Check for updates

Faster algorithms for *k*-subset sum and variations

Antonis Antonopoulos[1] ⓘ · **Aris Pagourtzis[1]** ⓘ · **Stavros Petsalakis[1]** ⓘ ·
Manolis Vasilakis[1] ⓘ

Abstract

We present new, faster pseudopolynomial time algorithms for the k-SUBSET SUM problem, defined as follows: given a set Z of n positive integers and k targets $t_1, \ldots, t_k$, determine whether there exist k disjoint subsets $Z_1, \ldots, Z_k \subseteq Z$, such that $\Sigma(Z_i) = t_i$, for $i = 1, \ldots, k$. Assuming $t = \max\{t_1, \ldots, t_k\}$ is the maximum among the given targets, a standard dynamic programming approach based on Bellman's algorithm can solve the problem in $O(nt^k)$ time. We build upon recent advances on SUBSET SUM due to Koiliaris and Xu, as well as Bringmann, in order to provide faster algorithms for k-SUBSET SUM. We devise two algorithms: a deterministic one of time complexity $\tilde{O}(n^{k/(k+1)}t^k)$ and a randomised one of $\tilde{O}(n + t^k)$ complexity. Additionally, we show how these algorithms can be modified in order to incorporate cardinality constraints enforced on the solution subsets. We further demonstrate how these algorithms can be used in order to cope with variations of k-SUBSET SUM, namely SUBSET SUM RATIO, k-SUBSET SUM RATIO and MULTIPLE SUBSET SUM.

Keywords Color coding · FFT · k-SUBSET SUM · MULTIPLE KNAPSACK · MULTIPLE SUBSET SUM · Pseudopolynomial algorithms · SUBSET SUM

✉ Manolis Vasilakis
mvasilakis@corelab.ntua.gr

Antonis Antonopoulos
aanton@corelab.ntua.gr

Aris Pagourtzis
pagour@cs.ntua.gr

Stavros Petsalakis
spetsalakis@corelab.ntua.gr

[1] School of Electrical and Computer Engineering, National Technical University of Athens, 15780 Athens, Greece

1 Introduction

One of Karp's (1972) 21 NP-complete problems, SUBSET SUM has seen astounding progress with respect to its pseudopolynomial time solvability over the last few years. Koiliaris and Xu (2019) and Bringmann (2017) have presented pseudopolynomial algorithms resulting in substantial improvements over the long-standing standard approach of Bellman (1957), and the improvement by Pisinger (1999).

EQUAL SUBSET SUM is a closely related algorithmic problem with applications to computational biology (Cieliebak et al. 2003b; Cieliebak and Eidenbenz 2004), computational social choice (Lipton et al. 2004), and cryptography (Voloch 2017), to name a few. Additionally, it is related to important theoretical concepts such as the complexity of search problems in the class TFNP (Papadimitriou 1994). This problem can be solved in $\tilde{O}(n + t)$ time[1] (where t is a bound for the sums we are interested in) by a simple extension of Bellman's (1957) algorithm, as it only asks for any two disjoint subsets of equal sum s, for some $s \leq t$, hence one can exploit the following *disjointness property*: if there exists a pair of sets with equal sum $s \leq t$, then by removing their common elements, one can produce two *disjoint* sets of equal sum $s' \leq s \leq t$. Therefore, the algorithm terminates as soon as a sum is produced for a second time and by using this property as well as an appropriate tree data structure, we can bound the total running time by the number of all the possible distinct subset sums up to t, hence obtaining an $\tilde{O}(n + t)$ algorithm.

A generalisation of SUBSET SUM is the problem 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 $t_1 = t_2$, which is closely related to EQUAL SUBSET SUM, the problem seems to escalate in complexity, since the aforementioned disjointness property does not hold any more. A special case of particular interest is when the sum of targets equals the sum of the input elements, that is, we ask for a partition of the input set to subsets of particular sums. Yet another interesting special case is when we want to partition the input set to k subsets of equal sum, i.e. all t_i's are equal to the sum of the input elements divided by k.

k-SUBSET SUM finds applications in a variety of fields, including, but not limited to, scheduling and (fair) resource allocation (Schreiber et al. 2018), sports team realignment (Recalde et al. 2018), as well as balanced control and treatment group formation for medical studies, focus groups for market research, and controlled sampling for opinion polling. Furthermore, since it generalises several subset sum problems, such as k- WAY NUMBER PARTITIONING and MULTIPLE SUBSET SUM, by enforcing cardinality constraints, one could use it to solve a number of other meaningful problems (Tsai 1992).

1.1 Related work

EQUAL SUBSET SUM as well as its optimisation version called SUBSET SUM RATIO (Bazgan et al. 2002) are closely related to problems appearing in many sci-

---

[1] $\tilde{O}$ notation ignores polylogarithmic factors.

entific areas. Some examples include the PARTIAL DIGEST problem, which comes from computational biology (Cieliebak et al. 2003b; Cieliebak and Eidenbenz 2004), the allocation of individual goods (Lipton et al. 2004), tournament construction (Khan 2017), and a variation of SUBSET SUM, called Multiple Integrated Sets SSP, which finds applications in the field of cryptography (Voloch 2017). Furthermore, 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 (Papadimitriou 1994), 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 NP-hard by Woeginger and Yu (1992) and several variations have been proven NP-hard by Cieliebak et al. (2003a, 2008). A 1.324-approximation algorithm has been proposed for SUBSET SUM RATIO in Woeginger and Yu (1992), while several FPTASs have appeared in bibliography (Bazgan et al. 2002; Nanongkai 2013; Melissinos and Pagourtzis 2018; Alonistiotis et al. 2022).

