CS277 Concrete Computational Complexity **Fall 1992**

Lecture 4: September 8

Lecturer: Sanjeev Arora *Scribe: Ramesh Subramonian*

4.1 Complexity of Selection

We continue with the proof of the lower bound on t-selection, and give the adversary strategy which shows the lower bound.

The **state** of the algorithm is defined as

1. A variable r; $r < t$
2. A set of r elements, $S = \{X_{(1)}, X_{(2)}, \ldots, X_{(r)}\}$. These are the items that the adversary has decided are the r smallest (in this order)
3. $\forall i, 1 \leq i \leq n, F(X_i)$, a potential for each element.

Invariant 4.1 *1.* $\sum_{i=1}^{i=n} F(X_i) = n$

2. $\forall i, 0 < F(X_i) < \frac{n}{t-1}$

3. if $F(X_i) > 0$ *then* X_i *has lost only to elements in* $\{X_{(1)}, X_{(2)}, \ldots, X_{(r)}\}$

4. X_i *has won* $\geq \lceil \lg F(X_i) \rceil$ *comparisons*

Claim 4.2 *Number of non-crucial comparisons* $\geq (t-1)\lceil \lg \frac{n}{t-1} \rceil$

Update Rule. This is how the adversary will update the potentials of the elements after each comparison. Consider a comparison between x and y.

if x or y or both are in S **then**
    answer accordingly
else
    Make the element with the larger score (say x) win the comparison
    if $F(x) + F(y) < \frac{n}{t-1}$ **then**
        $F(x) \leftarrow F(x) + F(y)$
        $F(y) \leftarrow 0$
    else

$S \leftarrow S \cup \{x\}$
$r \leftarrow r + 1$
endif
endif

Proof: [Of Claim 4.2] If x ever loses to y, then either x's potential goes to 0 or y enters S. The potential of a winner can at most double in each comparison. So, an element, x, with non-zero potential must have won $\geq \lceil \lg F(x) \rceil$ comparisons. By Invariant 4.1, $\forall x : F(x) < \frac{n}{t-1}$ When an element enters S, its potential must be at least $\frac{1}{2}(\frac{n}{t-1})$. Hence, it must have won $\lg\lceil \frac{n}{t-1} \rceil + 1$ comparisons. (The +1 is for the last comparison that put it into S.) Of these comparisons, at most 1 was crucial. Hence, the number of non-crucial comparisons is $\geq (t-1)\lg\lceil \frac{n}{t-1} \rceil$. ∎

4.2 Complexity of Undirected Connectivity

Suppose an undirected graph G is represented by an adjacency matrix $(a_{i,j})_{i,j=1..n}$, where $a_{i,j}$ is 1 if there is an edge (i,j) and 0 otherwise. We consider the complexity of an algorithm A that will answer the question: "Is G connected ? " (Our model is that each entry of $(a_{i,j})$ is covered with a flap that the algorithm has to lift in order to know that entry. The algorithm A has to be deterministic; however it is allowed to lift the flaps adaptively). The algorithm will be charged 1 for each entry of $(a_{i,j})$ that it reads.

Determining the cost that A must pay in the worst case is slightly non-trivial because A can terminate as soon as it has discovered a spanning tree, for which it would be charged only $n-1$. We want to determine the complexity of undirected connectivity in the usual way as the complexity of the best algorithm on a worst case input.

$$D(connectedness) = \min_{A} \max_{G} cost(A, G).$$

(Note: $D(connectedness) \leq \binom{n}{2}$).

Claim 4.3 $D(Connectedness = \binom{n}{2}$

We give an adversarial strategy which will force the algorithm to query all the edges of the graph. At the end of step i, A has asked i questions. The information state of the algorithm is given by

1. H_i = hidden edges. (edges of a that have not been queried)
2. P_i = those edges to which we have answered "yes" (they exist)

3. A_i = those edges to which we have answered "no" (they do not exist)

Invariant 4.4 *The adversary will ensure that for $i < \binom{n}{2}$, $H_i \cup P_i$ is connected and P_i is an unconnected forest. (Hence, the algorithm cannot claim to know the answer.)*

Update Rule. The adversary's strategy to maintain the invariant is to answer "no" when asked about an edge, except when doing so would disconnect the graph.

Proof: [Of Claim 4.3] Let $j < \binom{n}{2}$ be the first step in which the invariant is violated. The adversary *had* to answer "yes" when queried about the jth edge, e, or risk disconnecting $P_{j-1} \cup H_{j-1}$. Since $j < \binom{n}{2}$, H_j is not empty, and contains another edge, say e'. Any such edge e' must be able to put loops in P_j (we assert that P_j has no loops, being a minimal connected graph). This contradicts the assumption that $H_{j-1} \cup P_{j-1}$ becomes disconnected on deleting e.

■

4.2.1 Certificates

Consider the following problem. Somebody holds a graph G which she knows to be connected. What is the *least* number of edges that she needs to reveal in order to convince you that G is connected ? A necessary and sufficient condition for G to be connected is that it has a spanning tree. Hence, a spanning tree is a *certificate that G is connected* or, a *1-certificate* for short.

If G were unconnected, how many edges would she have to show you to convince you of this fact ? (Rather, how many edges would she have to show you that *did not* exist to convince you.) A *cut* of a graph $G = (V, E)$ is a partition of the nodes of the graph into 2 disjoint subsets S and $V - S$. So, showing us that none of the edges of the cut exist would constitute a certificate that G *is not connected*, or a *0-certificate* for short. How big must this certificate be? If $|S| = x$, the number of edges across the cut is $n(n - x)$, which attains a maximum of $\frac{n^2}{4}$ at $x = n/2$. Hence, the size of a *0-certificate* is $\leq \frac{n^2}{4}$ (that some graphs may need a certificate of this size is seen in the example where the graph consists of two separate connected components of size $n/2$ each).

