Solving Medium-Density Subset Sum Problems in Expected Polynomial Time

Abraham D. Flaxman
Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh, PA 15213, USA
e-mail: abie@cmu.edu

Bartosz Przydatek
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
e-mail: bartosz@cmu.edu

April 14, 2004

Abstract

The subset sum problem (SSP) (given n numbers and a target bound B, find a subset of the numbers summing to B), is one of the classical NP-hard problems. The hardness of SSP varies greatly with the density of the problem. In particular, if m, the logarithm of the largest input number, is at least $c \cdot n$ for some constant c, the problem can be solved by a reduction to finding a short vector in a lattice. On the other hand, when $m = \mathcal{O}(\log n)$ the problem can be solved in polynomial time using dynamic programming or some other algorithms especially designed for such dense instances. However, as far as we are aware all known algorithms for dense SSP take at least $\Omega(2^m)$ time, and no polynomial time algorithm is known which solves SSP with significantly lower density, i.e., when $m = \omega(\log n)$ (and $m = o(n)$).

We present an expected polynomial time algorithm for solving uniformly random instances of subset sum problem over the domain $\mathbb{Z}_{2^m}$, with $m = \mathcal{O}((\log n)^2 / \log\log n)$. To the best of our knowledge, this is the first algorithm working efficiently beyond the magnitude bound of $\mathcal{O}(\log n)$, thus narrowing the interval with hard-to-solve SSP instances.

1 Introduction

The subset sum problem (SSP), one of the classical NP-hard problems, is defined as follows: given n numbers and a target bound B, find a subset of the numbers whose sum is equal B. There are various flavors of the problem, depending on the application or the domain of the problem.

In this work we consider a common case when the numbers are represented by m-bit strings, and the sums are computed modulo 2^m, i.e. the computation is done in $\mathbb{Z}_{2^m}$. More precisely, the subset sum problem of dimensions n and m is:

GIVEN: n numbers $a_1, \ldots, a_n$, where $a_i \in \mathbb{Z}_{2^m}$, and a target $B \in \mathbb{Z}_{2^m}$

FIND: a subset $S \subset \{1, \ldots, n\}$, such that

$$\sum_{i \in S} a_i \equiv B \mod 2^m .$$

We focus on *random* instances of the problem, where both the input numbers and the bound are picked uniformly at random. Similar random instances (with different parameters than we will eventually select) were shown by Chvatal [Chv80] to be hard instances for a class of knapsack algorithms.

The hardness of random SSP instances varies significantly with the choice of parameters, in particular the magnitude of m as a function of n (cf. [IN96]):

$m > n$**:** such instances are "almost 1-1" (each subset has a different sum), and are efficiently solvable by a reduction to a short vector in a lattice when $m \geq c \cdot n$, for some constant c [LO85, Fri86, CJL$^+$92].

$m < n$**:** such instances are "almost onto" (with multiple solutions for most targets), and are efficiently solvable by various techniques in high-density case, i.e., for $m = \mathcal{O}(\log n)$ (e.g., by dynamic programming, or using methods of analytical number theory [CFG89, GM91]).

Despite various efficient approaches to dense instances, as far as we are aware, all these algorithms take at least $\Omega(2^m)$ time, and so none of them works in polynomial time when $m = \omega(\log n)$.

1.1 Contributions

In this work we propose an expected polynomial time algorithm which solves uniformly random SSP instances with m up to $\mathcal{O}((\log n)^2 / \log \log n)$. Our algorithm starts by dividing the input instance into small, efficiently solvable subinstances. The solutions of subinstances lead to a reduced instance (simpler than the original input), which can be solved recursively. Finally, the solution of the reduced instance is combined with the solutions of subinstances yield a solution of the original instance.

To the best of our knowledge, this is the first algorithm working efficiently beyond the magnitude bound of $\mathcal{O}(\log n)$, thus narrowing the interval with hard-to-solve SSP instances.

1.2 Notation and Conventions

For the clarity of presentation we occasionally neglect the discrete nature of the problem and avoid the use of rounding operations (floors and ceilings). However, this simplification does not compromise the validity of our results. All asymptotic notation is with respect to n, we write $f(n) \sim g(n)$ to mean $f(n)/g(n) \to 1$ as $n \to \infty$.

2 The New Algorithm

In the algorithm ℓ and ε are parameters whose values are to be determined later. For simplicity, the description presented below focuses on the core part of the algorithm, which can fail on some inputs. Later we show that the failures have sufficiently low probability so that upon a failure we can run a dynamic programming algorithm and obtain an *expected* polynomial time algorithm.

