{2} - 2\alpha, \beta)) \text{ and} \\ \log_2(N_1) & \approx & n \cdot (\ell/n & + & a(1 - \ell/n, \alpha)) \ . \end{array} \tag{5.11}$$

The size of the lists at level i, $L^{(i)}$, are equal to $2^{n \cdot \tilde{a}(n_1, n_{-1})}/N_i$ in the asymptotic case.

$$\begin{array}{rclll} \log_2(L^{(4)}) & \approx & n \cdot \tilde{a}(\frac{\ell/n}{16} + \alpha/8 + \beta/4 + \gamma/2 + \delta, \alpha/8 + \beta/4 + \gamma/2 + \delta) & - & \log_2(N_4) \ , \\ \log_2(L^{(3)}) & \approx & n \cdot \tilde{a}(\frac{\ell/n}{8} + \alpha/4 + \beta/2 + \gamma, \alpha/4 + \beta/2 + \gamma) & - & \log_2(N_3) \ , \\ \log_2(L^{(2)}) & \approx & n \cdot \tilde{a}(\frac{\ell/n}{4} + \alpha/2 + \beta, \alpha/2 + \beta) & - & \log_2(N_2), \\ \log_2(L^{(1)}) & \approx & n \cdot \tilde{a}(\frac{\ell/n}{2} + \alpha, \alpha) & - & \log_2(N_1) \ . \end{array}$$

At the bottom level, we count either B_{3l} or B_{4l} elements where

$$\begin{array}{rcl} \log_2(B_{4l}) & \approx & 0.5 \cdot n \cdot \tilde{a}(\frac{\ell/n}{16} + \alpha/8 + \beta/4 + \gamma/2 + \delta, \alpha/8 + \beta/4 + \gamma/2 + \delta) \text{ or} \\ \log_2(B_{3l}) & \approx & 0.5 \cdot n \cdot \tilde{a}(\frac{\ell/n}{8} + \alpha/4 + \beta/2 + \gamma, \alpha/4 + \beta/2 + \gamma) \ . \end{array}$$

The expected number of collisions becomes

$$\begin{array}{rclllll} \log_2(C_4) & \approx & 2 \cdot \log_2(L^{(4)}) & - & \log_2(N_3) & + & \log(N_4) \ , \\ \log_2(C_3) & \approx & 2 \cdot \log_2(L^{(3)}) & - & \log_2(N_2) & + & \log(N_3) \ , \\ \log_2(C_2) & \approx & 2 \cdot \log_2(L^{(2)}) & - & \log_2(N_1) & + & \log(N_2) \text{ and} \\ \log_2(C_1) & \approx & 2 \cdot \log_2(L^{(1)}) & - & 1 & + & \log(N_1) \ . \end{array}$$

Conditions for optimisation. We assume that $\ell/n \leq 0.5$ and search real values $\alpha, \beta, \gamma, \delta$ in the interval $[0, 0.25]$ under the natural condition that all cost shall be positive,

$$0 < \log_2 L^{(i)} \leq 1, \quad 0 \leq \log_2 C_i \leq 1, \text{ and } 0 < \log_2 B_{3l}, \log_2 B_{4l} \leq 1 \ ,$$

the intermediate knapsacks are random ($\log_2 N_1 \leq 1$) and the weight of the vectors, the number of non-zero coordinates, does never exceed their length:

$$\begin{array}{lcl} \ell/2 + 2\alpha n & \leq & n \ , \\ \ell/4 + \alpha n + 2\beta n & \leq & n \ , \\ \ell/8 + \alpha n/2 + \beta n + 2\gamma n & \leq & n \ \text{ and} \\ \ell/16 + \alpha n/4 + \beta n/2 + \gamma n + 2\delta n & \leq & n \ . \end{array}$$

The size of the applying modular constraint at the levels are of size of the number of representations except when noted otherwise. For variable size of moduli, we ensure that the product of the moduli is smaller than the size of the numbers in the initial knapsack.

Minimize running time for three levels

Under the above presented conditions, we can now find optimal parameters[1] that minimize the running time depending on the weight of the solution vector ℓ. Figure 5.3 shows the asymptotic running time $\mathcal{T}_{\text{3level-x}} = 2^{t(\ell/n)\,n+o(n)}$. The factor $t(\ell/n)$ in the exponent is considerably smaller in comparison to a simple representation technique (section 4.2).

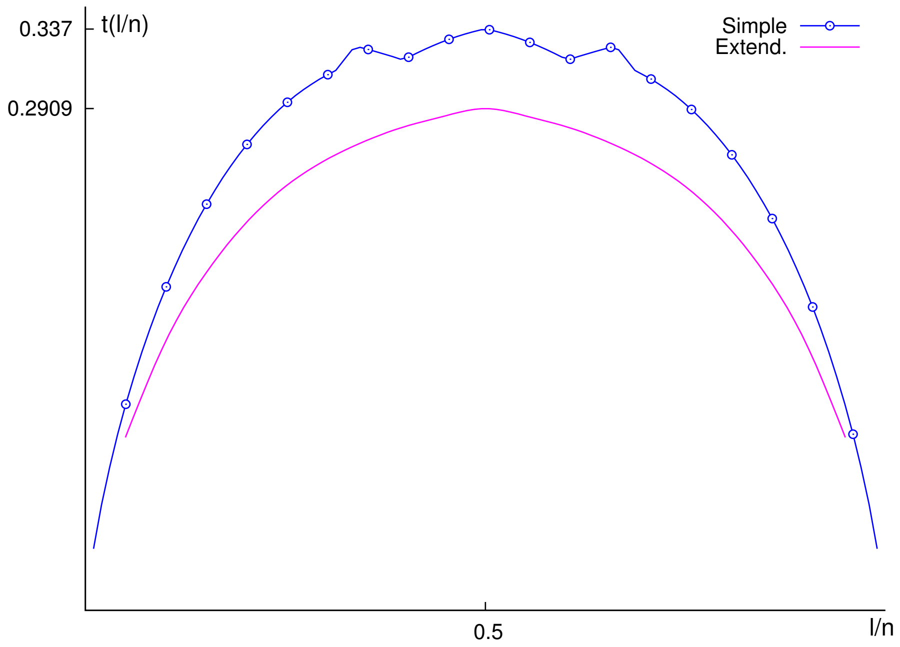


Figure 5.3.: *Compare asymptotic running time of basic and extended representation technique for variable weight ℓ of the solution.*

[1]Mathematica or Octave 3.2.4, function nelder_mead_min, http://www.octave.org

We give more details for the equibalanced case, i.e., when $\ell/n \approx 0.5$. The parameters

$$\alpha = 0.0267, \quad \beta = 0.0168, \quad \gamma = 0.0029 \ ,$$

minimize[2] the running time $\mathcal{T}_{\text{3levels-x}}(\alpha, \beta, \gamma)$ for which we obtain the following individual cost and size of moduli:

$$\begin{array}{lllllllll} B_{3l} & \approx & 2^{0.266\,n}, & L^{(3)} & \approx & 2^{0.291\,n}, & L^{(2)} & \approx & 2^{0.279\,n}, \\ L^{(1)} & \approx & 2^{0.217\,n}, & M_3 & \approx & 2^{0.241\,n}, & M_2 & \approx & 2^{0.291\,n}, \\ M_1 & \approx & 2^{0.267\,n}, & C_3 & \approx & 2^{0.291\,n}, & C_2 & \approx & 2^{0.291\,n}, \\ C_1 & \approx & 2^{0.233\,n} & & & & & & \end{array} \tag{5.12}$$

for which $M_3 \approx N_3$, $M_2M_3 \approx N_2$ and $M_3M_2M_1 \approx N_1$. As a consequence, we find that both the time and memory complexity are equal to $\tilde{\mathcal{O}}\left(2^{0.291\,n}\right)$. However, we remark that γ is so small that for any achievable knapsack size n, the number of -1s added at the last level is 0 in practice. Thus, in order to improve the practical choices of the number of -1s at the higher levels, it is better to adjust the minimization with the added constraint $\gamma = 0$. This leads to the alternative values:

$$\alpha = 0.0194, \quad \beta = 0.0119, \quad \gamma = 0 \ .$$

With these values, we obtain:

$$\begin{array}{llllllllllll} B_{3l} & \approx & 2^{0.231\,n}, & L^{(3)} & \approx & 2^{0.295\,n}, & L^{(2)} & \approx & 2^{0.284\,n}, & L^{(1)} & \approx & 2^{0.234\,n}, \\ M_3 & \approx & 2^{0.168\,n}, & M_2 & \approx & 2^{0.295\,n}, & M_1 & \approx & 2^{0.272\,n}, & & & \\ C_3 & \approx & 2^{0.295\,n}, & C_2 & \approx & 2^{0.295\,n}, & C_1 & \approx & 2^{0.204\,n} & & & \end{array}$$

leading to an asymptotic time and space of $\tilde{\mathcal{O}}\left(2^{0.295\,n}\right)$. If we only care for the running time and assume that enough memory is provided, we have found optimal parameter sets as we presented in [BCJ11]. The expected number of collisions between the second and first level, C_2, does dominate the time and is minimal for the above parameters. A closer look at the individual cost reveals that the space requirement is dominated by the lists at the third level. A fourth application of the representation technique can reduce the size of these lists. An alternative is to allow larger moduli in the levels which reduces the number of expected collisions. We present optimal parameters for both ideas in the following.

Minimize running time for four-levels

We assume that $\ell/n \approx 0.5$. The minimal running time $\mathcal{T}_{\text{4level-x}}(\alpha, \beta, \gamma, \delta)$ for a four level algorithm occurs for parameters

$$\alpha = 0.0257, \quad \beta = 0.0162, \quad \gamma = 0.0028, \quad \delta = 3.974E - 04 \ .$$

[2] Octave 3.2.4, function nelder_mead_min, http://www.octave.org

The minimal running time remains as before of order $\tilde{\mathcal{O}}\left(2^{0.291\,n}\right)$ given by C_2. The needed space drops to $L^{(3)} = \tilde{\mathcal{O}}\left(2^{0.289\,n}\right)$. The different terms that contribute to the expected complexity are:

$$\begin{array}{lllllllll} B_{4l} & \approx & 2^{0.159\,n}, & L^{(4)} & \approx & 2^{0.210\,n}, & L^{(3)} & \approx & 2^{0.289\,n}, \\ L^{(2)} & \approx & 2^{0.279\,n}, & L^{(1)} & \approx & 2^{0.219\,n}, & M_4 & \approx & 2^{0.107\,n}, \\ M_3 & \approx & 2^{0.130\,n}, & M_2 & \approx & 2^{0.286\,n}, & M_1 & \approx & 2^{0.268\,n}, \\ C_4 & \approx & 2^{0.291\,n}, & C_3 & \approx & 2^{0.291\,n}, & C_2 & \approx & 2^{0.291\,n} \\ C_1 & \approx & 2^{0.229\,n} \ . \end{array} \tag{5.13}$$

As γ and δ are quite small, we can also compute the expected complexity by setting them to zero. We obtain a minimal running time $\tilde{\mathcal{O}}\left(2^{0.295\,n}\right)$ with space requirement $\tilde{\mathcal{O}}\left(2^{0.292\,n}\right)$ for

$$\alpha = 0.0236, \quad \beta = 0.284, \quad \gamma = 0, \quad \delta = 0 \ .$$

The individual cost are:

$$\begin{array}{llllllllllll} B_{4l} & \approx & 2^{0.136\,n}, & L^{(4)} & \approx & 2^{0.189\,n}, & L^{(3)} & \approx & 2^{0.292\,n}, & L^{(2)} & \approx & 2^{0.284\,n}, \\ L^{(1)} & \approx & 2^{0.236\,n}, & M_4 & \approx & 2^{0.083\,n}, & M_3 & \approx & 2^{0.0833\,n}, & M_2 & \approx & 2^{0.290\,n}, \\ M_1 & \approx & 2^{0.273\,n}, & C_4 & \approx & 2^{0.295\,n}, & C_3 & \approx & 2^{0.295\,n}, & C_2 & \approx & 2^{0.295\,n} \\ C_1 & \approx & 2^{0.201\,n} \ . \end{array}$$

The memory needs can not further be reduced if we keep the moduli of size of the number of decompositions.

5.2. Improvement on the memory requirement

Let $\ell/n \approx 0.5$. The optimal setting for three levels, presented in section 5.1.1, shows that the memory requirement varies at the different levels and is high at levels two and three. We can augment the moduli as we proposed in section 4.2.2 to reduce the memory without an increase in the overall time. Per level i where we wish to reduce the size of the lists, we apply an additional modular constraint w.r.t. an integer m_i. We choose m_i co-prime to the previous moduli and change the target at each iteration until all values in $\mathbb{Z}/m_i\mathbb{Z}$ were chosen.

We will have a closer look at the subtask to create the elements at the first and second level that are obtained by joining the large lists of the levels below.

In order to find $L^{(1)}$ elements for each list in the first level, we will create them in blocks of expected size $L'^{(1)} = L^{(1)}/\Lambda$ for a real parameter $\Lambda \geq 1$. Our goal is of course to reduce the memory requirement needed to create all $L^{(1)}$ elements. Heuristically, we assume that the elements in the lists that we collide are random values. If we increase the modular constrains M_3M_2 of the second level by a factor Λ, such that the modulus becomes $m_2M_3M_2 \approx \Lambda \cdot N_3$, we expect to observe C_2/Λ collisions between lists of size $L'^{(2)} = L^{(2)}/\Lambda$. The expected number of elements for the first level within the C_2/Λ colliding elements is $L'^{(1)}$ as we required. If we repeat the steps Λ times for all possible targets in $\mathbb{Z}_{m_2}$, we create altogether $\Lambda \cdot L'^{(1)} = L^{(1)}$ elements per list.

We will now specify constraints on the parameter Λ. The minimal memory requirement is given by the size of the bottom lists $\mathcal{B}_{3l}$. If we increase the moduli, we can hope to decrease the lists to about the same size at best. Let $\mathcal{T}_i$ be the time needed by the lower levels to create lists at level i. We need to ensure that the time $\mathcal{T}_1$ to create all $L^{(1)}$ elements per list at level two stays unchanged. We can write $\mathcal{T}_1 = \max(L^{(1)}, C_1, \mathcal{T}_2)$ where

$$\mathcal{T}_2 \quad = \quad \Lambda \cdot \max(\tfrac{C_2}{\Lambda}, \tfrac{L^{(2)}}{\Lambda}, \mathcal{T}_3)$$

The used memory is so far given by $B_{3l}, L^{(1)}, L'^{(2)}$ and $L^{(3)}$ which is dominated by the lists at the third level.

We can apply the same idea to create all $L^{(2)}$ elements in blocks of expected size $L'^{(2)} = L^{(2)}/\Delta$ for real $\Delta \geq 1$. We increase the modular constraints in the third level by the factor Δ to $m_3M_3 \approx \Delta \cdot N_3$ thus reducing the lists to $L'^{(3)} = L^{(3)}/\Delta$ elements per run. The number of colliding elements in a merge-join is reduced by the same factor. After Δ repetitions, we have created $\Delta \cdot L'^{(2)} = L^{(2)}$ elements per list in level two. The time can be estimated as

$$\mathcal{T}_3 = \Delta \cdot \max(B_{3l}, \tfrac{C_3}{\Delta}, \tfrac{L^{(3)}}{\Delta}) \ \ .$$

As we would like to keep the overall running time at $\mathcal{T}_{\text{3levels-x}}$, we require that

$$\mathcal{T}_1 = \max(C_1, L^{(1)}, L^{(2)}, C_2, \Lambda \cdot L^{(3)}, \Lambda \cdot C_3, \Delta \cdot \Lambda B_{3l})$$

equals

$$\mathcal{T}_{\text{3levels-x}} = \tilde{\mathcal{O}}\left(C_3\right) \ \ .$$

For $\Lambda > 1$, the time would increase such that we leave the constraints at the second level at size N_2. It rests to ensure that the time to read the bottom lists, $\Delta \cdot B_{3l}$, stays in the limit given by C_3. The number of collisions at the third level, C_3, is of expected size $\frac{B_{3l}^2}{N_3}$. From the condition $B_{3l} \leq C_3$, we derive that $M_3 \approx \Delta \cdot N_3 \leq B_{3l}$. Setting $M_3 \approx B_{3l}$, leaves the maximal size of list as:

$$\max(B_{3l} \approx L^{(3)}, L^{(1)}, L^{(2)}) = L^{(2)} = \tilde{\mathcal{O}}\left(2^{0.279\,n}\right) \ \ .$$

For slightly increased modulo $m_3M_3 = \mathcal{O}\left(2^{0.266\,n}\right)$ and about $\Delta = \mathcal{O}\left(2^{0.025\,n}\right)$ additional repetitions in the third level, we obtain an algorithm of running time $\mathcal{T}_{\text{3levels-x}} = \tilde{\mathcal{O}}\left(2^{0.291\,n}\right)$ and memory $\mathcal{M}_{\text{3levels-x}}/\Delta = \tilde{\mathcal{O}}\left(2^{0.279\,n}\right)$.

Larger modular constraint for a four-level algorithm. We care to reduce the memory at level three. By the same reasoning than above, we construct elements of level two in blocks of size $L^{(2)}/\Lambda$. To this end, we choose a modulo $m_3M_3 \approx \lambda N_3$ under the condition that the cost to create all $L^{(2)}$ elements,

$$\mathcal{T}_2 = \max(\mathcal{T}_3, L^{(2)}, C_2) \ \ ,$$

in the second levels does not increase the overall running time $\mathcal{T}_{\text{4levels-x}} = \tilde{\mathcal{O}}\,(C_4)$. The term $\mathcal{T}_3$ denotes the time to create the lists in the third level. More precisely, we require that

$$\max(\mathcal{T}_3, C_2) \approx C_4$$

where

$$\mathcal{T}_3 \;=\; \Lambda \cdot \max(\mathcal{T}_4, \tfrac{L^{(3)}}{\Lambda}, \tfrac{C_3}{\Lambda}) = \max(\Lambda \cdot \mathcal{T}_4, L^{(3)}, C_3)$$

and

$$\Lambda \cdot \mathcal{T}_4 \;=\; \max(\Lambda \cdot B_{4l}, \Lambda \cdot L^{(4)}, \Lambda \cdot C_4) = \Lambda \cdot C_4 \ .$$

Observe that we have an increase in the running time for $\Lambda > 1$ due to term $\Lambda \cdot C_4$. We conclude that the memory can not be decreased in this way.

5.3. Implementation and experimental evidence

In order to verify the correctness and practicability of the techniques and heuristics presented in the previous section, we have implemented a three-level extended-representation-technique algorithm. We ran our implementation on 50 random knapsacks containing 80 elements on 80 bits. The target sum was constructed in each case as a sum of 40 knapsack elements. For each of these knapsacks, we ran our implementation several times, choosing new random modular constraints for each execution, until a solution was found. As shown in figure 5.4, we constructed the solution vector by vectors containing two -1s at the first, one -1 at the second and none at the third level. We also collected statistical information such as the real size of intermediate lists and the number of collisions.

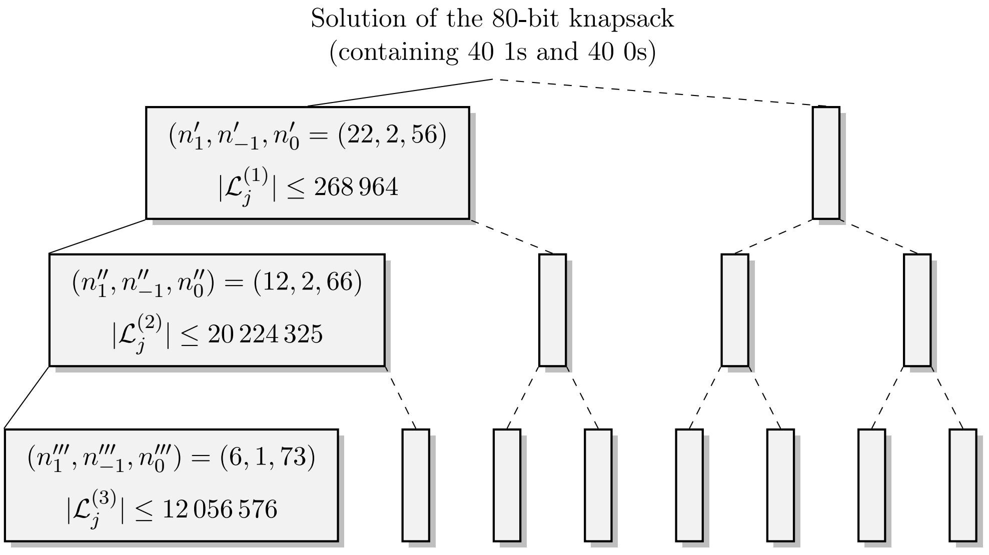

Solution of the 80-bit knapsack
(containing 40 1s and 40 0s)
$(n'_1, n'_{-1}, n'_0 = (22, 2, 56)$
$|\mathcal{L}_j^{(1)}| \leq 268\,964$
$(n''_1, n''_{-1}, n''_0) = (12, 2, 66)$
$|\mathcal{L}_j^{(2)}| \leq 20\,224\,325$
$(n'''_1, n'''_{-1}, n'''_0) = (6, 1, 73)$
$|\mathcal{L}_j^{(3)}| \leq 12\,056\,576$

Figure 5.4.: *Decomposition of a single solution for an equibalanced knapsack of size 80. The decomposition into 0s,1, and -1s is the same within each level.*

The total running time to solve the 50 knapsacks was 14 hours and 50 minutes on a Intel® Core™ i7 CPU M 620 at 2.67GHz. The total number of repetitions of the algorithm for the complete test case was equal to 280. We observed that a maximum of 16 repetitions (choice of a different random value in level one) was sufficient to find the solution. Also, 53% of the 50 random knapsacks needed only up to 4 repetitions. On average, each knapsack required 5.6 repetitions. The complete distribution of the number of repetitions is presented in table 5.1.

Table 5.1.: *Number of repetitions for 50 random knapsacks until a solution was found.*

Number of repetitions	Number of corresponding knapsacks	Number of repetitions	Number of corresponding knapsacks
1	8	2	6
3	9	4	4
5	2	6	5
7	1	8	1
9	1	10	5
11	4	12	1
13	0	14	1
15	0	16	2

The moduli were chosen as primes of size as discussed in section 5.1: $M_3 = 1\,847$, $M_2 = 2\,353\,689$, and $M_1 = 17\,394\,593$. The experimental and theoretical sizes of the lists and the number of collisions are given in table 5.2.

Table 5.2.: *Experimental versus estimated sizes of the intermediate lists and number of collisions.*

List type	Min. size	Max. size	Theoretical estimate
$\lvert\mathcal{L}_j^{(3)}\rvert$	12 024 816	12 056 576	$L^{(3)} = \frac{\binom{80}{6,1,73}}{1\,847} \approx 12\,039\,532$
C_3	61 487 864	61 725 556	$\frac{(L^{(3)})^2}{2\,352\,689} \approx 61\,610\,489$
$\lvert\mathcal{L}_j^{(2)}\rvert$	12 473 460	20 224 325	$L^{(2)} = \frac{\binom{80}{12,2,66}}{1\,847 \cdot 2\,352\,689} \approx 31\,583\,129$
C_2	14 409 247	23 453 644	$\frac{(L^{(2)})^2}{17\,394\,593} \approx 57\,345\,064$
$\lvert\mathcal{L}_j^{(1)}\rvert$	183 447	268 964	$L^{(1)} = \frac{\binom{80}{22,2,56}}{1\,847 \cdot 2\,352\,689 \cdot 17\,394\,593} \approx 592\,402$
C_1	178	1 090	$\frac{L_\nu^2 \cdot 1\,847 \cdot 2\,352\,689 \cdot 17\,394\,593}{2^{80}} \approx 21\,942$

The number of elements that were created in level i are $\lvert\mathcal{L}_j^{(i)}\rvert$ and C_i denoting the stored elements and the number of collisions, respectively. The value for $\lvert\mathcal{L}_j^{(i)}\rvert$ is the mean between all lists of the same level. If we compare the estimated and actual number of elements for the third level, we see that $|\mathcal{L}_j^{(3)}|$ and C_3 are very close to the predicted values and do not vary a lot within the different trials. The estimation for $|\mathcal{L}_j^{(2)}|$ and $|\mathcal{L}_j^{(1)}|$ ignores the loss of solutions which are incompatible to the modular constraints of the lower levels. The actual sizes of the lists is therefore smaller than the predicted one. The effect is forwarded from level two to level one resulting in an even bigger gap between theory and practice for $|\mathcal{L}_j^{(1)}|$ and C_1. The variable C_1 counts joined elements that have to be tested for a correct weight and equality to the target value over the integers. We recall that our theoretical estimate upper bounds the size as it counts collisions modulo $M_3 \cdot M_2 \cdot M_1$, a number close to 2^{80}, and neglects the conditions of the lower levels.

Experimental confirmation of the assumption of independence.

We also performed additional tests on 240 random knapsacks where we repeated the search for a solution 10 times per knapsack. Figure 5.5 shows the distribution of necessary repetitions until the solution was found. We observe an average of $\mu = 5.47$ and a minimum of 41 repetitions. In 95% of the cases less than 16 repetitions were enough to find the solution. Furthermore, the results seem to be conform with a random variable following the geometric distribution of expected value μ where we assume independence for each decomposition and level and the same probability of success $1/\mu$. Figure 5.5 also depicts the probability distribution of the random variable. None of the tested random knapsacks was distinctly easier or more difficult to solve than the others within the 10 runs.

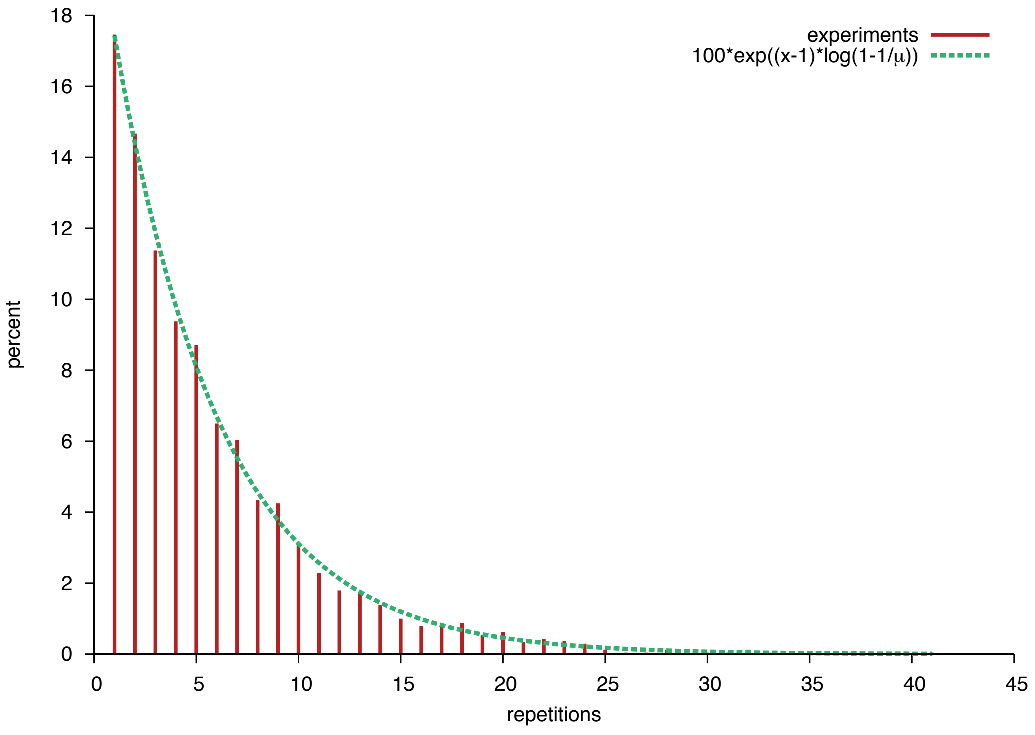


Figure 5.5.: *Percentage of 240 random knapsacks, each run 10 times, per number of repetitions (red bar); Random variable of geometric distribution in values x, probabilities $p_x = 100 \cdot \exp((x-1) \cdot \log(1 - 1/\mu))$, where μ is the average number of repetitions (green - -)*

The sizes of the intermediate lists $|\mathcal{L}_j^{(3)}|$, $|\mathcal{L}_j^{(2)}|$ and $|\mathcal{L}_j^{(1)}|$ are given in table 5.3 and compared to the estimate. We present the minimal and maximal sizes as well as the mean and the sample standard deviation for each of the lists. The average running time per found solution is 3.05 minutes per repetition and 17.53 minutes to find the solution on an Intel® Xeon™ CPU X5560 at 2.80GHz.

Table 5.3.: *Experimental sizes of the intermediate lists for 240 random knapsacks.*

List type	Min. size	Max. size	Mean	Sample standard deviation	Theoretical estimate
$\lvert\mathcal{L}_j^{(3)}\rvert$	12 009 444	12 068 959	12 039 526	3 391	12 039 532
$\lvert\mathcal{L}_j^{(2)}\rvert$	12 231 570	20 233 425	19 924 351	202 256	31 583 129
$\lvert\mathcal{L}_j^{(1)}\rvert$	177 662	269 786	263 337	3 437	592 402

Experimental behavior of decompositions.

It is useful to determine the probability of success of each decomposition. Three levels of decomposition occur. At the top level, a balanced golden solution with 40 zeros and 40 ones needs to be split into two partial solutions with 22 ones and two -1s each. At the middle level, a golden solution with 22 ones and two -1s is to be split into two partial solutions with 12 ones and two -1s. Finally, at the bottom level, we split 12 ones and two -1s into twice 6 ones and one -1.

At the top level, the number of possible decompositions of a golden solution is larger than $\binom{40}{22}\binom{40}{2,2,36} \approx 2^{56}$. As a consequence, it is not possible to perform experimental statistics of the modular values of such a large set. At the middle level, the number of decompositions is larger than $\binom{22}{11}\binom{2}{1}\binom{46}{1,1,44} \approx 2^{32}$. Thus, it is possible to perform some experiments, but doing a large number of tests to perform a statistical analysis of the modular values is very cumbersome. At the bottom level, the number of decompositions of a golden solution is $\binom{12}{6}\binom{2}{1} = 1848$. This is small enough to perform significant statistics and, in particular, to study the fraction of modular values which are not obtained (depending on a random choice of 14 knapsack elements, 12 1s and two -1s, to be split). The value of the modulus used in this experiment is 1847, the closest prime to 1848.

During our experimental study, we created one million modular subknapsacks from 14 randomly selected values modulo 1847. Among these values 12 elements correspond to additions and 2 to subtractions. From this set we computed (in $\mathbb{Z}_{1847}$) all of the 1848 values that can be obtained by summing 6 out of the 12 addition elements and subtracting one of the subtraction elements. In each experiment, we counted the number of values which were not obtained; the results are presented in figure 5.6. On the vertical axis we display the cumulative number of knapsacks which result in x or less skipped values. To allow comparison with the purely random model, we display the same curve computed for one million of experiments where 1848 random numbers modulo 1847 are chosen. In particular, we see on this graph that for 99.99% of the random knapsacks we have constructed the fraction of unobtained value stays below 2/3. This means that experimentally, the probability of success of a decomposition at the bottom level is, at least, 1/3 for a very large fraction of knapsacks. Assuming independence between the probability of success of the seven splits and a similar behavior of three levels[3], we conclude that for 99.93% of random knapsacks an average number of $3^7 = 2187$ repetitions suffices to solve the initial problem.

