Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Βελτιωμένοι αλγόριθμοι για προβλήματα αθροίσματος υποσυνόλων

Improved algorithms for subset sum problems

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΕΜΜΑΝΟΥΗΛ ΒΑΣΙΛΑΚΗΣ

Επιβλέπων : Αριστείδης Παγουρτζής
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2021

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Βελτιωμένοι αλγόριθμοι για προβλήματα αθροίσματος υποσυνόλων

Improved algorithms for subset sum problems

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΕΜΜΑΝΟΥΗΛ ΒΑΣΙΛΑΚΗΣ

Επιβλέπων : Αριστείδης Παγουρτζής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 3η Ιουνίου 2021.

Αριστείδης Παγουρτζής Καθηγητής Ε.Μ.Π.	Δημήτριος Φωτάκης Αν. Καθηγητής Ε.Μ.Π.	Ευάγγελος Μαρκάκης Αν. Καθηγητής Ο.Π.Α.

Αθήνα, Ιούνιος 2021

Εμμανουήλ Βασιλάκης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Μπροστά σε κύματα θολά
μιας νιότης που σιγοβράζει
αλήθεια, τι μας καρτερά
στο τέλος αυτού του δρόμου

Περίληψη

Στην παρούσα διπλωματική εργασία, παρουσιάζουμε τέσσερις αλγορίθμους ψευδοπολυωνυμικού χρόνου για το πρόβλημα Equal Subset Sum, οριζόμενο ως εξής: δεδομένου ενός συνόλου θετικών ακεραίων Z πληθυκότητας n, να προσδιοριστεί εάν υπάρχουν δύο ξένα υποσύνολα του Z, τα στοιχεία των οποίων αθροίζουν στην ίδια τιμή. Υποθέτοντας επιπλέον ότι μας παρέχεται και ένα άνω όριο $t > 0$ όσον αφορά το άθροισμα των δύο υποσυνόλων, μία κλασσική προσέγγιση κάνοντας χρήση δυναμικού προγραμματισμού μπορεί να λύσει το πρόβλημα σε χρόνο $\mathcal{O}(nt)$. Αξιοποιούμε την πρόσφατη πρόοδο σε σχέση με το πρόβλημα Subset Sum εξαιτίας των Koiliaris και Xu [Koil19] αλλά και του Bringmann [Brin17] με στόχο να σχεδιάσουμε γρηγορότερους αλγορίθμους για το πρόβλημα Equal Subset Sum, και πιο συγκεκριμένα για την *εκδοχή αναζήτησης*, όπου ζητούμενο είναι επιπλέον και η ανακατασκευή των συνόλων της λύσης. Αναπτύσσουμε τρεις αλγορίθμους που βασίζονται σε αυτήν την πρόοδο: έναν τυχαιοκρατικό χρονικής πολυπλοκότητας $\tilde{\mathcal{O}}(n + t)$, έναν ντετερμινιστικό με χρονική πολυπλοκότητα $\tilde{\mathcal{O}}(\sigma)$ και έναν ντετερμινιστικό χρονικής πολυπλοκότητας $\tilde{\mathcal{O}}(\sqrt{n}t)$, όπου με σ συμβολίζουμε το συνολικό άθροισμα των στοιχείων του συνόλου εισόδου. Επιπλέον, αναπτύσσουμε έναν απλό και αποδοτικό ντετερμινιστικό αλγόριθμο χρονικής πολυπλοκότητας $\tilde{\mathcal{O}}(n + t)$, που όμως δεν φαίνεται να μπορεί να επεκταθεί σε πιο γενικά προβλήματα.

Εμπνεόμενοι από αυτές τις επεκτάσεις, επεκτείνουμε περαιτέρω τις τεχνικές μας με σκοπό να αντιμετωπίσουμε μία πιο γενική παραλλαγή του προβλήματος Equal Subset Sum, που ονομάζεται k-Subset Sum, όπου αναζητούμε εάν υπάρχουν k ξένα υποσύνολα του συνόλου της εισόδου, τα στοιχεία του καθενός εκ των οποίων αθροίζουν σε μία συγκεκριμένη δοθείσα τιμή t_i αντίστοιχα. Προτείνουμε δύο αλγορίθμους για την εκδοχή απόφασης του προβλήματος k-Subset Sum: έναν τυχαιοκρατικό, χρόνου $\tilde{\mathcal{O}}(n + t^k)$ και έναν ντετερμινιστικό, χρόνου $\tilde{\mathcal{O}}(n^{k/k+1}t^k)$ αντίστοιχα. Τέλος, δείχνουμε πώς αυτοί οι αλγόριθμοι μπορούν να χρησιμοποιηθούν για την επίλυση των εκδοχών απόφασης διαφόρων παρεμφερών προβλημάτων, όπως λόγου χάρη τα προβλήματα Subset Sum Ratio, k-Subset Sum Ratio και Multiple Subset Sum.

Λέξεις κλειδιά

subset sum, equal subset sum, k-subset sum, γρήγορος μετασχηματισμός fourier, color-coding, αλγόριθμοι ψευδοπολυωνυμικού χρόνου.

Abstract

In this diploma dissertation, we present four pseudopolynomial time algorithms for the Equal Subset Sum problem, defined as follows: given a set Z of n positive integers, determine whether there exist two disjoint subsets of Z, the elements of which sum up to the same value. Assuming a given upper bound $t > 0$ on the sum of the two subsets, a standard dynamic programming approach can solve the problem in $\mathcal{O}(nt)$ time. We build on recent advances on Subset Sum due to Koiliaris and Xu [Koil19] and Bringmann [Brin17] in order to provide faster algorithms for Equal Subset Sum, and in particular for its *search version*, where a reconstruction of the solution sets is also sought. We devise three algorithms based on those advances: a randomised one of time complexity $\tilde{\mathcal{O}}(n + t)$, a deterministic one of $\tilde{\mathcal{O}}(\sigma)$ complexity and a deterministic $\tilde{\mathcal{O}}(\sqrt{n}t)$ time one, where σ is the total sum of the elements of the input set. Additionally, we present a simple and efficient $\tilde{\mathcal{O}}(n+t)$ deterministic algorithm, which however does not seem to be able to be extended to more general problems.

Inspired by these extensions, we further extend our techniques in order to cope with a more general variation of Equal Subset Sum, called k-Subset Sum, which asks for k disjoint subsets, whose elements sum up to a certain given value t_i respectively. We propose two algorithms for the decision version of k-Subset Sum: one running in randomised $\tilde{\mathcal{O}}(n+t^k)$ time and a deterministic $\tilde{\mathcal{O}}(n^{k/k+1}t^k)$ time respectively. We further demonstrate how these algorithms can be used to solve the decision versions of closely related variations of k-Subset Sum, namely Subset Sum Ratio, k-Subset Sum Ratio and Multiple Subset Sum.

Key words

subset sum, equal subset sum, k-subset sum, fast fourier transform, color-coding, pseudopolynomial time algorithms.

Ευχαριστίες

Αρχικά, οφείλω ένα μεγάλο ευχαριστώ στον επιβλέποντα καθηγητή της παρούσας διπλωματικής εργασίας, κ. Αριστείδη Παγουρτζή, καθώς και στους διδακτορικούς φοιτητές Αντώνη Αντωνόπουλο και Σταύρο Πετσαλάκη, για την ευχάριστη και απρόσκοπτη συνεργασία μας, καθώς και για τις διάφορες υποδείξεις και την πολύτιμη βοήθειά τους. Επιπλέον, θα ήθελα να ευχαριστήσω θερμά τον κ. Δημήτριο Φωτάκη για τις πολύτιμες συμβουλές του, καθώς και το έτερο μέλος της εξεταστική επιτροπής, κ. Ευάγγελο Μαρκάκη. Για την συγγραφή αυτής της διπλωματικής χρησιμοποιήθηκε το LaTeX template που αναπτύχθηκε από το εργαστήριο softlab της σχολής.

Η εργασία αυτή σηματοδοτεί το τέλος μίας πολύχρονης και καθοριστικής περιόδου της ζωής μου. Φυσικά, τίποτα δεν θα ήταν το ίδιο χωρίς τους φίλους και τις φίλες μου, παλιούς και νέους, που με συντροφεύουν σε αυτό το όμορφο ταξίδι. Οι παλιοί, που συμπορευόμαστε από μικρά παιδιά και μεγαλώσαμε μαζί (και ακόμα μεγαλώνουμε...), και οι νέοι, χωρίς τους οποίους τα χρόνια των σπουδών μου θα ήταν πολύ πιο ανιαρά, κουραστικά και άδεια, μου υπενθυμίζουν συνεχώς πόσο τυχερός είμαι που τους έχω δίπλα μου.

Πάνω από όλα όμως, δε μπορώ παρά να εκφράσω την βαθύτατη ευγνωμοσύνη και αγάπη μου στην οικογένειά μου, που τόσα χρόνια με στηρίζει, με καθοδηγεί και μου συμπαραστέκεται, δίχως την οποία δεν θα μπορούσα ποτέ μου να φτάσω μέχρι εδώ. Ελπίζω να τους κάνω υπερήφανους.

Εμμανουήλ Βασιλάκης,

Αθήνα, 3η Ιουνίου 2021

Contents

List of Figures

Κεφάλαιο 0

Εκτεταμένη Ελληνική Περίληψη

Σε αυτό το κεφάλαιο θα παρουσιάσουμε συνοπτικά την παρούσα διπλωματική εργασία μέσα από μία εκτεταμένη περίληψη στα Ελληνικά. Θα αποφευχθούν οι τεχνικές λεπτομέρειες και οι αποδείξεις, οι οποίες παρουσιάζονται ενδελεχώς στο αγγλικό κείμενο, και θα εστιάσουμε στην ουσία των αποτελεσμάτων μας.

0.1 Εισαγωγή

Δοθέντος ενός συνόλου S θετικών ακεραίων και ενός στόχου t, εξετάστε εάν υπάρχει κάποιο υποσύνολο $Z \subseteq S$, όπου το άθροισμα των στοιχείων του ισούται με t. Το παραπάνω πρόβλημα, που ονομάζεται Subset Sum, αποτελεί ένα από τα αρχικά NP-πλήρη προβλήματα, όπως παρουσιάστηκαν από τον Karp στο [Karp72], και θεωρείται ένα από τα πιο θεμελιώδη αλγοριθμικά προβλήματα, μαζί με τα προβλήματα Knapsack, SAT και άλλα. Πολλά εισαγωγικά πανεπιστημιακά μαθήματα αλγορίθμων παρουσιάζουν την έννοια της ψευδοπολυωνυμικής πολυπλοκότητας μέσω του κλασσικού αλγορίθμου του Bellman για το Subset Sum, όπως παρουσιάζεται στο [Bell57]. Παρότι πρόκειται για ένα ενδελεχώς μελετημένο αλγοριθμικό πρόβλημα, τα τελευταία χρόνια έχουν γίνει σημαντικές βελτιώσεις όσον αφορά την επιλυσιμότητά του σε ψευδοπολυωνυμικό χρόνο. Νέοι ντετερμινιστικοί [Koil19] και τυχαιοκρατικοί [Brin17, Jin19] αλγόριθμοι έχουν προταθεί, αποτελώντας τις πρώτες ουσιώδεις βελτιώσεις σε σχέση με την καθιερωμένη κλασσική προσέγγιση του Bellman [Bell57] καθώς και την βελτίωση του Pisinger [Pisi99].

Το πρόβλημα Equal Subset Sum είναι ένα λιγότερο μελετημένο, εντούτοις αξιοσημείωτο πρόβλημα που έχει προσελκύσει την προσοχή αρκετών ερευνητών, λόγω των ενδιαφερουσών εφαρμογών που βρίσκει σε διάφορα πεδία, όπως λόγου χάρη η υπολογιστική βιολογία [Ciel03b, Ciel04], η υπολογιστική κοινωνική επιλογή [Lipt04] και η κρυπτογραφία [Volo17]. Επιπλέον, σχετίζεται με σημαντικές θεωρητικές έννοιες, όπως παραδείγματος χάρη την πολυπλοκότητα των προβλημάτων αναζήτησης στην κλάση TFNP [Papa94]. Σε αυτήν την διπλωματική εργασία, θεωρούμε την περίπτωση όπου μας δίνεται επιπλέον ένα άνω όριο t και αναζητούμε δύο ξένα υποσύνολα ίσου αθροίσματος, το οποίο φράσσεται από το t.

Μία πιο γενική εκδοχή του προβλήματος, στην οποία θα αναφερόμαστε ως k-Subset Sum, ζητά να βρεθεί εάν υπάρχουν k ξένα υποσύνολα, τα αθροίσματα των στοιχείων των οποίων ισούνται με δοθείσες τιμές $t_i, i = 1, \ldots, k$ αντίστοιχα. Μπορεί κανείς να διαπιστώσει πως ακόμη και στην περίπτωση όπου $k = 2$ και $t_1 = t_2$, το πρόβλημα γίνεται πολύ πιο δύσκολο. Το πρόβλημα αυτό μπορεί να θεωρηθεί ως μία στοχευμένη εκδοχή του Equal Subset Sum, για την οποία δεν μπορούμε να αξιοποιήσουμε το ότι τα σύνολα της λύσης θα είναι ξένα μεταξύ τους λόγω κάποιου επιχειρήματος ελαχιστότητας. Μία ενδιαφέρουσα ειδική περίπτωση είναι όταν το άθροισμα των τιμών - στόχων t_i ισούται το άθροισμα των στοιχείων του συνόλου εισόδου, επομένως αναζητούμε μία διαμέριση των στοιχείων του συνόλου εισόδου σε υποσύνολα συγκεκριμένων αθροισμάτων. Μία ακόμη πιο ειδική περίπτωση είναι αυτή στην οποία θέλουμε να διαμερίσουμε το σύνολο εισόδου σε k υποσύνολα ίσου αθροίσματος. Η τελευταία περίπτωση βρίσκει εφαρμογές σε περιπτώσεις δίκαιων κατανομών.

Σχετική Βιβλιογραφία

Το πρόβλημα Equal Subset Sum και η εκδοχή βελτιστοποίησής του, που στη βιβλιογραφία καλείται Subset Sum Ratio [Bazg02], σχετίζονται με διάφορα προβλήματα που παρουσιάζονται σε μία πληθώρα επιστημονικών περιοχών. Μερικά παραδείγματα αποτελούν το πρόβλημα Partial Digest, που εμφανίζεται στην περιοχή της υπολογιστικής βιολογίας [Ciel03b, Ciel04], η εκχώρηση ατομικών αγαθών [Lipt04], η κατασκευή τουρνουά [Khan17], και μία παραλλαγή του Subset Sum, το Multiple Integrated Sets SSP, που βρίσκει εφαρμογές στο πεδίο της κρυπτογραφίας [Volo17]. Επιπλέον, σχετίζονται και με σημαντικές έννοιες της θεωρητικής επιστήμης των υπολογιστών. Για παράδειγμα, μία περιορισμένη εκδοχή του Equal Subset Sum ανήκει σε μία υποκλάση της κλάσης πολυπλοκότητας TFNP, συγκεκριμένα στην PPP [Papa94], μία κλάση που απαρτίζεται από προβλήματα αναζήτησης που πάντοτε έχουν λύση εξαιτίας κάποιου επιχειρήματος περιστερώνα, και κανένας πολυωνυμικός αλγόριθμος δεν είναι γνωστός για αυτήν την περιορισμένη εκδοχή.

Το πρόβλημα Equal Subset Sum έχει αποδειχθεί ότι είναι NP-hard από τους Woeginger και Yu [Woeg92] και αρκετές παραλλαγές του έχουν αποδειχθεί ότι είναι NP-hard από τους Cieliebak *et al.* στο [Ciel03a, Ciel08]. Έχει μελετηθεί κυρίως σε σχέση με την εκδοχή βελτιστοποίησής του, που αποκαλείται Subset Sum Ratio, η οποία ζητά την εύρεση δύο ξένων υποσυνόλων με όσο το δυνατόν μικρότερο λόγο αθροισμάτων. Ένας 1.324-προσεγγιστικός αλγόριθμος έχει προταθεί για το Subset Sum Ratio στο [Woeg92] και αρκετά FPTASs εμφανίστηκαν στα [Bazg02, Nano13, Meli18], με το γρηγορότερο έως τώρα να είναι αυτό που παρουσιάστηκε στο [Meli18] πολυπλοκότητας $\mathcal{O}(n^4/\varepsilon)$.

Σε ό,τι αφορά ακριβείς αλγορίθμους, πρόσφατα δείχθηκε ότι μπορεί να λυθεί πιθανοτικά σε χρόνο $\mathcal{O}^*(1.7088^n)$ [Much19], γρηγορότερα από μία στάνταρ 'meet-in-the-middle' προσέγγιση πολυπλοκότητας $\mathcal{O}^*(3^{n/2}) \leq \mathcal{O}^*(1.7321^n)$. Στην παρούσα εργασία, προτείνουμε τρεις αλγορίθμους ψευδοπολυωνυμικού χρόνου, επεκτείνοντας τα πρόσφατα αποτελέσματα για το Subset Sum από τους Koiliaris και Xu [Koil19] και Bringmann [Brin17] για το πρόβλημα Equal Subset Sum. Σημειώστε πως, ενώ η εφαρμογή αυτών των αλγορίθμων για την εκδοχή απόφασης του Equal Subset Sum είναι απλή, η προσαρμογή τους για την επίλυση της εκδοχής αναζήτησης, η οποία περιλαμβάνει την ανακατασκευή των συνόλων της λύσης, είναι αρκετά σύνθετη. Επιπλέον, παρουσιάζουμε έναν απλό και αποδοτικό αλγόριθμο που αποτελεί επέκταση του αλγορίθμου του Bellman [Bell57]. Προσέξτε όμως ότι οι τεχνικές που χρησιμοποιούνται στον τελευταίο αυτόν αλγόριθμο δεν μπορούν να εφαρμοστούν στο k-Subset Sum, κυρίως επειδή δεν μπορούμε να προσδιορίσουμε εάν τα εμπλεκόμενα υποσύνολα είναι ξένα δίχως να χρησιμοποιήσουμε κάποια συνθήκη ελαχιστότητας.

Το k-Subset Sum είναι ένα πολύ πιο γενικό πρόβλημα, που μπορεί να χρησιμοποιηθεί για την επίλυση διαφόρων παραλλαγών του Subset Sum, όπως εξηγούμε αναλυτικά στο αγγλικό κείμενο (Section 5.3). Εξ όσων γνωρίζουμε, δεν ήταν γνωστός κάποιος αλγόριθμος ψευδοπολυωνυμικού χρόνου αισθητά πιο γρήγορος από την κλασσική προσέγγιση δυναμικού προγραμματισμού πολυπλοκότητας $\mathcal{O}(nt^k)$ για το πρόβλημα k-Subset Sum πριν την παρούσα εργασία.

Αυτά τα προβλήματα είναι άρρηκτα συνδεδεμένα με το Subset Sum. Το τελευταίο, πρόσφατα παρουσίασε εντυπωσιακή πρόοδο, εξαιτίας των Koiliaris και Xu [Koil19] που παρουσίασαν έναν αλγόριθμο πολυπλοκότητας $\tilde{\mathcal{O}}(\sqrt{n}t)$, όπου n είναι το πλήθος των στοιχείων εισόδου και t είναι ο στόχος, και από τον Bringmann [Brin17] που παρουσίασε έναν πιθανοκρατικό αλγόριθμο πολυπλοκότητας $\tilde{\mathcal{O}}(n+t)$. Οι Jin και Wu πρότειναν έναν απλούστερο πιθανοκρατικό αλγόριθμο [Jin19] πετυχαίνοντας τα ίδια όρια με το [Brin17], που όμως δεν δείχνει να επεκτείνεται αποδοτικά ώστε να γίνεται ανακατασκευή των συνόλων της λύσης.[1] Πολύ πρόσφατα, οι Bringmann και Nakos [Brin20] παρουσίασαν έναν αλγόριθμο πολυπλοκότητας $\mathcal{O}(|\mathcal{S}_t(Z)|^{4/3}\text{poly}(\log t))$,

[1] Παρατηρήστε ότι ούτε ο αλγόριθμος του Bringmann [Brin17] λύνει την εκδοχή αναζήτησης του Subset Sum, όμως σε αυτή την εργασία θα δείξουμε πώς μπορούμε να επεκτείνουμε κατάλληλα τον αλγόριθμο ώστε να επιστρέφονται τα σύνολα της λύσης του Equal Subset Sum, χωρίς να αυξηθεί η πολυπλοκότητά του (αγνοώντας πολυλογαριθμικούς παράγοντες).

όπου $\mathcal{S}_t(Z)$ είναι το σύνολο όλων των πιθανών αθροισμάτων υποσυνόλων του συνόλου εισόδου Z που είναι μικρότερα του t, βασιζόμενοι στο top-k convolution.

Το πρόβλημα Multiple Subset Sum στην πραγματικότητα αποτελεί μία ειδική περίπτωση του προβλήματος Multiple Knapsack, όπου και τα δύο εκ των οποίων έχουν προσελκύσει ιδιαίτερο ενδιαφέρον. Σχετικά με το Multiple Subset Sum, οι Caprara *et al.* παρουσίασαν ένα PTAS για την περίπτωση όπου όλοι οι στόχοι είναι ίδιοι στο [Capr00], και στη συνέχεια στο [Capr03] παρουσιάζουν έναν 3/4 προσεγγιστικό αλγόριθμο. Το Multiple Knapsack έχει εξεταστεί εκτενέστερα τα τελευταία χρόνια, αφού εφαρμογές του συναντώνται σε διάφορα πεδία, όπως τα οικονομικά ή οι μετακινήσεις. Μερικές αξιοσημείωτες έρευνες πάνω σε παραλλαγές του προβλήματος παρέχονται από τους Lahyani *et al.* [Lahy19] και Dell'Amico *et al.* [Dell19]. Ειδικές περιπτώσεις και παραλλαγές του Multiple Subset Sum, όπως λόγου χάρη το πρόβλημα k-Subset Sum, έχουν μελετηθεί στα [Ciel03a, Ciel08], όπου προτάθηκαν απλοί ψευδοπολυωνυμικοί αλγόριθμοι.

Ερευνητικοί Στόχοι & Συνεισφορά

Αρχικά παρουσιάζουμε τέσσερις αλγορίθμους για το Equal Subset Sum: έναν τυχαιοκρατικό πολυπλοκότητας $\tilde{\mathcal{O}}(n+t)$, έναν ντετερμινιστικό πολυπλοκότητας $\tilde{\mathcal{O}}(\sigma)$, έναν ντετερμινιστικό πολυπλοκότητας $\tilde{\mathcal{O}}(\sqrt{n}t)$ και τέλος έναν ντετερμινιστικό πολυπλοκότητας $\tilde{\mathcal{O}}(n+t)$, όπου με σ συμβολίζουμε το συνολικό άθροισμα των στοιχείων του συνόλου εισόδου. Ενώ οι πρώτοι τρεις αλγόριθμοι βασίζονται σε τεχνικές που χρησιμοποιούνται σε αλγορίθμους για το πρόβλημα Subset Sum, και οι οποίοι μπορούν επιπλέον να επεκταθούν και για το πρόβλημα k-Subset Sum, ο τελευταίος μπορεί μόνο να χρησιμοποιηθεί για την αποτελεσματική επίλυση του Equal Subset Sum. Να σημειωθεί πως όλοι οι αλγόριθμοι λύνουν την *εκδοχή αναζήτησης* του Equal Subset Sum, δηλαδή κατασκευάζουν τα σύνολα της λύσης.

Επιπλέον, παρουσιάζουμε δύο αλγορίθμους που λύνουν την εκδοχή απόφασης του προβλήματος k-Subset Sum: έναν τυχαιοκρατικό πολυπλοκότητας $\tilde{\mathcal{O}}(n+t^k)$ και έναν ντετερμινιστικό πολυπλοκότητας $\tilde{\mathcal{O}}(n^{k/k+1}t^k)$. Στη συνέχεια, δείχνουμε πώς αυτές οι ιδέες μπορούν να επεκταθούν και να χρησιμοποιηθούν για την επίλυση των προβλημάτων Subset Sum Ratio, k-Subset Sum Ratio και Multiple Subset Sum.

Οι αλγόριθμοί μας βασίζονται και επεκτείνουν τους αλγορίθμους και τις τεχνικές που προτάθηκαν από τους Koiliaris και Xu [Koil19] και Bringmann [Brin17] για το Subset Sum. Πιο συγκεκριμένα, κάνουμε χρήση του αλγορίθμου Fast Fourier Transform, αριθμητικής modulo και της τεχνικής color-coding, ανάμεσα σε άλλα.

0.2 Προαπαιτούμενα

Σε αυτό το υποκεφάλαιο, θα παρουσιάσουμε συνοπτικά κάποια προαπαιτούμενα που χρειάζονται για την σαφή κατανόηση του ελληνικού κειμένου. Για μία εκτενέστερη παρουσίαση των εννοιών και των συμβολισμών της παρούσας εργασίας, παραπέμπουμε τον αναγνώστη στο κεφάλαιο 2 του αγγλικού κειμένου.

Συμβολισμός

Αρχικά θα παρουσιάσουμε τον συμβολισμό που χρησιμοποιείται σε αυτήν την ελληνική περίληψη. Σε μεγάλο βαθμό ακολουθούμε τον συμβολισμό που χρησιμοποιείται στα [Brin17] και [Koil19].

- $\tilde{\mathcal{O}}$ – Ευρέως χρησιμοποιούμενος συμβολισμός στο κομμάτι της υπολογιστικής πολυπλοκότητας, $f(n) \in \tilde{\mathcal{O}}(g(n))$ είναι συντομογραφία για το $\exists k : f(n) \in \mathcal{O}(g(n)\log^k g(n))$, δηλαδή χρησιμοποιείται για την απόκρυψη πολυλογαριθμικών παραγόντων.

- Με $[x \dots y] = \{x, x+1, \dots, y\}$ συμβολίζουμε το σύνολο των ακεραίων σε ένα διάστημα $[x, y]$. Ομοίως, $[x] = [0 \dots x]$.

- Δεδομένου ενός συνόλου $Z \subseteq \mathbb{N}$, συμβολίζουμε

  ▷ το άθροισμα των στοιχείων του με $\Sigma(Z) = \sum_{z \in Z} z$.

  ▷ το σύνολο όλων των δυνατών αθροισμάτων υποσυνόλων του Z με $\mathcal{S}(Z) = \{\Sigma(X) \mid X \subseteq Z\}$.

  ▷ το σύνολο όλων των δυνατών αθροισμάτων υποσυνόλων του Z μέχρι t με $\mathcal{S}_t(Z) = \mathcal{S}(Z) \cap [t]$.

  ▷ το σύνολο όλων των δυνατών αθροισμάτων υποσυνόλων του Z μαζί με την εκάστοτε πληθυκότητά τους με $\mathcal{SC}(Z) = \{(\Sigma(X), |X|) \mid X \subseteq Z\}$.

  ▷ το σύνολο όλων των δυνατών αθροισμάτων υποσυνόλων του Z μέχρι t μαζί με την εκάστοτε πληθυκότητά τους με $\mathcal{SC}_t(Z) = \mathcal{SC}(Z) \cap ([t] \times \mathbb{N})$.

- Δεδομένου δύο συνόλων $X, Y \subseteq \mathbb{N}$, συμβολίζουμε το σύνολο των δυνατών αθροισμάτων που προέρχονται από στοιχεία των δύο σύνολων ως $X \oplus Y = \{x + y \mid x \in X \cup \{0\}, y \in Y \cup \{0\}\}$. Επιπλέον, ορίζουμε $X \oplus_t Y = (X \oplus Y) \cap [t]$.

Θεωρητικό Υπόβαθρο

Σε αυτό το σημείο θα παρουσιάσουμε κάποιες βασικές θεωρητικές έννοιες που κρίνονται απαραίτητες για την κατανόηση της παρούσας εργασίας.

NP-completeness Στην θεωρία υπολογιστικής πολυπλοκότητας, η κλάση NP εμπεριέχει τα προβλήματα απόφασης για τα οποία, οι είσοδοι καταφατικής απάντησης έχουν αποδείξεις που μπορούν να πιστοποιηθούν σε πολυωνυμικό χρόνο από μία ντετερμινιστική μηχανή Turing. Με άλλα λόγια, δεδομένου ενός στιγμιότυπου εισόδου του προβλήματος, μπορούμε να πιστοποιήσουμε ότι η έξοδος είναι όντως καταφατική σε πολυωνυμικό χρόνο.