To find a solution for $(a_1, \ldots, a_n; B)$, we partition the set of input numbers into n/ℓ subsets of ℓ numbers, which yields n/ℓ subinstances of SSP. We solve the subinstances computing modulo $2^{(1-\varepsilon)\ell}$. Specifically, first we successively look for a subinstance which has a solution for the target B (modulo $2^{(1-\varepsilon)\ell}$), i.e. we want the subinstance to match the $(1-\varepsilon)\ell$ least significant bits of B. When we succeed, we try to solve all the remaining subinstances for the target 0, i.e., we aim to zero the $(1-\varepsilon)\ell$ least significant bits in a non-trivial way. We denote the subinstance solved for the target B as a "B-type" subinstance, and ones solved for target 0 as "0-type".

Subsequently, we take the solutions of the 0-type subinstances, and treat the sum achieved by each solution (now modulo 2^m) as an input number to a reduced instance of SSP, and proceed recursively. Since the sums achieved have their least significant bits equal to zero, in recursive calls we discard the zeroed bits, i.e., the numbers of the reduced instance are smaller than the original numbers. The target of the reduced instance is determined by the original target and the solution of the B-type subinstance. Figure 1 presents the entire algorithm in pseudocode. A few explanations follow: The variable j iterates through all n/ℓ subinstances, while the variable k counts the total number of subinstances that succeeded. For the i-th successful subinstance, $\mathrm{idx}[i]$ represents the actual position[1] of the subinstance among all n/ℓ subinstances, to . For a set S and an integer i, the expression $S \oplus i$ denotes the set $\{a + i : a \in S\}$.

[1] Actually, to simplify the arithmetic expressions, $\mathrm{idx}[i]$ is equal to the position of the subinstance minus 1.

procedure DenseSSP$(a_1, \ldots, a_n; B, m)$
  $S := \emptyset$
  if $B \neq 0$ **then**
    $k := 0$
    $\beta := B$
    $L := 2^{(1-\varepsilon)\ell}$
    for $j := 1 \ldots n/\ell$ **do**
      /** *solve j-th subinstance of ℓ numbers with target* $\beta \mod L$ **/
      using exhaustive search find a non-empty solution $S_j \subset \{1, \ldots, \ell\}$ of
        $(a_{1+\ell(j-1)}, a_{2+\ell(j-1)}, \ldots, a_{\ell+\ell(j-1)}; \beta) \mod L$,
      if a solution $S_j \neq \emptyset$ exists **then**
        $k := k + 1$
        $\mathrm{idx}[k] := j - 1$
        $a'_k := \sum_{i \in S_j} a_{i+\ell(j-1)} \mod 2^m$
        $\beta := 0$
      end if
    end for
    if $k > 1$ **then**
      $B' := ((B - a'_1) \mod m)/L$
      $S' := \mathrm{DenseSSP}(a'_2/L, \ldots, a'_k/L; B', m - (1-\varepsilon)\ell)$
      if S' is a "success" **then**
        $S := (S_1 \oplus (\ell \cdot \mathrm{idx}[1])) \cup \bigcup_{i \in S'} (S_{i+1} \oplus (\ell \cdot \mathrm{idx}[i+1]))$
      end if
    end if
  end if
  if S is a solution **then**
    return S, "success"
  else
    return "failed"
  end if

Figure 1: The new algorithm for solving dense SSP instances

3 Analysis

In the algorithm the input of the recursive call, i.e., the reduced instance, is determined by the solutions of subinstances. However, some subinstances may not have solutions and then the size of the reduced instance will be smaller than n/ℓ. When too many subinstances fail, the density of the reduced SSP becomes too low to hope to find solution. Hence, in the analysis we introduce an additional parameter α, $0 \leq \alpha \leq 1$, and argue that at least an α fraction of subinstances succeed in each stage of the recursion.

3.1 Success Probability

Let $L = 2^{(1-\varepsilon)\ell}$. First note that all subinstances solved in our algorithm by exhaustive search consist of random ℓ-tuples of integers which are independent and uniformly distributed in $\mathbb{Z}_L$. This follows from the following claim and from the fact that every input number of each reduced instance is a sum of values uniformly distributed in its corresponding range.