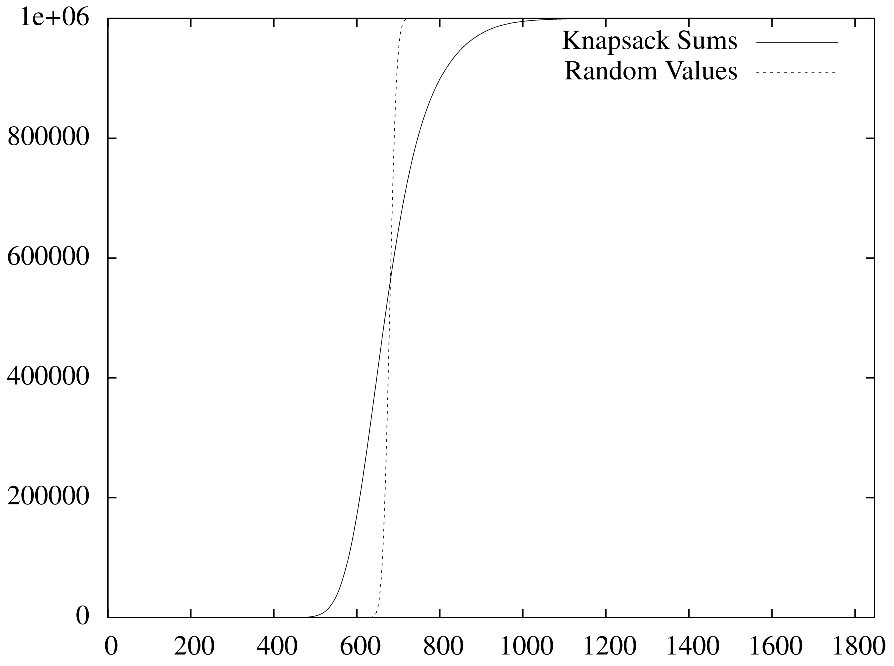


Figure 5.6.: *The horizontal axis represents all values $M \in \mathbb{Z}_{1847}$. The vertical axis counts at a position p the cumulative number of knapsacks* $\mathbf{a}$ *(in a million) for which at most p values in $\mathbb{Z}_{1847}$ can not be obtained as* $\mathbf{a} \cdot \mathbf{x} \mod 1847$ *where* $wt(\mathbf{x}) = (1s, -1s) = (12, 6)$.

[3]The limited number of experiments we have performed for the middle level seem to indicate a comparable behavior. We performed 100 experiments and the number of not obtained values remained in the range 42% – 43.1%.

Some Results with $n = 96$.

We also tested the algorithm on equibalanced 96-bit knapsacks. However, it was not possible to add the optimal number of -1s, because some of the lists required too much memory. Instead, we used the following suboptimal choices:

- Split the initial knapsack into two subknapsack with 25 ones and one -1.
- Split again into subknapsacks with 14 ones and two -1s.
- Finally split into subknapsacks with 7 ones and one -1.

The chosen moduli are $M_3 = 6\,863$, $M_2 = 248\,868\,793$ and $M_1 = 42\,589$. We tried 5 different knapsacks and solved all of them with an average number of repetitions equal to 7.8. The runtime for a single trial is 47 minutes on a Intel® Xeon$^{\text{TM}}$ CPU X5560 at 2.80GHz using 13 Gbytes of memory[4].

We can also compare the implementation to the practical implementation by Joux of the Howgrave-Graham–Joux algorithm. This variant takes an average of 15 minutes to solve a knapsack on 96 bits, using 1.6 Gbytes of memory. However, the program is much more optimized. Moreover, it contains heuristics to reuse the computations of intermediate lists many times, in order to run faster. The new algorithm can probably take practical profit of similar tricks. As a consequence, the running time on 96 bits are not so far from each other. We expect the cross-over point to occur around $n = 128$, which means that 96 bits is close to the cross-over point between the two algorithms.

5.4. Upper bounds on the size of the lists and number of collisions

Concerning the size of the lists that occur during the algorithm, both the simple heuristic model in section 5.1 and the experimental results presented in section 5.3 use the fact that the size of the lists are always very close to the theoretical average-case values. It remains to give an upper bound on the size of the various lists and number of collisions for knapsacks that lead to larger values.

Upper bound on the size of the lists. For the lists $\mathcal{L}_j^{(i)}$, we can use a direct application of theorem 4.1. The set $\mathcal{B}$ is the set of all vectors of length n with coefficients in $\{1, 0, -1\}$ fulfilling the weight conditions of level i. The modulus M is the product of all active moduli at the current and preceding levels. That is, for level three we have $M = M_3$, for level two, $M = M_3 \cdot M_2$ and for level one we take $M = M_3 \cdot M_2 \cdot M_1$. We compute the number of knapsacks H_λ which produce an overflow of some lists for certain targets which means that

[4]We would like to thank CEA/DAM (Commissariat à l'énergie atomique, Direction des applications militaires) for kindly providing the necessary computing time on its servers.

considerably more than $|\mathcal{B}|/M$ vectors are found. Let λ be an integer and consider the number H_λ of knapsacks for which more than $M/(2\,\lambda)$ values $R \in \mathbb{Z}_M$ have a probability that satisfies:

$$P_{a_1,\cdots,a_n}(\mathcal{B},R) \geq \lambda/M \ \ .$$

Due to theorem 4.1 and the above assumptions, we find that

$$\frac{M-1}{M|\mathcal{B}|}M^n \geq H_\lambda \sum_{R\in\mathbb{Z}_M}\left(P_{a_1,\cdots,a_n}(\mathcal{B},R)-\frac{1}{M}\right)^2 \geq H_\lambda \cdot \frac{M}{2\,\lambda}\cdot\frac{(\lambda-1)^2}{M^2} \ \ .$$

As a consequence, we can derive the upper bound:

$$H_\lambda \leq \frac{2\,\lambda}{(\lambda-1)^2}\cdot\frac{M}{|\mathcal{B}|}\cdot\frac{M-1}{M}\cdot M^n \leq \frac{2\,\lambda}{(\lambda-1)^2}\cdot M^n \ \ .$$

We can conclude that for a knapsack which is not one of the H_λ knapsacks above and for most targets (all but at most $M/(2\,\lambda)$), the size of the lists $\mathcal{L}_j^{(i)}$ is at most λ times the expected value $|\mathcal{B}|/M$:

$$L^{(i)} \leq \lambda\frac{|\mathcal{B}|}{M} \ \ .$$

Upper bound on number of collisions. To bound the number of collisions C_i, we proceed slightly differently. The set $\mathcal{B}$ consists of vectors that are joined and checked for consistency to the current constraint w.r.t. a modulus M. We decompose M: $M = M_1 \cdot M_2$ where M_1 is the product of the moduli of the lower levels. All vectors that are to be collided fulfil a constraint modulo M_1. We add a constraint modulo M_2 for the join. Let $\sigma \pmod{M}$ denote the target for collision. Let $\sigma_L \pmod{M_1}$ and $\sigma_R \pmod{M_1}$ denote the values of the sums in the left and right lists, respectively. By construction, we have $\sigma_L + \sigma_R \equiv \sigma \pmod{M_1}$. We can write:

$$\begin{aligned} C_i &= \sum_{\substack{c\,\in\,\mathbb{Z}_M\\ c\,\equiv\,\sigma \pmod{M_1}}} (|\mathcal{B}|\cdot P_{a_1,\cdots,a_n}(\mathcal{B},c))\cdot(|\mathcal{B}|\cdot P_{a_1,\cdots,a_n}(\mathcal{B},\sigma-c)) \\ &\leq \left[\sum_{\substack{c\,\in\,\mathbb{Z}_M\\ c\,\equiv\,\sigma_L}} (|\mathcal{B}|\cdot P_{a_1,\cdots,a_n}(\mathcal{B},c))^2 \times \sum_{\substack{c\,\in\,\mathbb{Z}_M\\ c\,\equiv\,\sigma_R}} (|\mathcal{B}|\cdot P_{a_1,\cdots,a_n}(\mathcal{B},c))^2\right]^{1/2} \ \ . \end{aligned} \tag{5.14}$$

Thus to estimate the number of collisions, we need to find an upper bound for the value of sums of the form:

$$\sum_{\substack{c\,\in\,\mathbb{Z}_M\\ c\,\equiv\,c_1 \pmod{M_1}}} P_{a_1,\cdots,a_n}(\mathcal{B},c)^2 \ \ .$$

To do this, it is useful to rewrite the relation from Theorem 4.1 as

$$\frac{1}{M^n} \sum_{(a_1,\cdots,a_n)\in\mathbb{Z}_M^n} \sum_{c\in\mathbb{Z}_M} P_{a_1,\cdots,a_n}(\mathcal{B},c)^2 = \frac{M+|\mathcal{B}|-1}{M|\mathcal{B}|} \quad .$$

Given λ, we let G_λ denote the number of knapsacks for which more than $M_1/(8\,\lambda)$ values c_1 have a sum of squared probabilities that satisfy:

$$\sum_{\substack{c\in\mathbb{Z}_M \\ c\equiv c_1 \pmod{M_1}}} P_{a_1,\cdots,a_n}(\mathcal{B},c)^2 \geq \frac{\lambda^2}{M_1^2 M_2} \quad .$$

We find that

$$\frac{G_\lambda}{M^n}\cdot\frac{M_1}{8\,\lambda}\cdot\frac{\lambda^2}{M_1^2 M_2} \leq \frac{M+|\mathcal{B}|-1}{M|\mathcal{B}|} \quad ;$$

and as a consequence

$$G_\lambda \leq \frac{8}{\lambda}\cdot\frac{M+|\mathcal{B}|}{|\mathcal{B}|}\cdot M^n \quad .$$

Moreover, we can check with our concrete algorithm that we always have $|\mathcal{B}| \geq M$ for the join. Thus we have

$$G_\lambda \leq (16/\lambda)M^n \quad .$$

For a knapsack which is not one of the G_λ knapsacks above and for most values (all but at most $2M_1/(8\,\lambda)$) of $\sigma_L \bmod M_1$, the number of collisions C_i is smaller than λ^2 times the expected value $|\mathcal{B}|^2/(M_1^2\, M_2)$:

$$C_i \leq \frac{\lambda^2\,|\mathcal{B}|^2}{M_1^2\cdot M_2} \quad .$$

5.5. Analysis of the probability of success

The proposed three-level algorithm of section 5.1 searches the unique golden solution to a knapsack problem by decomposing it seven times. Each of the decompositions is considered successful if the golden solution admits at least one representation which satisfies the modular constraint per level. Suppose that at least one representation of the solution satisfies the modular constraints at the first level. We then need to represent each of the partial solutions which means that at least one of their representations needs to satisfy the modular constraint at the next lower level etc. Clearly, if each of the seven decompositions succeeds, the initial solution can be found by the algorithm.

Heuristically, we can assume that the modular sum $\mathbf{a}\cdot\mathbf{x} \bmod M$ are random values distributed uniformly in $\mathbb{Z}_M$ as discussed in section 4.1. Assuming independence between the different decompositions, we can lower bound the overall probability by the product of the probability of success of the individual decompositions. It can be larger if multiple representations satisfy the constraints. The assumption of independence is a heuristic one based on experiments performed on random modular knapsacks such as presented in section 5.3. The modular sums are clearly dependant as they are all subsums of the same set of elements. One can also construct examples for which the assumption clearly does not hold. A set of integers $a_i \mod M$, for $i-1, ..., n$, that contains only even elements or that are all zero clearly does not lead to equally distributed values $\mathbf{a}\cdot\mathbf{x} \mod M$ in $\mathbb{Z}_M$ and the elements in the lists will clearly share strong characteristics. If we do not assume independence, we can still obtain a weak upper bound of the probability of failure which is smaller than the individual probabilities of failure.

We follow the heuristic model and assume independence due to the results of section 4.1 which shows that the values $\mathbf{a} \cdot \mathbf{x} \mod M$ behave well for almost all modular knapsacks $\mathbf{a} = (a_1, .., a_n) \in \mathbb{Z}_M^n$. Depending on a fixed integer parameter $\lambda > 0$, we ask that at least M/λ values in $\mathbb{Z}_M$ are attained. The number of knapsacks $F(\lambda)$ which do not comply to this condition can then be upper bounded:

$$F(\lambda) \leq \frac{M-1}{(\lambda-1)|\mathcal{B}|} \cdot M^n \leq \frac{1}{\lambda-1} M^n$$

for $M \leq |\mathcal{B}|$.

Our algorithm will fail to find the solution or have a substantial increase in its complexity if the modular subsums are cumulated on few values. In this case we may fail to pick the corresponding target or we experience a considerable increase in the number of collisions and elements to be stored. In the first case, the solution is not found. If we detect an increase in the number of produced elements, we can stop and change the target. We hence redefine our notion of a bad knapsack as one that has subsums that are far from uniform such that no representation fulfils the modular constraints or for which the number of collisions or the lists are too large in the sense of section 5.4. We find that the total fraction of bad knapsacks for the seven decompositions is smaller than

$$7\left(\frac{1}{\lambda-1} + \frac{2\,\lambda}{(\lambda-1)^2} + \frac{16}{\lambda}\right) \leq \frac{140}{\lambda} \quad \text{for } \lambda \geq 7 \ .$$

By choosing a large enough value for λ, this fraction can become arbitrarily small.

For a good knapsack, we can bound the proportion of random target values which do not lead to a solution, V_λ, due at least one of the three reasons above. We remind that at most $1 - 1/\lambda$ values are not attained, at most $1/(2\cdot\lambda)$ values let the lists overflow and at most $2/(8\cdot\lambda)$ values lead to too many collisions. The proportion of random target values that are not in our favour as they let a decomposition fail is thus smaller than

$$\frac{\lambda-1}{\lambda} + \frac{1}{2\,\lambda} + \frac{2}{8\,\lambda} = 1 - \frac{1}{4\,\lambda} \ .$$

By repeating each decomposition 8λ times on average with changed target chosen randomly and independently, we make sure that the probability of failure of one decomposition, P_{fail}, is at most

$$\left(1 - \frac{1}{4\lambda}\right)^{8\lambda} \approx e^{-2} \approx 0.135 \ .$$

The running time is multiplied by $(8\lambda)^7$ while the memory requirement stays as before. We assume here that we create $(8\lambda)^7$ target tuples for which we run the algorithm. Assuming independence between the seven decompositions leads to a global probability of success of $1 - 7 \cdot P_{fail} \approx 5\%$, which becomes exponentially close to 1 by repeating polynomially many times.
Given a real $\varepsilon > 0$, we set $\lambda = 2^{\varepsilon\, n}$ and can generalize the above result:

Theorem 5.1
For any real $\varepsilon > 0$ and for a fraction of at least $1 - 140 \cdot 2^{-\varepsilon\, n}$ of equibalanced knapsacks with density $D < 1$ given by an n-tuple $(a_1, \cdots, a_n)$ and a target value S, if $\epsilon = (\epsilon_1, \cdots, \epsilon_n)$ is a solution of the knapsack then the algorithm described in section 5.3 modified as above finds the solution and runs in time $\tilde{\mathcal{O}}\left(2^{(0.291+7\varepsilon)\, n}\right)$.

We recall that in the theorem, the term *equibalanced* means that the solution ϵ contains exactly the same number of 0s and 1s.

Chapter 6

Constant-memory algorithms

The knapsack problem can also be solved using a negligible amount of memory by use of a constant-memory-cycle-finding algorithm [BCJ11]. For simplicity we assume that n is a multiple of 4 and that the Hamming weight, the number of non-zero coordinates, of the knapsack solution $\mathbf{x}$ is exactly $n/2$.

6.1. Classical decomposition

We redefine the problem as a collision search problem [vOW96] as follows: Let $f_1, f_2 : \{0,1\}^{n/2} \to \{0,1\}^{n/2}$ be two functions defined as

$$f_1(\mathbf{y}) = \sum_{i=1}^{n/2} a_i y_i \mod 2^{n/2} \text{ and } f_2(\mathbf{z}) = S - \sum_{i=n/2+1}^{n} a_i z_i \mod 2^{n/2}$$

where y_i denotes the i-th bit of $\mathbf{y} \in \{0,1\}^{n/2}$, and similarly for z_i. If we can find $\mathbf{x}, \mathbf{y}$ such that $f_1(\mathbf{y}) = f_2(\mathbf{z})$, then we get:

$$\sum_{i=1}^{n/2} a_i y_i + \sum_{i=n/2+1}^{n} a_i z_i = S \mod 2^{n/2}$$

which we have to verify over the integers. Heuristically, there are $2^{n/2}$ such tuples $(\mathbf{y}, \mathbf{z})$ and only a single one that fulfils the equation over $\mathbb{Z}$ (for a hard knapsack). This means that whenever we find $\mathbf{y}$ and $\mathbf{z}$ such that $f_1(\mathbf{y}) = f_2(\mathbf{z})$, we have found the correct knapsack solution with probability roughly $2^{-\frac{n}{2}}$.

From the two functions f_1, f_2 we define the function $f : \{0,1\}^{n/2} \to \{0,1\}^{n/2}$ where:

$$f(\mathbf{y}) = \begin{cases} f_1(\mathbf{y}) & \text{if } g(\mathbf{y}) = 0 \\ f_2(\mathbf{y}) & \text{if } g(\mathbf{y}) = 1 \end{cases}$$

where $g : \{0,1\}^{n/2} \to \{0,1\}$ is a function that outputs 0 or 1 on input $\mathbf{y}$ with probability 1/2. Then a collision of f, i.e., when $f(\mathbf{y}) = f(\mathbf{z})$, gives a desired collision $f_1(\mathbf{y}) = f_2(\mathbf{z})$ or $f_2(\mathbf{y}) = f_1(\mathbf{z})$ with probability 1/2. The two cases where $\mathbf{y}$ and $\mathbf{z}$ are evaluated with the same function do not give a solution to the original problem.

The function $f : \{0,1\}^{n/2} \to \{0,1\}^{n/2}$ is a random function, therefore using Floyd's cycle finding algorithm [Knu81] we can find a collision for f in time $2^{n/4}$ and constant memory. We may fail to find the solution with probability $2^{-\frac{n}{2}}$ even if we have found a collision over $2^{\frac{n}{2}}$. We can expect to find the solution by repeating Floyd's algorithm on a different sequence of $\mathbf{y}$'s about $2^{\frac{n}{2}}$ times. For each new start, we can apply a random permutation on the elements in the knapsack or change the sequence of $\mathbf{y}$'s. In the latter case, we also need to randomize g by choosing a random element $\mathbf{r} \in \{0,1\}^{\frac{n}{2}}$ and evaluation g on $\mathbf{r} \oplus \mathbf{y}$. This gives an algorithm with total running time $\tilde{\mathcal{O}}\left(2^{3n/4}\right)$ and constant memory.

Bisson and Sutherland presented a similar idea [BS12] to solve a generalized subset-sum problem in a generic finite group by use of Pollard's Rho algorithm. Their technique applies to cases of large density in which case many solutions may exist.

6.2. Decomposition inspired by representation technique

We now show how to slightly decrease the running time down to $\tilde{\mathcal{O}}\left(2^{0.72n}\right)$, still with constant memory by use of the representation technique.

We let $\mathcal{B}^n_{n/4}$ be the set of n-bit strings of Hamming weight $n/4$. We have $|\mathcal{B}^n_{n/4}| = \binom{n}{n/4} \approx 2^{n\,h(1/4)} \approx 2^{0.81n}$. We define the two functions $f_1, f_2 : \mathcal{B}^n_{n/4} \to \{0,1\}^{h(1/4)n}$:

$$f_1(\mathbf{y}) = \sum_{i=1}^{n} a_i y_i \mod 2^{h(1/4)n}, \quad f_2(\mathbf{z}) = S - \sum_{i=1}^{n} a_i z_i \mod 2^{h(1/4)n}$$

where y_i denotes the i-th bit of $\mathbf{y}$, and similarly for z_i. In a first step, we search for $\mathbf{y}, \mathbf{z} \in \mathcal{B}^n_{n/4}$ such that:

$$f_1(\mathbf{y}) = f_2(\mathbf{z}) \tag{6.1}$$

which is equivalent to:

$$\sum_{i=1}^{n} a_i y_i + \sum_{i=1}^{n} a_i z_i = S \mod 2^{h(1/4)n} \ .$$

Since f_1 and f_2 are random functions, there are heuristically about $2^{h(1/4)n}$ solutions to (6.1). For a correct solution $\mathbf{x}$ there are $\binom{n/2}{n/4} \simeq 2^{n/2}$ representations meaning that there are about $2^{n/2}$ ways of writing

$$\sum_{i=1}^{n} a_i y_i + \sum_{i=1}^{n} a_i z_i = S$$

where $\mathbf{x} = \mathbf{y} + \mathbf{z}$ for $\mathbf{y}, \mathbf{z} \in \mathcal{B}^n_{n/4}$. All these $2^{n/2}$ solutions are solutions of (6.1). Therefore the probability $\mathcal{P}$ that a random solution of (6.1) leads to the correct knapsack solution is

$$\mathcal{P} = \frac{2^{n/2}}{2^{h(1/4)n}} \approx 2^{-0.31\,n} \ .$$

The input space of f_1, f_2 has size $2^{h(1/4)n}$. Therefore using the same cycle-finding algorithm as in the previous section, a random solution of (6.1) can be found in time $\tilde{\mathcal{O}}\left(2^{h(1/4)n/2}\right)$. The total time complexity is therefore:

$$\begin{aligned}\tilde{\mathcal{O}}\left(2^{h(1/4)n/2}\right)\cdot\mathcal{P}^{-1} &= \tilde{\mathcal{O}}\left(2^{h(1/4)n/2}\right)\cdot 2^{(h(1/4)-1/2)n} \\ &= \tilde{\mathcal{O}}\left(2^{(3h(1/4)/2-1/2)n}\right) = \tilde{\mathcal{O}}\left(2^{0.72n}\right) \quad .\end{aligned}$$

Finally, we note that it is possible to further improve this complexity by adding -1s in the decomposition (as in chapter 5) but the time complexity improvement is almost negligible.

Chapter 7

Summary – Comparison of complexity

The previous chapters presented generic algorithms that solve the subset-sum problem (definition 2.2) of density close to one in exponential time and memory. The most efficient classical algorithm, proposed by Shamir and Schroeppel (section 3.1), solves all instances in time $\tilde{\mathcal{O}}\left(2^{\frac{n}{2}}\right)$ using space $\tilde{\mathcal{O}}\left(2^{\frac{n}{4}}\right)$ given a knapsack of n elements. The algorithms splits the set of knapsack elements into four disjoint sets and enumerates all possible subsums within the small sets. The solution vector is in this way constructed by concatenation of four vectors. One can trade memory against time (section 3.3) and obtain an algorithm of slightly increased running time $\tilde{\mathcal{O}}\left(2^{(13/16-\varepsilon)\,n}\right)$ using less memory: $\tilde{\mathcal{O}}\left(2^{(1/16+\varepsilon)\,n}\right)$ for real ε, $0 \leq \varepsilon \leq 3/16$.

An alternative represents the algorithm by Howgrave-Graham and Joux (section 4.2). The solution is recovered from a sum of four vectors of same length than the solution vector and of weight one forth each. As this decomposition is non unique one obtains degrees of freedom. To reduce the cost in return, the authors propose modular constraints that render the search probabilistic. The original algorithm needs $\tilde{\mathcal{O}}\left(2^{0.337\,n}\right)$ time and $\tilde{\mathcal{O}}\left(2^{0.311\,n}\right)$ space on average. An exponentially small fraction of knapsacks can not be solved in this way. The algorithm performs well in practice and that the complexity is close to the expected one. We show (section 4.2.2) that the memory requirement can be reduced to $\tilde{\mathcal{O}}\left(2^{0.272\,n}\right)$ at the cost of repetitions by increasing the modular constraints. The overall asymptotic running time stays unchanged.

Permitting a slightly larger weight for the vectors that build the solution as a sum, one can improve the running time (section 5). The average running time and memory requirement becomes $\tilde{\mathcal{O}}\left(2^{0.291\,n}\right)$. Admitting larger modular constraints, the memory can be reduced to $\tilde{\mathcal{O}}\left(2^{0.279\,n}\right)$ while the asymptotic running time remains unaffected. The algorithm may fail to find the solution for some knapsack problems for which the assumptions of average subsums do not hold. The proportion is exponentially small. Experiments show that the algorithm behaves well in practice.

The largest running time using a constant amount of memory can be achieved by use of a cycle finding algorithm (section 6). The time is $2^{\frac{3}{4}n}$ and can be reduced to $2^{0.72\,n}$ using the ideas by Howgrave-Graham and Joux. Table 7.1 summarizes the results.

Table 7.1.: *Asymptotic time and memory to solve the subset-sum problem for a random instance of density one and size n where the solution vector has Hamming weight $\approx \frac{n}{2}$.*

Algorithm	Time	Memory	Section
Simple birthday paradox	$2^{\frac{n}{2}}$	$2^{\frac{n}{2}}$	3
Shamir-Schroeppel	$2^{\frac{n}{2}}$	$2^{\frac{n}{4}}$	3.1
Shamir-Schroeppel trade-off ($\varepsilon \leq 3/16$)	$2^{(11/16-\varepsilon)\,n}$	$2^{(1/16+\varepsilon)\,n}$	3.3
Simple representation technique	$2^{0.337\,n}$	$2^{0.311\,n}$	4.2.1
Simple representation technique (larger moduli)	$2^{0.337\,n}$	$2^{0.272\,n}$	4.2.2
Extended representation technique	$2^{0.291\,n}$	$2^{0.291\,n}$	5.1
Extended representation technique (larger moduli)	$2^{0.291\,n}$	$2^{0.279\,n}$	5.2
Cycle finding	$2^{\frac{3}{4}n}$	constant	6
Cycle finding (representation technique)	$2^{0.72\,n}$	constant	6.2

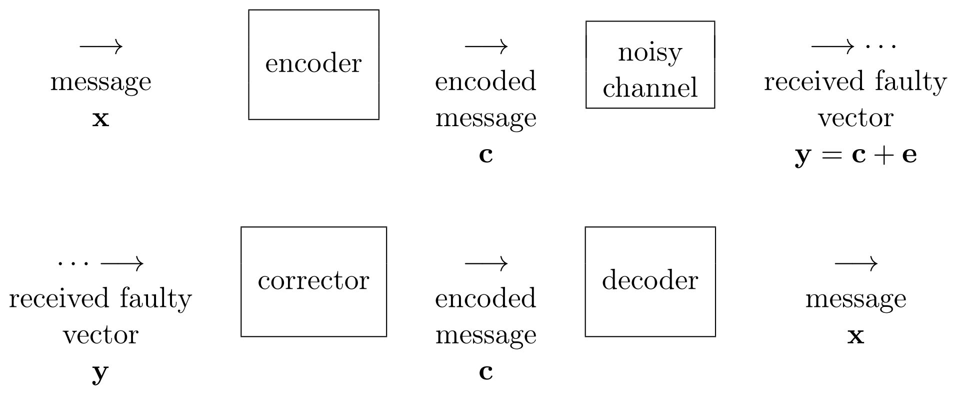
Part III

Improving information-set decoding

Chapter 8

Error-correcting codes in cryptography

Error-correcting codes aim at improving the reliability of a communication over a noisy channel by detection and correction of errors that occur during the transmission. In order to enable a recovery of the original message, the sender adds redundancy during the encoding process and obtains a codeword. The codeword is then sent over the unreliable channel and the receiver checks for consistency. If errors occurred, the receiver corrects and decodes the received information to obtain the message. The coding and decoding process is usually presented as in figure 8.1. Error-correcting codes are furthermore used to correct stored data,



message $\mathbf{x}$
encoder
encoded message $\mathbf{c}$
noisy channel
received faulty vector $\mathbf{y} = \mathbf{c} + \mathbf{e}$
received faulty vector $\mathbf{y}$
corrector
encoded message $\mathbf{c}$
decoder
message $\mathbf{x}$

Figure 8.1.: *Encoding and decoding of a message sent over a noisy channel.*

e.g. on hard drives, CDs or DVDs. Chapter 8.2 introduces cryptography based on linear error correcting codes.

A simple method to detect errors in the received message is to encode the message as a binary vector and to add a parity bit. Such a code can detect an odd number of errors in the transmitted codeword. However, it is limited in its usage as it can not detect an even number of changes to the bits of the codeword and can not specify the positions in which errors have occurred. A simple repetition of the message can detect which positions where changed if the errors have not occurred on the exact same positions of the codeword and its repetition. This method has the drawback of long codewords.

We will now formalize the description of a family of codes introducing linear codes that are used to build cryptographic systems.

8.1. Linear error correcting codes

The codewords of a linear code are obtained by applying a linear map to the messages or information vector which are represented as vectors of length k over the field $\mathbb{F}_q$. The linear code C is the set of all codeword vectors of length n, $n > k$, over $\mathbb{F}_q$ that are images of the message under the linear map; it is thus a vectorial subspace of $\mathbb{F}_q^n$ of dimension k and is generated by k linearly independent vectors in $\mathbb{F}_q^n$ which form the rows of a matrix G. The codewords are obtained as linear combinations of these basis vectors and coefficients in the underlying field $\mathbb{F}_q$.

Definition 8.1 (Linear Code)
Given a matrix $k \times n$ matrix G over $\mathbb{F}_q$ of rank k. The linear code *C is defined as the set $C := \{c = xG : x \in \mathbb{F}_q^k\}$; it has length n and dimension k. The matrix G is called a* generator matrix *of the code.*

We can alternatively define a linear code by use of the its dual code.

Definition 8.2 (Dual Code)
The dual code *$C^\perp$ of a code C, is the set of all vectors that are orthogonal to the codewords of C.*

We can write the generator matrix in systematic form $G = [I_k|A]$ where I_k is the identity matrix of dimension k and A is a $k\times(n-k)$ matrix. Given a code C with systematic generator matrix G, the generator matrix H of $C^\perp$ equals $[-A^T|I_{n-k}]$ in systematic form. The matrix H provides a parity check for C as it shows how certain linear combinations of the digits of a codeword sum up to zero; it is therefore called the parity check matrix.

The code C can then also be defined as the kernel of the map defined by matrix H, i.e.,

$$C = \left\{c \in \mathbb{F}_q^n : Hc^T = 0\right\} \quad .$$

The generator matrix and the parity check matrix of a code are not unique. The matrices are unique up to multiplication by a regular $k \times k$ matrix over $\mathbb{F}_q$.

We can also imagine to apply the same coordinate permutation on all codewords or a permutation on the symbols of the alphabet Q in which we express the messages. We derive an equivalent code of same length, dimension and minimum distance. We consider that $Q = \mathbb{F}_q$ and equivalent codes can be obtained as follows.

Definition 8.3 (Equivalent code)
We call two codes equivalent *if the generator matrices can be transformed into one another by permutations or scaling of rows or columns, or by addition of rows.*

Note that Gaussian elimination applies only these operations and can be used to bring G and H into systematic form or to check for equivalence.

The generator matrix of a code G is a $k \times n$ matrix over $\mathbb{F}_q$ of rank k which implies that the linear map $\mathbb{F}_q^k \to \mathbb{F}_q^n$, $x \mapsto xG$ is injective. A message is uniquely encoded to a codeword. In case of a systematic generator matrix, a message x is uniquely encoded to the codeword $c = xG$ such that the first k entries of c equal the message. The corresponding coordinates are called *information symbols*. The last $n-k$ coordinates are the *parity-check symbols* and provide redundancy. Figure 8.2 shows the encoding by use of a systematic generator matrix.