Εάν ένα πρόβλημα Π_1 ανάγεται σε ένα πρόβλημα Π_2 σε πολυωνυμικό χρόνο ($\Pi_1 \leq_P \Pi_2$), τότε:

- Υπάρχει μία συνάρτηση $R : \Sigma^* \to \Sigma^*$ που μπορεί να υπολογιστεί σε πολυωνυμικό χρόνο, τέτοια ώστε $\forall x \in \Sigma^*$, ισχύει ότι $x \in \Pi_1 \iff R(x) \in \Pi_2$, όπου με Σ^* αναπαριστούμε το σύνολο όλων των πιθανών εισόδων και $x \in \Pi$ αν και μόνο αν η είσοδος x έχει ως αποτέλεσμα καταφατική έξοδο για το πρόβλημα Π.

- Η συνάρτηση R καλείται *πολυωνυμική αναγωγή*.

- Ουσιαστικά, μετασχηματίζουμε (αποδοτικά) κάθε στιγμιότυπο εισόδου του προβλήματος Π_1 σε ένα στιγμιότυπο εισόδου του προβλήματος Π_2 και στη συνέχεια επιλύουμε το πρόβλημα Π_2.

Ένα πρόβλημα απόφασης Π είναι *NP-complete* εάν:

1. Το Π ανήκει στην κλάση NP, και

2. Κάθε πρόβλημα στην κλάση NP μπορεί να αναχθεί στο Π σε πολυωνυμικό χρόνο.

Τα NP-complete προβλήματα αναπαριστούν τα δυσκολότερα προβλήματα της κλάσης NP και συνοψίζουν την υπολογιστική δυσκολία της ίδιας της κλάσης.

Ψευδοπολυωνυμικοί Αλγόριθμοι Υποσύνολο των NP-complete αποτελούν τα weakly NP-complete προβλήματα. Ένα πρόβλημα αποκαλείται *weakly NP-complete* εάν υπάρχει αλγόριθμος που το επιλύει, του οποίου ο χρόνος εκτέλεσης είναι πολυωνυμικός ως προς την αριθμητική τιμή της εισόδου (ο μεγαλύτερος ακέραιος που υπάρχει στην είσοδο) αλλά όχι απαραίτητα στο μήκος της εισόδου (τον αριθμό των bits που χρειάζονται για να την αναπαραστήσουν), που συμβαίνει για τους αλγορίθμους πολυωνυμικού χρόνου. Ένας τέτοιος αλγόριθμος καλείται *ψευδοπολυωνυμικός*, αφού η αριθμητική τιμή της εισόδου είναι εκθετική ως προς το μήκος της εισόδου.

Τυχαιοκρατικοί Αλγόριθμοι Ένας αλγόριθμος αποκαλείται *τυχαιοκρατικός* εάν εμπεριέχει έναν βαθμό τυχαιότητας ως μέρος της λογικής του. Σε αντίθεση με έναν *ντετερμινιστικό* (ή αλλιώς *αιτιοκρατικό*) αλγόριθμο, η έξοδος ενός τυχαιοκρατικού μπορεί να διαφέρει ανάμεσα σε ξεχωριστές κλήσεις στο ίδιο στιγμιότυπο εισόδου. Επομένως, υπάρχει η πιθανότητα παραγωγής λανθασμένου αποτελέσματος όπου συνήθως συμβολίζεται με δ. Αυτοί οι αλγόριθμοι αποκαλούνται επίσης *πιθανοτικοί ή πιθανοκρατικοί* και η πολυπλοκότητά τους επηρεάζεται από τις επιτρεπτές τιμές της πιθανότητας λάθους δ.

Συνδυασμός Συνόλων Ένα κοινό χαρακτηριστικό των αλγορίθμων που εξετάζονται σε αυτήν την διπλωματική εργασία είναι η διαίρεση του αρχικού συνόλου εισόδου Z σε υποσύνολα $Z_1, \ldots, Z_k$ και ο αναδρομικός υπολογισμός των αθροισμάτων υποσυνόλων τους. Μετέπειτα, αυτά τα αθροίσματα υποσυνόλων συνδυάζονται με σκοπό τον υπολογισμό των αθροισμάτων υποσυνόλων του αρχικού συνόλου Z. Επομένως, είναι πολύ σημαντικό να πραγματοποιείται αποδοτικά αυτός ο συνδυασμός, αφού έχει κεντρικό ρόλο στον καθορισμό της πολυπλοκότητας των αλγορίθμων. Αυτό γίνεται με χρήση του αλγορίθμου FFT, η λογική και η χρησιμότητα του οποίου αναλύεται διεξοδικά στο αγγλικό κείμενο. Προσέξτε πως ο αλγόριθμος FFT μπορεί να επεκταθεί και σε σύνολα k-διάστατων σημείων, όπου μπορεί να πραγματοποιηθεί αποδοτικά το άθροισμα των τιμών των εκάστοτε διαστάσεων.

Κλάσεις Ισοτιμίας Δοθέντος ενός ακεραίου $n > 1$, δύο ακέραιοι a, b είναι *ισότιμοι modulo* n, αν το n είναι διαιρέτης της διαφοράς τους (με άλλα λόγια, υπάρχει ακέραιος $k \in \mathbb{Z}$ τέτοιος ώστε $a - b = kn$). Η ισοτιμία modulo n είναι σχέση ισοδυναμίας και συμβολίζεται με $a \equiv b \pmod{n}$.

Το σύνολο όλων των κλάσεων ισοτιμίας modulo n συμβολίζεται με $\mathbb{Z}_n = \{\overline{0}, \ldots, \overline{n-1}\}$. Καθένα από τα στοιχεία του $\overline{i}$ αναπαριστά το σύνολο των ακεραίων που είναι ισότιμοι modulo n με το i, με άλλα λόγια $\forall x \in \mathbb{Z}, x \in \overline{i} \iff x \equiv i \pmod{n}$.

Εύρεση Μαρτύρων μέσω Peeling Κάθε φορά που αναζητούμε τα δυνατά αθροίσματα που μπορούν να σχηματιστούν από τον συνδυασμό δύο συνόλων, είναι καίριας σημασίας ο αποδοτικός προσδιορισμός των επιμέρους αθροισμάτων που αθροίστηκαν για τον υπολογισμό κάθε αθροίσματος. Αυτό είναι απαραίτητο για την ανακατασκευή των συνόλων λύσης και επομένως της επίλυσης της *εκδοχής αναζήτησης* του Equal Subset Sum. Οι Koiliaris και Xu [Koil19] ανάγουν αυτό το πρόβλημα στο *πρόβλημα ανακατασκευής*, όπως αυτό αναφέρεται στο [Auma11] και επιχειρηματολογούν σχετικά με την πολυπλοκότητά του, καταλήγοντας στο συμπέρασμα ότι υπάρχει μόνο πολυλογαριθμική επιβάρυνση στον υπολογισμό των σχετιζόμενων μαρτύρων[2]. Αυτό είναι ένα πολύ σημαντικό αποτέλεσμα, που υποδεικνύει ότι ο υπολογισμός των μαρτύρων δεν προκαλεί κάποιο 'άλμα' στην χρονική πολυπλοκότητα.

[2] Ως *μάρτυρες* ενός αθροίσματος s, ορίζουμε τα επιμέρους αθροίσματα s' και $s - s'$, από τα οποία σχηματίστηκε το άθροισμα s.

0.3 Αλγόριθμοι για το πρόβλημα Subset Sum

Σε αυτό το υποκεφάλαιο, θα παρουσιάσουμε συνοπτικά τους αλγορίθμους που προτάθηκαν από τους Bringmann [Brin17] και Koiliaris και Xu [Koil19], τους οποίους θα τροποποιήσουμε και επεκτείνουμε στην συνέχεια αυτής της εργασίας. Κυρίως θα επικεντρωθούμε στην λογική και στην φιλοσοφία των αλγορίθμων και δεν θα παρουσιάσουμε τις τεχνικές τους λεπτομέρειες, οι οποίες παρουσιάζονται αναλυτικά στο κεφάλαιο 3.

Για το υπόλοιπο αυτού του κεφαλαίου, υποθέστε ότι $Z = \{z_1, \ldots, z_n\} \subseteq \mathbb{N}$ είναι το σύνολο εισόδου, όπου $n = |Z|$, και $t > 0$ είναι το δοθέν άνω φράγμα. Επομένως, ζητούμενο των παρακάτω αλγορίθμων είναι ο προσδιορισμός του εάν υπάρχει ένα υποσύνολο του Z, το άθροισμα των στοιχείων του οποίου ισούται με t. Για να το πετύχουν αυτό, οι παρακάτω αλγόριθμοι βρίσκουν στην πραγματικότητα το σύνολο $\mathcal{S}_t(Z)$ όλων των δυνατών αθροισμάτων που μπορούν να σχηματιστούν από υποσύνολα του συνόλου εισόδου Z και το άθροισμα των οποίων δεν ξεπερνά το t. Έπειτα, για να απαντήσουμε στο ερώτημά μας, αρκεί να εξετάσουμε εάν $t \in \mathcal{S}_t(Z)$.

Τυχαιοκρατικός αλγόριθμος πολυπλοκότητας $\tilde{\mathcal{O}}(n+t)$

Ο Bringmann παρουσίασε στο [Brin17] τον πρώτο ψευδογραμμικό ως προς t και n αλγόριθμο για το πρόβλημα Subset Sum. Τον τυχαιοκρατικό αυτό αλγόριθμο περιγράφουμε αναλυτικά στο υποκεφάλαιο 3.1. Στη σύντομη αυτή παρουσίαση θα περιγράψουμε διαισθητικά και χωρίς αποδείξεις τα βήματα του αλγορίθμου. Ο πιθανοτικός αυτός αλγόριθμος, έχει πιθανότητα σφάλματος δ.

1. Αρχικά χωρίζουμε το σύνολο εισόδου Z σε $\log n$ υποσύνολα Z_i, έτσι ώστε κάθε προκύπτον υποσύνολο να αποτελεί ένα *l-layer* σύνολο[3].

2. Κάθε τέτοιο *l-layer* υποσύνολο Z_i το χωρίζουμε σε m τυχαία υποσύνολα $S_{i1}, \ldots, S_{im}$, όπου το m επιλέγεται κατάλληλα.

3. Για κάθε προκύπτον υποσύνολο S_{ij}, τρέχουμε έναν αλγόριθμο Subset Sum που βρίσκει, με μεγάλη πιθανότητα, τα δυνατά αθροίσματα μέχρι μία τιμή $2\gamma t/l$ που μπορούν να σχηματιστούν από τα υποσύνολα του S_{ij} που έχουν πληθυκότητα το πολύ γ, όπου το γ ορίζεται κατάλληλα και το l είναι τιμή σχετιζόμενη με το εκάστοτε *l-layer* υπερσύνολο Z_i από τη διάσπαση του οποίου προέκυψε το S_{ij}.

4. Συνδυάζουμε κατάλληλα αυτά τα πιθανά δυνατά αθροίσματα των S_{ij} για να βρούμε τα πιθανά αθροίσματα υποσυνόλων του εκάστοτε Z_i.

5. Τέλος, συνδυάζουμε τα πιθανά αθροίσματα υποσυνόλων των διαφορετικών Z_i για να λάβουμε τα πιθανά αθροίσματα υποσυνόλων του αρχικού συνόλου εισόδου Z.

Οι τιμές $2\gamma t/l, m$ και γ επιλέγονται κατάλληλα ώστε η πιθανότητα σφάλματος του αλγορίθμου να είναι το πολύ δ και η επιλογή τους αιτιολογείται στο αγγλικό κείμενο. Οι συνδυασμοί των πιθανών αθροισμάτων υποσυνόλων γίνονται με κατάλληλες κλήσεις του αλγορίθμου για το FFT, ενώ το κόστος των κλήσεων αυτών υπολογίζεται αναλυτικά με σκοπό τον ακριβή προσδιορισμό της συνολικής πολυπλοκότητας του αλγορίθμου. Η τελική ποπλυπλοκότητα του αλγορίθμου είναι $\mathcal{O}(t \log t \log^3(n/\delta) \log n) = \tilde{\mathcal{O}}(n+t)$.

Ντετερμινιστικός αλγόριθμος πολυπλοκότητας $\tilde{\mathcal{O}}(\sigma)$

Ο αλγόριθμος αυτός, σε αντίθεση με τους άλλους αλγορίθμους που παρουσιάζονται, δεν έχει κάποιο άνω φράγμα t ως μέρος της εισόδου. Επομένως, το αποτέλεσμα του αλγορίθμου είναι το σύνολο $\mathcal{S}(Z)$.

[3] Ένας ορισμός των l-layer συνόλων αλλά και οι χρήσιμες ιδιότητές τους δίνονται στην αναλυτική παρουσίαση του αλγορίθμου στο αγγλικό κείμενο.

Ουσιαστικά, πρόκειται για τον απλό διαίρει και βασίλευε αλγόριθμο, που επαναληπτικά σπάει το σύνολο εισόδου Z σε δύο ισοπληθή υποσύνολα Z_1, Z_2, υπολογίζει αναδρομικά τα εκάστοτε δυνατά αθροίσματα που προκύπτουν από αυτά $\mathcal{S}(Z_1), \mathcal{S}(Z_2)$, και στη συνέχεια τα συνδυάζει, λαμβάνοντας το σύνολο $\mathcal{S}(Z) = \mathcal{S}(Z_1) \oplus \mathcal{S}(Z_2)$. Με μία προσεκτική ανάλυση του δέντρου αναδρομής, προκύπτει ότι η πολυπλοκότητα του αλγορίθμου είναι $\mathcal{O}(\sigma \log n) = \tilde{\mathcal{O}}(\sigma)$, όπου με σ συμβολίζουμε το άθροισμα των στοιχείων του συνόλου εισόδου.

Ντετερμινιστικός αλγόριθμος πολυπλοκότητας $\tilde{\mathcal{O}}(\sqrt{nt})$

Οι Koiliaris και Xu στο [Koil19] αναπτύσσουν έναν απλό, αποδοτικό αλγόριθμο για το πρόβλημα Subset Sum, που εκμεταλλεύεται τις ιδιότητες των αριθμών που ανήκουν στην ίδια κλάση ισοτιμίας. Πιο συγκεκριμένα, ο αλγόριθμος αποτελείται από μερικά απλά βήματα:

1. Αρχικά, χωρίζουμε τα στοιχεία του συνόλου εισόδου $Z = \{z_1, \ldots, z_n\}$ βάσει του υπολοίπου τους modulo b. Επομένως, προκύπτουν τα υποσύνολα Z_i, όπου $i \in \{0, \ldots, b-1\}$, καθένα εκ των οποίων περιέχει τα στοιχεία που έχουν υπόλοιπο i από την διαίρεση με b.

2. Για κάθε προκύπτον υποσύνολο Z_i, δημιουργούμε το σύνολο $Q_i = \{z \text{ div } b \mid z \in Z_i\}$ που περιέχει τα πηλίκα των στοιχείων του εκάστοτε συνόλου Z_i με το b.

3. Για κάθε σύνολο Q_i, υπολογίζουμε το σύνολο $\mathcal{SC}_{t/b}(Q_i)$ που περιέχει όλα τα δυνατά αθροίσματα μέχρι t/b που μπορούν να σχηματιστούν από υποσύνολα του Q_i μαζί με την πληθυκότητα του εκάστοτε υποσυνόλου. Παρατηρήστε ότι $\mathcal{SC}_{t/b}(Q_i) \subseteq \mathcal{S}_{t/b}(Q_i) \times [n]$ και ότι εάν ένα άθροισμα σχηματίζεται από δύο υποσύνολα διαφορετικής πληθυκότητας, τότε και οι δύο τρόποι σχηματισμού θα περιλαμβάνονται στο $\mathcal{SC}_{t/b}(Q_i)$.

4. Από κάθε σύνολο $\mathcal{SC}_{t/b}(Q_i)$, εξάγουμε το σύνολο $\mathcal{S}_t(Z_i)$, βασιζόμενοι στην παρακάτω σχέση, όπου X είναι υποσύνολο του Z_i πληθυκότητας j και αθροίσματος s.

$$s \in \mathcal{S}_t(Z_i) \iff \exists X \subseteq Z_i : \Sigma(X) = s \text{ και } \sum_{x \in X} x = \sum_{k=1}^{j} (y_k b + l) = \left(\sum_{k=1}^{j} y_k \right) b + ji.$$

   Από τον υπολογισμό των $\mathcal{SC}_{t/b}(Q_i)$ στο προηγούμενο βήμα, μας είναι γνωστοί όλοι οι έγκυροι συνδυασμοί $(\sum(y_k), j)$, επομένως μπορούμε να εξάγουμε επιτυχώς τα $\mathcal{S}_t(Z_i)$.

5. Τέλος, αρκεί να υπολογίσουμε το $\mathcal{S}_t(Z) = \mathcal{S}_t(Z_1) \oplus_t \ldots \oplus_t \mathcal{S}_t(Z_{b-1})$.

Ο αποδοτικός υπολογισμός των $\mathcal{SC}_{t/b}(Q_i)$ καθώς και οι τεχνικές λεπτομέρειες παρουσιάζονται και αιτιολογούνται επαρκώς στο υποκεφάλαιο 3.3. Σε αυτήν την συνοπτική περιγραφή, θα αρκεστούμε στο να αναφέρουμε ότι κάθε υπολογισμός $\mathcal{S}_t(Z_i)$ τελικά κοστίζει $\mathcal{O}\left((t/b) n_i \log n_i \log t\right)$, όπου $n_i = |Z_i|$. Επειδή $n_0 + \ldots + n_{b-1} = n$, έχουμε ότι το κόστος όλων των υπολογισμών $\mathcal{S}_t(Z_i)$ είναι

$$\sum_{l \in [b-1]} \mathcal{O}((t/b) n_l \log n_l \log t) = \mathcal{O}((t/b) n \log n \log t).$$

Ο συνδυασμός των $\mathcal{S}_t(Z_i)$ στο τελευταίο βήμα κοστίζει χρόνο $\mathcal{O}(bt \log t)$. Επομένως, η συνολική πολυπλοκότητα του αλγορίθμου θα είναι

$$\mathcal{O}\left(t \log t \left(\frac{n \log n}{b} + b \right) \right) = \mathcal{O}(\sqrt{n \log n} t \log t) = \tilde{\mathcal{O}}(\sqrt{nt}).$$

0.4 Αλγόριθμοι για το πρόβλημα Equal Subset Sum

Σε αυτό το υποκεφάλαιο, θα παρουσιάσουμε συνοπτικά τρεις τροποποιημένους αλγορίθμους που επιλύουν το πρόβλημα Equal Subset Sum, το οποίο μας ζητάει να κατασκευάσουμε, εάν υπάρχουν, δύο ξένα υποσύνολα του συνόλου εισόδου ίδιου αθροίσματος, όπου το άθροισμα αυτό είναι μικρότερο ή ίσο ενός δοθέντος ορίου t. Η σειρά με την οποία παρουσιάζονται οι τροποποιημένοι αλγόριθμοι ακολουθεί την σειρά με την οποία παρουσιάστηκαν οι αρχικοί αλγόριθμοι για το Subset Sum στο προηγούμενο υποκεφάλαιο, των οποίων αποτελούν επέκταση. Επιπροσθέτως, περιγράφουμε και έναν τέταρτο, απλό και αποδοτικό αλγόριθμο για το Equal Subset Sum, που παρότι υπερτερεί των υπολοίπων, δεν μπορεί να επεκταθεί σε πιο γενικά προβλήματα, όπως είναι το k-Subset Sum. Αποφεύγοντας τις τεχνικές λεπτομέρειες που παρουσιάζονται στο κεφάλαιο 4 του αγγλικού κειμένου, θα εστιάσουμε στις τροποποιήσεις που κάναμε και στην λογική των αλγορίθμων.

Τυχαιοκρατικός αλγόριθμος πολυπλοκότητας $\tilde{\mathcal{O}}(n+t)$

Με μερικές τροποποιήσεις μπορούμε να επεκτείνουμε τον αρχικό αλγόριθμο του Bringmann για το πρόβλημα Equal Subset Sum, χωρίς να παρατηρηθεί αισθητή αύξηση της πολυπλοκότητας. Κάνουμε τις εξής αλλαγές:

1. Εισάγουμε μία μεταβλητή c και δύο πίνακες t θέσεων W, F. Ο πίνακας W χρησιμοποιείται για να κρατάμε τον μάρτυρα του εκάστοτε αθροίσματος, ενώ η μεταβλητή c και ο πίνακας F εγγυώνται την συνέπεια του πίνακα W, αφού εξασφαλίζουν ότι κάθε άθροισμα ανά πάσα στιγμή της εκτέλεσης του αλγορίθμου σχηματίζεται με *μοναδικό* τρόπο.

2. Πλέον καθ' όλη τη διάρκεια εκτέλεσης του αλγορίθμου, κάθε φορά που καλούμε τον αλγόριθμο FFT, βρίσκουμε μαζί με το αποτέλεσμα και τους μάρτυρες όλων των προκύπτων αθροισμάτων. Εκτός αυτού, σε περίπτωση που είτε βρεθεί συντελεστής από το FFT μεγαλύτερος του 1, είτε αντιληφθούμε ότι ένα άθροισμα σχηματίζεται με παραπάνω του ενός τρόπους μέσω του πίνακα μαρτύρων W, τερματίζουμε κατασκευάζοντας τα σύνολα εξόδου αξιοποιώντας τον πίνακα W για το μικρότερο τέτοιο άθροισμα.

3. Τέλος, τροποποιούμε κατάλληλα τον αλγόριθμο που χρησιμοποιήθηκε για την εύρεση των αθροισμάτων υποσυνόλων για περιορισμένη πληθυκότητα των υποσυνόλων αυτών, ώστε με μεγάλη πιθανότητα να εντοπίζει τυχόν σύγκρουση και να επιλύει το Equal Subset Sum σε αυτή την περίπτωση.

Η πολυπλοκότητα του αλγορίθμου επηρεάζεται μόνο από την τελευταία τροποποίηση, που ασυμπτωτικά δεν αυξάνει την πολυπλοκότητα, και από το αυξημένο πλέον κόστος κλήσεων του FFT, εξαιτίας της εύρεσης των μαρτύρων των αθροισμάτων. Η τελική πολυπλοκότητα του αλγορίθμου παραμένει $\tilde{\mathcal{O}}(n+t)$, κατασκευάζοντας επιτυχώς τα υποσύνολα της λύσης.

Ντετερμινιστικός αλγόριθμος πολυπλοκότητας $\tilde{\mathcal{O}}(\sigma)$

Ο τροποποιημένος αυτός αλγόριθμος για το Equal Subset Sum αφορά την περίπτωση που δεν παρέχεται κάποιο άνω φράγμα t ως μέρος της εισόδου, αφού σε διαφορετική περίπτωση δεν είναι αρκετά αποδοτικός.

Όπως και στην περίπτωση του αλγορίθμου του Bringmann, εισάγουμε έναν πίνακα μαρτύρων W και αναλαμβάνουμε τον υπολογισμό τους κάθε φορά που καλούμε τον αλγόριθμο για το FFT. Επομένως, κάθε φορά που εκτελούμε έναν συνδυασμό συνόλων μέσω FFT, αρχικά υπολογίζουμε τους σχετικούς μάρτυρες, στη συνέχεια αναζητούμε εάν έχει προκύψει κάποιος συντελεστής μεγαλύτερος του 1, και τέλος ελέγχουμε εάν έχει προκύψει σύγκρουση μέσω του πίνακα W. Σε περίπτωση που σχηματίζεται κάποιο άθροισμα με δύο (ή και παραπάνω) διαφορετικούς τρόπους, σχηματίζουμε τα σύνολα εξόδου για το ελάχιστο τέτοιο άθροισμα αξιοποιώντας

τον πίνακα W και τους εμπλεκόμενους μάρτυρες. Η ανάλυση του δέντρου αναδρομής μας δείχνει ότι η πολυπλοκότητα του αλγορίθμου δεν αυξάνεται αισθητά, αφού ο τελικός αλγόριθμος έχει πολυπλοκότητα $\tilde{\mathcal{O}}(\sigma \log n) = \tilde{\mathcal{O}}(\sigma)$, όπου με σ συμβολίζουμε το άθροισμα των στοιχείων του συνόλου εισόδου. Η εγκυρότητα του αλγορίθμου πηγάζει από το γεγονός πως ανά πάσα στιγμή κατά τη διάρκεια εκτέλεσης του αλγορίθμου, ο πίνακας W εξασφαλίζει ότι οποιοδήποτε άθροισμα σχηματίζεται με μοναδικό τρόπο. Σε διαφορετική περίπτωση, η σύγκρουση εντοπίζεται μέσω του πίνακα και ακολουθεί η κατασκευή των υποσυνόλων της λύσης.

Ντετερμινιστικός αλγόριθμος πολυπλοκότητας $\tilde{\mathcal{O}}(\sqrt{n}t)$

Για τον αλγόριθμο αυτόν, οι τροποποιήσεις που θα χρειαστούν είναι ελάχιστες. Αρχικά, παρατηρούμε ότι εάν σε κάποια στιγμή εκτέλεσης του αλγορίθμου προκύψει σε κάποια πράξη FFT συντελεστής μεγαλύτερος του 1 για κάποιο άθροισμα, τότε για το ελάχιστο τέτοιο άθροισμα υπάρχουν δύο ξένα υποσύνολα που το σχηματίζουν. Επομένως, αρκεί να εντοπίσουμε ένα τέτοιο άθροισμα για το οποίο υπάρχουν ξένα υποσύνολα που το σχηματίζουν και στη συνέχεια να τρέξουμε δύο φορές έναν αλγόριθμο Subset Sum που τυπώνει το σύνολο εξόδου ώστε να σχηματίσουμε τα σύνολα εξόδου, όπου στην δεύτερη κλήση έχουμε αφαιρέσει τα στοιχεία του πρώτου συνόλου εξόδου από το σύνολο εισόδου. Η πολυπλοκότητα του αλγορίθμου παραμένει $\tilde{\mathcal{O}}(\sqrt{n}t)$, αφού η μόνη αύξηση της πολυπλοκότητας προκύπτει από το επιπλέον κόστος του FFT για τον υπολογισμό των μαρτύρων στις δύο κλήσεις του αλγορίθμου για το Subset Sum, που όμως είναι λογαριθμικής τάξης.

Ντετερμινιστικός αλγόριθμος πολυπλοκότητας $\tilde{\mathcal{O}}(n + t)$

Ο απλός αυτός αλγόριθμος αποτελεί μία αποδοτική επέκταση του κλασσικού αλγορίθμου του Bellman για το Subset Sum. Η ανάπτυξή του βασίστηκε σε σχόλια ανώνυμων κριτών και αποτελεί τον πιο αποδοτικό αλγόριθμο για το Equal Subset Sum, σε σχέση με τους υπόλοιπους που παρουσιάζονται στην παρούσα εργασία. Η κύρια διαφορά του όμως, είναι πως δεν μπορεί να προσαρμοστεί κατάλληλα ώστε να επιλύει το πολύ πιο γενικό (και δύσκολο) k-Subset Sum.

Σε αυτόν τον αλγόριθμο, η εύρεση πιθανών συγκρούσεων γίνεται αποκλειστικά μέσω ενός πίνακα μαρτύρων W, ο οποίος χρησιμοποιείται επίσης για την ανακατασκευή των συνόλων της λύσης. Η κύρια παρατήρηση στην οποία βασίζεται ο αλγόριθμος, είναι ότι θα προσπελάσουμε τον πίνακα αυτόν το πολύ t φορές, αφού σε διαφορετική περίπτωση θα υπάρξει σύγκρουση. Ο αλγόριθμος λειτουργεί επαναληπτικά, κρατώντας όλα τα δυνατά αθροίσματα που μπορούν να σχηματιστούν από τα πρώτα i στοιχεία σε μία δεντρική δομή. Σε κάθε επανάληψη, μπορούμε να βρούμε αποδοτικά τα νέα στοιχεία που προκύπτουν εξαιτίας του νέου στοιχείου και να τα εντάξουμε στο δέντρο δυνατών αθροισμάτων, εφόσον βέβαια δεν προκύψει κάποια σύγκρουση μέσω του πίνακα W. Έχοντας άνω όριο t στο μέγεθος του δέντρου, και χρησιμοποιώντας κατάλληλη δεντρική δομή, η τελική πολυπλοκότητα του αλγορίθμου είναι $\mathcal{O}(n + t \log t) = \tilde{\mathcal{O}}(n + t)$.