As far as exact algorithms are concerned, recent progress has shown that EQUAL SUBSET SUM can be solved probabilistically in $O^*(1.7088^n)$ time (Mucha et al. 2019), which is faster than a standard "meet-in-the-middle" approach yielding an $O^*(3^{n/2}) \leq O^*(1.7321^n)$ time algorithm. Additionally, as explained earlier, an exact solution can be obtained in pseudopolynomial $\tilde{O}(n+t)$ time using an extension of Bellman's (1957) algorithm, when an upper bound t on the sums is provided in the input. However, these techniques do not seem to apply to k-SUBSET SUM, mainly due to the fact that we cannot assume the minimality of the involved subsets. To the best of our knowledge, no pseudopolynomial time algorithm substantially faster than the standard $O(nt^k)$ dynamic programming approach was known for k-SUBSET SUM prior to this work.

EQUAL SUBSET SUM and k-SUBSET SUM are tightly connected to SUBSET SUM, which has seen impressive advances recently, due to Koiliaris and Xu (2019) who gave a deterministic $\tilde{O}(\sqrt{n}t)$ algorithm, where n is the number of input elements and t is the target, and by Bringmann (2017) who gave a $\tilde{O}(n + t)$ randomised algorithm. Jin and Wu proposed a simpler randomised algorithm (Jin and Wu 2018) achieving the same bounds as (Bringmann 2017), which however seems to only solve the decision version of the problem. Both of these algorithms are essentially optimal under SETH (Abboud et al. 2022). Recently, Bringmann and Nakos (2020) have presented an $O(|\mathcal{S}_t(Z)|^{4/3}\text{poly}(\log t))$ 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 MULTIPLE KNAPSACK, 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 (Caprara et al. 2000), and subsequently in Caprara et al. (2003) 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. (2019) and Dell'Amico et al. (2019). Special cases and variants of MULTIPLE SUBSET SUM, such as the k-SUBSET SUM problem, have been studied in Cieliebak et al. (2003a, 2008) where simple pseudopolynomial algorithms were proposed.

1.2 Our contribution

We first present two algorithms for k-SUBSET SUM: a deterministic one of complexity $\tilde{O}(n^{k/(k+1)}t^k)$ and a randomised one of complexity $\tilde{O}(n + t^k)$. Then, we describe the necessary modifications required in order to solve k-SUBSET SUM in the case where there exist some cardinality constraints regarding the solution subsets. 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 (2019) and Bringmann (2017) for SUBSET SUM. In particular, we make use of FFT computations, modular arithmetic and color-coding, among others.

We start by presenting some necessary background in Sect. 2. Then, we present the two k-SUBSET SUM algorithms in Sect. 3, followed by the two algorithms for the cardinality constrained version of the problem in Sect. 4. We next show how these algorithms can be used to efficiently decide multiple related subset problems. Finally, we conclude the paper by presenting some directions for future work.

2 Preliminaries

2.1 Notation

We largely follow the notation used in Koiliaris and Xu (2019) and Bringmann (2017).

- Let $[x] = \{0, \dots, x\}$ denote the set of integers in the interval $[0, x]$.
- Given a set $Z \subseteq \mathbb{N}$, we denote:
    - the sum of its elements by $\Sigma(Z) = \sum_{z \in Z} z$.
    - the *characteristic polynomial of Z* by $f_Z(x) = \sum_{z \in Z} x^z$.
    - the *k-modified characteristic polynomial of Z* by $f_Z^k(\overrightarrow{x}) = \sum_{z \in Z} \sum_{i=1}^{k} x_i^z$, where $\overrightarrow{x} = (x_1, \dots, x_k)$.
    - the *set of all subset sums of Z up to t* by $\mathcal{S}_t(Z) = \{\Sigma(X) \mid X \subseteq Z\} \cap [t]$.

- 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 $(y_1, \dots, y_k) \in Y \cup \{0\}^k$.

2.2 Using FFT for subset sum

Given two sets $A, B \subseteq \mathbb{N}$ and an upper bound $t > 0$, one can compute the t-capped pairwise sum set $A \oplus_t B$ using FFT to get the monomials of the product $f_A \cdot f_B$ that

have exponent $\leq t$. This operation can be completed in time $O(t \log t)$. Observe that the coefficients of each monomial x^i represent the number of pairs (a, b) that sum up to i, where $a \in A \cup \{0\}$ and $b \in B \cup \{0\}$.

Also note that an FFT operation can be extended to polynomials of multiple variables. Thus, assuming an upper bound t for the exponents involved, it is possible to compute $f_A^k \cdot f_B^k$ in $O(t^k \log t)$ time.

Lemma 1 *Given two sets of points $S, T \subseteq [t]^k$ one can use multidimensional FFT to compute the set of pairwise sums (that are smaller than t) in time $O(t^k \log t)$.*

As shown in Cormen et al. (2009),Ch. 30 given two sets of points $S, T \subseteq [t]^k$ one can pipeline k one-dimensional FFT operations in order to compute a multi-dimensional FFT in time $O(t^k \log t)$.

3 k-subset sum

In this section we propose algorithms that build on the techniques of Koiliaris and Xu (2019) and Bringmann (2017) in order to solve k-SUBSET SUM: given a set Z of n positive integers and k targets $t_1, \ldots, t_k$, determine whether there exist k disjoint subsets $Z_1, \ldots, Z_k \subseteq Z$, such that $\Sigma(Z_i) = t_i$, for $i = 1, \ldots, k$. Note that k is fixed and not part of the input. For the rest of this section, assume that $Z = \{z_1, \ldots, z_n\}$ is the input set, $t_1, \ldots, t_k$ are the given targets and $t = \max\{t_1, \ldots, t_k\}$.

The main challenge in our approach is the fact that the existence of subsets summing up to the target numbers (or any other pair of numbers) does not imply the disjointness of said subsets. Hence, it is essential to guarantee the disjointness of the resulting subsets through the algorithm.

Note that one can extend Bellman's classic dynamic programming algorithm for SUBSET SUM (Bellman 1957) to solve this problem in $O(nt^k)$ time, as seen in algorithm 1, where we solve the special case $k = 2$. Similarly, the algorithm can be extended to apply for any arbitrary k.