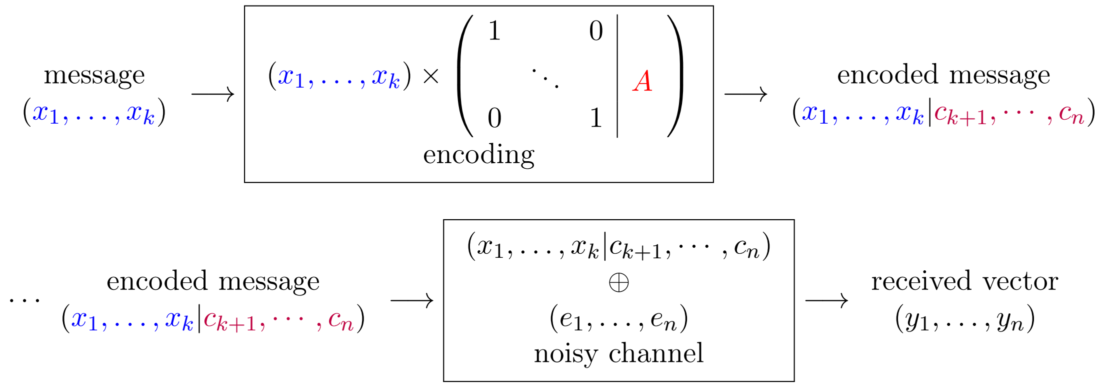

message
$(x_1, \dots, x_k)$
$(x_1, \dots, x_k) \times \left(\begin{array}{ccc|c} 1 & & 0 & \\ & \ddots & & A \\ 0 & & 1 & \end{array}\right)$
encoding
encoded message
$(x_1, \dots, x_k | c_{k+1}, \cdots, c_n)$
$\dots$
encoded message
$(x_1, \dots, x_k | c_{k+1}, \cdots, c_n)$
$(x_1, \dots, x_k | c_{k+1}, \cdots, c_n)$
$\oplus$
$(e_1, \dots, e_n)$
noisy channel
received vector
$(y_1, \dots, y_n)$

Figure 8.2.: *Encoding and transmission of a message by a linear code with systematic generator matrix.*

This leads to the the notion of an *information-set*.

Definition 8.4 (Information set)
Given a generator matrix G for a linear code. The information set $I \subset \{1, .., n\}$ indexes k columns of G such that they form an invertible $k \times k$ submatrix of G. The columns $\{1, .., n\} \setminus I$ of a parity check matrix H of the code form an $n-k \times n-k$ invertible submatrix of H.

We can also set the length of a message in relation to the length of a codeword and express in this way the amount of information that a symbol in a codeword contains.

Definition 8.5 (Information rate)
The information rate *R, the ratio between the length k of a message and the length n of a codeword, $R := k/n$.*

To further characterize a code, we need the notion of a distance over $\mathbb{F}_q^n$. A widely used distance is the Hamming distance.

Definition 8.6 (Hamming distance, Hamming weight)
The Hamming *distance between two vectors $x = (x_1, .., x_n), y = (y_1, ..y_n) \in \mathbb{F}_q^n$ is the number of coordinates in which x and y differ. It is defined as the non-negative integer*

$$d(x, y) := |\{1 \le i \le n : x_i \neq y_i\}| \ .$$

We denote by wt(x) *the* H*amming weight of x which is defined as*

$$\mathrm{wt}(x) := d(x, 0) \ ;$$

It denotes the number of non-zero positions of x.

An important characteristic of a code C is the minimal distance between two different codewords which is defined by use of the Hamming distance.

Definition 8.7 (Minimum distance)
The minimum distance *d between two codewords is the minimum number of different coordinates of two codewords,*

$$d := \min \{d(x, y) : x, y \in C, x \neq y\} \ .$$

In the case of a linear code, the difference of two codewords is also in the code, which means that d is equal to the length of the shortest codeword, i.e,

$$d = \min \{\mathrm{wt}(c) : c \in C \setminus \{0\}\} \ .$$

Note that for linear codes, the minimum distance is defined by the shortest codeword and it therefore requires only $|C| - 1$ operations to find it. In the general case one would have to compute the difference of all tuples of codewords, which are $\binom{|C|}{2}$ many.
We denote a binary linear code C of length n, dimension k and minimum distance d as an $[n, k, d]$-code.

A code is furthermore specified by

- the *error-detection capability*, the maximal number of coordinates of an erroneous codeword that can be detected and
- the *error-correction capability t*, the maximal number of coordinates of an erroneous codeword that can be corrected.

We will see that both properties depend on the minimum distance of the code.
An encoder and a decoder for an $[n, k, d]$-codecan be described formally as follows.

Definition 8.8 (Encoder, Decoder)
Let G be the generator matrix of a code C. An encoder *is an injective map $Enc : \mathbb{F}_q^k \to C \subseteq \mathbb{F}_q^n$, $x \mapsto xG$. A* d*ecoder is a map $Dec : \mathbb{F}_q^n \to \mathbb{F}_q^k$ that maps $c \in C$ to x.*

A decoder that always outputs the closest codeword c to a received vector $y \in \mathbb{F}_q^2$ (w.r.t. the Hamming distance) is called a *maximum-likelihood decoder*. Note that a maximum-likelihood decoder does not ensure a decoding of a received vector to the correct codeword. If a maximum-likelihood decoder rejects the encoding in case of an error that exceeds a bound such as $t = (d - 1)/2$, we call it a *bounded-distance decoder*.

Corresponding to the bounded-distance decoder, we define a t-corrector [Dal10].

Definition 8.9 (t-corrector)
A t-corrector *is a map* $Corr : \mathbb{F}_q^n \to \mathbb{F}_q^k$ *that maps* $y \in \mathbb{F}_q^n$ *to* c *if and only if* $d(y,c) \leq t$ *and outputs* $\perp$ *(error) else.*

For the design of a code-based system, it is essential to find a fast decoding algorithm. There are many code families with fast decoding algorithms, e.g., Goppa codes, (generalized) Reed–Solomon codes, Gabidulin codes, Reed-Muller codes, algebraic- geometric codes, BCH codes and others. Section 8.1.2 introduces classical Goppa codes which are often used for cryptographic schemes as we explain in section 8.2. More details about families of codes can be found in [MS77, HP03].

Fig 8.3 depicts the codewords of an $[n,k,d]$-code C as discrete points in $\mathbb{F}_q^n$ of minimal distance d. If we draw a ball around a codeword c of radius strictly smaller than $t = (d-1)/2$, the *packing radius*, we enclose all vectors $y \in \mathbb{F}_q^n$ that are closer to $c \in C$ than to any other codeword. In other words, the vector y equals $c + e$ for some $e \in F_q^n$ of maximal weight $(d-1)/2$.

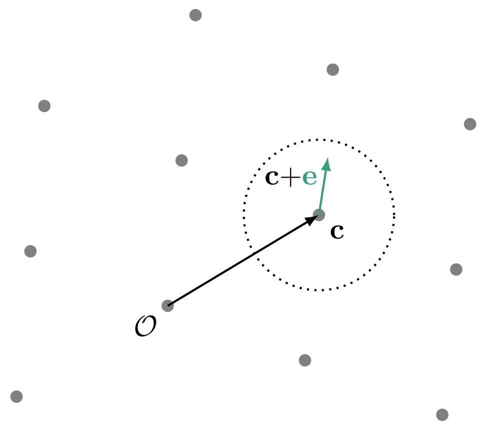


Figure 8.3.: *Codewords of a linear code with minimal distance d.*

Assume that we receive a faulty vector y and that the weight of the error is bounded by the packing radius t. Hence we know that there is a unique codeword c which lies in the sphere around y of radius t. The code has therefore an *error-correction capability* t. If the error in y increases up to a weight of $d-1$, it can be detected by testing if y lies in the code. The code has an *error-detection capability* of $d-1$. However, the correct codeword might no longer be the closest one to y.

We can also see that a designer of a code would like to obtain a large minimum distance that separates the codewords in order to get a high error-correction capability. A good code aims to have a high information rate and error-detection/correction capacity while holding the cost for encoding and decoding low.
By the observations above, we can make the following claim.

Corollary 8.1
An $[n,k,d]$-code together with a bounded distance decoder provides a unique decoding for an error in at most $t = \lfloor \frac{d-1}{2} \rfloor$ coordinates. It detects an error if the error vector has weight at most $d-1$.

To obtain a bounded distance decoder, we introduce the definition of a syndrome.

Definition 8.10 (Syndrome)
Given the parity check matrix H of an $[n,k,d]$-code, we call the vector $s = Hy^t$ the syndrome of the vector $y \in \mathbb{F}_q^n$.

For $c \in C$, we know that $Hc^t = 0$ by definition. For a received vector $y = c + e, e \notin C$, we obtain $Hy^t = Hc^t + He^t = He^t$ by linearity. This means that the syndrome of a received vector y is characterized by the syndrome of the error pattern e that we need to remove in order to retrieve c. For a binary linear code, the syndrome is the sum of the columns of H indexed by e.

Decoding in general means to find the closest codeword to a received vector. On reception of y, we have to compare y with all codewords, which takes time $|C|$ for a general code. For a linear code however, we know that for a received vector $y = c + e$, the syndrome $s = Hy^t$ defines the error as $s = He^t$. Decoding then means to find e given H, s such that $y - e$ lies in the code and e has minimal weight.
To recover the message, we thus need to solve the following problem.

Problem 8.1 (Syndrome-decoding problem (SD))
Given a parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ of a binary code, a syndrome $s \in \mathbb{F}_2^{n-k}$ and an integer ω, find a vector $\mathbf{e} \in \mathbb{F}_2^n$ of weight ω such that $H\mathbf{e}^t = s$.

The decisional version is proven to be NP-complete [BMvT78] for binary linear codes. For the q-ary case see [Bar98, Theorem 4.1]. We will focus on the binary case. It is equivalent [LDW94] to the bounded-distance-decoding problem which is defined as follows.

Problem 8.2 (Bounded-distance-decoding problem)
Given a generator matrix $G \in \mathbb{F}_2^{k\times n}$ of a binary linear code C, a random vector $y \in \mathbb{F}_2^n$ and an integer ω, find a vector $\mathbf{e} \in \mathbb{F}_2^n$ of weight at most ω such that $y + e \in C$.

We focus on syndrome decoding that solves the syndrome decoding problem. The notion of a *coset* comes in handy.
For $x \in \mathbb{F}_q^n$, we define a coset which has size $q^k = |C|$:

$$\mathbf{x} + C = \{\mathbf{x} + \mathbf{c} \,|\, c \in C\} \ .$$

Each vector in $\mathbb{F}_q^n$ belongs to exactly one coset such that we obtain a partitioning of the space:

$$\mathbb{F}_q^n = C \cup (\mathbf{x}_1 + C) \cup \ldots \cup (\mathbf{x}_\ell + C)$$

where $\ell = q^{n-k} - 1$ and $\mathbf{x}_i \in \mathbb{F}_q^n$. Each coset has a shortest vector, the *coset leader.*

Two vectors belong to the same coset if and only if there difference lies in the code: Let $\mathbf{y}_1 = \mathbf{x} + \mathbf{c}_1 \in \mathbb{F}_q^n$ and $\mathbf{y} = \mathbf{x}_i + \mathbf{c}_2 \in \mathbb{F}_q^n$ for $\mathbf{c}_1, \mathbf{c}_2 \in C$. Then $\mathbf{y}_1 - \mathbf{y}_2 = \mathbf{c}_1 - \mathbf{c}_2 \in C$.

Conversely, if $\mathbf{y}_1, \mathbf{y}_2 \in C$, they clearly belong to the coset $0 + C$. Otherwise, let $\mathbf{y}_1, \mathbf{y}_2 \notin C$ and $\mathbf{y}_1 - \mathbf{y}_2 \in C$. Then $\mathbf{y}_1 = \mathbf{x} + \mathbf{c}_1$ and $\mathbf{y}_2 = \mathbf{x} + \mathbf{c}_2$ for $\mathbf{c}_1, \mathbf{c}_2 \in C$ and $\mathbf{x} \notin C$. So $\mathbf{y}_1, \mathbf{y}_2$ both belong to the same coset than $\mathbf{x}$.

Due to linearity, we deduce that all elements of a coset share the same syndrome s: Let $\mathbf{y}_1, \mathbf{y}_2 \in \mathbf{x} + C$. Then $\mathbf{y}_1 = \mathbf{x} + \mathbf{c}_1$ and $\mathbf{y}_2 = \mathbf{x} + \mathbf{c}_2$ for $\mathbf{c}_1, \mathbf{c}_2 \in C$. By linearity and definition, we derive that $H\mathbf{y}_i^t = H(\mathbf{x} + \mathbf{c}_i)^t = H\mathbf{x}^t$ for $i = 1, 2$.

Syndrome decoding performs the following steps. Suppose that we receive an erroneous message $\mathbf{y} = \mathbf{e} + \mathbf{c}$ at short distance $\mathbf{e}$ from the code. Let $\mathbf{y} \in \mathbf{x} + C$. We can compute its syndrome w.r.t. the parity check matrix of the code: $H\mathbf{y}^t = H(\mathbf{c} + \mathbf{e})^t = H\mathbf{e}^t = s$. Note that the error vector $\mathbf{e}$ that we search belongs to the same coset than $\mathbf{y}$. One strategy for decoding is to find the coset to which $\mathbf{y}$ belongs, to find the coset leader $\mathbf{x}$ and to check if $\mathbf{y} - \mathbf{x}$ lies in the code. If this is the case, $\mathbf{y} - \mathbf{x}$ is a good maximum-likelihood estimate for the sent codeword. This requires to store a hashtable of all coset leaders indexed by their syndrome.

8.1.1. Bounds for linear codes

We can count the number of words at distance ℓ from $y \in \mathbb{F}_q^n$, $N_\ell(y)$. We have to enumerate all vectors that differ in up to ℓ positions and there are $q - 1$ possibilities for each position:

$$N_\ell(y) = \sum_{i=0}^{\ell} \binom{n}{i} (q-1)^i \ .$$

We remark that the largest binomial in the sum is the last which counts the number of vectors furthest away from y.

We saw above that all spheres around the codewords of radius $t = \lfloor (d-1)/2 \rfloor$ are disjoint and we know that there are q^n distinct words in $\mathbb{F}_q^n$. We conclude that $N_t(y) \leq q^n$ and have shown the following theorem.

Theorem 8.1 (Hamming bound)
For a q-ary $[n, k, d]$-code C that corrects up to $t = \lfloor (d-1)/2 \rfloor$ errors,

$$|C| \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^n \ . \tag{8.1}$$

We call the code C perfect, if equality holds in the above theorem; the spheres of radius t around the codewords then cover $\mathbb{F}_q^n$. The maximal integer d_0 such that

$$|C| \sum_{i=0}^{d_0 - 1} \binom{n}{i} (q-1)^i \leq q^n \tag{8.2}$$

is called the *Gilbert-Varshamov distance*. Note that if we add the binomial $\binom{n}{d_0}$, we exceed the bound q^n.

Let us consider a fixed code rate $R := k/n$ and error rate $D := d/n$. For increasing length

n, the last binomial of the sum in (8.2) dominates and we derive that

$$h_q(D) \geq 1 - R$$

where h_q is the entropy function. The sections A.1 and A.2 in the appendix present the entropy function and asymptotic approximations that lead to the above result.

The largest integer $\delta_0 = Dn$ for which equality holds is called the *relative Gilbert-Varshamov distance* or *asymptotic Gilbert-Varshamov bound* [Bar98, section 1.2]. One can show that for a random linear codes of code rate R and growing length n, for any $\varepsilon > 0$ the fraction of codes with relative distance

$$\delta \leq \delta_0 - \varepsilon$$

is negligible [Bar98, section 1.3]. This means in return that most random linear codes meet the Gilbert-Varshamov bound and have minimal distance δ_0 for growing length. We can hence assume that $h_q(D) = (1 - R) + o(1)$ for most random linear codes.

Another lower bound on the number of codewords is the Singleton bound. We have seen that an $[n, k, d]$-code can detect up to $d-1$ as the unique codeword can still be identified even with an error in $d - 1$ coordinates. This implies that every codeword can be indexed by only $n - (d - 1)$ coordinates. As over a field of size q, we can only have q^{n-d+1} elements, we have found a natural bound for the number of codewords.

Theorem 8.2 (Singleton bound)
For a q-ary $[n, k, d]$-code C it holds that

$$|C| \leq q^{n-d+1} \quad . \tag{8.3}$$

A code for which equality holds, in (8.3) is called *maximum-distance-separable*; it attains the maximal possible distance $d = 1 + n - k$. For fixed information rate $R = k/n$ and growing q, the distance derived from the Gilbert-Varshamov bound approaches quickly the Singleton bound and the code is maximum-distance-separable with high probability.

As long as for an $[n, k, d]$-code, the spheres of radius $d - 1$ do not cover $\mathbb{F}_q^n$, i.e.,

$$|C| \sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i < q^n \quad ,$$

we can find a word not included in the spheres and add it to the code. The following theorem summarizes the result and gives a lower bound on the size of the code.

Theorem 8.3 (Gilbert-Varshamov bound)
There exists a q-ary $[n, k, d]$-code C provided that

$$|C| \sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i \geq q^n \quad .$$

The proof can be found in [Moo05]. For a linear code, we have that $|C| = q^k$ and the Gilbert-Varshamov bound is given by

$$\sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i \geq q^{n-k} \quad .$$

The theory of error-correcting codes is motivated by Claude Shannon who lay the foundation for communication theory [Sha48]. He theoretically answered the question how much redundancy is needed to ensure that a message sent over a noisy channel can be recovered.

Theorem 8.4 (Shannon's Second Theorem (1949))
Let p be the probability of each bit being in error for a message of length k. Let k be large. To transmit a message, we require a proportion $h(p) = p\log_2(1/p) + (1-p)\log_2(1/(1-p))$ of additional redundant bits. Thus, if we let the total number of bits needed be n, we have $n = k + nh(p)$. Not only must the code be at least this long, but most (almost all) codes with longer bit length ck, for any $c > 1/(1-h(p))$, allow us to recover the message under the given assumptions, with probability approaching 1 as the length of the code goes to ∞.

The theorem says that for a random code of length as above, we can obtain the original message most of the time. In practice it is quite difficult to construct codes that meet the bound in the asymptotic limit.

8.1.2. Goppa code

The classical Goppa codes were introduced by Valery D. Goppa [Gop71a, Gop71b] and belong to the more general family of alternant codes/generalized Reed-Solomon codes [MS77].

Definition 8.11 (Goppa code)
Let $S = \{\alpha_1, .., \alpha_n\}$ be subset of $\mathbb{F}_{q^m}$ of size n. Let $g(x)$ be a polynomial of degree t in $\mathbb{F}_{q^m}[x]$ for which $g(\alpha_i) \neq 0$ for any α_i. The Goppa code is then defined as

$$\Gamma(S, g) = \{c = (c_1, .., c_n) \in \mathbb{F}_q^n : \sum_{i=1}^{n} \frac{c_i}{x - \alpha_i} \equiv 0 \mod g(x)\} \quad .$$

The set of elements a_i is called the support of the code and g is its generator polynomial. To guarantee that g does not vanish at its support, it is common practice to choose g to be a nonlinear irreducible element of $\mathbb{F}_{q^m}[x]$. The code Γ is then called an irreducible Goppa code.

We remark that though constructed using support elements over F_{q^m} and a polynomial in $F_{q^m}[x]$ the codewords are only those n-tuples with entries in F_q satisfying the defining equation.

Theorem 8.5
A Goppa code $\Gamma(S, g)$ as defined above is a linear code of length n, dimension $k \geq n - mt$ and minimum distance $d \geq t + 1$.

Theorem 8.6
Let $\Gamma(S, g)$ be a Goppa code as defined above where $g(x)$ is defined over $\mathbb{F}_{2^m}$ of degree t and has no multiple zeros. Then, the minimum distance is at least $2t + 1$.

Let $\Gamma(S, g)$ be a binary Goppa codes defined by an irreducible Goppa polynomial $g(x)$ of degree t in $\mathbb{F}_{2^m}[x]$. Then the code is t-error-correcting by theorem 8.6 and corollary 8.1.

For proofs of the above statements and more details about error correcting codes, we refer to [MS77, HP03].

8.2. Code-based cryptography

The present chapter introduces well known code-based cryptographic schemes: the McEliece and Niederreiter public-key cryptosystem as well as the CFS signature. Other cryptographic primitives and schemes based on codes are for example zero-knowledge identification schemes [Ste93, MGS11], ring signature schemes [MCGL11, DV09] and hash functions [AFS05].

8.2.1. McEliece's public-key cryptosystem

McEliece proposed code-based cryptography based on binary Goppa codes [McE78]. The system can also be used with Goppa codes over a larger alphabet [Pet10].

Let $n, k \in \mathbb{N}$ and $\omega \in \mathbb{N}$ be public system parameters. Let $\Gamma = \Gamma(S, g)$ be a random binary Goppa code over $\mathbb{F}_2$ of length n and dimension k with an error-correction capability ω. The generator matrix G, as well as an $n \times n$ permutation matrix P and an invertible $k \times k$ scrambling matrix S are randomly generated. The matrices G, P and S are kept secret. We derive the public key as $\hat{G} = SGP$ which is a pseudo-random generator matrix of a Goppa code C'. As $\hat{G}$ shall be indistinguishable from a random matrix no efficient decoding algorithm is known for the codes C'. Conversely, the knowledge of P, S and G allows for an efficient decoding.

We assume that an efficient decoding algorithm exists w.r.t. G but that $\hat{G}$ is pseudo-random. The message $\mathbf{m}$ is a binary vector of length k. Encryption is performed by encoding $\mathbf{m}$ using the public key $\hat{G}$ and adding a random error vector $\mathbf{e} \in \mathbb{F}_2^n$ of weight ω:

$$\mathbf{y} = \mathbf{m}\hat{G} + \mathbf{e} \ . \tag{8.4}$$

A receiver of $\mathbf{y}$ needs to remove the error $\mathbf{e}$ and decode $\mathbf{y} + \mathbf{e}$. A legitimate receiver can make use of the hidden structure of the code that permits an efficient decoder and use the private key. She computes $\mathbf{y}P^{-1} = \mathbf{m}SG + \mathbf{e}P^{-1}$ and uses the efficient decoder to retrieve $\mathbf{m}S$. A last linear algebra step recovers $\mathbf{m}$. The first parameters proposed [McE78] were $n = 1024$, $k \geq 524$ and $\omega = 50$; more recent propositions can be found in [BLP08].

8.2.2. Niederreiter's public-key cryptosystem

Niederreiter proposed a public-key cryptosystem [Nie86] in which the plaintext is encoded into an error pattern of length n and constant weight t by use of an invertible function [Sch72, FS96] $\phi_{n,t}$:

$$\phi_{n,t} : \{0,1\}^l \to W_{n,t}$$

where $W_{n,t} := \{\mathbf{e} \in \mathbb{F}_2^n : \mathrm{wt}(\mathbf{e}) = t\}$ and $l = \lfloor \log_2(|W_{n,t}|) \rfloor$ for security reasons. The scheme was originally proposed to use Goppa codes or Reed-Solomon codes. Together with generalized Reed-Solomon codes it was broken by Sidelnikov and Shestakov [SS92]. Niederreiter's system is the dual variant of the McEliece's cryptosystem and the security of both is equivalent [LDW94] when using Goppa codes.

We present a version of Niederreiter's scheme that uses Goppa codes and achieves a shorter public key by use of a *systematic* generator matrix [CS98, BS08].

Let $m, t \in \mathbb{N}$ be two system parameters and define $n = 2^m$. Let $\Gamma = \Gamma(S, g)$ be a binary Goppa code of dimension k defined by a polynomial $g(x) \in \mathbb{F}_{2^m}[x]$ of degree t and its support $S = (\alpha_1, .., \alpha_n) \in \mathbb{F}_{2^m}$, i.e., $g(\alpha_i) \neq 0$. The code Γ is t-error-correcting and we assume that an efficient decoding algorithm D_Γ is known (to the owner of the private key only). Let H be an $mt \times n$ binary parity check matrix in systematic form for Γ. The public key is H and the private key is (S, g, D_Γ).

To encrypt a plaintext $\mathbf{x}$, we first have to convert it to an error pattern $\mathbf{e}$ of length n and weight t. We define the set of such vectors by $W_{n,t} := \{\mathbf{e} \in \mathbb{F}_2^n : \mathrm{wt}(\mathbf{e}) = t\}$ and use an invertible function $\phi_{n,t}$ to derive $\mathbf{e}$ as $\mathbf{e} = \phi_{n,t}(\mathbf{x})$. The ciphertext is then $\mathbf{s} = H\mathbf{e}^t$.

On reception of a ciphertext, the owner of the private key applies the decoding algorithm D_Γ to obtain $\mathbf{e}$ and $\phi_{n,t}^{-1}$ to recover the plaintext $\mathbf{x}$:

$$\phi_{n,t}^{-1}(D_\Gamma(\mathbf{s})) = \phi_{n,t}^{-1}(\mathbf{e}) = \mathbf{x} \ .$$

8.2.3. Security of encryption schemes

The McEliece and Niederreiter scheme are equivalent [LDW94] when used with the same code. Two assumptions guarantee the security of the system. First, an attacker should not be able to attack a permuted Goppa code easier than a random code with the same parameters where the random instance shall be difficult to solve. These approaches are called structural attacks. Recently progress about the instinguishability problem has been achieved [FOPT10a, FOPT10b, Sen00]. However, the results do not permit to ameliorate the attacks against binary (quasi-dyadic, quasi-cyclic) Goppa codes as used in Niederreiter's and McEliece's systems. We will therefore assume that an attack against these Goppa codes is as hard as for a random code.

The second security assumption, is based on decoding: The recovery of $\mathbf{e}$ of weight ω, given a vector $\mathbf{s} = H\mathbf{e}^t$, must be difficult without the knowledge of the private key. Remember that this is the syndrome-decoding problem from section 8.1.

Canteaut and Chabaud [CC98] explained how to decode a binary linear code, thus breaking McEliece, by applying an algorithm that finds a low-weight codeword. We can reduce the decoding problem to the search of a low-weight codeword and conversely.

Let C be the code of length n and ω a positive integer smaller than half the minimum distance of the code. Suppose that one receives an encrypted message $\mathbf{y} \in \mathbb{F}_2^n$ at distance ω from the code (8.4). Necessarily, there is a closest codeword to $\mathbf{y}$, denoted $\mathbf{c}$, and the difference $\mathbf{y} - \mathbf{c}$ is a ω-weight codeword in $\langle C, \mathbf{y} \rangle$. We remark that the code has a slightly larger dimension, namely $k + 1$. Conversely, let C have a minimum distance larger than 2ω. An error-vector $\mathbf{e} \in \langle C, \mathbf{y} \rangle$ of weight ω can thus not be in the code C but is in the larger code $\langle C, \mathbf{y} \rangle$. We thus know that $\mathbf{y} - \mathbf{e}$ is in C at distance ω from $\mathbf{y}$.

For a given decoding problem based on a McEliece cryptosystem, one can simply append the ciphertext $\mathbf{y}$ to the rows of the generator matrix of the code C and obtain the code $\langle C, \mathbf{y} \rangle$. Since the error-vector has a weight smaller than the minimum distance of the code C, it belongs to $\langle C, \mathbf{y} \rangle$. Applying an algorithm that finds a low-weight codeword on $\langle C, y \rangle$ then allows to recover the message.

When attacking code-based cryptosystems, the instances one has to solve have ω smaller than half the minimal distance d of the code C: $\omega \le \lfloor \frac{(d-1)}{2} \rfloor$. The solution is then unique and the setting is called *half-distance* or *bounded-distance decoding*. All known half-distance decoding algorithms achieve their worst-case behavior for the choice $\omega = \lfloor \frac{(d-1)}{2} \rfloor$.

Information-set decoding (ISD) is the most efficient method to solve the syndrome-decoding problem 8.1 for a random linear code. Various research [Dum91, Bar98, BLP08, BLP11, CS98, MMT11, FS09, CC98, BJMM12] has been done which indicates that for an arbitrary linear code syndrome decoding is difficult. We present different techniques for information-set decoding in chapter 9 and present our improved algorithm in chapter 10.

8.2.4. CFS signature scheme

The system [CFS01] is set up as in Niederreiter's scheme in Sect. 8.2.2 for a t-error correcting code Γ from the class of $[n, k = n - mt, 2t + 1]$ binary irreducible Goppa codes. The public and private key are $H \in \mathbb{F}_2^{mt \times n}$ and (S, g, D_Γ), respectively. Let h be a hash function that applied on the document D outputs a vector $s \in \mathbb{F}_2^{mt}$, $s = h(D)$.

The signer applies h and decodes s using the decoding algorithm D_Γ obtaining the signature as the error pattern. The decoding is successful for a given vector $s \in \mathbb{F}_2^{mt}$ if s is at distance at most t from a codeword in Γ. The decoding may therefore fail. We assume that n is large and that $n \gg t$. Enumerating all vectors in the ball of radius t around a codeword gives the number of decodable vectors as

$$\sum_{i=1}^{t} \binom{n}{i} \approx \binom{n}{t} \approx n^t/t! = 2^{mt}/t! \ .$$

We compare this number with all possible vectors in $\mathbb{F}_2^{mt}$ to obtain the probability P that s is decodable, $P = 2^{mt}/t \cdot 1/2^{mt} = 1/t!$. To obtain a signature for D, the signer has to produce $t!$ different s on average to obtain one that is decodable. Two methods have been proposed to compute s.

Counter-Method One can alterate the document by appending a counter before hashing until the decoding is successful. The signer produces syndromes $s_j = h(D||j)$ successively until the decoder outputs an error pattern $e = D_\Gamma(s_j)$ of weight t. The last step of the signing process is to apply $\phi^{-1}_{n,t}$ to e. The signature contains the counter j and $\phi^{-1}_{n,t}(e)$: $(p||j) = (\phi^{-1}_{n,t}(e)||j)$. The verifier computes $\phi^{-1}_{n,t}(p) = e$ and $s = h(D||j)$. He accepts the signature if $s = He^t$.

Complete decoding The alternative lies in a change of the decoding algorithm such that (almost) all vectors are decodable. The covering radius r_0 is the smallest integer such that balls of radius r_0 around the codewords cover the whole space; it is difficult to calculate. Decoding up to r_0 would be ideal. We can obtain a good approximation using the following observation. For a random vector $y \in \mathbb{F}_2^{n-k}$, $mt = n - k$, the shortest vector e such that $He^t = y$ has most likely a weight larger than t. The weight d_0 is given by the Gilbert-Varshamov distance, the largest integer such that $\sum_{j=0}^{d_0-1} \binom{n}{j} \leq 2^{n-k}$.

A t-error decoder will therefore fail as the designed error-capability t is smaller than d_0. We would like to be able to encode a syndrome $s = h(D)$ into an error pattern of weight $\delta + t \approx d_0$. If we can erase δ positions in s, by addition of columns of H, such that the decoder outputs $\hat{e}$ of weight t, we can express s as $H\hat{e}^t + Hz^t$ where z indexes δ columns. The vector $e = \hat{e} + z$ is of weight at most $t + \delta$. Modifying the size of δ, we can regulate the proportion of vectors which are not decodable. If we take δ as the smallest integer such that $\binom{n}{t+\delta} > 2^{mt}$, then any s is decodable with high probability. The signature is $(p) = (\phi^{-1}_{n,t+\delta}(e))$. For verification, one computes again $\phi^{-1}_{n,t+\delta}(p) = e$ and $s = h(D)$ and compares s to He^t.

The choice of the parameters m, t has a great impact on signing, verification and the signature length. For the counter method, the signing costs performs $t!$ trials dominated by the cost to calculate the roots of a polynomial; the total cost are of order $\mathcal{O}\left(t!t^2m^3\right)$. The signature is a tuple (e, i). There are $\binom{n}{t}$ error-vectors which can be indexed and there are about $t!$ different indices i. The signature length is about $\log(\binom{n}{t} \cdot t!)$. There are possibilities to transmit not all error bits, down to $t - 2$, and to guess them at the receivers side [CFS01, CFS02]. This leads to a trade-off between verification time and signature length.
Storing the public key H needs memory of size $\mathcal{O}\left(mt2^m\right)$ in the classical Niederreiter scheme or $\mathcal{O}\left(mt(2^m - mt)\right)$ using a systematic key. The verification process than adds t columns of H and compares the value to the hash s of the message. We see that signing and signature length increase rapidly with t and that the key size depends especially on m. The first proposed parameter sets were $(t = 10, m \geq 15)$ and $(t = 9, m \geq 16)$. They considered the most efficient attack at that time [CC98].

8.2.5. Attacks against the CFS signature

One can distinguish between structural and decoding attacks that either reveal a private key or forge a signature. We will assume that a structural attack is infeasible and explain previous achievements of decoding algorithms in the following.

