Smoothed analysis for the subset sum and knapsack problems

Uriel Feige

May 18, 2022

1 Introduction

We shall consider two problems that involve sums of integers. (More generally, they may involve non-integer entries. The integrality assumption is made here only so as to simplify the presentation.)

In the **subset sum** problem, there is a set U of n items, where every item i has positive integer weight w_i. In addition there is a target integer weight W. The goal is to select a set S of items satisfying $\sum_{i \in S} w_i = W$ (or to determine that no such set S exists).

In the **knapsack** problem, there is a set U of n items, where every item i has positive integer weight w_i and positive integer profit p_i. In addition there is a knapsack of integer capacity W. The goal is to select a set S of items maximizing the profit $\sum_{i \in S} p_i$, subject to not violating the capacity constraint $\sum_{i \in S} w_i \leq W$.

Both problems are NP-hard, but have pseudo-polynomial time algorithms. Namely, they can be solved (using dynamic programming) in time polynomial in (n, W) (but NP-hard for $(n, \log W)$). For example, for the knapsack problem, one can inductively fill up an n by W table T, where entry $T_{i,w}$ specifies the most profitable solution of weight w, among those that include item i but no items from $[i+1, \ldots, n]$. Initialize $T_{0,0} = 0$ and $T_{0,w} = -\infty$ for all $w \neq 0$. To fill up the table, use the relation $T_{i,w} = \max_{j<i}[T_{j,\, w-w_i} + p_i]$.

We shall present algorithms that solve (with high probability) random and semi-random instances of these problems.

2 Average case analysis and smoothed analysis

NP-hard problems are difficult to solve on worst case instances, but may be easy on typical or average case instances. In average case analysis one considers some natural distribution over input instances, and tries to determine whether there are polynomial time algorithms (and if so, which ones) that with high probability (over choice of input instance) manage to solve random instances from this distribution. For the subset sum problem, we shall consider the following distribution, parametrized by integers n and N.

Random subset sum. Each weight w_i is selected independently uniformly at random in the range $[N+1, 2N]$. A set R of agents is chosen uniformly at random (R is referred to as the *planted solution*), where each agent is in R independently with probability $\frac{1}{2}$. Then W is set to be equal to $\sum_{i \in R} w_i$. The input to the problem is $(w_1, \ldots, w_n, W)$. The desired solution is any set S (not necessarily R) for which $\sum_{i \in S} w_i = W$.

As noted earlier, subset sum can be solved in time polynomial in n and N. Hence the above distribution is interesting only when N is super-polynomial in n. We shall show that when N is sufficiently large compared to n (exponential in n^2), most instances from the above distribution can be solved in polynomial time.

For some wide range of intermediate values of N, no polynomial time algorithm is known, and also, there is no strong evidence (such as NP-hardness) that no such algorithm exists. It is conjectured that there is a range of values of N (e.g., $N = 2^n$) for which the problem is indeed hard.

Smoothed analysis is a framework for illustrating that certain classes of NP-hard problems with numerical data parameters are likely not to be difficult in practice. The framework applies to situations in which there is some randomness in the process of generating the input data. Input instances are modelled as generated in two phases. In the first phase, an adversary generates an arbitrary input instance. In the second stage, the numerical parameters are perturbed at random. The goal is to design algorithms that with high probability solve the perturbed instances optimally, where the probability is taken only over the perturbation.

When the perturbations are large compared to the original values in the adversarial input instance, the smoothed instance resembles an average case instance. When the perturbations are very small (with no perturbation at all being the extreme case), the smoothed instance resembles a worst case instance. Hence the interesting range of parameters for smoothed analysis is when the perturbations are relatively small, but not too small.

Another possible application for the smoothed analysis framework is as follows. Given an arbitrary instance, first smooth it (here it is assumed that the smoothing operation can be done in random polynomial time), and then solve the smoothed instance. If the optimal solution (or its value) enjoys some Lipschitz continuity with respect to the smoothing operation, and the smoothing operation is "small", then the solution to the smoothed instance may provide a good approximation to the solution for the original instance.

We shall consider the following smoothed version of the knapsack problem.