Algorithm 1 `Bellman`(Z, t_1, t_2)

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 two disjoint subsets $Z_1, Z_2 \subseteq Z$, with $\Sigma(Z_1) = t_1$ and $\Sigma(Z_2) = t_2$, else false.
1: $t \leftarrow \max\{t_1, t_2\}$
2: initialise table $T[t][t][n] \leftarrow false$ everywhere
3: $T[0][0][0] \leftarrow true$ ▷ Initially, only $(\emptyset, \emptyset)$ is possible.
4: **for** $n_i = 1, \ldots, n$ **do**
5:     **for** $(i, j) \in [t] \times [t]$ **do**
6:         **if** $T[i][j][n_i - 1] = true$ **then**
7:             $T[i][j][n_i] = true$ ▷ $z_{n_i} \notin Z_1 \cup Z_2$
8:             $T[i + z_i][j][n_i] = true$ ▷ $z_{n_i} \in Z_1$
9:             $T[i][j + z_i][n_i] = true$ ▷ $z_{n_i} \in Z_2$
10: **return** $T[t_1][t_2][n]$

3.1 Solving *k*-subset sum in deterministic $\tilde{O}(n^{k/(k+1)}t^k)$ time

In this subsection we show how to decide k-SUBSET SUM in $\tilde{O}(n^{k/(k+1)}t^k)$ time, where $t = \max\{t_1, \ldots, t_k\}$. To this end, we extend the algorithm proposed by Koiliaris and Xu (2019). We first describe briefly the original algorithm for completeness and clarity.

The algorithm recursively partitions the input set S into two subsets of equal size and returns all pairwise sums of those sets, along with cardinality information for each sum. This is achieved using FFT for pairwise sums as discussed in Sect. 2.

Using properties of congruence classes, it is possible to further capitalise on the efficiency of FFT for subset sums as follows. Given bound t, partition the elements of the initial set into congruence classes mod b, for some integer $b > 0$. Subsequently, divide the elements by b and keep their quotient, hence reducing the size of the maximum exponent value from t to t/b. One can compute the t-capped sum of the initial elements by computing the t/b-capped sum of each congruence class and combining the results of each such operation in a final t-capped FFT operation, taking into account the remainder of each congruence class w.r.t. the number of elements (cardinality) of each subset involved. In order to achieve this, it is necessary to keep cardinality information for each sum (or monomial) involved, which can be done by adding another variable with minimal expense in complexity.

Thus, the overall complexity of FFT operations for each congruence class $l \in \{0, 1, \ldots, b-1\}$ is $O((t/b)n_l \log n_l \log t)$, where n_l denotes the number of the elements belonging to said congruence class. Combining these classes takes $O(bt \log t)$ time so the final complexity is $O(t \log t(\frac{n \log n}{b} + b))$. Setting $b = \lfloor \sqrt{n \log n} \rfloor$ gives $O(\sqrt{n \log n}\, t \log t) = \tilde{O}(\sqrt{n}\, t)$. After combining the sums in the final step, we end up with a polynomial that contains each realisable sum represented by a monomial, the coefficient of which represents the number of different (not necessarily disjoint) subsets that sum up to the corresponding sum.

Our algorithm begins by using the modified characteristic polynomials we proposed in the preliminaries, thereby representing each $z \in Z$ as $\Sigma_{i=1}^{k} x_i^z$ in the base polynomial at the leaves of the recursion. We also use additional dimensions for the cardinalities of the subsets involved; each cardinality is represented by the exponent of some x_i, with index i greater than k. We then proceed with using multi-variate FFT in each step of the recursion in an analogous manner as in the original algorithm, thereby producing polynomials with terms $x_1^{t'_1} \ldots x_k^{t'_k} x_{k+1}^{c_1} \ldots x_{2k}^{c_k}$, each of which corresponds to a $2k$-tuple of disjoint subsets of sums $t'_1, \ldots, t'_k$ and cardinalities $c_1, \ldots, c_k$ respectively.

This results in FFT operations on tuples of $2k$ dimensions, having k dimensions of max value t/b, and another k dimensions of max value n_l which represent the cardinalities of the involved subsets, requiring $O(n_l^k (t/b)^k \log(n_l) \log(n_l t/b))$ time for congruence class $l \in \{0, 1, \ldots, b-1\}$.

The above procedure is implemented in Algorithm 2, while the main algorithm is Algorithm 3, which combines the results from each congruence class, taking additional $O(bt^k \log t)$ time.

Lemma 2 *There is a term $x_1^{t'_1} \ldots x_k^{t'_k}$ in the polynomial returned by Algorithm 3 if and only if there exist k disjoint subsets $Z_1, \ldots, Z_k \subseteq Z$ such that $\Sigma(Z_i) = t'_i$.*

Proof We observe that each of the terms of the form $x_1^{t_1'} \ldots x_k^{t_k'}$ has been produced at some point of the recursion via an FFT operation combining two terms that belong to different subtrees, ergo containing different elements in each subset involved. As such, $t_1', \ldots, t_k'$ are sums of disjoint subsets of Z. □

Theorem 1 *Given a set $Z = \{z_1, \ldots, z_n\} \subseteq \mathbb{N}$ of size n and targets $t_1, \ldots, t_k$, Algorithm 3 can decide k-SUBSET SUM in time $\tilde{O}(n^{k/(k+1)} t^k)$, where $t = \max\{t_1, \ldots, t_k\}$.*

Proof The correctness of the algorithm stems from Lemma 2.

