CS760 Topics in Computational Complexity **2024-25 Sem I**

Lecture 6: 20-08-2024

Scribe: Nandan Manjunath Immadisetty, Dheeraj Kurukunda *Lecturer: Rohit Gurjar*

1 Some Problems to Work on

- **Longest Path Problem**: Given two vertices a, b in a directed-graph find the longest path from a to b. This is an NP-hard problem and HAM $\leq$ longest path (Proof not given in class).
- (**HW**) HAMILTONIAN path problem is equivalent to HAMILTONIAN cycle problem. This means one problem reduces to other and vice-versa. (Hint: Just try to add additional vertices to prove this result in original graph)
- (**HW**) Directed longest path $\leq$ Undirected longest Path, i.e., undirected longest path problem is NP-hard.
- Given a directed graph and vertices a, b is there a path of even length from a to b. For undirected graph you can find such a path in linear time.
- **3-SAT $\leq$ IND-SET**

  Here 3-SAT is defined as the problem of determining whether a given Boolean formula, expressed in conjunctive normal form (CNF) where each clause contains exactly three literals, has an assignment of truth values to its variables that makes the entire formula evaluate to true.

  The IND-SET (Independent Set) problem is the problem of determining whether a given graph contains an independent set of a specified size. An independent set is a subset of vertices in the graph such that no two vertices in the subset are adjacent (i.e., there is no edge connecting any pair of vertices in the set).
- **IND-SET $\leq$ MAX-CUT** Here MAX-CUT is defined as the problem of finding a cut in a given graph that maximizes the number of edges between two disjoint subsets of vertices. A cut is defined as a partition of the vertices of the graph into two subsets, and the value of the cut is the number of edges that have one endpoint in each subset. (Proof was not completed in class).
- **INTEGER PROGRAMMING** Is an optimization problem where the objective is to maximize or minimize a linear function subject to a set of linear inequality constraints, where all variables must be integers. We can also use INTEGER PROGRAMMING to check are there any integer solutions which satisfy given linear inequalities.

  3-SAT $\leq$ INTEGER PROGRAMMING Here we are only considering the case where we don't have to maximize an objective function, i.e., the feasibility question. One can reduce optimization question to feasibility via binary search.
- **QUADRATIC PROGRAMMING**: Here instead of linear constraints, we will have degree 2 constraints and our objective is to check whether there exist real solutions (no need to be integers). This problem is NP-hard.

  3-SAT $\leq$ QUADRATIC PROGRAMMING PROGRAMMING

Below we describe some of the above reductions.

1.1 Proof for 3-SAT $\leq$ IND-SET

Given a CNF formula Φ, we will try to generate a graph G_Φ and corresponding number k_Φ such that if Φ is satisfiable then there exists an independent set of size k_Φ in G_Φ and vice-versa.

Take every clause in Φ and add vertices to graph corresponding to each literal in a clause and connect all these three vertices (3 because it is a 3-SAT) to make a triangle. Do this for every clause. Also across clauses if there are vertices such that one is negation of another then connect these vertices. We take k_Φ to be the number of clauses.

If there exists an INDSET of size k_Φ then there is an assignment to variables such that Φ is satisfiable and vice-versa.

Why does it work. If there is an IND-SET of size k_Φ (number of clauses) then each vertex should come from the triangle corresponding to each clause and only one vertex can come corresponding to each sub-graph as all vertices in this sub-graph are connected. If you assign literal corresponding to each selected vertex in IND-SET as true then we will get a satisfying assignment. Note that in the selected vertices no two literals corresponding to vertices can be negation of each other as these two vertices will be connected and won't be part of IND-SET.

To prove the converse, consider a satisfying assignment for Φ. This will set at least one literal true from every clause. Pick one such literal from every clause, and take the corresponding set of vertices in G_Φ. It is easy to verify that the obtained set is an independent set.

1.2 Proof for 3-SAT $\leq$ INTEGER PROGRAMMING

Let Φ be a CNF formula. Now to reduce it to INTEGER PROGRAMMING, we will create a set of linear inequalities corresponding to each clause. For each clause replace $\vee$ with $+$, replace $\neg x$ with $1-x$ and make inequality as the corresponding linear expression ≥ 1

Example:

$$(x_1 \vee x_2 \vee \neg x3) \wedge (\neg x_1 \vee x_2)$$

is converted to

$$x_1 + x_2 + (1 - x_3) \geq 1 \tag{1}$$
$$(1 - x_1) + x_2 \geq 1 \tag{2}$$
$$1 \geq x_i \geq 0 \tag{3}$$

Note that we need to find integer solutions.

