NP-Completeness of Subset-Sum problem

Rahul R. Huilgol	11010156
Simrat Singh Chhabra	11010165
Shubham Luhadia	11010176

September 7, 2013

Problem Statement

In the SUBSET SUM problem, we are given a list of n numbers $A_1, ..., A_n$ and a number T and need to decide whether there exists a subset S $\subseteq$ [n] such that

$$\sum_{i \epsilon S} A_i = T$$

(the problem size is the sum of all the bit representations of all the numbers). Prove that SUBSET SUM is NP-complete.

Contents

a Intuition

We are reducing 3-SAT to EXACTLY 1 3-SAT which is further reduced to M (defined below). This M can be reduced to SUBSET-SUM problem.

M is defined as the problem of finding whether a solution exists for a set of k equations E_1 to E_k. Each equation E_i is of the form

$$a_{i1}x_1 + a_{i2}x_2 + ...a_{in}x_n = b$$

where b = [0 0 0...0 1] a_{ij} = [0 0 0...0 0] or [0 0 0...0 1].
All these are binary numbers of length $\lceil log(n) + 1 \rceil$.
Each of the variables x_i can only take the values 0 or 1.
i ϵ {1, 2, ..., k} and j ϵ {1, 2, ..., n}

3-SAT $\leq_p$ EXACTLY 1 3-SAT $\leq_p$ M $\leq_p$ SUBSET-SUM

b SUBSET-SUM is NP

b.1 Solution polynomial sized

A solution of the subset-sum problem is a set S of indices i from the set [n], which correspond to the A_is which sum up to the number T. So the solution size could be at most n. So it is polynomial in the input size.

b.2 Polynomial time verification

Once we have the set S, we can verify the solution by summing up the corresponding A_is and comparing this sum with T. The number of additions is at most n-1. So the addition and comparision can be done in polynomial time. Hence, SUBSET-SUM is in NP.

c Claim 1 : 3-SAT $\leq_p$ EXACTLY 1 3-SAT

In the EXACTLY ONE 3SAT problem, we are given a 3CNF formula ϕ and need to decide if there exists a satisfying assignment u for ϕ such that every clause of ϕ has exactly one TRUE literal.

Suppose Ψ is a 3-SAT expression. Therefore, it will be of the form

$$\Psi = C_1 \wedge C_2 \wedge ... \wedge C_k$$

where each clause

$$C_i = (x \vee y \vee z)$$

where i ϵ {1, 2, ..., k}

We would convert the expression Ψ to an EXACTLY ONE 3SAT expression ϕ by making the following changes for every clause of Ψ: For every clause $C_i = (x \vee y \vee z)$ in Ψ, introduce 6 new variables $a_x, b_x, a_y, b_y, a_z, b_z$ and form the equivalent clause

$$D_i = (\neg x \vee a_x \vee b_x) \wedge (\neg y \vee a_y \vee b_y) \wedge (\neg z \vee a_z \vee b_z) \wedge (a_x \vee a_y \vee a_z)$$

Claim : $C_i \equiv D_i$
Proof : For any particular assignment which satisfies C_i, we will choose our additional literals $(a_x, b_x, a_y, b_y, a_z, b_z)$ in such a way that exactly one of (a_x, a_y, a_z) will become true. Suppose ω is false, then the clause $(\neg\omega \vee a_\omega \vee b_\omega)$ becomes true automatically and so a_ω and b_ω are chosen to be false. If ω is true, then either one of a_ω or b_ω are made true depending on whether any other of $a_i s$ are true, as we want the last clause of D_i to be true. Also, we cannot have x, y and z all false as C_i is satisfied, so we don't need to consider the case where a_x, a_y and a_z all are false at the same time.

On the other hand, if the clause C_i is false, then all of x,y,z have to be false. This would mean that a_x, a_y and a_z are all false and so the last clause of D_i will become false and hence D_i will not be satisfied
Hence, when C_i has a satisfying assignment so does D_i and when former does not have a solution, latter is also false.

Also, since we are adding only 6 extra literals for each clause, so each D_i will be constructed in constant time, and therefore the whole reduction will take only polynomial time.

d Claim 2 : EXACTLY 1 3-SAT $\leq_p$ M

Suppose ϕ is a 3-SAT expression. Therefore, it will be of the form

$$\phi = C_1 \wedge C_2 \wedge ... \wedge C_k$$

where each clause

$$C_i = (x \vee y \vee z)$$

where i ϵ {1, 2, ..., k}

We convert an instance of this problem into an instance of the problem M by the following steps:
For each clause C_i, we will introduce an equation E_i of the form

$$a_{i1}x_1 + a'_{i1}x'_1 + a_{i2}x_2 + a'_{i2}x'_2 + ... + a_{in}x_n + a'_{in}x'_n = b$$

where n= total number of variables in ϕ

x_j=boolean variable corresponding to the Exactly1 3-SAT literal x_j
x'_j=boolean variable corresponding to the Exactly1 3-SAT literal $\neg x_j$

a_{ij} take values [0 0 ...0 0] if x_j is not present in the clause C_i
[0 0 ...0 1] if x_j is present in the clause C_i
a'_{ij} take values [0 0 ...0 0] if $\neg x_j$ is not present in the clause C_i
[0 0 ...0 1] if $\neg x_j$ is present in the clause C_i
b=[0 0 ...0 1]
length of all above binary numbers is $\lceil log(2n) + 1 \rceil$