Smoothed knapsack. The adversary first selects an arbitrary instance for the knapsack problem. Thereafter, weights of items are perturbed at random. For simplicity and concreteness, we consider the following perturbation, parametrized by $\delta > 0$. For each weight w_i, we select uniformly at random r_i in the range $[0, \delta W]$, and the perturbed weight becomes $\hat{w}_i = w_i + r_i$. (The perturbation is not symmetric around 0, so that we never end up with an item of negative weight.) The profits of items are not perturbed.

The smaller the parameter δ, the more the smoothed input instance resembles the adversarial input instance. The running time of the algorithm that we shall present for smoothed knapsack runs in time polynomial in n and $\frac{1}{\delta}$, and hence remains polynomial as long at $\delta \geq \Omega\left(\frac{1}{n^{O(1)}}\right)$.

As subset sum seems like a simpler problem than knapsack (it has fewer parameters), and average case analysis seems simpler than smoothed analysis (there is no adversary), it is natural to first present an algorithm for random subset sum, and only afterwards present an algorithm for smoothed knapsack. Nevertheless, we shall consider smoothed knapsack before we consider random subset sum, as the algorithm for smoothed knapsack is in a sense more elementary.

3 The smoothed knapsack problem and the isolating lemma

The book [8] contains three chapters on smoothed analysis. This section is based on Chapter 15 (a part in that chapter that is based on [2]), but is simplified so as to mostly convey the main ideas, at the cost of presenting a less general result.

Let us first recall that there are fully polynomial time approximation schemes (FPTAS) for knapsack. The following lemma is based on one of them.

Lemma 1 *Let $W^* \leq W$ denote the weight of an optimal solution to the knapsack problem and let P^* denote its profit. Suppose that for a given input instance it is known that for some explicit $\epsilon > 0$, no solution of weight in the range $(W^*, W^* + \epsilon W]$ has profit at least P^*. Then the input instance can be solved in time polynomial in $(n, \frac{1}{\epsilon})$.*

Proof: We shall run two algorithms, at least one of which will produce the optimal solution.

The first algorithm assumes that $W^* > (1 - \frac{\epsilon}{2})W$. Modify the weights as follows. Let $k = \frac{\epsilon W}{2n}$. Round down each weight w_i to the nearest multiple of k (consequently, also W can be rounded down to the nearest multiple of k), giving weight w'_i (and W'). Observe that with the new weights, the original optimal solution remains both feasible and optimal. Now the dynamic programming table has size $n \cdot \lfloor \frac{W}{k} \rfloor = O(\frac{n^2}{\epsilon})$, and the problem instance can be solved in time polynomial in $(n, \frac{1}{\epsilon})$.

The second algorithm assumes that $W^* \leq (1 - \frac{\epsilon}{2})W$. In this case each weight w_i is rounded up (rather than down) to the nearest multiple of k (again, W can be rounded down to the nearest multiple of k), and the analysis proceeds as for the first algorithm. □

We now consider smoothed instances of the knapsack problem.

Assume some consistent tie breaking rule among solutions of equal profit. We assume that this rule breaks ties in favour of lower weight solutions. Let G denote the optimal solution for the smoothed instance, let P^* denote its profit, and let W^* denote its weight. Note that $W^* \leq W$. The following lemma is a variation on the *isolating lemma* of [7].

Lemma 2 *With probability at least $1 - \frac{\epsilon n}{\delta}$, there is no solution of weight in the range $(W^*, W^* + \epsilon W]$ that has profit at least P^*.*

The lemma may sound surprising, as the number of solutions of weight in a range of width ϵW may well be exponential in n.

Proof: We denote the perturbed weights by $\hat{w}_i$. If $\sum_i \hat{w}_i \leq W^*$, then there is no set of weight above W^*, and there is nothing to prove. Hence we assume that $\sum_i \hat{w}_i > W^*$. Consequently, G misses at least one item.

For $1 \leq i \leq n$, let U_i denote the collection of all sets of items that do not include item i. Let P_i denote the maximum profit of a feasible solution (of weight at most W) within U_i, and let G_i denote the corresponding solution. Observe that for every $i \notin G$ it holds that $G = G_i$. Let W_i denote the minimum weight of a feasible solution within U_i among those that have profit at least $P_i - p_i$, and let B_i denote the corresponding solution. (It might happen that $B_i = G_i$.) Note that $W_i + \hat{w}_i \geq W^*$, as otherwise G would not be the optimal solution (recall the tie breaking rule favouring solutions of lower weight).

