CS 3510 Algorithms 4/6/2023

Lecture 18: Subset Sum

Lecturer: Abrahim Ladha *Scribe(s): Joseph Gulian*

You may remember we talked about the KNAPSACK problem during our Dynamic Programming section. You should recall that we had trouble solving KNAPSACK with better than an exponential runtime. Today we'll talk about a problem similar to KNAPSACK called SUBSETSUM. We'll proceed to relate this back to KNAPSACK by showing that the two problems are both NP-COMPLETE.

1 SUBSETSUM

SUBSETSUMis a problem where you're given a list of numbers and you want to find a subset which can sum to some target number (think `2sum` but for arbitrarily sized sets). This is formalized by having a set S, a subset $S' \subset S$ and a target t, where $\sum_{s \in S'} s = t$. An exponential time dynamic programming solution is the best known approach to this problem. As it turns out, SUBSETSUM is actually NP-COMPLETE; we'll begin the proof.

First we need to show that SUBSETSUM is in the class NP. We take the subset as a witness and we take the sum over the set. If the subset sum is equal to the target, then the witness is valid. We have shown that SUBSETSUM is in the class NP.

Now we must show that SUBSETSUM is NP-HARD; we'll reduce from 3SAT. Overflow, the carrying of bits will make things complicated. We choose the base arbitrarily high such that there is no overflow.

We can model an exclusive or relationship by adding two ones together and testing whether they equal one or not. This is essentially the relationship between inverted literals, i.e., x_1 and $\overline{x}_1$. So we'll create numbers for each literal. We create the target such that it is one for all the literals, meaning either x_1 or $\overline{x}_1$ need to be true. If both the numbers for x_1 and $\overline{x}_1$ are selected to be in the subset, it can not be the target because the target has a one in the x_1 place (again no overflow).

	x_1	x_2	x_3
x_1	1	0	0
$\overline{x}_1$	1	0	0
x_2	0	1	0
$\overline{x}_2$	0	1	0
x_3	0	0	1
$\overline{x}_3$	0	0	1
t	1	1	1

Figure 1: Figure showing just variables and literals.

Now we need to add clauses; we'll use the clauses $(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$ as an example. We add clauses on columns so that rows can represent literals in clauses.

Then for each clause and literal, we set the place to be the number of times that literal appears in the clause. We just need to extend the target for the clauses.

	x_1	x_2	x_3	c_1	c_2	c_3
x_1	1	0	0	1	0	0
$\overline{x}_1$	1	0	0	0	1	1
x_2	0	1	0	1	1	0
$\overline{x}_2$	0	1	0	0	0	1
x_3	0	0	1	1	0	1
$\overline{x}_3$	0	0	1	0	1	0
t	1	1	1	x	x	x

Figure 2: Figure with literals for clauses with the number of times the literal appears in a clause.

We can't have it be three because that would imply that all three literals must be true. However if we make it one, then we can't have two be true. For instance the following 3SAT instance would not be able to be solved $x_1 \wedge (x_1 \vee x_2) \wedge x_2$ because the middle clause would need to have both of its literals be true. This is because we'd need to take both of the rows for the literals x_1 and x_2 in the subset, but this would make clause have a value of two which is not one. Then what do we do? Here comes our last trick: we add slack variables near the bottom and we require a clause to have three true variables. Since there are at three variables in 3SAT (we're doing 3SAT with exactly three literals per clause), we only need two slack variables, making it so that if one value is taken in the subset, it can use the two slack variables. If three values with a positive value for a clause are taken, no slack variables need to be taken.

	x_1	x_2	x_3	c_1	c_2	c_3
x_1	1	0	0	1	0	0
$\overline{x}_1$	1	0	0	0	1	1
x_2	0	1	0	1	1	0
$\overline{x}_2$	0	1	0	0	0	1
x_3	0	0	1	1	0	1
$\overline{x}_3$	0	0	1	0	1	0
s_1	0	0	0	1	0	0
s_1	0	0	0	1	0	0
s_2	0	0	0	0	1	0
s_2	0	0	0	0	1	0
s_3	0	0	0	0	0	1
s_3	0	0	0	0	0	1
t	1	1	1	3	3	3

Figure 3: Figure with slack variables.

Then we've completed our reduction for SUBSETSUM, and now we'll show the correctness. Say a 3SAT CNF instance is satisfiable, this would imply that there is a satisfying assignment of the variables such that all the clauses have at least one true literal. First, all literals by

definition are either true or false, meaning the columns with the variables will total one. Next we know that since there's a satisfying assignment, meaning the clause has at least one literal which is true. Since there's a taken literal, we know that each clause column is at least one. The slack variables could be used to make the total exactly three for each variable. Then the SubsetSum instance has this subset to sum to the target.

If 3Sat is not satisfiable, then some clause column does not have a true value, meaning even with the slack variables it can sum to at most two. Since this is for every subset, no subset sums to the target t. Then we've completed the proof that SubsetSum is in the class NP-Complete.

2 Knapsack

From here, showing Knapsack is in NP-Complete is very easy. We first formalize it as a decision problem

$$\text{Knapsack} = \{\langle I, W, V\rangle \mid I = \{(w_1, v_1), ..., (w_n, v_n) \text{ and } \sum v_i \geq V \text{ and } \sum w_i \leq W\}\}$$

This is obviously a generalization of subsetsum, so our reduction is simple. For each element $s_i \in S$, add item $(s_i, s_i) \in I$. Create a set of items where the weight and value of each item are equal. Then set $W = V = t$. Our transformation is quite literally $f(\langle S, t\rangle) = \langle S \times S, t, t\rangle$.

If $\langle S, t\rangle \in$ SubsetSum then there exists a subset $S' \subset S$ which sums to t. Then consider the set of items corresponding to this subset S'. The sum of the weights of these items are $\sum w_i = \sum s_i = t = W \leq W$. The sum of the values are $\sum v_i = \sum s_i = t = V \geq V$. So we see that $\langle S \times S, t, t\rangle \in$ Knapsack.

$\langle S, t\rangle \notin$ SubsetSum, then every single subset $S' \subset S$ has the sum of the items strictly greater than or less than t. If the sum of the items is greater than t, then $\sum w_i = \sum s_i > t = W$, so our capacity condition is not satisfied. If the sum of the items is less than t, then $\sum v_i = \sum s_i < t = V$ so our value condition is not satisfied. Either way, for all possible choice of items, one of the conditions never holds, so we see that $\langle S \times S, t, t\rangle \notin$ Knapsack.

This reduction also takes polynomial time, so we see that Knapsack is NP-hard. It is also obviously in NP, as your witness would be the subset of items, that you sum and check two inequalities. We conclude that Knapsack is Np-complete.