Donghyuk Kim
CSCI3390
Professor Straubing
December 13, 2018

Proof of Hardness: Subset-Sum Problem

Problem Statement

The subset-sum problem asks: given a set Q of positive integers and a target integer of t, is there a subset $Q' \subseteq Q$ whose elements sum to t? Problem defined as a language:

SUBSET-SUM = $\{\langle Q, t\rangle : there\ exists\ Q' \subseteq Q\ such\ that\ t = \sum_{q \in Q'} q\}$

For example, suppose we are given $Q = \{1, 2, 3, 4\}$ and $t = 6$. The subset $Q' = \{1,2,3\}$ is a "yes" instance to this problem because $1 + 2 + 3 = 6$.

Is SUBSET-SUM NP?

A problem is said to be in NP if it satisfies these two conditions:

1) The witness (yes instance of the problem) is polynomial sized.
   - Given an input of set Q with n integers, the witness is a subset Q' that can have at most n integers. Therefore, the witness is polynomial in the input size.
2) There exists an algorithm that verifies the witness in polynomial time.
   - Given a witness, all we need to do is add up all the elements and compare the value with the target integer t. There can be at most n-1 number of additions, and we operate one comparison. A pseudocode of this algorithm is shown below:

```
def sum_verify(Q′, t) :
        x = 0
        for each integer in Q′:
                add to x
        if x = = t:
                return True
        else:
                return False
```

Satisfying both conditions, the subset-sum problem is in NP. Is it in P? Probably not.

Though there exists an "efficient" dynamic programming algorithm that solves the subset-sum in pseudo-polynomial time, there is no known algorithm that solves the problem in polynomial time in the length of the input.

IS SUBSET-SUM NP-complete?

A problem is said to be in NP-complete if it is both NP and NP-hard.

We showed above that SUBSET-SUM is in NP. To prove that it is NP-hard, we need to show that every problem in NP can be reduced to $SUBSET-SUM$.

The Cook-Levin Theorem showed that the Boolean Satisfiability Problem SAT is NP-complete by showing that for any $L \in NP$, L $\leq_p$ SAT.

It is also known that SAT $\leq_p$ $3-SAT$. Since polynomial reductions are transitive, this tells us that

$3-SAT$ is also NP-hard.

We will show that $3-SAT \leq_p SUBSET-SUM$ to prove that $SUBSET-SUM$ is NP-hard.

Construction of $3-SAT \leq_p SUBSET-SUM$

$3-SAT$ formula ∅ has variables $x_1, x_2, \ldots, x_y$ and clauses $C_1, C_2, \ldots, C_z$.

Given ∅, we will construct an instance $\langle Q, t\rangle$ of $SUBSET-SUM$ such that ∅ is satisfiable if and only if $\langle Q, t\rangle$ is satisfiable. Our reduction will show construction of ∅ that has 3 variables and 4 clauses. Suppose $\emptyset = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$

For each variable x_i in ∅ we will create two integers k_i and k'_i in set Q.

For each clause C_j in ∅ we will create two integers s_j and s'_j in set Q.

For instance, ∅ with 3 variables and 4 clauses will create total of $3(2) + 4(2) = 14$ integers in set Q. Q = $\{k_1, k'_1, \ldots k_3, k'_3, s_1, s'_1, \ldots s_4, s'_4\}$.

Each of these integers will contain $y + z$ number of digits (where y is number of variables, z is number of clauses in $\emptyset$). Each digit will correspond to either one variable or one clause. The digits that correspond to the clauses will be the least significant digits, the digits that correspond to the variables will be the most significant digits. We will also create one target integer t following same rules. The construction we have made so far is illustrated by the table in figure 1 below. The bases have not been assigned a number yet and have been labeled N as a placeholder.

