In the Name of God

Theory of Computer Science - HW5

Ali Fathi - 99204943 Instructor: Dr. Ebrahimi

1 SPACE to TIME relationship

Consider the Language $L \in \text{SPACE}(f(n))$. It means there exists some deterministic single-tape Turing machine M which decides L using $O(f(n))$ cells of its tape, when running on input w with $|w| = n$. Using $O(f(n))$ cells of the tape, the number of total configuration of M would be as follow:

$$|\mathcal{C}| = |Q| \times O(f(n)) \times 2^{O(f(n))} = 2^{O(\log f(n)) + O(f(n))} = 2^{O(f(n))}$$

In which $|Q|$ is the number of states of M, $O(f(n))$ is all possible places for the header and $2^{O(f(n))}$ is the number of all possible contents of tape. As M is a decider, it doesn't have any loop during its process, so it could't visit any configuration twice. Therefore the M's running time on w couldn't be more than the number of its configurations in $O(f(n))$ space usage, which is $2^{O(f(n))}$. Therefore M decides w in $O(2^{O(f(n))})$, then we would have $L \in \text{TIME}(2^{O(f(n))})$. ■

2 SAT Decision to Search

Consider O_{SAT} as an oracle for SAT decision problem and ϕ as a logical formula, consists of n variables $x_1, \cdots, x_n$. First, we check whether ϕ is satisfiable or not, by passing it to O_{SAT}. If it is not satisfiable, there isn't anything to do but otherwise, we provide the following polynomial-time procedure to find a satisfying assignment $\alpha_1, \cdots, \alpha_n$ for ϕ:

Result: a satisfying assignment $\alpha_1, \cdots, \alpha_n$ for ϕ
for $i = 1$ **to** n **do**
    (So far, variables $x_1, \cdots, x_{i-1}$ are assigned to $\alpha_1, \cdots, \alpha_{i-1}$)
    Assign x_i to 0, and simplify formula ϕ by inserting values $(\alpha_1, \cdots, \alpha_{i-1}, 0)$ instead of variables $(x_1, \cdots, x_i)$ to get ϕ_i ;
    if O_{SAT} *accepts* ϕ_i *as a satisfiable formula* **then**
        let $\alpha_i = 0$;
    else
        let $\alpha_i = 1$;
    end
end

Algorithm 1: Procedure for SAT assignment problem

Moving step by step, at each step we make sure that there exists some assignment to $x_{i+1}, \cdots, x_n$ such that ϕ would remain satisfied having $(x_1, \cdots, x_i) = (\alpha_1, \cdots, \alpha_i)$, and the formula couldn't be unsatisfiable in both of the cases $\alpha_i = 0$ and $\alpha_i = 1$ because the last step, has insured $x_{i+1}, \cdots, x_n$ to have some satisfying assignment. This procedure has n steps and in each step, building ϕ_i doesn't need much time (only to fill $x_1, \cdots, x_i$ by constants; which is $O(n)$) therefore it's running time would be polynomial with respect to $|\langle\phi\rangle|$ (and even linear). ■

---

3 CoNP

3.1 P $\subseteq$ NP $\cap$ CoNP

Consider an arbitrary language $L \in$ P. Then there exists a deterministic one-tape Turing machine D to decide L in polynomial-time. So the verifier $V(w, c) \coloneqq D(w)$, would verify $w \in L$ in polynomial-time (therefore $L \in$ NP) and the verifier $V'(w, c) \coloneqq 1 - D(w)$ (D in which accept and reject states are substituted) would verify $w \in \bar{L}$ (therefore $L \in$ CoNP). Thus, it is $L \in$ NP $\cap$ CoNP, and consequently we would have P $\subseteq$ NP $\cap$ CoNP. ■

3.2 P = NP $\Rightarrow$ NP=CoNP

As we showed in previous part, P $\subseteq$ NP $\cap$ CoNP so if P=NP, it says that NP $\subseteq$ NP $\cap$ CoNP and because of the set theoretical relation $A \cap B \subseteq A$,

we would have NP = NP$\cap$ CoNP, which implies that NP $\subseteq$ CoNP.
On the other hand, for every language $L \in$ CoNP, we have $\bar{L} \in$ NP=P, so we have a deterministic Turing machine D to decide L in polynomial-time. Therefore D' (the same D in which accept and reject states are substituted) would decide L in polynomial-time, which shows that $L \in$ P which means CoNP $\subseteq$ P=NP.
Putting these together, we would deduce that CoNP=NP. $\blacksquare$.