Note that a spanning tree crosses every cut and so a graph cannot have both a 0-certificate and a 1-certificate.

4.3 Boolean Decision Trees

Now we study the ideas of the previous section in a general setting. A string property is any function $F_N : \{0,1\}^N \to \{0,1\}$. The input is a string $s \in \{0,1\}^N$. The output is $F(s)$. Consider a deterministic algorithm A to determine $F(s)$. Let $cost(A,s)$ be the number of bits that A reads on s (again, our model is that each bit is covered with a flap and the algorithm raises flaps adaptively). The *deterministic decision tree complexity* of F is

$$D(F) = min_A\ max_s\ \{cost(A,s)\}.$$

(Note: $D(F) \leq N$.) Why the name "decision tree complexity?" The reason is that one can unravel the actions of the algorithm to get a decision tree in which each interior node is labelled with a variable and the algorithm branches depending on the value of this variable. Each leaf is labelled with a 0/1 answer. Every input follows a path from the root to a leaf, whose label is the algorithm's answer for $F(s)$. The cost of the algorithm, $cost(A,s)$, is the depth of the leaf reached by s.

4.3.1 Certificate Complexity

Lets consider the certificate complexity. Let a "variable" of s be an element of the string s. A 1-certificate of F is an assignment of to some subset of variables $s' \subseteq s$ that will force $F(s) = 1$. Similarly, a 0-certificate is an assignment of to some $s' \subseteq s$ that will force $F(s) = 0$. A 1-certificate should be thought of as a substring of s which is sufficient to convince you that $F(s) = 1$. It follows that a string cannot have both a 1-certificate and a 0-certificate (in particular, a 0-certificate and a 1-certificate must assign some variable in opposite ways). Note that certificate complexity is defined purely in terms of the function F, without taking recourse to any algorithm.

Definition 4.5 *The certificate complexity of F on s, $C_s(F)$, is the size of the smallest certificate which fits s. The certificate complexity of F is denoted by $C(F)$ and $C(F) = max_s\ \{C_s(F)\}$.*

Motivation. The original motivation for studying certificate complexity came from the fact that it can be though of as a kind of non-deterministic complexity in the decision tree model (a non-deterministic algorithm is one that is allowed to make lucky "guesses"). For this reason one often sees the name *nondeterministic decision tree complexity* used instead of certificate complexity.

Example 1 Consider connectedness. $D(connectedness) = \binom{n}{2}$. $C(Connectedness) = \frac{n^2}{4}$.

Example 2 Consider a function $F : \{0,1\}^{2N} \to \{0,1\}$. $F(s) = 1$ is s has exactly N 1's and 0 otherwise. It is easily seen that $D(F) = 2N$ and $C(F) = 2N$.

Theorem 4.6 $C(F) \leq D(F) \leq C(F)^2$

Proof: First we show that $C(F) \leq D(F)$. Let s be the input such that $C_s(F) = D(F)$. Consider the path taken by s in any decision tree for F. We claim that the variables seen along that path constitute a certificate for s. For if not, then the unseen variables could be assigned to in ways that makes the answer wrong. This proves that $C(F) \leq D(F)$.

Now we show that $D(F) \leq C(F)^2$. Let $C(F) = k$. For every string s, pick a smallest certificate for s and call it π_s Define $S_0 = \{\pi_s | F(s) = 0\}$ and $S_1 = \{\pi_s | F(s) = 1\}$. Notice that for any string w $F(w) = 1$ iff some $\pi \in S_1$ fits w.

We now describe a deterministic algorithm A that examines k^2 bits in w to determine whether or not there is a certificate in S_1 that fits w. A, will proceed in k phases and make at most k queries per phase. At the end of the ith phase, it will keep around only those certificates which are consistent with the bits of w seen so far.

Define

$$S_0^i = \{\pi_s |\ \pi_s \in S_0 \text{ and is consistent with bits seen till the } i\text{th phase}\}$$

Define S_1^i similarly.

Since it is pointless to carry around that portion of a certificate which has already been matched to w, the algorithm focuses only on the unmatched portion of the certificates in S_0^i and S_1^i.

In other words, the algorithm will PRUNE the sets as follows after reading variable x_j in phase $i + 1$:

`If` $x_j = 1$ `then delete all certificates from` S_0^i `and` S_1^i `in which` $x_j = 0$`. Keep around those certificates in which` $x_j = 1$`, but with their lengths reduced by 1.`

The algorithm is the following:

`In phase` i+1`, pick any certificate from` S_0^i `and query all the variables in it. PRUNE, and go to next phase.`

We will show that the length of the certificates in S_1^i is at most $k - i$. Therefore, after k phases, either all the certificates in S_1 have been found to not fit w (i.e. $f(w) = 0$), or one of them has been matched completely (so $f(w) = 1$). This will prove $D(F) \leq k^2$.

To see why our claim is true recall our old observation that every $\pi \in S_0^i$ and every $\sigma \in S_1^i$ must assign some variable in opposite ways (otherwise nothing prevents them from occuring in the same string). Therefore a certificate in S_0^i must contain *at least* 1 variable in common with every certificate in S_1^i. This proves our claim that every phase reduces the size of each certificate in S_1^i by *at least* 1 (if not altogether killing it).

(A subtle point to be noted here is the following. We also have to show that the common

variable in question was not queried in an earlier phase. This is true because π and σ assign it in opposite ways, and so both the certificates could not have survived the pruning.) ■