ORIE 6300 Mathematical Programming I November 11, 2008

Lecture 20

Lecturer: Yogeshwer Sharma *Scribe: Yucel Altug*

1 Decision Problem as a Subset

Definition 1 *The set of all binary strings, is defined as* $\{0,1\}^* = \{0, 1, 00, 01, 10, 11, 000, \ldots\}$ *For any* $\pi \subseteq \Sigma^*$, $\bar{\pi}$ *denotes its* complement *in the set* Σ^*.

Next, let's give following examples of decision problems: Given a graph $G = (V, E)$,

$$ST = \{< G, s, t >: \text{There is a path from } s \text{ to } t \text{ in graph } G\}.$$

$\overline{ST}$, the complement of ST is given by:

$$\overline{ST} = \{< G, s, t >: \text{There is no path from } s \text{ to } t \text{ in graph } G\}.$$

Therefore given any $< G, s, t >$ finding whether $< G, s, t > \in ST$ or not is a decision problem.

As another example, consider Linear Programming (LP) problem:

$$LP = \{< \mathbf{A}, \mathbf{b} >: \exists \mathbf{x}, \text{ s.t. } \mathbf{Ax} \leq \mathbf{b}\}.$$

Note that for LP, Σ^* is the set of all possible $< \mathbf{A}, \mathbf{b} >$ and $\overline{LP}$ is given by

$$\overline{LP} = \{< \mathbf{A}, \mathbf{b} >: \forall \mathbf{x}, \mathbf{Ax} \not\leq \mathbf{b}\}.$$

Hence, given any $< \mathbf{A}, \mathbf{b} > \in \Sigma^*$ deciding whether $< \mathbf{A}, \mathbf{b} > \in LP$ or not is a decision problem.

Similarly, a linear optimization problem may also viewed as a decision problem. Consider following optimization problem:

$$\max \ \mathbf{c}^T\mathbf{x}, \text{ subject to } \mathbf{Ax} \leq \mathbf{b}.$$

If we define π as

$$\pi = \{< \mathbf{A}, \mathbf{b}, \mathbf{c}, t > . \text{ There is a solution } \mathbf{x} \text{ s.t. } \mathbf{Ax} \leq \mathbf{b} \text{ and } \mathbf{c}^T\mathbf{x} \geq t\},$$

and beginning from $t = -\infty$ decide whether $< \mathbf{A}, \mathbf{b}, \mathbf{c}, t > \in \pi$ or not, we can find optimal solution t^* as the point where the answer "switches" from YES to NO.

2 Definition of Polynomial Time

If we denote a computational problem as π, then the set of polynomial time problems, denoted by $\mathcal{P}$, is defined as:

$$\mathcal{P} = \{\pi : \text{There is an algorithm to decide } \pi \text{ in polynomial time}\}.$$

A necessary and sufficient condition for a problem to be an element of polynomial time is given by

$$[\pi \in \mathcal{P}] \Leftrightarrow [\exists A \text{ s.t. running time of } A \text{ on all inputs of length } n \text{ is } \leq n^d \text{ and } \pi = \{x \in \Sigma^* : A(x) = \text{YES}\}],$$

where d is a constant. The aforementioned algorithm A is called *polynomial time acceptor* for π.

Examples: ST problem defined in previous part.

$MWST = \{< G, W >: \exists \text{ a spanning tree such that weight is } \leq W\}$, where $MWST$ stands for "minimum cost spanning tree".

3 Definition of Non-deterministic Polynomial Time

The following is a necessary and sufficient condition for a computational problem π to be an element of $\mathcal{NP}$:

$$[\pi \in \mathcal{NP}] \Leftrightarrow [\exists B \text{ with running time } |x|^d \text{ and } \pi = \{x \in \Sigma^* : \exists c_x, \text{ s.t. } B(x, c_x) = \text{YES}\}],$$

where c_x is called *certificate* (proof, hint).
Example: $\pi = SAT$ (Satisfiability). Before defining SAT problem, we should state following

- Variables: $x_1, x_2, \ldots, x_n$.
- Literals: $l_1, l_2, \ldots, l_n$, where $\forall i$, $l_i \in \{x_i, \bar{x}_i\}$.
- Clauses: $c_1, c_2, \ldots, c_n$, where $\forall i$, $c_i = (l_{j_1} \vee \ldots \vee l_{j_n})$
- Formula: $\Phi = c_1 \wedge \ldots \wedge c_n$

Now we are ready to define SAT problem as follows:

$$SAT = \{\Phi : \text{ We can assign } 0, 1 \text{ to variables of } \Phi \text{ s.t. all clauses are satified}\},$$

where by satisfying we mean at least one one clause has a positive literal x_i assigned to 1, or a negative literal $\bar{x}_j$ assigned to 0.