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
k_1	=	N	N	N	N	N	N	N
k'_1	=	N	N	N	N	N	N	N
k_2	=	N	N	N	N	N	N	N
k'_2	=	N	N	N	N	N	N	N
k_3	=	N	N	N	N	N	N	N
k'_3	=	N	N	N	N	N	N	N
s_1	=	N	N	N	N	N	N	N
s'_1	=	N	N	N	N	N	N	N
s_2	=	N	N	N	N	N	N	N
s'_2	=	N	N	N	N	N	N	N
s_3	=	N	N	N	N	N	N	N
s'_3	=	N	N	N	N	N	N	N
s_4	=	N	N	N	N	N	N	N
s'_4	=	N	N	N	N	N	N	N
t	**=**	**N**	**N**	**N**	**N**	**N**	**N**	**N**

Figure. 1

Now, the digits will be assigned a number with following rules:

1) For each variable x_i, we created integers k_i and k'_i. For both k_i and k'_i, assign '1' in the digit corresponding to variable x_i and '0' in digits corresponding to all other variables. This rule will assign all the digits in the upper-left shaded region of the table.

2) For each clause C_j, we created integers s_j and s'_j. For s_j, assign '1' in the digit corresponding to clause C_j and '0' in the digits corresponding to all other clauses. For s'_j, assign '2' in the digit corresponding to clause C_j and '0' in the digits corresponding to all

other clauses. This rule will assign all the digits in the bottom-right shaded region of the table.

3) For all integers s_j and s'_j that were created by the clauses, assign '0' in each of the digits corresponding to the variables. This rule will assign all the digits in the bottom-left non-shaded region of the table.

4) For target integer t, assign '1' in the digits corresponding to variables and assign '4' in the digits corresponding to variables

Using these four rules, we filled up the table as shown below in figure 2.

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
k_1	=	1	0	0	N	N	N	N
k'_1	=	1	0	0	N	N	N	N
k_2	=	0	1	0	N	N	N	N
k'_2	=	0	1	0	N	N	N	N
k_3	=	0	0	1	N	N	N	N
k'_3	=	0	0	1	N	N	N	N
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	**=**	**1**	**1**	**1**	**4**	**4**	**4**	**4**

Figure 2.

These rules will apply no matter how many variables or clauses ∅ has, and the patterns shown in the table will be same.

The value of digits in upper-right region of the table depend on the formula ∅.

Recall, $\emptyset = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$

Note that C_1 refers to leftmost clause, as in: $\emptyset = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

For each variable x_i, we created integers k_i and k'_i.

For k_i if literal x_i appears in C_j, assign the digit corresponding to C_j with ‘1’. If literal x_i does not appear in C_j, assign the digit corresponding to C_j with ‘0’.

- For instance, let’s look at integer k_1. Literal x_1 appears in clauses C_1, and C_2. Therefore, digits of k_1 that correspond to C_1 and C_2 are assigned 1. The digits that correspond to other clauses are assigned 0.

For k'_i if literal $\overline{x_i}$ appears in C_j, assign the digit corresponding to C_j with ‘1’. If literal $\overline{x_i}$ does not appear in C_j, assign the digit corresponding to C_j with ‘0’.

Using these rules, we completed the construction as illustrated by the table in figure 3 below.

		x_1	**x_2**	**x_3**	**C_1**	**C_2**	**C_3**	**C_4**
k_1	=	1	0	0	1	1	0	0
k'_1	=	1	0	0	0	0	1	1
k_2	=	0	1	0	0	1	0	1
k'_2	=	0	1	0	1	0	1	0
k_3	=	0	0	1	1	0	1	0
k'_3	=	0	0	1	0	1	0	1
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	**=**	**1**	**1**	**1**	**4**	**4**	**4**	**4**

Figure 3.

Set Q contains 14 integers. Q = {1001100, 1000011, 100101, 101010, 11010, 10101, 1000, 2000, 100, 200, 10, 20, 1, 2}. Our target t = 1114444

$3 - SAT$ formula $\emptyset = (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$ is satisfiable if and only if $\langle Q, t \rangle$ of subset-sum problem is satisfiable, where Q = {1001100, 1000011, 100101, 101010, 11010, 10101, 1000, 2000, 100, 200, 10, 20, 1, 2} and t = 1114444. One satisfying assignment for $3-SAT$ formula $\emptyset$ is: $x_1 = T, x_2 = F, x_3 = F$

The corresponding witness (subset $Q' \subseteq Q$ $such\ that\ t = \sum_{q\,\in Q'} q$), is found as follows:

1) If x_i = T, put k_1 in Q'. If x_i = F, put k'_i in Q'.