Complexity The overall complexity of the algorithm, stemming from the computation of subset sums inside the congruence classes and the combination of those sums, is

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

where the right-hand side is obtained by setting $b = \sqrt[k+1]{n^k \log n}$. □

Algorithm 2 `DisjointSC(S, t)`

Input : A set S of n positive integers and an upper bound integer t.
Output : The set $Z \subseteq (\mathcal{S}_t(S))^k \times [n]^k$ 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**
2:     **return** $\{0\}^{2k} \cup \{(s, \underbrace{0, \ldots, 0}_{k-1}, 1, \underbrace{0, \ldots, 0}_{k-1})\} \cup \ldots \cup \{(\underbrace{0, \ldots, 0}_{k-1}, s, \underbrace{0, \ldots, 0}_{k-1}, 1)\}$
3: $T \leftarrow$ an arbitrary subset of S of size $\lfloor \frac{n}{2} \rfloor$
4: **return** `DisjointSC`$(T, t) \oplus$ `DisjointSC`$(S \setminus T, t)$

Algorithm 3 `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:     $S_l \leftarrow Z \cap \{x \in \mathbb{N} \mid x \equiv l \pmod{b}\}$
4:     $Q_l \leftarrow \{\lfloor x/b \rfloor \mid x \in S_l\}$
5:     $\mathcal{R}(Q_l) \leftarrow$ `DisjointSC`$(Q_l, \lfloor t/b \rfloor)$
6:     $R_l \leftarrow \{(z_1 b + j_1 l, \ldots, z_k b + j_k l) \mid (z_1, \ldots, z_k, j_1, \ldots, j_k) \in \mathcal{R}(Q_l)\}$
7: **return** $R_0 \oplus_t \cdots \oplus_t R_{b-1}$

3.2 Solving k-subset sum in randomised $\tilde{O}(n + t^k)$ time

We will show that one can decide k-SUBSET SUM in $\tilde{O}(n + t^k)$ time, where $t = \max\{t_1, \ldots, t_k\}$, by building upon the techniques used in Bringmann (2017). In particular, we will present an algorithm that successfully detects, with high probability, whether there exist k disjoint subsets each summing up to t_i respectively. In comparison to the algorithm of Bringmann (2017), a couple of modifications are required, which we will first briefly summarise prior to presenting the complete algorithm.

- In our version of `ColorCoding`, the number of repetitions of random partitioning is increased, without asymptotically affecting the complexity of the algorithm, since it remains $O(\log(1/\delta))$.
- In our version of `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, except that now 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 realisable sums by 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.

3.2.1 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, given an integer c, 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 with high probability whether there exists a tuple $(t_1, \ldots, t_k)$ among them, for some given values t_i.

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[2] $\Sigma(Y)$. 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, the question is to determine 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 c^2 different bins, as mentioned in Bringmann (2017). Next, we compute the probability to split k random disjoint subsets $Y_1, \ldots, Y_k$ in the same partition.

---

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

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.

Algorithm 4 `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 $|Y_1|, \ldots, |Y_k| \leq c$.
1: $S \leftarrow \emptyset$
2: $\beta \leftarrow 4^k/(4^k - 1)$
3: **for** $j = 1, \ldots, \lceil \log_\beta(1/\delta) \rceil$ **do**
4:   randomly partition $Z = Z_1 \cup Z_2 \cup \cdots \cup Z_{c^2}$
5:   **for** $i = 1, \ldots, c^2$ **do**
6:     $Z_i' \leftarrow (Z_i \times \{0\}^{k-1}) \cup \ldots \cup (\{0\}^{k-1} \times Z_i)$
7:   $S_j \leftarrow Z_1' \oplus_t \cdots \oplus_t Z_{c^2}'$
8:   $S \leftarrow S \cup S_j$
9: **return** S

Lemma 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 $O(c^2 \log(1/\delta) t^k \log t)$ time.*

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 produces the k-tuple.

Complexity The algorithm performs $O(\log(1/\delta))$ repetitions. To compute a pairwise sum of k variables up to t, $O(t^k \log t)$ time is required. In each repetition,

c^2 pairwise sums are computed. Hence, the total complexity of the algorithm is $O(c^2 \log(1/\delta)t^k \log t)$.   □

3.2.2 Solving k-subset sum for l-layer instances

In this part, we will prove that we can use the algorithm `ColorCodingLayer` from Bringmann (2017), to successfully solve k-SUBSET SUM for l-layer instances, defined below.

For $l \geq 1$, we call (Z, t) an l-*layer instance* if $Z \subseteq [t/l, 2t/l]$ 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 Bringmann (2017) successfully solves the SUBSET SUM problem for l-layer instances. We will show that by calling the modified `ColorCoding` algorithm and modifying the FFT operations so that they concern sets of points, the algorithm successfully solves k-SUBSET SUM in such instances.

Algorithm 5 `ColorCodingLayer`(Z, t, l, δ)

Input : An l-layer instance (Z, t) and an error probability $\delta \in (0, 1/2^{k+1}]$.
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 $\Sigma(Y_1), \ldots, \Sigma(Y_k) \leq t$.
1: **if** $l < \log(l/\delta)$ **then return** `ColorCoding`(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:    $S_j \leftarrow$ `ColorCoding`$(Z_j, 2\gamma t/l, \gamma, \delta/l)$
7: **for** $h = 1, \ldots, \log m$ **do**   ▷ combine S_j in a binary-tree-like way
8:    **for** $j = 1, \ldots, m/2^h$ **do**
9:        $S_j \leftarrow S_{2j-1} \oplus_{2^h \cdot 2\gamma t/l} S_{2j}$
10: **return** $S_1 \cap [t]^k$

We will now prove the correctness of the algorithm. Let $X^1, \ldots, X^k \subseteq Z$ be disjoint subsets with $\Sigma(X^1), \ldots, \Sigma(X^k) \leq t$. By Bringmann (2017), Claim 3.3, 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 $|X_i^j| \leq 6\log(l/\delta)$, for all $i = 1, \ldots, m$ and $j = 1, \ldots, k$ is

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

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

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

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

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

Combining the resulting sets costs

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

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

$$= O\left(\frac{t^k \log t}{m^{k-1}} \left(2^{k-1}\right)^{\log m}\right)$$

$$= O(t^k \log t),$$

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

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

3.2.3 General case

It remains to show that for every instance (Z, t), we can construct l-layer instances and take advantage of `ColorCodingLayer` in order 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 $O(\log n)$ l-layers $Z_1, \ldots, Z_{\lceil \log n \rceil}$. On each layer we run `ColorCodingLayer` and subsequently combine the results using pairwise sums.

We will now prove the main theorem of this section.

