A Faster Pseudopolynomial Time Algorithm for Subset Sum

Konstantinos Koiliaris[*]   Chao Xu[†]

Abstract

Given a multiset S of n positive integers and a target integer t, the subset sum problem is to decide if there is a subset of S that sums up to t. We present a new divide-and-conquer algorithm that computes *all* the realizable subset sums up to an integer u in $\widetilde{O}(\min\{\sqrt{n}u, u^{4/3}, \sigma\})$, where σ is the sum of all elements in S and $\widetilde{O}$ hides polylogarithmic factors. This result improves upon the standard dynamic programming algorithm that runs in $O(nu)$ time. To the best of our knowledge, the new algorithm is the fastest general deterministic algorithm for this problem. We also present a modified algorithm for finite cyclic groups, which computes all the realizable subset sums within the group in $\widetilde{O}(\min\{\sqrt{n}m, m^{5/4}\})$ time, where m is the order of the group.

1. Introduction

Given a multiset S of n positive integers and an integer target value t, the *subset sum problem* is to decide if there is a subset of S that sums to t. The subset sum problem is related to the knapsack problem [11] and it is one of Karp's original NP-complete problems [25]. The subset sum is a fundamental problem used as a standard example of a problem that can be solved in weakly polynomial time in many undergraduate algorithms/complexity classes. As a weakly NP-complete problem, there is a standard pseudopolynomial time algorithm using a dynamic programming, due to Bellman, that solves it in $O(nt)$ time [2] (see also [9, Chapter 34.5]). The current state of the art has since been improved by a $\log t$ factor using a bit-packing technique [32]. There is extensive work on the subset sum problem, see Table 1.1 for a summary of previous deterministic pseudopolynomial time results [2, 33, 15, 31, 27, 32, 29, 37, 38].

Moreover, there are results on subset sum that depend on properties of the input, as well as data structures that maintain subset sums under standard operations. In particular, when the maximum value of any integer in S is relatively small compared to the number of elements n, and the target value t lies close to one-half the total sum of the elements, then one can solve the subset sum problem in almost linear time [16]. This was improved by Chaimovich [7]. Furthermore, Eppstein described a data structure which efficiently maintains all subset sums up to a given value u, under insertion and deletion of elements, in $O(u \log u \log n)$ time per update, which can be accelerated to $O(u \log u)$ when additional information about future updates is known [14]. The probabilistic convolution tree, by Serang [37, 38], is also able to solve the subset sum problem in $\widetilde{O}(n \max(S))$ time, where $\widetilde{O}$ hides polylogarithmic factors.

If randomization is allowed, more algorithms are possible. In particular, Bringmann showed a randomized algorithm that solves the problem in $\widetilde{O}(nt)$ time, using only $\widetilde{O}(n \log t)$ space under the Extended Riemann Hypothesis [4]. Bringmann also provided a randomized near linear time algorithm $\widetilde{O}(n + t)$ – it remains open whether this algorithm can be derandomized.

---

[*]Department of Computer Science, University of Illinois, Urbana - Champaign. E-mail: `koiliar2@illinois.edu`

[†]Department of Computer Science, University of Illinois, Urbana - Champaign. E-mail: `chaoxu3@illinois.edu`

Result	Time	Space	Comments
Bellman [2]	$O(nt)$	$O(t)$	original DP solution
Pisinger [32]	$O\left(\frac{nt}{\log t}\right)$	$O\left(\frac{t}{\log t}\right)$	RAM model implementation of Bellman
Pisinger [33]	$O(n \max S)$	$O(t)$	fast if small $\max S$
Faaland [15], Pferschy [31]	$O(n't)$	$O(t)$	fast for small n'
Klinz et al. [27]	$O(\sigma^{3/2})$	$O(t)$	fast for small σ, obtainable from above because $n' = O\left(\sqrt{\sigma}\right)$
Eppstein [14], Serang [37, 38]	$\widetilde{O}(n \max S)$	$O(t \log t)$	data structure
Lokshtanov et al. [29]	$\widetilde{O}(n^3 t)$	$\widetilde{O}(n^2)$	polynomial space
current work	$\widetilde{O}\left(\min\left\{\sqrt{n'}\,t, t^{4/3}, \sigma\right\}\right)$ Theorem 2.17	$O(t)$	see Section 1.2

Table 1.1: Summary of deterministic pseudopolynomial time results on the subset sum problem. The input is a target number t and a multiset S of n numbers, with n' distinct values up to t, and σ denotes the sum of all elements in S.

Finally, it is unlikely that any subset sum algorithm runs in time $O(t^{1-\epsilon}\,n^c)$, for any constant c and $\epsilon > 0$, as such an algorithm would imply that there are faster algorithms for a wide variety of problems including set cover [4, 10].

1.1. Applications of the subset sum problem.

The subset sum problem has a variety of applications including: power indices [42], scheduling [17, 34, 19], set-based queries in databases [41], breaking precise query protocols [12] and various other graph problems with cardinality constraints [6, 13, 5, 18, 27, 14] (for a survey of further applications see [26]).

A faster pseudopolynomial time algorithm for the subset sum would imply faster *polynomial time* algorithms for a number of problems. The *bottleneck graph partition* problem on weighted graphs is one such example. It asks to split the vertices of the graph into two equal-sized sets such that the value of the bottleneck (maximum-weight) edge, over all edges across the cut, is minimized. The impact of our results on this problem and other selected applications is highlighted in Section 5.

1.2. Our contributions.

The new results are summarized in Table 1.2 – we consider the following *all subset sums* problem: Given a multiset S of n elements, with n' distinct values, with σ being the total sum of its elements, compute *all* the realizable subset sums up to a prespecified integer u. Computing all subset sums for some $u \geq t$ also answers the standard subset sum problem with target value t.

Our main contribution is a new algorithm for computing the all subset sums problem in $\widetilde{O}\big(\min\{\sqrt{n}u, u^{4/3}, \sigma\}\big)$ time. The new algorithm improves over all previous work (see Table 1.2). To the best of our knowledge, it is the fastest general deterministic pseudopoly-

Parameters	Previous best	Current work
n and t	$O(nt/\log t)$	$\widetilde{O}\big(\min\big\{\sqrt{n}t,\, t^{4/3}\big\}\big)$
n' and t	$O(n't)$	$\widetilde{O}\big(\min\big\{\sqrt{n'}t,\, t^{4/3}\big\}\big)$
σ	$O(\sigma^{3/2})$	$\widetilde{O}(\sigma)$

Table 1.2: Our contribution on the subset sum problem compared to the previous best known results. The input S is a multiset of n numbers with n' distinct values, σ denotes the sum of all elements in S and t is the target number.

nomial time algorithm for the all subset sum problem, and consequently, for the subset sum problem.

Our second contribution is an algorithm that solves the *all subset sums* problem modulo m, in $O\big(\min\{\sqrt{n}m, m^{5/4}\}\log^2 m\big)$ time. Though the time bound is superficially similar to the first algorithm, this algorithm uses a significantly different approach.

Both algorithms can be augmented to return the solution; i.e., the subset summing up to each number, with a polylogarithmic slowdown (see Section 4 for details).