0.5 Το πρόβλημα k-Subset Sum και σχετικές επεκτάσεις

Οι αλγόριθμοι που προτάθηκαν από τους Bringmann [Brin17] και Koiliaris και Xu [Koil19] μπορούν να επεκταθούν για να λύσουν και το πρόβλημα k-Subset Sum[4]. Ακόμη, η ίδια λογική μπορεί να χρησιμοποιηθεί περαιτέρω για να επιλυθούν και μία πληθώρα προβλημάτων σχετιζόμενα με το Subset Sum. Στο παρόν υποκεφάλαιο θα παρουσιάσουμε συνοπτικά τον τρόπο με τον οποίο θα επεκταθούν οι αλγόριθμοι, καθώς και τα αποτελέσματα που δίνουν σε διάφορα σχετικά προβλήματα. Θα παραλείψουμε την λεπτομερή τεχνική ανάλυση, η οποία άλλωστε πραγματοποιείται στο κεφάλαιο 5 του αγγλικού κειμένου, και θα επικεντρωθούμε στην λογική της επέκτασης, καθώς και στο αντίκτυπο που αυτή έχει.

[4] Σημειώστε πως επεκτείνοντας τον αλγόριθμο του Bellman, μπορούμε επίσης να λύσουμε το πρόβλημα σε ντετερμινιστικό χρόνο $O(nt^k)$.

Αφετηρία μας είναι το πρόβλημα k-Subset Sum, το οποίο ζητά, δεδομένου ενός σύνολου εισόδου θετικών ακεραίων Z και k τιμών - στόχων t_i, όπου $i = 1, \ldots, k$, να βρεθεί εάν υπάρχουν k ξένα υποσύνολα $Z_1, \ldots, Z_k \subseteq Z$, τέτοια ώστε $\Sigma(Z_i) = t_i$ για $i = 1, \ldots, k$. Αυτό το πρόβλημα διαφέρει αισθητά από το Equal Subset Sum, αφού ακόμη και για την περίπτωση $k = 2$ και $t_1 = t_2$, δεν μπορούμε να συμπεράνουμε ότι η ύπαρξη υποσυνόλων που αθροίζουν στις ζητούμενες τιμές υπονοεί την ανεξαρτησία των συνόλων αυτών. Επομένως, είναι σημαντικό να επιβεβαιώνουμε καθ' όλη την διάρκεια του αλγορίθμου την ανεξαρτησία των εμπλεκόμενων συνόλων. Για το υπόλοιπο του υποκεφαλαίου, υποθέστε ότι $t = \max\{t_1, \ldots, t_k\}$. Επιπλέον, στην ασυμπτωτική μας ανάλυση υποθέτουμε ότι το πλήθος των συνόλων k δεν αποτελεί μέρος της εισόδου του προβλήματος και αντιμετωπίζεται ως σταθερά.

Επεκτείνοντας τα πολυώνυμα των προηγούμενων αλγορίθμων και εισάγοντας νέες μεταβλητές που χρησιμοποιούνται για την αποθήκευση αθροισμάτων υποσυνόλων ξένων μεταξύ τους, πλέον έχουμε να κάνουμε με πολυώνυμα της μορφής $\sum a_{s_1 \ldots s_k} x_1^{s_1} \ldots x_k^{s_k}$, όπου ο συντελεστής κάθε όρου αναπαριστά το πλήθος των διαφορετικών τρόπων που μπορούν να σχηματιστούν τα αθροίσματα $s_1, \ldots, s_k$ από ξένα υποσύνολα του συνόλου εισόδου. Αναπαριστώντας κατάλληλα κάθε στοιχείο του αρχικού συνόλου ώστε να μπορεί να προστεθεί το πολύ σε ένα από τα υποσύνολα που σχηματίζουν κάθε συνδυασμό k αθροισμάτων υποσυνόλων, προκύπτουν μόνο έγκυροι συνδυασμοί. Τελικό ζητούμενο είναι ο προσδιορισμός της τιμής $a_{t_1 \ldots t_k}$. Αν αυτή είναι 0, τότε ο αλγόριθμος τερματίζει αρνητικά, αφού αυτό σημαίνει πως δεν γίνεται να σχηματιστούν τα k ζητούμενα αθροίσματα από ανεξάρτητα υποσύνολα. Σε διαφορετική περίπτωση, ο αλγόριθμος τερματίζει καταφατικά.

Τυχαιοκρατικός αλγόριθμος πολυπλοκότητας $\tilde{\mathcal{O}}(n + t^k)$

Βασική ιδέα της τροποποίησης στην περίπτωση του αλγορίθμου του Bringmann [Brin17] είναι η επέκταση του χαρακτηριστικού πολυωνύμου και η αναπαράσταση κάθε στοιχείου $z \in Z$ από ένα πολυώνυμο $\sum_{i=1}^{k} x_i^z$ όπου ανταποκρίνεται στις k περιπτώσεις[5] του να προστίθεται το στοιχείο z στο σύνολο του οποίου το άθροισμα αναπαρίσταται από τον εκθέτη της μεταβλητής x_i. Εκτός αυτού και του σχετικού κόστους που επιφέρει αυτή λόγω των κλήσεων του FFT (οι οποίες πλέον αφορούν πολυώνυμα k μεταβλητών), τροποποιείται κατάλληλα, χωρίς ωστόσο να προκαλέσει αλλαγή στην πολυπλοκότητα, ο αλγόριθμος που υπολογίζει τα δυνατά αθροίσματα υποσυνόλων περιορισμένης πληθυκότητας, ώστε να υπολογίζεται με μεγάλη πιθανότητα κάθε δυνατός συνδυασμός αθροισμάτων ξένων υποσυνόλων. Ο τελικός αλγόριθμος έχει χρονική πολυπλοκότητα $\tilde{\mathcal{O}}(n + t^k)$.

Ντετερμινιστικός αλγόριθμος πολυπλοκότητας $\tilde{\mathcal{O}}(n^{k/k+1} t^k)$

Βασική ιδέα του αλγορίθμου παραμένει η επέκταση του αρχικού αλγορίθμου των Koiliaris και Xu [Koil19] που εκμεταλλεύεται τις ιδιότητες των κλάσεων ισοτιμίας, με χρήση επιπλέον μεταβλητών στο χαρακτηριστικό πολυώνυμο των συνόλων.

Αναπαριστώντας πλέον κάθε στοιχείο z ενός συνόλου με το πολυώνυμο $\sum_{i=1}^{k} x_i^z$ και επεκτείνοντας κατάλληλα τον αλγόριθμο του FFT με επιπλέον διαστάσεις για τις νέες μεταβλητές, προσδιορίζοντας παράλληλα και την αύξηση στο κόστος που επιφέρει αυτή η τροποποίηση, καταλήγουμε τελικά σε έναν ντετερμινιστικό αλγόριθμο χρονικής πολυπλοκότητας $\tilde{\mathcal{O}}(n^{k/k+1} t^k)$. Να σημειώσουμε επίσης ότι για τον προσδιορισμό των έγκυρων συνδυασμών αθροισμάτων υποσυνόλων για στοιχεία που ανήκουν στην ίδια κλάση ισοτιμίας, εισάγουμε επιπλέον μεταβλητές στα πολυώνυμα, οι οποίες χρησιμοποιούνται για την αποθήκευση της πληθυκότητας του εκάστοτε υποσυνόλου του συνδυασμού.

[5] Η επιπλέον περίπτωση που το στοιχείο z δεν προστίθεται σε κανένα από τα k πιθανά υποσύνολα καλύπτεται από τον ορισμό της πράξης $\oplus$.

Επέκταση σε παρεμφερή προβλήματα αθροισμάτων υποσυνόλων

Η έξοδος των δύο προαναφερθέντων αλγορίθμων είναι ένα πολυώνυμο $\sum a_{s_1 \dots s_k} x_1^{s_1} \dots x_k^{s_k}$ μεταβλητών $x_1, \dots, x_k$, όπου $s_i \leq t$, και $a_{s_1 \dots s_k} \neq 0$ ισοδυναμεί με δυνατότητα σχηματισμού των αθροισμάτων $s_1, \dots, s_k$ από ξένα υποσύνολα του συνόλου εισόδου. Για την επίλυση του Subset Sum Ratio, δεν έχουμε παρά να θέσουμε $k = 2$, να βρούμε όλα αυτά τα δυνατά ζευγάρια έγκυρων αθροισμάτων και στη συνέχεια να τα διατρέξουμε και να βρούμε το ζευγάρι εκθετών με κλάσμα όσο το δυνατόν πιο κοντά στο 1. Ανάλογα με το ποιος αλγόριθμος θα χρησιμοποιηθεί, η πολυπλοκότητα θα είναι είτε $\tilde{\mathcal{O}}(n^{2/3}t^2)$ στην περίπτωση του ντετερμινιστικού, είτε $\tilde{\mathcal{O}}(n + t^2)$ στην περίπτωση του τυχαιοκρατικού.

Χρησιμοποιώντας τους παραπάνω αλγορίθμους για το k-Subset Sum σαν black box, μπορούμε ακόμη να λύσουμε μία πληθώρα προβλημάτων που σχετίζονται με το Subset Sum. Αναζητώντας τον καλύτερο λόγο ανάμεσα στο μεγαλύτερο και στο μικρότερο σχηματιζόμενο άθροισμα (εντός των τεθέντων ορίων $t_1, \dots, t_k$), επιλύουμε το πρόβλημα k-Subset Sum Ratio καθώς και την ειδική περίπτωση που θέλουμε να διαμοιράσουμε τα στοιχεία της εισόδου σε k όσο το δυνατόν πιο ίσα υποσύνολα και ονομάζουμε k-Partition, ενώ αναζητώντας το μέγιστο άθροισμα των εκθετών των όρων του τελικού πολυωνύμου, λύνουμε το πρόβλημα Multiple Subset Sum.

0.6 Συμπεράσματα & Μελλοντικές Προεκτάσεις

Συμπεράσματα

Πρόσφατα έχει υπάρξει εντυπωσιακή πρόοδος σχετικά με το πρόβλημα Subset Sum, καθώς νέοι, πιο αποδοτικοί ψευδοπολυωνυμικοί αλγόριθμοι συνεχώς προτείνονται. Αυτή η πρόοδος μπορεί να επηρεάσει έντονα άλλα σχετιζόμενα προβλήματα. Στην παρούσα εργασία, εξετάζουμε κυρίως δύο σχετικά αλγοριθμικά προβλήματα, τα Equal Subset Sum και k-Subset Sum, καθώς και πώς βελτιώσεις στο Subset Sum μπορούν ενδεχομένως να τα επηρεάσουν.

Δείχνουμε ότι, στην περίπτωση του Equal Subset Sum, ενώ μπορούμε να επεκτείνουμε πρόσφατους αλγορίθμους για το Subset Sum, παράγοντας έτσι τρεις ψευδοπολυωνυμικούς αλγορίθμους, σε αυτή τη συγκεκριμένη περίπτωση υπάρχει στην πραγματικότητα μία απλούστερη και αποδοτικότερη προσέγγιση, που βασίζεται έντονα στους περιορισμούς και την φύση του προβλήματος.

Όσον αφορά το k-Subset Sum, παρουσιάσαμε δύο ψευδοπολυωνυμικούς αλγόριθμους που λύνουν αποδοτικά αυτό το πολύ πιο γενικό και δύσκολο πρόβλημα.

Μία πληθώρα προβλημάτων μπορούν να επηρεαστούν και να εκμεταλλευτούν αυτήν την πρόοδο, με αποτέλεσμα νέους, πιο αποδοτικούς αλγορίθμους για πολλά αλγοριθμικά προβλήματα, αναδεικνύοντας έτσι την στενή σχέση που έχουν με το Subset Sum, καθώς και το πώς βελτιώσεις αλγορίθμων αυτού ενδεχομένως να τα επηρεάσουν.

Μελλοντικές Προεκτάσεις

Αρχικά, σκοπεύουμε να διερευνήσουμε το κατά πόσο τα αποτελέσματά μας μπορούν ενδεχομένως να γενικευτούν σε μία κλάση προβλημάτων αθροίσματος υποσυνόλων, π.χ. χρησιμοποιώντας το πλαίσιο που αναπτύχθηκε στο [Meli20]. Επιπλέον, ενδιαφέρον παρουσιάζουν τα προσεγγιστικά σχήματα για τις εκδοχές μέτρησης προβλημάτων σακιδίου και αθροίσματος υποσυνόλων (δείτε για παράδειγμα τα [Gopa11, Stef12, Trio20] αλλά και τα [Gawr18, Rizz19]). Μερικά από αυτά τα προβλήματα ανήκουν σε κλάσεις κάτω από την #P, όπως την κλάση TotP [Pago06], γεγονός που ενδεχομένως να μπορεί να χρησιμοποιηθεί για την απόδειξη αποτελεσμάτων σχετικά με την δυνατότητα προσεγγισημότητάς τους [Baka20].

Οι Jin και Wu πρότειναν έναν αποδοτικό $\tilde{\mathcal{O}}(n+t)$ τυχαιοκρατικό αλγόριθμο για την επίλυση του Subset Sum στο [Jin19]. Ο αλγόριθμος αυτός είναι πολύ απλούστερος και πιο αποδοτικός από αυτόν που προτάθηκε από τον Bringmann. Είναι ενδιαφέρον να εξεταστεί το κατά πόσο

μπορεί να επεκταθεί αυτός ο αλγόριθμος για το k-Subset Sum, όπως συνέβη στην περίπτωση του αλγορίθμου του Bringmann, αφού αυτό θα είχε ως αποτέλεσμα μία απλούστερη και ενδεχομένως πιο αποδοτική εναλλακτική προσέγγιση.

Ο ντετερμινιστικός αλγόριθμος για το k-Subset Sum περιλαμβάνει τον υπολογισμό των πιθανών αθροισμάτων υποσυνόλων μαζί με τις αντίστοιχες πληθυκότητες των υποσυνόλων αυτών, για στοιχεία που ανήκουν στην ίδια κλάση ισοτιμίας. Για να το πετύχουμε αυτό, επεκτείνουμε τις κλήσεις FFT σε πολλές μεταβλητές, όπου η κάθε μία αναπαριστά είτε ένα πιθανό άθροισμα υποσυνόλου, είτε την πληθυκότητά του. Επομένως, για k υποσύνολα και n στοιχεία, πραγματοποιούμε κλήσεις FFT σε πολυώνυμα με μεταβλητές $x_1, \ldots, x_k, c_1, \ldots, c_k$, όπου οι εκθέτες των x_i ανήκουν στο $[t/b]$ για κάποιο δοθέν άνω φράγμα t, ενώ οι εκθέτες των c_i στο $[n]$. Παρατηρήστε όμως πως κάθε στοιχείο του συνόλου μας θα χρησιμοποιηθεί μόνο σε ένα υποσύνολο, επομένως για έναν όρο $x_1^{s_1} \ldots x_k^{s_k} c_1^{n_1} \ldots c_k^{n_k}$ του παραγόμενου πολυωνύμου, ισχύει ότι $s_i \leq t/b$ και $\sum n_i \leq n$. Αυτό διαφέρει αισθητά από την ανάλυση που πραγματοποιούμε, όπου στην ουσία υποθέτουμε ότι $n_i \leq n$, που είναι σημαντικά λιγότερο αυστηρό. Μία αυστηρότερη ασυμπτωτική ανάλυση ενδεχομένως να είναι εφικτή σε αυτές τις κλήσεις FFT, με πιθανό αποτέλεσμα την βελτίωση της συνολικής πολυπλοκότητας του αλγορίθμου.

Οι αλγόριθμοι που παρουσιάζονται στην παρούσα διπλωματική εργασία επιλύουν την *εκδοχή απόφασης* του προβλήματος k-Subset Sum. Με άλλα λόγια, η έξοδός τους είναι μία καταφατική ή αρνητική απάντηση, ανάλογα με το εάν είναι δυνατόν να υπάρξουν k ξένα υποσύνολα του συνόλου εισόδου, τα αθροίσματα των οποίων ισούνται με κάποιες δοθείσες τιμές $t_1, \ldots, t_k$ αντίστοιχα. Μία επέκταση αυτών των αλγορίθμων θα μπορούσε να συμπεριλαμβάνει την ανακατασκευή των k υποσυνόλων της λύσης. Οι Koiliaris και Xu [Koil19] επιχειρηματολογούν σχετικά με το ότι γίνεται να ανακατασκευαστεί το σύνολο της λύσης του Subset Sum με μόνο πολυλογαριθμικό επιπλέον κόστος. Αυτό είναι εφικτό εξάγοντας προσεκτικά τους *μάρτυρες* κάθε αθροίσματος κάθε φορά που πραγματοποιείται μία κλήση FFT. Αυτοί οι μάρτυρες είναι στην πραγματικότητα τα μερικά αθροίσματα που χρησιμοποιούνται για τον υπολογισμό του νέου αθροίσματος. Έτσι, ανάγοντας αυτό το πρόβλημα στο *πρόβλημα ανακατασκευής* όπως αναφέρεται στο [Auma11], καταλήγουν ότι είναι δυνατόν να υπολογιστούν όλοι οι μάρτυρες μίας κλήσης FFT χωρίς ιδιαίτερη αύξηση της πολυπλοκότητας. Αυτό συμβαίνει για κλήσεις FFT σε πολυώνυμα μίας μεταβλητής, οπότε ενδεχομένως να μπορούν να χρησιμοποιηθούν ανάλογα επιχειρήματα και για την περίπτωση πολλών μεταβλητών.

Μία τελείως διαφορετική προσέγγιση μπορεί να βασίζεται στην χωρική πολυπλοκότητα. Οι Lokshtanov και Nederlof πρώτοι πρότειναν αλγόριθμους πολυωνυμικού χώρου για διάφορα NP-Complete προβλήματα, ανάμεσά τους και το Subset Sum, στο [Loks10]. Ο Bringmann αργότερα βελτίωσε τα προηγούμενα όρια στο [Brin17]. Πολύ πρόσφατα, οι Jin *et al.* [Jin21] έκαναν πρόοδο σε αυτή την κατεύθυνση. Αυτοί οι αλγόριθμοι είναι πιθανό να μπορούν να προσαρμοστούν για το πρόβλημα Equal Subset Sum και να παραχθεί ένας αλγόριθμος ψευδοπολυωνυμικού χρόνου και *πολυωνυμικού* χώρου.

Τέλος, ενδιαφέρον παρουσιάζει η σχέση ανάμεσα στα προβλήματα Orthogonal Vector και Equal Subset Sum. Έστω ένα σύνολο εισόδου $Z = \{z_1, \ldots, z_n\}$. Ορίζουμε το *inclusion vector* ενός υποσυνόλου $V \subseteq Z$ ως το διάνυσμα n θέσεων $v = (v_1, \ldots, v_n)$, όπου $v_i = 1$ αν και μόνο αν $z_i \in V$, διαφορετικά $v_i = 0$. Έπεται ότι αν δύο υποσύνολα A, B είναι ξένα μεταξύ τους, τότε τα inclusion vectors τους είναι ορθογώνια, αφού είτε $a_i = 0$ είτε $b_i = 0$, για κάθε $i \in [n]$. Επίσης, δοθέντος ενός πίνακα συντελεστών $C \in \{0,1\}^{m \times n}$, που αναπαριστά m διανύσματα n θέσεων (καθένα εκ των οποίων αντιστοιχεί σε ένα πιθανό inclusion vector), ένα άθροισμα $s \in \mathbb{N}$, που αναπαριστά ένα πιθανό άθροισμα για το οποίο υπάρχουν πολλά ξένα υποσύνολα που αθροίζουν σε αυτό, και έναν πίνακα $B \in \{s\}^{m \times 1}$, λύνοντας την εξίσωση $C \cdot X = B$ και περιορίζοντας τον χώρο των λύσεων $X \in \mathbb{N}^{n \times 1}$ σε θετικούς ακεραίους που όλοι διαφέρουν μεταξύ τους (δηλαδή $x_i \neq x_j$ για κάθε i, j), μπορεί κανείς να κατασκευάσει ένα στιγμιότυπο του προβλήματος Equal Subset Sum, δοθέντος ενός τέτοιου (περιορισμένου) στιγμιότυπου του προβλήματος Orthogonal Vector. Η εξίσωση μπορεί να λυθεί χρησιμοποιώντας *ακέραιο*

προγραμματισμό και, επειδή ο πίνακας C είναι *totally unimodular*, η πολυπλοκότητα της λύσης μπορεί να είναι μικρότερη απ' όσο θα περίμενε κανείς. Αυτές οι παρατηρήσεις μας οδηγούν στο να πιστεύουμε πως μία αναγωγή Orthogonal Vector $\leq$ Equal Subset Sum μπορεί να είναι εφικτή, αρχικά περιορίζοντας ένα τυχαίο στιγμιότυπο του Orthogonal Vector σε ένα που ικανοποιεί τους περιορισμούς που περιγράψαμε προηγουμένως και στη συνέχεια παράγοντας ένα στιγμιότυπο του Equal Subset Sum από αυτό.

Chapter 1

Introduction

Given a set S of positive integers and a target t, determine whether there exists a subset $Z \subseteq S$, such that the sum of its elements equals t. One of Karp's original NP-complete problems [Karp72], Subset Sum is considered one of the most fundamental algorithmic problems, along with Knapsack, SAT and more. Many university undergraduate classes introduce the very concept of pseudopolynomial complexity via the classical algorithm of Bellman, as presented in [Bell57]. Despite being an extensively studied algorithmic problem, there have been many important improvements in the past few years with respect to its pseudopolynomial time solvability. New deterministic [Koil19] and randomised [Brin17, Jin19] algorithms have been introduced, representing the first substantial improvements over the long-standing standard approach of Bellman [Bell57] and the improvement by Pisinger [Pisi99].

Equal Subset Sum is a less studied, nevertheless noteworthy problem which has attracted the attention of several researchers, as it finds interesting applications in computational biology [Ciel03b, Ciel04], computational social choice [Lipt04] and cryptography [Volo17], to name a few. Moreover, it is related to important theoretical concepts such as the complexity of search problems in the class TFNP [Papa94]. In this thesis, we consider the case where an upper bound t is also given and we seek for two subsets of equal sum that is bounded by t.

A more general version of the problem is the one that asks for k disjoint subsets the sums of which are respectively equal to targets $t_i, i = 1, \ldots, k$, henceforth referred to as the k-Subset Sum problem. One can see that even in the case of $k = 2$ and $t_1 = t_2$, the problem becomes much harder; this can be seen as a targeted version of Equal Subset Sum, for which the disjointness of minimal sets property does not apply. An interesting special case is when the sum of targets equals the sum of the elements of the input set, that is, we ask for a partition of the input set to subsets of particular sums; an even more special case is the one in which we want to partition the input set to a number of subsets of equal sum. The latter can find applications in fair allocation situations.

1.1 Related Work

Equal Subset Sum and its optimisation version called Subset Sum Ratio [Bazg02] are well studied problems, closely related to problems appearing in many scientific areas. Some examples are the Partial Digest problem, which comes from computational biology [Ciel03b, Ciel04], the allocation of individual goods [Lipt04], tournament construction [Khan17], and a variation of Subset Sum, namely the Multiple Integrated Sets SSP, which finds applications in the field of cryptography [Volo17]. Moreover, it is related to important concepts in theoretical computer science; for example, a restricted version of Equal Subset Sum lies in a subclass of the complexity class TFNP, namely in PPP [Papa94], a class consisting of search problems that always have a solution due to some pigeonhole argument, and no polynomial time algorithm is known for this restricted version.

Equal Subset Sum has been proven to be NP-hard by Woeginger and Yu in [Woeg92]. Several variations have also been proven NP-hard by Cieliebak *et al.* in [Ciel03a, Ciel08].

It has been mostly studied with respect to its optimisation version, also referred to as Subset Sum Ratio, that is, to find two disjoint subsets with minimal ratio of sums. A 1.324-approximation algorithm has been proposed for Subset Sum Ratio in [Woeg92] and several FPTASs appeared in [Bazg02, Nano13, Meli18], the fastest so far being the one in [Meli18] of complexity $\mathcal{O}(n^4/\varepsilon)$.

Regarding exact algorithms, recent progress has shown that it can be solved probabilistically in $\mathcal{O}^*(1.7088^n)$ time [Much19], faster than a standard 'meet-in-the-middle' approach yielding a $\mathcal{O}^*(3^{n/2}) \leq \mathcal{O}^*(1.7321^n)$ time complexity. In this thesis, we propose three pseudopolynomial time algorithms by extending the below mentioned recent results for Subset Sum by Koiliaris and Xu [Koil19] and Bringmann [Brin17] to Equal Subset Sum. Note that, while the application of those algorithms to the decision version of the problem is rather trivial, their adaptation for solving the search version, which includes the reconstruction of the solution sets, is quite involved. Additionally, we present a simple and efficient extension of Bellman's [Bell57] algorithm. Notice however that the techniques involved in the last algorithm do not seem to apply to k-Subset Sum, mainly because we cannot assume the minimality of the involved subsets.

k-Subset Sum is a much more general problem, which can be used to solve multiple variations of Subset Sum, as it is thoroughly explained in Section 5.3. To the best of our knowledge, no pseudopolynomial time algorithm substantially faster than the standard $\mathcal{O}(nt^k)$ dynamic programming approach was known for k-Subset Sum prior to this work.

These problems are tightly connected to Subset Sum, which has seen impressive advances recently, due to Koiliaris and Xu [Koil19], who introduced a $\tilde{\mathcal{O}}(\sqrt{n}t)$ algorithm, where n is the number of input elements and t is the target, and Bringmann [Brin17], who introduced a $\tilde{\mathcal{O}}(n+t)$ randomised algorithm. Jin and Wu proposed a simpler randomised algorithm [Jin19] achieving the same bounds as [Brin17], which however does not seem to provide any intuition on how to reconstruct the solution set.[1] In a very recent work, Bringmann and Nakos [Brin20] have presented an $\mathcal{O}(|\mathcal{S}_t(Z)|^{4/3}\text{poly}(\log t))$ algorithm, where $\mathcal{S}_t(Z)$ is the set of all subset sums of the input set Z that are smaller than t, based on top-k convolution.

Multiple Subset Sum is a special case of the Multiple Knapsack problem, both of which have attracted considerable attention. Regarding Multiple Subset Sum, Caprara *et al.* present a PTAS for the case where all targets are the same [Capr00], and subsequently in [Capr03] they introduce a 3/4 approximation algorithm. The Multiple Knapsack problem has been more intensively studied in recent years as applications for it arise naturally (in fields such as transportation, industry, and finance, to name a few). Some notable studies on variations of the problem are given by Lahyani *et al.* [Lahy19] and Dell'Amico *et al.* [Dell19]. Special cases and variants of Multiple Subset Sum, such as the k-Subset Sum problem, have been studied in [Ciel03a, Ciel08] where simple pseudopolynomial algorithms were proposed.

1.2 Research Objectives & Contribution

We first present four algorithms for Equal Subset Sum: a randomised one of complexity $\tilde{\mathcal{O}}(n+t)$, a deterministic one of complexity $\tilde{\mathcal{O}}(\sqrt{n}t)$, a deterministic $\tilde{\mathcal{O}}(\sigma)$ algorithm and lastly a deterministic $\tilde{\mathcal{O}}(n+t)$ one, where σ is the total sum of the elements of the input set. While the first three algorithms are based on techniques used on Subset Sum algorithms, which can be additionally extended to the k-Subset Sum problem, the latter is not, thus can only be used to solve efficiently Equal Subset Sum. Note that all of the presented algorithms solve the *search version* of Equal Subset Sum, that is, with reconstruction of the solution

[1] Notice that neither Bringmann's algorithm [Brin17] solves the search version of Subset Sum, however we will show in this thesis that it can be appropriately strengthened in order to also provide the solution sets of Equal Subset Sum, retaining the same time complexity (disregarding polylogarithmic factors).

sets.

In addition, we present two algorithms to solve the decision version of k-Subset Sum: a randomised one of complexity $\tilde{\mathcal{O}}(n + t^k)$ and a deterministic one of complexity $\tilde{\mathcal{O}}(n^{k/k+1}t^k)$. We subsequently show how these ideas can be extended to solve the decision versions of Subset Sum Ratio, k-Subset Sum Ratio and Multiple Subset Sum.