Claim 1 *If X is uniformly distributed in $\mathbb{Z}_M$, then so is $(X + c) \mod M$, for each $c \in \mathbb{Z}$.*

To estimate the total number of the subinstances, observe that we solve n/ℓ subinstances in first stage, and then recurse and solve $(n/\ell - 1)/\ell$ subinstances, etc. Hence the total number of subinstances to be solved using dynamic programming is at most

$$\frac{n}{\ell} + \frac{n}{\ell^2} + \frac{n}{\ell^3} + \cdots + 1 \approx \frac{n-1}{\ell-1} .$$

We now bound the probability that less than an α fraction of subinstances succeed at a given stage (i.e., recursion level). Let $Y_1, \ldots, Y_\ell$ be uniform i.i.d. random variables from domain $\mathbb{Z}_L$. For each subset $S \subset \{1, \ldots, \ell\}$ and for any constant target $b \in \mathbb{Z}_L$, let

$$Z_S = \begin{cases} 1 & \text{if } \sum_{i \in S} Y_i = b \mod L \\ 0 & \text{otherwise} \end{cases}$$

Finally, let $Z = \sum_{S \neq \emptyset} Z_S$. We have

$$\begin{aligned}
\mathrm{E}[Z] &= (2^\ell - 1)2^{-(1-\varepsilon)\ell} = 2^{\varepsilon\ell} - 2^{-\ell+\varepsilon\ell} \sim 2^{\varepsilon\ell} \\
\mathrm{E}\left[Z^2\right] &= \mathrm{E}\left[\left(\sum_{S\neq\emptyset} Z_S\right)^2\right] = \sum_{S\neq\emptyset} \mathrm{E}\left[Z_S^2\right] + \sum_{S\neq T;\, S,T\neq\emptyset} \mathrm{E}\left[Z_S Z_T\right] \\
&= \left(2^\ell - 1\right) 2^{-(1-\varepsilon)\ell} + \left((2^\ell-1)^2 - 2^\ell + 1\right) 2^{-2(1-\varepsilon)\ell} \\
&= 2^{\varepsilon\ell} - 2^{-\ell+\varepsilon\ell} + 2^{2\varepsilon\ell} - 3\cdot 2^{-\ell+2\varepsilon\ell} + 2\cdot 2^{-2\ell+2\varepsilon\ell} \\
&\sim 2^{2\varepsilon\ell} + 2^{\varepsilon\ell} \\
\mathrm{Var}[Z] &= \mathrm{E}\left[Z^2\right] - \mathrm{E}[Z]^2 \\
&\sim 2^{2\varepsilon\ell} + 2^{\varepsilon\ell} - 2^{2\varepsilon\ell} \;=\; 2^{\varepsilon\ell}
\end{aligned}$$

The random variable Z counts the number of non-empty solutions of a SSP instance determined by $Y_1, \ldots, Y_\ell$ and b: $Z = 0$ means that none of the subsets sums up to b, while $Z > 0$ denotes that there is at least one solution. Then Chebyshev's inequality gives

$$\begin{aligned}
\mathsf{Pr}[Z = 0] &\leq \mathsf{Pr}[|Z - \mathrm{E}[Z]| \geq \mathrm{E}[Z]] \\
&\leq \frac{\mathrm{Var}[Z]}{\mathrm{E}[Z]^2} \\
&\leq \frac{2^{\varepsilon\ell}}{2^{2\varepsilon\ell}} \\
&\leq \frac{1}{2^{\varepsilon\ell}}\,. \qquad (1)
\end{aligned}$$

So

$$\mathsf{Pr}[Z > 0] \;\geq\; 1 - \frac{1}{2^{\varepsilon\ell}}\,.$$

Now we can bound the probability that at stage k in the recursion there is less than an α fraction of the subinstances succeed. Suppose that in every stage before stage k we have had at least an α fraction of successes. Then there are at least $n\,\alpha^{k-1}/\ell^k$ independent subinstances during stage k, and by (1) each of them fails with probability at most $2^{-\varepsilon\ell}$. Formally, let $\mathcal{B}_k$ denote the event that less than an α

fraction of the subinstances in stage k succeed. Then

$$\begin{aligned}\Pr\left[\mathcal{B}_k \mid \overline{\mathcal{B}_1}, \ldots, \overline{\mathcal{B}_{k-1}}\right] &\leq \Pr\left[\mathrm{Bi}(n\,\alpha^{k-1}/\ell^k, 2^{-\varepsilon\ell}) \geq (1-\alpha)n\,\alpha^{k-1}/\ell^k\right] \\ &\leq \exp\left(-\frac{(1-\alpha-2^{-\varepsilon\ell})^2(n\,\alpha^{k-1}/\ell^k)}{2^{-\varepsilon\ell}+\frac{2}{3}(1-\alpha-2^{-\varepsilon\ell})}\right),\end{aligned}$$

using a standard version of Chernoff's inequality [JŁR00, Theorem 2.1].