To forge a CFS signature without knowledge of the private key, an attacker has to decode one vector s_j, derived from the document by hashing, without knowing the efficient decoding algorithm; he has to solve one out of many instances of the syndrome-decoding problem for the same code:

Problem 8.3 (One-out-of-many-syndrome decoding (OMSD))
Given the parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ for a code C, a set of vectors $S = \{\mathbf{s}_j \in \mathbb{F}_2^{n-k}\}$ and an integer ω, find an error pattern $\mathbf{e} \in \mathbb{F}_2^n$ of maximal weight ω such that $H\mathbf{e}^t = \mathbf{s}_j$ for at least one j.

Information-set decoding (ISD) is the most efficient method to solve the syndrome-decoding problem 8.1 with a *single* target and has been applied to the case of many instances 8.3 [JJ02]. The asymptotic analysis [Sen11] shows that the method gains a factor of $\sqrt{|S|}$; the exponential in the complexity is of order $2^{mt/2}$. A more efficient way is to make use of the birthday paradox which leads to the generalized birthday algorithm.

Generalized birthday algorithm (GBA)

The generalized birthday algorithm (GBA) was first presented by Wagner to solve the k-sum problem [Wag02] and later extended in [MS09]. Bleichenbacher then proposed to use GBA on coding problems in order to solve OMSD (P 8.3) applied on the CFS signature. It is effective to break a CFS signature with the initially proposed parameter sets ($t = 9, 10$) in time complexity less than 2^{80}; it weakens (almost) all parameter sets of interest as detailed in [FS09]. We review the attack to forge a CFS signature, as proposed by Bleichenbacher and presented by Finiasz and Sendrier [FS09], in the following.

Given the parity check matrix H of an CFS signature scheme, a document D and its syndromes $\mathbf{s}_j$, we want to find an error pattern $\mathbf{e} \in \mathbb{F}_2^n$ of weight ω such that $H\mathbf{e}^t = \mathbf{s}_j$ for at least one j. We restate the problem as follows: Find elements $\mathbf{e}_i \in W_{n,\omega_i}$ such that $H(\mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3)^t = \mathbf{s}_j$ where $wt(\mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3) = w$. The vectors $\mathbf{e}_i$ are indexing the columns of H that sum up to a syndrome $\mathbf{s}_j$. The weights ω_i, $i = 1, 2, 3$ such that $\omega_1 + w_2 + w_3 = w$ are algorithm parameters.

- The first step is to create 3 lists L_i, of column sums of H, that is, of random elements $(H\mathbf{e}_i^t, \mathbf{e}_i)$, $\mathbf{e}_i \in W_{n,\omega_i}$ and one list S of syndromes $\mathbf{s}_j$.

- In the second step, we join the list L_1 with S and the list L_2 with L_3 and obtain elements $(\mathbf{s}_j + H\mathbf{e}_1^t, \mathbf{e}_1) \in L_{1S}$ and $(H\mathbf{e}_2^t + H\mathbf{e}_3^t, \mathbf{e}_2 + \mathbf{e}_3) \in L_{23}$, respectively. We keep those elements for which the first component is already zero on the λ MSBs. The optimal λ is chosen to reduce time and memory requirements as we will outline later.

- Joining these elements in the last step, leads to tuples $(\mathbf{s}_j + H\mathbf{e}_1^t + H\mathbf{e}_2^t + H\mathbf{e}_3^t, \mathbf{e}_1 + \mathbf{e}_2 + \mathbf{e}_3)$. We have found a solution, if the column sum $\mathbf{s}_j + H\mathbf{e}_1^t + H\mathbf{e}_2^t + H\mathbf{e}_3^t$ is zero for at least one joined element.

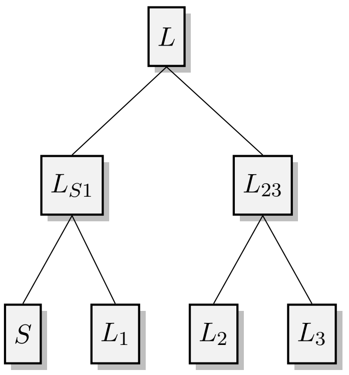


Figure 8.4.: *Generalized birthday attack.*

Fig. 8.4 illustrates the algorithm. We assume that $\mathbf{e}_i, \mathbf{s}_j$ are random and that all column sums are equally distributed elements. To reduce the memory usage, we can create the elements in S, L_{S1} and L_2 on the fly. The memory requirement of the algorithm is given by the size of the largest list:

$$M = \mathcal{O}\left(\max(|L_i|, |L_{23}|)\right) \ .$$

The starting lists L_i are of maximal size $\binom{n}{\omega_i}$. The size of L_{23} is of order

$$\mathcal{O}\left(\min(\binom{n}{\omega_2 + \omega_3}, |L_1| \cdot |L_2|) \cdot 2^{-\lambda}\right) \ .$$

The syndromes are generated when needed to reduce the memory requirements. The algorithm has a memory complexity of

$$\mathcal{M} = M \cdot \log(M) \ .$$

A join algorithm that searches collisions between two lists L_i, L_j and outputs a list L performs a sorting w.r.t. λ bits (e.g. MSB) and then compares for collisions. Neglecting polynomial factors[1] it takes time

$$\mathcal{T}_{join} = \tilde{\mathcal{O}}\left(\max(|L_i|, |L_j|, C)\right)$$

where C counts the number of comparisons. The memory requirement is given by the largest lists handled:

$$\mathcal{M}_{join} = \tilde{\mathcal{O}}\left(\max(|L_i|, |L_j|, |L|)\right) \ .$$

We expect in the second step a number of comparisons of

$$C_{1S} = |L_{1S}| = \mathcal{O}\left(|S| \cdot |L_1| \cdot 2^{-\lambda}\right) \text{ and } C_{23} = \mathcal{O}\left(|L_2| \cdot |L_3| \cdot 2^{-\lambda}\right) \ .$$

[1]We denote by $\tilde{\mathcal{O}}()$ the order of the exponential factors appearing in the complexity and omit logarithmic factors.

The last join then searches the solution comparing about $C_l = \mathcal{O}\left(|L_{23}| \cdot |L_{1S}| \cdot 2^{\lambda - mt}\right)$ elements. Altogether, the total time complexity is

$$\mathcal{T} = \tilde{\mathcal{O}}\left(\max(|L_1|, |L_2|, |L_3|, |S|, C_{1S}, C_{23}, |L_{23}|, C_l\right) \ .$$

As we want to find at least one element in the final list $|L|$, we obtain a lower limit on the number of syndromes in S given by the limits on the parameters ω_i so that

$$|S| = \mathcal{O}\left(\frac{2^{r+\lambda}}{\binom{n}{\omega_2+\omega_3}\binom{n}{\omega_1}}\right) \ .$$

We assume here that we create all elements at the bottom level and store about $\binom{n}{\omega_2+\omega_3} \cdot 2^{-\lambda}$ elements in $|L_{23}|$. Notice that the size of the lists L_1; L_{23} is in this way related to the minimal number of syndromes. If $|S|$ is large, we can create only a part of the elements in the starting lists and reduce S.

In practice, we will choose all ω_i close to $\omega/3$:

$$\omega_1 = \lfloor \binom{n}{\omega/3} \rfloor, \quad \omega_2 = \lceil \binom{n}{\omega/3} \rceil, \quad \omega_3 = w - \omega_1 - \omega_2 \ .$$

The restrictions λ are then chosen due to the following observation. We distinguish two cases. For fixed r, m, ω, ω_i, the complexity may either be dominated by the time to create the (fictive) list L_{1S} or by the size of the lists S, L_{23}. In the first case, the longest time spend will be of order

$$|L_{1S}| = \mathcal{O}\left(2^r / \binom{n}{\omega_2+\omega_3}\right)$$

and we minimize the storage use by choosing

$$\lambda = \lfloor \log_2 (\binom{n}{\omega_2+\omega_3} / \binom{n}{\omega_{max}}) \rfloor$$

where $\omega_{max} = max_i(\omega_i)$. In the second case, we attain the optimal situation if we set $\lambda = \lfloor 0.5 \cdot \log_2 (\binom{n}{\omega_2+\omega_3}^2 \binom{n}{\omega_1} / 2^r) \rfloor$ balancing the size of the lists in this way. The necessary memory can further be reduced by creating L_3 on the fly and creating not all possible $\binom{n}{\omega_1}$ elements. The overall time and memory complexity are

$$\mathcal{T} = T \cdot \log(T) \text{ and } \mathcal{M} = M \cdot \log(M)$$

where

$$\begin{aligned} T &= M = \mathcal{O}\left(\sqrt[2]{2^r / \binom{n}{\omega_1}}\right) && \text{if } 2^r / \binom{n}{\omega_2+\omega_3} > \sqrt[2]{2^r / \binom{n}{\omega_1}} \\ T &= \mathcal{O}\left(2^r / \binom{n}{\omega_2+\omega_3}\right) \text{ and } M = \mathcal{O}\left(\binom{n}{\omega_{max}}\right) && \text{else.} \end{aligned}$$

Preventing GBA. An alternative to the standard CFS signature is to create two (or more) dependant signatures of the same document using different hash functions. This version is called parallel-CFS [Fin10]; it is more efficient than augmenting the parameters in CFS for the same level of security. The drawback of the technique is that it doubles the signature length and the time for signing and verification. (A similar technique was presented in [NPS01].) Now, an GBA has to find two signatures for the same document in asymptotic time complexity $\mathcal{O}\left(2^{mt(3/7+o(1))}\right)$.

Chapter 9

Information-set decoding by classical means

To solve an instance of the syndrome-decoding problem over a binary linear code, one has to find a set of ω columns of a given matrix $H \in \mathbb{F}_2^{n-k\times n}$ that sum up to a syndrome $\mathbf{s}$ over $\mathbb{F}_2^{n-k}$. This is equivalent to say that we search a binary vector[1] $\mathbf{e} \in \mathbb{F}_2^n$ such that $H\mathbf{e} = \mathbf{s}$. The syndrome depends on a message $\mathbf{x}$, we want to recover. We hence sometimes write $s(\mathbf{x})$ for $\mathbf{s}$. A brute-force algorithm would require to compute $\binom{n}{\omega}$ column sums. A more efficient approach is called *information-set decoding*. It was already mentioned in the original work by McEliece [McE78] and is inspired by the work of Prange [Pra62].

information-set decoding proceeds in two mayor steps, an initial transformation step and a search step. Both steps are executed repeatedly until the algorithm succeeds. The initial transformation randomly permutes the columns of H. In particular, this permutes the ω columns of H that sum to $s(\mathbf{x})$, and thus permutes the coordinates of $\mathbf{e}$. Then we apply Gaussian elimination on the rows of H in order to obtain a systematic form

$$H' = [Q \mid I_{n-k}]$$

where $Q \in \mathbb{F}_2^{(n-k)\times k}$ and I_{n-k} is the $(n-k)$-dimensional identity matrix.

Researchers have proposed many techniques to lower the cost of the init phase. We will shortly review them in the following. Our focus lies however on the search phase. Section 9.1 presents previous techniques. We can considerably speed-up the running time by applying the extended representation technique as we explain in chapter 10.

Init phase. At each iteration of the ISD algorithm one chooses an information set, a random set of $n-k$ columns of H, and applies row operations to bring H into systematic form. Instead of starting each time from the original parity check matrix, one can exchange only some columns in the previous information set [CC94, CC98, BLP08]. Also the Gaussian elimination can be sped-up by adding the sum of rows to other rows [Bar06, BLP08] to obtain zeros in the first columns. Other improvements emerge from reuse of previous column additions [BLP08].

[1]Vectors are denoted as column vectors in the following.

9.1. Techniques for the search phase - A short survey

Let H' be in systematic form:

$$H' = [Q \mid I_{n-k}]$$

where $Q \in \mathbb{F}_2^{(n-k)\times k}$ and I_{n-k} is the $(n-k)$-dimensional identity matrix. The Gaussian elimination operations are also applied to $s(\mathbf{x})$ which results in $\tilde{s}(\mathbf{x})$.

Lee and Brickell. Let us fix an integer $p < \omega$. We assume that the permutation was such that the corresponding error vector has few p non-zero coordinates in its first k coordinates and that most of its weight is in the last $n-k$ positions:

$$\tilde{\mathbf{e}} = (\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2)$$

where $\mathrm{wt}(\tilde{\mathbf{e}}_1) = p$ and $\mathrm{wt}(\tilde{\mathbf{e}}_2) = \omega - p$. This approach is due to Lee and Brickell [LB88]. In the search step of ISD, we try to find $\tilde{\mathbf{e}}_1$ by an exhaustive search. We compute for every linear combination of p columns from Q its Hamming distance to $\tilde{\mathbf{s}}$: $\mathrm{wt}(Q\tilde{\mathbf{e}}_1 + \mathbf{s})$. If the distance is exactly $\omega - p$, we can add to our p columns those $\omega - p$ unit vectors from I_{n-k} that exactly yield $\tilde{s}(\mathbf{x})$. Undoing the Gauss elimination recovers the desired error vector $\mathbf{e}$. Obviously, we can only succeed if the initial column permutation results in a permuted $\mathbf{e}$ that has exactly p ones in its first k coordinates and $\omega - p$ ones in its last $n-k$ coordinates. The probability for this event is

$$\mathcal{P}_{LB} = \frac{\binom{k}{p}\binom{n-k}{\omega-p}}{\binom{n}{\omega}}$$

such that $\mathcal{P}_{LB}^{-1}$ corresponds to the average number of iterations. Per run, we need to enumerate

$$\mathcal{T}_{LB} = \binom{k}{p}$$

column sums that we check immediately for the right distance to the target. The overall time is then

$$\mathcal{T}_{LB} \cdot \mathcal{P}_{LB}^{-1} \ .$$

The space requirement is constant.

Optimization of p leads to a running time of $2^{0.05752n}$ in the worst case, that is, for k/n that maximizes the time.

Leon and Stern. Leon observed that one can speed up Lee–Brickell's algorithm by considering only the vectors who have a block of ℓ bits that are zero , $\ell < n-k$. The probability for such an event is

$$P_{Leon} = \frac{\binom{k}{p}\binom{n-k-\ell}{w-p}}{\binom{n}{w}} \ .$$

Stern [Ste89] observed that one can further improve on the running time when replacing in the search step the brute-force search for weight-p linear combinations by a collision search. Dumer proposed [Dum98] the same idea independently and derived a similar algorithm [Dum91, Bar98]. Stern proposes to search for a vector that has few p non-zero positions in the first k coordinates, an ℓ-block of zeros and most of its weight in the last $n-k-\ell$ positions as illustrated in figure 9.1.

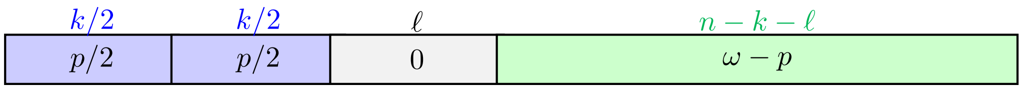


Figure 9.1.: *Weight distribution of $\tilde{e}$ and classical concatenation by Stern.*

Consider the projection of $Q\tilde{\mathbf{e}}$ to its first ℓ coordinates, denoted by $(Q\tilde{\mathbf{e}})_{[\ell]}$. Due to the special structure of $\tilde{\mathbf{e}}$, it holds that

$$(Q\tilde{\mathbf{e}})_{[\ell]} = Q_{[\ell]}\tilde{\mathbf{e}}_1$$

where $\tilde{\mathbf{e}}_1$ denotes the first ℓ coordinates of $\tilde{\mathbf{e}}$. This means that after the search for $\tilde{\mathbf{e}}_1$, the vector matches the target already on ℓ coordinates.

To find the p columns of $Q_{[\ell]}$ that sum up to $\tilde{\mathbf{s}}_{[\ell]}$, we split $Q_{[\ell]}$ into two disjoint column sets

$$Q_1 = \{\mathbf{q}_i \,|\, i \in [1, \frac{k}{2}]\} \text{ and } Q_2 = \{\mathbf{q}_i \,|\, i \in [\frac{k}{2}+1, k]\}$$

each of size $k/2$ that are indexed by I_1 and I_2, respectively. We can write the collision problem as

$$\sum_{i \in I_1} \mathbf{q}_i = \tilde{\mathbf{s}}_{[\ell]} + \sum_{i \in I_2} \mathbf{q}_i \tag{9.1}$$

where $I_1 \subset \left[1, \frac{k}{2}\right]$, $I_2 \subset \left[\frac{k}{2}+1, k\right]$ and $|I_1| = |I_2| = \frac{p}{2}$. We create two lists $\mathcal{L}_1, \mathcal{L}_2$ that contain all possible sums of the left and right side of (9.1), respectively. The lists are of size $L = \binom{(k+\ell)/2}{p/2}$. A collision between the lists is equivalent to a set $I = I_1 \cup I_2$ of p columns that sum up to the syndrome on the first ℓ coordinates. We expect to find $L^2/2^\ell$ collisions. If the remaining coordinates differ from $\tilde{s}(\mathbf{x})$ by a weight-$(\omega - p)$ vector, we can correct these positions by adding suitable unit vectors from I_{n-k}.

The running time of each iteration is given by the size of the lists and the number of collisions (section 1.2):

$$\mathcal{T}_{Stern} = \max(\binom{k/2}{p/2}, \frac{\binom{k/2}{p/2}^2}{2^\ell}) \ .$$

Multiplied by the inverse of the probability that the initial phase chose the right permutation:

$$\mathcal{P}_{Stern} = \frac{\binom{k/2}{p/2}^2 \binom{n-k-\ell}{\omega-p}}{\binom{n}{\omega}} \ ,$$

we obtain the total running time:

$$\mathcal{T}_{Stern} \cdot \mathcal{P}_{Stern}^{-1} \ .$$

Under the constraints that $0 \leq p \leq \omega$ and $0 \leq \ell \leq n-k-\omega+p$, we can minimize the running time. The worst-case time complexity is $2^{0.05562\,n}$ where optimal parameters are $p = 0.0031\,n$ and $\ell = 0.0134\,n$. The memory is of order $2^{0.0134\,n}$.

Ball-collision decoding. The ball-collision technique of Bernstein, Lange and Peters [BLP11] from 2011 lowers the complexity by allowing few, q, ones in the ℓ-block. They enumerate all possible $p/2$-column sums within the intervals $[1, k/2]$ and $[k/2+1, k]$ as in Stern's algorithm. Additionally, the algorithm computes all $q/2$-column sums within the intervals $[k+1, k+\ell/2]$ and $[k+\ell/2+1, k+\ell]$. The number of all combined elements per list is $S_{Ball} = \binom{k/2}{p/2}\binom{l/2}{q/2}$. Analogous to the previous analysis, we obtain an asymptotic time

$$\mathcal{T}_{Ball} = \max(S_{Ball}, \frac{S_{Ball}^2}{2^\ell}) \ .$$

The probability is slightly increased:

$$\mathcal{P}_{Ball} = \frac{\binom{k/2}{p/2}^2 \binom{l/2}{q/2}^2 \binom{n-k-\ell}{\omega-p-q}}{\binom{n}{\omega}} \ .$$

The overall time complexity is $\mathcal{T}_{Ball} \cdot \mathcal{P}_{Ball}^{-1}$. The worst-case occurs for $k/n = 0.4548$ where it takes the value $2^{0.05559\,n}$ using $2^{0.139\,n}$ space. The optimal parameters are $p = 0.032n$, $\ell = 0.139n$ and $q = 9.6E-05n$.

Ball-collision decoding is very similar to a variant of ISD proposed by Finiasz and Sendrier [FS09] in 2009. Both algorithms as well as Dumer's algorithm [Dum91, Bar98] have the same asymptotic worst-case complexity.

Finiasz and Sendrier. Finiasz and Sendrier [FS09] proposed to transform H into quasi-systematic form

$$\tilde{H} = \left[Q \mid \begin{matrix} 0 \\ I_{n-k-\ell} \end{matrix} \right]$$

with $Q \in \mathbb{F}_2^{(n-k)\times(k+\ell)}$. The lists $\mathcal{L}_1, \mathcal{L}_2$ each contain all weight-$\frac{p}{2}$ sums out of $\frac{k+\ell}{2}$ columns. The collision problem we need to solve is the following: Find index sets $I_1 \subset [1, \frac{k+\ell}{2}]$ and $I_2 \subset \left[\frac{k+\ell}{2}+1, k+\ell\right]$ of size $p/2$ such that

$$\sum_{i \in I_1} \mathbf{q}_i = \tilde{\mathbf{s}}_{[\ell]} + \sum_{i \in I_2} \mathbf{q}_i \ . \tag{9.2}$$

The lists are of size $S_{FS} = \binom{(k+\ell)/2}{p/2}$ which represents the used memory. The time per iteration is given by

$$\mathcal{T}_{FS} = \max(S_{FS}, \frac{S_{FS}^2}{2^\ell}) \ .$$

The probability that the error vector has the required weight distribution is

$$\mathcal{P}_{FS} = \frac{\binom{(k+\ell)/2}{p/2}^2 \binom{n-k-\ell}{\omega-p}}{\binom{n}{\omega}} \ .$$

We remark that for $\ell > 0$ the probability is larger than $\mathcal{P}_{Stern}$ which in return means that we need to perform less iterations on average. The overall time complexity is $\mathcal{T}_{FS} \cdot \mathcal{P}_{FS}^{-1}$. The worst-case is the same as for Ball-collision decoding. For an information rate $k/n = 0.4548$ the time is $2^{0.05559\,n}$ using $2^{0.139\,n}$ space. The optimal parameters are $p = 0.032n$ and $\ell = 0.139n$.

The setting by Finiasz and Sendrier is known as *generalized* information-set decoding which we present in more detail in the following section. This research as well as ball-collision show that it makes sense to spread the smaller part of the weight of e over a window of $k + \ell$ coordinates and to eliminate the block of only zeros (figure 9.3).

9.2. Generalized information-set decoding

We now give a detailed description of a generalized information-set-decoding (ISD) framework as described by Finiasz and Sendrier [FS09] in 2009. Recall that the input to an ISD algorithm is a tuple $(H, \mathbf{s})$ where $H \in \mathbb{F}_2^{(n-k)\times n}$ is a parity check matrix of a random linear $[n, k, d]$-code and $\mathbf{s} = H\mathbf{e}$ is the syndrome of the unknown error vector $\mathbf{e}$ of weight $\omega := \mathrm{wt}(\mathbf{e}) = \lfloor \frac{d-1}{2} \rfloor$.

ISD is a randomized algorithm that iterates two mayor steps until the solution $\mathbf{e}$ is found. The first part is a linear transformation of the parity check matrix H depending on a random permutations of the columns. The second step is a search phase.

During the initial transformation, we permute the columns of H by multiplying with a random permutation matrix $P \in \mathbb{F}_2^{n\times n}$. Then we perform Gaussian elimination on the rows of HP by multiplying with an invertible matrix $T \in \mathbb{F}_2^{(n-k)\times(n-k)}$. This yields a parity check matrix $\tilde{H} = THP$ in quasi-systematic form containing a 0-submatrix in the right upper corner as illustrated in figure 9.2. We denote by Q^I the projection of Q to the rows defined by the index set $I \subset \{1, \ldots, n-k\}$. Analogously, we denote by Q_I the projection of Q to its columns. In particular we define $[\ell] := \{1, \ldots, \ell\}$ and $[\ell, n-k] := \{\ell, \ldots, n-k\}$. We denote the initial transformation $\mathsf{Init}(H) := THP$.

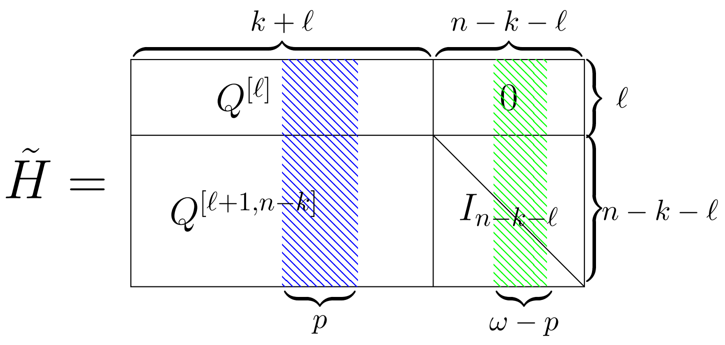


Figure 9.2.: *Parity check matrix $\tilde{H}$ in quasi-systematic form.*

We set $\tilde{\mathbf{s}} := T\mathbf{s}$ and look for an ISD-solution $\tilde{\mathbf{e}}$ of $(\tilde{H}, \tilde{\mathbf{s}})$, i.e., we look for an $\tilde{\mathbf{e}}$ satisfying $\tilde{H}\tilde{\mathbf{e}} = \tilde{\mathbf{s}}$ and $\mathrm{wt}(\tilde{\mathbf{e}}) = \omega$. This yields a solution $\mathbf{e} = P\tilde{\mathbf{e}}$ to the original problem. Notice that applying the permutation matrix to $\tilde{\mathbf{e}}$ leaves the weight unchanged, i.e., $\mathrm{wt}(\mathbf{e}) = \omega$, and $TH\mathbf{e} = \tilde{H}\tilde{\mathbf{e}} = \tilde{\mathbf{s}} = T\mathbf{s}$ implies $H\mathbf{e} = \mathbf{s}$ as desired. In the search phase, we try to find all error vectors $\tilde{\mathbf{e}}$ that have a specific weight distribution, i.e., we search for vectors that can be decomposed into $\tilde{\mathbf{e}} = (\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2) \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{n-k-\ell}$ where $\mathrm{wt}(\tilde{\mathbf{e}}_1) = p$ and $\mathrm{wt}(\tilde{\mathbf{e}}_2) = \omega - p$. Figure 9.3 shows the distribution of the ones in $\tilde{\mathbf{e}}$. Since P shuffles $\mathbf{e}$'s coordinates into random

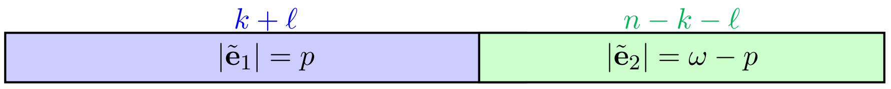

$k+\ell$
$n-k-\ell$
$|\tilde{\mathbf{e}}_1| = p$
$|\tilde{\mathbf{e}}_2| = \omega - p$

Figure 9.3.: *Weight distribution of* $\tilde{\mathbf{e}}$.

positions, $\tilde{\mathbf{e}}$ has the above weight distribution with probability

$$\mathcal{P} = \frac{\binom{k+l}{p}\binom{n-k-l}{\omega-p}}{\binom{n}{\omega}} \quad . \tag{9.3}$$

The inverse probability $\mathcal{P}^{-1}$ is the expected number of repetitions we need to perform until $\tilde{\mathbf{e}}$ has the desired distribution.

Due to the systematic form of $\tilde{H}$, we see that

$$\tilde{H}\tilde{\mathbf{e}} = \begin{bmatrix} Q^{[\ell]}\tilde{\mathbf{e}}_1 \\ Q^{[\ell+1,n-k]}\tilde{\mathbf{e}}_1 + \tilde{\mathbf{e}}_2 \end{bmatrix} = \tilde{\mathbf{s}} \ .$$

This allows us to search at first for candidates $\tilde{\mathbf{e}}_1$ and to set $\tilde{\mathbf{e}}_2$ to match the syndrome.

First, we search the truncated vector $\tilde{\mathbf{e}}_1 \in \mathbb{F}_2^{k+\ell}$ that represents the position of the first p ones. For the computation of $\tilde{\mathbf{e}}_1$ we focus on the submatrix $Q^{[\ell]} \in \mathbb{F}_2^{\ell\times(k+\ell)}$. Since we fixed the 0-submatrix in the right-hand part of $\tilde{H}$, we ensure that $Q^{[\ell]}\tilde{\mathbf{e}}_1$ exactly matches the syndrome $\tilde{\mathbf{s}}$ on its first ℓ coordinates.

Having found candidates $\tilde{\mathbf{e}}_1$, we recover the full error vector $\tilde{\mathbf{e}} = (\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2)$, the missing coordinates $\tilde{\mathbf{e}}_2$ are obtained as the last $n-k-\ell$ coordinates of $Q\tilde{\mathbf{e}}_1 + \tilde{\mathbf{s}}$.

The goal in the ISD search phase is to compute the truncated error vector $\tilde{\mathbf{e}}_1$ efficiently. Finding an $\tilde{\mathbf{e}}_1$ with such a property is called the *submatrix matching problem* [MMT11].

Definition 9.1 (Submatrix Matching Problem)
Given a random matrix $Q \in_R \mathbb{F}_2^{\ell\times(k+\ell)}$ *and a target vector* $\mathbf{s} \in \mathbb{F}_2^{\ell}$*, the* submatrix matching problem (SMP) *consists in finding a set* I *of size* p *such that the corresponding columns of* Q *sum up to* $\mathbf{s}$*, i.e., to find* $I \subseteq [1, k+\ell]$*,* $|I| = p$ *such that*

$$\sigma(Q_I) := \sum_{i\in I} q_i = \mathbf{s}, \ \textit{where } q_i \textit{ is the } i\textit{-th column of } Q.$$

Note that the SMP itself can be seen as just another syndrome-decoding instance with parity check matrix Q, syndrome $\mathbf{s} \in \mathbb{F}_2^{\ell}$ and parameters $[k+\ell, \ell, p]$.

We already saw in the previous section several ideas how to obtain $\tilde{\mathbf{e}}_1$ in a classical way by exhaustive or collision search. The next chapter presents a new approach that uses the simple or extended representation technique from chapter 4 and 5.

Let ColumnMatch be the algorithm chosen in the search phase. We describe the ISD in pseudocode in algorithm 9.1. The initial phase permutes some columns of the parity check matrix and perform Gaussian elimination to bring the matrix in quasi-systematic form. The running time is polynomial in n. The last step goes through all elements that are found by ColumnMatch and needs to perform $|\mathcal{L}|p$ column additions to check for the weight. Both parts represent low cost in comparison to the search phase that builds the list $\mathcal{L}$ and the number of iterations.

Algorithm 9.1: Generalized ISD

```
Input:      Parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, syndrome $\mathbf{s} = H\mathbf{e}$ with $\mathrm{wt}(\mathbf{e}) = \omega$
Output:     Error $\mathbf{e} \in \mathbb{F}_2^n$
Parameters: $p$, $\ell$

Repeat ▷ about $\mathcal{P}^{-1}$ times
    Compute $\tilde{H} \leftarrow \mathsf{Init}(H)$ and $\tilde{\mathbf{s}} \leftarrow T\mathbf{s}$.
    Compute $\mathcal{L} = \textsc{ColumnMatch}(Q^{[\ell]}, \tilde{\mathbf{s}}_{[\ell]}, p)$
    For each Solution $\tilde{\mathbf{e}}_1 \in \mathcal{L}$
        If $\mathrm{wt}(Q\tilde{\mathbf{e}}_1 + \tilde{\mathbf{s}}) = \omega - p$ then
            Compute $\tilde{\mathbf{e}} \leftarrow (\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2) \in \mathbb{F}_2^n$ where $\tilde{\mathbf{e}}_2 \leftarrow (Q\tilde{\mathbf{e}}_1 + \tilde{\mathbf{s}})_{[\ell+1,n-k]}$
            Output $\mathbf{e} = \tilde{\mathbf{e}}P$.
```

Let $\mathcal{T} := \mathcal{T}(n, R; p, \ell)$ denote the running time of ColumnMatch. The average running time of algorithm 9.1 is $\mathcal{P}^{-1} \cdot \mathcal{T}$.

9.3. Complexity of generic decoding algorithms for random linear codes

The running time of decoding algorithms for linear codes is a function of the three code parameters: the code length n, the dimension k and the minimal distance d. With overwhelming probability random binary linear codes attain an information rate $R := \frac{k}{n}$ which is close to the Gilbert Varshamov bound $1 - h(D)$ (section 8.1) where $D := \frac{d}{n}$ and h is the binary entropy function.