1.3. Sketch of techniques.

The straightforward divide-and-conquer algorithm for solving the subset sum problem [23], partitions the set of numbers into two sets, recursively computes their subset sums and combines them together using FFT [14, 37, 38] (Fast Fourier Transform [9, Chapter 30]). This algorithm has a running time of $O(\sigma \log \sigma \log n)$.

Sketch of the first algorithm (on integers). Our main new idea is to improve the "conquer" step by taking advantage of the structure of the sets. In particular, if S and T lie in a short interval, then one can combine their subset sums quickly, due to their special structure. On the other hand, if S and T lie in a long interval, but the smallest number of the interval is large, then one can combine their subset sums quickly by ignoring most of the sums that exceed the upper bound.

The new algorithm works by first partitioning the interval $[\![0:u]\!]$ into a logarithmic number of exponentially long intervals. Then computes these partial sums recursively and combines them together by aggressively deploying the above observation.

Sketch of the second algorithm (modulo m). Assume m is a prime number. Using known results from number theory, we show that for any ℓ one can partition the input set into $\widetilde{O}(|S|/\ell)$ subsets, such that every such subset is contained in an arithmetic progression of the form $x, 2x, \ldots, \ell x$. The subset sums for such a set can be quickly computed by dividing and later multiplying the numbers by ℓ. Then combine all these subset sums to get the result.

Sadly, m is not always prime. Fortunately, all the numbers that are relative prime to m can be handled in the same way as above. For the remaining numbers we use a recursive partition classifying each number, in a sieve-like process, according to which prime factors it shares with m. In the resulting subproblems all the numbers are coprime to the moduli used, and as such the above algorithm can be used. Finally, the algorithm combines the subset sums of the subproblems.

Paper organization. Section 2 covers the algorithm for positive integers. Section 3 describes the algorithm for the case of modulo m. Section 4 shows how we can recover the

subsets summing to each set, and Section 5 presents the impact of the results on selected applications of the problem.

2. The algorithm for integers

2.1. Notations.

Let $[\![x : y]\!] = \{x, x+1, \ldots, y\}$ denote the set of integers in the interval $[x, y]$. Similarly, $[\![x]\!] = [\![1 : x]\!]$. For two sets X and Y, we denote by $X \oplus Y$ the set $\{x + y \mid x \in X \text{ and } y \in Y\}$. If X and Y are sets of points in the plane, $X \oplus Y$ is the set $\{(x_1 + y_1, x_2 + y_2) \mid x_1, x_2 \in X \text{ and } y_1, y_2 \in Y\}$.

For an element s in a multiset S, its ***multiplicity*** in S is denoted by $\mathbb{1}_S(s)$. We denote by $\text{set}(S)$ the set of *distinct elements* appearing in the multiset S. The ***size*** of a multiset S is the number of distinct elements in S (i.e., $|\text{set}(S)|$). The ***cardinality*** of S, is $\text{card}(S) = \sum_{s \in S} \mathbb{1}_S(s)$. We denote that a multiset S has all its elements in the interval $[\![x : y]\!]$ by $S \subseteq [\![x : y]\!]$.

For a multiset S of integers, let $\Sigma_S = \sum_{s \in S} \mathbb{1}_S(s) \cdot s$ denote the total ***sum*** of the elements of S. The set of ***all subset sums*** is denoted by

$$\sum(S) = \{\Sigma_T \mid T \subseteq S\} \ .$$

The pair of the set of all subset sums using sets of size at most α along with their associated cardinality is denoted by $\sum^{\leq \alpha}[S] = \{(\Sigma_T, |T|) \mid T \subseteq S, |T| \leq \alpha\}$. The set of all subset sums of a set S up to a number u is denoted by $\sum_{\leq u}(S) = \sum(S) \cap [\![0 : u]\!]$.

2.2. From multisets to sets.

Here, we show that the case where the input is a multiset can be reduced to the case of a set. The reduction idea is somewhat standard (see [26, Section 7.1.1]), and first appeared in [28]. We present it here for completeness.

Lemma 2.1. *Given a multiset S of integers, and a number $s \in S$, with $\mathbb{1}_S(s) \geq 3$. Consider the multiset S' resulting from removing two copies of s from S, and adding the number $2s$ to it. Then, $\sum_{\leq u}(S) = \sum_{\leq u}(S')$. Observe that $\text{card}(S') = \text{card}(S) - 1$.*

Proof: Consider any multiset $T \subseteq S$. If T contains two or more copies of s, then replace two copies by a single copy of $2s$. The resulting subset is $T' \subseteq S'$, and $\Sigma_T = \Sigma_{T'}$, establishing the claim. ∎

Lemma 2.2. *Given a multiset S of integers in $[\![u]\!]$ of cardinality n with n' unique values, one can compute, in $O(n' \log^2 u)$ time, a multiset T, such that: (i) $\sum_{\leq u}(S) = \sum_{\leq u}(T)$,*
(ii) $\text{card}(T) \leq \text{card}(S)$,
(iii) $\text{card}(T) = O(n' \log u)$, and
(iv) no element in T has multiplicity exceeding two.

Proof: Copy the elements of S into a working multiset X. Maintain the elements of $\text{set}(X)$ in a heap D, and let T initially be the empty set. In each iteration, extract the minimum element x from the heap D. If $x > u$, we stop.

If $\mathbb{1}_X(x) \leq 2$, then delete x from X, and add x, with its appropriate multiplicity, to the output multiset T, and continue to the next iteration.

If $\mathbb{1}_X(x) > 2$, then delete x from X, add x to the output set T (with multiplicity one), insert the number $2x$ into X with multiplicity $m' = \lfloor (\mathbb{1}_X(x) - 1)/2 \rfloor$, (updating also the heap

4

D – by adding $2x$ if it is not already in it), and set $\mathbb{1}_X(x) \leftarrow \mathbb{1}_X(x) - 2m'$. The algorithm now continues to the next iteration.

At any point in time, we have that $\sum_{\leq u}(S) = \sum_{\leq u}(X \cup T)$, and every iteration takes $O(\log u)$ time, and and as such overall, the running time is $O(\mathrm{card}(T) \log u)$, as each iteration increases $\mathrm{card}(T)$ by at most two. Finally, notice that every element in T is of the form $2^i x, x \in S$ for some i, where $i \leq \log n$, and thus $\mathrm{card}(T) = O(n' \log u)$. ∎

Note that the following lemma refers to sets.

Lemma 2.3. *Given two sets* $S, T \subseteq [\![0 : u]\!]$, *one can compute* $S \oplus T$ *in* $O(u \log u)$ *time.*

Proof: Let $f_S(x) = \sum_{i \in S} x^i$ be the *characteristic polynomial* of S. Construct, in a similar fashion, the polynomial f_T and let $g = f_S * f_T$. Observe that the coefficient of x^i in g is greater than 0 if and only if $i \in S \oplus T$. As such, using FFT, one can compute the polynomial g in $O(u \log u)$ time, and extract $S \oplus T$ from it. ∎

Observation 2.4. *If* P *and* Q *form a partition of multiset* S, *then* $\sum(S) = \sum(P) \oplus \sum(Q)$.

Combining all of the above together, we can now state the following lemma which simplifies the upcoming analysis.