Our algorithms extend and build upon the algorithms and techniques proposed by Koiliaris and Xu [Koil19] and Bringmann [Brin17] for Subset Sum. In particular, we make use of FFT computations, modular arithmetic and color-coding, among others.

1.3 Thesis Outline

The thesis is structured as follows:

- ▷ We start by presenting some necessary theoretical background in Chapter 2. Here, we provide some necessary lemmas used throughout the rest of the thesis. We also explain the notation used as well as some of the used techniques.
- ▷ Afterwards, in Chapter 3 we present and thoroughly analyse the original algorithms for Subset Sum, as introduced by Bringmann [Brin17] and Koiliaris and Xu [Koil19]. These are the algorithms we extend in order to efficiently solve Equal Subset Sum and k-Subset Sum.
- ▷ Next, in Chapter 4 we introduce and analyse the modified algorithms that efficiently solve the *search version* of Equal Subset Sum, thus successfully returning the solution sets. Additionally, we present a more efficient deterministic algorithm that solves Equal Subset Sum, which is not based on those presented on Chapter 3.
- ▷ Subsequently, in Chapter 5 we present the modified algorithms that successfully solve k-Subset Sum, along with some intuition regarding the extension of these algorithms, in order to cope with multiple variations of the Subset Sum problem.
- ▷ Finally, in Chapter 6 we summarise our results, as well as provide some intuition regarding possible extensions and research areas.

Chapter 2

Preliminaries

We will firstly present and explain the semantics used throughout the diploma thesis, some theoretical background, as well as some necessary lemmas and techniques used at the following algorithms.

2.1 Notation

We largely follow the notation used at [Brin17] and [Koil19].

- $\tilde{\mathcal{O}}$ – Standard complexity notation, used to ignore polylogarithmic factors. Thus, it holds that $f(n) \in \tilde{\mathcal{O}}(g(n))$ is shorthand for $\exists k : f(n) \in \mathcal{O}(g(n) \log^k g(n))$.
- Let $[x \dots y] = \{x, x+1, \dots, y\}$ denote the set of integers in the interval $[x, y]$. Similarly, $[x] = [0 \dots x]$.
- Given a set $Z \subseteq \mathbb{N}$, we denote
  - ▷ the sum of its elements by $\Sigma(Z) = \sum_{z \in Z} z$.
  - ▷ the *set of all subset sums of* Z by $\mathcal{S}(Z) = \{\Sigma(X) \mid X \subseteq Z\}$.
  - ▷ the *set of all subset sums of* Z *up to* t by $\mathcal{S}_t(Z) = \mathcal{S}(Z) \cap [t]$.
  - ▷ the *set of all subset sums of* Z *along with their respective cardinality* by $\mathcal{SC}(Z) = \{(\Sigma(X), |X|) \mid X \subseteq Z\}$.
  - ▷ the *set of all subset sums of* Z *up to* t *along with their respective cardinality* by $\mathcal{SC}_t(Z) = \mathcal{SC}(Z) \cap ([t] \times \mathbb{N})$.
  - ▷ the *characteristic polynomial of* Z by $f_Z(x) = \sum_{z \in Z} x^z$.
  - ▷ the *k-modified characteristic polynomial of* Z by $f_Z^k(\vec{x}) = \sum_{z \in Z} \sum_{i=1}^{k} x_i^z$, where $\vec{x} = (x_1, \dots, x_k)$.
- For two sets $X, Y \subseteq \mathbb{N}$, let
  - ▷ $X \oplus Y = \{x + y \mid x \in X \cup \{0\}, y \in Y \cup \{0\}\}$ denote the *sumset* or *pairwise sum* of sets X and Y.
  - ▷ $X \oplus_t Y = (X \oplus Y) \cap [t]$ denote the *t-capped sumset* or *t-capped pairwise sum* of sets X and Y. Note that $t > 0$.
- The pairwise sum operations can be extended to sets of multiple dimensions. Formally, let $X, Y \subseteq \mathbb{N}^k$. Then, $X \oplus Y = \{(x_1 + y_1, \dots, x_k + y_k)\}$, where $(x_1, \dots, x_k) \in X \cup \{0\}^k$ and $(y_1, \dots, y_k) \in Y \cup \{0\}^k$. Similarly, $X \oplus_t Y = (X \oplus Y) \cap ([t])^k$.

2.2 Theoretical Background

2.2.1 NP-completeness, Pseudopolynomial and Randomised Algorithms

In computational complexity theory, NP is the set of decision problems[1] for which the problem instances, where the answer is "yes", have proofs verifiable in polynomial time by a deterministic Turing machine. That is, given an input instance of the problem, we can verify that the output is indeed "yes" in polynomial time. Obviously, if a problem is solvable in polynomial time (class P), then it is also verifiable in polynomial time, hence P $\subseteq$ NP.

If a problem Π_1 is reducible to a problem Π_2 in polynomial time ($\Pi_1 \leq_P \Pi_2$), then:

- There exists a polynomially computable function $R : \Sigma^* \to \Sigma^*$, such that $\forall x \in \Sigma^*$, it holds that $x \in \Pi_1 \iff R(x) \in \Pi_2$, where Σ^* is the set of all possible inputs and $x \in \Pi$ if and only if input x results in an output "yes" for problem Π.
- Function R is called a *polynomial reduction*.
- Essentially, we (efficiently) transform each input instance of problem Π_1 to an input instance of problem Π_2 and subsequently solve problem Π_2.

A decision problem Π is *NP-hard* if every problem in NP is reducible to Π in polynomial time. Additionally, if Π is in NP, then Π is *NP-complete*. The NP-complete problems represent the hardest problems in class NP. We subsequently define a specific subset of NP-complete problems: weakly NP-complete problems.

A problem is *weakly NP-complete* if there is an algorithm for it, whose running time is polynomial in the numeric value of the input (the largest integer present in the input) but not necessarily in the length of the input (the number of bits required to represent it), which is the case for polynomial time algorithms. Such an algorithm is called *pseudopolynomial*, since the numeric value of the input is exponential in the input length.

A *randomised* algorithm is an algorithm that employs a degree of randomness as part of its logic. In contrast to a *deterministic* algorithm, the output of a randomised one may be different among distinct calls with the same input. Hence, there is a chance of producing an incorrect result (commonly symbolised by δ). These algorithms are also referred to as *probabilistic* and their complexity is affected by the allowed values of error probability δ.

2.2.2 Polynomials, FFT and Subset Sum

In this subsection, we will show the relationship between Subset Sum and the Fast Fourier Transform algorithm.

Let $Z = \{z_1, \ldots, z_n\}$ be a set of positive integers and $t > 0$ be a given target. We seek to determine whether there exists a subset $S \subseteq Z$, such that the sum of its elements equals t. For each $z_i \in Z$, we define a polynomial $x^0 + x^{z_i} = 1 + x^{z_i}$. This polynomial is actually the characteristic polynomial of a set $\{0, z_i\}$. In this case, $x^0 = 1$ represents the exclusion of the element z_i in the result of the product, whereas x^{z_i} represents its inclusion. Then, notice that by computing the product of all these polynomials, we can solve Subset Sum. Indeed, the product results in a polynomial

$$A_Z(x) = \prod_{z_i \in Z} (1 + x^{z_i}) = 1 + x^{z_1} + x^{z_2} + \ldots + x^{z_1+z_2} + \ldots + x^{z_1+\ldots+z_n},$$

which has a nonzero coefficient for every term x^s with an exponent $s \in \mathcal{S}(Z)$ a possible subset sum of Z.

[1] Decision problems are those that have as an output either "yes" or "no".

Consequently, if, given two sets $A, B \subseteq \mathbb{N}$, we seek their pairwise sum $A \oplus B$, we can represent each set by its respective characteristic polynomial f_A, f_B and subsequently multiply these polynomials, producing a polynomial consisting of terms $1, x^a, x^b$ and x^{a+b} with nonzero coefficient, where $a \in A$ and $b \in B$.

Thus, it is very important to seek efficient algorithms regarding polynomial multiplication. The most efficient such algorithm is called *Fast Fourier Transform* (or FFT for short). There are multiple algorithms implementing FFT sharing asymptotically the same complexity of $\Theta(n \log n)$ for polynomials of degree[2] n, although there is no known proof that a lower complexity score is impossible.

Prior to shortly describing the FFT algorithm, we note the following:

- Any integer strictly greater than the degree of a polynomial is a *degree-bound* of that polynomial. Therefore, the degree of a polynomial of degree-bound n may be any integer between 0 and $n - 1$, inclusive.
- A polynomial $A(x) = \sum_{i=1}^{n} a_i x^i$ of degree n can be described by a vector $v_A = (a_1, \ldots, a_n)$. This is called its *coefficient representation*.
- We can compute the value $A(x_0) = \sum_{i=1}^{n} a_i {x_0}^i$ of a polynomial of degree n at a given point x_0 in time $\mathcal{O}(n)$, hence producing a point - value pair $(x_0, A(x_0))$. This is also called the evaluation of the polynomial at a given point. A *point-value representation* of a polynomial $A(x)$ of degree-bound n is a set of n such point-value pairs.
- It is possible to determine the coefficient form of a polynomial given its point-value representation. This procedure is called *interpolation* and it is possible if the desired interpolating polynomial has a degree-bound equal to the given number of point-value pairs.

Given two polynomials $A(x), B(x)$ of degree n, FFT includes the following steps:

1. We firstly extend polynomials $A(x), B(x)$ by adding n high-order zero coefficients to each.
2. Next, we evaluate the polynomials $A(x)$ and $B(x)$ at $2n$ different points $x_1, \ldots, x_{2n}$.
3. Subsequently, we compute a point-value representation for the product polynomial $C(x) = A(x) \cdot B(x)$ by multiplying these values together pointwise. Note that $2n$ is a degree-bound of $C(x)$.
4. Lastly, we interpolate polynomial $C(x)$ from its point-value representation produced in the previous step.

Steps 1 and 3 take time $\mathcal{O}(n)$. Steps 2 and 4 take time $\mathcal{O}(n \log n)$ if we carefully choose appropriate points x_i, as explained in [Corm09, Section 30.2]. Thus, the total complexity for the multiplication of two polynomials of degree n is $\Theta(n \log n)$.

Note that the algorithm can be extended to multiple dimensions, hence it is possible to compute the product of polynomials $A(x_1, \ldots, x_k) \cdot B(x_1, \ldots, x_k)$, where polynomial A is defined as

$$A(x_1, \ldots, x_k) = \sum_{(i_1, \ldots, i_k) \in R} a_{i_1 \ldots i_k} x_1^{i_1} \ldots x_k^{i_k} \text{ and } R = [n_1] \times \ldots \times [n_k]$$

and polynomial B is defined accordingly. To compute the product in this case, we simply perform a sequence of k one-dimensional FFTs: firstly we transform along the n_1 dimension,

[2] A polynomial $P(x) = \sum_{i=0}^{n} a_i x^i$ is of degree n if its highest nonzero coefficient is a_n.

then along the n_2 dimension, and so on (or actually, any ordering works). This method has time complexity $\mathcal{O}(N \log N)$, where $N = n_1 \cdot n_2 \cdot \cdots \cdot n_k$ is the total number of data points transformed. In particular, there are N/n_1 transforms of size n_1, N/n_2 transforms of size n_2 and so on, so the complexity of the sequence of FFTs is:

$$\frac{N}{n_1}\mathcal{O}(n_1 \log n_1) + \ldots + \frac{N}{n_k}\mathcal{O}(n_k \log n_k) = \mathcal{O}(N \sum_{i=1}^{k} \log n_i) = \mathcal{O}(N \log N).$$

2.2.3 Relationship between Subset Sum and Equal Subset Sum

Mucha *et al.* present a reduction Subset Sum $\leq$ Equal Subset Sum in the full version of their paper [Much19]. This reduction is slightly sharper than the one presented in [Woeg92] and shows the intrinsic relationship between the two algorithmic problems.

Theorem. *If* Equal Subset Sum *can be solved in time* $\mathcal{O}^*((2-\varepsilon)^{0.25n})$ *for some* $\varepsilon > 0$, *then* Subset Sum *can be solved in time* $\mathcal{O}^*((2-\varepsilon')^{0.5n})$ *for some constant* $\varepsilon' > 0$.

Proof. Assume that we have a black-box access to the Equal Subset Sum algorithm running in time $\mathcal{O}^*((2-\varepsilon)^{0.25n})$ for some $\varepsilon > 0$. We will show how to use this algorithm to obtain an algorithm for Subset Sum running in time $\mathcal{O}^*((2-\varepsilon)^{0.5n})$.

Given an instance (S, t) of Subset Sum such that $S = \{s_1, \ldots, s_n\}$, we will construct an equivalent instance Z of Equal Subset Sum such that $Z = \{z_1, \ldots, z_{2n+1}\}$. The construction is as follows:

- for $1 \leq i \leq n$, let $z_i = s_i \cdot 10^{n+1} + 2 \cdot 10^i$,
- for $1 \leq i \leq n$, let $z_{i+n} = 1 \cdot 10^i$,
- let $z_{2n+1} = t \cdot 10^{n+1} + \sum_{i=1}^{n} 1 \cdot 10^i$.

Now, we will show that if (S, t) is a YES instance for Subset Sum, then Z is a YES instance for Equal Subset Sum. Let $X \subseteq [n]$, such that $\sum_{i \in X} s_i = t$, be the set of the indexes of the elements of the solution set. Then, sets $A = \{z_i \mid i \in X\} \cup \{z_{i+n} \mid i \notin X\}$ and $B = \{z_{i+n} \mid i \in X\} \cup \{z_{2n+1}\}$ are a valid solution to Equal Subset Sum on instance Z, since $\Sigma(A) = \Sigma(B)$ and $A \cap B = \emptyset$.

For the other direction, we will prove that if Z is a YES instance of Equal Subset Sum, then (S, t) is a YES instance of Subset Sum. Assume that Z is a YES instance and sets $A, B \subseteq Z$ is a correct solution pair. Observe that if for some $i \leq n$ element $z_i \in A$, then $z_{2n+1} \in B$. That is due to the fact that the sets A, B have an equal sum and only the elements z_i, z_{i+n} and z_{2n+1} have something nonzero at the i-th decimal place. Moreover, all smaller decimal places of all numbers sum up to something smaller than 10^i and therefore cannot interfere with the i-th decimal.

Finally, observe that numbers z_{i+n}, for $i \in [n]$, cannot produce a YES instance on their own. Hence, sets $A \cup B$ contain at least one number z_i for $i \in [n]$. Without loss of generality, let A be the set that contains such an z_i. Then, set B has to contain the element z_{2n+1}. That means that set B cannot contain any z_i for $i \in [n]$.

In particular, $\Sigma(A)/10^{n+1} = \Sigma(B)/10^{n+1}$. Only numbers z_i for $i \in [n]$ contribute to $\Sigma(A)/10^{n+1}$ and only number z_{2n+1} contributes to $\Sigma(B)/10^{n+1}$. Hence, there exists a subset $S' \subseteq S$, such that $\Sigma(S') = t$.

□

2.2.4 Congruence Classes

Given an integer $n > 1$, two integers a, b are said to be *congruent modulo* n, if n is a divisor of their difference (i.e., there exists an integer $k \in \mathbb{Z}$ such that $a-b = kn$). Congruence modulo n is an equivalence relation[3] that is compatible with the operations of addition, subtraction, and multiplication and is denoted by $a \equiv b \pmod{n}$.

The set of all congruence classes of the integers for a modulus n is called the ring of integers modulo n and is denoted by $\mathbb{Z}_n = \{\overline{0}, \ldots, \overline{n-1}\}$. Each of its elements $\overline{i}$ represents the set of integers that are congruent modulo n with i, i.e. $\forall x \in \mathbb{Z}, x \in \overline{i} \iff x \equiv i \pmod{n}$.

2.3 Techniques

Pairwise Sum A common trait of the algorithms studied in this thesis is the partition of the initial input set Z to subsets $Z_1, \ldots, Z_k$ and the recursive computation of the subset sums of the resulting sets. Afterwards, these subset sums are combined to compute the subset sums of the original input set Z. Thus, it is of great importance the efficient computation of the sumset operations, since they have a central role in the definition of the overall complexity of the algorithm.

Color-coding Color-coding is an algorithmic technique that was first used for the k-Path problem: Given a graph G, decide whether it contains a path of length k [Alon95]. The idea is to randomly color the vertices of G with k colors, so that for a fixed path of length k in G with probability $1/k!$ it is colored $(1, 2, \ldots, k)$, in which case we can find it by a simple dynamic programming algorithm on the layered graph obtained from keeping only the edges in G from color class i to $i+1$ (for each i). Over $\mathcal{O}(k! \log n)$ repetitions we find a k-path with high probability, if one exists. Research on color-coding has led to various improvements and derandomizations of this technique [Alon95, Naor95, Schm90].

We make use of color-coding for finding all sums generated by small subsets. More precisely, given a Subset Sum instance (Z, t) and a threshold k, we compute a set $S \subseteq \mathcal{S}_t(Z)$ containing any sum generated by a subset $Y \subseteq Z$ of size $|Y| \leq k$ with constant probability. This can be boosted to any high probability by repeating and taking the union. The main trick is to randomly partition $Z = Z_1 \cup \ldots \cup Z_{k^2}$ by assigning to any element $z \in Z$ a color in $\{1, \ldots, k^2\}$ independently and uniformly at random.

Layer Splitting Any Subset Sum instance (Z, t) can be partitioned into $\log n$ layers $Z_i \subseteq [t/2^i, t/2^{i-1}]$, plus a set $Z_0 \subseteq [0, t/n]$ that can be treated very similarly to layers and that we ignore here for simplicity. To obtain the desired set of all subset sums $\mathcal{S}_t(Z)$, it suffices to firstly compute the respective sets of all subset sums $\mathcal{S}_t(Z_i)$ of the layers and subsequently combine them using sumset computations.

It remains to compute $\mathcal{S}_t(Z_i)$ for the various layers. Note that by the definition of the layers, any set $Y \subseteq Z_i$ with $\Sigma(Y) \leq t$ has size at most $|Y| \leq 2^i$. Thus, the color-coding algorithm with $k = 2^i$ successfully computes $\mathcal{S}_t(Z_i)$ in time $\tilde{\mathcal{O}}(n + tk^2)$. We strive for an $\tilde{\mathcal{O}}(n + t)$ algorithm, hence we need to eliminate the factor k^2. We firstly note that all items in Z_i are bounded by $\mathcal{O}(t/k)$, which allows us to implement the sumset computations in the color-coding algorithm more efficiently. To remove the remaining factor k, we implement a two-stage approach: In the first stage, we partition Z_i into roughly $k = 2^i$ sets $Z_{i,j}$. This k is too small to split Y entirely, i.e., to have $|Z_{i,j} \cap Y| \leq 1$ for all j with high probability. For this property to hold, we would need to partition Z_i into k^2 sets. However, it holds

[3] An equivalence relation has the following properties: a) Reflexivity, b) Symmetry and c) Transitivity. In our case, that translates to a) $a \equiv a \pmod{n}$, b) $a \equiv b \pmod{n}$ if $b \equiv a \pmod{n}, \forall a, b, n$ and c) If $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$, then $a \equiv c \pmod{n}$.

that $|Z_{i,j} \cap Y| \leq \mathcal{O}(\log n)$ with high probability. Hence, in the second stage we can run the color-coding algorithm with size bound $k' = \mathcal{O}(\log n)$ on each $Z_{i,j}$, and then combine their computed sumsets in a straightforward way. Carefully implementing these ideas yields time $\tilde{\mathcal{O}}(n + t)$.

Finding Witnesses via Peeling Each time we compute a sumset operation, it is of utmost importance to be able to efficiently trace the partial sums summed to compute each sum. This is necessary, in order to reconstruct the solution sets and thus solve the *search version* of Equal Subset Sum. Koiliaris and Xu reduce this problem to the *reconstruction problem*, as mentioned in [Auma11] and argue about its complexity, concluding that there is only polylogarithmic overhead to the computation of the sumset operation along with the associated witnesses, in comparison to the computation of the sumset operation. This is a very important result, which indicates that the witness computation does not provoke any complexity 'leap'. In order to complete their reduction, they provide two procedures that successfully compute the number of witnesses for each sum resulting from a sumset operation $Y \oplus W$, as well as their sum. They note that the number of witnesses is represented by the coefficients of the sumset operation $Y \oplus W$, while an extra FFT operation

$$\sum_{y \in Y} yx^y \cdot \sum_{w \in W} x^w$$

is sufficient to compute the sum of the respective witnesses of the sums, represented by the coefficients of this polynomial.

2.4 Observations and Lemmas

The following observation is crucial for the correctness of the presented algorithms, since it plays a central role in all the divide and conquer approaches.

Observation 1. *Let $X, Y \subseteq \mathbb{N}$ be two disjoint sets. Then, $\mathcal{S}(X \cup Y) = \mathcal{S}(X) \oplus \mathcal{S}(Y)$ and $\mathcal{SC}(X \cup Y) = \mathcal{SC}(X) \oplus \mathcal{SC}(Y)$.*

The following lemmas will be used throughout the rest of this thesis.

Lemma 2.1. *Let g be a positive, superadditive function. Since g is a superadditive function, it holds that $g(x + y) \geq g(x) + g(y), \forall x, y$. Then, for a function f satisfying*

$$f(n, m) = \max_{m_1 + m_2 = m} \left\{ f(\frac{n}{2}, m_1) + f(\frac{n}{2}, m_2) + g(m) \right\},$$

it holds that $f(n, m) = \mathcal{O}(g(m) \log n)$.

Proof. Figure 2.1 demonstrates the recursion tree of function f. Each level of the recursion tree has a cost of $\mathcal{O}(g(m))$ and there are $\mathcal{O}(\log n)$ levels, hence the total complexity of the function is $\mathcal{O}(g(m) \log n)$. □

Lemma 2.2. *Given two sets $X, Y \subseteq [t]$, one can compute $X \oplus Y$ and $X \oplus_t Y$ in $\mathcal{O}(t \log t)$ time.*

Proof. Let f_X and f_Y be the characteristic polynomials of sets X and Y respectively and let $g = f_X \cdot f_Y$. Observe that for $i \leq 2t$, the coefficient of x^i in g is nonzero if and only if $i \in X \oplus Y$. Using Fast Fourier Transform(FFT) [Corm09, Chapter 30], one can compute the polynomial g in $\mathcal{O}(t \log t)$ time and extract $X \oplus Y$ and $X \oplus_t Y$ from it. □

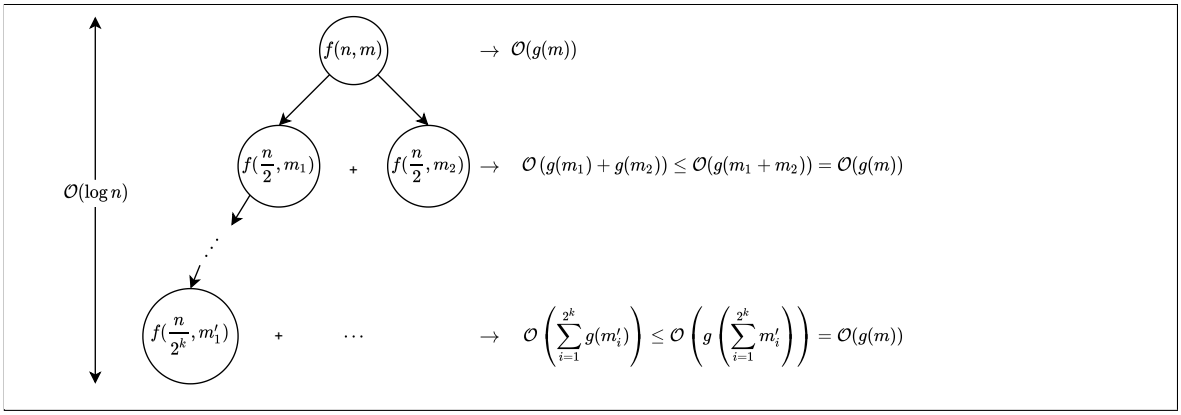

Figure 2.1: Recursion tree of function $f(n, m)$.

Additionally, one can perform the previous computation along with finding the associated witnesses in $\tilde{\mathcal{O}}(t)$, as argued in [Koil19].

Lemma 2.3. *Given two sets $S, T \subseteq [u] \times [v]$, one can compute $S \oplus T$ in $\mathcal{O}(uv \log(uv))$ time.*

Proof. Let $f_S = f_S(x, y) = \sum_{(i,j)\in S} x^i y^j$ and $f_T = f_T(x, y) = \sum_{(i,j)\in T} x^i y^j$ be the characteristic polynomials of sets S and T respectively and let $g = f_S \cdot f_T$. For $i \le u$ and $j \le v$, the coefficient of $x^i y^j$ is nonzero if and only if $(i, j) \in S \oplus T$. One can compute the polynomial g by a straightforward reduction to regular FFT (see multidimensional FFT [Blah85, Chapter 12.8]), in $\mathcal{O}(uv \log(uv))$ time, and extract $S \oplus T$ from it. □

Lemmas 2.2 and 2.3 can be extended to multiple dimensions, as we have already argued. Hence, the following corollary holds true.

Corollary. *Given two sets $X, Y \subseteq [t_1] \times \cdots \times [t_k]$, one can compute $X \oplus Y$ in time $\mathcal{O}(t_1 \cdots t_k \log(t_1 \cdots t_k)) = \mathcal{O}(t^k \log t)$, where $t = \max\{t_1, \ldots, t_k\}$.*

Chapter 3

Subset Sum Algorithms

In this chapter we will present the original algorithms proposed by Bringmann [Brin17] and Koiliaris and Xu [Koil19], which we will modify and rely upon for the rest of this thesis. These algorithms solve the Subset Sum problem, which asks, given a set Z of n positive integers and a target $t > 0$, to determine whether there exists a subset of Z whose elements sum up to t. The first pseudopolynomial time approach was introduced by Bellman in [Bell57] with complexity $\mathcal{O}(nt)$ and the algorithm is presented below. The algorithms presented in this chapter are significantly more efficient and involve modern techniques such as color-coding and FFT. For the rest of this chapter, suppose that $Z = \{z_1, \ldots, z_n\} \subseteq \mathbb{N}$ is the input set, where $n = |Z|$, and $t > 0$ is the given upper bound.

Algorithm 1: `Bellman(Z, t)`

Input : A set $Z = \{z_1, \ldots, z_n\}$ of positive integers and a target t.
Output: True if there exists a subset $S \subseteq Z$, such that $\Sigma(S) = t$, else false.

```
initialise table T[n][t] ← false everywhere
T[0][0] ← true                          // Initially, only Σ(∅) = 0 is valid.
for k = 1, ..., n do
    for i = 1, ..., t do
        if T[k − 1][i] = true then
            T[k][i] = true                                  //  z_k ∉ S
            T[k][i + z_i] = true                            //  z_k ∈ S
        end
    end
end
return T[n][t]
```

3.1 A randomised $\tilde{\mathcal{O}}(n + t)$ algorithm

In this section we will present the randomised Subset Sum algorithm introduced by Bringmann of complexity $\tilde{\mathcal{O}}(n + t)$, as presented in [Brin17]. This algorithm is certainly more convoluted in comparison to the rest of the presented algorithms, but constitutes the first pseudolinear (with regards to t) time approach to the Subset Sum problem.

The algorithm has the following structure:

1. The input set Z is initially split to $\log n$ *l-layer* subsets.
2. Afterwards, each *l-layer* subset is split to multiple *small solution size* subsets.
3. We solve Subset Sum for each such occurring *small solution size* subset.
4. We combine the solutions to solve Subset Sum for each *l-layer* subset.

5. We combine the solutions to solve Subset Sum for the initial input set Z.

Figure 3.1 depicts the partition of the original input set Z. Notice that all sets in the same level of the tree are disjoint, since they originate from the partition of a set from the previous level.

