Lecture 1 — September 11th, 2018

Lecturer G. Hanrot, E. Kirshanova *Scribe: Dániel Szilágyi*

Conventions and notation

Throughout the course, we use several conventions for the notation. The most frequently used ones are listed below:

Logarithms: Although in principle it does not matter, the logarithms are assumed to be binary, i.e. we write $\log(\cdot)$ for $\log_2(\cdot)$.

Indexing and counting: Array indices usually start from 1.

Soft-$\mathcal{O}$ notation: For brevity, sometimes we hide the logarithmic factors in asymptotic complexities. For example, $\widetilde{\mathcal{O}}(g(n)) = \mathcal{O}(g(n) \operatorname{poly} \log g(n))$, so $\widetilde{\mathcal{O}}(2^n)$ hides $\operatorname{poly}(n)$ factors.

Integers modulo N: $\mathbb{Z}_N$ denotes the ring of integers modulo N, $\mathbb{Z}/N\mathbb{Z}$.

Range of indices: $[n, m]$ denotes the set $\{n, n+1, \ldots, m\}$, and $[n]$ is a shorthand for $[1, n]$.

1 The Subset Sum (0-1 Knapsack) Problem

Definition 1 (Subset Sum Problem). *In the Subset Sum (abbreviated as SS) Problem, we are*

Given $a_1, \ldots, a_n, S \in \mathbb{N}$, *and we need to*

Find $I \subseteq [n]$ *such that* $\sum_{i \in I} a_i = S$.

Note that this is the *computational* (search) version of the SS Problem. The *decision* version asks whether there exists an index set i such that $\sum_{i \in I} a_i = S$. Formulated this way, the decision problem is NP-complete.

The search and decision problems are "equivalent" in the sense that (for the non-trivial direction) we can use an oracle for the decision problem to solve the computational problem with n calls to that oracle. More precisely, the reduction is achieved using Algorithm 1.

Definition 2 (Density). *The* density d *of a SS-instance is defined as* $d = \frac{n}{\log(\max_i a_i)}$

Intuitively, the density can be interpreted as the expected number of solutions of a random SS instance where a_i are chosen from the range $[0, 2^n - 1]$. We distinguish three types of SS instances based on their density:

$d \ll 1$ these are the *low-density* or *sparse* instances. Earlier, they were used for cryptographical purposes, but have since been broken [1].

Algorithm 1 Oracle reduction of computational SS to decision SS

Input: $a_1, \ldots, a_n, S \in \mathbb{N}$

Output: $I \subseteq [n]$ such that $\sum_{i \in I} a_i = S$

```
I ← ∅
for k ∈ [n] do
    if ∑_{i∈I} a_i = S then
        return I
    I' ← I ∪ {k}, S' = S − a_k
    if ! Decision-SS-Oracle({a_i | i ∈ [n] \ I'}, S') then
        I ← I', S ← S'
return no solution found
```

$d \gg 1$ these are the *high-density* instances, can be used for some hash functions.

$d \approx 1$ these are the hardest instances, it can be proven that for $d \in [1, 1.09]$ the problem is NP-complete.

Definition 3 (Modular Subset Sum Problem). *In the Modular Subset Sum (modular-SS) problem, we are*

Given $a_1, \ldots, a_n, S \in \mathbb{N}$*, a modulus* N*, and we need to*

Find $I \subseteq [n]$ *such that* $\sum_{i \in I} a_i \equiv S \mod N$*.*

Immediately, we note that up to poly(n)-factors, modular-SS and SS over the integers are "equivalent". More precisely, if we are given an oracle that solves modular-SS, we can solve an ordinary SS instance by calling the modular-SS oracle with $(a_i)_i, S, N = \max(\sum a_i, S) + 1$ as input. In the other direction, without loss of generality, assume that we have a modular-SS instance with modulus N, and values $a_i, S \in [0, N-1]$. We note that any sum of at most n a_i's is in $[0, n(N-1)]$, so we can just call the SS oracle for all target sums in $\{S, S+N, \ldots, S+(n-1)N\}$.

2 Asymptotics for binomial coefficients

In this section, we prove some asymptotic results for binomial coefficients, that will be useful in later analysis.

