CS 3510 — Algorithms, Spring 2022

# Lecture 20: Subset-Sum

April 14, 2022

*Lecturer: Frederic Faulkner* *Scribe: Aditya Diwakar*

## 1 Subset-Sum

Given a set $S$ of integers and a target $t$, is there a subset $S'$ of $S$ such that the sum of the elements of $S'$ is exactly $t$.

**Theorem.** *Subset-Sum is NP-Complete.*

*Proof.* In order to show Subset-Sum is NP-Complete, we can start by showing that Subset-Sum is NP-Hard. In order to determine if it is NP-Hard, we want to reduce a known NP-Complete problem to this problem.

Assume we had some blackbox algorithm to solve Subset-Sum problem, can we use this algorithm to solve some other problem (find a reduction)? We can pick 3SAT and try to solve 3SAT by using Subset-Sum. This requires us to convert (reduce) the inputs to 3SAT into the inputs of a subset sum problem.

Given a 3SAT boolean expression input with $n$ variables and $m$ clauses, create a set $S$ with target $t$ such that some subset of $S$ adds to $t$ if and only if $F$ is satisfiable.

We will construct $S$ by having $2(m+n)$ elements where each variable/clause will have two elements in the set $S$.

$$S = \{\underbrace{v_1, \neg v_1, v_2, \neg v_2, \ldots, v_n \neg v_n}_{\text{corresponds to variables}}, \underbrace{z_1, \neg z_2, \ldots, z_m, \neg z_m}_{\text{corresponds to clauses}}\}$$

In terms of the reduction, if $x_i$ is true, we want to include $v_i$ in the sum (and never $v_i$) and if $x_i$ is false, we want to include $v_i'$ and never $v_i$.

We can set each of the values for the various $v_i, v_i', z_i, z_i'$ as $n+m$ digit numbers. The table on the right shows how we can define the first $n$ digits. Notice that $v_i$ and $v_i'$ have the same 1 in each column, so Subset-Sum would not pick both $v_i$ and $v_i'$ since $t$ for that column is only 1. However, we need to add additional restrictions on these numbers such that each of the clauses are satisfied as well. We have $m$ additional digits to use, so we will encode information with them. To describe the $m$ digits, here is an example:

| | | | | | | |
|---|---|---|---|---|---|---|
| $v_1$ | 1 | 0 | 0 | 0 | $\cdots$ | 0 |
| $v_1'$ | 1 | 0 | 0 | 0 | $\cdots$ | 0 |
| $v_2$ | 0 | 1 | 0 | 0 | $\cdots$ | 0 |
| $v_2'$ | 0 | 1 | 0 | 0 | $\cdots$ | 0 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\cdots$ | 0 |
| $v_n$ | 0 | 0 | 0 | 0 | $\cdots$ | 1 |
| $v_n'$ | 0 | 0 | 0 | 0 | $\cdots$ | 1 |
| $t$ | 1 | 1 | 1 | 1 | $\cdots$ | 1 |

$$(\neg x_1 \vee \neg x_2 \vee \neg x_3) \vee (\neg x_1 \vee \neg x_2 \vee x_3) \vee (x_1 \vee \neg x_3 \vee x_3)$$

In this, we have $n = 3$ variables and $m = 3$ clauses. Hence, we will have variables $v_1, v_1', v_2, v_2', v_3, v_3'$ and $z_1, z_1', z_2, z_2', z_3, z_3'$ for a total of $2(m+n) = 12$ variables.

| | | | | | | |
|---|---|---|---|---|---|---|
| $v_1$ | 1 | | | | | 1 |
| $v_1'$ | 1 | | | 1 | 1 | |
| $v_2$ | | 1 | | | | |
| $v_2'$ | | 1 | | 1 | 1 | 1 |
| $v_3$ | | | 1 | | 1 | 1 |
| $v_3'$ | | | 1 | 1 | | |
| $z_1$ | | | | 1 | | |
| $z_1'$ | | | | 1 | | |
| $z_2$ | | | | | 1 | |
| $z_2'$ | | | | | 1 | |
| $z_3$ | | | | | | 1 |
| $z_3'$ | | | | | | 1 |
| $t$ | 1 | 1 | 1 | 3 | 3 | 3 |

We define the first $n$ columns the same as before for all of the $v_i$ but have now introduced additional columns for the last $m$ digits. How do we fill these values? Simply put, in the $n+j$ column for any $v_i$, we put a 1 if $x_i \in c_j$ and for any $v_i'$, we put a 1 if $\neg x_i \in c_j$. For example, since $x_1 \in c_3$ and $\neg x_1 \in c_1, c_2$ and therefore the first two rows are 100001 and 100110. The remaining rows have these *buffer* variables $z_i$ and $z_i'$ for $c_i$. If the possible sum can be less than 3, we add 1s in these buffer positions such that we reach 3. These have been filled in as well on the table at the left. The table on the left also has 0s omitted so you can see which entries have a 1, but the remaining entries are 0.

In the above, we set $t = 111333$ which ensures that we are picking one of $v_1$ or $v_1'$ (the first 3 digits). This is equivalent to setting each literal to either true or false. The remaining digits are 3 to ensure that the clauses are satisfied. If it is not possible to make the sum exactly 3, it would not be possible to satisfy the original boolean expression. Hence, we have shown how we can convert from SAT to Subset-Sum. The correctness proof for this reduction is an exercise left to the reader. We have proved this is NP-Hard. The reader should show this is in NP.

# 2 Challenges

## 2.1 Exact Knapsack

Given a list of items with corresponding values and weights, can we get a value exactly $V$ with a weight capacity of $B$?

**Theorem.** *Exact Knapsack is NP-Complete.*

*Proof.* We will show that Exact Knapsack is NP-Hard. It is an exercise that this is NP. Assume we have some blackbox algorithm for Exact Knapsack, then we claim that we can solve Subset-Sum.

The input to Subset-Sum is a set of items $S = \{s_1, \ldots, s_n\}$ and some value $t$ while Knapsack takes in a list of items with values and weights. If we have a blackbox algorithm for Exact Knapsack, we could set $V = t$, $B = 1$ and set the weight of each item to $0$ with the value being the value of that item itself, then we have reduced Subset-Sum to Exact Knapsack. The proof of this reduction is omitted.

## 2.2 General Knapsack

Given a list of items with corresponding values and weights, can we get a value $\geq V$ with a weight capacity of $B$?

**Theorem.** *Knapsack is NP-Complete.*

*Proof.* The proof follows similarly to Exact Knapsack except we set the value *and* weight of each element to the value from the Subset-Sum input. Since Knapsack returns whether or not it is possible to pick items under a weight $B$ with value $\geq V$, if we set $B = V = t$, we will try to get items with weight $\leq t$ and values $\geq t$. Since the weight and value is the same, if this is possible, then this must be exactly equal to $t$. The proof of this reduction and many of this proof's formalities are omitted, as are much of the formalities of this proof. However, this is a valid reduction, hence Knapsack is NP-Complete.