Claim 1 $SAT \in \mathcal{NP}$

Proof: We construct an algorithm with following parameters:
Input: $< \Phi, n$ bit vector $v >$.
Procedure: Check whether v satisfies Φ or not.
Output: YES - NO.
A couple of observations next. First of all note that B is polynomial time. Indeed its running time is linear, since it simply scans thorough vector v and decides YES if it encounters a 1 and NO otherwise. Furthermore, if $\Phi \in SAT$, then $\exists v$, s.t. $B(\Phi, v) = \text{YES}$, where v is a satisfying assignment. Moreover, if $\Phi \notin SAT$, then $\forall v$, $B(\Phi, v) = \text{NO}$. Using all these there observations, we conclude that $SAT \in \mathcal{NP}$. □

Note that definition of $\mathcal{NP}$ is asymmetric, i.e.

$$\begin{aligned} \Phi \in SAT &\Rightarrow \exists c_\Phi, \text{ which leads to acceptence.} \\ \Phi \notin SAT &\Rightarrow \forall c_\Phi, B \text{ rejects.} \end{aligned}$$

As another example of $\mathcal{NP}$ problem, consider LP problem. Recall that we have

$$LP = \{< \mathbf{A}, \mathbf{b} >: \text{ system } \mathbf{Ax} \leq \mathbf{b} \text{ is feasible.}\}$$

Certificate for LP is a feasible solution $\mathbf{x}$. An algortihm B calculates $\mathbf{Ax}$ and decides whether $\mathbf{x}$ is feasible or not. Running time of B is quadratic in length of $\mathbf{x}$ (by recalling the complexity of matrix vector multiplication).

Another example of a problem in $\mathcal{NP}$ is the following:

$$COMPOSITE = \{n \in \mathbb{N} : \ n \text{ is composite}\}.$$

A certificate for $COMPOSITE$ problem is $n_1, n_2 \in \mathbb{N}$, s.t. $n_1, n_2 \notin \{1, n\}$ and $n_1 n_2 = n$.

4 $\mathcal{NP}$, CO–$\mathcal{NP}$

Definition 2

$$[\pi \subseteq \Sigma^*, \ s.t. \ \pi \in CO\text{–}\mathcal{NP}] \Leftrightarrow [\bar{\pi} \in \mathcal{NP}].$$

As an example of a computational problem which is in CO–$\mathcal{NP}$, consider $PRIME = \overline{COMPOSITE}$. Since $COMPOSITE \in \mathcal{NP}$ (as argued above), $PRIME \in CO$–$\mathcal{NP}$. Therefore, for any given input $n \in \mathbb{N}$, we have certificate that states that n is not a prime.

As another example of a problem in CO–$\mathcal{NP}$, consider $\overline{LP}$ defined as follows:

$$\overline{LP} = \{< \mathbf{A}, \mathbf{b} > : \ \mathbf{Ax} \leq \mathbf{b} \text{ is not feasible}\}.$$

If $< \mathbf{A}, \mathbf{b} > \notin \overline{LP}$, there exists a short certificate to verify $< \mathbf{A}, \mathbf{b} > \in LP$, which implies that $\overline{LP} \in CO$–$\mathcal{NP}$.