Lemma 2.5. *Given an algorithm that computes* $\sum_{\leq u}(S)$ *in* $\mathrm{T}(n, u) = \Omega(u \log^2 u)$ *time, for any set* $S \subseteq [\![u]\!]$ *with* n *elements, then one can compute* $\sum_{\leq u}(S')$ *for any multiset* $S' \subseteq [\![u]\!]$, *with* n' *distinct elements, in* $O\big(\mathrm{T}(n' \log u, u)\big)$ *time.*

Proof: First, from S, compute the multiset T as described in Lemma 2.2, in $O(u \log^2 u)$ time. As every element in T appears at most twice, partition it into two *sets* P and Q. Then $\sum_{\leq u}(T) = \big(\sum_{\leq u}(P) \oplus \sum_{\leq u}(Q)\big) \cap [\![0 : u]\!]$, which is computed using Lemma 2.3, in $O(u \log u)$ time. This reduces all subset sums for multisets of n' distinct elements to two instances of all subset sums for sets of size $O(n' \log u)$. ∎

2.3. The input is a set of positive integers.

In the previous section it was shown that there is little loss in generality and running time if the input is restricted to sets instead of multisets. For simplicity of exposition, we assume the input is a *set* from here on.

Here, we present the main algorithm: At a high level it uses a geometric partitioning on the input range $[\![0 : u]\!]$ to split the numbers into groups of exponentially long intervals. Each of these groups is then processed separately abusing their interval range that bounds the cardinality of the sets from that group.

Observation 2.6. *Let* g *be a positive, superadditive (i.e.* $g(x + y) \geq g(x) + g(y), \forall x, y$) *function. For a function* $f(n, m)$ *satisfying*

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

we have that $f(n, m) = O\left(g(m) \log n\right)$.

Theorem 2.7. *Given a set of positive integers* S *with total sum* σ, *one can compute the set of all subset sums* $\sum(S)$ *in* $O(\sigma \log \sigma \log n)$ *time.*

Proof: Partition S into two sets L, R of (roughly) equal cardinality, and compute recursively $L' = \sum(L)$ and $R' = \sum(R)$. Next, compute $\sum(S) = L' \oplus R'$ using Lemma 2.3. The recurrence for the running time is $f(n, \sigma) = \max_{\sigma_1 + \sigma_2 = \sigma} \{f(n/2, \sigma_1) + f(n/2, \sigma_2) + O(\sigma \log \sigma)\}$, and the solution to this recurrence, by Observation 2.6, is $O(\sigma \log \sigma \log n)$. ∎

Remark 2.8. The standard divide-and-conquer algorithm of Theorem 2.7 was already known in [38, 14], here we showed a better analysis. Note, that the basic divide-and-conquer algorithm without the FFT addition was known much earlier [23].

Lemma 2.9 ([38, 14]). *Given a set $S \subseteq [\![\Delta]\!]$ of size n, one can compute the set $\sum(S)$ in $O\big(n\Delta \log(n\Delta) \log n\big)$ time.*

Proof: Observe that $\Sigma_S \leq \Delta n$ and apply Theorem 2.7. ∎

Lemma 2.10. *Given two sets of points $S, T \subseteq [\![0:u]\!] \times [\![0:v]\!]$, one can compute $S \oplus T$ in $O\big(uv \log(uv)\big)$ time.*

Proof: Let $f_S(x, y) = \sum_{(i,j) \in S} x^i y^j$ be the characteristic polynomial of S. Construct, similarly, the polynomial f_T, and let $g = f_S * f_T$. Note that the coefficient of $x^i y^j$ is greater than 0 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 [3, Chapter 12.8]), in $O(uv \log uv)$ time, and extract $S \oplus T$ from it. ∎

Lemma 2.11. *Given two disjoint sets $B, C \subseteq [\![x : x + \ell]\!]$ and $\sum^{\leq \alpha}[B]$, $\sum^{\leq \alpha}[C]$, one can compute $\sum^{\leq \alpha}[B \cup C]$ in $O\big(\ell\alpha^2 \log(\ell\alpha)\big)$ time.*

Proof: Consider the function $f\big((i, j)\big) = (i - xj, j)$. Let $X = f\big(\sum^{\leq \alpha}[B]\big)$ and $Y = f\big(\sum^{\leq \alpha}[C]\big)$. If $(i, j) \in \sum^{\leq \alpha}[B] \cup \sum^{\leq \alpha}[C]$, then $i = jx + y$ for $y \in [\![0 : \ell j]\!]$. Hence $X, Y \subseteq [\![0 : \ell\alpha]\!] \times [\![0 : \alpha]\!]$.

Computing $X \oplus Y$ using the algorithm of Lemma 2.10 can be done in $O\big(\ell\alpha^2 \log(\ell\alpha)\big)$ time. Let $Z = (X \oplus Y) \cap \big([\![0 : \ell\alpha]\!] \times [\![0 : \alpha]\!]\big)$. The set $\sum^{\leq \alpha}[B \cup C]$ is then precisely $f^{-1}(Z)$. Projecting Z back takes an additional $O\big(\ell\alpha^2 \log(\ell\alpha)\big)$ time. ∎

Lemma 2.12. *Given a set $S \subseteq [\![x : x + \ell]\!]$ of size n, computing the set $\sum^{\leq \alpha}[S]$ takes $O\big(\ell\alpha^2 \log(\ell\alpha) \log n\big)$ time.*

Proof: Compute the median of S, denoted by δ, in linear time. Next, partition S into two sets $L = S \cap [\![\delta]\!]$ and $R = S \cap [\![\delta + 1 : x + \ell]\!]$. Compute recursively $L' = \sum^{\leq \alpha}[L]$ and $R' = \sum^{\leq \alpha}[R]$, and combine them into $\sum^{\leq \alpha}[L \cup R]$ using Lemma 2.11. The recurrence for the running time is:

$$f(n, \ell) = \max_{\ell_1 + \ell_2 = \ell} \left\{ f\Big(\frac{n}{2}, \ell_1\Big) + f\Big(\frac{n}{2}, \ell_2\Big) + O\big(\ell\alpha^2 \log(\ell\alpha)\big) \right\},$$

which takes $O\big(\ell\alpha^2 \log(\ell\alpha) \log n\big)$ time, by Observation 2.6. ∎

Lemma 2.13. *Given a set $S \subseteq [\![x : x + \ell]\!]$ of size n, computing the set $\sum_{\leq u}(S)$ takes $O\big((u/x)^2 \ell \log(\ell u/x) \log n\big)$ time.*

Proof: Apply Lemma 2.12 by setting $\alpha = \lfloor u/x \rfloor$ to get $\sum^{\leq \alpha}[S]$. Projecting down by ignoring the last coordinate and then intersecting with $[\![0 : u]\!]$ gives the set $\sum_{\leq u}(S)$. ∎

Lemma 2.14. *Given a set $S \subseteq [\![u]\!]$ of size n and a parameter $r_0 \geq 1$, partition S as follows:*
- *$S_0 = S \cap [\![r_0]\!]$, and*
- *for $i > 0$, $S_i = S \cap [\![r_{i-1} + 1 : r_i]\!]$, where $r_i = \lfloor 2^i r_0 \rfloor$.*

The resulting partition is composed of $\nu = O(\log u)$ sets $S_0, S_1, \ldots, S_\nu$ and can be computed in $O(n \log n)$ time.