Theorem 2 *Given a set* $Z \subseteq \mathbb{N}$ *of size n and targets* $t_1, \ldots, t_k$, *one can decide k-*SUBSET SUM *in* $\tilde{O}(n + t^k)$ *time w.h.p., where* $t = \max\{t_1, \ldots, t_k\}$.

Algorithm 6 kSubsetSum(Z, δ, t)

Input : A set of positive integers Z of cardinality n, an upper bound t and an error probability δ.
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 $\Sigma(Y_1), \ldots, \Sigma(Y_k) \le t$.
1: 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 $Z_{\lceil \log n \rceil} \leftarrow Z \cap [0, t/2^{\lceil \log n \rceil - 1}]$
2: $S \leftarrow \emptyset$
3: **for** $i = 1, \ldots, \lceil \log n \rceil$ **do**
4:     $S_i \leftarrow$ ColorCodingLayer$(Z_i, t, 2^i, \delta/\lceil \log n \rceil)$
5:     $S \leftarrow S \oplus_t S_i$
6: **return** S

Proof Let $X^1, \ldots, X^k \subseteq Z$ be k disjoint subsets 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

$$\Pr[\text{ColorCodingLayer returns}(\Sigma(X_i^1), \ldots, \Sigma(X_i^k)), \forall i]$$

$$= 1 - \Pr[\text{some call fails}] \ge 1 - \sum_{i=1}^{\lceil \log n \rceil} \frac{\delta}{\lceil \log n \rceil}$$

$$= 1 - \lceil \log n \rceil \cdot \frac{\delta}{\lceil \log n \rceil} = 1 - \delta$$

If all calls return the corresponding k-tuple, the algorithm successfully constructs 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. In each of the $\Theta(\log n)$ repetitions of the algorithm, we make a call to ColorCodingLayer, plus compute a pairwise sum. The computation of the pairwise sum requires $O(t^k \log t)$ time since it concerns k-tuples. For each call to ColorCodingLayer, we require

$$O\left(t^k \log t \frac{\log^{k+2}\left(\frac{2^i \log n}{\delta}\right)}{2^{i(k-1)}}\right) \text{ time. Hence, the overall complexity is}$$

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

4 *k*-subset sum with cardinality constraints

The algorithms presented in the previous section can be extended to a variation of the k-SUBSET SUM problem, where along with the target values, the cardinality of the respective solution sets is given as part of the input.

Therefore, consider $c_i, i = 1, \ldots, k$ such that $\sum c_i \leq n$, to be the desired cardinality of set Z_i, where $\Sigma(Z_i) = t_i$, and let $c = \max\{c_1, \ldots, c_k\}$. We seek to determine whether there exist disjoint subsets $Z_i \subseteq Z, i = 1, \ldots, k$ of sum $\Sigma(Z_i) = t_i$ and cardinality $|Z_i| = c_i$.

As it will become evident, the previously proposed algorithms for k-SUBSET SUM can be slightly modified in order to cope with this additional restriction. The main idea lies on the addition of extra variables in the resulting polynomial, each of which represents the cardinality of one of the solution sets. This results in a blowup of the order of $O(c^k)$ in the final complexity of the algorithms, since a variable is added for each of the k subsets of the solution.

4.1 A deterministic approach

In the case of the deterministic algorithm proposed in Sect. 3.1, it suffices to modify the procedure in line 6 of Algorithm 3.

Here, instead of simply reconstructing the actual partial sums obtained from elements belonging to the same congruence class, it is necessary to additionally keep the relevant cardinality information. Subsequently, the final FFT operation takes into account this information in order to verify the existence of a solution, thus requiring additional time due to the extra variables.

Theorem 3 *Given a set $Z = \{z_1, \ldots, z_n\} \subseteq \mathbb{N}$ of size n, targets $t_1, \ldots, t_k$ and cardinalities $c_1, \ldots, c_k$, Algorithm 7 can decide k-SUBSET SUM with cardinality constraints in time $\tilde{O}(\sqrt[k+1]{n^k c^{k^2} t^k})$, where $t = \max\{t_1, \ldots, t_k\}$ and $c = \max\{c_1, \ldots, c_k\}$.*

Proof The correctness of the algorithm can be shown by analogous arguments as those used in Lemma 2. In particular, in this case the subset cardinalities computed in each congruence class are kept in order to verify the final cardinality of the solution subsets obtained from the addition of elements belonging in different congruence classes.

Complexity The overall complexity of the algorithm, stemming from the computation of subset sums inside the congruence classes and the combination of those sums, is

$$O\left(\frac{n^k \log n}{b^k} t^k \log t + b t^k c^k \log(tc)\right) = O\left(\frac{n^k \log n}{b^k} t^k \log t + b t^k c^k \log t\right)$$
$$= O\left(t^k \log t \left(\frac{n^k \log n}{b^k} + bc^k\right)\right)$$
$$= \tilde{O}(\sqrt[k+1]{n^k c^{k^2} t^k}),$$

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

4.2 A randomised approach

With slight modifications, it is possible to successfully adapt the algorithm of Sect. 3.2 in order to consider the additional cardinality constraints of this section.

Algorithm 7 `CardDisjointSS`(Z, t)

Input : A set Z of n positive integers, an upper bound integer t and a size bound $c \geq 1$.
Output : The set $S \subseteq (\mathcal{S}_t(Z) \times [c])^k$ of all k-tuples of subset sums up to t along with their respective cardinality on each dimension, occurring from disjoint subsets of Z with cardinality up to c.

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

More specifically, there are two important remarks that lead to those modifications:

1. It is necessary to verify the cardinalities of the involved subsets in the resulting polynomial. This can be achieved by introducing additional variables, similar to 4.1.
2. Whereas in the algorithm of Sect. 3.2 there was a necessity of partitioning the input to layers in order to associate the cardinality of a subset with its target value, in this case such a procedure is redundant, since the upper bound on the cardinality of the sets is part of the input.

