Subset Sum Made Simple

Konstantinos Koiliaris [*] Chao Xu [†]

Abstract

SUBSET SUM is a classical optimization problem in computer science, taught to undergraduates as an example of an NP-hard problem, which is amenable to dynamic programming, yielding polynomial running time if the input numbers are relatively small. Formally, given a set 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. Dynamic programming yields an algorithm with running time $O(nt)$. Recently, the authors [15] improved the running time to $\tilde{O}\left(\sqrt{n}t\right)$, and it was further improved to $\tilde{O}\left(n+t\right)$ by a somewhat involved randomized algorithm by Bringmann [4] (where $\tilde{O}$ hides polylogarithmic factors).

Here, we present a new and greatly simplified algorithm with running time $\tilde{O}\left(\sqrt{n}t\right)$. We believe the new algorithm and analysis are simple enough that they can be presented in an algorithms class as a striking example of applying FFT to a problem that seems (at first) unrelated. In particular, the algorithm and its analysis can be described in full detail in one and a half pages (see pages 2–3).

1 Introduction

Given a (multi) set S of n positive integers and an integer target value t, the SUBSET SUM problem is to decide if there is a (multi) subset of S that sums to t. The SUBSET SUM is a classical problem with long history. It is one of Karp's original NP-complete problems [12], closely related to other fundamental NP-complete problems such as KNAPSACK [6], CONSTRAINED SHORTEST PATH, and various other graph problems with cardinality constraints [8,11,14]. Furthermore, it is one of the initial *weakly* NP-complete problems; problems that admit *pseudopolynomial* time algorithms – a classification identified by Garey and Johnson in [10]. The first such algorithm was given in 1957 [1] by Bellman, who showed how to solve the problem in $O(nt)$ time using dynamic programming [2].

The importance of the SUBSET SUM problem in computer science is further highlighted by its role in teaching. Both the problem and its algorithm have been included in undergraduate algorithms courses' curriculums and textbooks for several decades ([5, Chapter 34.5.5], used as archetypal examples for introducing the notions of weak NP-completeness and pseudopolynomial time algorithms to college students [13, Chapter 8.8]. In addition, the conceptually simple problem statement makes this problem a great candidate in the study of NP-completeness [7, Chapter 8.1]), and, finally, Bellman's algorithm is also often introduced in the context of teaching dynamic programming [9, Chapter 5.6].

Extensive work has been done on finding better and faster pseudopolynomial time algorithms for the SUBSET SUM (for a collection of previous results see [15, Table 1.1]). The first improvement on the running time was a $O(nt/\log t)$ time algorithm by [16], almost two decades go. Recently, the *state-of-the-art* was improved significantly to $\tilde{O}\left(\sqrt{n}t\right)$ time by the authors [15]. Shortly after, in a follow up work, the running time was further improved to $\tilde{O}(n+t)$ time by Bringmann [4] – the algorithm is randomized and somewhat involved. It was also shown by Abboud et al. [1] that it is unlikely that any SUBSET SUM algorithm runs in time $O\left(t^{1-\varepsilon}\,2^{o(n)}\right)$, for any constant $\varepsilon > 0$ and target number t, as such an algorithm would imply the Strong Exponential Time Hypothesis (SETH) to be false.

In this paper, we present a new *simple* algorithm for the SUBSET SUM problem. The algorithm follows the divide-and-conquer paradigm and exploits fast Fourier transforms (FFT), matching the best deterministic running time $\tilde{O}\left(\sqrt{n}u\right)$ of [15] with a cleaner and more straightforward analysis. At a high level, the algorithm partitions the input set by congruence classes, computes the subset sums of each class recursively, and combines the results fast

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

[†]*Current affiliation:* Yahoo! Research, chao.xu@oath.com *Previous affiliation:* Department of Computer Science, University of Illinois Urbana-Champaign

[1] Note that Bellman wrote this paper before the definition of pseudopolynomial time algorithms was provided by Garey and Johnson in 1977.

due to their special structure. We believe this new simple algorithm, despite not improving upon the state-of-the-art, greatly reduces the conceptual complexity of the problem and increases our understanding of it. As such, we believe that it can be used in teaching as the new prime example of a pseudopolynomial time algorithm for the SUBSET SUM problem, as well as a striking example of applying FFT to a seemingly unrelated problem.