Lemma 4. *For all* $0 \leq \alpha \leq 1$*, we have*

$$\binom{n}{\alpha n} = \widetilde{\Theta}\left(2^{nH(\alpha)}\right),$$

where $H(\alpha) = -\alpha \log \alpha - (1-\alpha)\log(1-\alpha)$ *is the binary entropy function.*

Proof. We recall that Stirling's formula gives us the asymptotic approximation $n! \approx \sqrt{2\pi n}\left(\frac{n}{r}\right)^n$ (or, more precisely, $n! = (1+o(1))\sqrt{2\pi n}\left(\frac{n}{r}\right)^n$). So, by substituting this into

$$\binom{n}{\alpha n} = \frac{n!}{(\alpha n)!((1-\alpha)n)!},$$

we obtain

$$\begin{aligned}\binom{n}{\alpha n} &= \widetilde{\Theta}\left(\frac{(n/e)^n}{(\alpha n/e)^{\alpha n}((1-\alpha)n/e)^{(1-\alpha)n}}\right)\\ &= \widetilde{\Theta}\left(2^{n\log\frac{n}{e}-\alpha n\log\frac{\alpha n}{e}-(1-\alpha)n\log\frac{(1-\alpha)n}{e}}\right)\\ &= \widetilde{\Theta}\left(2^{n(\log n-\log e-\alpha\log(\alpha n)+\alpha\log e-(1-\alpha)\log((1-\alpha)n)+(1-\alpha)\log e)}\right)\\ &= \widetilde{\Theta}\left(2^{n(\log n-\alpha\log\alpha-\alpha\log n-(1-\alpha)\log(1-\alpha)-(1-\alpha)\log n)}\right)\\ &= \widetilde{\Theta}\left(2^{nH(\alpha)}\right).\end{aligned}$$

□

Corollary 5. *For $0 \le \alpha \le \beta \le 1$, we have*

$$\binom{\beta n}{\alpha n} = \binom{\beta n}{\frac{\alpha}{\beta}\beta n} = \widetilde{\Theta}\left(2^{H(\alpha/\beta)\beta n}\right).$$

3 Algorithms for SS

In this section, we assume that a solution always exists, and we are interested in finding 1 solution (otherwise, our complexity would depend on the actual number of solutions). Furthermore, we assume that for a given index set I, we can compute $\sum_{i\in I} a_i$ in $\mathcal{O}(\text{poly}(n))$ time.

We present 3 algorithms: brute-force, meet in the middle, and a simplified version due to [3] of Schroeppel-Shamir algorithm originally presented [4].

3.1 Brute-Force

Algorithm 2 just tests all $I \subseteq [n]$ with $|I| = n/2$. This is actually enough since if $|I| > n/2$, we can simply run the algorithm on input $(a_1 \dots a_n, \sum_{i=1}^n a_i - S)$, and take the complement of the returned solution.

Algorithm 2 Brute-force algorithm for SS

Input: $a_1, \dots, a_n, S \in \mathbb{N}$
Output: $I \subseteq [n]$ such that $\sum_{i\in I} a_i = S$

1: **for** $t \in [n/2]$ **do**
2:     **for all** $I \subseteq [n]$ s.t. $|I| = t$ **do**
3:         check if $\sum_{i\in I} a_i = S$

Theorem 6. *Algorithm 2 solves the SS problem in time*

$$T(\textit{Brute-Force}) = \widetilde{\mathcal{O}}\left(\sum_{t=1}^{n/2}\binom{n}{t}\right) = \widetilde{\mathcal{O}}\left(2^n\right),$$

using memory

$$M(\textit{Brute-Force}) = \mathcal{O}\left(n \log\left(\max_i a_i\right)\right).$$

3.2 Meet-in-the-Middle (MitM)

Algorithm 3 is due to [2], and trades time for space. Without loss of generality, we assume that n is divisible by 4 and that $|I| = n/2$. The idea is to express

$$\sum_{i\in I} a_i = S \text{ as } \sum_{i\in I_1} a_i = S - \sum_{i \in I_2} a_i, \text{ where } I_1 \cup I_2 = I \text{ and } |I_1| = |I_2| = \frac{n}{4}.$$