We use this to bound the probability DenseSSP fails. If there are d stages total then

$$\begin{aligned}\Pr[\text{failure}] &\leq \Pr[\mathcal{B}_1] + \Pr[\mathcal{B}_2 \mid \overline{\mathcal{B}_1}] + \cdots + \Pr[\mathcal{B}_d \mid \overline{\mathcal{B}_1}, \ldots, \overline{\mathcal{B}_{d-1}}] \\ &\leq d\exp\left(-\frac{3\left(1-\alpha-2^{-\varepsilon\ell}\right)^2(n\alpha^{d-1}/\ell^d)}{2-2\alpha+2^{-\varepsilon\ell}}\right).\end{aligned}$$

3.2 Running Time

The running time of the algorithm is dominated by the time required to solve all the subinstances, which is bounded by $(n-1)/(\ell-1)\cdot\mathcal{O}(2^\ell)$.

3.3 Choice of Parameters

So far we used ℓ, ε, and α as parameters to be determined later. The choice of these parameters influences the running time of the algorithm and the probability of its success, and also implies the maximum magnitude m of instances likely to be solved by DenseSSP.

Since DenseSSP uses exhaustive search to solve SSP instances with up to ℓ numbers, we must take $\ell = \mathcal{O}(\log n)$ to keep the total running time polynomial. For a particular choice of ℓ, to conclude that the bad events $\mathcal{B}_k$ are unlikely to occur, we take $n\alpha^{d-1}/\ell^d \geq (\log n)^2$, or equivalently,

$$d \leq \frac{\log n - 2\log\log n - \log\alpha}{\log\ell - \log\alpha}.$$

When all subinstances at recursive call of depth k have been solved, then $k\ell(1-\varepsilon)$ least significant bits of the target have been matched. Therefore, to match the full target when the recursion stops, it must hold that

$$m \leq d(1-\varepsilon)\ell\ .$$

For $\ell = c\log n$, where c is a constant, this yields an upper bound on m:

$$m \leq \frac{\log n - 2\log\log n - \log\alpha}{\log\ell - \log\alpha}(1-\varepsilon)(c\log n) = (1-\varepsilon-o(1))\frac{(\log n)^2}{\log\log n}\ .$$

If we take $\alpha = 1/2$, then the probability of success of DenseSSP is bounded by

$$\mathsf{Pr}\left[\text{success}\right] \geq 1 - de^{-\frac{3}{2}(\log n)^2(1-o(1))} \geq 1 - e^{(\log n)^2} \ .$$

Hence the probability of success converges to 1 with the growing size of the input, and the failure probability is so small that in the rare instances where the algorithm does fail, we can solve the instance by dynamic programming in time $O(2^{(\log n)^2})$ and obtain an complete algorithm that runs in expected polynomial time.

4 Conclusions and Open Problems

We have presented an expected polynomial time algorithm for solving uniformly random subset sum problems of medium density over $\mathbb{Z}_{2^m}$, with m bounded by $\mathcal{O}((\log n)^2/\log\log n)$, where n is the number of the input numbers. As far as we are aware, this is the first algorithm for dense instances that works efficiently beyond the magnitude bound of $\mathcal{O}(\log n)$, thus narrowing the interval with hard-to-solve SSP instances. A natural open question is whether the bound on the magnitude can be further extended, e.g. up to $(\log n)^z$ for some $z \geq 2$.

Arguably, the proposed algorithm exploits in an essential way the special structure of $\mathbb{Z}_{2^m}$, and although it can be easily adapted to work over domain $\mathbb{Z}_{c^m}$ for any constant $c \in \mathbb{N}$, we see no straightforward way of modifying it to work over $\mathbb{Z}_M$ for arbitrary $M \in \mathbb{N}$.

Finally, recall that DenseSSP is a deterministic algorithm which can fail with non-zero probability. Since this probability is very low, upon failure we can run a dynamic programming algorithm and still obtain expected polynomial time in total. A different way of handling failures might be to run DenseSSP again on randomly permuted input. Note however that such multiple trials are not fully independent, thus complicating the analysis. It is an interesting problem to compare this alternative approach with the one we have analyzed.

References

[CFG89] M. Chaimovich, G. Freiman, and Z. Galil. Solving dense subset-sum problems by using analytical number theory. *J. Complex.*, 5(3):271–282, 1989.

[Chv80] V. Chvatal. Hard knapsack problems. *Operations Research*, 28:1402–1411, 1980.

[CJL+92] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Comput. Complex.*, 2(2):111–128, 1992.

[Fri86] Alan Frieze. On the Lagarias-Odlyzko algorithm for the subset sum problem. *SIAM J. Comput.*, 15(2):536–539, 1986.

[GM91] Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. *SIAM J. Comput.*, 20(6):1157–1189, 1991.

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, Fall 1996.

[JŁR00] Svante Janson, Tomasz Łuczak, and Andrzej Rucinski. *Random graphs*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience, New York, 2000.

[LO85] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.