Let $\mathcal{B}$ denote the collection of sets of weight in the range $(W^*, W^* + \epsilon W]$ that have profit at least P^*. We need to show that $\mathcal{B}$ is empty. Suppose otherwise. Observe that for every

set $B \in \mathcal{B}$ there must be an item i in B but not in G. For every $i \notin G$, let $\mathcal{B}_i \subset \mathcal{B}$ denote the collection of sets in $\mathcal{B}$ that contain i. For $\mathcal{B}$ to be non-empty, it must be that at least one of the $\mathcal{B}_i$ is non-empty. For $\mathcal{B}_i$ to be non-empty, it must be that $W^* < W_i + \hat{w}_i \leq W^* + \epsilon W$. We refer to this event as E_i. Fixing all weights and perturbations except for r_i, we see that $Pr[E_i] \leq \frac{\epsilon W}{\delta W}$. The lemma follows from a union bound over all $i \notin G$ (and noting that there are at most n such values of i). □

Corollary 3 *In the smoothed model for knapsack described above, the optimal solution can be found with high probability in time polynomial in n and in $\frac{1}{\delta}$.*

Proof: Choose ϵ to be much smaller than $\frac{\delta}{n}$. By Lemma 2, there is high probability that there is no solution of weight in the range $(W^*, W^* + \epsilon W]$ that has profit at least P^*. If this event holds, Lemma 1 provides an algorithm with the desired time complexity. □

4 The random subset sum problem and lattice basis reduction

The algorithm that we present for the random subset sum problem is taken from [4], and is based on an algorithm for lattice basis reduction [5]. We shall start with some basic background on lattices (a minimum that suffices in order to understand the results of this section). The interested reader is advised to consult other sources (such as [6], on which the following presentation is based) for more information on lattices and the associated algorithms.

Let $a_1, \ldots, a_n \in Q^n$ be a set of n linearly independent vectors of dimension n, with rational entries. Let $A = (a_1, \ldots, a_n)$ be the associated n by n matrix with the vectors as its columns. The *lattice generated by* A, denoted by $L(A)$, is the set of all vectors that are integer linear combinations of columns of A. Namely, $L(A) = \{\sum_{i=1}^{n} \lambda_i a_i \mid \lambda_i \in Z\}$. The vectors $a_1, \ldots, a_n$ form a *basis* for $L(A)$. The same lattice L may have several different bases (matrices A as above that generate it), but all basis matrices have the same determinant (up to sign). Hence we define the determinant of the lattice, $det(L)$ as $|det(A)|$, where A is any basis matrix for the lattice L. Geometrically, $|det(A)|$ is the volume of the parallelohedra spanned by the basis vectors. (E.g., for $n = 2$, it is the area of the parallelogram whose vertices are $\{(0,0), a_1, a_2, a_1 + a_2\}$.) Denoting the ℓ_2 norm of a vector a by $|a|$, we have Hadamard's inequality $|det(A)| \leq \Pi_{i=1}^{n} |a_i|$.

Given a lattice $L(A)$, the **basis reduction problem** is to find a basis $b_1, \ldots, b_n$ of $L(A)$ with $\Pi_{i=1}^{n} |b_i|$ as small as possible. This problem is NP-hard.

Given a lattice $L(A)$, the **shortest vector problem** (SVP) is to find a nonzero vector $b \in L(A)$ with $|b|$ as small as possible. This problem is NP-hard under randomized reductions [1]. The length of the shortest non-zero lattice vector in a lattice L is denoted by $\lambda(L)$. It is known that $\lambda(L) \leq O(\sqrt{n}) \cdot (det(L))^{1/n}$. There is no polynomial time algorithm known for finding a non-zero lattice vector of length $n^{O(1)} \cdot (det(L))^{1/n}$.

Lenstra, Lenstra and Lovasz [5] defined the concept of a *reduced basis* of a lattice. The exact definition involves bounds on the relations between the basis vectors and their Gram-Schmidt orthoganalization, and on the relations between norms of consecutive basis vectors, and will not be presented here. They proved additional results that imply the following theorem (and much more).