We distinguish between bounded-distance and full-distance decoding which determines the integer ω in the decoding problem. For *bounded-distance decoding*, we set $W := \omega/n = D/2$ which is the setting of the syndrome-decoding problem 8.1.

For *full decoding*, in the worst case we need to decode a highest weight coset leader of the code C, its weight ω corresponds to the *covering radius* of C which is defined as the smallest radius r such that C can be covered by discrete balls of radius r. The Goblick bound [TG62] ensures that $r \geq nH^{-1}(1 - R) + o(n)$ for *all* linear codes. Independently, Blinovskii [Bli87] and Levitin [Lev88] further proved that this bound is tight for *almost all* linear codes, i.e., $r = nH^{-1}(1-R) + o(n)$. This justifies our choice $W = H^{-1}(1-R)$ for the full decoding scenario.

We can thus express the running time $\mathcal{T}(n, R)$ as a function in n and R only. One usually measures the complexity of decoding algorithms asymptotically in the code length n. Since all generic decoding algorithms run in exponential time, a reasonable metric is the complexity coefficient $F(R)$ as defined in [CG90]:

$$F(R) = \lim_{n\to\infty} \frac{1}{n} \log \mathcal{T}(n, R)$$

which suppresses polynomial factors since $\lim \frac{1}{n} \log p(n) = 0$ for any polynomial $p(n)$. With this notation, the asymptotic running time of a generic decoding algorithm (such as algorithm 9.1) is

$$\mathcal{T}(n, R) = 2^{nF(R)+o(n)} \leq 2^{n\lceil F(R)\rceil_\rho}$$

for large enough n. We obtain the worst-case complexity for the information rate R that maximizes $F(R)$, that is, by taking $\max_{0<R<1}\lceil F(R)\rceil_\rho$. Here, $\lceil x\rceil_\rho := \lceil x \cdot 10^\rho\rceil \cdot 10^{-\rho}$ denotes rounding up $x \in \mathbb{R}$ to a certain number of $\rho \in \mathbb{N}$ decimal places.

The classical ISD algorithms as presented in section 9.1 obtain an asymptotic worst case complexity as presented in table 9.1. We distinguish between bounded-distance and full-distance decoding, that is, where $W = D/2$ and $W = D$, respectively.

Algorithm	half-dist.		full dec.	
	time	space	time	space
Lee-Brickell	0.05752	-	0.1208	-
Stern	0.05564	0.0135	0.1167	0.0318
Ball-collision	0.05559	0.0148	0.1164	0.0374

Table 9.1.: *Comparison of worst-case complexity coefficients, i.e., the time columns represent the maximal complexity coefficient $F(R)$ for $0 < R < 1$.*

Chapter 10

Improved information-set decoding

We adopt the setting of generalized information-set decoding which we introduced in section 9.2. Let $\tilde{H}$ be a parity check matrix in quasi-systematic form $\left[Q \mid {}^{\;0}_{I_{n-k-\ell}}\right]$ (like in figure 9.2) where $Q \in \mathbb{F}_2^{(n-k)\times(k+\ell)}$ is a random matrix. We seek to reduce the running time needed to solve a decoding problem by information-set decoding (ISD) w.r.t. $\tilde{H}$ and a syndrome $\tilde{\mathbf{s}}$. In this section, we deal with the subproblem of finding a vector $\tilde{\mathbf{e}}_1 \in \mathbb{F}_2^{k+\ell}$ of weight p such that $Q^{[\ell]}\tilde{\mathbf{e}}_1 = \tilde{\mathbf{s}}_{[\ell]}$ where $Q^{[\ell]}$ are the upper ℓ rows of Q.

The running time of an ISD algorithm depends essentially on the time to find $\tilde{\mathbf{e}}_1$ and the probability $\mathcal{P}$ that the vector is a part of the scrambled solution to the original decoding problem 9.3. Whenever the search phase for $\tilde{\mathbf{e}}_1$ does not reveal the solution, we need to repeat with a permuted matrix. The average number of repetitions is given by the inverse of $\mathcal{P}$.

For a simpler notation, we redefine this decoding problem in the present chapter by setting $Q = Q^{[\ell]}$, $\mathbf{e} = \tilde{\mathbf{e}}_1$ and $\mathbf{s} = \tilde{\mathbf{s}}_{[\ell]}$. Notice that finding a sum of p columns of Q that exactly matches $\mathbf{s}$ is a vectorial version of the subset-sum problem over $\mathbb{F}_2$. The present chapter shows how we can apply the most efficient techniques to solve the subset-sum problem (chapters 4 and 5) in order to solve the column match problem. A faster subroutine to find the truncated vector $\tilde{\mathbf{e}}_1$ then improves the overall running time of information-set decoding (algorithm 9.1).

This chapter is an extended version of the paper [BJMM12] by B., Joux, Meurer and May. It is organized as follows. We briefly describe the application of the *simple representation technique* on ISD in section 10.1. It was presented by May, Meurer and Thomae [MMT11] and discovered at the same time by Johansson and Löndahl [JL11]. We propose a method to reduce the memory requirement of the original algorithm in section 10.1.2. The following section 10.2 explains in detail how to extend the approach by use of the *extended representation technique*. We develop the new algorithm that solves the submatrix matching problem and present a complexity analysis. Subsection 10.2.4 shows how to reduce the memory requirement. We provide parameters and complexity results derived from a numerical optimization for bounded-distance and full-distance decoding. Section 10.2.3 compares the asymptotic running time of the presented ISD algorithms and shows that the running time can be decreased considerably by our technique. An implementation of our algorithm reveals that the subtask of an efficient collision search is important in practice. We propose a variant of a merge-join routine in section 10.4 and present the results of our experiments in section 10.5. We provide a theoretical analysis for cases in which our algorithm will exceed the claimed complexity in section 10.6 and present a provable version of our algorithm in section 10.7.

10.1. Simple representation technique in the search phase

The collision search problem in the classical approach (section 9.1) searches disjoint index sets $I_1 \subset [1, (k+\ell)/2]$ and $I_2 \subset [(k+\ell)/2, k+\ell]$ of size $|I_1| = |I_2| = \frac{p}{2}$ such that

$$\sum_{i \in I_1} \mathbf{q}_i = \sum_{i \in I_2} \mathbf{q}_i + \mathbf{s} \ . \tag{10.1}$$

This ensures that the combined set $I = I_1 \cup I_2$ has exactly size p.

Already in [FS09, Appendix A], the authors consider overlapping sets. May, Meurer and Thomae [MM11] provide a detailed description of the best strategy and show that the running time for information-set decoding can be reduced in comparison to previous attempts at the cost of an increased memory requirement. Johansson and Löndahl [JL11] present the same idea. Using the representation technique, one chooses I_1 and I_2 no longer from half-sized intervals but they both are chosen from the whole interval $[1, k+\ell]$ such that $I_1 \cap I_2 = \emptyset$. The vector $\mathbf{e}$ is no longer built as a concatenation but as a sum of two vectors $\mathbf{e}_i \in F_2^{k+\ell}$ of half weight: $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$. Figure 10.1 illustrates the idea.

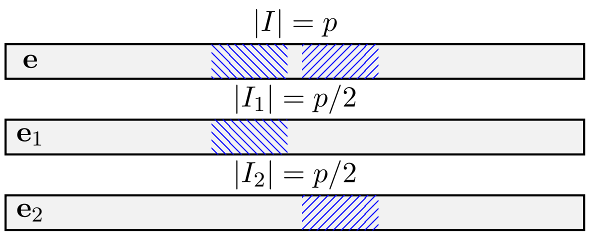


Figure 10.1.: *Decomposition of* $\mathbf{e}$ *as sum of vectors* $\mathbf{e}_1$ *and* $\mathbf{e}_2$*.*

Thus, every solution I admits many such representations (I_1, I_2) and equivalently every solution vector $\mathbf{e}$ permits representations $(\mathbf{e}_1, \mathbf{e}_2)$. The number of ways to pick $p/2$ out of p columns determines the number of representations:

$$N_{MMT} = \binom{p}{p/2} \ .$$

Note that the $N_{MMT} \approx 2^p$ which is exponential in n and that all these representations obviously lead to the same solution. A classical collision search in contrast searches a unique pair of sets I_1, I_2 indexing either the first half or second half of columns of $Q^{[\ell]}$.

An algorithm could enumerate all possible $p/2$-column sums out of $k+\ell$ columns in two lists $\mathcal{L}_1, \mathcal{L}_2$ where to all elements in the second list we additionally add $\mathbf{s}$. It then applies a collision search to find same column sums. We will output all colliding index sets $I = I_1 \cup I_2$ of size p. We would even find the solution N_{MMT} times in this way.

This first approach is very expensive as we increase the number of created elements in $\mathcal{L}_1, \mathcal{L}_2$ from $\binom{(k+\ell)/2}{p/2}$ to $\binom{k+\ell}{p/2}$ compared to Stern's algorithm. We have not made use of the fact that there are exponentially many representations in the colliding elements $\mathcal{L}_1 \bowtie \mathcal{L}_2$.

We can overcome this problem as follows. Let us assume that the column sums are equally distributed vectors in $\mathbb{F}_2^\ell$. This is a reasonable assumption as we assume that the given matrix is indistinguishable from a random matrix. Let $0 \le r \le \ell$ be a parameter. The probability that

$$(\sum_{i \in I_1} \mathbf{q}_i)_{[r]} = \mathbf{t} \tag{10.2}$$

for some random vector $\mathbf{t} \in \mathbb{F}_2^r$ is then 2^{-r}. If we choose the parameter

$$r \approx \log_2(N_{MMT}) \ ,$$

we can expect that on average one I_1, belonging to a representation (I_1, I_2), satisfies (10.2). Consequently, $I \setminus I_1$ satisfies

$$(\sum_{i \in I \setminus I_1} \mathbf{q}_i)_{[r]} = \mathbf{s}_{[r]} + \mathbf{t} \ . \tag{10.3}$$

Conversely, for arbitrary index sets $(J_1, J_2) \in \mathcal{L}_1 \times \mathcal{L}_2$ that fulfil (10.2) and (10.3), we know that

$$(\sum_{i \in J_1 \cup J_2} \mathbf{q}_i)_{[r]} = \mathbf{s}_{[r]} \ .$$

It remains to check if equality hold on all ℓ coordinates and if $J_1 \cap J_2 = \emptyset$ in which case we have found the solution $I = J_1 \cup J_2$ and (J_1, J_2) is one out the many representations. For an unlucky guess of $\mathbf{t}$, no representation might fulfil (10.2) and (10.3). We then need to repeat with a changed target.

10.1.1. A simple representation technique algorithm

An algorithm along these lines executes the following steps. Pick a target vector $\mathbf{t} \in \mathbb{F}_2^r$ at random. Create $p/2$-column sums out of $k+\ell$ columns of Q that sum up to $\mathbf{t}$ or $\mathbf{s}_{[r]} + \mathbf{t}$ on its first r coordinates. This is equivalent to enumerating vectors $\mathbf{e}_1, \mathbf{e}_2 \in \mathbb{F}_2^{k+\ell}$ of weight $p/2$ such that

$$(Q\mathbf{e}_1)_{[r]} = \mathbf{t} \tag{10.4}$$

and

$$(Q\mathbf{e}_2)_{[r]} = \mathbf{s}_{[r]} + \mathbf{t} \ . \tag{10.5}$$

The elements are enumerated into two lists:

$$\begin{aligned}\mathcal{L}_1 &= \{\mathbf{e}_1 \in \mathbb{F}_2^{k+\ell} \mid \text{wt}(\mathbf{e}_1) = \frac{p}{2} \text{ and } (Q\mathbf{e}_1)_{[r]} = \mathbf{t}\} \text{ and}\\ \mathcal{L}_2 &= \{\mathbf{e}_2 \in \mathbb{F}_2^{k+\ell} \mid \text{wt}(\mathbf{e}_2) = \frac{p}{2} \text{ and } (Q\mathbf{e}_1)_{[r]} = \mathbf{s}_{[r]} + \mathbf{t}\}\end{aligned}$$

of expected size

$$L = \mathbb{E}\left[|\mathcal{L}_1|\right] = \mathbb{E}\left[|\mathcal{L}_2|\right] = \frac{\binom{k+\ell}{p/2}}{2^r} = \frac{\binom{k+\ell}{p/2}}{N_{MMT}} \quad . \tag{10.6}$$

We then call an efficient join routine that searches for collisions over all ℓ coordinates, i.e.,

$$Q(\mathbf{e}_1 + \mathbf{e}_2) = s \ ,$$

and checks if the colliding elements have correct weight: $\text{wt}(\mathbf{e}_1 + \mathbf{e}_2) = p$. The join can be performed as described in section 1.2 by sorting the elements in lexicographical order and passing through the sorted lists. The complexity is $\max(L, C, N_{Sol})$ in time and L in memory where C is the number of collisions that occur during the collision search and N_{Sol} is the final number of solutions. For uniformly distributed elements, the probability for a match on all ℓ coordinates is $2^{-\ell+r}$. We expect that $C \approx \frac{L^2}{2^{n-r}}$.

So far, we have not explained how to create the lists $\mathcal{L}_1, \mathcal{L}_2$. We invoke a classical strategy and build the elements $\mathbf{e}_1 \in \mathcal{L}_1$ and $\mathbf{e}_2 \in \mathcal{L}_2$ from elements of half length and half weight. That is we split the columns of Q into two sets: $Q = [Q_1|Q_2]$ and create lists:

$$\mathcal{B}_1 = \{(\mathbf{x}, Q_1\mathbf{x}) \,|\, \mathbf{x} \in \mathbb{F}_2^{(k+\ell)/2}, \text{wt}(\mathbf{x}) = \frac{p}{4}\} \text{ and} \mathcal{B}_2 = \{(\mathbf{y}, Q_2\mathbf{y}) \,|\, \mathbf{y} \in \mathbb{F}_2^{(k+\ell)/2}, \text{wt}(\mathbf{y}) = \frac{p}{4}\}$$

of size $B = \binom{(k+\ell)/2}{p/4+\varepsilon/2}$. The number of elements $\mathbf{e}_i = [x \,|\, y]$ where $(\mathbf{x}, \mathbf{y}) \in \mathcal{B}_1 \times \mathcal{B}_2$ and which satisfy (10.4) or (10.5) can be estimated as $C_B \approx B^2/2^r$. The cost to create the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ by a merge-join routine is then $\max(B, C_B, L)$ in time and $\max(B, L)$ in space. The overall running time to find the solution is $\mathcal{T}_{MMT} = \max(B, C_B, L, C, N_{Sol})$ using a memory $\max(L, B)$.

Assume that the given submatrix matching problem $Q\mathbf{e} = \mathbf{s}$ has a solution $\mathbf{e}$ of weight p. We can only find it in the above presented way, if the ones are equally distributed such that $\mathbf{e}$ has exactly $p/2$ one-coordinates in the intervals $[1, (k+\ell)/2]$ and $[(k+\ell)/2+1, k+\ell]$ as the bottom procedure does not enumerate all possible column sums. It restricts itself to twice $p/4$-column sums. We can precede as in section 3.1.1 where we proposed either a deterministic window method or a probabilistic approach. The probability for that the vector splits exactly as we wish is :

$$\mathcal{P}_{\mathcal{B}} = \frac{\binom{(k+\ell)/2}{p/2}^2}{\binom{k+\ell}{p}} \tag{10.7}$$

which is inverse polynomial in n. Define $c_{k\ell} = (k+\ell)/n$ and $c_p = p/n$. Due to Sterling's formula, we can compute that

$$\mathcal{P}_{\mathcal{B}} \approx \sqrt{\frac{2(k+\ell)}{\pi p(k+\ell-p)}} = \sqrt{\frac{2c_{k\ell}}{\pi c_p(c_{k\ell}-c_p)}}\frac{1}{\sqrt{n}} = \mathcal{O}\left(\frac{1}{\sqrt{n}}\right) \quad .$$

We can choose partitions P_1, P_2 of $[1, k+\ell]$ and create baselists

$$\mathcal{B}_1 = \{\mathbf{x} \in \mathbb{F}_2^{k+\ell} \mid \mathrm{wt}(\mathbf{x}) = \frac{p}{4} \text{ and support}(\mathbf{x}) \subset P_1\}$$

and

$$\mathcal{B}_2 = \{\mathbf{y} \in \mathbb{F}_2^{k+\ell} \mid \mathrm{wt}(\mathbf{x}) = \frac{p}{4} \text{ and support}(\mathbf{y}) \subset P_2\}$$

of size B. If we choose independent random partitions for $\mathcal{L}_1$ and $\mathcal{L}_2$, we can guarantee an independent splitting for their elements of probability $(\mathcal{P}_\mathcal{B})^2$. A merge-join creates elements of weight $p/2$. We repeat the creation of elements for $\mathcal{L}_1$ and $\mathcal{L}_2$ in this way by choosing new partitions each time. Repeating a polynomial number of times, the probability of success to pick the right partition goes exponentially close to 1.

The right choice of the permutation in the initial phase of the information-set-decoding algorithm determines if the submatrix matching problem leads to a solution to the original decoding problem. As we assume that the solution is unique, we will only find e for the right permutation. The probability that the solution to the original decoding problem after permutation has a truncated part $\mathbf{e}$ of weight p and that most of its weight lies in the last $n-k-\ell$ coordinates is $\mathcal{P}$ as given in (9.3). The average number of repetitions of the above algorithm is then $\mathcal{P}^{-1}$.

For small n, we need to set

$$\mathcal{P} = \frac{\binom{(k+\ell)/2}{p/2}^2 \binom{n-k-\ell}{\omega-p}}{\binom{n}{\omega}}$$

due to the disjoint split at the bottom level. For large n, we can assume that $\binom{(k+\ell)/2}{p/2}^2 \approx \binom{k+\ell}{p}$ and use (9.3). The ratio between the two binomials is quickly dropping to zero for increasing n as we showed on the previous page.

Worst case complexity. For optimal parameters $p \approx 0.0064\,n, \ell \approx 0.0279\,n$, the algorithm runs in time

$$\mathcal{T}_{MMT} \cdot \mathcal{P}^{-1} = \tilde{\mathcal{O}}\left(2^{0.05363\,n}\right)$$

for the worst information rate $R = k/n = 0.4639$. The complexity is obtained as explained in section 9.3. The individual cost are: $B \approx 2^{0.0139\,n}$, $C_B \approx L \approx C \approx 2^{0.02146\,n}$ with about $2^{0.0322\,n}$ iterations. Compared to the most efficient classical algorithm, ball-collision decoding, that runs in time $2^{0.05559\,n}$ (section 9.3), we decrease the running time. The memory cost increases however from $2^{0.0148\,n}$ to $2^{0.0215\,n}$.

The worst-case complexity for full-distance decoding occurs for $R = 0.44$ where $W = 0.13094$. For optimal parameters $p \approx 0.0215\,n, \ell \approx 0.0753\,n$, we obtain a running time of about $2^{0.1115\,n}$. The individual cost are: $B \approx 2^{0.0376\,n}$, $C_B \approx L \approx C \approx 2^{0.0538\,n}$ with about $2^{0.0577\,n}$ repetitions.

The results are obtained by the octave code in appendix B. We remark that depending on the used optimization method and precision, the result may vary slightly. In comparison to [MMT11], we find a slightly better worst case running time differing in the fifth decimal position. We can reduce the memory requirement by choosing slightly larger r as we describe in the following section.

The algorithm does not exploit the full potential of the representation technique. We can allow a small proportion of intersections between I_1 and I_2. In this way, we augment the number of representations and the degrees of freedom. Section10.2 describes our new algorithm in detail.

Remark on the complexity notation. We aim to study and improve time and memory requirements in the asymptotic case and hence apply two simplifications when analyzing the complexity. While the actual complexity may be given as a sum of partial cost we can compute the overall cost by the maximal term. We do also neglect polynomial and logarithmic factors by use of the soft-$\mathcal{O}$ notation, denoted by $\tilde{\mathcal{O}}$. It conceals constants as well as logarithmic and polynomial factors in n. Logarithmic and polynomial factors appear in our algorithms due to sorting, addition and storage of the elements as well when we need to repeat to create the bottom lists. We also assume that $2 \mid k + \ell$. The exponential factor in the complexity is however unchanged if the assumption does not hold.

The presented complexity analysis provides thereby a comparison in the asymptotic case and a theoretical recommendation for very large n. In practice, the polynomial and logarithmic factors due to sorting, the number of lists, the storage of elements etc. have to be taken into account and influence the memory cost and time consumption considerably.

10.1.2. Reducing the memory requirement under heuristic assumptions

May, Meurer and Thomae choose r to minimize the lists $\mathcal{L}_1, \mathcal{L}_2$ while still expecting to find one representation on average: $r \approx \log_2(N_{MMT})$. Let $\delta > 0$ be a real parameter and define $\Delta = 2^\delta$ which is exponential in n. We propose to add an additional constraint on the column sums for the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ in order to reduce their size. At each run, we pick one $\mathbf{t}' \in \mathbb{F}_2^\delta$ and require that

$$\begin{aligned} (Q\mathbf{e}_1)_{[r+\delta]} &= \begin{bmatrix} \mathbf{t} \\ \mathbf{t}' \end{bmatrix} \text{ or} \\ (Q\mathbf{e}_2)_{[r+\delta]} &= \begin{bmatrix} \mathbf{t} \\ \mathbf{t}' \end{bmatrix} + \mathbf{s}_{[r+\delta]} \ . \end{aligned}$$

The probability that the corresponding column-sum to a representation matches the target from $\mathbb{F}_2^{r+\delta}$ is now $1/2^{r+\delta}$. Under the heuristic assumption that the number of elements that satisfy the conditions for all possible $\mathbf{t}'$, we can expect that the lists are smaller by the factor Δ: $L' = L/\Delta$. We expect to find $C' = C/\Delta$ collisions between $\mathcal{L}_1$ and $\mathcal{L}_2$. We repeat by choosing the next target $\mathbf{t}'$ until all Δ possibilities are tried. Altogether we produce the same number of collisions, C, as before but in blocks of heuristic size C'. The price we have to pay is that we need to create Δ times the lists $\mathcal{L}_1, \mathcal{L}_2$ which costs $\mathcal{T}_1 = \Delta \cdot \max(B, C'_B, L')$ where $C'_B = C_B/\Delta \approx L/\Delta$. The overall cost are $\mathcal{T} = \Delta \cdot \max(B, C'_B, L', C') = \max(\Delta \cdot B, C_B, L, C)$.
As we wish to keep the overall time at its asymptotic minimum while minimizing L', we choose Δ such that $\Delta \cdot B$ equals the maximum of the other terms in $\mathcal{T}$ which is given by L.

We hence choose $\Delta = L/B$ which means that we reduce the lists $\mathcal{L}_1, \mathcal{L}_2$ by exactly the factor that makes them larger than the bottom lists. The minimal memory requirement is given by B and with the above technique all the lists are now of same size. We need 2 lists per level in an implementation such that the memory is reduced from

$$2B + 2L \quad \text{to} \quad 4B \ .$$

We reevaluate the worst-case running time for half-distance decoding. With the above method, all lists are of length $2^{0.0139\,n}$. We need to repeat the creation of the lists $\Delta \approx 2^{0.0075\,n}$ times. Heuristically, the asymptotic running time is unchanged at $2^{0.05363\,n}$. Analogously, for full-distance decoding all lists are of size $2^{0.0376\,n}$ and $\Delta \approx 2^{0.0162\,n}$.

10.2. Extended representation technique in the search phase

We wish to solve the collision problem (10.1):

$$\sum_{i \in I_1} \mathbf{q}_i = \mathbf{s} + \sum_{i \in I_2} \mathbf{q}_i \ .$$

In Stern's algorithm both index sets I_1, I_2 are chosen in a disjoint fashion. Thus every solution I only has a unique representation as the union of I_1 and I_2. MMT choose the sets I_1, I_2 within the complete interval $[1, k+\ell]$ but without intersections. Each column of the solution set I is thus either in I_1 or in I_2. The number of ways to decompose I is given by the possibilities to pick $p/2$ columns out of p:

$$N_{MMT} = \binom{p}{p/2} \ .$$

We propose to choose $|I_1| = |I_2| = \frac{p}{2} + \varepsilon$ for some $\varepsilon > 0$ such that $|I_1 \cap I_2| = \varepsilon$. So we allow for ε columns $\mathbf{q}_i$ that appear on both sides of the above equation. Thus every solution I is written as the symmetric difference $I = I_1 \Delta I_2 := I_1 \cup I_2 \setminus (I_1 \cap I_2)$ where we cancel out all ε elements in the intersection of I_1 and I_2 as illustrated in figure 10.2.

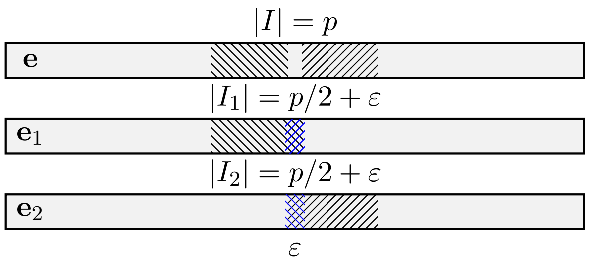


Figure 10.2.: *Decomposition of an index set I into two overlapping index sets.*

If we allow ε columns to be in I_1 and I_2, the number of representations (I_1, I_2) increases by the possibilities to choose ε additional columns. It is equal to

$$N = \binom{p}{p/2}\binom{k+\ell-p}{\varepsilon} \ . \tag{10.8}$$

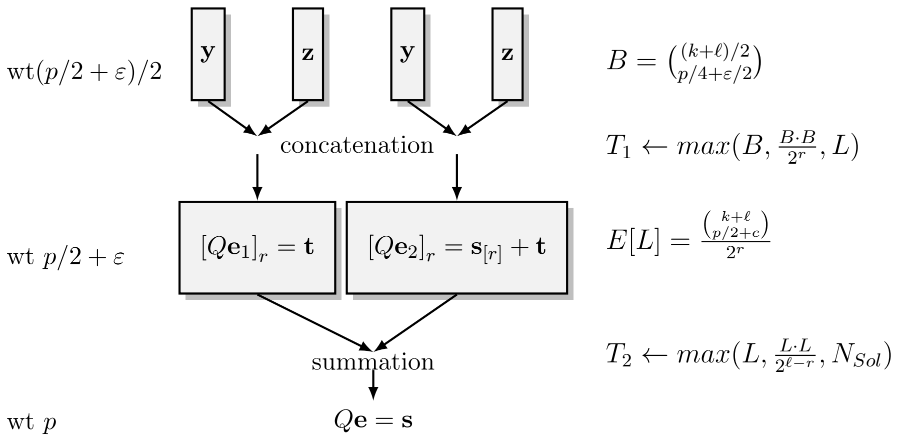
For random code instances the error vector $\mathbf{e}$ is sparse and p is small in comparison to $k+\ell$ which means that N increases quickly for small choices of ε due to the second factor in (10.8). An increased number of representations allows us to impose stronger constraints on the colliding elements realized by a larger r for the r-bit target vector $\mathbf{t}$. This in return means we will need to create less elements resulting in smaller lists and a lower running time. The next section describes a first algorithm that is suboptimal and shows that we need two levels of representations to obtain a lower running time.

Comment aside. Consider the binary solution vector $\mathbf{e}$ that we represent as a sum of binary vectors. This means that each one-entry is split into either 1+0 or 0+1 in the MMT-algorithm presented in the previous section. Now, we also allow to split each zero-entry of $\mathbf{e}$ into either $0+0$ or $1+1$. Hence our benefit comes from using the equation $1+1=0$ in $\mathbb{F}_2$ which gave raise to the subtitle of our paper [BJMM12]: "Decoding Random Binary Linear Codes in $2^{n/20}$: How $1+1=0$ Improves Information Set Decoding".

10.2.1. A first attempt - How to choose the number of levels

Suppose that the solution of weight p disperses its ones uniformly such that it has exactly $p/2$ ones in the first and second half of $[1, k+\ell]$. Let ε be a parameter and split Q into two sets of disjoint column sums: $Q = [Q_1|Q_2]$ where $Q_i \in \mathbb{F}_2^{\ell\times(k+\ell)/2}$.

We develop the same algorithm as done for the simple representation technique with the difference that the weight of the intermediate vectors is $p/2+\varepsilon$ and $p/4+\varepsilon/e$ for the first and second level. Figure 10.3 illustrates the algorithm.



$\mathrm{wt}(p/2+\varepsilon)/2$
$\mathbf{y}$
$\mathbf{z}$
$\mathbf{y}$
$\mathbf{z}$
$B = \binom{(k+\ell)/2}{p/4+\varepsilon/2}$
concatenation
$T_1 \leftarrow max(B, \frac{B\cdot B}{2^r}, L)$
wt $p/2+\varepsilon$
$[Q\mathbf{e}_1]_r = \mathbf{t}$
$[Q\mathbf{e}_2]_r = \mathbf{s}_{[r]} + \mathbf{t}$
$E[L] = \frac{\binom{k+\ell}{p/2+c}}{2^r}$
summation
$T_2 \leftarrow max(L, \frac{L\cdot L}{2^{\ell-r}}, N_{Sol})$
wt p
$Q\mathbf{e} = \mathbf{s}$

Figure 10.3.: *One level - suboptimal.*

We start by creating two base lists of vectors of length $k+\ell$ and weight $p/4+\varepsilon/2$ and their corresponding column sums. The lists are

$$\mathcal{B}_1 := \{(\mathbf{z}, Q_1\mathbf{y}) \mid \mathbf{y} \in \mathbb{F}_2^{(k+\ell)/2}, \operatorname{wt}(\mathbf{y}) = \frac{p}{4} + \frac{\varepsilon}{2}\}$$

and

$$\mathcal{B}_2 := \{(\mathbf{z}, Q_2\mathbf{z}) \mid \mathbf{z} \in \mathbb{F}_2^{(k+\ell)/2}, \operatorname{wt}(\mathbf{z}) = \frac{p}{4} + \frac{\varepsilon}{2}\}$$

which are of exact size $B = \binom{(k+\ell)/2}{p/4+\varepsilon/2}$. A call to a merge-join routine creates elements $[\mathbf{y}|\mathbf{z}]$ of exact weight $p/2+\varepsilon$ for which the column sum $Q([\mathbf{y}|\mathbf{z}])$ matches a target vector $\mathbf{t}$ or $\mathbf{t}+\mathbf{s}_{[r]}$ for some randomly chosen target vector $\mathbf{t} \in \mathbb{F}_2^r$. The elements are stored in lists $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. So far this is a standard approach that concatenates vectors. The probability for such a collision is 2^{-r} for a uniform matrix Q. We expect that about $C_B = B^2/2^r$ elements match each of the targets. The two lists we create, one for each target, are of expected size $L = \binom{k+\ell}{p+\varepsilon}2^{-r}$. We choose r such that we can expect to find one representation in the lists: $r \approx \log_2(N)$. The running time of the collision search is $\mathcal{T}_1 = \max(B, C_B, L)$. Note that for well chosen $\varepsilon > 0$, we can hope to reduce $\mathcal{T}_1$.

A second collisions search then searches the solution as a vector that matches the target $\mathbf{s}$ on all ℓ coordinates. The probability is $2^{r-\ell}$ such that we expect to find $C = L^2/2^{\ell-r}$ colliding elements. We reject duplicates and all elements that have not weight p. The cost in time are $\mathcal{T}_2 = \max(L, C, N_{Sol})$ for the second merge. As N_{Sol} is necessarily smaller than C, we can neglect it. The overall running time in the asymptotic case is hence $\mathcal{T} = \max(\mathcal{T}_1, \mathcal{T}_2) = \max(B, C_B, L, C)$.