Proof: Sort the numbers in S, and throw them into the sets, in the obvious fashion. As for the number of sets, observe that $2^i r_0 > u$ when $i > \log u$. As such, after $\log n$ sets, $r_\nu > u$. ∎

Lemma 2.15. *Given a set $S \subseteq \llbracket u \rrbracket$ of size n. For $i = 0, \ldots, \nu = O(\log u)$, let S_i be the ith set in the above partition and let $|S_i| = n_i$. One can compute $\sum_{\leq u}(S_i)$, for all i, in overall $O\left((u^2/r_0 + \min\{r_0, n\}r_0)\log^2 u\right)$ time.*

Proof: Because $S \subseteq \llbracket u \rrbracket$, $n = O(u)$. If $i = 0$, then $S_0 \subseteq \llbracket r_0 \rrbracket$, and one can compute $\sum_{\leq u}(S_0)$, in $O(n_0 r_0 \log(n_0 r_0) \log n_0)$ time, using Lemma 2.9. Since $n_0 \leq r_0$ and $n_0 \leq n$, this simplifies to $O\left(\min\{n, r_0\}r_0 \log^2 u\right)$.

For $i > 0$, the sets S_i contain numbers at least as large as r_{i-1}. Moreover, each set S_i is contained in an interval of length $\ell_i = r_i - r_{i-1} = r_{i-1}$. Now, using Lemma 2.13, one can compute $\sum_{\leq u}(S_i)$ in $O\left((u/r_{i-1})^2 \ell_i \log(\ell_i u/r_{i-1}) \log n_i\right) = O\left(\frac{u^2}{r_{i-1}} \log^2 u\right)$ time. Summing this bound, for $i = 1, \ldots, \nu$, results in $O\left(\frac{u^2}{r_0} \log^2 u\right)$ running time. ∎

Theorem 2.16. *Let $S \subseteq \llbracket u \rrbracket$ be a set of n elements. Computing the set of all subset sums $\sum_{\leq u}(S)$ takes $O\left(\min\{\sqrt{n}u, u^{4/3}\} \log^2 u\right)$ time.*

Proof: Assuming the partition of Lemma 2.14, compute the subset sums $T_i = \sum_{\leq u}(S_i)$, for $i = 0, \ldots, \nu$. Let $P_1 = T_1$, and let $P_i = (P_{i-1} \oplus T_i) \cap \llbracket u \rrbracket$. Each P_i can be computed using the algorithm of Lemma 2.3. Do this for $i = 1, \ldots, \nu$, and observe that the running time to compute P_ν, given all T_i, is $O(\nu(u \log u)) = O(u \log^2 u)$.

Finally, for all $i = 1, \ldots, \nu$ calculating the T_i's:
- By setting r_0 equal to $u^{2/3}$ and using Lemma 2.15 takes $O\left(u^{4/3} \log^2 u\right)$.
- By setting r_0 equal to $\frac{u}{\sqrt{n}}$ and using Lemma 2.15 takes $O\left(\sqrt{n}u \log^2 u\right)$.

Taking the minimum of these two, proves the theorem. ∎

Putting together Theorem 2.7, Theorem 2.16 and Lemma 2.5, results in the following when the input is a *multiset*.

Theorem 2.17 (Main theorem). *Let $S \subseteq \llbracket u \rrbracket$ be a multiset of n' distinct elements, with total sum σ, computing the set of all subset sums $\sum_{\leq u}(S)$ takes*

$$O\left(\min\left\{\sqrt{n'}\, u \log^{\frac{5}{2}} u, u^{\frac{4}{3}} \log^2 u, \sigma \log \sigma \log\left(n' \log u\right)\right\}\right)$$

time.

3. Subset sums for finite cyclic groups

In this section, we demonstrate the robustness of the idea underlying the algorithm of Section 2 by showing how to extend it to work for finite cyclic groups. The challenge is that the previous algorithm throws away many sums that fall outside of $\llbracket u \rrbracket$ during its execution, but this can no longer be done for finite cyclic groups, since these sums stay in the group and as such must be accounted for.

3.1. Notations.

For any positive integer m, the set of integers *modulo* m with the operation of addition forms a finite ***cyclic group***, the group $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ of *order* m. Every finite cyclic group of order m is isomorphic to the group $\mathbb{Z}_m$ (as such it is sufficient for our purposes to work with $\mathbb{Z}_m$). Let $U(\mathbb{Z}_m) = \{x \in \mathbb{Z}_m \mid \gcd(x, m) = 1\}$ be the ***set of units*** of $\mathbb{Z}_m$, and let *Euler's totient* function $\varphi(m) = |U(\mathbb{Z}_m)|$ be the ***number of units*** of $\mathbb{Z}_m$. We remind the

reader that two integers α and β such that $\gcd(\alpha, \beta) = 1$ are *coprime* (or relatively prime). The set

$$x[\![\ell]\!] = \{x, 2x, \ldots, \ell x\}$$

is a finite arithmetic progression, henceforth referred to as a **segment** of **length** $|x[\![\ell]\!]| = \ell$. Finally, let $S/x = \{s/x \mid s \in S \text{ and } x \mid s\}$ and $S \% x = \{s \in S \mid x \nmid s\}$, where $x \mid s$ and $x \nmid s$ denote that "s divides q" and "s does not divide q", respectively. For an integer x, let $\sigma_0(x)$ denote the **number of divisors** of x and $\sigma_1(x)$ the **sum of its divisors**.

3.2. Subset sums and segments.

Lemma 3.1. *For a set $S \subseteq \mathbb{Z}_m$ of size n, such that $S \subseteq x[\![\ell]\!]$, the set $\sum(S)$ can be computed in $O(n\ell \log(n\ell) \log n)$ time.*

Proof: All elements of $x[\![\ell]\!]$ are multiplicities of x, and thus $S' := S/x \subseteq [\![\ell]\!]$ is a well defined set of integers. Next, compute $\sum(S')$ in $O(n\ell \log(n\ell) \log n)$ time using the algorithm of Lemma 2.9 (over the integers). Finally, compute the set $\{\sigma x \pmod{m} \mid \sigma \in \sum(S')\} = \sum(S)$ in linear time. ∎

Lemma 3.2. *Let $S \subseteq \mathbb{Z}_m$ be a set of size n covered by segments $x_1[\![\ell]\!], \ldots, x_k[\![\ell]\!]$, formally $S \subseteq \bigcup_{i=1}^{k} x_i[\![\ell]\!]$, then the set $\sum(S)$ can be computed in $O(km \log m + n\ell \log(n\ell) \log n)$ time.*

Proof: Partition, in $O(kn)$ time, the elements of S into k sets $S_1, \ldots, S_k$, such that $S_i \subseteq x_i[\![\ell]\!]$, for $i \in [\![k]\!]$. Next, compute the subset sums $T_i = \sum(S_i)$ using the algorithm of Lemma 3.1, for $i \in [\![k]\!]$. Then, compute $T_1 \oplus T_2 \oplus \ldots \oplus T_k = \sum(S)$, by $k-1$ applications of Lemma 2.3. The resulting running time is $O\big((k-1)m \log m + \sum_i |S_i|\ell \log(|S_i|\ell) \log |S_i|\big) = O(km \log m + n\ell \log(n\ell) \log n)$. ∎

3.3. Covering a subset of $U(\mathbb{Z}_m)$ by segments.

Somewhat surprisingly, one can always find a short but "heavy" segment.