2 Notations

Let $[u] = \{0, 1, \ldots, u\}$ denote the set of integers in the interval $[0, u]$. Given a set $S \subseteq \mathbb{N}$, denote the *set of all subset sums of* S by

$$\mathcal{S}(S) = \left\{ \sum_{s \in T} s \;\middle|\; T \subseteq S \right\},$$

and by $\mathcal{S}_u(S)$ the set $\mathcal{S}(S) \cap [u]$. Let X and Y be two sets, then denote the *set of the pairwise sums of* X *and* Y by $X \oplus Y = \{x + y \mid x \in X \text{ and } y \in Y\}$. Observe that if $X = Y \cup Z$ and $Y \cap Z = \emptyset$, then $\mathcal{S}(X) = \mathcal{S}(Y) \oplus \mathcal{S}(Z)$.

Note that the case where the input is a multiset can be reduced to the case of a set with little loss in generality and running time (see [15, Section 2.2]), hence for simplicity of exposition we assume the input is a *set* throughout the paper.

Finally, given a set S of n elements, we define computing *all* the realizable subset sums up to an upper bound integer u, the ALL SUBSET SUMS problem. Clearly, computing all subset sums up to u also decides SUBSET SUM with target value $t \leq u$.

3 The algorithm

In this section, we present a new simple algorithm for the ALL SUBSET SUMS problem. It is based on divide-and-conquer and repeated uses of FFT. The new algorithm is described in Figure 3.1.

1. Lift the elements of S_i from the line to a dense area on the plane. 2. Recursively compute the subset sums, diving the area to two parts each time and combining them via FFT. 3. Project back the points to the line, getting $\mathcal{S}_u(S_i)$.

INPUT: A set S of n integers and an upper bound integer u.
OUTPUT: The set of all realizable subset sums of S up to u.

1. Partition S into $b = \lfloor \sqrt{n \log n} \rfloor$ sets $S_i = S \cap \{x \in \mathbb{N} \mid x \equiv i \pmod{b}\}$, $i \in [b-1]$.
2. For $i \in [b-1]$ do:
3. Compute the set of all subset sums $\mathcal{S}_u(S_i)$ via repeated applications of FFT.
4. Return $\left(\mathcal{S}_u(S_0) \bigoplus \mathcal{S}_u(S_1) \bigoplus \cdots \bigoplus \mathcal{S}_u(S_{b-1})\right) \cap [u]$.

Figure 3.1. The algorithm.

3.1 Analysis

The first lemma describes how to compute pairwise sums between sets in almost linear time in their ranges using FFT.

Lemma 3.1 (Pairwise Sums ⊕) *The following are true:*

1. *Given two sets* S, $T \subseteq [u]$, *one can compute* $S \oplus T$ *in* $O(u \log u)$ *time.*
2. *Given two sets of points* S, $T \subseteq [u] \times [v]$, *one can compute* $S \oplus T$ *in* $O(uv \log(uv))$ *time.*

Proof: Let $f_S = f_S(x) = \sum_{i \in S} x^i$ (equivalently $f_S = f_S(x, y) = \sum_{(i,j) \in S} x^i y^j$) 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 (equivalently $x^i y^j$) in g is greater than 0 if and only if $i \in S \oplus T$ (equivalently $(i, j) \in S \oplus T$ [2]). Using FFT, one can

[2] If $X, Y \subseteq \mathbb{N} \times \mathbb{N}$ are sets of points in the plane $X \oplus Y = \{(x_1 + y_1, x_2 + y_2) \mid x_1, x_2 \in X, y_1, y_2 \in Y\}$.

compute the polynomial g in $O(u \log u)$ time (equivalently compute g via multidimensional FFT [3, Chapter 12.8], in $O(u\,v \log u\,v)$ time), and extract $S \oplus T$ from it. □

The next lemma shows that the set of all subset sums of each S_i can be computed fast due to their special structure.