Algorithm 3 Meet-in-the-Middle algorithm for SS

Input: $a_1, \ldots, a_n, S \in \mathbb{N}$

Output: $I \subseteq [n]$ such that $\sum_{i\in I} a_i = S$

1: Randomly permute $a_1, \ldots, a_n$
2: $L \leftarrow \{\}$
3: **for all** $I_1 \subset \left[1, \frac{n}{2}\right]$ s.t. $|I_1| = \frac{n}{4}$ **do**
4: $\quad L \leftarrow L \cup \left\{\left(I_1, \sum_{i\in I_1} a_i\right)\right\}$
5: Sort L with respect to the 2nd coordinate
6: **for all** $I_2 \subset \left[\frac{n}{2}+1, n\right]$ s.t. $|I_2| = \frac{n}{4}$ **do**
7: $\quad$ **if** $\exists i$ s.t. $L[i][2] = S - \sum_{i\in I_2} a_i$ **then**
8: $\quad\quad$ **return** $I = L[i][1] \cup I_2$
9: If no solution found, go to step 1

Theorem 7. *Algorithm 3 is correct and it runs in time*

$$T(\mathit{MitM}) = \widetilde{\mathcal{O}}\left(2^{n/2}\right),$$

and space

$$M(\mathit{MitM}) = \widetilde{\mathcal{O}}\left(2^{n/2}\right).$$

Proof. At step 1, the algorithm requires a permutation π of a_i's such that $\left|I \cap \left[1, \frac{n}{2}\right]\right| = \frac{n}{4}$. Using Corollary 5, we can compute that such an event occurs with probability

$$\Pr\{\pi\} = \frac{\binom{n/2}{n/4}\binom{n-n/2}{n/4}}{\binom{n}{n/2}} = \widetilde{\Omega}\left(\frac{2^{n/2}\cdot 2^{n/2}}{2^n}\right) = \Omega\left(\frac{1}{\operatorname{poly} n}\right).$$

Therefore, we only need to reshuffle poly(n) times. Apart from that, we have the following bounds for the runtime:

$\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ for constructing L,

$\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ for sorting L,

$\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ for finding a match in step 7.

The only memory we use is for storing L, so it is bounded by $|L| = \widetilde{\mathcal{O}}\left(2^{n/2}\right)$. $\square$

At this point, we can also drop the assumptions on n being divisible by 4, and $|I|$ being exactly $n/2$, by running Algorithm 3 for all $|I| \le n/2$ and adjusting $|I_1|$ and $|I_2|$ appropriately. This only affects polynomial prefactors.

3.3 Schroeppel-Shamir

Algorithm 4 can be regarded as a memory-efficient version of MitM. Here, we describe the simplified version of the algorithm, due to [3].

The main idea is to split the sum $\sum_{i \in I} a_i = S$ into 4 sums, i.e. to partition I as $I = I_1 \cup I_2 \cup I_3 \cup I_4$ such that

$$\begin{aligned}
L_1 &= \left\{ \left(I_1 \subset \left[1, \frac{n}{4}\right], \sum_{i \in I_1} a_i \right) \right\}, \\
L_2 &= \left\{ \left(I_2 \subset \left[\frac{n}{4}+1, \frac{n}{2}\right], \sum_{i \in I_1} a_i \right) \right\}, \\
L_3 &= \left\{ \left(I_3 \subset \left[\frac{n}{2}+1, \frac{3n}{4}\right], \sum_{i \in I_1} a_i \right) \right\}, \\
L_4 &= \left\{ \left(I_4 \subset \left[\frac{3n}{4}+1, n\right], \sum_{i \in I_1} a_i \right) \right\},
\end{aligned}$$

where $|L_i| = \mathcal{O}\left(2^{n/4}\right)$. Thus, the SS problem amounts to finding a 4-tuple $(\sigma_1, \ldots, \sigma_4) \in L_1 \times \cdots \times L_4$ satisfying

$$\sigma_1[2] + \sigma_2[2] = S - \sigma_3[2] - \sigma_4[2].$$

This implies that there exists an integer σ_N and an appropriately chosen modulus N (to be defined later) such that

$$\sigma_N = \sigma_1[2] + \sigma_2[2] \bmod N = S - \sigma_3[2] - \sigma_4[2] \bmod N.$$

In fact, given N, we can just try all possible values for σ_N from $[0, N-1]$.