Lemma 3.3. *Let $S \subseteq U = U(\mathbb{Z}_m)$, there exists a constant c, for any ℓ such that $c2^{\frac{\ln m}{\ln \ln m}} \leq \ell \leq m$ there exists an element $x \in U$ such that $|x[\![\ell]\!] \cap S| = \Omega\left(\frac{\ell}{m}|S|\right)$.*

Proof: Fix a $\beta \in U$. For $i \in U \cap [\![\ell]\!]$ consider the modular equation $ix \equiv \beta \pmod{m}$, this equation has a unique solution $x \in U$ – here we are using the property that i and β are coprime to m. Let $\alpha = |U|/2m$. Let $\omega(m)$ be the number of distinct prime factors of m, and $\theta(m) = 2^{\omega(m)}$ be the number of distinct square-free divisors of m. Then $\theta(m) \leq c2^{\frac{\ln m}{\ln \ln m}} < \alpha\ell$ [35]. There are at least $2\alpha\ell - \theta(m) \geq \alpha\ell$ elements in $U \cap [\![\ell]\!]$ [40, Equation (1.4)].

Hence, when $\beta \in U$ is fixed, the number of values of x such that $\beta \in x[\![\ell]\!]$ is at least $\alpha\ell$. Namely, every element of $S \subseteq U$ is covered by at least $\alpha\ell$ segments $\{x[\![\ell]\!] \mid x \in U\}$. As such, for a random $x \in U$ the expected number of elements of S that are contained in $x[\![\ell]\!]$ is $(|S|\alpha\ell)/|U| = \frac{\ell}{2m}|S|$. Therefore, there must be a choice of x such that $|x[\![\ell]\!] \cap S|$ is larger than the average, implying the claim. ∎

One can always find a small number of segments of length ℓ that contain all the elements of $U(\mathbb{Z}_m)$.

Lemma 3.4. *Let $S \subseteq U(\mathbb{Z}_m)$ of size n, then for any ℓ such that $\ell \geq m^{1/2}$ there is a collection $\mathcal{L}$ of $O(\frac{m}{\ell} \ln n)$ segments, each of length ℓ, such that $S \subseteq \bigcup_{x \in \mathcal{L}} x[\![\ell]\!]$. Furthermore, such a cover can be computed in $O\big((n + \log m)\ell\big)$ time.*

8

Proof: Consider the set system defined by the ground set $\mathbb{Z}_m$ and the sets $\{x[\![\ell]\!] \mid x \in U(\mathbb{Z}_m)\}$. Next, consider the standard greedy set cover algorithm [24, 39, 30]: Pick a segment $x[\![\ell]\!]$ such that $|x[\![\ell]\!] \cap S|$ is maximized, remove all elements of S covered by $x[\![\ell]\!]$, add $x[\![\ell]\!]$ to the cover, and repeat. By Lemma 3.3, there is a choice of x such that the segment $x[\![\ell]\!]$ contains at least a $c\ell/m$ fraction of S, for some constant c. After $m/c\ell$ iterations of this process, there will be at most $(1 - c\ell/m)^{m/c\ell} n \leq n/e$ elements remaining. As such, after $O(\frac{m}{\ell} \ln n)$ iterations the original set S is covered.

To implement this efficiently, in the preprocessing stage compute the modular inverses of every element in $[\![\ell]\!]$ using the extended Euclidean algorithm, in $O(\ell \log m)$ time [9, Section 31.2]. Then, for every $b \in S$ and every $i \in [\![\ell]\!]$, find the unique x (if it exists) such that $ix \equiv b \pmod m$, using the inverse i^{-1} in $O(1)$ time. This indicates that b is in $x[\![\ell]\!] \cap S$. Now, the algorithm computes $x[\![\ell]\!] \cap S$, for all x, in time $O(n\ell + \ell \log m)$. Next, feed the sets $x[\![\ell]\!] \cap S$, for all x, to a linear time greedy set cover algorithm and return the desired segments in $O(n\ell)$ time [9, Section 35.3]. The total running time is $O\big((n + \log m)\, \ell\big)$. ∎

3.4. Subset sums when all numbers are coprime to m.

Lemma 3.5. *Let $S \subseteq U(\mathbb{Z}_m)$ be a set of size n. Computing the set of all subset sums $\sum(S)$ takes $O\big(\min\{\sqrt{n}m,\, m^{5/4}\} \log m \log n\big)$ time.*

Proof: If $|S| \geq 2\sqrt{m}$, then $\sum(S) = \mathbb{Z}_m$ [20, Theorem 1.1]. As such, the case where $n = |S| \geq 2\sqrt{m}$ is immediate.

For the case that $n < 2\sqrt{m}$ we do the following. Apply the algorithm of Lemma 3.4 for $\ell = m/\sqrt{n} \geq m^{1/2}$. This results in a cover of S by $O(\frac{m}{\ell} \log n)$ segments (each of length ℓ), which takes $O\big((n + \log m)\, \ell\big) = O(\sqrt{n}m \log m)$ time. Next, apply the algorithm of Lemma 3.2 to compute $\sum(S)$ in $O(n\ell \log(n\ell) \log n) = O(\sqrt{n}m \log m \log n)$ time. Since, $n = O(\sqrt{m})$ this running time is $O\big(\min\{\sqrt{n}m,\, m^{5/4}\} \log m \log n\big)$. ∎

3.5. The algorithm: Input is a subset of $\mathbb{Z}_m$.

In this section, we show how to tackle the general case when S is a subset of $\mathbb{Z}_m$.

3.5.1. Algorithm.

The input instance is a triple (Γ, μ, τ), where Γ is a set, μ its *modulus* and τ an auxiliary parameter. For such an instance (Γ, μ, τ) the algorithm computes the set of all subset sums of Γ modulo μ. The initial instance is (S, m, m).

Let q be the smallest prime factor of τ, referred to as *pivot*. Partition Γ into the two sets:

$$\Gamma/q = \big\{s/q \mid s \in \Gamma \text{ and } q \mid s\big\} \text{ and } \Gamma\%q = \big\{s \in \Gamma \mid q \nmid s\big\}.$$

Recursively compute the (partial) subset sums $\sum(\Gamma/q)$ and $\sum(\Gamma\%q)$, of the instances $(\Gamma/q, \mu/q, \tau/q)$ and $(\Gamma\%q, \mu, \tau/q)$, respectively. Then compute the set of all subset sums $\sum(\Gamma) = \big\{qx \mid x \in \sum(\Gamma/q)\big\} \oplus \sum(\Gamma\%q)$ by combining them together using Lemma 2.3. At the bottom of the recursion, when $\tau = 1$, for each set compute its subset sums, using the algorithm of Lemma 3.5.