Lemma 3.2 *Let i and b be integers with $i < b$. Given a set $R \subseteq \{x \in \mathbb{N} \mid x \equiv i \pmod b\}$ of size n, one can compute $\mathcal{S}_u(R)$ in $O\big((u/b)\, n \log n \log u\big)$ time.*

Proof: Look at an element $x \in R$, it is of the form $x = y\,b + i$, with $y \le u/b$. Let $Q = \{(y, 1) \mid x = y\,b + i, x \in R\}$. One can see that $\mathcal{S}(R) = \big\{y\,b + i\,j \mid (y, j) \in \mathcal{S}(Q)\big\}$ [3]. As such, it suffices to find $\mathcal{S}(Q) \cap \big([\frac{u}{b} + 1] \times [n]\big)$. Partition Q into two sets Q_1 and Q_2 of roughly the same size. Then, $\mathcal{S}(Q) \cap \big([\frac{u}{b} + 1] \times [n]\big) = \big(\mathcal{S}(Q_1) \oplus \mathcal{S}(Q_2)\big) \cap \big([\frac{u}{b} + 1] \times [n]\big) = \big(\mathcal{S}(Q_1) \cap \big([\frac{u}{b} + 1] \times [\frac{n}{2}]\big)\big) \oplus \big(\mathcal{S}(Q_2) \cap \big([\frac{u}{b} + 1] \times [\frac{n}{2}]\big)\big)$. Compute the last summands via two applications of Lemma 3.1.

The running time follows the recursive formula $T(n) = 2 \cdot T(n/2) + O\big((u/b)\, n \log u\big)$, which solves to $O\big((u/b)\, n \log u \log n\big)$ proving the claim. □

Combining sets of sums can be done quickly as shown by the following lemma.

Lemma 3.3 *Let $S \subseteq [u]$ be a set and $\{S_i\}_{i=1}^{k}$ be a partition of S into k subsets. Given $\mathcal{S}_u(S_1), \ldots, \mathcal{S}_u(S_k)$, one can compute $\mathcal{S}_u(S) = \big(\bigoplus_{i=1}^{k} \mathcal{S}_u(S_i)\big) \cap [u]$ in $O(k\,u \log u)$ time.*

Proof: Let $P_1 = \mathcal{S}_u(S_1)$, and let $P_i = \big(P_{i-1} \oplus \mathcal{S}_u(S_i)\big) \cap [u]$, for $i \in [2, k]$. Then $\mathcal{S}_u(S) = P_k$. Compute each P_i, from P_{i-1} and $\mathcal{S}_u(S_i)$, in $O\big(u \log u\big)$ time each using Lemma 3.1. The total running time is $O\big(k\,u \log u\big)$. □

Putting everything together, the next theorem completes the analysis.

Theorem 3.4 *Let $S \subseteq [u]$ be a set of n elements. Computing the set of all subset sums, $\mathcal{S}_u(S)$, takes $O\big(\sqrt{n \log n}\, u \log u\big)$ time.*

Proof: Partition S into b sets $S_i = S \cap \{x \in \mathbb{N} \mid x \equiv i \pmod b\}$, each of n_i elements. For each S_i, compute the set of all subset sums $\mathcal{S}_u(S_i)$ in $O\big((u/b)\, n_i \log n_i \log u\big)$ time by applying Lemma 3.2. The time spent to compute all $\mathcal{S}_u(S_i)$ is $\sum_{i \in [b-1]} O\big((u/b)\, n_i \log n_i \log u\big) = O\big((u/b)\, n \log n \log u\big)$. Combine all $\mathcal{S}_u(S_i)$ via Lemma 3.3 in $O(b\,u \log u)$ time. The total running time is then $O\big((u/b)\, n \log n \log u + b\,u \log u\big)$. Choosing $b = \lfloor \sqrt{n \log n} \rfloor$ proves the theorem. □

Acknowledgements We would like to thank Sariel Har-Peled for his help in the editing of this paper.

References

[1] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for subset sum and bicriteria path. *CoRR*, abs/1704.04546, 2017.

[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. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1073–1084. Society for Industrial and Applied Mathematics, 2017.

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

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

[3] We abuse notation slightly by writing $\mathcal{S}(Q)$ for a set $Q \subseteq \mathbb{N} \times \mathbb{N}$, but the definition extends naturally.

[7] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 2008.

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

[9] Jeff Erickson. Algorithms, etc., January 2015. Course materials, 1250 pages.

[10] Michael R Garey and David S Johnson. "strong" np-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.

[11] 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.

[12] 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.

[13] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

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

[15] Konstantinos Koiliaris and Chao Xu. A faster pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1062–1072. SIAM, 2017.

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