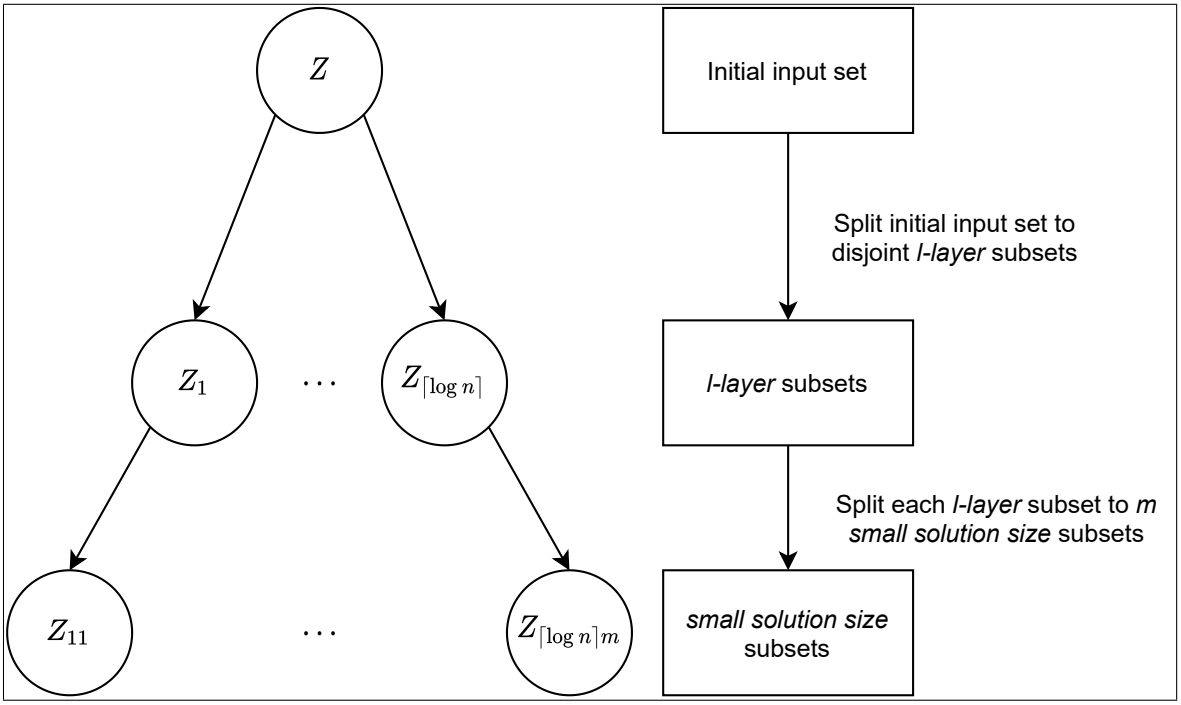

Figure 3.1: Partition of the input set Z, according to Bringmann's algorithm.

3.1.1 Subset Sum algorithm for small solution size subsets

We initially describe an algorithm for solving instances of Subset Sum when the solution size is small, i.e., an algorithm that finds sums $\Sigma(Y_i) \leq t$ generated by sets $Y_i \subseteq Z$ of size at most k, for some given (small) k.

We randomly partition our initial set Z to k^2 subsets $Z_1, \ldots, Z_{k^2}$, i.e., we assign each $z \in Z$ to a set Z_i where i is chosen independently and uniformly at random from $\{1, \ldots, k^2\}$. We say that this random partition *splits* $Y \subseteq Z$ if $|Y \cap Z_i| \leq 1, \forall i$. If such a split occurs, the set returned by Algorithm 2 will contain $\Sigma(Y)$. Indeed, by choosing the element of Y for those Z_i that $|Y \cap Z_i| = 1$ and 0 for the rest, we successfully generate $\Sigma(Y)$ through the sumset operations. The algorithm returns only valid sums (i.e. the solution set S is a subset of $\mathcal{S}_t(Z)$), since no element is used more than once in each sum, because each element is assigned uniquely to a Z_i for each distinct partition.

Repeating this procedure sufficiently often with fresh randomness, and taking the union over all computed sumsets $Z_1 \oplus_t \cdots \oplus_t Z_{k^2}$ yields a set $S \subseteq \mathcal{S}_t(Z)$ containing any $\Sigma(Y) \leq t$ with $Y \subseteq Z$ and $|Y| \leq k$ with probability at least $1 - \delta$.

For any $Y \subseteq Z$ with $|Y| \leq k$, observe that the probability that some random partition splits subset Y is the same as having $|Y|$ different balls in $|Y|$ distinct bins, when throwing $|Y|$ balls into k^2 different bins. This is equivalent to the second ball falling into a different bin than the first one, the third ball falling into a different bin than the first two, and so on, which has probability

$$\frac{k^2-1}{k^2} \cdot \frac{k^2-2}{k^2} \cdots \frac{k^2-(|Y|-1)}{k^2} \geq \left(\frac{k^2-(|Y|-1)}{k^2}\right)^{|Y|} \geq \left(1-\frac{1}{k}\right)^k \geq \left(\frac{1}{2}\right)^2 = \frac{1}{4}.$$

Hence, $r = \lceil \log_{4/3}(1/\delta) \rceil$ repetitions yield the desired success probability of $1 - (1 - 1/4)^r \geq 1 - \delta$. In other words, after r random partitions, for any random subset Y, there exists, with probability at least $1 - \delta$, a partition that splits it.

Complexity. Algorithm 2 performs $\mathcal{O}(\log(1/\delta))$ repetitions. To compute a pairwise sum up to t, $\mathcal{O}(t \log t)$ time is required. In each repetition, k^2 pairwise sums are computed. Hence, the total complexity of the algorithm is $\mathcal{O}(tk^2 \log t \log(1/\delta))$.

Algorithm 2: `ColorCoding(Z, t, k, δ)`

Input : A set Z of positive integers, an upper bound t, a size bound $k \geq 1$ and an error probability $\delta > 0$.

Output: A set $S \subseteq \mathcal{S}_t(Z)$ containing any $\Sigma(Y) \leq t$ with probability at least $1 - \delta$, where $Y \subseteq Z$ and $|Y| \leq k$.

```
S ← ∅
for j = 1, ..., ⌈log_{4/3}(1/δ)⌉ do
    randomly partition Z = Z_1 ∪ Z_2 ∪ ··· ∪ Z_{k^2}
    S_j ← Z_1 ⊕_t ··· ⊕_t Z_{k^2}
    S ← S ∪ S_j
end
return S
```

Lemma 3.1. *Algorithm 2 computes in time $\mathcal{O}(tk^2 \log t \log(1/\delta))$ a set $S \subseteq \mathcal{S}_t(Z)$, such that for any $Y \subseteq Z$ with $|Y| \leq k$ and $\Sigma(Y) \leq t$, we have $\Sigma(Y) \in S$ with probability at least $1 - \delta$.*

3.1.2 Subset Sum algorithm for l-layer instances

In this subsection we will present an efficient randomised algorithm for solving Subset Sum for l-layer instances. Let (Z, t) be a Subset Sum instance, where $n = |Z|$. For $l \geq 1$, we call (Z, t) an *l-layer* instance if

$$Z \subseteq [t/l, 2t/l] \qquad \text{or} \qquad Z \subseteq [0, 2t/l] \text{ and } l \geq n.$$

In both cases, we have that $Z \subseteq [0, 2t/l]$. Additionally, note that any $Y \subseteq Z$ with $\Sigma(Y) \leq t$ has size $|Y| \leq l$. In the first case, if $|Y| > l$, then $\Sigma(Y) \geq |Y| \cdot t/l > l \cdot t/l = t$. In the second case, this holds since $|Y| \leq |Z| = n$. Thus, the `ColorCoding` algorithm previously presented, with size bound $k = l$ can be used to solve l-layer instances. We will show that the running time can be improved for layers, thereby essentially removing the quadratic dependence on l entirely.

Let $m \leftarrow l/\log(l/\delta)$ rounded up to the next power of 2. We randomly partition Z into subsets $Z_1, \ldots, Z_m$ and for each such occurring subset run Algorithm `ColorCoding` with size bound $k = 6\log(l/\delta)$, target $12\log(l/\delta)t/l$ and error probability δ/l, yielding sets $S_1, \ldots, S_m$ respectively. Afterwards, we combine the sets $S_1, \ldots, S_m$ in a natural, binary-tree-like way by computing $S_1 \oplus S_2, \ldots, S_{m-1} \oplus S_m$ in the first round, $(S_1 \oplus S_2) \oplus (S_3 \oplus S_4), \ldots, (S_{m-3} \oplus S_{m-2}) \oplus (S_{m-1} \oplus S_m)$ in the second round, and so on, until we reach the set $S_1 \oplus \ldots \oplus S_m$. Note that in the h-th round of this procedure we combine 2^h sets S_j, each initially containing integers bounded by $12\log(l/\delta)t/l$. Thus, we may use $\oplus_{2^h \cdot 12\log(l/\delta)t/l}$ in the h-th round. This explains the following algorithm.

Henceforth, assume that $Y \subseteq Z$ is a fixed subset of input set Z, with $\Sigma(Y) \leq t$. Also, let $Y_j \leftarrow Y \cap Z_j$ for $1 \leq j \leq m$, where $Z_1, \ldots, Z_m$ are the sets occurring from the partitioning of set Z to m subsets. We will firstly prove a crucial property for subset Y.

Claim 3.2. *It holds that* $\Pr[|Y_j| \geq 6\log(l/\delta)] \leq \delta/l$.

Algorithm 3: `ColorCodingLayer(Z, t, l, δ)`

Input : An l-layer instance (Z, t) and an error probability $\delta \in (0, 1/4]$.
Output: A set $S \subseteq \mathcal{S}_t(Z)$ containing any $s \in \mathcal{S}_t(Z)$ with probability at least $1 - \delta$.

```
if l < log(l/δ) then return ColorCoding(Z, t, l, δ)
m ← l/log(l/δ) rounded up to the next power of 2
randomly partition Z = Z_1 ∪ Z_2 ∪ ··· ∪ Z_m
γ ← 6 log(l/δ)
for j = 1, ..., m do
    S_j ← ColorCoding(Z_j, 2γt/l, γ, δ/l)
end
for h = 1, ..., log m do                    // combine S_j in a binary-tree-like way
    for j = 1, ..., m/2^h do
        S_j ← S_{2j-1} ⊕_{2^h · 2γt/l} S_{2j}
    end
end
return S_1 ∩ {0, ..., t}
```

Proof. Note that $|Y_j|$ is distributed as the sum of $|Y|$ independent Bernoulli random variables with success probability $1/m$. In particular, $\mu = \mathbb{E}[|Y_j|] = |Y|/m$. A standard Chernoff bound yields that $\Pr[|Y_j| \geq \lambda] \leq 2^{-\lambda}$ for any $\lambda \geq 2e\mu$. Recall that (Z, t) is an l-layer instance, thus $|Y| \leq l$, hence $\mu = |Y|/m \leq l/m \leq \log(l/\delta)$, by definition of m. Thus, the inequality holds for $\lambda = 6\log(l/\delta)$, and we obtain $\Pr[|Y_j| \geq 6\log(l/\delta)] \leq 2^{-6\log(l/\delta)} = (\delta/l)^6 \leq \delta/l$. □

Lemma 3.3. *For an l-layer instance (Z, t) and $\delta \in (0, 1/4]$, Algorithm 3 computes in time $\mathcal{O}(t \log t \log^3(l/\delta))$ a set $S \subseteq \mathcal{S}_t(Z)$ containing any $s \in \mathcal{S}_t(Z)$ with probability at least $1 - \delta$.*

Proof. The case $l < \log(l/\delta)$ follows from Lemma 3.1. Inclusion $S \subseteq \mathcal{S}_t(Z)$ also follows from Lemma 3.1 and also from the fact that we only compute sumsets over partitionings[1], hence $(\mathcal{S}_{t_1}(Z_1) \oplus_{t'} \mathcal{S}_{t_2}(Z_2)) \cap [t] \subseteq \mathcal{S}_t(Z)$ for a partitioning $Z = Z_1 \cup Z_2$ and any $t, t', t_1, t_2 \geq 1$.

By Claim 3.2, we may assume that $|Y_j| \leq 6\log(l/\delta)$ holds for each $1 \leq j \leq m$; this happens with probability

$$\Pr[\bigwedge_{j=1}^{m} (|Y_j| \leq 6\log(l/\delta))] \geq 1 - \sum_{j=1}^{m} \Pr[|Y_j| > 6\log(l/\delta)] \geq 1 - m\delta/l.$$

Since $Z \subseteq [0, 2t/l]$, any subset of Z_j of size at most $6\log(l/\delta)$ has sum at most $12\log(l/\delta)t/l$. From Lemma 3.1 follows that the call `ColorCoding`$(Z_j, 12\log(l/\delta) \cdot t/l, 6\log(l/\delta), \delta/l)$ finds $\Sigma(Y_j)$ with probability at least $1 - \delta/l$. Assume that this event holds for each $1 \leq j \leq m$; this happens with probability at least $1 - m \cdot \delta/l$. Then S_j contains $\Sigma(Y_j)$, and the tree-like sumset computation indeed yields a set containing $\Sigma(Y_1) + \ldots + \Sigma(Y_m) = \Sigma(Y)$.

The total error probability is $2m\delta/l$. Since $l \geq 1$ and $\delta \leq 1/4$, we have $\log(l/\delta) \geq 2$ and obtain $m \leq l/2$. Hence, the total error probability is bounded by δ. □

Complexity. By Lemma 3.1, each call to `ColoCoding` takes $\mathcal{O}(t/l \cdot \log^4(1/\delta) \log t)$ time. There are $m = \Theta(l/\log(l/\delta))$ such calls, thus $\mathcal{O}(t \log^3(1/\delta) \log t)$ is required for all of them. To combine the resulting sets costs

$$\mathcal{O}\left(\sum_{h=1}^{\log m} \frac{m}{2^h} \cdot 2^h \log(l/\delta)t/l \cdot \log t\right) = \mathcal{O}(t \log t \log m),$$

which is dominated by the total time for calling `ColorCoding`.

[1] See Figure 3.1 at the start of the section and also Observation 1 at Chapter 2.

3.1.3 General case algorithm

Algorithm 4 demonstrates how we can split a random input set Z to $\mathcal{O}(\log n)$ l-layer subsets, yielding layers $Z_1, \ldots, Z_{\lceil \log n \rceil}$. On each layer we run algorithm `ColorCodingLayer`, which was previously presented, and subsequently combine the resulting sumsets S_i. The error probabilities of the calls to `ColorCodingLayer` are appropriately chosen in order to sum up to at most δ. Correctness stems from Lemma 3.3.

Complexity. For each call `ColorCodingLayer`(Z, t, l, δ), $\mathcal{O}(t \log t \log^3(l/\delta))$ time is required. To compute a pairwise sum up to t, $\mathcal{O}(t \log t)$ time is required. $\mathcal{O}(n)$ time is required for reading the input set. Hence, Algorithm 4 has total complexity

$$\mathcal{O}\left(n + \sum_{i=1}^{\log n}\left(t \log t \log^3(\frac{2^i \log n}{\delta}) + t \log t\right)\right) = \mathcal{O}(n + t \log t \log^3(n/\delta) \log n) = \tilde{\mathcal{O}}(n + t).$$

Algorithm 4: `FasterSubsetSum`(Z, δ, t)

Input : A set of positive integers Z, an upper bound t and an error probability δ.
Output: A set $S \subseteq \mathcal{S}_t(Z)$ containing any $s \in \mathcal{S}_t(Z)$ with probability at least $1 - \delta$.

partition Z into $Z_i \leftarrow Z \cap (t/2^i, t/2^{i-1}]$ for $i = 1, \ldots, \lceil \log n \rceil - 1$ and $Z_{\lceil \log n \rceil} \leftarrow Z \cap [0, t/2^{\lceil \log n \rceil - 1}]$
$S \leftarrow \emptyset$
for $i = 1, \ldots, \lceil \log n \rceil$ **do**
  $S_i \leftarrow$ `ColorCodingLayer`$(Z_i, t, 2^i, \delta/\lceil \log n \rceil)$
  $S \leftarrow S \oplus_t S_i$
end
return S

Theorem 3.4. Subset Sum *can be solved in time* $\mathcal{O}(n + t \log t \log^3(n/\delta) \log n) = \tilde{\mathcal{O}}(n + t)$ *by a randomised, one-sided error algorithm with error probability* $\delta = (n + t)^{-\Omega(1)}$.

3.2 A deterministic $\tilde{\mathcal{O}}(\sigma)$ algorithm

Koiliaris and Xu have shown in [Koil19, Theorem 3.1] that one can compute all subset sums of a set Z in $\mathcal{O}(\sigma \log \sigma \log n)$, where $\sigma = \Sigma(Z)$ and $n = |Z|$.

Note that this algorithm is incomparable to the other Subset Sum algorithms mentioned in this chapter since we cannot efficiently solve bounded instances by using it. Indeed, for a given upper bound t, its recursion tree yields $\mathcal{O}(t \log t \sum_{i=0}^{\log n} 2^i) = \mathcal{O}(nt \log t)$ complexity.

Theorem 3.5. *Given a set* $Z \subseteq \mathbb{N}$, *where* $n = |Z|$ *and* $\sigma = \Sigma(Z)$, *one can compute the set of all subset sums* $\mathcal{S}(Z)$ *in* $\mathcal{O}(\sigma \log \sigma \log n) = \tilde{\mathcal{O}}(\sigma)$ *time.*

Proof. Partition Z into two sets L, R of (roughly) equal cardinality and compute recursively $L' = \mathcal{S}(L)$ and $R' = \mathcal{S}(R)$. Then, by Observation 1, compute $\mathcal{S}(Z) = L' \oplus R'$. The recurrence for the running time is

$$f(n, \sigma) = \max_{\sigma_1 + \sigma_2 = \sigma} \{f(n/2, \sigma_1) + f(n/2, \sigma_2) + \mathcal{O}(\sigma \log \sigma)\}.$$

Since $\mathcal{O}(\sigma_1 \log \sigma_1) + \mathcal{O}(\sigma_2 \log \sigma_2) \leq \mathcal{O}(\sigma \log \sigma)$, i.e. a superadditive function, we have that $f(n, \sigma) = \mathcal{O}(\sigma \log \sigma \log n)$, since Lemma 2.1 holds. □

Algorithm 5: `SSNoBoundSum(Z)`

Input : A set Z of n positive integers.
Output: The set $\mathcal{S}(Z)$ of all subset sums of Z.

```
1 if Z = {z} then return {0, z}
2 σ ← Σ(Z)
3 L ← an arbitrary subset of Z of size ⌊n/2⌋
4 R ← Z \ L
5 L' ← SSNoBoundSum(L)
6 R' ← SSNoBoundSum(R)
7 return L' ⊕_σ R'
```

3.3 A deterministic $\tilde{\mathcal{O}}(\sqrt{n}t)$ algorithm

In this section we will present the algorithm of Koiliaris and Xu of complexity $\tilde{\mathcal{O}}(\sqrt{n}t)$, as presented in [Koil19]. This is a simple, yet efficient algorithm, that partitions the input set by congruence into classes, computes the subset sums of each class recursively, and combines the results, hence computing the subset sums of the input set Z.

3.3.1 Computation of $\mathcal{SC}_t(S)$

An essential component of the algorithm is the efficient computation of $\mathcal{SC}_t(S)$ for some set $S \subseteq \mathbb{N}$. That is, we need an efficient algorithm to compute the set of all possible subset sums from the elements of a set S, along with the cardinality of the corresponding subsets.

Lemma 3.6. *Given a set $S \subseteq [t]$ of n elements, one can compute, in $\mathcal{O}(nt \log n \log t)$ time, the set $\mathcal{SC}_t(S)$, that is, the set of all possible subset sums along with the cardinality of the corresponding subsets.*

Proof. Partition S into two sets S_1 and S_2 of roughly the same size. Compute $\mathcal{SC}_t(S_1)$ and $\mathcal{SC}_t(S_2)$ recursively. Note that $\mathcal{SC}_t(S_1), \mathcal{SC}_t(S_2) \subseteq ([t] \times [\frac{n}{2}])$. Furthermore, note that $\mathcal{SC}_t(S_1) \oplus_t \mathcal{SC}_t(S_2) = \mathcal{SC}_t(S)$, since S_1, S_2 are disjoint and Observation 1 holds. By Lemma 2.3, we can compute $\mathcal{SC}_t(S)$ in $\mathcal{O}(nt \log(nt)) = \mathcal{O}(nt \log t)$ time. The running time follows the recursive formula $T(n) = 2 \cdot T(n/2) + \mathcal{O}(nt \log t)$, which is $\mathcal{O}(nt \log n \log t)$, thus proving the claim. □

The following algorithm successfully computes $\mathcal{SC}_t(S)$ for a given set S of n integers, with time complexity $\mathcal{O}(nt \log n \log t)$.

Algorithm 6: `SSCBoundSum(S, t)`

Input : A set S of n positive integers and an upper bound integer t.
Output: The set $\mathcal{SC}_t(S)$ of all subset sums with cardinality information of S up to t.

```
1 if S = {s} then return {(0, 0), (s, 1)}
2 T ← an arbitrary subset of S of size ⌊n/2⌋
3 return SSCBoundSum(T, t) ⊕_t SSCBoundSum(S \ T, t)
```

3.3.2 Computation of $\mathcal{S}_t(S)$ for elements in the same congruence class

In this subsection, we will show that, if the elements of the input set are in the same congruence class, then it is possible to compute $\mathcal{S}_t(S)$ quickly through the use of Algorithm 6.

Lemma 3.7. *Let $l, b \in \mathbb{N}$, with $l < b$. Given a set $S \subseteq \{x \in \mathbb{N} \mid x \equiv l \pmod b\}$ of size n, one can compute $\mathcal{S}_t(S)$ in $\mathcal{O}((t/b)n \log n \log t)$ time.*

Proof. Each element $x \in S$ can be written as $x = yb + l$. Let $Q = \{y \mid yb + l \in S\}$. Then, for any subset $X = \{y_1 b + l, \ldots, y_j b + l\} \subseteq S$ of size j, we have that

$$\sum_{x \in X} x = \sum_{i=1}^{j} (y_i b + l) = \left(\sum_{i=1}^{j} y_i \right) b + jl.$$

In particular, a pair $(z, j) \in \mathcal{SC}_{t/b}(Q)$ corresponds to a set $Y = \{y_1, \ldots, y_j\} \subseteq Q$ of size j, such that $\Sigma(Y) = z$. The set Y in turn corresponds to the set $X = \{y_1 b + l, \ldots, y_j b + l\} \subseteq S$, where $\Sigma(X) = zb + jl$. As such, compute $\mathcal{SC}_{t/b}(Q)$ using Algorithm 6 and subsequently return $\{zb + jl \mid (z, j) \in \mathcal{SC}_{t/b}(Q)\} = \mathcal{S}_t(S)$ as the desired result. □

3.3.3 The main algorithm

The main algorithm makes use of the previously presented lemmas. First, partition the input set into subsets by congruence. Then, compute $\mathcal{SC}_{t/b}(S)$ for each such set S and lastly combine the results.

Algorithm 7: `SSBoundSum(Z, t)`

Input : A set Z of n positive integers and an upper bound integer t.
Output: The set $\mathcal{S}_t(Z)$ of all subset sums of Z up to t.

1 $b \leftarrow \lfloor \sqrt{n \log n} \rfloor$
2 **for** $l \in [b-1]$ **do**
3 $\quad S_l \leftarrow Z \cap \{x \in \mathbb{N} \mid x \equiv l \pmod b\}$
4 $\quad Q_l \leftarrow \{\lfloor x/b \rfloor \mid x \in S_l\}$
5 $\quad \mathcal{SC}_{t/b}(Q_l) \leftarrow$ `SSCBoundSum`$(Q_l, \lfloor t/b \rfloor)$
6 $\quad \mathcal{S}_t(S_l) \leftarrow \{zb + jl \mid (z, j) \in \mathcal{SC}_{t/b}(Q_l)\}$
7 **end**
8 **return** $\mathcal{S}_t(S_0) \oplus_t \cdots \oplus_t \mathcal{S}_t(S_{b-1})$

Theorem 3.8. *Let $Z \subseteq [t]$ be a given set of n positive integers. One can compute the set $\mathcal{S}_t(Z)$ in $\mathcal{O}(\sqrt{n \log n} t \log t) = \tilde{\mathcal{O}}(\sqrt{n} t)$ time, hence solving* Subset Sum.

Proof. Partition Z into $b = \lfloor \sqrt{n \log n} \rfloor$ sets $S_l = Z \cap \{x \in \mathbb{N} \mid x \equiv l \pmod b\}$ of size n_l respectively, for $l \in [b-1]$. For each S_l, compute the set of all subset sums $\mathcal{S}_t(S_l)$ in $\mathcal{O}\left((t/b) n_l \log n_l \log t\right)$ time by Lemma 3.7. The time to compute all $\mathcal{S}_t(S_l)$ is

$$\sum_{l \in [b-1]} \mathcal{O}((t/b) n_l \log n_l \log t) = \mathcal{O}((t/b) n \log n \log t).$$

Combining the resulting subsets $\mathcal{S}_t(S_0) \oplus_t \cdots \oplus_t \mathcal{S}_t(S_{b-1})$ takes time $\mathcal{O}(bt \log t)$. Thus, the total running time is

$$\mathcal{O}\left(t \log t \left(\frac{n \log n}{b} + b\right)\right) = \mathcal{O}\left(\sqrt{n \log n} t \log t\right) = \tilde{\mathcal{O}}\left(\sqrt{n} t\right).$$

□

Chapter 4

Equal Subset Sum Algorithms

In this chapter, we present four different algorithms that successfully solve the *search version* of Equal Subset Sum. The first three algorithms are based on those presented in Chapter 3. The last one is a simple and efficient deterministic approach that solves Equal Subset Sum, along with producing the solution sets.

We firstly present a randomised algorithm of time complexity $\tilde{\mathcal{O}}(n+t)$, afterwards a deterministic one of complexity $\tilde{\mathcal{O}}(\sigma)$, next a deterministic algorithm of complexity $\tilde{\mathcal{O}}(\sqrt{n}t)$ and lastly a deterministic $\tilde{\mathcal{O}}(n+t)$ one, where n denotes the cardinality of the input set, $t > 0$ an upper bound and σ the sum of the elements of the input set.

The first three algorithms are demonstrating how the techniques presented in the previous chapter can be extended to Equal Subset Sum, whereas the last one was developed based on feedback from anonymous reviewers. It is important to note that, while the first three algorithms are less efficient than the last one, they demonstrate how Subset Sum solutions can be successfully modified and extended to related algorithmic problems. This idea inspired the extensions presented in Chapter 5, where we efficiently solve k-Subset Sum using techniques initially developed for Subset Sum.

4.1 A randomised $\tilde{\mathcal{O}}(n+t)$ algorithm for Equal Subset Sum

In this section, we describe a randomized algorithm which solves Equal Subset Sum for an input set Z in $\tilde{\mathcal{O}}(n+t)$ time, where $n = |Z|$ and t is an upper bound on the subset sums sought. Additionally, this algorithm efficiently reconstructs the solution sets.

The decision version of Equal Subset Sum can be solved in time $\tilde{\mathcal{O}}(n+t)$, by the algorithm of Bringmann [Brin17] for Subset Sum, simply by checking whether there exists a coefficient ≥ 2 in the resulting polynomial, and by increasing the number of repetitions at algorithm `ColorCoding`. The original algorithm(s) of [Brin17] are presented in Section 3.1. Figure 3.1 on page 44 is very helpful regarding the structure of the modified algorithms. Below we present only the required modifications for solving the Equal Subset Sum problem:

- We introduce a witness table W, which holds a witness[1] for sum t as well as a flag table F of integer counters, which will be described below. We assume that W, F are global variables and are initialised by 0 everywhere. More specifically, $W[t] = x$ if sum t occurred by the addition of x and $t-x$. Recursively, $W[x]$ and $W[t-x]$ hold one of the two sums used to compute them respectively. Note that for each element z_i of the initial set Z, we have $W[z_i] = z_i$.

- The detection of a collision is primarily done through the computation of the pairwise sums, i.e. by checking the coefficients of FFT. Apart from the use of FFT, we also take advantage of tables W and F, as well as of a global counter c, that differentiates disjoint

[1] As we have already described in Chapter 2, it is possible to find the witnesses of a pairwise sum operation with only polylogarithmic overhead.

subsets examined by the algorithm in various phases, in order to check if there exists a collision for sum t.

- We change the number of repetitions in Algorithm `ColorCoding`, so that for each pair of sums of disjoint subsets, there exists with high probability a repetition in which both sums are computed.

Before presenting the modified algorithm(s), we will first describe the way the algorithm detects a collision. After successfully detecting a collision, the algorithm reconstructs the solution sets as is thoroughly explained at the end of the section. In general, each time a pairwise sum is computed, we make the following checks:

1. We first check the coefficients of FFT to find if there exists some value greater than 1. If that is the case, we have successfully detected a collision, since each coefficient represents the number of ways a sum can be produced.