Theorem 4 *There is a polynomial time algorithm that given a basis matrix A for a lattice L, finds a reduced basis $B = (b_1, \ldots, b_n)$. Moreover, for every reduced basis, $|b_1| \leq 2^{(n-1)/2}\lambda(L)$ and $\Pi_{i=1}^{n}|b_i| \leq 2^{n^2/4} \cdot det(L)$.*

Later Schnorr [9] showed that for every fixed $\epsilon > 0$ one can find a basis with $|b_1| < (1+\epsilon)^n\lambda(L)$ and $\Pi_{i=1}^{n}|b_i| \leq (1+\epsilon)^n \cdot det(L)$, in time polynomial in n (the degree of the polynomial grows as ϵ decreases).

We now return to the random subset sum problem, and describe the algorithm of Lagarias and Odlyzko [4] for it.

Recall that the input to the problem is $(w_1, \ldots, w_n, W)$, that we wish to find a set S satisfying $\sum_{i \in S} w_i = W$, and that such a set is guaranteed to exist (the set R in the procedure of generating random subset sum instances). We assume that $W \neq 0$ as otherwise the empty set is the unique solution.

Let k be the smallest integer satisfying $k > 2^{n/2}\sqrt{n}$. Set up a lattice L of dimension $n+1$ whose basis vectors are:

$a_1 = (1, 0, \ldots, 0, k \cdot w_1)$

$a_2 = (0, 1, \ldots, 0, k \cdot w_2)$

$\ldots$

$a_n = (0, 0, \ldots, 1, k \cdot w_n)$

$a_{n+1} = (0, 0, \ldots, 0, k \cdot W)$

Observe that all vectors in this lattice have integer entries, and entries divisible by k in coordinate $n+1$.

The set R induces a vector v^R of norm $\sqrt{|R|} \leq \sqrt{n}$ in this lattice. This vector v^R is obtained by adding those vectors a_i for $i \in R$ and subtracting a_{n+1}. Hence, $v^R(i) = 1$ if $i \in R$, $v^R(i) = 0$ if $i \notin R$, and $v^R(n+1) = 0$.

When N is sufficiently large, it turns out that with high probability over the choice of the random subset sum instance, the planted R is the unique solution to the subset sum problem, and moreover, v^R is the shortest non-zero vector in the lattice L. In fact, all other vectors in the lattice either have norm exponentially larger than that of v^R, or are multiples of v^R. Consequently, using the algorithm of Theorem 4 (which we shall refer to as the LLL algorithm) to find in L a short non-zero vector, the returned vector x will be a multiple of v^R, and then R can easily be recovered from x.

We shall indeed use the above lattice and the LLL algorithm in order to find a vector x from which we recover R. However, our procedure for recovering R from x will be more complicated than necessary. In particular, it might use several (at most n) iterations of the LLL algorithm, each time on instances with fewer items of non-zero weight. The advantage of our procedure is that its analysis is somewhat simpler. We now present our procedure for recovering R from x.

Let x denote the solution returned by the LLL algorithm in the current iteration, and let S be the set of those coordinates of x whose value in x is non-zero.

1. If $n+1 \in S$, the algorithm fails.
2. If $\sum_{i \in S} w_i = W$, return S and end.
3. If $\sum_{i \in S} w_i > W$, the algorithm fails.

4. If $\sum_{i \in S} w_i < W$, include S as part of the solution, and set up a new subset sum instance to find the remaining part of the solution. In the new instance the weight of every item in S is changed to 0, and the weights of other items are unchanged. The new value of W is $W - \sum_{i \in S} w_i$. Run the next iteration with this new instance.

We now prove that if N is sufficiently large, the algorithm (with the recovery procedure as above) is unlikely to fail (over the choice of random subset sum instance).