If there exists a solution to ϕ, we get values of all the variables $(x_1, x_2, ..., x_n)$ from which we can get the corresponding values for M's variables $(x_1, x_2, ..., x_n, x'_1, x'_2, ..., x'_n)$. Since ϕ is satisfied, each of its clauses is satisfied. If we consider the equation E_i corresponding to the clause C_i, then the former would be true as of all the variables in the equation whose coefficient is a non-zero binary number (which means they are present in the clause) exactly one will be having the value 1, so the equation holds.

If there does not exist a satisfying solution to ϕ, then there must be atleast a single clause with a false value, so all literals in that clause will be false and hence the equation corresponding to that clause will not hold.

The length of each equation will be :
b and all the $a_{ij}s$ will take $\lceil log(2n) + 1 \rceil$ bits
the variables x_is will take 1 bit each
So, each equation will take $O(nlog(n))$ space.
Hence, the output takes polynomial time for printing.

During reduction, the computations involved are: 1. Checking whether a literal exists in a particular clause (to find the value of a_{ij}) takes constant time per literal. 2. Addition on n binary numbers for checking whether a solution satisfies the equation could take a $O(nlog(n))$ steps. So, computations can be done in polynomial time.
Hence proved.

e Claim 3 : M $\leq_p$ SUBSET-SUM

M is defined as the problem of finding whether a solution exists for a set of k equations E_1 to E_k. Each equation E_i is of the form

$$a_{i1}x_1 + a_{i2}x_2 + ...a_{in}x_n = b$$

where b = [0 0 0 ...0 1] a_{ij} = [0 0 0 ...0 0] or [0 0 0 ...0 1].
All these are binary numbers of length $\lceil log(n) + 1 \rceil$.
Each of the variables x_j can only take the values 0 or 1.
i ϵ {1, 2, ..., k} and j ϵ {1, 2, ..., n}

For every instance of the problem M, we create an instance of the Subset-Sum problem in the following way:
Define n numbers $A_1, A_2, ..., A_n$ such that $A_j = a_{1j}a_{2j}...a_{ij}...a_{kj}$ i.e. each A_j consists of $k\lceil log(n) + 1\rceil$ bits.
Also, define the number $T = bbb....b$ (k times) so T also has $k\lceil log(n)+1\rceil$ bits.

If the given set of equations have a solution, then form a subset S such that j ϵS if $x_j = 1$ in the satisfying solution. If $x_{m_1}, x_{m_2}, ..., x_{m_l}$ take value 1 in the satisfying solution (l $\leq n$), and $m_1, m_2, ..., m_l\ \epsilon\{1, 2, ..., n\}$, then

$$\sum_{t=m_1}^{m_l} a_{it} = b \qquad \forall i\epsilon\{1, 2, ..., k\}$$

Hence,

$$\sum_{j\epsilon S} A_j = (\sum_{t=m_1}^{m_l} a_{1t})(\sum_{t=m_1}^{m_l} a_{2t})...(\sum_{t=m_1}^{m_l} a_{kt})$$
$$= bb...b(ktimes)$$
$$\mathbf{=T}$$

On the other hand, if there is no solution for the given set of equations, then for any assignment of $(x_1, x_2, ..., x_n)$, atleast one of the equations wouldn't be satisfied, i.e.

$$\sum_{t=m_1}^{m_l} a_{it} = c$$

for some i, and $c \neq b$, therefore,

$$\sum_{j\epsilon S} A_j = (\sum_{t=m_1}^{m_l} a_{1t})(\sum_{t=m_1}^{m_l} a_{2t})...(\sum_{t=m_1}^{m_l} a_{kt})$$
$$= bb...bcb...b(ktimes)$$
$$\neq T$$

so we wouldn't get a solution for the subset-sum problem.
NOTE: The case where $c = 1b$ will never arise because we are taking $\lceil log(n) + 1\rceil$ bits and the n binary numbers $a_{ij}s$ could only go up till we get all ones in c, i.e. no carry overflow is possible.

For every instance of the problem M, we can print the corresponding subset-sum problem in $O(knlog(n))$ steps as T and each A_j takes n times $\lceil log(n)+1\rceil$ bits.
Also, during computation, in worst case, when we need to add all the A_js, we would need $O(n^2log(n))$ steps for total n-1 additions.
Hence proved.

f Conclusion

SUBSET-SUM is in NP
3-SAT $\leq_p$ EXACTLY 1 3-SAT $\leq_p$ M $\leq_p$ SUBSET-SUM
3-SAT is NP-Complete.
Using the transitivity of reduction:
From Claim-1, EXACTLY 1 3-SAT becomes NP-Complete.
From Claim-2, M becomes NP-Complete.
And finally from Claim-3, SUBSET-SUM becomes NP-Complete.

Hence Proved.