Note that we can construct L_{12} efficiently by creating a list

$$L_2(N) = \{(L_2[i][2] \bmod N, i) \mid i \in [|L_2|]\}.$$

If this list is sorted by the first coordinate of its elements, for any $\sigma_1 \in L_1$ we can efficiently find the index of the corresponding $\sigma_2 \in L_2$ such that $\sigma_1[2] + \sigma_2[2] \equiv \sigma_N \bmod N$ by performing a binary search for $\sigma_N - \sigma_1[2] \bmod N$ in $L_2(N)$. This way, we can construct L_{12} (and L_{34}) in time $\widetilde{\mathcal{O}}\left(\max\{2^{n/4}, |L_{12}|\}\right)$.

Algorithm 4 Schroeppel-Shamir algorithm for SS

Input: $a_1, \ldots, a_n, S \in \mathbb{N}$

Output: $I \subseteq [n]$ such that $\sum_{i \in I} a_i = S$

Construct the lists $L_1, \ldots, L_4$

Choose $N \xleftarrow{\$} \left[2^{(1/4-\epsilon)n}, 2 \cdot 2^{(1/4-\epsilon)n}\right]$ (choose N to be a random value from this range)

for all $\sigma_N \in [0, N-1]$ **do**

  $L_{12} \leftarrow \{(\sigma_1, \sigma_2) \in L_1 \times L_2 \mid \sigma_1[2] + \sigma_2[2] = \sigma_N \mod N\}$

  Sort L_{12} with respect to $\sigma_1[2] + \sigma_2[2]$.

  for all $(\sigma_3, \sigma_4) \in L_3 \times L_4$ s.t. $S - \sigma_3[2] - \sigma_4[2] = \sigma_N \bmod N$ **do**

    if $S - \sigma_3[2] - \sigma_4[2]$ appears in L_{12} **then**

      return $\sigma_1[1] \cup \sigma_2[1] \cup \sigma_3[1] \cup \sigma_4[1]$

Now, the main question that arises when analyzing this algorithm is how large is L_{12}. Of course, it depends on N: the larger N is, the smaller is L_{12}, but we have more values to try for σ_N. The following claims point us into that direction.

Theorem 8. *For any set $\mathcal{B} \subseteq \mathbb{Z}_N^n$ and $c, a_1, \ldots, a_n \in \mathbb{Z}_N$, let $P_{a_1,\ldots,a_n}(\mathcal{B}, c)$ denote the probability that $\sum_{i=1}^n a_i x_i \equiv c \mod N$ for a random $(x_1, \ldots, x_n)$ drawn uniformly from $\mathcal{B}$, i.e.*

$$P_{a_1,\ldots,a_n}(\mathcal{B}, c) = \frac{1}{|\mathcal{B}|} \left| \left\{ (x_1, \ldots, x_n) \in \mathcal{B} \mid \sum a_i x_i \equiv c \bmod N \right\} \right|.$$

Then, the following holds:

$$\frac{1}{N^n} \sum_{(a_1,\ldots,a_n) \in \mathbb{Z}_N^n} \sum_{c \in \mathbb{Z}_N} \left(P_{a_1,\ldots,a_n}(\mathcal{B}, c) - \frac{1}{N} \right)^2 = \frac{N-1}{N|\mathcal{B}|}.$$