Suppose that $k/n = 0.7557$ and $\omega = 0.04$. Suppose that Q is uniform. We try to find optimal parameters p, ℓ, ε for a one-level extended representation technique algorithm as presented in the above paragraph that minimizes the running time. For optimal parameters, we obtain the individual cost

$$B \approx 2^{0.0180\,n} \text{ and } C_B = L = C \approx 2^{0.0281\,n}\ .$$

The number of repetitions is about $2^{0.0536\,n}$ which leads to a running time $\mathcal{T} \approx 2^{0.0817\,n}$. We see that the cost to create the lists $\mathcal{L}_1$ and $\mathcal{L}_2$, $\mathcal{T}_1$, is one dominating factor in the overall running time. We can reduce the cost for creation of $\mathcal{L}_1, \mathcal{L}_2$ by building the elements in the same way than $\mathbf{e}$ as a sum of vectors. The following section describes our new algorithm that achieves a lower asymptotic running time than previous approaches. It has a second level in which it applies the representation technique

10.2.2. Algorithm by extended representations technique

Our algorithm can be described as a computation tree of depth three, see figure 10.4 for an illustration. We enumerate the levels from bottom to top where the third level is the initial computation of disjoint base lists $\mathcal{B}_1$ and $\mathcal{B}_2$ and the zero level identifies the final output list $\mathcal{L}$ that might contain the solution.

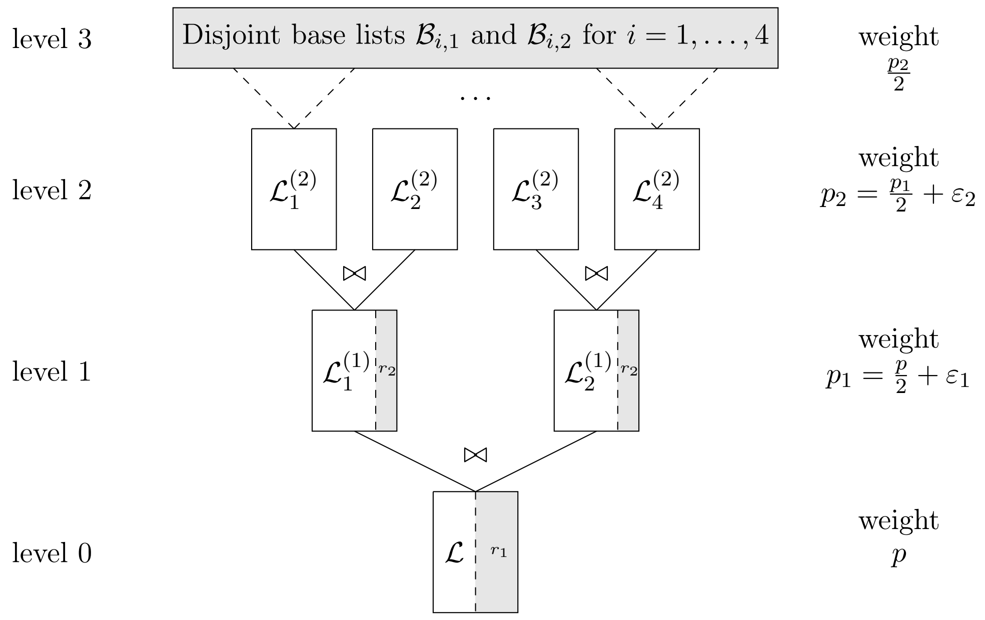

level 3
Disjoint base lists $\mathcal{B}_{i,1}$ and $\mathcal{B}_{i,2}$ for $i = 1, \dots, 4$
weight $\frac{p_2}{2}$
level 2
$\mathcal{L}_1^{(2)}$
$\mathcal{L}_2^{(2)}$
$\mathcal{L}_3^{(2)}$
$\mathcal{L}_4^{(2)}$
weight $p_2 = \frac{p_1}{2} + \varepsilon_2$
$\bowtie$
level 1
$\mathcal{L}_1^{(1)}$ r_2
$\mathcal{L}_2^{(1)}$ r_2
weight $p_1 = \frac{p}{2} + \varepsilon_1$
level 0
$\mathcal{L}$ r_1
weight p

Figure 10.4.: *Illustration of the* ColumnMatch *algorithm using the extend. representation technique at level one and two while the third layer is a disjoint composition of the elements.*

We introduce parameters ε_1 and ε_2 representing the number of additional 1's we allow on the first and second level, respectively. At each representation level, we impose a bit constraint of r_i bits, $i = 1, 2$, where $0 \leq r_2 \leq r_1 \leq \ell$. In the following description, we equip every object with an upper index that indicates its computation level, e.g., a list $\mathcal{L}_j^{(2)}$ is contained in the second level.

On the first level, we search for index sets $I_1^{(1)}$ and $I_2^{(1)}$ in $[k + \ell]$ of size $p_1 := \frac{p}{2} + \varepsilon_1$ which intersect in exactly ε_1 coordinates such that $I = I_1^{(1)} \Delta I_2^{(1)}$. In other words, we create lists of binary vectors $\mathbf{e}_1^{(1)}$ and $\mathbf{e}_2^{(1)}$ of weight p_1 and search for tuples $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)})$ such that $\mathrm{wt}(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}) = p$ and $Q(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}) = \mathbf{s}$. Note that the number of tuples $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)})$ that represent a single solution vector $\mathbf{e}$ is

$$R_1(p, \ell; \varepsilon_1) := \binom{p}{\frac{p}{2}} \binom{k + l - p}{\varepsilon_1} \tag{10.9}$$

as we derived in (10.8). To optimize the running time, we impose a constraint on $r_1 \approx \log_2 R_1$ coordinates of the corresponding vectors $Q\mathbf{e}_i^{(1)}$ such that we can still expect to find one representation of the desired solution $\mathbf{e}$. This is analogous to what we explained in section 10.1 for the simple representation technique. The bit-constraint on the column sums allows us to reduce the size of the lists $\mathcal{L}_j^{(1))}$ by restricting the vectors $\mathbf{e}_i^{(1)}$ to those where $Q\mathbf{e}_i^{(1)}$ equals some target vector. More precisely, the algorithm proceeds as follows. We first fix a random vector $\mathbf{t}_1^{(1)} \in_R \mathbb{F}_2^{r_1}$, set $\mathbf{t}_2^{(1)} := \mathbf{s}_{[r_1]} + \mathbf{t}_2^{(1)}$ and compute two lists

$$\mathcal{L}_i^{(1)} = \{e_i{}^{(1)} \in \mathbb{F}_2^{k+\ell} \mid \mathrm{wt}(e_i) = p_1 \text{ and } (Q\mathbf{e}_i^{(1)})_{[r_1]} = \mathbf{t}_i^{(1)}\} \text{ for } i = 1, 2.$$

Observe that any two elements $\mathbf{e}_i^{(1)} \in \mathcal{L}_i^{(1)}$, $i = 1, 2$, already fulfill by construction the equation $(Q(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}))_{[r_1]} = \mathbf{s}_{[r_1]}$, i.e. they already match the syndrome $\mathbf{s}$ on r_1 coordinates. In order to solve the SMP, we are interested in a solution $\mathbf{e} = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ that matches the syndrome $\mathbf{s}$ on *all* ℓ positions and has weight *exactly* p. Once $\mathcal{L}_1^{(1)}$ and $\mathcal{L}_2^{(1)}$ have been created, this can be accomplished by calling the Merge-Join-Decode algorithm from section 1.2 on input $\mathcal{L}_1^{(1)}, \mathcal{L}_2^{(1)}$ with target $\mathbf{s}$, weight p and parameter ℓ.

It remains to show how to construct $\mathcal{L}_1^{(1)}$ and $\mathcal{L}_2^{(1)}$ from the lists at the second level.

We represent $\mathbf{e}_i^{(1)}$ as a sum of two overlapping vectors $\mathbf{e}_{2i-1}^{(2)}, \mathbf{e}_{2i}^{(2)}$ both of weight $p_2 := \frac{p_1}{2} + \varepsilon_2$, i.e. we require the two vectors to intersect in exactly ε_2 coordinates. Altogether, the solution $\mathbf{e}$ is now decomposed as

$$\mathbf{e} = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)} = \mathbf{e}_1^{(2)} + \mathbf{e}_2^{(2)} + \mathbf{e}_3^{(2)} + \mathbf{e}_4^{(2)} \ .$$

Clearly, there are

$$R_2(p, \ell; \varepsilon_1, \varepsilon_2) = \binom{p_1}{p_1/2} \cdot \binom{k + \ell - p_1}{\varepsilon_2}$$

many representations for $\mathbf{e}_j^{(1)}$ where $p_1 = \frac{p}{2} + \varepsilon_1$. Similarly to the first level, this allows us to fix $r_2 \approx \log R_2$ coordinates of the partial sums $Q\mathbf{e}_i^{(2)}$ to some target values $\mathbf{t}_i^{(2)}$. More precisely, we draw two target vectors $\mathbf{t}_1^{(2)}, \mathbf{t}_3^{(2)} \in \mathbb{F}_2^{r_2}$, set $\mathbf{t}_{2j}^{(2)} = (\mathbf{t}_j^{(1)})_{[r_2]} + \mathbf{t}_{2j-1}^{(2)}$ for $j = 1, 2$, and compute four lists

$$\mathcal{L}_i^{(2)} = \{\mathbf{e}_i^{(2)} \in \mathbb{F}_2^{k+l} \ | \ \mathrm{wt}(\mathbf{e}_i^{(2)}) = p_2 \text{ and } (Q\mathbf{e}_i^{(2)})_{[r_2]} = \mathbf{t}_i^{(2)}\} \text{ for } i = 1, \ldots, 4.$$

Notice that by construction all combinations of two elements from either $\mathcal{L}_1^{(2)}, \mathcal{L}_2^{(2)}$ or $\mathcal{L}_3^{(2)}, \mathcal{L}_4^{(2)}$ match their respective target vector $\mathbf{t}_j^{(1)}$ on r_2 coordinates.

How to create the lists $\mathcal{L}_1^{(2)}, \ldots, \mathcal{L}_4^{(2)}$ at the second level.

We exemplary explain how to create $\mathcal{L}_1^{(2)}$. The remaining lists can be constructed analogously. We apply a classical meet-in-the-middle collision search. We decompose $\mathbf{e}_1^{(2)}$ as $\mathbf{e}_1^{(2)} = \mathbf{y} + \mathbf{z}$ by two non-overlapping vectors $\mathbf{y}$ and $\mathbf{z}$ of length $k + \ell$. To be more precise, we first choose a random partition of $[k + \ell]$ into two equal sized sets P_1 and P_2, i.e. $[k + \ell] = P_1 \cup P_2$ with $|P_1| = |P_2| = \frac{k+\ell}{2}$, and force $\mathbf{y}$ to have its $\frac{p_2}{2}$ 1-entries in P_1 and $\mathbf{z}$ to have its $\frac{p_2}{2}$ 1-entries in P_2. That is we construct two base lists

$$\mathcal{B}_1 := \{y \in \mathbb{F}_2^{k+\ell} \mid \mathrm{wt}(y) = \frac{p_2}{2} \text{ and } y_i = 0 \, \forall i \in P_2\}$$

and

$$\mathcal{B}_2 := \{\mathbf{z} \in \mathbb{F}_2^{k+\ell} \mid \mathrm{wt}(\mathbf{z}) = \frac{p_2}{2} \text{ and } z_i = 0 \, \forall i \in P_1\}.$$

We invoke Merge-Join-Decode to compute

$$\mathcal{L}_1^{(2)} = \text{Merge-Join-Decode } (\mathcal{B}_1, \mathcal{B}_2, r_2, p_2, \mathbf{t}_1^{(2)}) \ .$$

Let $S_3 = |\mathcal{B}_1| = |\mathcal{B}_2|$ denote the size of the base lists and let C_3 be the total number of matched vectors that occur in Merge-Join-Decode (since the splitting is disjoint, neither duplicates nor inconsistencies can arise). Then Merge-Join-Decode needs time

$$T_3(p, \ell; \varepsilon_1, \varepsilon_2) = \tilde{\mathcal{O}}\left(\max\{S_3, C_3\}\right).$$

Clearly, we have

$$S_3 := S_3(p, \ell; \varepsilon_1, \varepsilon_2) = \binom{(k+\ell)/2}{p_2/2}.$$

Assuming uniformly distributed partial sums we obtain

$$\mathbb{E}\left[C_3\right] = \frac{S_3^2}{2^{r_2}} \quad .$$

For a unlucky pick of the partition P_1, P_2, the solution will be missed as only a representation can be found that spreads its ones exactly equally over the two sets. Decomposing $\mathbf{e}_1^{(2)}$ into $\mathbf{x}$ and $\mathbf{y}$ from disjoint sets P_1 and P_2 hence introduces a probability of loosing the vector $\mathbf{e}_1^{(2)}$ and hence the solution $\mathbf{e} = \mathbf{e}_1^{(2)} + \mathbf{e}_2^{(2)} + \mathbf{e}_3^{(2)} + \mathbf{e}_4^{(2)}$. For a randomly chosen partition P_1, P_2, the probability that one representation $(\mathbf{e}_1^{(2)}, \mathbf{e}_2^{(2)})$ equally distributes its 1-entries over P_1 and P_2 is given by (10.7):

$$\mathcal{P}_{\mathcal{B}} = \frac{\binom{(k+\ell)/2}{p_2/2}^2}{\binom{k+\ell}{p_2}}$$

which is asymptotically inverse-polynomial in n

$$\mathcal{P}_{\mathcal{B}} = \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$$

(see section 10.1.1).

Choosing independent partitions[1] $P_{i,1}, P_{i,2}$ and creating corresponding base lists $\mathcal{B}_{i,1}, \mathcal{B}_{i,2}$ for all four lists $\mathcal{L}_i^{(2)}$, we can guarantee *independent* splitting conditions for all the $\mathbf{e}_i^{(2)}$. This yields a total splitting probability of $\mathcal{P}_{\text{Split}} = (\mathcal{P}_{\mathcal{B}})^4$ which is still inverse-polynomial in n. Repeating polynomial many times the choice of the partitions, the probability of success goes exponentially to one. The asymptotic running time which is exponential is not increased.

After having created the lists $\mathcal{L}_i^{(2)}$, $i = 1, \ldots, 4$ on the second level, two more applications of the MergeJoin algorithm suffice to compute the lists $\mathcal{L}_j^{(1)}$ on the first level. Eventually, a last application of MergeJoin yields $\mathcal{L}$, whose entries are solutions to the SMP. See algorithm 10.1 for a complete pseudocode description.

[1]The choice of independent partitions at the bottom level was suggested by Daniel Bernstein.

Algorithm 10.1: ColumnMatch – Submatrix matching algorithm

```
Input:      $Q \in \mathbb{F}_2^{\ell \times k+\ell}$, $\mathbf{s} \in \mathbb{F}_2^{\ell}$, $p \leq k+\ell$
Output:     List $\mathcal{L}$ of vectors in $\mathbf{e} \in \mathbb{F}_2^{k+\ell}$ with $\mathrm{wt}(\mathbf{e}) = p$ and $Q\mathbf{e} = \mathbf{s}$
Parameters: Choose optimal $\varepsilon_1, \varepsilon_2$ and set $p_1 = p/2 + \varepsilon_1$ and $p_2 = p_1/2 + \varepsilon_2$.

Choose random partitions $P_{i,1}, P_{i,2}$ of $[k+\ell]$.
Create the base lists $\mathcal{B}_{i,1}$ and $\mathcal{B}_{i,2}$.
Repeat
    Choose a target $\mathbf{t}_1^{(1)} \in_R \mathbb{F}_2^{r_1}$ and set $\mathbf{t}_2^{(1)} = \mathbf{s}_{[r_1]} + \mathbf{t}_1^{(1)}$.
    Choose $\mathbf{t}_1^{(2)}, \mathbf{t}_3^{(2)} \in_R \mathbb{F}_2^{r_2}$.
    Set $\mathbf{t}_2^{(2)} = (\mathbf{t}_1^{(1)})_{[r_2]} + \mathbf{t}_1^{(2)}$ and $\mathbf{t}_4^{(2)} = (\mathbf{t}_2^{(1)})_{[r_2]} + \mathbf{t}_3^{(2)}$.

    List $\mathcal{L}_i^{(2)} \leftarrow$ Merge-Join-Decode $(\mathcal{B}_{i,1}, \mathcal{B}_{i,2}, r_2, p_2, \mathbf{t}_i^{(2)})$ for $i = 1, \ldots, 4$.
    List $\mathcal{L}_i^{(1)} \leftarrow$ Merge-Join-Decode $(\mathcal{L}_{2i-1}^{(2)}, \mathcal{L}_{2i}^{(2)}, r_1, p_1, \mathbf{t}_i^{(1)})$ for $i = 1, 2$.
    List $\mathcal{L} \leftarrow$ Merge-Join-Decode $(\mathcal{L}_1^{(1)}, \mathcal{L}_2^{(1)}, \ell, p, \mathbf{s})$.
    Output $\mathcal{L}$.
```

10.2.3. Complexity analysis

We estimate the complexity of ColumnMatch as a function of the parameters $(p, \ell; \varepsilon_1, \varepsilon_2)$, where $(\varepsilon_1, \varepsilon_2)$ are optimization parameters. Notice that the values r_i and p_i are fully determined by $(p, \ell; \varepsilon_1, \varepsilon_2)$. The base lists $\mathcal{B}_1$ and $\mathcal{B}_2$ are of size $S_3(p, \ell; \varepsilon_1, \varepsilon_2)$ as defined above.

The three consecutive calls to the merge-join routine create lists $\mathcal{L}_j^{(2)}$ of size $S_2(p, \ell; \varepsilon_1, \varepsilon_2)$, the lists $\mathcal{L}_j^{(1)}$ of size $S_1(p, \ell; \varepsilon_1, \varepsilon_2)$ and the final list $\mathcal{L}$ (which has not to be stored). More precisely, we obtain

$$S_i(p, \ell; \varepsilon_1, \varepsilon_2) = \mathbb{E}\left[|\mathcal{L}_j^{(i)}|\right] = \binom{k+\ell}{p_i} \cdot 2^{-r_i} \text{ for } i = 1, 2.$$

Here we assume uniformly distributed partial sums $Q\mathbf{e}_i^{(j)}$. This permits us to assume that the bit-constraints are satisfied with probability 2^{-r_i}.

Let C_i for $i = 1, 2, 3$ denote the number of all matching vectors (including possible inconsistencies or duplicates) that occur in the three Merge-Join steps. If we set $r_3 = 0$ and $r_0 = \ell$, then

$$\mathbb{E}\left[C_i\right] = S_i^2 \cdot 2^{r_i - r_{i-1}}.$$

Following the analysis of Merge-Join in section 1.2, the time $\mathcal{T}_i$ of the three Merge-Join steps is given by

$$\mathcal{T}_i(p, \ell; \varepsilon_1, \varepsilon_2) = \max\left\{S_i, C_i\right\}.$$

The overall time and space complexity is thus given by

$$\mathcal{T}(p,\ell;\varepsilon_1,\varepsilon_2) = \max\{\mathcal{T}_3,\mathcal{T}_2,\mathcal{T}_1\} \tag{10.10}$$

and

$$S(p,\ell;\varepsilon_1,\varepsilon_2) = \max\{S_3,S_2,S_1\} \quad .$$

We remark that the above estimates work well in practice. Heuristically, we can assume that the C_i achieve their expected values up to a constant factor and that the number of remaining elements in the lists is close to the expected value (compare to experimental results in section 10.5).

Since our heuristic analysis also relies on the fact that projected partial sums of the form $(Q\mathbf{e})_{[r]}$ yield uniformly distributed vectors in $\mathbb{F}_2^r$, a more precise theoretical analysis needs to take care of a certain class of malformed input parity check matrices H. We show how to obtain a provable variant of our algorithm that works for all but a negligible amount of input matrices H in section 10.7. The provable variant aborts the computation if the observed C_i or S_i differ too much from their expectation which happens for an exponentially small fraction of input matrices.

Worst-case complexity

As we wish to minimize the running time, we need to minimize $\mathcal{T}(p,\ell;\varepsilon_1,\varepsilon_2)$ by choosing optimal parameters p,ℓ,ε_1 and ε_2 for a given decoding problem in an $[n,k,d]$-code.

For a fixed code rate $R := k/n$, we need to optimize the parameters[2] such that the expression

$$\mathcal{T}(p,\ell;\varepsilon_1,\varepsilon_2)\cdot\mathcal{P}(p,\ell)^{-1} \tag{10.11}$$

is minimized under the natural constraints

$$\begin{aligned}
&0 < \ell < \min\{n-k, n-k-\omega-p\}\\
&0 < p < \min\{\omega, k+\ell\}\\
&0 < \varepsilon_1 < k+\ell-p\\
&0 < \varepsilon_2 < k+\ell-p_1\\
&0 < R_2(p,\ell;\varepsilon_1,\varepsilon_2) < R_1(p,\ell;\varepsilon_1,\varepsilon_2) < \ell \quad .
\end{aligned}$$

The time per iteration $\mathcal{T}$ is given by (10.10) and the number of iterations $\mathcal{P}^{-1}$ equals $\left(\binom{k+\ell}{p}\binom{n-k-\ell}{\omega-p}/\binom{n}{\omega}\right)^{-1}$ as given in (9.3). To be more precise, we need to set

$$\mathcal{P} = \frac{\binom{(k+\ell)/2}{p/2}^2\binom{n-k-\ell}{\omega-p}}{\binom{n}{\omega}}$$

due to the disjoint split at the bottom level. For large n, we can assume that $\binom{(k+\ell)/2}{p/2}^2 \approx \binom{k+\ell}{p}$ and use (9.3).

[2]e.g., Octave or Mathematica

Using the approximation $\binom{\alpha n}{\beta n} = 2^{\alpha h(\beta/\alpha)n+o(n)}$ (section A.2 in the appendix) we can express $\mathcal{T}$ in terms of n, k/n, d/n, p/n, ℓ/n, ε_1/n and ε_2/n. For large n and random linear codes, we can even relate $R := k/n$ and $D := d/n$ by the Gilbert-Varshamov bound (section 8.1). Thus asymptotically we obtain $D = h^{-1}(1-R) + o(1)$ for a given R where h is the binary entropy function. In this way, we can express all cost in terms of n and k/n and obtain $\mathcal{T}(p, \ell; \varepsilon_1, \varepsilon_2) \approx 2^{F(R)\,n}$ as explained in section 9.3 where $F(R)$ is determined by the used algorithm. We obtain the following numerical results by use of the octave code in appendix B.

Worst-case complexity. For *bounded-distance decoding*, we set $W := \omega/n = D/2$. We numerically determine the best parameters for equidistant information rates and obtain the worst asymptotic time complexity for $R = 0.4575$ where $W = 0.062311$. Remark that the same running time is obtained for information rates nearby $R \in \{0.4574, 0.4575, 0.45795, 0.45805\}$. We picked the value that results in the lowest memory requirement. For parameters

$$p \approx 0.0141n, \quad \ell \approx 0.0737n, \quad \varepsilon_1 \approx 0.0033n \text{ and } \varepsilon_2 \approx 0.0002n \ ,$$

we obtain the individual cost

$$S_3 \approx 2^{0.02162\,n}, \quad S_2 \approx 2^{0.0307\,n}, \quad S_1 \approx 2^{0.0306\,n} \ ,$$

$$C_3 \approx C_2 \approx C_1 \approx 2^{0.0307\,n} \ .$$

Due to about $2^{0.0186\,n}$ repetitions, the overall running time becomes $2^{0.04933\,n}$ using $2^{0.0307\,n}$ space. The exactness of the result depends clearly on the numerical method and the precision of all parameters, especially the one when calculating D. We used the nelder-mead function of octave and the code B.3 as presented in the appendix. The worst complexity for full-distance decoding occurs for $R = 0.44100$ and $W = 0.13057$. For parameters

$$p \approx 0.0546, \quad \ell \approx 0.2075n, \quad \varepsilon_1 \approx 0.0106n \text{ and } \varepsilon_2 \approx 0.0015n \ ,$$

we obtain the individual cost

$$S_3 \approx 2^{0.0656\,n}, \quad S_2 \approx 2^{0.0778\,n}, \quad S_1 \approx 2^{0.0771\,n} \ ,$$

$$C_3 \approx C_2 \approx C_1 \approx 2^{0.0778\,n} \ .$$

Due to about $2^{0.024\,n}$ repetitions, the overall running time is $2^{0.1018\,n}$ using $2^{0.0778\,n}$ space.

10.2.4. Reducing the memory requirement under heuristic assumptions

Similar to the improvement in section 10.1.2, we can allow larger restrictions and trade time for memory. If we choose the size of the restriction with care, we can obtain an algorithm of same asymptotic running time and reduced memory requirement.

We start with the lists at the first level and increase the constraints per run to: $r_1 + \delta_1$ for real parameter $\delta_1 \geq 0$. The parameter is not completely free as we require a same overall running time. Remember that we have a fixed target on r_1 bits and that we iterate over all possible δ_1-bit vectors $\mathbf{t}'$. We assume that the additional constraint is satisfied for about the same number of vectors for varying $\mathbf{t}'$. Let $\Delta_1 = 2^{\delta_1}$. Per iteration, we search the solution within the collisions between two lists of expected size $S_1' \approx \frac{S_1}{\Delta_1}$. This reduces the expected number of collisions by a factor Δ_1. We repeat Δ_1 times with changed target to produce all C_1 collisions than before. Let $\mathcal{T}_i$ denote the time to create the lists of level i. The expected time to find the solution can then be computed as

$$\mathcal{T}_0 = \Delta_1 \cdot max(\frac{C_1}{\Delta_1}, \mathcal{T}_1)$$

where

$$\Delta_1 \cdot \mathcal{T}_1 = \Delta_1 \cdot max(\frac{S_1}{\Delta_1}, \frac{C_2}{\Delta_1}, \mathcal{T}_2) = \max(S_1, C_2, \Delta_1 \cdot \mathcal{T}_2) \ .$$

The memory in the first level is now reduced but the overall memory requirement is still determined by the lists at the second level, we apply the same idea again. Increasing the constraints to $r_2 + \delta_2$ for a real factor $\delta_2 \geq 0$, reduces the expected size of the lists to $S_2' = S_2/\Delta_2$ where $\Delta_2 = 2^{\delta_2}$. Now, we can expect to find only C_2/Δ_2 collisions for the lists in the level above. However, if we repeatedly create lists of size S_2' with changed target on δ_2 bits, we can expect to create S_1 elements in the upper level after Δ_2 iterations. The time to create all lists of level two of size S_2 is given by the time to construct Δ_2 times lists of size S_2':

$$\mathcal{T}_2 = \Delta_2 \cdot max(\frac{S_2}{\Delta_2}, \frac{C_3}{\Delta_2}, S_3) = max(S_2, C_3, \Delta_2 \cdot S_3) \ .$$

Heuristically, the memory requirement is

$$\max(S_3, \frac{S_2}{\Delta_2}, \frac{S_1}{\Delta_1}) \ .$$

We now determine the optimal size of δ_i. The running time is unchanged if $\mathcal{T}_0 = \mathcal{T}$ from (10.10 on page 128), that is, if

$$\mathcal{T}_0 = \max(C_1, S_1, C_2, \Delta_1 \cdot S_2, \Delta_1 \cdot C_3, \Delta_1 \cdot \Delta_2 \cdot S_3)$$

equals

$$\mathcal{T} \approx S_2 \approx C_2 \approx C_1 \ .$$

As S_2 is already dominating the time, we can only choose $\delta_1 = 0$ and thus not reduce the lists at the first level. We can however reduce the lists at the second level by a maximal factor $\Delta_2 = S_3/R_2$ such that the lists at the bottom and middle level are of same length.

Using two lists per level, we drop the number of stored elements from

$$2S_1 + 2S_2 + 2S_3 \quad \text{to} \quad 2S_1 + 4S_3$$

where S_2 is larger than S_3 by the factor Δ_2 which is exponential in n.

For half-distance decoding in the worst case, the overall asymptotic memory requirement is almost unchanged at $2^{0.0306\,n} = \tilde{\mathcal{O}}\left(S_1\right)$ which is very close to the previous maximum $S_2 \approx 2^{0.0307\,n}$. The number of inner repetitions is $\Delta_2 \approx 2^{0.0091\,n}$.

In the case of full-distance decoding, the asymptotic memory requirement is about $2^{0.0771\,n} = \tilde{\mathcal{O}}\left(S_1\right)$ in comparison to the previous maximum $S_2 \approx 2^{0.0778\,n}$. The number of inner repetitions is $\Delta_2 \approx 2^{0.0122\,n}$.

10.3. Comparison of asymptotic complexity

We now show that we improve the running time of information-set decoding by an exponential factor in comparison to the latest results presented in 2011 [BLP11, MMT11].

The comparison to other algorithms is based on the complexity coefficient $F(R)$ as explained in section 9.3. We express the running time as

$$\mathcal{T}(n, R) = 2^{nF(R)+o(n)} \leq 2^{n\lceil F(R)\rceil_\rho}$$

for large enough n. The coefficient $F(R)$ is computed in the worst-case complexity, for the information rate R that maximizes $F(R)$. For random linear codes, we can relate $R = k/n$ and $D = d/n$ via the Gilbert-Varshamov bound. Thus asymptotically we obtain $D = h^{-1}(1-R)+o(1)$ for a given R where h is the binary entropy function. For *bounded-distance decoding*, we set $W := \omega/n = D/2$. We numerically determined the optimal parameters for several equidistant rates R and interpolated $F(R)$. The results are shown in figure 10.5.

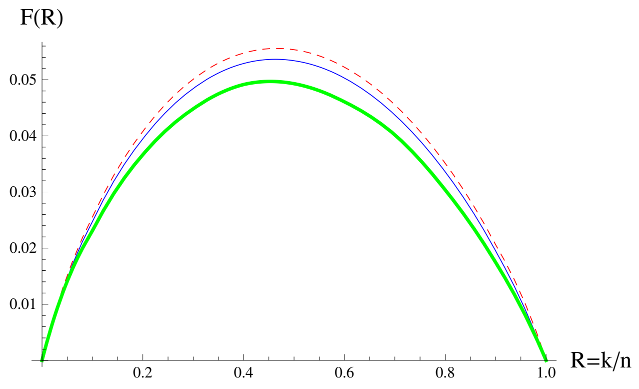


Figure 10.5.: *Comparison of $F(R)$ for code rates $0 < R < 1$ for bounded-distance decoding. Our algorithm is represented by the thick curve, MMT is the thin curve and Ball-collision is the dashed curve.*

For *full-distance decoding*, we need to find a closest codeword at maximal distance $W = D = h^{-1}(1-R)$. Doing the same numerical optimization as described before, we obtain the curve for $F(R)$ as shown in figure 10.6. We see that the worst case appears for almost same information rates in all three algorithms (around 0.46 for half-distance decoding and around 0.42 for full-distance decoding).

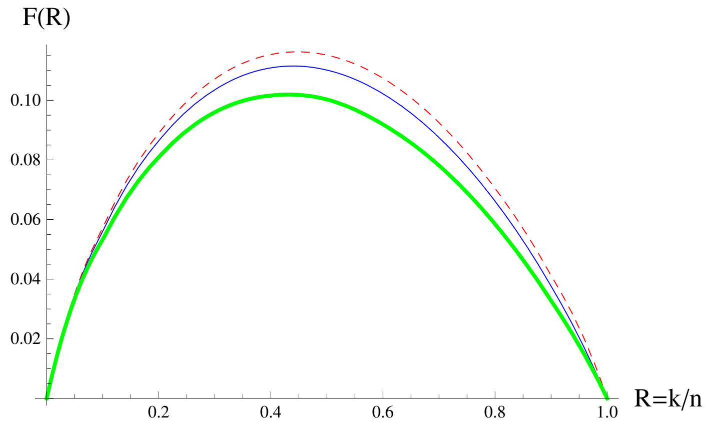


Figure 10.6.: *Complexity coefficient $F(R)$ for full-distance decoding. Our algorithm is represented by the thick curve, MMT is the thin curve and Ball-collision is the dashed curve.*

We take a closer look at the *worst-case* complexities of decoding algorithms for random linear codes. Table 10.1 compares the classical techniques used for information-set decoding with the algorithms based on the representation technique for a worst-case information rate. We already saw the complexity for the algorithms by Lee-Brickel, Stern, Ball-collision and MMT in the introduction to ISD (section 9.3, table 9.1). Comparing the time coefficient in the exponential, we see that our ISD algorithm from section 10.2.2 that uses the extended representation technique has a significantly lower running time (1st and 4th column). The price we need to pay is an augmented memory requirement (3rd and 5th column).