Following these remarks, we firstly modify the `ColorCoding` algorithm of the previous section, in order to additionally return cardinality information for the involved subsets. This does not affect the correctness of the algorithm whatsoever and only incurs a blowup in the final complexity.

Algorithm 8 `CardColorCoding`(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 $S \subseteq (\mathcal{S}_t(Z))^k \times [c]^k$ such that it contains any tuple $(\Sigma(Y_1), \ldots, \Sigma(Y_k), |Y_1|, \ldots, |Y_k|)$ with probability at least $1 - \delta$, where $Y_1, \ldots, Y_k \subseteq Z$ disjoint subsets of Z with $\Sigma(Y_1), \ldots, \Sigma(Y_k) \leq t$ and $|Y_1|, \ldots, |Y_k| \leq c$.

1: $S \leftarrow \emptyset$
2: $\beta \leftarrow 4^k/(4^k - 1)$
3: **for** $j = 1, \ldots, \lceil \log_\beta (1/\delta) \rceil$ **do**
4:     randomly partition $Z = Z_1 \cup Z_2 \cup \cdots \cup Z_{c^2}$
5:     **for** $i = 1, \ldots, c^2$ **do**
6:         $Z_i' \leftarrow (Z_i \times \{0\}^{k-1} \times \{1\} \times \{0\}^{k-1}) \cup \ldots \cup (\{0\}^{k-1} \times Z_i \times \{0\}^{k-1} \times \{1\})$
7:     $S_j \leftarrow Z_1' \oplus_t \cdots \oplus_t Z_{c^2}'$
8:     $S \leftarrow S \cup S_j$
9: **return** S

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

Algorithm description   In this paragraph, we briefly describe the algorithm that successfully solves k-SUBSET SUM with cardinality constraints. The proposed algorithm is called `CardColorCodingLayer` and is identical to Algorithm 5 of Sect. 3.2 with the following modifications:

(a) The calls to the `ColorCoding` algorithm on line 6 actually consider the `CardColorCoding` algorithm previously described, hence the cardinality information is included.

(b) The FFT operations now involve polynomials of $2k$ variables, since an additional variable is considered for each subset, depicting its cardinality.

Thus, in contrast to Sect. 3.2 where we had to run a `ColorCodingLayer` algorithm for each distinct layer, in this case, it suffices to run `CardColorCodingLayer` (Z, t, c, δ).

Theorem 4 *Given a set $Z = \{z_1, \ldots, z_n\} \subseteq \mathbb{N}$ of size n, targets $t_1, \ldots, t_k$ and cardinalities $c_1, \ldots, c_k$, as well as an error probability $\delta \in (0, 1/2^{k+1}]$, Algorithm* `CardColorCodingLayer` *can decide k-SUBSET SUM with cardinality constraints with probability at least $1 - \delta$ in time $\tilde{O}(n + t^k c^k)$, where $t = \max\{t_1, \ldots, t_k\}$ and $c = \max\{c_1, \ldots, c_k\}$.*

Proof The correctness analysis is not affected by the aforementioned modifications, thus we only have to compute the complexity of the modified algorithm.

Complexity In order to read the input set, $O(n)$ time is required. The cost of the m calls to `CardColorCoding` is

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

$$= O\left(\frac{\log^{2k+2}(c/\delta)}{c^{k-1}} t^k \log t\right).$$

Combining the resulting sets costs

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

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

$$= O\left(\frac{t^k c^k \log t}{m^{k-1}} \left(2^{k-1}\right)^{\log m}\right)$$

$$= O(t^k c^k \log t),$$

since for $c > 1$, it holds that $O\left(\sum_{k=0}^{n} c^k\right) = O(c^n)$.

Thus, the total complexity is

$$O\left(n + t^k \log t \left(c^k + \frac{\log^{2k+2}(c/\delta)}{c^{k-1}}\right)\right) = \tilde{O}(n + t^k c^k)$$

$\square$

5 Faster algorithms for multiple subset problems

The techniques developed in Sects. 3 and 4 can be further applied to give faster pseudopolynomial algorithms for the decision version of the problems 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 $\Sigma(Z_1) = s_1, \ldots, \Sigma(Z_k) = s_k$. Hereinafter, when we refer to a solution of a k-SUBSET SUM input, we actually refer to this resulting polynomial, unless explicitly stated otherwise.

It is important to note that, while the deterministic algorithm of Sect. 3.1 returns a polynomial consisting of *all* terms corresponding to such k-tuples of realisable sums by disjoint subsets, the randomised algorithm of Sect. 3.2 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.

5.1 Subset sum ratio

The first variation we will discuss is SUBSET SUM RATIO, which asks 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 $\tilde{O}(n^{2/3}t^2)$ time using the algorithm proposed in Sect. 3.1 by simply iterating over the terms of the final polynomial that involve both parameters x_1 and x_2 and checking the ratio of their exponents. SUBSET SUM RATIO can also be solved with high probability in randomised $\tilde{O}(n + t^2)$ time using the algorithm proposed in Sect. 3.2 instead.

5.2 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 $\tilde{O}(n^{k/(k+1)} \cdot t^k)$ or randomised $\tilde{O}(n + t^k)$ 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.

5.3 k-way number partitioning

Another closely related problem called k-WAY NUMBER PARTITIONING asks, given a set $Z \subseteq \mathbb{N}$ of size n, to partition its elements into k subsets $Z_1, \ldots, Z_k$, whose sums $\Sigma(Z_i)$ are as similar as possible. There are multiple different objective functions which may be used in order to define this concept of similarity. For instance, one could ask for the minimum difference between the largest and smallest sums, the minimum ratio between the largest and smallest sums, the maximum possible smallest sum or alternatively the minimum possible largest sum. Note that each of these objective functions may lead to a different solution (Korf 2010).