3.5.2. Handling multiplicities.

During the execution of the algorithm there is a natural tree formed by the recursion. Consider an instance (Γ, μ, τ) such that the pivot q divides τ (and μ) with multiplicity r. The top level recursion would generate instances with sets Γ/q and $\Gamma\%q$. In the next level, Γ/q is partitioned into Γ/q^2 and $(\Gamma/q)\%q$. On the other side of the recursion $\Gamma\%q$ gets partitioned

9

(naively) into $(\Gamma\%q)/q$ (which is an empty set) and $(\Gamma\%q)\%q = \Gamma\%q$. As such, this is a superfluous step and can be skipped. Hence, compressing the r levels of the recursion for this instance results in $r+1$ instances:

$$\Gamma\%q, \; (\Gamma/q)\%q, \; \ldots, \; (\Gamma/q^{r-1})\%q, \; \Gamma/q^r \; .$$

The total size of these sets is equal to the size of Γ. In particular, compress this subtree into a single level of recursion with the original call having $r+1$ children. At each such level of the tree label the edges by $0, 1, 2, \ldots, r$, based on the multiplicity of the divisor of the resulting (node) instance (i.e., an edge between instance sets Γ and $(\Gamma/q^2)\%q$ would be labeled by "2").

3.5.3. Analysis.

The recursion tree formed by the execution of the algorithm has a level for each of the $k = O(\log m / \log \log m)$ distinct prime factors of m [35] – assume the root level is the 0th level.

Lemma 3.6. *Consider running the algorithm on input* (S, m, m). *Then the values of the moduli at the leaves of the recursion tree are* unique, *and are precisely the divisors of* m.

Proof: Let $m = \prod_{i=1}^{k} q_i^{r_i}$ be the prime factorization of m, where $q_i < q_{i+1}$ for all $1 \le i < k$. Then every vector $\boldsymbol{x} = (x_1, \ldots, x_k)$, with $0 \le x_i \le r_i$, defines a path from the root to a leaf of modulus $m / \prod_{i=1}^{k} q_i^{x_i}$ in the natural way: Starting at the root, at each level of the tree follow the edge labeled x_i. If for two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ there is an $i \in [\![k]\!]$ such that $x_i \ne y_i$, then the two paths they define will be different (starting at the ith level). And, by the unique factorization of integers, the values of the moduli at the two leaves will also be different. Finally, note that every divisor of m, $\prod_{i=1}^{k} q_i^{\rho_i}$ with $0 \le \rho_i \le r_i$, occurs as a modulus of a leaf, and can be reached by following the path $(r_1 - \rho_1, \ldots, r_k - \rho_k)$ down the tree. ∎

Theorem 3.7. *Let* $S \subseteq \mathbb{Z}_m$ *be a set of size* n. *Computing the set of all subset sums* $\sum(S)$ *takes* $O\left(\min\left\{\sqrt{n}m, m^{5/4}\right\} \log^2 m\right)$ *time.*

Proof: The algorithm is described in Section 3.5.1, when the input is (S, m, m). We break down the running time analysis into two parts: The running time at the leaves, and the running time at internal nodes.

Let δ be the number of leaves of the recursion tree. Arrange them so the modulus of the ith leaf, μ_i, is the ith largest divisor of m. Note that μ_i is at most m/i, for all $i \in [\![\delta]\!]$. Using Lemma 3.5, the running time is bounded by

$$O\left(\sum_{i=1}^{\delta} \min\left\{\sqrt{n_i}\,\mu_i, \mu_i^{5/4}\right\} \log n_i \log \mu_i\right) = O\left(\log m \log n \sum_{i=1}^{\delta} \min\left\{\sqrt{n_i}\,\frac{m}{i}, \left(\frac{m}{i}\right)^{5/4}\right\}\right) \; .$$

Using Cauchy-Schwartz, the first sum of the min is bounded by

$$m \sum_{i=1}^{\delta} \frac{\sqrt{n_i}}{i} \le m \sqrt{\left(\sum_{i=1}^{\delta} (\sqrt{n_i})^2\right)\left(\sum_{i=1}^{\delta} \frac{1}{i^2}\right)} = O(\sqrt{n}m) \; ,$$

and the second by $O(m^{5/4})$. Putting it all together, the total work done at the leaves is $O\left(\min\{\sqrt{n}m, m^{5/4}\} \log m \log n\right)$.

Next, consider an internal node of modulus μ, pivot q and $r+1$ children. The algorithm combines these instances, by applying r times Lemma 2.3. The total running time necessary

10

for this process is described next. As the moduli of the instances decrease geometrically, pair up the two smallest instances, combine them together, and in turn combine the result with the next (third) smallest instance, and so on. This yields a running time of

$$O\left(\sum_{i=1}^{r}\frac{\mu}{q^i}\log\frac{\mu}{q^i}\right)=O(\mu\log\mu)\,.$$

At the leaf level, by Lemma 3.6, the sum of the moduli $\sum_{i=1}^{\delta}\mu_i$ equals to $\sigma_1(m)$, and it is known that $\sigma_1(m)=O(m\log\log m)$ [21, Theorem 323]. As such, the sum of the moduli of all internal nodes is bounded by $O(km\log\log m)=O(m\log m)$, as the sum of each level is bounded by the sum at the leaf level, and there are k levels. As each internal node, with modulus μ, takes $O(\mu\log\mu)$ time and $x\log x$ is a convex function, the total running time spent on all internal nodes is $O\big(m\log m\log(m\log m)\big)=O(m\log^2 m)$.

Aggregating everything together, the complete running time of the algorithm is bounded by $O\big(\min\{\sqrt{n}m,m^{5/4}\}\log^2 m\big)$, implying the theorem. ∎

The results of this section, along with the analysis of the recursion tree above, conclude the following corollary on covering $\mathbb{Z}_m$ with a small number of segments. The result is useful for error correction codes, and improves the recent bound of Chen et al. by a factor of $\sqrt{\ell}$ [8].

Corollary 3.8. *There exist a constant c, for all ℓ such that $c2^{\frac{\ln m}{\ln\ln m}}\le\ell\le m$, one can cover $\mathbb{Z}_m$ with $O\left((\sigma_1(m)\ln m)/\ell\right)+\sigma_0(m)$ segments of length ℓ. Furthermore, such a cover can be computed in $O(m\ell)$ time.*

Proof: Let $S_{m/d}=\{x/(m/d)\mid x\in\mathbb{Z}_m$ and $\gcd(x,m)=m/d\}$, for all $d\mid m$. Note that $S_{m/d}=U(\mathbb{Z}_d)$, hence by Lemma 3.4, each $S_{m/d}$ has a cover of $O((d\ln d)/\ell)$ segments. Next, "lift" the segments of each set $S_{m/d}$ back up to $\mathbb{Z}_m$ (by multiplying by m/d) forming a cover of $\mathbb{Z}_m$. The number of segments in the final cover is bounded by

$$\sum_{\substack{d\mid m\\\ell\le d}}O\left(\frac{d}{\ell}\ln m\right)+\sum_{\substack{d\mid m\\\ell>d}}1=O\left(\frac{\sigma_1(m)\ln m}{\ell}\right)+\sigma_0(m)\,.$$

The time to cover each $S_{m/d}$, by Lemma 3.4, is $O\big((n+\log m)\,\ell\big)=O\big((\varphi(d)+\log d)\,\ell\big)$, since there are $\varphi(d)$ elements in $S_{m/d}$, and $S_{m/d}\subseteq\mathbb{Z}_d$. Also, $\varphi(d)$ dominates $\log d$, as $O\big(\varphi(d)\big)=\Omega(d/\log\log d)$ [21, Theorem 328], therefore the running time simplifies to $O\big(\varphi(d)\ell\big)$. Summing over all $S_{m/d}$ we have

$$\sum_{d\mid m}O\big(\varphi(d)\ell\big)=O\left(\ell\sum_{d\mid m}\varphi(d)\right)=O(m\ell)\,,$$

since $\sum_{d\mid m}\varphi(d)=m$ [21, Sec 16.2], implying the corollary. ∎

If $\ell<c2^{\frac{\ln m}{\ln\ln m}}$, then $\ell=m^{o(1)}$. The corollary above then shows that for all ℓ, there is a cover of $\mathbb{Z}_m$ with $m^{1+o(1)}/\ell$ segments.