Claim 2 *$\overline{LP} \in \mathcal{NP} \cap CO$–$\mathcal{NP}$.*

Proof: Consider following systems

$$\mathbf{Ax} \leq \mathbf{b}, \tag{1}$$

and

$$\mathbf{A}^T\mathbf{y} = \mathbf{0}, \ \mathbf{y} \geq 0, \ \mathbf{b}^T\mathbf{y} < 0. \tag{2}$$

Using Farkas' lemma, we know that either (1) or (2) is feasible but not both. In the same way as above, we can write a polynomial time acceptor for (2), so we know that detecting the infeasibility of an LP is in $\mathcal{NP}$, or rather $\overline{LP} \in \mathcal{NP}$. Hence $LP \in CO$–$\mathcal{NP}$. Hence, we have $LP \in \mathcal{NP} \cap CO$–$\mathcal{NP}$. □

Before discussing $\mathcal{NP}$–completeness, we state two long standing open problems in computer science: First, is

$$\mathcal{P} = \mathcal{NP} \cap CO\text{–}\mathcal{NP}?$$

Second, is

$$\mathcal{P} = \mathcal{NP}?$$

It's clear that $\mathcal{P} \subseteq \mathcal{NP} \cap CO$–$\mathcal{NP}$ and that $\mathcal{P} \subseteq \mathcal{NP}$, but it's unclear whether equality holds.

5 $\mathcal{NP}$–Completeness

Intuitively, $\mathcal{NP}$–complete problems may be considered as 'the hardest' problems in $\mathcal{NP}$, in the sense that every problem in $\mathcal{NP}$ can be reduced to a $\mathcal{NP}$–complete problem in polynomial time. Before stating the definition of $\mathcal{NP}$–completeness, we need following definition.

Definition 3 *$\pi, \pi \subseteq \Sigma^*$, π' is reducible to π, denoted as $\pi' \leq \pi$, provided that*

$$\exists f \ : \ \Sigma^* \to \Sigma^*, \ \textit{s.t.} \ x \in \pi' \Leftrightarrow f(x) \in \pi \ \textit{and} \ f \ \textit{runs in polynomial time.}$$

Next, we state the rigorous definition of $\mathcal{NP}$–completeness:

Definition 4 *π is $\mathcal{NP}$–complete provided that:*

1. *$\pi \in \mathcal{NP}$.*
2. *$\forall \pi' \in \mathcal{NP}$, $\pi' \leq \pi$,*

Claim 3 *If π is $\mathcal{NP}$–complete and $\pi \in \mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$.*

Proof: Suppose we have an $\mathcal{NP}$–complete π, such that $\pi \in \mathcal{P}$, which implies that $\exists A_\pi$, which runs in n^d. Using this A_π, we can construct the following algorithm for any $\pi' \in \mathcal{NP}$:
Given input x for π',
- Compute $f(x)$
- Run A_π on $f(x)$
- If $A_\pi(f(x)) =$ YES, then output YES, if $A_\pi(f(x)) =$ NO, then output NO.

Now, observe that from the definition of $\mathcal{NP}$-completeness, the aforementioned algorithm works correctly; moreover, since $x \in \mathcal{P}$, we have n^d as the running time of A_π and n^c for f (recall the definition of reducibility). Hence, the algorithm's running time is n^{c+d}, which is also polynomial. Therefore, we conclude that $\pi' \in \mathcal{P}$. If $\pi \in \mathcal{P}$ for any $\mathcal{NP}$–complete π, then we have $\forall \pi' \in \mathcal{NP}$, $\pi' \in \mathcal{P}$. Hence the claim follows. □

6 Steps for $\mathcal{NP}$–Completeness Proofs

In this part, we will show that $SAT \leq 0-1\,IP$, where IP stands for 'integer program'. We should construct a polynomial time algorithm with following properties:
Input: Φ.
Output: A $0-1\ IP$.
For each x_i of SAT, we have $y_i \in \{0,1\}$ for $0-1$ IP , where $y_i = 0$ means x_i is false and $y_i = 1$ means x_i is true. For any clause $c_j = \bigvee_{i=P_j \subseteq [n]} x_i \vee \bigvee_{k \in N_j \subseteq [n]} \overline{x}_j$, we have $\sum_{i \in P_j} y_i + \sum_{k \in N_j} (1 - y_k)$ for $0-1$ IP . Observe that the aforementioned algorithm is polynomial time. Therefore, we conclude $SAT \leq 0-1$ IP, which implies that $0-1$ IP $\in \mathcal{NP}$.

Theorem 4 *(Cook, Levin '71) SAT is $\mathcal{NP}$–complete.*

Since $SAT \leq 0-1$ IP (as we have argued above), $0-1$ IP is also $\mathcal{NP}$–complete. Therefore, intuitively speaking it is also one of the 'hardest problems in $\mathcal{NP}$'.