2. In the case where every coefficient is ≤ 1, we find the corresponding witnesses w for every detected sum s as described in Chapter 2. Then, we check the witness table W to determine whether we have already computed a witness for the given sum s. If $W[s] = 0$ (the initial value), we set $W[s] = w, F[s] = c$. If $W[s] \neq 0$, then we compare the value of counter c with the one that is stored in $F[s]$. If $F[s] = c$, we proceed as if $W[s] = 0$. Otherwise, we have detected a collision and can reconstruct the sets of our solution from the witness we just computed for s and the stored witness values.

Algorithm 8: $X \odot_t Y$

Input : $X \subseteq \mathcal{S}(Z_1), Y \subseteq \mathcal{S}(Z_2)$, where Z_1, Z_2 are disjoint subsets of initial input set Z, and an upper bound t.
Output: Two disjoint subsets $S_1, S_2 \subseteq Z$ such that $\Sigma(S_1) = \Sigma(S_2)$ and $S_1 \subseteq (Z_1 \cup Z_2) \vee S_2 \subseteq (Z_1 \cup Z_2)$ or (if no two such sets are found) $X \oplus_t Y$.

```
// The witness table W, the flag table F and the counter c are global
   variables defined by the calling procedure
S ← X ⊕_t Y                                          // by using FFT with witnesses
if ∃s : c_s > 1, where c_s is the coefficient of x^s in the FFT result then
    let s* be the minimum such sum
    reconstruct two subsets S_1, S_2 of sum s*          // see Subsection 4.1.4
    return S_1, S_2
else
    foreach s ∈ S do
        w ← the newly computed witness for sum s
        if w ≠ 0 and w ≠ s then                                   // s ∉ X ∪ Y
            if W[s] = 0 or F[s] = c then                          // F, W, c global
                W[s] ← w ; F[s] ← c
            else                                                // see Subsection 4.1.4
                S_1 ← SetConstruction(W, s)
                S_2 ← SetConstruction(W, w) ∪ SetConstruction(W, s − w)
                return S_1, S_2
        end
    end
return S
```

Algorithm 8 presents the process we follow each time we compute a pairwise sum. Note that if a newly computed witness is the same as the sum itself or 0, the corresponding

sum is already present in one of the two initial sets. The complexity of the algorithm is $\tilde{\mathcal{O}}(t)$, since the computation of all pairwise sums up to t along with the witnesses costs $\tilde{\mathcal{O}}(t)$ (Chapter 2). Henceforth, in this section each time we refer to a pairwise sum, we actually refer to Algorithm 8. Therefore, the possible detection of a collision and the reconstruction process are implied each time we make a pairwise sum computation. The correctness of collision detection through the witness table is due to the following lemma which concerns the main algorithm.

Lemma 4.1. *If during the execution of Algorithm 11, $W[s] \neq 0$ for some sum s and $s \in X \oplus Y$ while $s \notin X \cup Y$ and $F[s] \neq c$, then we have a collision.*

The validity of Lemma 4.1 is explained along with the presentation of the modified algorithms.

4.1.1 Small cardinality solutions

We will first describe the procedure `ColorCodingMod` which can successfully solve instances of the Equal Subset Sum problem if the solution size is small, i.e., an algorithm that finds sums $\Sigma(Y_i) \leq t$ generated by sets $Y_i \subseteq Z$ of size at most k and determines if there are two (or more) disjoint subsets with the same sum.

We randomly partition our initial set Z to k^2 subsets $Z_1, \ldots, Z_{k^2}$, i.e., we assign each $z \in Z$ to a set Z_i where i is chosen independently and uniformly at random from $\{1, \ldots, k^2\}$. We say that this random partition *splits* $Y \subseteq Z$ if $|Y \cap Z_i| \leq 1, \forall i$. If such a split occurs, the set returned by `ColorCodingMod` will contain $\Sigma(Y)$. Indeed, by choosing the element of Y for those Z_i that $|Y \cap Z_i| = 1$ and 0 for the rest, we successfully generate $\Sigma(Y)$ through the pairwise sum. The algorithm returns only valid sums, since no element is used more than once in each sum, because each element is assigned uniquely to a Z_i for each distinct partition.

Our intended goal is thus, for any random pair of disjoint subsets, to have a partition that splits them both. Such a partition allows us to detect the collision of the two subsets through the use of FFT. Hence, the question is, how many random partitions are required to achieve this with high probability?

The answer is obtained by observing that the probability to split a subset Y is the same as having $|Y|$ different balls in $|Y|$ distinct bins, when throwing $|Y|$ balls into k^2 different bins, as mentioned in [Brin17]. Next, we compute the probability to split two random disjoint subsets Y_1 and Y_2 at the same partition.

The probability that a split occurs at a random partition for two random disjoint subsets $Y_1, Y_2 \subseteq Z$ is

$$\begin{aligned}
&\Pr[Y_1 \text{ and } Y_2 \text{ are split}] = \Pr[Y_1 \text{ is split}] \cdot \Pr[Y_2 \text{ is split}] = \\
&\frac{k^2-1}{k^2} \cdot \frac{k^2-2}{k^2} \cdots \frac{k^2-(|Y_1|-1)}{k^2} \cdot \frac{k^2-1}{k^2} \cdot \frac{k^2-2}{k^2} \cdots \frac{k^2-(|Y_2|-1)}{k^2} \geq \\
&\left(\frac{k^2-(|Y_1|-1)}{k^2}\right)^{|Y_1|} \cdot \left(\frac{k^2-(|Y_2|-1)}{k^2}\right)^{|Y_2|} \geq \\
&\left(1-\frac{1}{k}\right)^k \cdot \left(1-\frac{1}{k}\right)^k \geq \left(\frac{1}{2}\right)^2 \cdot \left(\frac{1}{2}\right)^2 = \frac{1}{16}
\end{aligned}$$

Hence, $r = \lceil \log_{16/15}(1/\delta) \rceil$ repetitions are sufficient to yield the desired success probability of $1 - (1 - 1/16)^r \geq 1 - \delta$. In other words, after r random partitions, for any two random disjoint subsets Y_1, Y_2, there exists, with probability at least $1 - \delta$, a partition that splits them both.

Due to the multiple random partitions happening at `ColorCodingMod`, it is possible to compute the same sum with the same elements in different random partitions. This happens

Algorithm 9: `ColorCodingMod`(Z, t, k, δ)

Input : A set Z of positive integers, an upper bound t, a size bound $k \geq 1$ and an error probability $\delta > 0$.

Output: Two disjoint subsets of Z of equal sum or (if no two such sets are found) a set $S \subseteq \mathcal{S}_t(Z)$ containing any two $\Sigma(Y_1), \Sigma(Y_2) \leq t$ with probability at least $1 - \delta$, where $Y_1, Y_2 \subseteq Z$ disjoint subsets and $|Y_1|, |Y_2| \leq k$.

`// The witness table` W`, the flag table` F `and the counter` c `are global variables defined by the calling procedure`

1 $c \leftarrow c + 1$
2 $S \leftarrow \emptyset$
3 **for** $j = 1, \ldots, \lceil \log_{16/15}(1/\delta) \rceil$ **do**
4 randomly partition $Z = Z_1 \cup Z_2 \cup \cdots \cup Z_{k^2}$
5 $S_j \leftarrow Z_1 \odot_t \cdots \odot_t Z_{k^2}$
6 $S \leftarrow S \cup S_j$
7 **end**
8 **return** S

because it is possible for a subset to be split in two (or more) different partitions. We have to exclude such a "collision" from our solution, hence apart from the witness of a sum, we keep the associated counter at the flag table. If we have previously computed the same sum s (i.e. $W[s] \neq 0$) and the counter has the same value as the flag table ($F[s] = c$), this means that there has been an earlier random partition (in the same `ColorCodingMod` call) that split a (possibly different) subset with the same sum. We ignore this collision and we just update the corresponding witness at $W[s]$. Essentially, we disregard this collision and assume it is caused by the same subset. This does not affect the correctness of our algorithm, since if we *actually* have a collision, with high probability, at some repetition we will have a split for both the subsets of collision at the same time, hence the collision will be detected through the FFT operation.

If we have computed the same sum s with $F[s] \neq c$, then it means that we have detected a collision among different calls of `ColorCodingMod`. As will become evident later[2], different calls of `ColorCodingMod` process disjoint subsets of our initial input set Z, thus we will have a collision and Lemma 4.1 holds. If this happens, it is possible to reconstruct the sets of our solution, since all the associated witnesses are properly kept in the witness table.

Lemma 4.2. *Given a set Z of positive integers, a sum bound t, a size bound $k \geq 1$, an error probability $\delta > 0$, a counter c, a witness table W and a flag table F, `ColorCodingMod`(Z, t, k, δ) returns two disjoint sets Y_1, Y_2 with $\Sigma(Y_1) = \Sigma(Y_2) \leq t$ and $|Y_1|, |Y_2| \leq k$ (assuming existence), in $\tilde{\mathcal{O}}(tk^2 \log(1/\delta))$ time with probability at least $1 - \delta$. If no two such sets exist, the algorithm returns a set $S \subseteq \mathcal{S}_t(Z)$ containing any two $\Sigma(Y_1), \Sigma(Y_2) \leq t$ with $Y_1, Y_2 \subseteq Z$ disjoint subsets and $|Y_1|, |Y_2| \leq k$, with probability at least $1 - \delta$.*

Complexity. The algorithm performs $\mathcal{O}(\log(1/\delta))$ repetitions. To compute a pairwise sum up to t along with its witnesses, $\tilde{\mathcal{O}}(t)$ time is required. In each repetition, k^2 pairwise sums are computed, along with their associated witnesses. Hence, the total complexity of the algorithm is $\mathcal{O}(k^2 \log(1/\delta)) \cdot \tilde{\mathcal{O}}(t) = \tilde{\mathcal{O}}\left(tk^2 \log(1/\delta)\right)$.

4.1.2 Solving Equal Subset Sum for l-layer instances of Z

In this subsection, we will prove that we can use the algorithm `ColorCodingLayer` as presented in [Brin17], to successfully solve Equal Subset Sum for l-layer instances.

[2] Also refer to page 44 and Figure 3.1.

An instance (Z, t) is called an l-layer instance if either $Z \subseteq [t/l, 2t/l]$ or $Z \subseteq [0, 2t/l]$ and $l \geq n$. In both cases, $Z \subseteq [0, 2t/l]$ and for any $Y \subseteq Z$ with $\Sigma(Y) \leq t$, we have $|Y| \leq l$. The algorithm of [Brin17] successfully solves the Subset Sum problem for l-layer instances. We will show that by calling the algorithm `ColorCodingMod` and, along with a pairwise sum we compute its witnesses and make the appropriate checks, the algorithm successfully solves Equal Subset Sum.

Algorithm 10: `ColorCodingLayerMod`(Z, t, l, δ)

Input : An *l-layer* instance (Z, t) and an error probability $\delta \in (0, 1/8]$.
Output: Two disjoint subsets of Z of equal sum or (if no two such sets are found) a set $S \subseteq \mathcal{S}_t(Z)$ containing any two $\Sigma(Y_1), \Sigma(Y_2) \leq t$ with probability at least $1 - \delta$, where $Y_1, Y_2 \subseteq Z$ disjoint subsets.

`// The witness table` W`, the flag table` F `and the counter` c `are global variables defined by the calling procedure`

1 **if** $l < \log(l/\delta)$ **then return** `ColorCodingMod`(Z, t, l, δ)
2 $m \leftarrow l/\log(l/\delta)$ rounded up to the next power of 2
3 randomly partition $Z = Z_1 \cup Z_2 \cup \cdots \cup Z_m$
4 $\gamma \leftarrow 6\log(l/\delta)$
5 **for** $j = 1, \ldots, m$ **do**
6 $\quad S_j \leftarrow$ `ColorCodingMod`$(Z_j, 2\gamma t/l, \gamma, \delta/l)$
7 **end**
8 **for** $h = 1, \ldots, \log m$ **do**
9 $\quad$ **for** $j = 1, \ldots, m/2^h$ **do**
10 $\quad\quad c \leftarrow c + 1$
11 $\quad\quad S_j \leftarrow S_{2j-1} \odot_{2^h \cdot 2\gamma t/l} S_{2j}$
12 $\quad$ **end**
13 **end**
14 **return** $S_1 \cap [t]$

Firstly, we observe that Lemma 4.1 holds. Indeed, as it will become evident later, different calls of `ColorCodingLayerMod` process disjoint subsets of the ground set Z and each such call increases the value of c. So, if we have a collision among different calls of `ColorCodingLayerMod` it will be successfully detected (as the condition in line 10 of Algorithm 8 will be falsified). Additionally, each call to `ColorCodingMod` increases c and processes a subset of Z that is disjoint from subsets processed by any other `ColorCodingMod` call (no matter if it is in the same or in different `ColorCodingLayerMod` calls). Lastly, we increase the value of c at line 10 in the modified algorithm in order to detect collisions among S_i's. For example, it is possible to compute the same sum by $S_1 \oplus S_2$ and $S_{m-1} \oplus S_m$. Without increasing the counter c, we cannot detect such a collision. Instead, we would have to wait until the corresponding FFT operation is realised, which would result in a coefficient greater than 1, but the reconstruction of the solution sets in that case would not be possible, since in such a setting, the sums in the witness table W would not be computed in a unique way.

Next, we will prove the correctness of the algorithm. Suppose that $X, Y \subseteq Z$, and $\Sigma(X) = \Sigma(Y) \leq t$, with X, Y being disjoint sets. By Claim 3.2 as presented on page 45, we have that $\Pr[|Y_i| \geq 6\log(l/\delta)] \leq \delta/l$, where $Y_i = Y \cap Z_i$, for any $Y \subseteq Z$ with at most l elements. Hence, the probability that $|X_i| \leq 6\log(l/\delta)$ and $|Y_i| \leq 6\log(l/\delta)$ for all

$i = 1, \ldots, m$ is

$$\Pr[\bigwedge_{i=1}^{m}(|X_i| \leq 6\log(l/\delta) \wedge |Y_i| \leq 6\log(l/\delta))] \geq$$
$$1 - \sum_{i=1}^{m}(\Pr[|X_i| > 6\log(l/\delta)]) - \sum_{i=1}^{m}(\Pr[|Y_i| > 6\log(l/\delta)]) \geq 1 - 2m\delta/l$$

`ColorCodingMod` computes $\Sigma(X_i)$ and $\Sigma(Y_i)$ with probability at least $1-\delta$. This happens for all i with probability at least $1-m\delta/l$. Then, combining the resulting sets indeed yields both $\Sigma(X)$ and $\Sigma(Y)$ and detects the collision by checking the coefficients of FFT. The total error probability is at most $3m\delta/l$. Assume that $\delta \in (0, 1/8]$. Since $l \geq 1$ and $\delta \leq 1/8$, we have $\log(l/\delta) \geq 3$. Hence, the total error probability is bounded by δ. This gives the following.

Lemma 4.3. *Given an l-layer instance (Z, t), error probability $\delta \in (0, 1/8]$, a counter c, a witness table W and a flag table F, `ColorCodingLayerMod`(Z, t, l, δ) returns two disjoint sets Y_1, Y_2 with $\Sigma(Y_1) = \Sigma(Y_2) \leq t$ (assuming existence), in $\tilde{\mathcal{O}}(t\log^3(l/\delta)$ time with probability at least $1-\delta$. If no two such sets exist, the algorithm returns a set $S \subseteq \mathcal{S}_t(Z)$ containing any two $\Sigma(Y_1), \Sigma(Y_2) \leq t$ with $Y_1, Y_2 \subseteq Z$ disjoint subsets, with probability at least $1-\delta$.*

Complexity. Each call to `ColorCodingMod` takes time $\mathcal{O}(\log^3(l/\delta)) \cdot \tilde{\mathcal{O}}(\gamma t/l)$. There are $m = \Theta(l/\log(l/\delta))$ calls to `ColorCodingMod`, hence $\mathcal{O}(l\log^2(l/\delta)) \cdot \tilde{\mathcal{O}}(\gamma t/l) = \tilde{\mathcal{O}}(t\log^3(l/\delta))$ total time. The combination of the resulting sets takes time

$$\tilde{\mathcal{O}}\left(\sum_{h=1}^{\log m}\frac{m}{2^h}\cdot 2^h\log(l/\delta)t/l\right) = \tilde{\mathcal{O}}(t\log m) = \tilde{\mathcal{O}}\left(t\log\left(\frac{l}{\log(l/\delta)}\right)\right).$$

Hence, the algorithm has total running time $\tilde{\mathcal{O}}(t\log^3(l/\delta))$.

4.1.3 General Case

It remains to show that for every Subset Sum instance (Z, t), we can construct l-layer instances and take advantage of `ColorCodingLayerMod` to solve Equal Subset Sum in pseudolinear time (with regard to t) for the general case. This is possible by partitioning set Z at $t/2^i$ for $i = 1, \ldots, \lceil\log n\rceil - 1$. Thus, we have $\mathcal{O}(\log n)$ l-layers $Z_1, \ldots, Z_{\lceil\log n\rceil}$. On each layer we run `ColorCodingLayerMod`, and then we combine the results using pairwise sums. Lemma 4.1 holds since the different calls to `ColorCodingLayerMod` process disjoint subsets.

Theorem 4.4. *Given a set $Z \subseteq \mathbb{N}$ of size n, an error margin δ and an upper bound t, Algorithm `EqualSubsetSum`(Z, δ, t) solves* Equal Subset Sum *(that is, it outputs two disjoint sets $S_1, S_2 \subseteq Z$, with $\Sigma(S_1) = \Sigma(S_2) \leq t$, if such sets exist), in $\tilde{\mathcal{O}}(n+t)$ time, with probability at least $1-\delta$.*

Proof. Suppose there exist two disjoint subsets $X, Y \subseteq Z$ with $\Sigma(X) = \Sigma(Y) \leq t$, and $X_i = X \cap Z_i, \quad Y_i = Y \cap Z_i$. Each call to `ColorCodingLayerMod` returns $\Sigma(X_i)$ and $\Sigma(Y_i)$ with probability at least $1 - \delta/\lceil\log n\rceil$, hence the probability that all calls return $\Sigma(X_i)$ and $\Sigma(Y_i)$ is

$$\Pr[\texttt{ColorCodingLayerMod returns both } \Sigma(X_i), \Sigma(Y_i), \forall i] = 1 - \Pr[\text{some call fails}] \geq$$
$$1 - \sum_{i=1}^{\lceil\log n\rceil}\frac{\delta}{\lceil\log n\rceil} = 1 - \lceil\log n\rceil\cdot\frac{\delta}{\lceil\log n\rceil} = 1-\delta$$

If all calls return the corresponding sums, the algorithm successfully detects the collision and reconstructs the subsets. Thus, with probability at least $1-\delta$, the algorithm solves Equal Subset Sum. □

Algorithm 11: `EqualSubsetSum`(Z, δ, t)

Input : A set of positive integers Z, an upper bound t and an error probability δ.
Output: Two disjoint subsets of Z of equal sum or (if no two such sets are found) a set $S \subseteq \mathcal{S}_t(Z)$ containing any two $\Sigma(Y_1), \Sigma(Y_2) \leq t$ with $Y_1, Y_2 \subseteq Z$ disjoint subsets, with probability at least $1 - \delta$.

$c \leftarrow 0$
for $i = 1, \ldots, t$ **do** $W[i] \leftarrow 0$; $F[i] \leftarrow 0$
foreach $z_i \in Z$ **do** $W[z_i] \leftarrow z_i$
partition Z into $Z_i \leftarrow Z \cap (t/2^i, t/2^{i-1}]$ for $i = 1, \ldots, \lceil \log n \rceil - 1$ and $Z_{\lceil \log n \rceil} \leftarrow Z \cap [0, t/2^{\lceil \log n \rceil - 1}]$
$S \leftarrow \emptyset$
for $i = 1, \ldots, \lceil \log n \rceil$ **do**
  $S_i \leftarrow$ `ColorCodingLayerMod`$(Z_i, t, 2^i, \delta/\lceil \log n \rceil)$
  $S \leftarrow S \odot_t S_i$
end
return S

Complexity. Reading the input requires $\Theta(n)$ time. The initialisation of the witness and flag tables requires $\Theta(t)$ time. The algorithm has $\Theta(\log n)$ repetitions, and in each repetition we make a call to `ColorCodingLayerMod`, plus compute a pairwise sum. The computation of the pairwise sum requires $\tilde{\mathcal{O}}(t)$ time (along with the witnesses). For each call to algorithm `ColorCodingLayerMod`, we require $\tilde{\mathcal{O}}(t \log^3(2^i/\delta))$ time. Hence the total complexity is $\Theta(n) + \Theta(t) + \sum_{i=1}^{\log n} \tilde{\mathcal{O}}(t \log^3(2^i/\delta)) = \tilde{\mathcal{O}}(n + t \log^3(n/\delta) \log n) = \tilde{\mathcal{O}}(n + t)$.

4.1.4 Reconstruction of the solution sets

The decision version of Equal Subset Sum actually does not require the use of the counter c or the tables W and F. It suffices to modify the number of repetitions at Algorithm `ColorCoding` and check the coefficients of FFT every time we compute a pairwise sum. In this section, we will show that using counter c and tables W, F is essential in order to efficiently compute the solution sets.

A first approach, that does not make use of the counter or a universal witness table, would be to store alongside each sum its corresponding elements, i.e. the elements of the subset of Z that has that subset sum. This solution would require for each detected sum to store up to n elements. We also note that some sums may be detected multiple times. It is evident that such a solution is particularly costly as far as space is concerned.

We now show that given a witness table as described above, there is an efficient way to find the elements that sum up to a specific sum t. We call this algorithm `SetConstruction`.

Algorithm 12: `SetConstruction`(W, t)

Input : A witness table W and a target t.
Output: A set of numbers that sum up to t.

if $t = 0$ **then return** $\emptyset$
if $W[t] = t$ **then return** t
return `SetConstruction`$(W, W[t]) \cup$ `SetConstruction`$(W, t - W[t])$

The correctness of the algorithm stems from the fact that there is no single element that is used multiple times, i.e. `SetConstruction`$(W, W[t]) \cap$ `SetConstruction`$(W, t - W[t]) = \emptyset$. This is a crucial property that is actually the reason a counter or a slight modification is necessary. The complexity of the algorithm is $\mathcal{O}(n)$.

The following section introduces an Equal Subset Sum algorithm without any additional space requirements apart from table W. The problem with this approach is that it makes use of existing Subset Sum algorithms in order to reconstruct the solution sets, hence possibly affecting the complexity of the algorithm (or requiring additional probability guarantees). In order to avoid the use of a Subset Sum algorithm, it is necessary to detect a collision as soon as it happens. By the use of the counter and the flag table, we manage to ensure that throughout the execution of our algorithm, each sum is computed in a *unique* way, thus it is possible to reconstruct it via the use of the witness table.

If a collision is detected through the use of counter for sum s, then it is possible to efficiently compute the solution sets Sol_1 and Sol_2:

- By calling `SetConstruction`(W, s), we can reconstruct Sol_1.
- By calling `SetConstruction`$(W, s') \cup$ `SetConstruction`$(W, s - s')$, where s' is the last computed witness for sum s, we can reconstruct Sol_2.

If we detect the collision through the coefficients of FFT (which is done before checking the witnesses of the sums), then it is also possible to efficiently compute the solution sets. Suppose that the algorithm detects a collision at $X \oplus Y$ and $x_1 + y_1 = x_2 + y_2 = s$, where $x_1, x_2 \in X, y_1, y_2 \in Y$. The sum s is also the minimum sum for which we have detected a collision through FFT. We also have that $x_1 \neq x_2$ and $y_1 \neq y_2$ since otherwise a collision would have been detected at a previous pairwise sum. It also holds that at most one among x_1, x_2, y_1, y_2 is equal to 0 (if not, suppose that $x_1 = y_2 = 0$, then $x_2 = y_1 = s$, but then the collision would have already been detected through the witness table check). We also have that those four partial sums are all distinct amongst themselves, since if for example $x_1 = y_1$, this collision would have been already detected through the witness table check. The counter guarantees that each sum is computed in a unique way, and by the structure of the algorithm, that no single element retrieved from the witness table is concurrently used for a sum in both X and Y. Hence, for the reconstruction of the solution sets, it suffices to find the partial sums x_1, x_2, y_1 and y_2 and then construct the solution sets by the use of `SetConstruction`. When we find a pair of witnesses for the sum, we actually retrieve one of the solution sets, for example the one composed of $x_1 + y_1$. For the other solution set, it suffices to compute $(X \setminus \{x_1, y_1\}) \oplus_s Y$ and find the other set of witnesses for sum s. This can be done in $\tilde{\mathcal{O}}(s) = \tilde{\mathcal{O}}(t)$ so it does not affect the complexity of the algorithm.

4.1.5 An algorithm without counter and flag table.

Firstly, we will justify the need for some minor modifications, in order to completely avoid the use of the counter and the flag table. This modified version of the algorithm is worse, in the sense that the reconstruction of one of the two solution sets requires to run an algorithm for Subset Sum, thus possibly increasing the complexity of our algorithm.

Assume that for each possible sum we keep only its partial sum and not any kind of flag. As shown before, due to the random partitions happening at `ColorCoding`, the detection of the collisions is now done *exclusively* through the coefficients of FFT. Hence, the witness table actually holds one of the partial sums *last used* to compute the associated sum. Suppose that a collision occurs at some point of the execution of our algorithm, i.e. $X \oplus_t Y$ has a coefficient greater than 1 at FFT. Then, there exist $x_1, x_2 \in X$ and $y_1, y_2 \in Y$ with $x_1 + y_1 = x_2 + y_2 = s$. We also have that $x_1 \neq x_2$ and $y_1 \neq y_2$, since if otherwise, a collision would have been detected earlier in the execution of our algorithm. Suppose that x_1, x_2, y_1 and y_2 are known. The next logical step of our set construction would be to find the elements used to compute those sums. This would be done through the use of the universal witness table. More specifically, the elements used to compute x_1 and y_1 constitute the first set Sol_1 of our solution and the elements used to compute x_2 and y_2 constitute the second set Sol_2. Let $U_{x_1}, U_{x_2}, U_{y_1}, U_{y_2}$ be

the set of elements of our initial set Z used to sum up to x_1, x_2, y_1, y_2 respectively. Then, following the approach described above, we have that $Sol_1 = U_{x_1} \cup U_{y_1}$ and $Sol_2 = U_{x_2} \cup U_{y_2}$. The problem is that we can not ensure that $U_{x_1} \cap U_{y_1} = \emptyset$ and $U_{x_2} \cap U_{y_2} = \emptyset$. In other words, by following the values saved at the witness table, it is possible for an element to be used more than once. For example, at `ColorCodingLayerMod` line 6, suppose that $X = S_1, Y = S_2$ and also $x_1, y_1 \in S_m$. The computation of S_m is done last, so $W[x_1]$ and $W[y_1]$ will have the partial sums used there to compute them. That means that it is possible for an element to be added in both the addition for x_1 and also the addition for y_1, resulting in a faulty construction of the solution set, when the collision is detected at $S_1 \oplus S_2$.

A way to counter this problem is by modifying the order of the pairwise sums and the computation of S_i at `ColorCodingLayerMod`:

1. We firstly compute S_1 and S_2.
2. Then we compute $S_{12} = S_1 \oplus_{2 \cdot 2\gamma t/l} S_2$.
3. Then we compute S_3 and S_4.
4. Then we compute $S_{34} = S_3 \oplus_{2 \cdot 2\gamma t/l} S_4$.
5. Then we compute $S_{1234} = S_{12} \oplus_{4 \cdot 2\gamma t/l} S_{34}$.
6. We continue accordingly.

By making this modification, we firstly note that we have not changed the complexity of the algorithm so far. We will now show that this algorithm can successfully reconstruct the solution sets.

Suppose that the algorithm detects a collision at $X \oplus Y$ and $x_1 + y_1 = x_2 + y_2 = s$, where $x_1, x_2 \in X, y_1, y_2 \in Y$. The sum s is also the minimum sum for which we have detected a collision through FFT. We also have that $x_1 \neq x_2$ and $y_1 \neq y_2$ since otherwise a collision would have been detected at a previous pairwise sum. Now there are three distinct cases :

1. All $x_1, x_2, y_1, y_2 \neq 0$.
2. One of x_1, x_2, y_1, y_2 is 0.
3. $x_1 = y_2 = 0$ or $x_2 = y_1 = 0$.

For cases 1 and 2, it holds that $x_1 \neq y_1, y_2$ and $x_2 \neq y_1, y_2$ since otherwise s would not have been minimum. We claim that the witness table for y_1 and y_2 reconstructs them the way they are computed in Y. This means that only elements present in Y are used. We also claim that the witness table for x_1 and x_2 reconstructs them the way they are computed in X, since they are not computed again in Y, hence the value at the witness table remains unchanged. Thus, we have 4 distinct values, and we can efficiently compute the solution sets the same way we do as in the case we have a counter and a flag table and we detect the collision through the FFT.