In order to solve this problem for any of the previously mentioned objective functions, it suffices to solve k-SUBSET SUM for $t = \max(Z) + (\Sigma(Z)/k)$, where $\max(Z)$ denotes the largest element of Z, while only considering the terms $x_1^{s_1} \ldots x_k^{s_k}$ of the final polynomial for which $\sum s_i = \Sigma(Z)$. This holds due to the fact that any candidate solution involves sets of subset sum at most equal to t, since if there exists a solution involving a subset $Z' \subseteq Z$ with $\Sigma(Z') > t$, that means that in the same solution, there exists a subset $Z'' \subseteq Z$ such that $\Sigma(Z'') < \Sigma(Z)/k$, on which case, removing any element from set Z' and adding it to the set Z'' results in a potentially better solution, irrespective of the objective function used to define the concept of similarity.

5.4 Multiple subset sum

Finally, we consider the MULTIPLE SUBSET SUM problem that 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 set 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 there is a simple reduction of k-SUBSET SUM to it. It should be clear that the same techniques as those e.g. used for k-SUBSET SUM RATIO apply directly to MULTIPLE SUBSET SUM, leading to the same time complexity bounds of $\tilde{O}(n^{k/(k+1)} \cdot t^k)$ deterministically and $\tilde{O}(n + t^k)$ probabilistically.

5.5 Adding cardinality constraints

Note that for all of the presented algorithmic problems, if specific cardinalities for the subsets involved in a solution are required, then we can obtain a solution by applying the algorithms of Sect. 4 in an analogous manner. In this case, the resulting polynomial

is of $2k$ variables, k of which are used to specify the cardinality of the subsets. Thus, by checking those variables, it is possible to obtain a solution in this more restricted version of the problems.

5.6 Counting versions

In the deterministic version of the algorithm, one can see that the coefficients in the resulting polynomial correspond to the number of different solutions for each problem. As such, the same techniques can be applied in order to solve the counting versions of these problems. On the other hand, in the randomised version, more probabilistic analysis is required in order to guarantee the correct computation of each count.

6 Future work

6.1 Possible optimisations regarding FFT

The algorithm of Sect. 3.1 as well as those presented in Sect. 4 involve the computation of the possible sums by disjoint subsets along with their respective cardinality. 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 $x_1, \ldots, x_k, x_{k+1}, \ldots, x_{2k}$, where the exponents of $x_i, i \leq k$ are in $[t]$ for some given upper bound t, whereas the exponents of $x_i, i > k$ in $[n]$. Notice however that each element is used only on a single subset, hence for a term $x_1^{s_1} \ldots x_k^{s_k} x_{k+1}^{n_1} \ldots x_{2k}^{n_k}$ of the produced polynomial, it holds that $s_i \leq t$ 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 algorithm. However, although this improvement might be useful for practical applications, no asymptotic improvement can be achieved via this route.

Also notice that, in our proposed algorithms, 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 increase on the number of terms does not influence the asymptotic analysis of our algorithms, nevertheless can be restricted (e.g. by sorting and pruning) 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 paper, 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 $\tilde{O}(n^{k/(k+1)} T)$ and $\tilde{O}(n + T)$ for the deterministic and the randomised algorithm respectively, where $T = \prod t_i$. These can be further extended to the case where we additionally have cardinality constraints.

It remains to be seen whether there is a corresponding lower bound for the problem based on popular conjectures such as (S)ETH; on the other hand, it seems that further improvement would require a different algorithmic approach.

6.2 Extension of randomised algorithm for counting version

As we discussed in Sect. 5, one can use the same techniques as in the deterministic version in order to solve the counting versions of each problem respectively. In the case of the randomised algorithm, it is possible to obtain an approximation of these counts (e.g. by using repetitions to compute the distribution of realisable sums), but it remains open whether one can find the exact count (w.h.p) more efficiently.

6.3 Recovery of solution sets

The algorithms introduced in this paper 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 (and their corresponding cardinalities are equal to $c_1, \ldots, c_k$ in the cardinality constrained version of the problem). An extension of these algorithms could involve the reconstruction of the k solution subsets. Koiliaris and Xu (2019) 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 Aumann et al. (2011), 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.

6.4 Extension of subset sum algorithm introduced by Jin and Wu

Jin and Wu introduced an efficient $\tilde{O}(n+t)$ randomised algorithm for solving Subset Sum in Jin and Wu (2018). This algorithm is much simpler than Bringmann's, while actually providing a slightly better upper bound for the problem. It is interesting to investigate whether this algorithm can be extended to cope with k-Subset Sum (and the variations mentioned in Sect. 5), as was the case for Bringmann's algorithm, since that would result in a simpler alternative approach.

Data availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

References

Abboud A, Bringmann K, Hermelin D, Shabtay D (2022) Seth-based lower bounds for subset sum and bicriteria path. ACM Trans Algorithms 18(1):6. https://doi.org/10.1145/3450524

Alonistiotis G, Antonopoulos A, Melissinos N, Pagourtzis A, Petsalakis S, Vasilakis M (2022) Approximating subset sum ratio via subset sum computations. In: Combinatorial algorithms—33rd international workshop, IWOCA 2022. Lecture Notes in computer science, vol 13270, pp 73–85. Springer, Cham. https://doi.org/10.1007/978-3-031-06678-8_6

Antonopoulos A, Pagourtzis A, Petsalakis S, Vasilakis M (2021) Faster algorithms for k-subset sum and variations. In: Frontiers of algorithmics, IJTCS-FAW 2021. Lecture notes in computer science, vol 12874, pp 37–52. Springer, Cham. https://doi.org/10.1007/978-3-030-97099-4_3

Aumann Y, Lewenstein M, Lewenstein N, Tsur D (2011) Finding witnesses by peeling. ACM Trans Algorithms 7(2):1–15. https://doi.org/10.1145/1921659.1921670

Bazgan C, Santha M, Tuza Z (2002) Efficient approximation algorithms for the SUBSET-SUMS EQUALITY problem. J Comput Syst Sci 64(2):160–170. https://doi.org/10.1006/jcss.2001.1784

Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton

Bringmann K (2017) A near-linear pseudopolynomial time algorithm for subset sum. In: Proceedings of the twenty-eighth annual ACM-SIAM symposium on discrete algorithms, SODA 2017, pp 1073–1084. SIAM, Philadelphia. https://doi.org/10.1137/1.9781611974782.69