2) If clause C_j has three satisfying literals (all T), put s_j in Q'. If clause C_j has two satisfying literals, put s'_j in Q'. If clause C_j has one satisfying literal, put both s_j and s'_j in Q'.

We find that Q' = { k_1, k'_2, k'_3, s'_1, s'_2, s_3, s'_3, s_4, s'_4}. Figure 4 below have shaded integers in Q'

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
k_1	=	1	0	0	1	1	0	0
k'_1	=	1	0	0	0	0	1	1
k_2	=	0	1	0	0	1	0	1
k'_2	=	0	1	0	1	0	1	0
k_3	=	0	0	1	1	0	1	0
k'_3	=	0	0	1	0	1	0	1
s_1	=	0	0	0	1	0	0	0
s'_1	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s'_2	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s'_3	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s'_4	=	0	0	0	0	0	0	2
t	**=**	**1**	**1**	**1**	**4**	**4**	**4**	**4**

Figure 4.

Q'={1001100, 101010, 10101, 2000, 200, 10, 20, 1, 2} and t = 1114444

This is a 'yes' instance because 1001100+101010+10101+2000+200+10+20+1+2=1114444.

Though we showed it the other way for illustration, the reduction really shows that because there is a 'yes' instance for the sub-set problem, there is also a 'yes' instance for the $3-SAT$ problem.

This reduction is operated in polynomial time: if y = number of variables, z = number of clauses, set Q contains 2y + 2z number of integers, each with y + z number of digits. Creating each digits takes polynomial time in y + z.

Proof of Correctness

If x_i = T, we put k_1 in Q'. If x_i = F, we put k'_i in Q'. In other words, we always include either k_i or k'_i in set Q' but never both. If we included both k_i and k'_i in Q', the digit corresponding to x_i will sum to 2, and we will never achieve our target, which has '1' in that digit.

Now, suppose ∅ has a satisfying assignment. This means that every clause has at least one satisfying literal. Recall from the construction above how we filled up the digits in the upper-right region of the table: if x_i appears in C_j, the digit of k_i that corresponds to C_j is assigned '1' and '0' otherwise. Conversely, if x'_i appears in C_j, the digit of k'_i that corresponds to C_j is assigned '1' and '0' otherwise. If every clause has at least one satisfying literal, then at least one of the integers k_i or k'_i will have '1' assigned in the digit that correspond to each clause.

For ∅ with satisfying assignment, each clause may have one, two, or three satisfying literals. Only using the integers k_i or k'_i the sum of each digit that correspond to each clause can sum up to 1, 2, or 3. To match the target digit value of 4, we add the 'helper' integers s_j or s'_j or both. s_j have '1' assigned in digit corresponding to C_j, and s'_j have '2' assigned in digit corresponding to C_j.

If the clause digit of all the variable integers sum up to 1, we add both clause integers corresponding to that clause digit to add 3. If the clause digit of all the variable integers sum up to 2, we add clause integer s' of the corresponding clause digit to add 2. If the clause digit of all the variable integers sum up to 3, we add clause integer s of the corresponding clause digit to add 1. Notice that the clause integers can only sum up to 3. If ∅ has no satisfying assignment, every assignment will result in at least one clause which none of the literals are satisfied. If the clause has no satisfying literal, none of the chosen variable integers in Q' will have '1' in the digit corresponding to that clause, and that digit will never sum up to 4.

Here is an example of 3-SAT formula that is unsatisfiable:

$$\emptyset = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

Using this formula, we construct $\langle Q, t\rangle \ of \ SUBSET - SUM$ illustrated by table below.