For case number 3, it means that $s \in X \cap Y$. The reconstructed solution set Sol_1 provides us with the elements of Y that add up to s and can be efficiently computed by `SetConstruction`. The problem is that we have no knowledge about how sum s came to be at set X, since the witness table has information about the *last* way a sum was computed. Our only option is to run an algorithm for Subset Sum with target s on the set $Z \setminus Sol_1$. This results in a potential increase of the complexity of the algorithm. By introducing the use of the counter and the flag table, we can avoid this.

4.2 A deterministic $\tilde{\mathcal{O}}(\sigma)$ algorithm

As we have already presented in Section 3.2, Koiliaris and Xu have shown in [Koil19] that one can compute all subset sums of a set Z in $\mathcal{O}(\sigma \log \sigma \log n)$, where $\sigma = \Sigma(Z)$ and $n = |Z|$.

We remind the reader that this algorithm is incomparable to the rest of Subset Sum algorithms mentioned, since we cannot efficiently solve bounded instances by using it. That is, for a given upper bound t, its recursion tree yields $\mathcal{O}(t \log t \sum_{i=0}^{\log n} 2^i) = \mathcal{O}(nt \log t)$ complexity[3]. By slightly modifying this algorithm, one can obtain the same results for Equal Subset Sum.

Algorithm 13: `ESSNoBound(Z)`

Input : A set Z of n positive integers.
Output: Two disjoint subsets $S_1, S_2 \subseteq Z$, where $\Sigma(S_1) = \Sigma(S_2)$ or (if no two such sets exist) $\mathcal{S}(Z)$.

`// We assume the witness matrix` W `is global and initialised to 0`
if $Z = \{z\}$ **then**
  if $W[z] = 0$ **then**
    $W[z] \leftarrow z$
    return Z
  else `// see Subsection 4.1.4`
    $S_1 \leftarrow \{z\}$
    $S_2 \leftarrow$ `SetConstruction`(W, z)
    return S_1, S_2
end
$L \leftarrow$ an arbitrary subset of Z of size $\lfloor \frac{n}{2} \rfloor$
$R \leftarrow Z \setminus L$
$\sigma \leftarrow \Sigma(Z)$
$Z_1 \leftarrow$ `ESSNoBound`(L) ; $Z_2 \leftarrow$ `ESSNoBound`(R)
$S \leftarrow Z_1 \oplus_\sigma Z_2$ `// by using FFT, also compute the witnesses in` $\tilde{\mathcal{O}}(\sigma)$
if $\exists s : c_s > 1$, *where* c_s *is the coefficient of* x^s *in the FFT* **then**
  let s^* be the minimum such sum
  reconstruct two subsets S_1, S_2 of sum s^* `// see Subsection 4.1.4`
  return S_1, S_2
else
  foreach $s \in S$ **do**
    $w \leftarrow$ the newly computed witness for sum s
    if $w \neq 0$ *and* $w \neq s$ **then** `//` $s \notin Z_1 \cup Z_2$
      if $W[s] = 0$ **then**
        $W[s] \leftarrow w$
      else `// see Subsection 4.1.4`
        $S_1 \leftarrow$ `SetConstruction`(W, s)
        $S_2 \leftarrow$ `SetConstruction`$(W, w) \cup$ `SetConstruction`$(W, s - w)$
        return S_1, S_2
    end
  end
return S

Theorem 4.5. *Given a set $Z \subseteq \mathbb{N}$ of size n, one can construct two disjoint sets $S_1, S_2 \subseteq Z$, where $\Sigma(S_1) = \Sigma(S_2)$ (assuming existence), thereby solving* Equal Subset Sum, *in $\tilde{\mathcal{O}}(\sigma)$ deterministic time, where $\sigma = \Sigma(Z)$.*

[3] Refer to Lemma 2.1 and Figure 2.1 on page 41.

Algorithm 13 presents the pseudocode for the modified algorithm. We largely follow the techniques developed in Section 4.1. Namely, we introduce a witness table W which guarantees the uniqueness of each sum and we update W each time a pairwise sum operation occurs. The reconstruction phase is exactly the same as in the case of Section 4.1.

The correctness of the algorithm stems directly from the fact that, at any point of execution of the algorithm, each sum is uniquely computed (i.e., there is only one way up to this point to construct each distinct sum). As soon as a sum is computed via two distinct ways, the algorithm terminates, reconstructing the minimum such sum. Due to the fact that it is the minimum, it is formulated from disjoint sets, since if otherwise, we could remove the common elements, hence attaining a lesser sum formulated with two ways.

The complexity of the algorithm is $\mathcal{O}(\log n) \cdot \tilde{\mathcal{O}}(\sigma) = \tilde{\mathcal{O}}(\sigma)$, because instead of a simple FFT, here at the 'conquer' step of the algorithm we also compute the witnesses and check for collisions through the witness table. This can be done with polylogarithmic overhead, hence costs $\tilde{\mathcal{O}}(u)$ for a pairwise sum up to u and by Lemma 2.1 we obtain the complexity of the algorithm.

4.3 A deterministic $\tilde{\mathcal{O}}(\sqrt{nt})$ algorithm for Equal Subset Sum

In this section, we show that Equal Subset Sum is solvable in $\tilde{\mathcal{O}}(\sqrt{nt})$. Specifically, one can use a slight variation of the algorithm from Section 3.3 in order to reduce Equal Subset Sum to Subset Sum.

In this modified algorithm, instead of translating each polynomial back into a set of sums, we keep the representation by polynomials throughout the whole recursion. In order to improve the complexity, in [Koil19] they initially divide the input numbers by b and then restrict the operations to elements whose original values belong to the same congruence class (mod b), where $b = \lfloor \sqrt{n \log n} \rfloor$, thus reducing the running time of each FFT operation. Subsequently, they recover the correct sums by adding to each sum the respective residue multiplied by the cardinality of the subset involved. Finally, they once again use FFT to combine these sums to find sums that span different congruence classes. This outputs a list of all realisable sums up to t in time $\tilde{\mathcal{O}}(\sqrt{nt})$. Observe that all these operations can be done using the polynomial representation without slowing down the algorithm.

Notice that:

- At the end of the recursion, we end up with a polynomial that contains each realisable sum represented by the exponent of a term of the polynomial.
- The term coefficients represent the number of different (not necessarily disjoint) subsets that sum up to the corresponding sum.
- If a sum can be made up from two (or more) different non-disjoint subsets, then there exists a smaller sum that can be made from two (or more) disjoint sets (by removing the common elements).

Therefore, the term with the smallest exponent s that has coefficient ≥ 2 corresponds to a sum that can be made up from two (or more) disjoint sets. Note that if we are interested only in the sum that can be made up of (at least) two disjoint subsets, we can stop at this point and output s, thus solving the *decision version* of Equal Subset Sum without any modification.

Reconstruction. In the case of this modified algorithm, we can efficiently recover the solution sets (given s) by running a subset sum algorithm with target s once, delete the

returned subset from the initial set, and run subset sum once again with target s. This can be seen as an $\tilde{\mathcal{O}}(\sqrt{n}t)$ reduction to subset sum, thereby providing an argument that these these two problems are of similar complexity. Koiliaris and Xu argue in [Koil19, Section 6] that one can indeed return the solution set of their proposed Subset Sum algorithms with only polylogarithmic overhead. Thus, the overall complexity of the algorithm remains $\tilde{\mathcal{O}}(\sqrt{n}t)$.

4.4 A deterministic $\tilde{\mathcal{O}}(n+t)$ algorithm for Equal Subset Sum

In this section, we will present a simple extension of Bellman's algorithm to cope with the Equal Subset Sum problem. This efficient extension successfully reconstructs the solution sets in pseudopolynomial $\tilde{\mathcal{O}}(n+t)$ deterministic time, hence is preferred over the rest of the presented Equal Subset Sum algorithm of this chapter. Notice however, that while this approach is the most efficient for the Equal Subset Sum problem, it cannot be extended to efficiently solve the k-Subset Sum problem, whereas algorithms based on those presented on Sections 3.1 and 3.3 can. That is due to the fact that it heavily relies on the uniqueness of the formulation of each sum, a property that does not hold for k-Subset Sum.

This algorithm was developed based on feedback from anonymous reviewers, and can be seen as a simple extension of Bellman's dynamic programming approach, as presented in [Bell57].

Firstly, we note that table W has two distinct uses:

1. It is used to verify that each sum s is obtained in a unique way. As soon as a sum s can be obtained with two different ways, the algorithm formulates the solution sets and returns them, since $W[s] \neq 0$.

2. Additionally, it is used to keep the last added element of the corresponding sum, in order to recursively reconstruct the solution sets.

Thus, we can infer that we will access table W at most t times, since if otherwise, this would imply a collision has occurred.

If T is a self-balancing binary search tree (e.g. AVL tree) and $|T_i|$ denotes the number of elements in T at repetition i, then we can find the elements $e \in T$, such that $e+z_i \leq t$, in time $\mathcal{O}(\log|T_i|)$. Subsequently, $\mathcal{O}(|L_i|)$ time is needed to construct list L_i and $\mathcal{O}(|L_i|\log|T_{i+1}|)$ to insert its elements in T, where $|L_i|$ denotes the number of elements of list L_i. Also notice that $|T_i| = \sum|L_{i'}|$, for $i' < i$ and the size of tree T is at most t, since there are at most t distinct sums, thus $\sum|L_i| \leq t$. Thus, the total complexity of the algorithm is

$$\mathcal{O}\left(n + \sum_{i=1}^{n}|L_i|\log|T_{i+1}|\right) = \mathcal{O}\left(n + \sum_{i=1}^{n}|L_i|\log t\right) = \mathcal{O}(n + t\log t) = \tilde{\mathcal{O}}(n+t).$$

Algorithm 14: `Simple(Z, t)`

Input : A set $Z = \{z_1, \ldots, z_n\}$ of n positive integers and an upper bound integer t.
Output: Two disjoint subsets $S_1, S_2 \subseteq Z$, where $\Sigma(S_1) = \Sigma(S_2) \leq t$ or (if no two such sets exist) false.

initialise table $W[0 \ldots t] \leftarrow 0$
tree T consisting only of sum 0 `// T consists of all the realisable sums`
for $i = 1, \ldots, n$ **do**
  $L_i = [\,]$ `// List` L_i `consists of all the newly formed realisable sums`
  foreach $e \in T$ *such that* $e + z_i \leq t$ **do**
    $s \leftarrow e + z_i$
    if $W[s] = 0$ **then** `// first appearance of` s
      $W[s] \leftarrow z_i$
      add s to L_i `//` s `added to list` L_i
    else `//` s `has already appeared`
      $s_1 \leftarrow s$
      $S_1 \leftarrow \emptyset$
      while $s_1 \neq 0$ **do** `// reconstruction process for` S_1
        $S_1 \leftarrow S_1 \cup W[s_1]$
        $s_1 \leftarrow s_1 - W[s_1]$
      end
      $S_2 \leftarrow \{z_i\}$
      $s_2 \leftarrow e$
      while $s_2 \neq 0$ **do** `// reconstruction process for` S_2
        $S_2 \leftarrow S_2 \cup W[s_2]$
        $s_2 \leftarrow s_2 - W[s_2]$
      end
      return S_1, S_2
  end
  foreach $l \in L_i$ **do**
    add l to T `// insert newly formed sums to` T
  end
end
return false

Chapter 5

k-Subset Sum and Further Applications

In this section, we propose extensions of the algorithms of Bringmann [Brin17] and Koiliaris and Xu [Koil19], as presented in Chapter 3, and more specifically Sections 3.1 and 3.3, in order to solve k-Subset Sum which asks, given a set Z of n positive integers and k targets $t_1, \ldots, t_k > 0$, to determine whether there exist k disjoint subsets $Z_1, \ldots, Z_k \subseteq Z$, such that $\Sigma(Z_i) = t_i$, for $i = 1, \ldots, k$. One can see that even in the case of $k = 2$ and $t_1 = t_2$, the problem becomes much harder; this can be seen as a targeted version of Equal Subset Sum, for which the disjointness of minimal sets property does not apply. We also show how there ideas influence a multitude of subset problems. For the rest of this chapter, assume that $Z = \{z_1, \ldots, z_n\}$ is the input set, $t_1, \ldots, t_k$ are the given targets and $t = \max\{t_1, \ldots, t_k\}$. Additionally, the number of subsets k is considered a constant and not part of the input.

This problem differs substantially from Equal Subset Sum, as we cannot assume that the existence of subsets summing up to the target numbers (or any other pair of numbers) implies the existence of corresponding solution(s) consisting of disjoint subsets. In particular, for Equal Subset Sum a minimality property holds: for the minimal sum that can be obtained by multiple subsets, these subsets are necessarily disjoint; here, no analogous property holds. Therefore, we need to take special care in order to verify the disjointness property throughout the whole process.

Note that one can trivially extend Bellman's classic dynamic programming algorithm for Subset Sum to solve this problem in $\mathcal{O}(nt^k)$ time. The extended algorithm for 2-Subset Sum is presented below and can be simply modified to cope with the general case. Suppose that there exist disjoint subsets $Z_1, Z_2 \subseteq Z$, where $\Sigma(Z_1) = t_1$ and $\Sigma(Z_2) = t_2$. For each element z_i of the input set Z, there are three distinct possibilities: a)$z_i \in Z_1$, b)$z_i \in Z_2$, c)$z_i \notin Z_1 \cup Z_2$. The algorithm simply iterates through all the possible disjoint subset sum combinations composed of elements $z_{i'}$, where $i' \leq i$, until $i' = n$, hence successfully solving the problem.

In the following sections, we will use the k-modified characteristic polynomial of a set of integers, as described in Chapter 2. Thus, given a set of integers $Z = \{z_1, \ldots, z_n\}$, the k-modified characteristic polynomial is defined to be

$$f_Z^k(\vec{x}) = \sum_{z \in Z} \sum_{i=1}^{k} x_i^z = x_1^{z_1} + \ldots + x_k^{z_1} + \ldots + x_k^{z_k},$$

where $\vec{x} = (x_1, \ldots, x_k)$.

Intuitively, the goal is to obtain sums in k different dimensions, represented by x_i, by multiplying such polynomials. Then, each term $x_1^{s_1} \ldots x_k^{s_k}$ obtained will correspond to k disjoint subsets, each summing up to s_i. Each $x_j^{z_i}$ used to produce a term consisting of x_j^s translates to $z_i \in S_j$.

Additionally, we will use the following lemma, which stems directly from the preliminaries, and the corollary of Lemmas 2.2 and 2.3.

Lemma 5.1. *Given two sets of points $S, T \subseteq [t]^k$ one can use multidimensional FFT to compute the set of pairwise sums $S \oplus_t T$ in time $\mathcal{O}(t^k \log t)$.*

Algorithm 15: `Bellman(Z, t1, t2)`

Input : A set $Z = \{z_1, \ldots, z_n\}$ of positive integers and targets $t_1, t_2 \leq \Sigma(Z)$.
Output: True if there are disjoint subsets $S_1, S_2 \subseteq Z$, such that $\Sigma(S_1) = t_1$ and $\Sigma(S_2) = t_2$, else false.

```
1  t ← max{t1, t2}
2  initialise table T[n][t][t] ← false everywhere
3  T[0][0][0] ← true                    // Initially, only (Σ(∅), Σ(∅)) is valid.
4  for k = 1, ..., n do
5  |  foreach (i, j) ∈ [t] × [t] do
6  |  |  if T[k − 1][i][j] = true then
7  |  |  |  T[k][i][j] = true                          //  z_k ∉ S1 ∪ S2
8  |  |  |  T[k][i + z_i][j] = true                    //  z_k ∈ S1
9  |  |  |  T[k][i][j + z_i] = true                    //  z_k ∈ S2
10 |  |  end
11 |  end
12 end
13 return T[n][t1][t2]
```

Lastly, notice that, in the following proposed algorithms for k-Subset Sum, each combination of valid sums appears $k!$ times. This means that for every k disjoint subsets $S_1, \ldots, S_k$ of the input set, there are $k!$ different terms in the resulting polynomial of the algorithm representing the combination of sums $\Sigma(S_1), \ldots, \Sigma(S_k)$. This massive increase on the number of terms does not influence the asymptotic analysis of our algorithms, nevertheless can be restricted for better performance. Additionally, one can limit the FFT operations to different bounds for each variable, resulting in slightly improved complexity analysis without changing the algorithms whatsoever. In this thesis, we preferred to analyse the complexity of the algorithms using $t = \max\{t_1, \ldots, t_k\}$ for the sake of simplicity, but one can alternatively obtain time complexities of $\tilde{\mathcal{O}}(n + T)$ and $\tilde{\mathcal{O}}(n^{k/(k+1)}T)$ for the randomised and the deterministic algorithm respectively, where $T = \prod t_i$.

5.1 Solving k-Subset Sum in randomised $\tilde{\mathcal{O}}(n + t^k)$ time

Theorem 5.2. *Given a set $Z \subseteq \mathbb{N}$ of size n and targets $t_1, \ldots, t_k$, one can decide the problem k-Subset Sum in $\tilde{\mathcal{O}}(n + t^k)$ time, where $t = \max\{t_1, \ldots, t_k\}$.*

We will show this by extending the techniques used in [Brin17], as presented in 3.1. The presented algorithm successfully detects, with high probability, whether there exist k disjoint subsets each summing up to t_i respectively. In comparison to the algorithm of [Brin17], a couple of modifications are required, which we will first briefly summarise prior to presenting the complete algorithm.

- At algorithm `ColorCoding`, the number of repetitions is increased, without however asymptotically affecting the complexity of the algorithm, since it remains $\mathcal{O}(\log(1/\delta))$.
- At algorithm `ColorCoding`, after each partition of the elements, we execute the FFT operations on the k-modified characteristic polynomials of the resulting sets. Thus, for each element s_i we introduce k points $(s_i, 0, \ldots, 0), \ldots, (0, \ldots, s_i)$, represented by polynomial $x_1^{s_i} + \ldots + x_k^{s_i}$. Hence `ColorCoding` returns a set of points, each of which corresponds to k sums, realisable by disjoint subsets of the initial set.
- Algorithm `ColorCodingLayer` needs no modification. Note that the FFT operations concern sets of points and not sets of integers, hence the complexity analysis differs.

Additionally, the algorithm returns a set of points, each of which corresponds to some realisable combination of sums of disjoint subsets of the l-layer input instance.

- The initial partition of the original set to l-layer instances remains as is, and the FFT operations once more concern sets of points instead of sets of integers.

As was the case for the original algorithm of Bringmann, we will firstly present an algorithm for small cardinality solutions, subsequently one for l-layer instances and lastly a general case solution. We refer the reader to page 44, where a relevant diagram regarding the partition of the input set for the general case is presented.

5.1.1 Small cardinality solutions

We will first describe the modified procedure `ColorCoding` for solving k-Subset Sum if the solution size is small, i.e., an algorithm that finds k-tuples of sums $(\Sigma(Y_1), \ldots, \Sigma(Y_k))$, where $\Sigma(Y_i) \leq t$ and $Y_i \subseteq Z$ are disjoint subsets of input set Z of cardinality at most c, and determines whether there exists a tuple $(t_1, \ldots, t_k)$, for some given values t_i.

As was the case for the original algorithm proposed by Bringmann, we randomly partition our initial set Z to c^2 subsets $Z_1, \ldots, Z_{c^2}$, i.e., we assign each $z \in Z$ to a set Z_i where i is chosen independently and uniformly at random from $\{1, \ldots, c^2\}$. We say that this random partition *splits* $Y \subseteq Z$ if $|Y \cap Z_i| \leq 1, \forall i$. If such a split occurs, the set returned by `ColorCoding` will contain[1] $\Sigma(Y)$. Indeed, by choosing the element of Y for those Z_i that $|Y \cap Z_i| = 1$ and 0 for the rest, we successfully generate k-tuples containing $\Sigma(Y)$ through the pairwise sum operations. The algorithm returns only valid sums, since no element is used more than once in each sum, because each element is assigned uniquely to a Z_i for each distinct partition.

Our intended goal is thus, for any k random disjoint subsets, to have a partition that splits them all. Such a partition allows us to construct a k-tuple consisting of all their respective sums through the use of FFT. Hence, we have to determine how many random partitions are required to achieve this with high probability.

The probability that a split occurs at a random partition for k random disjoint subsets $Y_1, \ldots, Y_k \subseteq Z$ is

$$\Pr[\text{all } Y_i \text{ are split}] = \prod_{i=1}^{k} \Pr[Y_i \text{ is split}] =$$

$$\frac{c^2-1}{c^2} \cdots \frac{c^2-(|Y_1|-1)}{c^2} \cdots \frac{c^2-1}{c^2} \cdots \frac{c^2-(|Y_k|-1)}{c^2} \geq$$

$$\left(\frac{c^2-(|Y_1|-1)}{c^2}\right)^{|Y_1|} \cdots \left(\frac{c^2-(|Y_k|-1)}{c^2}\right)^{|Y_k|} \geq$$

$$\left(1-\frac{1}{c}\right)^{c} \cdots \left(1-\frac{1}{c}\right)^{c} \geq \left(\frac{1}{2}\right)^{2} \cdots \left(\frac{1}{2}\right)^{2} = \frac{1}{4^k}$$

Hence, for $\beta = 4^k/(4^k-1)$, $r = \lceil \log_\beta(1/\delta) \rceil$ repetitions yield the desired success probability of $1-(1-1/4^k)^r \geq 1-\delta$. In other words, after r random partitions, for any k random disjoint subsets $Y_1, \ldots, Y_k$, there exists, with probability at least $1-\delta$, a partition that splits them all.

Lemma 5.3. *Given a set Z of positive integers, a sum bound t, a size bound $c \geq 1$ and an error probability $\delta > 0$, `ColorCoding`(Z, t, k, δ) returns a set S consisting of any k-tuple $(\Sigma(Y_1), \ldots, \Sigma(Y_k))$ with probability at least $1-\delta$, where $Y_1, \ldots, Y_k \subseteq Z$ are disjoint subsets with $\Sigma(Y_1), \ldots, \Sigma(Y_k) \leq t$ and $|Y_1|, \ldots, |Y_k| \leq c$, in $\mathcal{O}(c^2 \log(1/\delta) t^k \log t)$ time.*

[1] In this context, "contain" is used to denote that $\Sigma(Y) = s_i$ for some i in a k-tuple $s = (s_1, \ldots, s_k), s \in S$, where S is the resulting set of `ColorCoding`.

Algorithm 16: `ColorCoding(Z, t, c, δ)`

Input : A set Z of positive integers, an upper bound t, a size bound $c \geq 1$ and an error probability $\delta > 0$.

Output: A set $S \subseteq (\mathcal{S}_t(Z))^k$ containing any $(\Sigma(Y_1), \ldots, \Sigma(Y_k))$ with probability at least $1 - \delta$, where $Y_1, \ldots, Y_k \subseteq Z$ disjoint subsets with $\Sigma(Y_1), \ldots, \Sigma(Y_k) \leq t$ and $|Y_1|, \ldots, |Y_k| \leq c$.

```
S ← ∅
β ← 4^k/(4^k − 1)
for j = 1, ..., ⌈log_β(1/δ)⌉ do
    randomly partition Z = Z_1 ∪ Z_2 ∪ ··· ∪ Z_{c^2}
    for i = 1, ..., c^2 do
        Z'_i ← (Z_i × {0}^{k−1}) ∪ ... ∪ ({0}^{k−1} × Z_i)
    end
    S_j ← Z'_1 ⊕_t ··· ⊕_t Z'_{c^2}
    S ← S ∪ S_j
end
return S
```

Proof. As we have already explained, if there exist k disjoint subsets $Y_1, \ldots, Y_k \subseteq Z$ with $\Sigma(Y_1), \ldots, \Sigma(Y_k) \leq t$ and $|Y_1|, \ldots, |Y_k| \leq c$, our algorithm guarantees that with probability at least $1 - \delta$, there exists a partition that splits them all. Subsequently, the FFT operations on the corresponding points produce the k-tuple.

Complexity. The algorithm performs $\mathcal{O}(\log(1/\delta))$ repetitions. To compute a pairwise sum of k variables up to t, $\mathcal{O}(t^k \log t)$ time is required. In each repetition, c^2 pairwise sums are computed. Hence, the total complexity of the algorithm is $\mathcal{O}(c^2 \log(1/\delta) t^k \log t)$. □

5.1.2 Solving k-Subset Sum for l-layer instances of Z

In this part, we will prove that we can use the algorithm `ColorCodingLayer` from [Brin17], to successfully solve k-Subset Sum for l-layer instances. We will show that by calling the modified `ColorCoding` algorithm presented previously in this section and modifying the FFT operations of `ColorCodingLayer` so they concern sets of points, the algorithm successfully solves k-Subset Sum in such instances. We refer the reader to page 46, where the original algorithm by Bringmann is presented.

Suppose that $X^1, \ldots, X^k \subseteq Z$ are disjoint subsets with $\Sigma(X^1), \ldots, \Sigma(X^k) \leq t$. By Claim 3.2 on page 45, we have that $\Pr[|Y_i| \geq 6\log(l/\delta)] \leq \delta/l$, where $Y_i = Y \cap Z_i$, for any $Y \subseteq Z$ with at most l elements. Hence, the probability that $|X_i^1| \leq 6\log(l/\delta)$ and $|X_i^2| \leq 6\log(l/\delta)$ and so on, for all $i = 1, \ldots, m$ is

$$\Pr[\bigwedge_{i=1}^{m}(|X_i^1| \leq 6\log(l/\delta) \wedge \ldots \wedge |X_i^k| \leq 6\log(l/\delta))] \geq$$

$$1 - \sum_{i=1}^{m}\left(\Pr[|X_i^1| > 6\log(l/\delta)]\right) - \ldots - \sum_{i=1}^{m}\Big(\Pr[|X_i^k| > 6\log(l/\delta)]\Big) \geq 1 - km\delta/l.$$

`ColorCoding` computes $(\Sigma(X_i^1), \ldots, \Sigma(X_i^k))$ with probability at least $1 - \delta$. This happens for all i with probability at least $1 - m\delta/l$. Then, combining the resulting sets indeed yields the k-tuple $(\Sigma(X^1), \ldots, \Sigma(X^k))$. The total error probability is at most $(k+1)m\delta/l$. Assume that $\delta \in (0, 1/2^{k+1}]$. Since $l \geq 1$ and $\delta \leq 1/2^{k+1}$, we have $\log(l/\delta) \geq (k+1)$. Hence, the total error probability is bounded by δ. This gives the following.

Lemma 5.4. *Given an l-layer instance (Z,t), upper bound t and error probability $\delta \in (0,1/2^{k+1}]$, `ColorCodingLayer`(Z,t,l,δ) solves k-Subset Sum with sum at most t in time $\mathcal{O}\left(t^k \log t \frac{\log^{k+2}(l/\delta)}{l^{k-1}}\right)$ with probability at least $1-\delta$.*

Complexity. The time to compute the sets of k-tuples $S_1,\ldots,S_m$ by calling `ColorCoding` is

$$\mathcal{O}(m\cdot\gamma^2\log(l/\delta)(\gamma t/l)^k\log(\gamma t/l)) = \mathcal{O}\left(\frac{\gamma^{k+2}}{l^{k-1}}t^k\log t\right) = \mathcal{O}\left(\frac{\log^{k+2}(l/\delta)}{l^{k-1}}t^k\log t\right).$$

Combining the resulting sets costs

$$\mathcal{O}\left(\sum_{h=1}^{\log m}\frac{m}{2^h}(2^h\gamma t/l)^k\log(2^h\gamma t/l)\right) = \mathcal{O}\left(\sum_{h=1}^{\log m}\frac{2^{h(k-1)}}{m^{k-1}}t^k\log t\right) = \mathcal{O}\left(\frac{t^k\log t}{m^{k-1}}\sum_{h=1}^{\log m}\left(2^{k-1}\right)^h\right)$$
$$= \mathcal{O}\left(\frac{t^k\log t}{m^{k-1}}\left(2^{k-1}\right)^{\log m}\right) = \mathcal{O}(t^k\log t)$$

since for $c>1$, we have that $\mathcal{O}\left(\sum_{k=0}^{n}c^k\right)=\mathcal{O}(c^n)$, which is dominated by the computation of $S_1,\ldots,S_m$.