Bringmann K, Nakos V (2020) Top-k-convolution and the quest for near-linear output-sensitive subset sum. In: Proceedings of the 52nd annual ACM SIGACT symposium on theory of computing, STOC 2020, pp 982–995. ACM, New York. https://doi.org/10.1145/3357713.3384308

Caprara A, Kellerer H, Pferschy U (2000) A PTAS for the multiple subset sum problem with different knapsack capacities. Inf Process Lett 73(3–4):111–118. https://doi.org/10.1016/S0020-0190(00)00010-7

Caprara A, Kellerer H, Pferschy U (2003) A 3/4-approximation algorithm for multiple subset sum. J Heuristics 9(2):99–111. https://doi.org/10.1023/A:1022584312032

Cieliebak M, Eidenbenz SJ (2004) Measurement errors make the partial digest problem NP-hard. In: LATIN 2004: theoretical informatics, 6th Latin American symposium. Lecture notes in computer science, vol 2976, pp 379–390. Springer, Berlin. https://doi.org/10.1007/978-3-540-24698-5_42

Cieliebak M, Eidenbenz SJ, Pagourtzis A (2003a) Composing equipotent teams. In: Fundamentals of computation theory, 14th international symposium, FCT 2003. Lecture notes in computer science, vol 2751, pp 98–108. Springer, Berlin. https://doi.org/10.1007/978-3-540-45077-1_10

Cieliebak M, Eidenbenz SJ, Penna P (2003b) Noisy data make the partial digest problem NP-hard. In: Algorithms in bioinformatics, third international workshop, WABI 2003. Lecture notes in computer science, vol 2812, pp 111–123. Springer, Berlin. https://doi.org/10.1007/978-3-540-39763-2_9

Cieliebak M, Eidenbenz SJ, Pagourtzis A, Schlude K (2008) On the complexity of variations of equal sum subsets. Nord J Comput 14(3):151–172

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms, 3rd edn. The MIT Press, London

Dell'Amico M, Delorme M, Iori M, Martello S (2019) Mathematical models and decomposition methods for the multiple Knapsack problem. Eur J Oper Res 274(3):886–899. https://doi.org/10.1016/j.ejor.2018.10.043

Jin C, Wu H (2018) A simple near-linear pseudopolynomial time randomized algorithm for subset sum. In: 2nd Symposium on simplicity in algorithms (SOSA 2019), vol 69, pp 17. https://doi.org/10.4230/OASIcs.SOSA.2019.17

Karp RM (1972) Reducibility among combinatorial problems. In: Complexity of computer computations. The IBM research symposia series, pp 85–103. Springer, Boston. https://doi.org/10.1007/978-1-4684-2001-2_9

Khan MA (2017) Some problems on graphs and arrangements of convex bodies. PRISM. https://doi.org/10.11575/PRISM/10182

Koiliaris K, Xu C (2019) Faster pseudopolynomial time algorithms for subset sum. ACM Trans Algorithms 15(3):40. https://doi.org/10.1145/3329863

Korf RE (2010) Objective functions for multi-way number partitioning. In: Proceedings of the third annual symposium on combinatorial search, SOCS 2010. Stone Mountain, Atlanta, July 8–10, 2010. AAAI Press

Lahyani R, Chebil K, Khemakhem M, Coelho LC (2019) Matheuristics for solving the multiple knapsack problem with setup. Comput Ind Eng 129:76–89. https://doi.org/10.1016/j.cie.2019.01.010

Lipton RJ, Markakis E, Mossel E, Saberi A (2004) On approximately fair allocations of indivisible goods. In: Proceedings of the 5th ACM conference on electronic commerce (EC-2004), pp 125–131. ACM, New York. https://doi.org/10.1145/988772.988792

Melissinos N, Pagourtzis A (2018) A faster FPTAS for the subset-sums ratio problem. In: Computing and combinatorics—24th international conference, COCOON 2018. Lecture notes in computer science, vol 10976, pp 602–614. Springer, Cham. https://doi.org/10.1007/978-3-319-94776-1_50

Mucha M, Nederlof J, Pawlewicz J, Wegrzycki K (2019) Equal-subset-sum faster than the meet-in-the-middle. In: 27th Annual European symposium on algorithms, ESA. LIPIcs, vol 144, pp 73 (2019). https://doi.org/10.4230/LIPIcs.ESA.2019.73

Nanongkai D (2013) Simple FPTAS for the subset-sums ratio problem. Inf Process Lett 113(19–21):750–753. https://doi.org/10.1016/j.ipl.2013.07.009

Papadimitriou CH (1994) On the complexity of the parity argument and other inefficient proofs of existence. J Comput Syst Sci 48(3):498–532. https://doi.org/10.1016/S0022-0000(05)80063-7

Pisinger D (1999) Linear time algorithms for knapsack problems with bounded weights. J Algorithms 33(1):1–14. https://doi.org/10.1006/jagm.1999.1034

Recalde D, Severín D, Torres R, Vaca P (2018) An exact approach for the balanced k-way partitioning problem with weight constraints and its application to sports team realignment. J Comb Optim 36(3):916–936. https://doi.org/10.1007/s10878-018-0254-1

Schreiber EL, Korf RE, Moffitt MD (2018) Optimal multi-way number partitioning. J ACM 65(4):24. https://doi.org/10.1145/3184400

Tsai L (1992) Asymptotic analysis of an algorithm for balanced parallel processor scheduling. SIAM J Comput 21(1):59–64. https://doi.org/10.1137/0221007

Voloch N (2017) MSSP for 2-d sets with unknown parameters and a cryptographic application. Contemp Eng Sci 10:921–931. https://doi.org/10.12988/ces.2017.79101

Woeginger GJ, Yu Z (1992) On the equal-subset-sum problem. Inf Process Lett 42(6):299–302. https://doi.org/10.1016/0020-0190(92)90226-L