Lemma 5 *With probability at least* $1 - 2^{(\frac{n^2}{2} + O(n \log n))}/N$ *over choice of random subset sum instance, the instance is such that the above algorithm cannot fail in its first iteration.*

Proof: Fix R (the set of items in the planted solution for the subset sum instance). Recalling that $\lambda(L) \leq |v^R| \leq \sqrt{n}$, Theorem 4 implies that $|x| \leq 2^{((n+1)-1)/2}\lambda(L) \leq 2^{n/2}\sqrt{n} < k$. Together with the fact that entries in coordinate $n+1$ are multiples of k, this implies that $n+1 \notin S$. In remains to show that $\sum_{i \in S} w_i \leq W$. For the sake of contradiction, suppose that $\sum_{i \in S} w_i > W$. Then there is a coordinate $i \notin R$ for which $x(i) \neq 0$. For $i \notin R$, let Y_i denote the set of all vectors in L for which coordinate i is non-zero, and the ℓ_2 norm is smaller than k. To prove the lemma, we need to prove that with high probability (over choice of random input instance) Y_i is empty.

Every coordinate of a vector in Y_i has absolute value smaller than k, and coordinate $n+1$ must be 0. Our proof considers all possible candidate vectors in Y_i. For simplicity, we take the set of candidate vectors to be all vectors x satisfying $x = (x(1), \ldots, x(n), 0) \in [-k+1, k-1]^{n+1}$, even though this set includes vectors whose norm is too large. Hence the number of candidate vectors is at most $(2k)^n$. For each candidate vector we bound from above the probability (over choice of random subset sum instance) that it is indeed a lattice vector. Observe that $x = (x(1), \ldots, x(n), 0)$ is a lattice vector if and only if $\sum_{i=1}^{n} w(i) \cdot x(i) = c \cdot W$ for some integer c (and then $x = \sum_{i=1}^{n} x(i) \cdot a_i - c \cdot a_{n+1}$).

Observe that for every vector $x = (x(1), \ldots, x(n), 0) \in [-k+1, k-1]^{n+1}$, it holds that $|\sum_{i=1}^{n} w(i) \cdot x(i)| < 2N \cdot k \cdot n$. Let m denote the number of different integer multiples of W in the range $[-2kNn, 2kNn]$. Observe that $m \leq 4kn$, as $W > N$.

Consider a candidate vector $x = (x(1), \ldots, x(n), 0)$, and $i \notin R$ such that $x(i) \neq 0$. In the random subset sum instance, first pick at random all value w_j for $j \neq i$. This already determines W. Let c be an integer such that $-2kNn \leq cW \leq 2kNn$, and recall that there are at most m such choices of c. Consider the bad event $b_{i,c}$ that $\sum_{j=1}^{n} w_j \cdot x(j) = c \cdot W$. For $b_{i,c}$ to hold, we need to pick w_i so that $w_i = \frac{1}{x(i)} \left(c \cdot W - \sum_{j \neq i} w_j \cdot x(j) \right)$. There is at most one integer value for w_i that satisfies this last equality. Now pick w_i at random in the range $[N+1, 2N]$. The probability that $b_{i,c}$ holds is at most $\frac{1}{N}$. Summing over all possible values for c, the probability that at least one bad event $b_{i,c}$ holds is at most $\frac{m}{N} \leq \frac{4kn}{N}$. Taking a union bound over all candidate vectors, the probability of a bad event is at most $(2k)^n \frac{4kn}{N} < 2^{(\frac{n^2}{2} + O(n \log n))}/N$. $\square$

Theorem 6 *With probability at least* $1 - 2^{(\frac{n^2}{2} + O(n \log n))}/N$ *over choice of random subset sum instance, the instance is such that the above algorithm finds a feasible solution.*

Proof: Observe that the proof of Lemma 5 shows that with probability at least $1 - 2^{(\frac{n^2}{2}+O(n \log n))}/N$, the sets Y_i defined in that proof are empty. We show that if these sets are empty, the algorithm cannot fail (in any of its iterations).

Suppose that the algorithm fails to recover R. Let $\ell > 1$ be the first iteration in which S of that iteration contains a coordinate not in R. Modify the vector x returned in iteration ℓ so as to get a vector x', by adding 1 in every coordinate that belongs to a set S that was returned in an iteration before ℓ. (Without loss of generality, all these coordinates are 0 in x, because the weights of the corresponding items are 0 in the instance that x is supposed to solve.) We have that $|x'| \leq |x| + n$. For $\ell > 1$ we have that $|x| \leq 2^{n/2}\sqrt{n-1} \leq k - n$ (the last inequality holds only for large enough n, but the proof of the theorem can be made to work for all n by making minor modifications to the algorithm). Hence $|x'| \leq k$ and $x' \in Y_i$, contradicting the assumptions that Y_i is empty. $\square$

Summarizing, if N is sufficiently large ($N \geq 2^{n^2}$ suffices if n is sufficiently large), the LLL algorithm solves (with high probability over choice of instance) random subset sum instances with planted solutions.