4. Recovering the solution

Given sets X and Y, a number x is a **witness** for $i\in X\oplus Y$, if $x\in X$ and $i-x\in Y$. A function $w:X\oplus Y\to X$ is a **witness function**, if $w(i)$ is a witness of i.

If one can find a witness function for each $X\oplus Y$ computation of the algorithm, then we can traceback the recursion tree and reconstruct the subset that sums up to t in $O(n)$ time. The problem of finding a witness function quickly can be reduced to the *reconstruction problem* defined next.

4.1. Reduction to the reconstruction problem.

In the reconstruction problem, there are hidden sets $S_1, \ldots, S_n \subseteq [\![m]\!]$ and we have two oracles SIZE and SUM that take as input a query set Q.

- SIZE(Q) returns the size of each intersection:

$$\left(|S_1 \cap Q|, |S_2 \cap Q|, \ldots, |S_n \cap Q|\right)$$

- SUM(Q) returns the sum of elements in each intersection:

$$\left(\sum_{s \in S_1 \cap Q} s, \sum_{s \in S_2 \cap Q} s, \ldots, \sum_{s \in S_n \cap Q} s\right)$$

The reconstruction problem asks to find n values $x_1, \ldots, x_n$ such that for all i, if S_i is non-empty, $x_i \in S_i$. Let f be the running time of calling the oracles, and assume $f = \Omega(m + n)$, then is it known that one can find $x_1, \ldots, x_n$ in $O(f \log n \operatorname{polylog} m)$ time [1].

If $X, Y \subseteq [\![u]\!]$, finding the witness of $X \oplus Y$ is just a reconstruction problem. Here the hidden sets are $W_0, \ldots, W_{2u} \subseteq [\![2u]\!]$, where $W_i = \{x \mid x + y = i \text{ and } x \in X, y \in Y\}$ is the set of witnesses of i. Next, define the polynomials $\chi_Q(x) = \sum_{i \in Q} x^i$ and $I_Q(x) = \sum_{i \in Q} i x^i$. The coefficient for x^i in $\chi_Q \chi_Y$ is $|W_i \cap Q|$ and in $I_Q \chi_Y$ is $\sum_{s \in W_i \cap Q} s$, which are precisely the ith coordinate of SIZE(Q) and SUM(Q), respectively. Hence, the oracles can be implemented using polynomial multiplication, in $\widetilde{O}(u)$ time per call. This yields an $\widetilde{O}(u)$ time deterministic algorithm to compute $X \oplus Y$ *with* its witness function.

Hence, with a polylogarithmic slowdown, we can find a witness function every time we perform a $\oplus$ operation, thus, effectively, maintaining which subsets sum up to which sum.

5. Applications and extensions

Since every algorithm that uses subset sum as a subroutine can benefit from the new algorithm, we only highlight certain selected applications and some interesting extensions. Most of these applications are derived directly from the divide-and-conquer approach.

5.1. Bottleneck graph partition.

Let $G = (V, E)$ be a graph with n vertices m edges and let $w : E \to \mathbb{R}^+$ be a weight function on the edges. The *bottleneck graph partition* problem is to split the vertices into two equal-sized sets such that the value of the bottleneck (maximum-weight) edge, over all edges across the cut, is minimized. This is the simplest example of a graph partition problem with cardinality constraints. The standard divide-and-conquer algorithm reduces this problem to solving $O(\log n)$ subset sum problems: Pick a weight, delete all edges with smaller weight and decide if there exists an arrangement of components that satisfy the size requirement [22]. The integers being summed are the various sizes of the components, the target value is $n/2$, and the sum of all inputs is n. Previously, using the $O(\sigma^{3/2})$ algorithm by Klinz and Woeginger, the best known running time was $O(m + n^{3/2} \log n)$ [27]. Using Theorem 2.7, this is improved to $O(m) + \widetilde{O}(n)$ time.

5.2. All subset sums with cardinality information.

Let $S = \{s_1, s_2, \ldots, s_n\}$. Define $\sum_{\leq u}^{\leq n}(S)$ to be the set of pairs (i, j), such that $(i, j) \in \sum_{\leq u}^{\leq n}(S)$ if and only if $i \leq u, j \leq n$ and there exists a subset of size j in S that sums up to i. We are interested in computing the set $\sum_{\leq u}^{\leq n}(S)$.

We are only aware of a folklore dynamic programming algorithm for this problem that runs in $O(n^2 u)$ time. We include it here for completion. Let $D[i, j, k]$ be `true` if and only if there exists a subset of size j that sums to i using the first k elements. The recursive relation is

$$D[i, j, k] = \begin{cases} \text{true} & \text{if } i = j = k = 0 \\ \text{false} & \text{if } i > j = k = 0 \\ D[i, j, k-1] \vee D[i - s_k, j-1, k-1] & \text{otherwise} \end{cases}$$

where we want to compute $D[i, j, n]$ for all $i \leq u$ and $j \leq n$. In the following we show how to do (significantly) better.

Theorem 5.1. *Let $S \subseteq [\![u]\!]$ be a set of size n, then one can compute the set $\sum_{\leq u}^{\leq n}(S)$ in $O\big(nu \log(nu) \log n\big)$ time.*

Proof: Partition S into two (roughly) equally sized sets S_1 and S_2. Find $\sum_{\leq u}^{\leq n/2}(S_1)$ and $\sum_{\leq u}^{\leq n/2}(S_2)$ recursively, and combine them using Lemma 2.10, in $O\big(nu \log(nu)\big)$ time. The final running time is then given by Observation 2.6. ∎

5.3. Counting and power index.

Here we show that the standard divide-and-conquer algorithm can also answer the counting version of all subset sums. Namely, computing the function $N_{u,S}(x)$: the number of subsets of S that sum up to x, where $x \leq u$.