Corollary 9. *For any real $\lambda > 0$, the fraction of n-tuples $(a_1, \ldots, a_n) \in \mathbb{Z}_N^n$ for which there exists a $c \in \mathbb{Z}_N$ that satisfies $|P_{a_1,\ldots,a_n}(\mathcal{B}, c) - \frac{1}{N}| \geq \frac{\lambda}{N}$ is at most*

$$\frac{N^2}{\lambda^2 \cdot |\mathcal{B}|}.$$

Proof of the Corollary. By contradiction. Assume that $\exists c \in \mathbb{Z}_N$ that satisfies $|P_{a_1,\ldots,a_n}(\mathcal{B}, c) - \frac{1}{N}| \geq \frac{\lambda}{N}$ for strictly more than a $\frac{N^2}{\lambda^2 \cdot |\mathcal{B}|}$-fraction of $(a_1, \ldots, a_n)$'s. Let S be the set of such n-tuples. But then,

$$\begin{aligned}
\frac{1}{N^n} \sum_{(a_1,\ldots,a_n) \in \mathbb{Z}_N^n} \sum_{c \in \mathbb{Z}_N} \left(P_{a_1,\ldots,a_n}(\mathcal{B}, c) - \frac{1}{N} \right)^2 &\geq \frac{1}{N^n} \sum_{(a_1,\ldots,a_n) \in S} \sum_{c \in \mathbb{Z}_N} \left(P_{a_1,\ldots,a_n}(\mathcal{B}, c) - \frac{1}{N} \right)^2 \\
&\geq \frac{1}{N^n} \sum_{(a_1,\ldots,a_n) \in S} \sum_{c \in \mathbb{Z}_N} \left(\frac{\lambda}{N} \right)^2 \\
&\geq \frac{N^2}{\lambda^2 |\mathcal{B}|} \cdot \frac{\lambda^2}{N^2} = \frac{1}{|\mathcal{B}|} > \frac{N-1}{N|\mathcal{B}|},
\end{aligned}$$

a contradiction. □

Proof of the Theorem. Let $e_N = \exp\left(\frac{2\pi i}{N}\right)$. By summing the geometric series, we can show that for any $u \in \mathbb{Z}$

$$\sum_{\lambda=0}^{N-1} e_N^{u-\lambda} = \begin{cases} 0, & \text{if } u \not\equiv 0 \bmod N \\ N, & \text{else.} \end{cases}$$

Denote $\vec{a} = (a_1, \ldots, a_n)$, and $N_{\vec{a}}(\mathcal{B}, c) = |\mathcal{B}| \cdot P_{\vec{a}}(\mathcal{B}, c)$. Then,

$$N_{\vec{a}}(\mathcal{B}, c) = \frac{1}{N} \sum_{\vec{x} \in \mathcal{B}} \sum_{\lambda=0}^{N-1} e_N^{\lambda(\langle \vec{a}, \vec{x} \rangle - c)}.$$

If we fix $\lambda = 0$, we get

$$\frac{1}{N} \sum_{\vec{x} \in \mathcal{B}} e_N^0 = \frac{|\mathcal{B}|}{N}.$$

Therefore,

$$\begin{aligned}
\sum_{c \in \mathbb{Z}_N} \left(N_{\vec{a}}(\mathcal{B}, c) - \frac{|\mathcal{B}|}{N} \right)^2 &= \sum_{c \in \mathbb{Z}_N} \left(\frac{1}{N} \sum_{\lambda=1}^{N-1} e_N^{-\lambda c} \sum_{\vec{x} \in \mathcal{B}} e_N^{\lambda \langle \vec{a}, \vec{x} \rangle} \right)^2 \\
&= \frac{1}{N^2} \sum_{c \in \mathbb{Z}_N} \sum_{\lambda, \mu=1}^{N-1} e_N^{-c(\lambda+\mu)} \sum_{\vec{x}, \vec{y} \in \mathcal{B}} e_N^{\lambda \langle \vec{a}, \vec{x} \rangle + \mu \langle \vec{a}, \vec{y} \rangle} \\
&= \frac{1}{N^2} \sum_{\lambda, \mu=1}^{N-1} \sum_{\vec{x}, \vec{y} \in \mathcal{B}} e_N^{\lambda \langle \vec{a}, \vec{x} \rangle + \mu \langle \vec{a}, \vec{y} \rangle} \underbrace{\sum_{c \in \mathbb{Z}_N} e_N^{-c(\lambda+\mu)}}_{N \text{ if } \lambda \equiv -\mu \bmod N, \text{ else } 0} \\
&= \frac{1}{N^2} N \sum_{\lambda=1}^{N-1} \sum_{\vec{x}, \vec{y} \in \mathcal{B}} e_N^{\lambda(\langle \vec{a}, \vec{x} \rangle - \langle \vec{a}, \vec{y} \rangle)}.
\end{aligned}$$