5 Homework

1. Give an algorithm that solves the knapsack problem in time polynomial in $(n, \max_i p_i)$. (Such an algorithm is useful when profits are polynomially bounded but weights are not.)

2. In our random planted model for the subset sum problem, the weights are chosen independently uniformly at random in the range $[N+1, 2N]$. Adapt the analysis of the algorithm to the case that the range is $[1, N]$.

3. Suppose that you are given an algorithm A that finds the optimal solution to the shortest lattice vector problem (SVP) in time $t(n)$.

   Consider the following problem, referred to here as *small even set* (SES). The input is a set S of m vectors $v_1, \ldots v_m$, with each v_i in $\{0,1\}^d$. The goal it to find a non-empty small subset $S' \subset S$ of vectors such that $\sum_{i \in S'} v_i$ gives a vector all whose entries are even. (If addition is performed modulo 2, this corresponds to a 0-vector, and hence a linear dependency in the finite field $(GF_2)^d$. In particular, this implies that $|S'|$ need not be larger than $d+1$.) Suppose that S is such so that there are two small disjoint subsets, S_1 and S_2, such that $\sum_{i \in S_1} v_i = \sum_{j \in S_2} v_j$. (Observe that $S_1 \cup S_2$ is a solution to the SES problem, but there may be other solutions to the SES problem that cannot be decomposed into S_1 and S_2 as above.) Show that algorithm A (which does not know S_1 and S_2) can be used in order to find a solution for the SES problem of size at most $|S_1| + |S_2|$, in time $t(m+d) \cdot (m+d)^{O(1)}$.

   (The above question is motivated by an approach of [3] for refuting random instances of 3SAT.)

The algorithm presented for the smoothed model for the knapsack problem was based on dynamic programming. The algorithm presented for the random model for subset sum was based on lattice basis reduction. The next set of questions are somewhat open ended, and can serve as a basis for discussions.

1. Can the dynamic programming approach replace the lattice basis reduction approach in the setting of the random model for subset sum? If yes, how? If no, why not?

2. Can the lattice basis reduction approach replace the dynamic programming approach in the setting of the smoothed model for the knapsack problem? If yes, how? If no, why not?

Another open ended question is the following. Propose a smoothed model for the subset sum problem, and an interesting range of parameters (for the model) for which a polynomial time algorithm solves instances in your model (with high probability, over choice of random smoothing).

References

[1] Miklos Ajtai: The Shortest Vector Problem in L_2 is NP-hard for Randomized Reductions. STOC 1998: 10–19.

[2] Rene Beier, Berthold Vocking: Random knapsack in expected polynomial time. J. Comput. Syst. Sci. 69(3): 306–329 (2004).

[3] Uriel Feige, Jeong Han Kim, Eran Ofek: Witnesses for non-satisfiability of dense random 3CNF formulas. FOCS 2006: 497–508.

[4] Jeffrey Lagarias and Andrew Odlyzko: Solving Low-Density Subset Sum Problems. J. ACM 32(1): 229–246 (1985).

[5] Arjen K Lenstra, Hendrik Willem Lenstra, Laszlo Lovasz: Factoring polynomials with rational coefficients. Mathematische annalen 261, 515–534 (1982).

[6] Laszlo Lovasz: An Algorithmic Theory of Numbers, Graphs and Convexity, SIAM (1986). https://epubs.siam.org/doi/book/10.1137/1.9781611970203

[7] Ketan Mulmuley, Umesh V. Vazirani, Vijay V. Vazirani: Matching is as easy as matrix inversion. Comb. 7(1): 105–113 (1987).

[8] Tim Roughgarden (Editor): Beyond the Worst-Case Analysis of Algorithms. Cambridge: Cambridge University Press (2021).

[9] Claus-Peter Schnorr: A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms. Theor. Comput. Sci. 53: 201–224 (1987).