For two functions $f, g : X \to Y$, define $f \odot g : X \to Y$ to be

$$(f \odot g)(x) = \sum_{t \in X} f(x) g(x - t)$$

Corollary 5.2. *Given two functions $f, g : [\![0 : u]\!] \to \mathbb{N}$ such that $f(x), g(x) \leq b$ for all x, one can compute $f \odot g$ in $O(u \log u \log b)$ time.*

Proof: This is an immediate extension of Lemma 2.10 using the fact that multiplication of two degree u polynomials, with coefficient size at most b, takes $O(u \log u \log b)$ time [36]. ∎

Theorem 5.3. *Let S be a set of n positive integers. One can compute the function $N_{u,S}$ in $O(nu \log u \log n)$ time.*

Proof: Partition S into two (roughly) equally sized sets S_1 and S_2. Compute N_{u,S_1} and N_{u,S_2} recursively, and combine them into $N_{u,S} = N_{u,S_1} \odot N_{u,S_1}$ using Lemma 5.2, in $O(u \log u \log 2^n) = O(nu \log u)$ time. The final running time is then given by Observation 2.6. ∎

5.3.1. Power indices.

The *Banzhaf index* of a set S of n voters with cutoff u can be recovered from $N_{u,S}$ in linear time. The Theorem 5.3 yields an algorithm for computing the Banzhaf index in $\widetilde{O}(nu)$ time. Previous dynamic programming algorithms take $O(nu)$ arithmetic operations, which translates to $O(n^2 u)$ running time [42]. Similar speed-ups (of, roughly, a factor n) can be obtained for the *Shapley-Shubik index*.

Acknowledgments

References

[1] Yonatan Aumann, Moshe Lewenstein, Noa Lewenstein, and Dekel Tsur. Finding witnesses by peeling. *ACM Trans. Algorithms*, 7(2):24:1–24:15, March 2011.

[2] Richard Bellman. Notes on the theory of dynamic programming iv - maximization over discrete sets. *Naval Research Logistics Quarterly*, 3(1-2):67–70, 1956.

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

[4] Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. *To appear in SODA '17*, 2017.

[5] Leizhen Cai, Siu Man Chan, and Siu On Chan. *Random Separation: A New Method for Solving Fixed-Cardinality Optimization Problems*, pages 239–250. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[6] Yair Caro and Raphael Yuster. The characterization of zero-sum (mod 2) bipartite ramsey numbers. *Journal of Graph Theory*, 29(3):151–166, 1998.

[7] Mark Chaimovich. New algorithm for dense subset-sum problem. *Astrisque*, (258):363–373, 1999.

[8] Z. Chen, I. E. Shparlinski, and A. Winterhof. Covering sets for limited-magnitude errors. *IEEE Transactions on Information Theory*, 60(9):5315–5321, Sept 2014.

[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2014.

[10] Marek Cygan, Holger Dell, Daniel Lokshtanov, D'niel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlstrom. On problems as hard as cnf-sat. In *Proceedings of the 2012 IEEE Conference on Computational Complexity (CCC)*, CCC '12, pages 74–84, Washington, DC, USA, 2012. IEEE Computer Society.

[11] George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–277, 1957.

[12] Jonathan L. Dautrich, Jr. and Chinya V. Ravishankar. Compromising privacy in precise query protocols. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 155–166, New York, NY, USA, 2013. ACM.

[13] Josep Diaz, Fabrizio Grandoni, and AlbertoMarchetti Spaccamela. Balanced cut approximation in random geometric graphs. In Tetsuo Asano, editor, *Algorithms and Computation*, volume 4288 of *Lecture Notes in Computer Science*, pages 527–536. Springer Berlin Heidelberg, 2006.

[14] David Eppstein. Minimum range balanced cuts via dynamic subset sums. *Journal of Algorithms*, 23(2):375 – 385, 1997.

[15] Bruce Faaland. Solution of the value-independent knapsack problem by partitioning. *Operations Research*, 21(1):pp. 332–337, 1973.

[16] Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. *SIAM J. Comput.*, 20(6):1157–1189, December 1991.

[17] Celia A. Glass and Hans Kellerer. Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics (NRL)*, 54(3):250–257, 2007.

[18] Venkatesan Guruswami, Yury Makarychev, Prasad Raghavendra, David Steurer, and Yuan Zhou. Finding almost-perfect graph bisections. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 321–337, 2011.

[19] C. Guret and C. Prins. A new lower bound for the open-shop problem. *Annals of Operations Research*, 92(0):165–183, 1999.

[20] Y.O. Hamidoune, A.S. Llad, and O. Serra. On complete subsets of the cyclic group. *Journal of Combinatorial Theory, Series A*, 115(7):1279 – 1285, 2008.

[21] Godfrey Harold Hardy and Edward Maitland Wright. *An introduction to the theory of numbers*. Oxford Science Publications. Clarendon Press, Oxford, fifth edition, 1979. Autres tirages : 1983, 1985, 1988 (avec corrections), 1989, 1990, 1992, 1994, 1996, 1998.

[22] Dorit S. Hochbaum and Anu Pathria. The bottleneck graph partition problem. *Networks*, 28(4):221–225, 1996.

[23] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, April 1974.

[24] David S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, pages 38–49, New York, NY, USA, 1973. ACM.

[25] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[26] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[27] Bettina Klinz and Gerhard J. Woeginger. A note on the bottleneck graph partition problem. *Networks*, 33(3):189–191, 1999.

[28] Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

[29] Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 321–330, New York, NY, USA, 2010. ACM.

[30] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Math.*, 13(4):383–390, January 1975.

[31] U. Pferschy. Dynamic programming revisited: Improving knapsack algorithms. *Computing*, 63(4):419–430, 1999.

[32] Pisinger. Dynamic programming on the word ram. *Algorithmica*, 35(2):128–145, 2003.

[33] David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1 – 14, 1999.

[34] Xiangtong Qi. Coordinated logistics scheduling for in-house production and outsourcing. *Automation Science and Engineering, IEEE Transactions on*, 5(1):188–192, Jan 2008.

[35] Guy Robin. Estimation de la fonction de Tchebychef θ sur le k-ième nombre premier et grandes valeurs de la fonction $\omega(n)$ nombre de diviseurs premiers de n. *Acta Arith.*, 42(4):367–389, 1983.

[36] Arnold Schönhage. *Asymptotically fast algorithms for the numerical muitiplication and division of polynomials with complex coefficients*, pages 3–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 1982.

[37] Oliver Serang. The probabilistic convolution tree: Efficient exact bayesian inference for faster lc-ms/ms protein inference. *PLoS ONE*, 9(3):e91507, 03 2014.

[38] Oliver Serang. A fast numerical method for max-convolution and the application to efficient max-product inference in bayesian networks. *Journal of Computational Biology*, 22(8):770–783, 2015.

[39] S.K Stein. Two combinatorial covering theorems. *Journal of Combinatorial Theory, Series A*, 16(3):391 – 397, 1974.

[40] D. Suryanarayana. On $\Delta(x, n) = \phi(x, n) - x\phi(n)/n$. *Proceedings of the American Mathematical Society*, 44(1):17–21, jan 1974.

[41] Quoc Trung Tran, Chee-Yong Chan, and Guoping Wang. Evaluation of set-based queries with aggregation constraints. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 1495–1504, 2011.

[42] Takeaki Uno. Efficient computation of power indices for weighted majority games. In Kun-Mao Chao, Tsan-sheng Hsu, and Der-Tsai Lee, editors, *Algorithms and Computation*, volume 7676 of *Lecture Notes in Computer Science*, pages 679–689. Springer Berlin Heidelberg, 2012.