		x_1	x_2	x_3	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
k_1	=	1	0	0	1	1	1	1	0	0	0	0
k'_1	=	1	0	0	0	0	0	0	1	1	1	1
k_2	=	0	1	0	1	0	1	0	1	1	0	0
k'_2	=	0	1	0	0	1	0	1	0	0	1	1
k_3	=	0	0	1	1	1	0	0	1	0	1	0
k'_3	=	0	0	1	0	0	1	1	0	1	0	1
s_1	=	0	0	0	1	0	0	0	0	0	0	0
s'_1	=	0	0	0	2	0	0	0	0	0	0	0
s_2	=	0	0	0	0	1	0	0	0	0	0	0
s'_2	=	0	0	0	0	2	0	0	0	0	0	0
s_3	=	0	0	0	0	0	1	0	0	0	0	0
s'_3	=	0	0	0	0	0	2	0	0	0	0	0
s_4	=	0	0	0	0	0	0	1	0	0	0	0
s'_4	=	0	0	0	0	0	0	2	0	0	0	0
s_5	=	0	0	0	0	0	0	0	1	0	0	0
s'_5	=	0	0	0	0	0	0	0	2	0	0	0
s_6	=	0	0	0	0	0	0	0	0	1	0	0
s'_6	=	0	0	0	0	0	0	0	0	2	0	0
s_7	=	0	0	0	0	0	0	0	0	0	1	0
s'_7	=	0	0	0	0	0	0	0	0	0	2	0
s_8	=	0	0	0	0	0	0	0	0	0	0	1
s'_8	=	0	0	0	0	0	0	0	0	0	0	2
t	**=**	**1**	**1**	**1**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**

Figure 5.

There is no subset Q' whose elements add up to the target. The three most significant digits of the target value are all '1', which restricts us from putting both k_i and k'_i in Q'. This reflects the fact that a variable in 3-SAT formula can't be both true and false.

Satisfiability problem with 3 variables can have 8 different combinations of assignments. In the subset-sum problem, there are 8 different ways you can choose the variable integers while satisfying the three most significant digits of the target:

1) k_1 and k_2 and k_3 (digit C_8 sums to 0)
2) k_1 and k_2 and k'_3 (digit C_7 sums to 0)
3) k_1 and k'_2 and k_3 (digit C_6 sums to 0)
4) k_1 and k'_2 and k'_3 (digit C_5 sums to 0)
5) k'_1 and k_2 and k_3 (digit C_4 sums to 0)
6) k'_1 and k_2 and k'_3 (digit C_3 sums to 0)
7) k'_1 and k'_2 and k_3 (digit C_2 sums to 0)
8) k'_1 and k'_2 and k'_3 (digit C_1 sums to 0)

For all of these combinations of variable integers, one clause digit will sum to 0. For instance, k_1 and k_2 and k_3 are 10011110000 + 1010101100 + 111001010 = 1132212110. The least significant digit, which corresponds to C_8, is 0. This reflects the fact that if we assigned $x_1 = \mathrm{T}, x_2 = \mathrm{T}, x_3 = T$, clause 8 will have no satisfying literals.

Since $\langle Q, t \rangle$ of $SUBSET - SUM$ we created has no satisfying assignment, the $3 - SAT$ formula has no satisfying assignment.

Will this construction work with odd cases of 3-CNF formula such as clauses with tautology or clauses with less than 3 literals? let’s take a look.

$\emptyset = (x_1 \vee \overline{x_1}) \wedge (x_2 \vee \overline{x_3}) \wedge (x_3)$ This is a valid 3-CNF formula. One possible satisfying assignment is $x_1 = \mathrm{F}, x_2 = \mathrm{T}, x_3 = \mathrm{T}$

Using the same rules, we can construct $\langle Q, t \rangle$ of $SUBSET - SUM$ illustrated by the table below.

		x_1	x_2	x_3	C_1	C_2	C_3
k_1	=	1	0	0	1	0	0
k'_1	=	1	0	0	1	0	0
k_2	=	0	1	0	0	1	0
k'_2	=	0	1	0	0	0	0
k_3	=	0	0	1	0	0	1
k'_3	=	0	0	1	0	1	0
s_1	=	0	0	0	1	0	0
s'_1	=	0	0	0	2	0	0
s_2	=	0	0	0	0	1	0
s'_2	=	0	0	0	0	2	0
s_3	=	0	0	0	0	0	1
s'_3	=	0	0	0	0	0	2
t	**=**	**1**	**1**	**1**	**4**	**4**	**4**

Figure 6.

$Q' = \{k'_1, k_2, k_3, s_1, s'_1, s_2, s'_2, s_3, s'_3\} = \{100100, 10010, 1001, 100, 200, 10, 20, 1, 2\}$

t = 111444. The construction still works: 100100+10010+1001+100+200+10+20+1+2 = 111444.