It is easy to verify that Φ is satisfiable if and only if the constructed integer program has an integer solution.

1.3 Proof for 3-SAT $\leq$ QUADRATIC PROGRAMMING

It is similar to above case, but now we add a quadratic equation to ensure that a variable can take only values 0 or 1.

Example:

$$(x_1 \vee x_2 \vee \neg x3) \wedge (\neg x_1 \vee x_2)$$

is converted to

$$x_1 + x_2 + (1 - x_3) \geq 1 \tag{4}$$
$$(1 - x_1) + x_2 \geq 1 \tag{5}$$
$${x_i}^2 = x_i \text{ for } i = 1, 2, 3. \tag{6}$$

2 Understanding the Complexity Heirarchy:

The complexity class EXP (exponential time) is defined as follows.

$$\mathbf{EXP} = \bigcup_{c \geq 1} \text{DTime}\left(2^{n^c}\right)$$

There is another related class **E**, which is of interest.

$$\mathbf{E} = \bigcup_{c \geq 1} \mathrm{DTime}\left(2^{cn}\right)$$

It is proven that $\mathbf{P} \neq \mathbf{EXP}$, i.e., there are Problems (in terms of Turing machines etc.) that cannot be solved in polynomial time but is in **EXP**. But the questions of whether $\mathbf{P} = \mathbf{NP}$ or whether $\mathbf{NP} = \mathbf{EXP}$ is still unsolved.

2.1 Time Constructible Functions:

A function $f\mathbb{N} \to \mathbb{N}$ is said to be time constructible if $f(n)$ is computable in $O(f(n))$ time.

2.2 Time Hierarchy Theorem:

For two time Constructible functions $f(n)$ and $g(n)$, if $f(n)$ is significantly smaller than $g(n)$, i.e., if $f(n)\log(f(n)) = o(g(n))$, then

$$\mathrm{DTIME}(f(n)) \subsetneq \mathrm{DTIME}(g(n)).$$

That is, there are problems which can be solved in time $O(g(n))$ but not in time $O(f(n))$. Before beginning the proof we assume the following results:

- Every string in $\{0,1\}^*$ represents a TM.
- For any TM, there are infinitely many representations.
- There is a universal TM U with 6 tapes, alphabet as $\{0,1\}$, and a constant number of states such that for any input $\langle x, \alpha\rangle$, U will give the output $M_x(\alpha)$, where M_x is the Turing machine represented by x. If M_x halts in t steps on input α then $U(M_x, \alpha)$ halts in $O(t\log t)$ steps.

Proof of time hierarchy theorem
We start designing one such problem. Define function $D(x)$ as follows:

- Run the Universal TM on $\langle M_x, x\rangle$ for $g(|x|)$ steps
- If there is no Output (doesn't halt): Output 0
  Else if output is b : Output $1-b$ (i.e flip the output).

Clearly, this function could be computed in $O(g(n))$ time.

Now **assume** there exists a Turing Machine N which always gives the same output as $D(\cdot)$, but under $cf(n)$ time (c is some constant).

Claim: N will differ from $D(\cdot)$ on some input, and that input is one of the encodings of N.

Let y be one of the encodings of N. According to definition of function D, $D(y)$ is obtained by running the universal TM on $\langle N, y\rangle$ for $g(|y|)$ steps. Note that the universal TM can finish the simulation of N on y in time $c' \cdot c \cdot f(|y|)\log(c \cdot f(|y|))$. If $c' \cdot c \cdot f(|y|)\log(c \cdot f(|y|)) < g(|y|)$, then the universal TM on input $\langle N, y\rangle$ will halt, with output $N(y)$. In that case, by definition, $D(y)$ is the different from $N(y)$. Hence, y is a input where $D(\cdot)$ and N disagree.

One issue here is that the inequality $c' \cdot c \cdot f(|y|)\log(c \cdot f(|y|)) < g(|y|)$ holds only when $|y|$ is large enough. Since we assumed there are infinitely many representations of any TM, we can choose a large enough representation y of TM N.

Thus we proved that there exists a problem which can be computed in $O(g(n))$ time but not in $O(f(n))$ time. However, we do not know such separation for a natural looking problem.