3.3 CoNP-Complete $\cap$ NP-Complete $\neq \emptyset \Rightarrow$ NP=CoNP

Suppose that language L is both NP-Complete and CoNP-Complete. We use a lemma to conclude the result:

Lemma. *For two languages $A \leq_P B$:*

1. *If $B \in$ NP, then we would have $A \in$ NP.*
2. *If $B \in$ CoNP, then we would have $A \in$ CoNP.*

Proof. Consider f as the polynomial-time mapping reduction from A to B.

1. $B \in$ NP implies that there is some nondeterministic Turing machine N_B to decide B in polynomial-time. We construct N_A which at first, maps w to $f(w)$ and then decides it using M_B; in which the whole process is done in polynomial-time. Therefore $A \in$ NP.

2. We have $A \leq_P B \Leftrightarrow \bar{A} \leq_P \bar{B}$, and as $B \in$ CoNP means $\bar{B} \in$ NP, from the last item we conclude $\bar{A} \in$ NP which proves that $A \in$ CoNP.

$\square$

Using this lemma, for every language $A \in$ NP we have $A \leq_P L \in$ CoNP which results in $A \in$ CoNP then NP $\subseteq$ CoNP. Also for every language $A \in$ CoNP we have $A \leq_P L \in$ NP which results in $A \in$ NP then NP $\subseteq$ CoNP. Gathering the result, we would have NP=CoNP. $\blacksquare$

---

4 NP and Sets

We have $L_1, L_2 \in NP$ which means there are nondeterministic Turing Machines N_1 and N_2 which decide L_1 and L_2 in polynomial-time respectively.

4.1 $L_1 \cup L_2$

We construct the nondeterministic Turing Machines N to decide $L_1 \cup L_2$ as follows:
"N on input w:

- Simulate N_1 nondeterministically on w.
- If accepted, accept the input.
- If rejected, simulate N_2 nondeterministically on w and output as the same way it does."

$L(N)$ is clearly $L_1 \cup L_2$ as N accepts the input iff either N_1 or N_2 accepts it. Also its running time is polynomial, because all the branches in N_1 have polynomial length with respect to $n = |w|$ (with maximum l_1), and also the branches in N_2 (with the maximum l_2). Therefore all the branches in N, which are at most with the length $l_1 + l_2$, have polynomial length with respect to $n = |w|$ so N decides $L_1 \cup L_2$ in polynomial-time. ■

4.2 $L_1 \cap L_2$

We construct N in a similar way for $L_1 \cup L_2$:
"N on input w:

- Simulate N_1 nondeterministically on w.
- If rejected, reject the input.
- If accepted, simulate N_2 nondeterministically on w and output as the same way it does."

N accepts the input iff both N_1 and N_2 accepts it therefore $L(N) = L_1 \cap L_2$. Also for the same reason as above, N's running time would be polynomial with respect to $n = |w|$ so $L_1 \cap L_2 \in \text{NP}$. ■

---

5 INDSET

$$\text{INDSET} = \{\langle G, k\rangle \mid G \text{ is an undirected graph with some } k \text{ independent nodes}\}$$

First, we show that INDSET is NP. Certifier c would be a set of k nodes which are independent. Verifier $V(\langle G, k\rangle, c)$, first checks the size of the c to be k, then checks those nodes to be independent in G. This job is done by checking $\frac{k(k-1)}{2}$ edges in G's adjacency matrix and is clearly polynomial with respect to $|\langle G\rangle|$. Therefore, V is a polynomial certifier which implies that INDSET $\in$ NP.
Second, we would show that 3SAT $\leq_P$ INDSET by introducing a polynomial-time mapping reduction f that $f(\langle\phi\rangle) = \langle G, k\rangle$ in which $\langle\phi\rangle \in$ 3SAT iff $\langle G, k\rangle \in$ INDSET. Consider the 3CNF formula $\langle\phi\rangle$ as follows:

$$\langle\phi\rangle = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_m \vee b_m \vee c_m)$$

In which a_i, b_i and c_i are literals over n boolean variables $x_1, \cdots, x_n$.
We construct G, a graph with $3m$ nodes that are divided into m clusters with 3 nodes in every of them. The nodes in cluster i, are named with literals a_i, b_i and c_i and are fully connected together, like what is shown in the following picture:

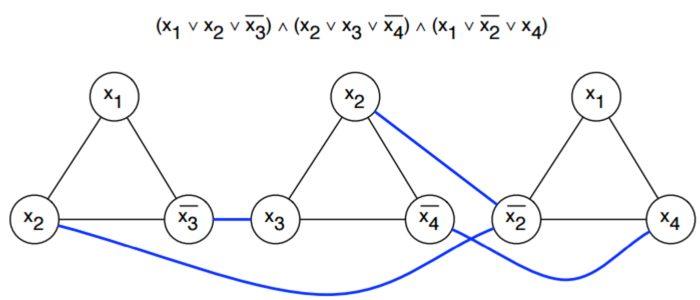

Figure 1: Reduction Example

In addition to internal edges, we connect all negated literals together too (e.g. every x_4 should be connected to all $\overline{x_4}$s). Finally we chose $k = m$ and the reduction would be completed be $\langle G, m\rangle$.
Our reduction is clearly a polynomial-time reduction. G is described by $3m$ nodes in $O(m)$ steps and by $3m$ internal edges and at most $O(m^2)$ negation edges in $O(m^2)$ steps. Also k would be introduced in $O(1)$ steps, then the total running time for f would be $O(m^2)$.
To show that our reduction is true, first we suppose that $\langle\phi\rangle \in$ 3SAT which

means there is a satisfying assignment for SAT, including at least one TRUE literal in each clause. From each clause, we chose one of the TRUE literals $l_i \in \{a_i, b_i, c_i\}$. There are no internal edges between $l_1, l_2, \cdots, l_m$ because they belong to different clusters, also no negation edges because they are all true and no pairs of them could be negated literals. This means nodes $l_1, l_2, \cdots, l_m$ make a m-nodes independent set in G, so $f(\langle\phi\rangle) = \langle G, k\rangle \in$ INDSET.
On the other hand, if m nodes $l_1, l_2, \cdots, l_m$ be a m-nodes independent set in G, an assignment with TRUE for these literals and FALSE for the others, would satisfy $\langle\phi\rangle$; Because not any pair of l_is could be for a same cluster (as clusters are fully connected) and also couldn't contain negated literals (which are connected in G). So, they are exactly from m distinct clusters and are consistent together, and would make every clause TRUE. ■

---

6 Spanning Tree

$$\text{SPANNING-TREE} = \{\langle G, l, u\rangle \mid G \text{ is an undirected weighted graph having some spanning tree with the weight between } l \text{ and } u\}$$

First, we show that SPANNING-TREE $\in$ NP. We take certifier c as a tree in G (or simpler, a list of edges). $V(\langle G, l, u\rangle, c)$, the verifier for SPANNING-TREE, first checks c to be a spanning tree (i.e. a list of connected edges containing all the nodes of G), then sums up the weights corresponding to these edges and accepts if this sum lies between l and u. First job is done in $O(n^2)$ (n is the number of edges in G) by a primary search on n nodes in the list, and then mark connected nodes (starting with marking an arbitrary one) if they have any edge to previously marked ones, then verify they all to be marked. Also sum of the weights is clearly polynomial ($O(nd)$ in which d is the number of bits to save weights of G), therefore SPANNING-TREE $\in$ NP.
Then we would show that SUBSET-SUM $\leq_P$ SPANNING-TREE by introducing a polynomial-time mapping reduction f that $f(\langle\{x_1, \cdots, x_n\}, t\rangle) = \langle G, l, u\rangle$ in which $\langle\{x_1, \cdots, x_n\}, t\rangle \in$ SUBSET-SUM if and only if $\langle G, l, u\rangle \in$ SPANNING-TREE.
The graph G we make, consists of n triangles connected to a central node. The weights in triangle i, are x_i, 0 and 0 which the edge with weight x_i is connected to the central node. An illustration of this G is as follows:

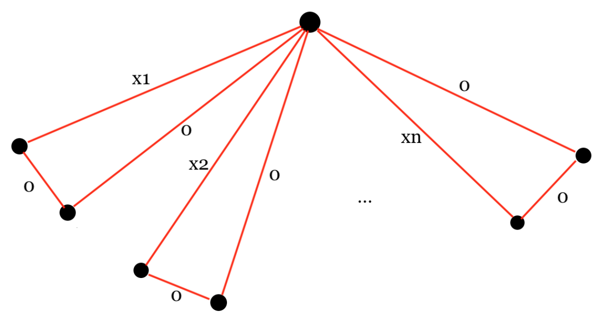

Figure 2: Graph G with respect to $\{x_1, \cdots, x_n\}$

Besides, we take both l and u to be equal to t (i.e. $l = t = u$). Every spanning tree in G, should contain two of the three edges in every triangle. Three different cases to do so, is depicted below in triangles 1, 2 and n:

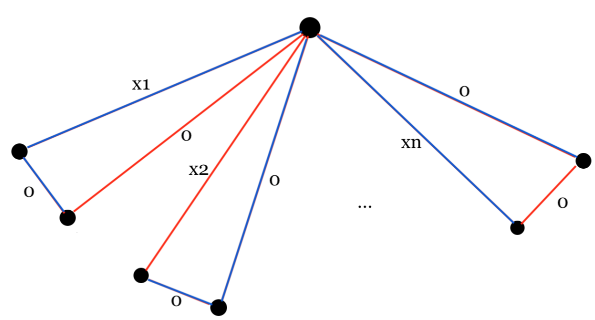

Figure 3: Cases to choose nodes. The edges in spanning tree are colored blue

In two of these cases, sum of the weights in the triangle is x_i and in one case, is 0. If there exists any set $A \subseteq \{1, 2, \cdots, n\}$ such that $\sum_{i \in A} x_i = t$, we make those $i \in A$ triangles to be in the first two cases (summed x_i) and for those with $i \notin A$, we make the triangle to be the second case (summed 0). Therefore this spanning tree would have the total weight t so $\langle G, l, u \rangle \in$

SPANNING-TREE. Also if there exists any spanning tree with total weight t, we consider its triangles. If the edge with the weight x_i is chosen in triangle i, we add i to A. Therefore we would have $\sum_{i \in A} x_i = t$ so $\langle \{x_1, \cdots, x_n\}, t\rangle \in$ SUBSET-SUM and our reduction is correct.
Besides, G has $2n+1$ nodes and $3n$ edges and weights saved in d bits (in which d is the number of bits needed to save x_is) which means $|\langle G\rangle|$ is polynomial with respect to $|\langle\{x_1, \cdots, x_n\}, t\rangle|$, so the reduction is polynomial. (Actually, the only important consideration is that we have not used the amount of x_is, which is $O(2^d)$ and we just have copied them in such d bits, as the weights information of G) ■

---

7 EXPs and Ps

Suppose NEXP $\neq$ EXP, so there exists some language $L \in$ NEXP which does not belong to EXP. By the definition of NEXP, $L \in$ NTIME(2^{n^k}) for some $k \in \mathbb{N}$.
Consider the N as the nondeterministic Turing machine which decides L in exponential time $O(2^{n^k})$. Then, we introduce the following language:

$$L_{pad} \coloneqq \{x\#^l \mid x \in L \,,\, l = 2^{n^k} - n \text{ in which } n = |x|\}$$

Then this following N' would decide L_{pad} in polynomial time:
"N' on the input w in the form of $x\#^l$:

- If w was not it that form, reject.
- Check whether $l = 2^{n^k} - n$ or not (in which $n = |x|$). If it wasn't, reject.
- If both tests got passed, simulate N on x and output the same as it does."

With simulating N on x, it is clear that $L(N') = L_{pad}$. But we conclude that N' decides L_{pad} in polynomial time with respect to $m \coloneqq |w|$. First two checking are done in $O(m)$ (only a counter with $O(\log m)$ bits and $O(m^2)$ movements of header would fulfill that). Besides, simulation of N on x needs $O(2^{n^k})$ time and as $m = l + n = 2^{n^k}$, the running time is $O(m)$ and polynomial with respect to $m = |w|$. So, N' decides L_{pad} in polynomial time therefore $L_{pad} \in$ NP.
Now, it is enough to show that $L_{pad} \notin$ P and our goal to prove that P$\neq$ NP

would be achieved. We do that by contradiction.
Suppose that $L_{pad} \in$ P; which means that there exists some deterministic Turing machine D which decides L_{pad} in polynomial time. The following deterministic decider D' would decide L in $O(2^{n^k})$:
"D' on input x:

- Add l's # to x to get $w = x\#^l$; in which $l = 2^{n^k} - n$ and $n = |x|$

- Simulate D on w and output as it does"

So $L(D') = L$ because $x \in L$ iff $x\#^l \in L_{pad} = L(D)$. For running time, adding l symbols needs $O(l) = O(2^{n^k})$ time steps, and simulating D on w is polynomial with respect to $|w| = 2^{n^k}$. Therefore D' decides L in exponential time ($O(2^{n^{k+1}})$ is enough) which means $L \in$ EXP, the contradictory result which implies that $L_{pad} \notin$ P therefore P $\neq$ NP. $\blacksquare$