Hence, `ColorCodingLayer` has total complexity $\mathcal{O}\left(t^k\log t\frac{\log^{k+2}(l/\delta)}{l^{k-1}}\right)$.

5.1.3 General Case

It remains to show that for every instance (Z,t), we can construct l-layer instances and take advantage of `ColorCodingLayer` to solve k-Subset Sum for the general case. This is possible by partitioning set Z at $t/2^i$ for $i=1,\ldots,\lceil\log n\rceil-1$. Thus, we have $\mathcal{O}(\log n)$ l-layers $Z_1,\ldots,Z_{\lceil\log n\rceil}$. On each layer we run `ColorCodingLayer`, and then we combine the results using pairwise sums. The algorithm is presented on page 47 and the only difference is that the FFT operations concern sets of points, hence the complexity analysis differs accordingly.

We will now show how Algorithm 4 on page 47 proves Theorem 5.2.

Proof. Suppose there exist k disjoint subsets $X^1,\ldots,X^k\subseteq Z$ with $\Sigma(X^1),\ldots,\Sigma(X^k)\le t$, and $X_i^j = X^j\cap Z_i$, for $j=1,\ldots,k$ and $i=1,\ldots,\lceil\log n\rceil$. Each call to `ColorCodingLayer` returns a k-tuple $(\Sigma(X_i^1),\ldots,\Sigma(X_i^k))$ with probability at least $1-\delta/\lceil\log n\rceil$, hence the probability that all calls return the corresponding k-tuple is

$$\Pr[\texttt{ColorCodingLayer returns } (\Sigma(X_i^1),\ldots,\Sigma(X_i^k)),\forall i] = 1-\Pr[\text{some call fails}] \ge$$
$$1-\sum_{i=1}^{\lceil\log n\rceil}\frac{\delta}{\lceil\log n\rceil} = 1-\lceil\log n\rceil\cdot\frac{\delta}{\lceil\log n\rceil} = 1-\delta$$

If all calls return the corresponding k-tuple, the algorithm will successfully construct the k-tuple $(\Sigma(X^1),\ldots,\Sigma(X^k))$. Thus, with probability at least $1-\delta$, the algorithm solves k-Subset Sum.

Complexity. Reading the input requires $\Theta(n)$ time. The algorithm has $\Theta(\log n)$ repetitions, and in each repetition we make a call to `ColorCodingLayer`, plus compute a pairwise sum. The computation of the pairwise sum requires $\mathcal{O}(t^k\log t)$ time since it concerns k-tuples. Each call to `ColorCodingLayer` requires time $\mathcal{O}\left(t^k\log t\frac{\log^{k+2}\left(\frac{2^i\log n}{\delta}\right)}{2^{i(k-1)}}\right)$, hence the overall complexity is

$$\mathcal{O}\left(n+t^k\log t\log n+\sum_{i=1}^{\log n}t^k\log t\frac{\log^{k+2}\left(\frac{2^i\log n}{\delta}\right)}{2^{i(k-1)}}\right) = \tilde{\mathcal{O}}(n+t^k).$$

□

5.2 Solving k-Subset Sum in $\tilde{\mathcal{O}}(n^{k/k+1}t^k)$ time

Theorem 5.5. *Given a set of positive integers $Z \subseteq \mathbb{N}$ of size n and targets $t_1, \ldots, t_k$, one can decide k-Subset Sum in $\tilde{\mathcal{O}}(n^{k/k+1}t^k)$, where $t = max\{t_1, \ldots, t_k\}$.*

In this section we show how to decide k-Subset Sum in $\tilde{\mathcal{O}}(n^{k/k+1}t^k)$ time, where $t = max\{t_1, \ldots, t_k\}$. To this end we extend the algorithm proposed by Koiliaris and Xu [Koil19], as presented in Section 3.3. The extended algorithm determines whether there exist k disjoint subsets each summing up to a given target t_i respectively. In comparison to the original algorithm, a couple of modifications are required, which we will first briefly summarise prior to presenting the complete algorithm.

Our extension begins by using the k-modified characteristic polynomials previously proposed, thereby representing each $z \in Z$ as $\sum_{i=1}^{k} x_i^z$ in the base polynomial at the leaves of the recursion. Furthermore, we use k additional dimensions $c_1, \ldots, c_k$ for the cardinality of the sums represented by the exponents of x_i. We then proceed with multiplying each step using FFT in the same way as in the original algorithm, thereby producing polynomials with terms that contain k-tuples of sums $x_1^{s_1} \ldots x_k^{s_k}$.

We now observe that each of the terms of the form $x_1^{s_1} \ldots x_k^{s_k}$ was produced at some point of the recursion via an FFT operation combining two terms that belonged to different subtrees, ergo containing different elements in each subset involved. As such, $s_1, \ldots, s_k$ are sums of disjoint subsets of Z. This guarantees the correctness of the algorithm.

It remains to complete the recursion and examine whether the final polynomial contains a term $x_1^{t_1} \ldots x_k^{t_k}$ to answer if there exist k disjoint subsets that sum up to $t_1, \ldots, t_k$ respectively.

Algorithm 17: `DisjointSC(S, t)`

Input : A set S of n positive integers and an upper bound integer t.
Output: The set $Z \subseteq (\mathcal{S}_t(S) \times [n])^k$ of all k-tuples of subset sums occurring from disjoint subsets of S up to t, along with their respective cardinality information.

1 **if** $S = \{s\}$ **then return** $\{0\}^{2k} \cup \{(s, 1, \overbrace{0, \ldots, 0}^{2(k-1)})\} \cup \ldots \cup \{(\overbrace{0, \ldots, 0}^{2(k-1)}, s, 1)\}$
2 $T \leftarrow$ an arbitrary subset of S of size $\lfloor \frac{n}{2} \rfloor$
3 **return** `DisjointSC`$(T, t) \oplus_t$ `DisjointSC`$(S \setminus T, t)$

Algorithm 18: `DisjointSS(Z, t)`

Input : A set Z of n positive integers and an upper bound integer t.
Output: The set $S \subseteq (\mathcal{S}_t(Z))^k$ of all k-tuples of subset sums up to t occurring from disjoint subsets of Z.

1 $b \leftarrow \lfloor \sqrt[k+1]{n^k \log n} \rfloor$
2 **for** $l \in [b-1]$ **do**
3 $\quad S_l \leftarrow Z \cap \{x \in \mathbb{N} \mid x \equiv l \pmod b\}$
4 $\quad Q_l \leftarrow \{\lfloor x/b \rfloor \mid x \in S_l\}$
5 $\quad \mathcal{R}(Q_l) \leftarrow$ `DisjointSC`$(Q_l, \lfloor t/b \rfloor)$
6 $\quad R_l \leftarrow \{(z_1 b + j_1 l, \ldots, z_k b + j_k l) \mid (z_1, j_1, \ldots, z_k, j_k) \in \mathcal{R}(Q_l)\}$
7 **end**
8 **return** $R_0 \oplus_t \cdots \oplus_t R_{b-1}$

Complexity. The overall complexity of the algorithm results from the computation of the sums inside the congruence classes and the combination of those sums.

The computation of the sums in the congruence classes costs

$$\sum_{l\in[b-1]} \mathcal{O}((t/b)^k n_l^k \log n_l \log t) = \mathcal{O}((t/b)^k n^k \log n \log t),$$

and their combination costs $\mathcal{O}(bt^k \log t)$, hence the total running time is

$$\mathcal{O}\left(t^k \log t \left(\frac{n^k \log n}{b^k} + b\right)\right) = \tilde{\mathcal{O}}(n^{k/k+1} t^k),$$

which is obtained by setting $b = \sqrt[k+1]{n^k \log n}$.

5.3 Faster Algorithms for Multiple Subset Problems

The techniques developed in this chapter can be further applied to give faster pseudopolynomial algorithms for the decision version of Subset Sum Ratio, k-Subset Sum Ratio and Multiple Subset Sum. In this section we will present how these algorithms can be used to (efficiently) solve these problems.

The algorithms we previously presented result in a polynomial $P(x_1, \ldots, x_k)$ consisting of terms each of which corresponds to a k-tuple of realisable sums by disjoint subsets of the initial input set Z. In other words, if there exists a term $x_1^{s_1}, \ldots x_k^{s_k}$ in the resulting polynomial, then there exist disjoint subsets $Z_1, \ldots, Z_k \subseteq Z$ such that $\Sigma(Z_1) = s_1, \ldots, \Sigma(Z_k) = s_k$.

It is important to note that, while the deterministic algorithm of Subsection 5.2 returns a polynomial consisting of *all* terms corresponding to such k-tuples of realisable sums by disjoint subsets, the randomised algorithm of Subsection 5.1 does not. However, that does not affect the correctness of the following algorithms, since it suffices to guarantee that the k-tuple corresponding to the optimal solution of the respective (optimisation) problem is included with high probability. That indeed happens, since the resulting polynomial consists of any viable term with high probability, as discussed previously.

Subset Sum Ratio The first variation we will discuss is the Subset Sum Ratio problem, which is to determine, given a set $Z \subseteq \mathbb{N}$ of size n and an upper bound t, what is the smallest ratio of sums between any two disjoint subsets $S_1, S_2 \subseteq Z$, where $\Sigma(S_1), \Sigma(S_2) \leq t$. This can be solved in deterministic $\tilde{\mathcal{O}}(n^{2/3}t^2)$ time using the algorithm proposed in section 5.2 by simply iterating over the terms of the final polynomial that involve both parameters x and y and checking the ratio of their exponents. Subset Sum Ratio can also be solved with high probability in randomised $\tilde{\mathcal{O}}(n + t^2)$ time using the algorithm proposed in subsection 5.1 instead.

k-Subset Sum Ratio An additional extension is the k-Subset Sum Ratio problem, which asks, given a set $Z \subseteq \mathbb{N}$ of size n and k bounds $t_1, \ldots, t_k$, to determine what is the smallest ratio between the largest and smallest sum of any set of k disjoint subsets $Z_1, \ldots, Z_k \subseteq Z$ such that $\Sigma(Z_i) \leq t_i$. Similar to k-Subset Sum, an interesting special case is when all t_i's are equal, in which case we search for k subsets that are as similar as possible in terms of sum.

Similarly, we can solve this in deterministic $\tilde{\mathcal{O}}(n^{k/(k+1)} \cdot t^k)$ or randomised $\tilde{\mathcal{O}}(n + t^k)$ time by using the corresponding algorithm to solve k-Subset Sum and subsequently iterating over the terms of the resulting polynomial that respect the corresponding bounds, and finally evaluating the ratio of the largest to smallest exponent.

k-Partition An interesting special case of k-Subset Sum Ratio, which finds application in the field of fair allocation, is the k-Partition problem, which asks, given a set $Z \subseteq \mathbb{N}$ of size n, to partition its elements into k subsets $Z_1, \ldots, Z_k$, while minimising the ratio among the sums $\Sigma(Z_i)$.

Notice that the optimal solution values are $\Sigma(Z_i) = \sigma/k$, where $\sigma = \Sigma(Z)$. Furthermore, suppose that there exists a subset $S_i \subseteq Z$ such that $\Sigma(S_i) \geq (\sigma/k) + max$, where max is the max element of Z and $max \in S_i$. Then, there exists another subset $S_j \subseteq Z$ such that $\Sigma(S_j) \leq (\sigma/k) - max$. In this case however, a better solution would consist of the sets $S_i \backslash max$ and $S_j \cup max$. Thus, in order to solve this problem it suffices to solve k-Subset Sum Ratio for $t = max + (\Sigma(Z)/k)$, where max is the max element of Z, while only considering the terms $x_1^{s_1} \ldots x_k^{s_k}$ of the final polynomial for which $\sum s_i = \Sigma(Z)$.

Multiple Subset Sum Finally, we consider Multiple Subset Sum, which asks, given a set $Z \subseteq \mathbb{N}$ of size n and k bounds $t_1, \ldots, t_k$, to determine what is the maximum sum of sums of any combination of k disjoint subsets $Z_1, \ldots, Z_k$ of Z, such that $\Sigma(Z_i) \leq t_i$. This problem is a special case of the Multiple Knapsack problem and can also be seen as a generalisation of k-Subset Sum. It should be clear that the same techniques as those e.g. used for k-Subset Sum Ratio apply directly, leading to the same time complexity bounds of deterministic $\tilde{\mathcal{O}}(n^{k/(k+1)} \cdot t^k)$ and randomised $\tilde{\mathcal{O}}(n + t^k)$ time.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

Subset Sum has seen impressive improvements as of recently, as far as pseudopolynomial time algorithms are concerned. These improvements are able to be extended to other heavily associated problems. In this diploma thesis, we primarily examined two relevant algorithmic problems, namely Equal Subset Sum and k-Subset Sum, and how improvements on the Subset Sum problem can possibly affect them.

We demonstrated that, in the case of Equal Subset Sum, while we can extend recent Subset Sum algorithms, producing three pseudopolynomial algorithms, in this specific case there is actually a simpler and more efficient approach, heavily based on the restrictions and the nature of the problem.

As for k-Subset Sum, we introduced two pseudopolynomial algorithms which efficiently solve this much more general case.

A multitude of problems, as presented in Section 5.3, can be influenced by those very same advances, resulting in new, more efficient algorithms for many algorithmic problems, based on techniques initially developed for Subset Sum, showing their close connection with the Subset Sum problem, and how advances in it can possibly affect them.

6.2 Future Work

Firstly, we plan to investigate whether our results can be generalised to a class of subset sum problems, e.g. by using the framework developed in [Meli20]. We are also interested in approximation schemes for counting versions of knapsack and subset sum problems (see e.g. [Gopa11, Stef12, Trio20] and also [Gawr18, Rizz19]); some of these problems belong to classes below #P, like the class TotP [Pago06], which may be employed towards proving approximability results for them [Baka20].

Jin and Wu introduced an efficient $\tilde{\mathcal{O}}(n+t)$ randomised algorithm for solving Subset Sum in [Jin19]. This algorithm is vastly simpler than Bringmann's and actually has slightly better complexity. It is interesting to research whether this algorithm can be extended to cope with k-Subset Sum (and the variations mentioned in Section 5.3), as was the case for Bringmann's, since that would result in a simpler and possibly more efficient alternative approach.

The algorithm of Section 5.2 involves the computation of the possible sums of disjoint subsets along with their respective cardinality, for elements in the same congruence class. To do so, we extend the FFT operations to multiple variables, each representing either a possible subset sum or its cardinality. Hence, for k subsets and n elements, we proceed with FFT operations on variables $x_1, \ldots, x_k, c_1, \ldots, c_k$, where the exponents of x_i are in $[t/b]$ for some given upper bound t, whereas the exponents of c_i in $[n]$. Notice however that each element of our set is used only on a single subset, hence for a term $x_1^{s_1} \ldots x_k^{s_k} c_1^{n_1} \ldots c_k^{n_k}$ of the produced polynomial, it holds that $s_i \leq t/b$ and $\sum n_i \leq n$. This differs substantially from our analysis, where we essentially only assume that $n_i \leq n$, which is significantly less strict. Hence, a

stricter complexity analysis may be possible on those FFT operations, resulting in a more efficient overall complexity for this algorithm.

The algorithms introduced in this thesis solve the *decision version* of the k-Subset Sum problem. In other words, their output is a binary response, indicating whether there exist k disjoint subsets whose sums are equal to given values $t_1, \ldots, t_k$ respectively. An extension of these algorithms could involve the reconstruction of the k solution subsets. Koiliaris and Xu [Koil19] argue that one can reconstruct the solution set of Subset Sum with only polylogarithmic overhead. That is possible by carefully extracting the *witnesses* of each sum every time an FFT operation is happening. These witnesses are actually the partial sums used to compute the new sum. Thus, by reducing this problem to the *reconstruction problem* as mentioned in [Auma11], they conclude that it is possible to compute all the witnesses of an FFT operation without considerably increasing the complexity. That is the case for one-dimensional FFT operations involving a single variable, so it may be possible to use analogous arguments for multiple variables.

A completely different approach could be based on improving the space complexity of the Equal Subset Sum problem. Lokshtanov and Nederlof first introduced polynomial space algorithms for Subset Sum in [Loks10]. Bringmann later improved the previous bounds in [Brin17]. Very recently, Jin *et al.* [Jin21] have made advances in this area. These algorithms are likely to be able to be adapted for the Equal Subset Sum problem, resulting in pseudopolynomial time and *polynomial* space complexity.

Lastly, the relationship between Equal Subset Sum and Orthogonal Vector seems very interesting and definitely worthy of research. Suppose that we have an input set $Z = \{z_1, \ldots, z_n\}$. We define the *inclusion vector* of a subset $V \subseteq Z$ as a vector $v = (v_1, \ldots, v_n)$ of n positions, where $v_i = 1$ if and only if $z_i \in V$, else $v_i = 0$. Then, it holds that if two subsets A, B are disjoint, then their inclusion vectors are orthogonal, since either $a_i = 0$ or $b_i = 0$, for all $i \in [n]$. Also note, that given a table of coefficients $C \in \{0, 1\}^{m \times n}$, representing m vectors of dimension n (each of which corresponds to a possible inclusion vector), a sum $s \in \mathbb{N}$, representing a possible sum for which there exist multiple disjoint subsets summing up to it, and a table $B \in \{s\}^{m \times 1}$, by solving the equation $C \cdot X = B$ and restricting the solution $X \in \mathbb{N}^{n \times 1}$ to distinct positive integers (i.e., $x_i \neq x_j$, for all $i \neq j$), one can construct a Equal Subset Sum instance given one such (restricted) instance of Orthogonal Vector. The equation may be solved using *integer programming* and, since matrix C is *totally unimodular*, the complexity of this solution may be less than expected. These remarks lead us to believe that a reduction Orthogonal Vector $\leq$ Equal Subset Sum may be possible, firstly by restricting a random instance of Orthogonal Vector to one satisfying the previously described requirements and subsequently producing an instance of Equal Subset Sum from it.

Bibliography

[Alon95] Noga Alon, Raphael Yuster and Uri Zwick, "Color-Coding", *J. ACM*, vol. 42, no. 4, pp. 844–856, 1995.

[Auma11] Yonatan Aumann, Moshe Lewenstein, Noa Lewenstein and Dekel Tsur, "Finding witnesses by peeling", *ACM Trans. Algorithms*, vol. 7, no. 2, pp. 24:1–24:15, 2011.

[Baka20] Eleni Bakali, Aggeliki Chalki and Aris Pagourtzis, "Characterizations and Approximability of Hard Counting Classes Below #P", in Jianer Chen, Qilong Feng and Jinhui Xu, editors, *Theory and Applications of Models of Computation, 16th International Conference, TAMC 2020, Changsha, China, October 18-20, 2020, Proceedings*, vol. 12337 of *Lecture Notes in Computer Science*, pp. 251–262, Springer, 2020.

[Bazg02] Cristina Bazgan, Miklos Santha and Zsolt Tuza, "Efficient Approximation Algorithms for the SUBSET-SUMS EQUALITY Problem", *J. Comput. Syst. Sci.*, vol. 64, no. 2, pp. 160–170, 2002.

[Bell57] Richard E. Bellman, *Dynamic programming*, Princeton University Press, 1957.

[Blah85] Richard E. Blahut, *Fast Algorithms for Digital Signal Processing*, Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1985.

[Bren76] Richard P. Brent, "Multiple-precision zero-finding methods and the complexity of elementary function evaluation", in J.F. Traub, editor, *Analytic Computational Complexity*, pp. 151–176, Academic Press, 1976.

[Brin17] Karl Bringmann, "A Near-Linear Pseudopolynomial Time Algorithm for Subset Sum", in Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pp. 1073–1084, SIAM, 2017.

[Brin20] Karl Bringmann and Vasileios Nakos, "Top-k-convolution and the quest for near-linear output-sensitive subset sum", in Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pp. 982–995, ACM, 2020.

[Capr00] Alberto Caprara, Hans Kellerer and Ulrich Pferschy, "A PTAS for the Multiple Subset Sum Problem with different knapsack capacities", *Inf. Process. Lett.*, vol. 73, no. 3-4, pp. 111–118, 2000.

[Capr03] Alberto Caprara, Hans Kellerer and Ulrich Pferschy, "A 3/4-Approximation Algorithm for Multiple Subset Sum", *J. Heuristics*, vol. 9, no. 2, pp. 99–111, 2003.

[Ciel03a] Mark Cieliebak, Stephan J. Eidenbenz and Aris Pagourtzis, "Composing Equipotent Teams", in Andrzej Lingas and Bengt J. Nilsson, editors, *Fundamentals of Computation Theory, 14th International Symposium, FCT 2003, Malmö, Sweden,*

August 12-15, 2003, Proceedings, vol. 2751 of *Lecture Notes in Computer Science*, pp. 98–108, Springer, 2003.

[Ciel03b] Mark Cieliebak, Stephan J. Eidenbenz and Paolo Penna, "Noisy Data Make the Partial Digest Problem NP-hard", in Gary Benson and Roderic D. M. Page, editors, *Algorithms in Bioinformatics, Third International Workshop, WABI 2003, Budapest, Hungary, September 15-20, 2003, Proceedings*, vol. 2812 of *Lecture Notes in Computer Science*, pp. 111–123, Springer, 2003.

[Ciel04] Mark Cieliebak and Stephan J. Eidenbenz, "Measurement Errors Make the Partial Digest Problem NP-Hard", in Martin Farach-Colton, editor, *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*, vol. 2976 of *Lecture Notes in Computer Science*, pp. 379–390, Springer, 2004.

[Ciel08] Mark Cieliebak, Stephan J. Eidenbenz, Aris Pagourtzis and Konrad Schlude, "On the Complexity of Variations of Equal Sum Subsets", *Nord. J. Comput.*, vol. 14, no. 3, pp. 151–172, 2008.

[Corm09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms, 3rd Edition*, MIT Press, 2009.

[Dell19] Mauro Dell'Amico, Maxence Delorme, Manuel Iori and Silvano Martello, "Mathematical models and decomposition methods for the multiple knapsack problem", *Eur. J. Oper. Res.*, vol. 274, no. 3, pp. 886–899, 2019.

[Gawr18] Pawel Gawrychowski, Liran Markin and Oren Weimann, "A Faster FPTAS for #Knapsack", in Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, vol. 107 of *LIPIcs*, pp. 64:1–64:13, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[Gopa11] Parikshit Gopalan, Adam R. Klivans, Raghu Meka, Daniel Stefankovic, Santosh S. Vempala and Eric Vigoda, "An FPTAS for #Knapsack and Related Counting Problems", in Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pp. 817–826, IEEE Computer Society, 2011.

[Jin19] Ce Jin and Hongxun Wu, "A Simple Near-Linear Pseudopolynomial Time Randomized Algorithm for Subset Sum", in Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, vol. 69 of *OASICS*, pp. 17:1–17:6, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[Jin21] Ce Jin, Nikhil Vyas and Ryan Williams, "Fast Low-Space Algorithms for Subset Sum", in Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pp. 1757–1776, SIAM, 2021.

[Karp72] Richard M. Karp, *Reducibility among Combinatorial Problems*, pp. 85–103, Springer US, Boston, MA, 1972.

[Khan17] Muhammad Ali Khan, "Some Problems on Graphs and Arrangements of Convex Bodies", 2017.

[Koil19] Konstantinos Koiliaris and Chao Xu, "Faster Pseudopolynomial Time Algorithms for Subset Sum", *ACM Trans. Algorithms*, vol. 15, no. 3, pp. 40:1–40:20, 2019.

[Lahy19] Rahma Lahyani, Khalil Chebil, Mahdi Khemakhem and Leandro C. Coelho, "Matheuristics for solving the Multiple Knapsack Problem with Setup", *Comput. Ind. Eng.*, vol. 129, pp. 76–89, 2019.

[Lipt04] Richard J. Lipton, Evangelos Markakis, Elchanan Mossel and Amin Saberi, "On approximately fair allocations of indivisible goods", in Jack S. Breese, Joan Feigenbaum and Margo I. Seltzer, editors, *Proceedings 5th ACM Conference on Electronic Commerce (EC-2004), New York, NY, USA, May 17-20, 2004*, pp. 125–131, ACM, 2004.

[Loks10] Daniel Lokshtanov and Jesper Nederlof, "Saving space by algebraization", in Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pp. 321–330, ACM, 2010.

[Meli18] Nikolaos Melissinos and Aris Pagourtzis, "A Faster FPTAS for the Subset-Sums Ratio Problem", in Lusheng Wang and Daming Zhu, editors, *Computing and Combinatorics - 24th International Conference, COCOON 2018, Qing Dao, China, July 2-4, 2018, Proceedings*, vol. 10976 of *Lecture Notes in Computer Science*, pp. 602–614, Springer, 2018.

[Meli20] Nikolaos Melissinos, Aris Pagourtzis and Theofilos Triommatis, "Approximation Schemes for Subset Sum Ratio Problems", in Minming Li, editor, *Frontiers in Algorithmics - 14th International Workshop, FAW 2020, Haikou, China, October 19-21, 2020, Proceedings*, vol. 12340 of *Lecture Notes in Computer Science*, pp. 96–107, Springer, 2020.

[Much19] Marcin Mucha, Jesper Nederlof, Jakub Pawlewicz and Karol Wegrzycki, "Equal-Subset-Sum Faster Than the Meet-in-the-Middle", in Michael A. Bender, Ola Svensson and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, vol. 144 of *LIPIcs*, pp. 73:1–73:16, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[Nano13] Danupon Nanongkai, "Simple FPTAS for the subset-sums ratio problem", *Inf. Process. Lett.*, vol. 113, no. 19-21, pp. 750–753, 2013.

[Naor95] Moni Naor, Leonard J. Schulman and Aravind Srinivasan, "Splitters and Near-Optimal Derandomization", in *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pp. 182–191, IEEE Computer Society, 1995.

[Pago06] Aris Pagourtzis and Stathis Zachos, "The Complexity of Counting Functions with Easy Decision Version", in Rastislav Kralovic and Pawel Urzyczyn, editors, *Mathematical Foundations of Computer Science 2006, 31st International Symposium, MFCS 2006, Stará Lesná, Slovakia, August 28-September 1, 2006, Proceedings*, vol. 4162 of *Lecture Notes in Computer Science*, pp. 741–752, Springer, 2006.

[Papa94] Christos H. Papadimitriou, "On the Complexity of the Parity Argument and Other Inefficient Proofs of Existence", *J. Comput. Syst. Sci.*, vol. 48, no. 3, pp. 498–532, 1994.

[Pisi99] David Pisinger, "Linear Time Algorithms for Knapsack Problems with Bounded Weights", *J. Algorithms*, vol. 33, no. 1, pp. 1–14, 1999.

[Rizz19] Romeo Rizzi and Alexandru I. Tomescu, "Faster FPTASes for counting and random generation of Knapsack solutions", *Inf. Comput.*, vol. 267, pp. 135–144, 2019.

[Schm90] Jeanette P. Schmidt and Alan Siegel, "The Spatial Complexity of Oblivious k-Probe Hash Functions", *SIAM J. Comput.*, vol. 19, no. 5, pp. 775–786, 1990.

[Stef12] Daniel Stefankovic, Santosh S. Vempala and Eric Vigoda, "A Deterministic Polynomial-Time Approximation Scheme for Counting Knapsack Solutions", *SIAM J. Comput.*, vol. 41, no. 2, pp. 356–366, 2012.

[Trio20] Theofilos Triommatis and Aris Pagourtzis, "Approximate #Knapsack Computations to Count Semi-fair Allocations", in Jianer Chen, Qilong Feng and Jinhui Xu, editors, *Theory and Applications of Models of Computation, 16th International Conference, TAMC 2020, Changsha, China, October 18-20, 2020, Proceedings*, vol. 12337 of *Lecture Notes in Computer Science*, pp. 239–250, Springer, 2020.

[Volo17] Nadav Voloch, "MSSP for 2-D sets with unknown parameters and a cryptographic application", *Contemporary Engineering Sciences*, vol. 10, pp. 921–931, 10 2017.

[Woeg92] Gerhard J. Woeginger and Zhongliang Yu, "On the Equal-Subset-Sum Problem", *Inf. Process. Lett.*, vol. 42, no. 6, pp. 299–302, 1992.