Hence, we obtain

$$|\mathcal{B}|^2 \sum_{c \in \mathbb{Z}_N} \left(P_{\vec{a}}(\mathcal{B}, c) - \frac{1}{N} \right)^2 = \frac{1}{N} \sum_{\lambda=1}^{N-1} \sum_{\vec{x}, \vec{y} \in \mathcal{B}} e_N^{\lambda \langle \vec{a}, \vec{x} - \vec{y} \rangle}. \tag{1}$$

Now, we show that for any $\lambda \not\equiv 0 \bmod N$, the average of the inner sum over all $\vec{a} \in \mathbb{Z}_N^n$ is $|\mathcal{B}|$, i.e.

$$\sum_{\vec{a} \in \mathbb{Z}_N^n} \sum_{\vec{x}, \vec{y} \in \mathcal{B}} e_N^{\lambda \langle \vec{a}, \vec{x} - \vec{y} \rangle} = N^n |\mathcal{B}|. \tag{2}$$

We immediately see that summing just over $\vec{x}, \vec{y}$ such that $\vec{x} = \vec{y}$ we get the desired sum, and thus, we need to show that for $\vec{x} \neq \vec{y}$, the terms sum to 0. To do that, we can apply a more sophisticated summation of a geometric series we already encountered. WLOG, assume that $\vec{x}$ and $\vec{y}$ differ at least at position n, i.e. $x_n \neq y_n$. Then,

$$\sum_{\vec{a} \in \mathbb{Z}_N^n} e_N^{\lambda \langle \vec{a}, \vec{x} - \vec{y} \rangle} = \sum_{(a_1, \ldots, a_{n-1}) \in \mathbb{Z}_N^{n-1}} e_N^{\lambda \sum_{j=1}^{n-1} a_j (x_j - y_j)} \underbrace{\sum_{a_n=0}^{N-1} e_N^{\lambda a_n (x_n - y_n)}}_{=0}.$$

Therefore, by combining (1) and (2), we get the desired result. $\square$

Finally, we can state the theorem about the correctness and performance of the Schroeppel-Shamir algorithm.

Theorem 10. *For any $\epsilon > 0$ and modulus N close to $2^{(1/4-\epsilon)n}$, Algorithm 4 solves the SS problem in time*

$$T(\textit{Schroeppel-Shamir}) = \widetilde{\mathcal{O}}(2^{n/2}),$$

using memory

$$M(\textit{Schroeppel-Shamir}) = \widetilde{\mathcal{O}}(2^{n/4})$$

for at least a $\left(1 - 2^{-2\epsilon n}\right)$-fraction of SS instances.

Proof idea. Use Corollary 9 with $\lambda = 1/2$, $\mathcal{B} = \{0,1\}^{n/2}$ twice on L_{12} and L_{34}. We also need to use the fact that a random SS instance is close to a random modular SS instance with modulus N. $\square$

This is the best known algorithm for density-1 SS , with $T \cdot M = \widetilde{\mathcal{O}}\left(2^{3n/4}\right)$. Other space/time tradeoffs are possible as well.

References

[1] Alan M Frieze. On the Lagarias-Odlyzko Algorithm for the Subset Sum Problem. *SIAM Journal on Computing*, 15(2):536–539, 1986.

[2] Ellis Horowitz and Sartaj Sahni. Computing Partitions with Applications to the Knapsack Problem. *Journal of the ACM (JACM)*, 21(2):277–292, 1974.

[3] Nick Howgrave-Graham and Antoine Joux. New Generic Algorithms for Hard Knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 235–256. Springer, 2010.

[4] Richard Schroeppel and Adi Shamir. A $T = \mathcal{O}(2^{n/2}), S = \mathcal{O}(2^{n/4})$ algorithm for certain NP-complete problems. In FOCS, pages 328–336, 1979.