Algorithm	half-dist.		full dec.		
	time	space	time	space	sect.
Lee-Brickell	0.05752	-	0.1208	-	9.1
Stern	0.05563	0.0134	0.1166	0.0318	9.1
Ball-collision	0.05559	0.0148	0.1164	0.0374	9.1
MMT	0.05363	0.0215	0.1115	0.0538	10.1
Heurist. MMT	0.05363	0.0139	0.1115	0.0376	10.1.2
New algorithm	0.04933	0.0307	0.1018	0.0778	10.2
Heurist. new algorithm	0.04933	0.0306	0.1018	0.0771	10.2.4

Table 10.1.: *Comparison of worst-case complexity coefficients, i.e., the time columns represent the maximal complexity coefficient $F(R)$ for $0 < R < 1$.*

Fixing the available memory to at most $2^{0.0317\,n}$ for all algorithms, we can still see an amelioration in the time coefficient: We can easily restrict Ball-collision, MMT and our new algorithm to the space complexity coefficient 0.0317 which is the rounded space requirement of Stern's algorithm ($\lfloor 0.03176 \rfloor$, for $k \approx 0.446784$). In this case, we obtain time complexities $F_{\text{ball}}(R) = 0.1163$, $F_{\text{MMT}}(R) = 0.1129$ and $F_{\text{our}}(R) = 0.1110$, which shows that our improvement is not a pure time memory trade-off.

10.4. Predicted merge-join

The large number of collisions that occur when merging lists in intermediate levels dominates the running time. We require that the colliding elements do intersect at few positions such that the weight of the combined element is smaller than the sum of the weights of the starting elements.

More precisely, we are given two binary vectors $\mathbf{e}_1, \mathbf{e}_2$ of length m and weight $p' = p/2 + \varepsilon$. The probability that they intersect in exactly ε positions such that their sum $\mathbf{e}_1 + \mathbf{e}_2$ has weight p is:

$$\mathcal{P} = \frac{\binom{m-p'}{p'-\varepsilon}\binom{p'}{\varepsilon}}{\binom{m}{p'}} \quad .$$

For practical parameters, the probability is small such that a lot of colliding elements are inconsistent with respect to the weight constraint and will be discarded. In order to avoid these unnecessary cost, we propose to perform a preprocessing of the lists that sorts the elements according to the positions of non-zero elements. The merge-join is then performed only on sets of elements that have a specified number of ones at the same position. It may only happen that they intersect on additional positions. The running time can decrease in this way at the cost of a higher space requirement.

We aim at constructing a list $\mathcal{L}^{(1)}$ of $L^{(1)}$ elements $\mathbf{e}_1 + \mathbf{e}_2 \in \mathbb{F}_2^m$ each of weight p such that $Q(\mathbf{e}_1 + \mathbf{e}_2)_{[\ell]} = \mathbf{t} \in \mathbb{F}_2^\ell$. The elements are obtained by merging two lists $\mathcal{L}_1^{(2)}$ and $\mathcal{L}_2^{(2)}$ that contain vectors $\mathbf{e}_1 \in \mathbb{F}_2^m$ and $\mathbf{e}_2 \in \mathbb{F}_2^m$, respectively, of weight $p' = \frac{p}{2} + \varepsilon$. We also know that $Q(\mathbf{e}_1 + \mathbf{e}_2)_{[r]} = \mathbf{t}$ where $r \leq \ell$.

The presorting can be realized as follows: To every vector $\mathbf{e}_1$ and $\mathbf{e}$ we associate a *set of labels* $L(\mathbf{e}_1)$ and $L(\mathbf{e}_2)$. Every label represents a selection of ε indices corresponding to non-zero coordinates of $\mathbf{e}_1$ or $\mathbf{e}_2$, respectively. For example, let $\varepsilon = 2$ and $\mathbf{e}_1 = (1,1,1,0,0,0)$, then $L(\mathbf{e}_1) = \{(1,1),(1,3),(2,3)\}$. Every vector matches exactly $\binom{p'}{\varepsilon}$ labels. When searching *consistent* collisions for a fixed given vector $\mathbf{e}_1$ with label set $L(\mathbf{e}_1)$, it now suffices to consider only vectors $\mathbf{e}_2$ that have at least one label in common.

For $\mathcal{L}_1^{(2)}$ we create $\binom{m}{\varepsilon}$ different boxes $\mathcal{B}_l^1$, one box for every possible label l. The same is done for $\mathcal{L}_2^{(2)}$ and boxes $\mathcal{B}_l^2$. We insert copies of every vector $\mathbf{e}_i$ into the boxes $\mathcal{B}_l^i$ for all its labels $l \in L(\mathbf{e}_i)$. We denote the resulting larger lists by $\mathcal{L}_i^\star$ and there size by $L_i^\star$. Now, all consistent collisions can be found by simply merging boxes of same label. This process can be seen as predicting the ε positions in which two vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ intersect. The whole procedure can be written in pseudo-code as algorithm 10.2.

Algorithm 10.2: Predicted Merge-Join

```
Input:  $\mathcal{L}_1^{(2)}, \mathcal{L}_2^{(2)}, p, \mathbf{t} \in \mathbb{F}_2^\ell$
Output: $\mathcal{L}^{(1)} = \mathcal{L}_1^{(2)} \bowtie \mathcal{L}_2^{(2)}$

Set collision counter $C^\star \leftarrow 0$
$\triangleright$ Presort $\mathcal{L}_i^\star$:
For each $\mathbf{e}_i \in \mathcal{L}_i^{(2)}$
    Copy $\mathbf{e}_i$ into box $\mathcal{B}_l^i$ for all $l \in L(\mathbf{e}_i)$
$\triangleright$ Search collisions using Alg. 1.2:
For each Label $l$
    $(\mathcal{T}, \text{tmpC}) \leftarrow$ Merge-Join $(\mathcal{B}_l^1, \mathcal{B}_l^2, \ell, p, t)$
    $\mathcal{L}^\star \leftarrow \mathcal{L}^\star \cup \mathcal{T}$ (i.e. filter out duplicates)
    $C^\star \leftarrow C^\star + \text{tmpC}$
```

The running time of such a Predicted Merge-Join is

$$T_i^\star = \tilde{\mathcal{O}}\left(\max\left\{L_1^\star, L_2^\star, C^\star\right\}\right)$$

where $L^{(1)}$ can be omitted since $L^{(1)} \leq C^\star$. It remains to determine the expected values for $L_i^\star$ and $C^\star$. Since every element $\mathbf{e}_i$ is copied $\binom{p}{\varepsilon}$ times, we obtain

$$L_i^\star = |\mathcal{L}_i| \cdot \binom{p}{\varepsilon} \quad .$$

Moreover, we expect the $\mathbf{e}_i$ to distribute equally over the boxes $\mathcal{B}_l^i$ which yields

$$\mathbb{E}\left[|\mathcal{B}_l^i|\right] = \frac{L_i^\star}{\binom{m}{\varepsilon}} \quad .$$

For a fixed label l, it holds

$$\mathbb{E}\left[\text{tmpC}\right] = \frac{\mathbb{E}\left[|\mathcal{B}_l^1|\right]\mathbb{E}\left[|\mathcal{B}_l^2|\right]}{2^{\ell-r}} = \frac{L_1^\star \cdot L_2^\star}{\binom{m}{\varepsilon}^2 \cdot 2^{\ell-r}}$$

according to the analysis of the merge-join algorithm in section 1. This implies

$$\mathbb{E}\left[C\right] = \#\text{labels} \cdot \mathbb{E}\left[\text{tmpC}\right] = \frac{L_1^\star \cdot L_2^\star}{\binom{m}{\varepsilon} \cdot 2^{\ell-r}} \quad .$$

In summary, we see that the above described Predicted Merge-Join allows to reduce the number of collisions at the cost of expanding the lists to $L_j^\star$. Replacing a standard merge-join is of use for parameters $(p, m, \varepsilon, \ell, r)$ where the the expected number of collisions is significantly larger than the expected number of returned elements. This is the case if the vectors are sparse such that an intersection is rare. For a given parameter set, we can easily check whether

$T_i^* < T_i$ holds in which case we use a predicted merge-join. The PREDICTED MERGE-JOIN has no impact on the asymptotic analysis.

10.5. Implementation for McEliece parameters $[1024, 524, 50]$

To show the practicability of the extended representation technique to solve the SMP, we have implemented the algorithm COLUMNMATCH (algorithm 10.1) and solved the SMP for the original McEliece [Jor83, McE78] parameters $[n, k, \omega] = [1024, 524, 50]$.

An optimization of the parameters p, ℓ, ε_1 and ε_2 by using the exact formulas of section 10.2.3 leads to $p = 8$, $l = 46$, $\varepsilon_1 = 2$, $\varepsilon_2 = 1$ which implies $p_1 = 6, p_2 = 4$ and $p_3 = 2$ and allows for an easy implementation since we obtain lists of same length within each level. We set restrictions of $r_1 = 23$ bits and $r_2 = 13$ bits at the intermediate and last level, respectively. This choice guarantees a good chance of success, over 50% per random target triple $(\mathbf{t}_1^{(1)}, \mathbf{t}_1^{(2)}, \mathbf{t}_3^{(2)})$, reducing the size of the intermediate lists at the same time.

According to the analysis in section 10.2.3, we can expect to have about

$$C_2 \approx S_2^2 \cdot 2^{r_2 - r_1} \approx 2^{28.04}$$

colliding elements in level two where

$$S_2 \approx \binom{k+\ell}{p_2} \cdot 2^{-r_2} \approx 2^{19.02}$$

They are reduced to about

$$S_1 \approx \binom{k+\ell}{p_1} \cdot 2^{-r_1} \approx 2^{22.40}$$

elements as we create duplicates or colliding element that do not comply to the weight constraint. The probability that the sum of two colliding vectors from level two has the correct weight is

$$\mathcal{P}_2 = \frac{\binom{k+\ell-p_2}{p_2-\varepsilon_2}\binom{p_2}{\varepsilon_2}}{\binom{k+\ell}{p_2}} = 2.76\% \ .$$

So we can already estimate that only about $2.76\% \cdot C_2 \approx 2^{22}$ collisions are of value. We additionally exclude all that are duplicates or have too small weight (very few).
These observations motivate the use of a PREDICTED MERGE-JOIN in the second level as introduced in section 10.4. The algorithm reduces the number of collisions we need to treat at the cost of slightly increased lists at the second level. We denote the size of the lists by $S_2^\star$ and the number of collisions by $C_2^\star$ to distinguish them from the values observed by a standard MERGE-JOIN-DECODE .

We run the algorithm[3] on 1000 randomly chosen SMP instances for one random set of target values per instance. For a choice of $r_1 = 23$ and $r_2 = 13$, we can solve 504 instances in 20.49 minutes using 430 MB of memory.

[3] Intel® Core™ i7-2820QM CPU at 2.3GHz

Using the Predicted Merge-Join algorithm, the average number of colliding elements in the experiments is $C_2^\star \approx 7\,615\,276 \approx 2^{22.86}$ which is close to the average number of consistent elements, $S_1 \approx 4\,696\,298 \approx 2^{22.16}$, we observed. The loss is mainly due to duplicates. The lists in the level below are augmented by a factor $\binom{p_2}{\varepsilon_2}$ in comparison to a simple merge-join. We observe an average number of $S_2^\star \approx 2\,125\,074 \approx 2^{21.02}$ elements at the second level. Table 10.2 lists further details about the number of elements. It gives the minimal and maximal values as well as the standard deviation.

Table 10.2.: *Experimental number of intermediate elements for 1000 random instances with parameters:* $[n, k, \omega] = [1024, 524, 50]$, $p = 8$, $l = 46$, $\varepsilon_1 = 2$, $\varepsilon_2 = 1$, $r_1 = 23$ *and* $\mathbf{r_2 = 13}$.

List type	Min. size	Max. size	Mean	Standard deviation	Theoretical estimate
$S_2^\star$	2 115 812	2 136 036	2 125 074	2 903	2 111 739
$C_2^\star$	7 557 815	7 668 618	7 615 276	24 150	7 719 093
S_1	4 602 869	4 791 863	4 696 298	15 375	5 534 417

The complexity analysis of Predicted Merge-Join and our ColumnMatch estimates that

$$S_2^\star \approx S_2 \cdot \binom{p_2}{\varepsilon_2} \approx 2^{21.01} \approx 2\,111\,739$$

elements appear in the second level per list and that we observe

$$C_2^\star \approx \frac{(S_2^\star)^2}{\binom{k+\ell}{\varepsilon_2} \cdot 2^{r_2 - r_1}} \approx 2^{22.88} \approx 7\,719\,093$$

collisions when we merge two of the lists. We expect to collect about

$$S_1 \approx \binom{k+\ell}{p_2} \cdot 2^{-r1} \approx 2^{22.40} \approx 5\,534\,417$$

elements per list in level one. The estimates are close to the experimental values for $S_2^\star$ and $C_2^\star$. The theoretical value for S_1 counts the number of vectors of length $k + \ell$ and weight p_1 that comply to a constraint on r_1 bits. This is a simple estimate that neglects completely how we create the vectors. We expect to find less elements as they are constructed from vectors at the lower level that need to comply to weight and bit-constraints. The actual number of elements is smaller by a factor 0.85. As we solve about 50% of the instances, we can run them a second time with changed random targets. We achieve to solve 765 instances multiplying the time by 1.5.

Augmenting the constraints. In a second experiment, we slightly increase the restrictions as we proposed in section 10.2.4 and choose $r_2 = 14$. We expect to find solutions for 25% of the cases on average as we add an additional constraint of one bit on the elements in the second level.

For 1000 instances, we achieve to solve 257 in 10.27 minutes using 220 MB of memory, that is, we find only 26% of the index sets but reduce the time by one half due to the smaller lists. The memory requirement is at 51%.

The average number of colliding elements in the experiments is $C_2^\star \approx 3\,806\,860 \approx 2^{21.86}$ which is again close to the average number of consistent elements, $S_1 \approx 2\,863\,550 \approx 2^{21.45}$. We observe an average number of $S_2^\star \approx 1\,062\,488 \approx 2^{20.02}$ elements at the second level. Table 10.3 lists further details and shows that the experimental values for $S_2^\star$ and $C_2^\star$ match the estimates. The theoretical bound for S_1 differs from the experimental value by a factor one half. This is due to the simplified formula for S_1 as explained above. As we augment the constraints at the lower level, even less elements are found for level one.

Table 10.3.: *Experimental number of intermediate elements for 1000 random instances with parameters:* $[n, k, \omega] = [1024, 524, 50]$, $p = 8$, $l = 46$, $\varepsilon_1 = 2$, $\varepsilon_2 = 1$, $r_1 = 23$ *and* $\mathbf{r_2 = 14}$.

List type	Min. size	Max. size	Mean	Standard deviation	Theoretical estimate
$S_2^\star$	1 057 528	1 067 872	1 062 488	1 967	1 055 869
$C_2^\star$	3 781 376	3 833 337	3 806 860	9 911	3 859 546
S_1	2 813 372	2 912 795	2 863 550	16 810	5 534 417

We can repeat the search of a solution for the unsolved instances, about 75%, by choosing new independent random targets. The running time is multiplied by $1\frac{3}{4}$ and we solve 456 instances which is roughly 46%.

A different approach is a dependent choice of targets for the unsolved instances. We can run each instance first for randomly picked targets $(\mathbf{t}_1^{(2)}, \mathbf{t}_3^{(2)})$ of $r_2 = 14$ bits in level two. If we do not solve the problem, we change (only) the targets at the second level by flipping the last bit in $\mathbf{t}_1^{(2)}$ or $\mathbf{t}_3^{(2)}$, denoted by $\mathbf{t}_j'^{(2)}$. We need to perform four merge-joins for lists w.r.t the targets $(\mathbf{t}_1^{(2)}, \mathbf{t}_3^{(2)})$, $(\mathbf{t}_1^{(2)}, \mathbf{t}_3'^{(2)})$, $(\mathbf{t}_1'^{(2)}, \mathbf{t}_3^{(2)})$ and $(\mathbf{t}_1'^{(2)}, \mathbf{t}_3'^{(2)})$. We hence solve the same instances as for same targets where we impose no constraint on the 14th coordinate. We thus are back to the experiment above where $r_2 = 13$. This time, we only need about 220 MB at each run.

10.6. Upper bounds on bad matrices, length of lists and number of collisions

The algorithm of section 10 fails to find a solution with the claimed complexity if we are unlucky to pick good targets such that no or too many collisions appear at same point. The section theoretically bounds the risk. We provide bounds on the number of matrices which produce few different column sums such that we need to repeat the choice of the target many times. The second and third part estimates the number of matrices for which many targets lead to lists of length above the expected value or to too many collisions. In these cases, the memory or time is highly above the average case.

We make use of the following helpful theorem that can be obtained by a straightforward modification of the result in [NSS01, Theorem 3.2] (compare to section 4.1).

Theorem 10.1
For a fixed matrix $Q \in \mathbb{F}_2^{m\times n}$, a target vector $\mathbf{t} \in \mathbb{F}_2^m$ and an arbitrary set $\mathcal{B} \subset \mathbb{F}_2^n$, we define

$$P_Q(\mathcal{B}, \mathbf{t}) := \frac{1}{|\mathcal{B}|} \left|\{\mathbf{x} \in \mathcal{B} \ : \ Q\mathbf{x} = \mathbf{t}\}\right| \ .$$

Then for all $\mathcal{B} \subset \mathbb{F}_2^n$ it holds that

$$\frac{1}{2^{mn}} \sum_{Q \in \mathbb{F}_2^{m\times n}} \sum_{\mathbf{t}\in\mathbb{F}_2^m} (P_Q(\mathcal{B}, \mathbf{t}) - \frac{1}{2^m})^2 = \frac{2^m - 1}{2^m |\mathcal{B}|} \ .$$

The following deductions are very similar to what we already saw in the integer case in section 4.1.

10.6.1. Bad distribution of column sums for few matrices

Theorem 10.1 states that for matrix $Q \in \mathbb{F}_2^{m\times n}$ and a target vector $\mathbf{t} \in \mathbb{F}_2^m$ there are $|B|/2^m$ values $\mathbf{x} \in \mathcal{B}$ on average such that $Q\mathbf{x} = \mathbf{t}$. The set $\mathcal{B}$ is the set of binary vectors of length n. Let λ be a positive integer. We bound the number of matrices, denotes by F_λ, for which less than $2^m/\lambda$ targets can be obtained as a column sum. If our algorithm receives such a matrix as input, it will succeed only on a small fraction of targets which in return augments the running time. From theorem 10.1 we derive that

$$F(\lambda) \sum_{\mathbf{t}\in\mathbb{F}_2^m} \left(P_Q(\mathcal{B}, \mathbf{t}) - \frac{1}{2^m} \right)^2 \le \frac{2^m - 1}{2^m |\mathcal{B}|} 2^{mn} \ . \tag{10.12}$$

We can lower bound the sum using the following observations. We abbreviate $P_\mathbf{t} = P_Q(\mathcal{B}, \mathbf{t})$. Denote by N_0 the number of of targets that are missed, i.e., for which $P_\mathbf{t} = 0$.

By our assumption, we know that $N_0 = 2^m \frac{(\lambda-1)}{\lambda}$. We compute:

$$\begin{aligned}
\sum_{\mathbf{t}\in\mathbb{F}_2^m} \left(P_{\mathbf{t}} - \frac{1}{2^m}\right)^2 &= \sum_{\mathbf{t}\in\mathbb{F}_2^m, P_{\mathbf{t}}\neq 0} \left(P_{\mathbf{t}} - \frac{1}{2^m}\right)^2 + \frac{N_0}{2^{2m}} \\
&= \sum_{\mathbf{t}\in\mathbb{F}_2^m, P_{\mathbf{t}}\neq 0} \left(P_{\mathbf{t}}^2 - \frac{2P_{\mathbf{t}}}{2^m}\right) + \frac{2^m - N_0}{2^{2m}} + \frac{N_0}{2^{2m}} \\
&= \sum_{\mathbf{t}\in\mathbb{F}_2^m, P_{\mathbf{t}}\neq 0} P_{\mathbf{t}}^2 - \frac{2}{2^m} \underbrace{\sum_{\mathbf{t}\in\mathbb{F}_2^m P_{\mathbf{t}}\neq 0} P_{\mathbf{t}}}_{1} + \frac{1}{2^m} \\
&= \sum_{\mathbf{t}\in\mathbb{F}_2^m, P_{\mathbf{t}}\neq 0} P_{\mathbf{t}}^2 - \frac{1}{2^m} \ .
\end{aligned}$$

As $\sum_{\mathbf{t}\in\mathbb{F}_2^m, P_{\mathbf{t}}\neq 0} P_R = 1$, we have that $\sum_{\mathbf{t}\in\mathbb{F}_2^m, P_{\mathbf{t}}\neq 0} P_{\mathbf{t}}^2 \geq 2^m/\lambda \cdot (\lambda/2^m)^2 = \lambda/2^m$ such that

$$\sum_{\mathbf{t}\in\mathbb{F}_2^m} \left(P_{\mathbf{t}} - \frac{1}{2^m}\right)^2 \geq \frac{\lambda - 1}{2^m} \ .$$

We can now estimate F_λ by combining the results:

$$F_\lambda \leq \frac{2^m - 1}{(\lambda-1)|\mathcal{B}|} 2^{mn} \leq \frac{1}{(\lambda-1)} 2^{mn} \tag{10.13}$$

if we choose m such that $|\mathcal{B}| \geq 2^m - 1$. The fraction becomes arbitrarily small choosing m such that 2^m is slightly smaller than $|\mathcal{B}|$ and λ large enough.

10.6.2. Bound on the size of the lists

We give an upper bound on the number of matrices that lead to an overflow in the lists for many targets which allows us to estimate the size of the lists. Let $\lambda > 0$ be a parameter. Consider the number of matrices, H_λ, for which more than $2^m/(2\lambda)$ targets have a probability

$$P(Q, \mathbf{t}) \geq \frac{\lambda}{2^m} \ .$$

Due to theorem 10.1, we can say that

$$2^{mn} \frac{2^m - 1}{2^m |\mathcal{B}|} \geq H_\lambda \sum_{\mathbf{t}\in\mathbb{F}_2^m} \left(P(\mathcal{B}, \mathbf{t}) - \frac{1}{2^m}\right)^2 \geq H_\lambda \frac{2^m}{2\lambda} \frac{(\lambda-1)}{2^m}^2 = \frac{H_\lambda}{2^m} \frac{(\lambda-1)^2}{2\lambda} \ .$$

We obtain that

$$H_\lambda \leq 2^{mn} \frac{2^m - 1}{2^m} \frac{2^m}{|\mathcal{B}|} \frac{2\lambda}{(\lambda-1)^2} \leq 2^{mn} \frac{2\lambda}{(\lambda-1)^2} \tag{10.14}$$

for m chosen such that $2^m \leq |\mathcal{B}|$. The fraction can become arbitrarily small depending on the choice for λ.

We conclude that for a matrix which is not one out of the H_λ matrices above and for most targets (all but at most $2^m/(2\,\lambda)$), the size of the lists $\mathcal{L}_j^{(i)}$ is at most λ times the expected value $|\mathcal{B}|/2^m$:

$$L^{(i)} \leq \lambda \frac{|\mathcal{B}|}{2^m} \ .$$

10.6.3. Bounding the number of collisions

The input to a merge-join routine is a matrix $Q \in F_2^{m\times n}$, a target $t' \in \mathbb{F}_2^m$ and two lists of elements $\mathbf{e}_1, \mathbf{e}_2 \in \mathcal{B}$ such that the corresponding column sums are: $Q\mathbf{e}_1 = \mathbf{t}_L$ and $Q\mathbf{e}_2 = \mathbf{t}_R$ for some $\mathbf{t}_R, \mathbf{t}_L \in \mathbb{F}_2^m$. We call an element $(\mathbf{e}_1, \mathbf{e}_2) \in \mathcal{B} \times \mathcal{B}$ a collision if the target is matched: $Q(\mathbf{e}_1 + \mathbf{e}_2) = \mathbf{t}$. By construction, we have that the the target is matched on some $r \leq m$ coordinates: $(\mathbf{t}_L + \mathbf{t}_R)_{[r]} = \mathbf{t}_{[r]}$. The number of collisions can be computed as

$$C = \sum_{\mathbf{t}\in\mathbb{F}_2^m, \mathbf{t}_{[r]}=\mathbf{t}'} |\mathcal{B}| P_Q(\mathcal{B}, \mathbf{t}) \cdot |\mathcal{B}| P_Q(\mathcal{B}, \mathbf{t} + \mathbf{t}'_{[r]})$$

and upper bounded as

$$C \leq \sqrt{\sum_{\mathbf{t}\in\mathbb{F}_2^m, \mathbf{t}_{[r]}=\mathbf{t}_L} (|\mathcal{B}| P_Q(\mathcal{B}, \mathbf{t}))^2 \cdot \sum_{\mathbf{t}\in\mathbb{F}_2^m, \mathbf{t}_{[r]}=\mathbf{t}_R} (|\mathcal{B}| P_Q(\mathcal{B}, \mathbf{t}))^2} \ .$$

For a parameter λ, let G_λ be the number of matrices for which the number of collisions exceeds the expected value. More precisely, for which more than $2^r/(8\lambda)$ targets satisfy:

$$\sum_{\mathbf{t}\in\mathbb{F}_2^m, \mathbf{t}_{[r]}=\mathbf{t}'} P_Q(\mathcal{B}, \mathbf{t})^2 \geq \left(\frac{\lambda}{2^r}\right)^2 2^{r-m} \ .$$

From theorem 10.1, we derive that

$$\sum_{Q\in\mathbb{F}_2^{m\times n}} \sum_{\mathbf{t}\in\mathbb{F}_2^m} P_Q(\mathcal{B}, \mathbf{t})^2 = 2^{mn} \frac{2^m + |\mathcal{B}| - 1}{2^m |\mathcal{B}|} \ .$$

Hence by our assumption,

$$2^{mn} \frac{2^m + |\mathcal{B}| - 1}{2^m |\mathcal{B}|} \geq G_\lambda \sum_{\mathbf{t}\in\mathbb{F}_2^m} P_Q(\mathcal{B}, \mathbf{t})^2 \geq G_\lambda \frac{2^r}{(8\lambda)} \frac{\lambda^2}{2^{2r}} \frac{2^m}{2^r} \ .$$

The number of bad matrices can then be bounded:

$$G_\lambda \leq 2^{mn} \frac{2^m + |\mathcal{B}| - 1}{|\mathcal{B}|} \frac{8}{\lambda} \frac{1}{2^{2(m-r)}} \leq 2^{mn} \frac{8}{\lambda} \left(\frac{2^m}{|\mathcal{B}|} + 1\right) \leq 2^{mn} \frac{16}{\lambda}$$

for m such that $|\mathcal{B}| > 2^m$. For all other matrices, we derive a number of collisions $C \leq \frac{\lambda^2}{2^{2r}} \frac{1}{2^{m-r}} |\mathcal{B}|^2$.

10.7. A provable variant of COLUMNMATCH

One iteration within COLUMNMATCH requires to pick three target vectors $\mathbf{t}_1^{(1)} \in \mathbb{F}_2^{r_1}$ and $\mathbf{t}_1^{(2)}, \mathbf{t}_3^{(2)} \in \mathbb{F}_2^{r_2}$. At each run this is done independently and at random. To obtain an algorithm of provable running time and memory requirement, we repeatedly invoke COLUMNMATCH with different independent target values $\mathbf{t}_i^{(j)}$ and add abort criteria that prevent the lists or the number of collisions in the computation from growing unexpectedly high. The new algorithm is called PROVABLECM and performs the following.

Let $\Lambda = 2^{\gamma n}$ be a parameter for a fixed constant $\gamma > 0$. We first choose independently random target values 8Λ times. The targets are $\mathbf{t}_{1,i}^{(1)} \in \mathbb{F}_2^{(r_1)}$ and $\mathbf{t}_{1,j}^{(2)}, \mathbf{t}_{3,k}^{(2)} \in \mathbb{F}_2^{r_2}$ for $1 \leq i, j, k \leq 8\Lambda$. For each set out of the $(8\Lambda)^3$ targets, we call COLUMNMATCH$(\mathbf{t}_{1,i}^{(1)}, \mathbf{t}_{1,j}^{(2)}, \mathbf{t}_{3,k}^{(2)})$ until a solution is found. Every execution might abort if two many collisions occur or if one of the lists exceeds its expected size. The abort happens if the expected value is exceeded by more than a factor of Λ.
For this variant of COLUMNMATCH we can prove a complexity and success probability.

Theorem 10.2
For every $\gamma > 0$, the modified algorithm PROVABLECM *outputs a solution $\mathbf{e} \in \mathbb{F}_2^{k+\ell}$ of weight p to $Q\mathbf{e} = \mathbf{s}$ for a fraction of at least $1 - 60 \cdot 2^{-\gamma n}$ randomly chosen $Q \in \mathbb{F}_2^{\ell \times (k+\ell)}$ with probability at least $1 - \frac{3}{e^2} > \frac{1}{2}$ in time $\tilde{\mathcal{O}}\left(\mathcal{T}(p, \ell; \varepsilon_1, \varepsilon_2) \cdot 2^{3\gamma n}\right)$ where $\mathcal{T}(p, \ell; \varepsilon_1, \varepsilon_2)$ is defined as in (10.10).*

Proof: The basic algorithm COLUMNMATCH performs three levels of repeated creation of lists which it joins by a collision search as shown in figure 10.7.

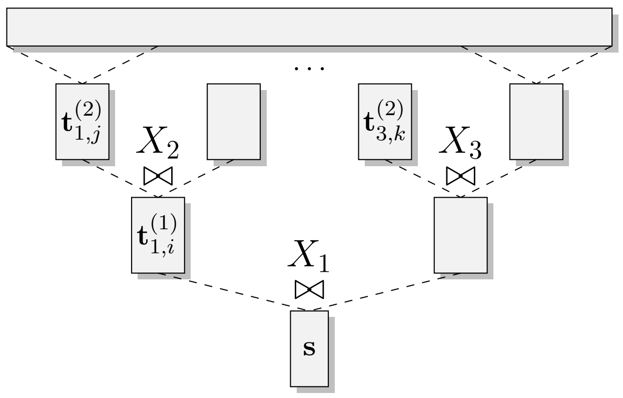


Figure 10.7.: *Illustration of different decomposition nodes.*

The solution is found if for each join-node $\bowtie$ at least one representation survives. It fails if the bit-constraint is not satisfied in at least one node, if too many collisions occur or the lists become too large.

For every such node, we introduce a random variable X_i indicating whether the algorithm fails at this point or not. The overall failure probability can then be upper bounded by using the union bound: $\mathbf{Pr}\left[\textsc{ProvableCM}\text{ fails }\right] \leq \sum \mathbf{Pr}\left[X_i = 0\right]$. We need to upper bound the failure probability of every single node. For this purpose, we divide every X_i into three events X_i^j and set $X_i := \prod X_i^j$. We now define these events for node X_1 as an example.

- X_1^1 represents the event that for at least one choice of the $\{\mathbf{t}_{1,i}^{(1)}\}$ the solution $\mathbf{e} \in \mathcal{L}$ has at least one representation $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ with $(Qe_1)_{[r_1]} = \mathbf{t}_{1,i}^{(1)}$ and $(Qe_2)_{[r_1]} = \mathbf{s}_{[r_1]} + \mathbf{t}_{1,i}^{(1)}$.
- X_1^2 represents the event that for at least one choice of the $\{\mathbf{t}_{1,i}^{(1)}\}$ the size of lists $\mathcal{L}_1^{(1)}$ and $\mathcal{L}_2^{(1)}$ do not exceed the expected value by more than a factor of $2^{\gamma n}$.
- X_1^3 represents the event that for at least one choice of the $\{\mathbf{t}_{1,i}^{(1)}\}$ the total number of collisions C_1 does not exceed its expected value by more than a factor of $2^{\gamma n}$.

All these events depend on the structure of the matrix Q and we need to exclude some pathological cases. We make use of the result in section 10.6. which we summarize:

1. For all but a $\frac{1}{\Lambda-1}$ fraction of the matrices, the proportion of bad targets w.r.t. to the event X_i^1 is smaller than $\frac{\Lambda-1}{\Lambda}$.
2. For all but a $\frac{2\Lambda}{(\Lambda-1)^2}$ fraction of the matrices, the proportion of bad targets w.r.t. to the event X_i^2 is smaller than $\frac{1}{2\Lambda}$.
3. For all but a $\frac{16}{\Lambda}$ fraction of the matrices, the proportion of bad targets w.r.t. to the event X_i^3 is smaller than $\frac{1}{4\Lambda}$.

One derives that the total fraction of matrices that lead to a fail in one node can be bounded by

$$\frac{1}{\Lambda-1} + \frac{2\Lambda}{(\Lambda-1)^2} + \frac{16}{\Lambda} \leq \frac{20}{\Lambda} \text{ for } \Lambda \geq 7 \enspace .$$

The total fraction of bad Q's for all three nodes is upper bounded by $\frac{60}{\Lambda}$. Furthermore, considering good Q's, the proportion of bad targets per node is given by

$$\frac{\Lambda-1}{\Lambda} + \frac{1}{2\Lambda} + \frac{1}{4\Lambda} = 1 - \frac{1}{4\Lambda}$$

and hence we have

$$\mathbf{Pr}\left[X_i = 0\right] = \mathbf{Pr}\left[\text{all } 8\Lambda \text{ many } \mathbf{t}\text{'s bad}\right] \leq \left(1 - \frac{1}{4\Lambda}\right)^{8\Lambda} \leq e^{-2} \enspace .$$

Eventually this yields

$$\mathbf{Pr}\left[\textsc{ProvableCM}\text{ fails }\right] \leq \frac{3}{e^2} < 41\%$$

for every good Q as stated in the theorem. Notice, that the worst-case running time of ProvableCM is given by a total number of $\Lambda^3 = 2^{3\gamma n}$ runs of ColumnMatch.

10.7.1. Trade-off between time and memory

We can reduce the time of our provable algorithm by augmenting the memory. We allow to increase the memory by a factor $\Lambda = 2^{\gamma n}$ by changing the targets $\mathbf{t}_1^{(2)}$ and $\mathbf{t}_3^{(2)}$ each 8Λ times. Note that this also changes the corresponding right targets $\mathbf{t}_2^{(2)} = \mathbf{t}_1^{(2)} + (\mathbf{t}_1^{(1)})_{[r_2]}$ and $\mathbf{t}_4^{(2)} = \mathbf{t}_3^{(2)} + (\mathbf{t}_2^{(1)})_{[r_2]}$, respectively. The list are of size $S_2 \cdot \Lambda$ and the cost for their creation is augmented by the same factor. We join all elements found per target $\mathbf{t}_1^{(2)}$ with the elements found for the corresponding target $\mathbf{t}_2^{(2)}$. We proceed analogously for the 8Λ targets $\mathbf{t}_3^{(2)}$. We abort and choose a different target whenever the number of collisions or the lists are larger than we expect. This results in lists at the first level of expected size $S_1 \cdot \Lambda$ and augments the collisions search by Λ. The last join needs to perform a merge on all elements which costs

$$\max(\Lambda S_1, C_1', N_{Sol})$$

where we expect that $C_1' \approx C_1 \Lambda^2$. Due to the very few solutions (one or a constant number), the number of collisions C_1 is small in comparison to S_1 and the cost of the last merge is dominated by the size of the lists, S_1. As long as we guarantee that $\Lambda^2 C_1 \leq \Lambda S_1$, the algorithm has a running time $\mathcal{T} \cdot \Lambda^2$ using space $S \cdot \Lambda$ where $\mathcal{T}$ and S are defined in (10.10).

Appendix A

Asymptotic approximations of binomials

A.1. Entropy

In information theory entropy measures uncertainty associated to a random variable. The concept was introduced by Shannon [Sha48] who quantified the expected value of the information contained in a message sent over an unreliable channel.

Definition A.1 (Shannon entropy)
Let X be a discrete random variable that takes values in $\{x_1, .., x_n\}$. Let $p(X)$ denote the probability mass function that gives the probability that X takes value x_i. The Shannon entropy can be written as

$$H_q(X) = -\sum_{i=1}^{n} p(x_i) \log_q p(x_i) \tag{A.1}$$

where q is the base of the logarithm.

The Shannon entropy is continuous and symmetric. It takes its maximal value if X is distributed uniformly. For $q = 2$ it is convention to denote H_2 by H. Closely related is the entropy function.

Definition A.2 (Entropy function)
Let q be an integer greater than 1. The q-entropy function, $h_q(\alpha) : (0, 1] \to \mathbb{R}$, is defined as follows:

$$h_q(\alpha) = \alpha \log_q(q-1) - \alpha \log_q(\alpha) - (1-\alpha) \log_q(1-\alpha) \ . \tag{A.2}$$

An interesting special case is the binary entropy function:

$$h(\alpha) = -\alpha \log_2(\alpha) - (1-\alpha) \log_2(1-\alpha) \ .$$

The function $h_q(\alpha)$ is continuous and increasing in the interval $[0, 1 - 1/q]$ where we define $h_q(0) = 0$ and $h_q(1 - 1/q) = 1$. Figure A.1 shows the graph of h and h_3.

The binary entropy function, $h(p)$, calculates the uncertainty of the outcome of a biased coin toss where heads has probability p and tails has probability $1 - p$ to come up. For a fair coin ($p = 1/2$) the uncertainty is maximal. Let X be a random variable taking values in

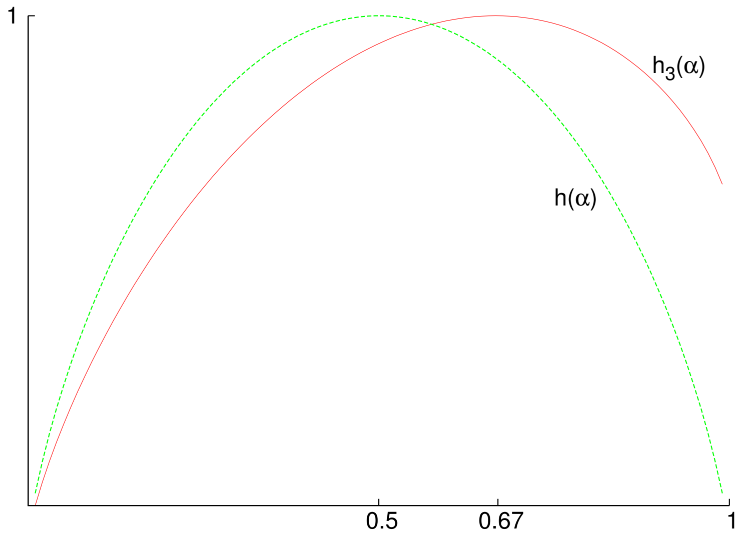


Figure A.1.: *Entropy function.*

$\{0,1\}$ such that $Prob(X = 1) = p$ and $Prob(X = 0) = 1 - p$. The Shannon entropy, $H(X)$, equals $h(p)$.

A.2. Asymptotic bounds for binomials and sums of binomials

Let n, q and k be integers. Let $\mathrm{Vol}_q(n,k)$ denote the volume of a Hamming ball or radius k in $\{0, 1, .., q-1\}$. The volume equals

$$\mathrm{Vol}_q(n,k) = \sum_{i=0}^{k} \binom{n}{i} (q-1)^i$$

and can be bounded [Bar98, section 1.2] by use of the entropy function (A.2):

$$c_k q^{nh_q(k/n)} \le \sum_{i=0}^{k} \binom{n}{i} (q-1)^i \le q^{nh_q(k/n)}$$

for $0 < k/n \le (q-1)/q$ and $c_k = 1/(\sqrt{8k(1-k/n)})$. We can hence write

$$\mathrm{Vol}_q(n,k) = \mathcal{O}\left(q^{nh_q(k/n)}\right) \quad .$$

The largest term of the sum is dominating in the asymptotic case and can thus be approximated for large n as follows:

$$\binom{n}{k} \approx \sum_{i=0}^{k} \binom{n}{i} (q-1)^i = \tilde{\mathcal{O}}\left(q^{nh_q(k/n)}\right) \quad .$$

We will also need to approximate multinomial coefficients of the form

$$\binom{n}{sn, tn, (1-s-t)n} = \frac{n!}{(sn)!(tn)!((1-s-t)n)!}$$

for $0 \leq s, t \leq 1$. Using Stirling's formula

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n (1 + o(1)) \ ,$$

we derive that

$$\binom{n}{sn, tn, (1-s-t)n} = \tilde{\mathcal{O}}\left(2^{na(s,t)}\right) \text{ where}$$
$$a(s,t) := -s\log_2 s - t\log_2 t - (1-s-t)\log_2(1-s-t) \ .$$

Appendix B

Octave code for optimisation

Octave code B.1: *Helper functions*

```
function y=xlog2(x)
  if (x<=0)
    y=0;
  else
    y=x*log(x)/log(2.0);
  endif
endfunction

#asymptotic value for binom(n,x n)
function y=H(x)
    y=-xlog2(x)-xlog2(1-x);
endfunction

#asymptotic value for binom(n x1,n  x2)
function y=H2(x1,x2)
 y=-xlog2(x2)-xlog2(x1-x2)+xlog2(x1);
endfunction

#from dimension get minimum distance due to gilbert varshamov bound
#bisection algorithm, start with a=0 and b=0.5
function y=findDrec(k,a,b,e)
    da=H(a);
    db=H(b);
    while(abs(da-db)>e)
        testb=a+(b-a)/2;
        testdb=H(testb);
        if(testdb > 1-k)
           b=a+(b-a)/2;
           db=H(b);
        else
           a=a+(b-a)/2;
           da=H(a);
        endif
     endwhile
     y=a;
endfunction
```

Octave code B.2: *Simple representation technique for ISD*

```
#include helper functions
function y=ComplexityMMT(params,w,k,info)
  p=params(1);
  l=params(2);
 if ((l>=0)&&(p>=0)&&(p<=w)&&(l<=1-k)&&(p<=k+l)&&(1-k-l>=w-p))
  lsize1=H2(k+l,p/2);
  d=H2(p,p/2);
 if ((d>lsize1))
     y=10000;
  else
   Cost(1)=lsize1-d;
   Cost(2)=H2((k+l)/2,p/4);
   Cost(3)=2*lsize1-2*d-(l-d);
   rep =H(w)-2*H2((k+l)/2,p/2)-H2((1-k-l),(w-p));
   y=rep+max(Cost);
   if (info)
       w,k
       rep
       Cost,p,l,d
    endif
   endif
  else
     y=10000;
   endif
endfunction
#_______________
#Input:
k =  0.46390
w =  0.061179
zze=[ 0.0064061   0.0278678]
ComplexityMMT(zze,w,k,1)

#Output:
#rep =  0.032170
#Cost =   0.021462   0.013934   0.021462
#compl =  0.053632
```

Octave code B.3: *Extended representation technique for ISD*

```
#include helper functions
function y=ComplexityNew(params,w,k,info)
  p=params(1);
  l=params(2);
  e1=params(3);
  e2=params(4);
 if ((l>=0)&&(p>=0)&&(e1>=0)&&(e2>=0)&&(p<=w)&&(l<=1-k)&&(p+e1<=k+l)&&(1-k-l>=w-p))
  lsize1=H2(k+l,p/2+e1);
  msize1=H2(p,p/2)+H2(k+l-p,e1);
  lsize2=H2(k+l,p/4+e1/2+e2);
  msize2=H2(p/2+e1,p/4+e1/2)+H2(k+l-p/2-e1,e2);
 if ((msize1>lsize1)||(msize2>lsize2)||(msize2>msize1))
     y=10000;
  else
   Cost(1)=2*lsize1-msize1-l;
   Cost(2)=lsize1-msize1;
   Cost(3)=2*lsize2-msize2-msize1;
   Cost(4)=lsize2-msize2;
   Cost(5)=lsize2/2;
   mem=max(Cost(2),Cost(4));
   y=H(w)-H2(k+l,p)-H2((1-k-l),(w-p))+max(Cost);
   if info
     rep=H(w)-H2(k+l,p)-H2((1-k-l),(w-p))
     Cost
     endif
  endif
 else
     y=10000;
  endif
endfunction

#_____________
#Input:
k =  0.44100
w =  0.13057
zze =[ 0.0546032   0.2074892   0.0105659   0.0015388]
ComplexityNew(zze,w,k,1)

#Output:
#rep =  0.024011
#Cost =  0.077759   0.077061  0.077759   0.077759   0.065563
#compl = 0.101770

#_____________
#Input:
k=0.7577
w=0.04
zzb=[2.8582e-02   1.4312e-01   6.6287e-03   3.9670e-04]
ComplexityNew(zzb,w,k,1)
#Output:
#rep =  0.0083409
#Cost = 0.058873   0.058612   0.058873   0.058873   0.042387
#compl =  0.067214
```

List of Algorithms

List of Figures

List of Tables

List of Code

Bibliography

[AD97] Miklós Ajtai and Cynthia Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *STOC*, pages 284–293, 1997. (Cited on page 36.)

[AFS05] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A family of fast syndrome based cryptographic hash functions. In *Proceedings of the 1st international conference on Progress in Cryptology in Malaysia*, Mycrypt'05, pages 64–83, Berlin, Heidelberg, 2005. Springer-Verlag. (Cited on pages 27 and 98.)

[Ajt98] Miklós Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions (extended abstract). In *STOC'98*, pages 10–19, 1998. (Cited on page 32.)

[Ale03] M. Alekhnovich. More on average case vs approximation complexity. *44th Symposium on Foundations of Computer Science (FOCS)*, pages 298–307, 2003. (Cited on pages 2 and 16.)

[Bar98] A. Barg. Complexity issues in coding theory. In V. S. Pless and W. C. Huffman, editors, *Handbook of Coding Theory*, volume I, chapter 7, pages 649–754. North-Holland, 1998. (Cited on pages 4, 17, 94, 96, 100, 109, 110, and 146.)

[Bar06] Gregory V. Bard. Accelerating cryptanalysis with the method of four russians. *IACR Cryptology ePrint Archive*, 2006:251, 2006. (Cited on page 107.)

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011. (Cited on pages 3, 17, 50, 65, 68, and 81.)

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012. (Cited on pages 4, 17, 100, 115, and 122.)

[Bli87] V.M. Blinovskii. Lower asymptotic bound on the number of linear code words in a sphere of given radius in $\mathbb{F}_q^n$. *In Probl. Peredach. Inform.*, 23:50–53, 1987. (Cited on page 113.)

[BLP08] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. In *PQCrypto*, pages 31–46, 2008. (Cited on pages 98, 100, and 107.)

[BLP11] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: ball-collision decoding. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011. (Cited on pages 100, 110, and 131.)

[BM97] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *EUROCRYPT*, pages 163–192, 1997. (Cited on page 28.)

[BMvT78] Elwyn R. Berlekamp, Robert J. Mceliece, and Henk C. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, 1978. (Cited on pages 9 and 94.)

[BO88] E.F. Brickell and A.M. Odlyzko. *Cryptanalysis: A Survey of Recent Results.* IEEE Proceedings. IEEE, 1988. (Cited on page 36.)

[BS08] Bhaskar Biswas and Nicolas Sendrier. McEliece cryptosystem implementation: Theory and practice. In *Proceedings of the 2nd International Workshop on Post-Quantum Cryptography*, PQCrypto '08, pages 47–62, Berlin, Heidelberg, 2008. Springer-Verlag. (Cited on page 99.)

[BS12] Gaetan Bisson and Andrew V. Sutherland. A low-memory algorithm for finding short product representations in finite groups. *Des. Codes Cryptography*, 63(1):1–13, April 2012. (Cited on page 82.)

[CC94] A. Canteaut and H. Chabanne. A further improvement of the work factor in an attempt at breaking McEliece's cryptosystem. In P. Charpin, editor, *EUROCODE 94, International Symposium on Coding Theory and Applications*, LNCS, 1994. (Cited on page 107.)

[CC98] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998. (Cited on pages 99, 100, 101, and 107.)

[CFS01] N. Courtois, M. Finiasz, and N. Sendrier. How to achieve a McEliece-based digital signature scheme. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 157–174. Springer, 2001. (Cited on pages 100 and 101.)

[CFS02] Nicolas Courtois, Matthieu Finiasz, and Nicolas Sendrier. Short McEliece-based digital signatures. In *Proceedings of the 2002 IEEE International Symposium on Information Theory*. IEEE, 2002. (Cited on page 101.)

[CG90] J.T. Coffey and R.M. Goodman. The complexity of information set decoding. *IEEE Transactions on Information Theory*, 36:1031–1037, 1990. (Cited on pages 2, 16, and 114.)

[CJ04] Jean-Sébastien Coron and Antoine Joux. Cryptanalysis of a provably secure cryptographic hash function. *IACR Cryptology ePrint Archive*, 2004:13, 2004. (Cited on page 27.)

[CJL$^+$92] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 2:111–128, 1992. (Cited on page 36.)

[CJM02] Philippe Chose, Antoine Joux, and Michel Mitton. Fast correlation attacks: An algorithmic point of view. In *EUROCRYPT*, pages 209–221, 2002. (Cited on page 27.)

[CS98] A. Canteaut and N. Sendrier. Cryptanalysis of the original McEliece cryptosystem. In *Advances in Cryptology - ASIACRYPT'98*, number 1514 in LNCS, pages 187–199. Springer-Verlag, 1998. (Cited on pages 99 and 100.)

[Dal10] Léonard Dallot. *Sécurité de protocoles cryptgraphiques fondés sur les codes correcteurs d'erreurs*. PhD thesis, Unversité de Caen, July 2010. (Cited on page 93.)

[Des88] Y.G. Desmedt. What happened to the knapsack cryptographic scheme? *Netherlands: Kluwer Academic Publishers*, Vol. 142, 1988. (Cited on page 36.)

[Dum91] I. Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52, Moscow, 1991. (Cited on pages 100, 109, and 110.)

[Dum98] Ilya Dumer. Two decoding algorithms for linear codes. *Problemy Peredachi Informatsii*, 25(1):24–32, 1998. (Cited on page 109.)

[DV09] L. Dallot and D. Vergnaud. Provably secure code-based threshold ring signatures. In Matthew G. Parker, editor, *12th IMA international conference, Cryptography and Coding 2009*, volume 5921, pages 222–235, Cirencester, UK, 2009. (Cited on page 98.)

[FG84] A.M. Frieze and Carnegie-Mellon University. Management Sciences Research Group. *On the Lagarias-Odlyzko Algorithm for the Subset Sum Problem*. Management sciences research report. Management Sciences Research Group, Graduate School of Industrial Administration, Carnegie-Mellon University, 1984. (Cited on page 36.)

[Fin10] Matthieu Finiasz. Parallel-CFS – strengthening the CFS McEliece-based signature scheme. In *Selected Areas in Cryptography*, pages 159–170, 2010. (Cited on page 105.)

[FOPT10a] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. A Distinguisher for High Rate McEliece Cryptosystems. eprint Report 2010/331, 2010. (Cited on page 99.)

[FOPT10b] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic Cryptanalysis of McEliece variants with compact keys. In *Proceedings of Eurocrypt 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 279–298. Springer Verlag, 2010. (Cited on page 99.)

[FS96] Jean-Bernard Fischer and Jacques Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In *EUROCRYPT*, pages 245–255, 1996. (Cited on page 99.)

[FS09] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In M. Matsui, editor, *Asiacrypt 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009. (Cited on pages 9, 100, 102, 110, 111, and 116.)

[GM91] Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. In *ICALP*, pages 719–727, 1991. (Cited on page 36.)

[Gop71a] Valery D. Goppa. A new class of linear error correcting codes. *Problemy Peredachi Informatsii*, 6(3):24–30, 1971. (Cited on page 97.)

[Gop71b] Valery D. Goppa. Rational representation of codes and (L, g)-codes. *Problemy Peredachi Informatsii*, 7(3):41–49, 1971. (Cited on page 97.)

[HB01] N.J. Hopper and M. Blum. Secure human identification protocols. In *Proceedings of Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 52–66, 2001. (Cited on pages 2 and 16.)

[HGJ10] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010. (Cited on pages 3, 6, 17, 39, 43, 47, 48, 50, and 60.)

[HP03] W. C. Huffman and V. Pless. *Fundamentals of error-correcting codes*. Cambridge Univ. Press, 2003. (Cited on pages 93 and 98.)

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21:277–292, April 1974. (Cited on pages 5 and 37.)

[IN89] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. In *FOCS*, pages 236–241, 1989. (Cited on pages 5 and 36.)

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptology*, 9(4):199–216, 1996. (Cited on pages 5 and 36.)

[JG94] Antoine Joux and Louis Granboulan. A practical attack against knapsack based hash functions (extended abstract). In *EUROCRYPT*, pages 58–66, 1994. (Cited on page 36.)

[JJ02] Thomas Johansson and Fredrik Jönsson. On the complexity of some cryptographic problems based on the general decoding problem. *IEEE Transactions on Information Theory*, 48(10):2669–2678, 2002. (Cited on page 102.)

[JL11] Thomas Johansson and Carl Löndahl. An improvement to Stern's algorithm. Report, Lund University, 2011. (Cited on pages 115 and 116.)

[Jor83] John P. Jordan. A variant of a public key cryptosystem based on goppa codes. *SIGACT News*, 15:61–66, January 1983. (Cited on page 135.)

[Jou09] Antoine Joux. *Algorithmic Cryptanalysis*. Chapman & Hall/CRC, 1st edition, 2009. (Cited on pages 3 and 17.)

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. (Cited on pages 4, 31, 32, and 34.)

[KKC$^+$01] E. Kiltz, K.Pietrzak, D. Cash, A. Jain, and D. Venturi. Efficient authentication from hard learning problems. In *In Advances in Cryptology - EUROCRYPT 2011*, pages 7–26. Springer, 2001. (Cited on pages 2 and 16.)

[Knu81] Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, 2nd edition, 1981. (Cited on page 82.)

[Knu98] D.E. Knuth. *Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley Professional, 2 edition, 1998. (Cited on page 22.)

[Lai01] Ming Kin Lai. Knapsack cryptosystems: The past and the future. Term paper, April 2001. (Cited on page 36.)

[LB88] P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *Advances in cryptology—EUROCRYPT '88*, volume 330 of *LNCS*, pages 275–280, 1988. (Cited on page 108.)

[LDW94] Yuan Xing Li, Robert H. Deng, and Xin Mei Wang. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, 40(1):271–273, 1994. (Cited on pages 9, 94, and 99.)

[Lev88] L.B. Levitin. Covering radius of almost all linear codes satisfies the goblick bound. *In* IEEE Internat. Symp. on Information Theory, 1988. (Cited on page 114.)

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982. (Cited on page 36.)

[LO85] Jeffrey C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985. (Cited on page 36.)

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *TCC*, pages 382–400, 2010. (Cited on page 36.)

[Lyu05] Vadim Lyubashevsky. On random high density subset sums. *Electronic Colloquium on Computational Complexity (ECCC)*, (007), 2005. (Cited on page 28.)

[McE78] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. In *Jet Propulsion Laboratory DSN Progress Report* 42–44, pages 114–116, 1978. (Cited on pages 98, 107, and 135.)

[MCGL11] Carlos Aguilar Melchor, Pierre-Louis Cayrel, Philippe Gaborit, and Fabien Laguillaumie. A new efficient threshold ring signature scheme based on coding theory. *IEEE Transactions on Information Theory*, 57(7):4833–4842, 2011. (Cited on page 98.)

[MGS11] Carlos Aguilar Melchor, Philippe Gaborit, and Julien Schrek. A new zero-knowledge code based identification scheme with reduced communication. *CoRR*, abs/1111.1644, 2011. (Cited on page 98.)

[MH78] Ralph C Merkle and Martin E Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, 1978. (Cited on page 35.)

[MM11] Alexander Meurer and Alexander May. Personal communication. 2011. (Cited on pages 50 and 116.)

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\mathcal{O}(2^{0.054n+o(n)})$. In Dong Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer Berlin / Heidelberg, 2011. (Cited on pages 100, 112, 115, 119, and 131.)

[Moo05] Todd K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005. (Cited on page 97.)

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, New York; Elsevier Science Publishers B. V., 1977. (Cited on pages 93, 97, and 98.)

[MS09] Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm. In *SODA*, pages 586–595, 2009. (Cited on pages 27 and 102.)

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15:19–34, 1986. (Cited on page 99.)

[NPS01] David Naccache, David Pointcheval, and Jacques Stern. Twin signatures: an alternative to the hash-and-sign paradigm. In *ACM Conference on Computer and Communications Security*, pages 20–27, 2001. (Cited on page 105.)

[NSS01] Phong Q. Nguyen, Igor E. Shparlinski, and Jacques Stern. Distribution of modular sums and the security of the server aided exponentiation. In *Progress in Computer Science and Applied Logic*, volume 20 of *Final proceedings of Cryptography and Computational Number Theory workshop, Singapore (1999)*, pages 331–224, 2001. (Cited on pages 45 and 138.)

[Odl90] A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In C. Pomerance, editor, *Cryptology and Computational Number Theory*, number 42 in Proc. Symp. Appl. Math., pages pp. 75–88. Am. Math. Soc., 1990. (Cited on page 36.)

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *Proc. of STOC '09*, pages 333–342. ACM, 2009. (Cited on page 36.)

[Pet10] Christiane Peters. Information-set decoding for linear codes over F_q. In *PQCrypto*, pages 81–94, 2010. (Cited on pages 2, 16, and 98.)

[Pet11] Christiane Peters. *Curves, Codes, and Cryptography*. PhD thesis, Technische Universiteit Eindhoven, the Netherlands, 2011. (Cited on pages 4 and 17.)

[Pra62] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transaction on Information Theory*, 8(5):5–9, 1962. (Cited on page 107.)

[Reg04] Oded Regev. New lattice-based cryptographic constructions. In *J. ACM*, volume 51, pages 899–942, 2004. (Cited on page 36.)

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 84–93, New York, 2005. ACM. (Cited on pages 2, 16, and 36.)

[Sch72] J.P.M. Schalkwijk. An algorithm for source coding. *IEEE Transactions on Information Theory*, 18(3):395–399, May 1972. (Cited on page 99.)

[Sch87] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987. (Cited on page 36.)

[Sen00] N. Sendrier. Finding the permutation between equivalent codes: the support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, July 2000. (Cited on page 99.)

[Sen11] N. Sendrier. Decoding one out of many. In B.-Y. Yang, editor, *PQCrypto 2011*, volume 7071 of *LNCS*, pages 51–67. Springer, 2011. (Cited on page 102.)

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27(379):623–656, 1948. (Cited on pages 97 and 145.)

[Sha82] Adi Shamir. A polynomial time algorithm for breaking the basic merkle-hellman cryptosystem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 145–152, Washington, DC, USA, 1982. IEEE Computer Society. (Cited on page 35.)

[Sha08] Andrew Shallue. An improved multi-set algorithm for the dense subset sum problem. In *ANTS*, pages 416–429, 2008. (Cited on page 28.)

[SS81] Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2}), S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981. (Cited on pages 5 and 37.)

[SS92] V. Sidelnikov and S. Shestakov. On insecurity of cryptosystems based on generalized reed-solomon codes. *Discrete Math. Appl.*, 2(4):439–444, 1992. (Cited on page 99.)

[Ste89] Jacques Stern. A method for finding codewords of small weight. In *Proceedings of the 3rd International Colloquium on Coding Theory and Applications*, pages 106–113, London, UK, 1989. Springer-Verlag. (Cited on pages 10 and 109.)

[Ste93] Jacques Stern. A new identification scheme based on syndrome decoding. In *CRYPTO*, pages 13–21, 1993. (Cited on page 98.)

[Sti02] D. R. Stinson. Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem. *Math. Comput.*, 71(237):379–391, January 2002. (Cited on page 39.)

[TG62] Jr. T.J. Goblick. Coding for a discrete information source with a distortion measure. *Ph.D. dissertation, Dept. of Elect. Eng., M.I.T., Cambridge, MA,*, 1962. (Cited on page 113.)

[TS86] Gottfried Tinhofer and H. Schreck. The bounded subset sum problem is almost everywhere randomly decidable in $\mathcal{O}(n)$. *Inf. Process. Lett.*, 23(1):11–17, 1986. (Cited on page 36.)

[vOW96] Paul C. van Oorschot and Michael J. Wiener. Improving implementable meet-in-the-middle attacks by orders of magnitude. In *CRYPTO*, pages 229–236, 1996. (Cited on page 81.)

[Wag02] David Wagner. A generalized birthday problem. In *CRYPTO'2002*, pages 288–303, 2002. (Cited on pages 22, 27, and 102.)

Résumé

Cette thèse porte sur les techniques algorithmiques pour résoudre des instances uniformes du problème du sac à dos exact (subset sum) et du décodage à distance d'un code linéaire aléatoire.

Le subset sum est une alternative aux problèmes utilisés classiquement en cryptographie (comme le problème de la factorisation et du logarithme discret). Il admet une description simple et ne nécessite de réaliser qu'une somme de nombre entiers. De plus, aucun algorithme quantique polynomial n'est connu pour résoudre ou approcher ce problème. Il est possible de construire des fonctions à sens unique, des générateurs de nombres pseudo aléatoires et des schémas de chiffrement à clé publique dont la sécurité est basée sur la difficulté du problème dans le cas moyen.

Les problèmes de décodage peuvent être vus comme une version vectorielle du problème du subset sum. Plus particulièrement le problème du décodage borné dans un code aléatoire, est à la base de plusieurs schémas cryptographiques. Il admet des schémas de chiffrement à clé publique, de signature numérique, d'identification et des fonctions de hachage.

Nous présentons différentes techniques algorithmiques génériques pour résoudre ces problèmes. En utilisant la technique de représentation généralisée, nous obtenons un algorithme pour le problème du subset sum dont la complexité en temps asymptotique est diminuée d'un facteur exponentiel dans le pire des cas. Nous montrons que la même technique s'applique dans le domaine des codes. Ce résultat permet d'améliorer le décodage par ensemble d'information qui résout le problème de décodage dans un code aléatoire. Le nouvel algorithme diminue la complexité en temps asymptotique d'un facteur exponentiel.

Abstract

The focus of this thesis is an algorithmic technique to solve the random, hard subset-sum problem and the distance-decoding problem in a random linear code.

The subset-sum problem provides an alternative to other hard problems used in cryptography (e.g., factoring or the discrete logarithm problem). Its description is simple and the computation of sums of integers is an easy task. Furthermore, no polynomial-time quantum algorithm for solving general knapsacks is known. One can construct one-way functions, pseudo-random generators and private-key encryption schemes from the hardness assumption of the average-case problem. Also some cryptosystems based on lattice problems are provably as secure as the difficulty of the average-case subset-sum problem.

Decoding problems can be seen as a vectorial subset-sum problem. Of particular interest is the bounded-distance-decoding problem in a random code. It permits public-key encryption, digital signatures, identification schemes and hash-functions.

We present different generic algorithmic tools to solve the above problems. By use of our extended representation technique, we obtain an algorithm of exponentially lower asymptotic running time than previous approaches for the hardest case of a random subset-sum problem. We show that the technique can be applied to the domain of code-based cryptography. This results in improved information-set decoding that solves the distance-decoding problem for random linear codes. The new algorithm is asymptotically faster by an exponential factor.