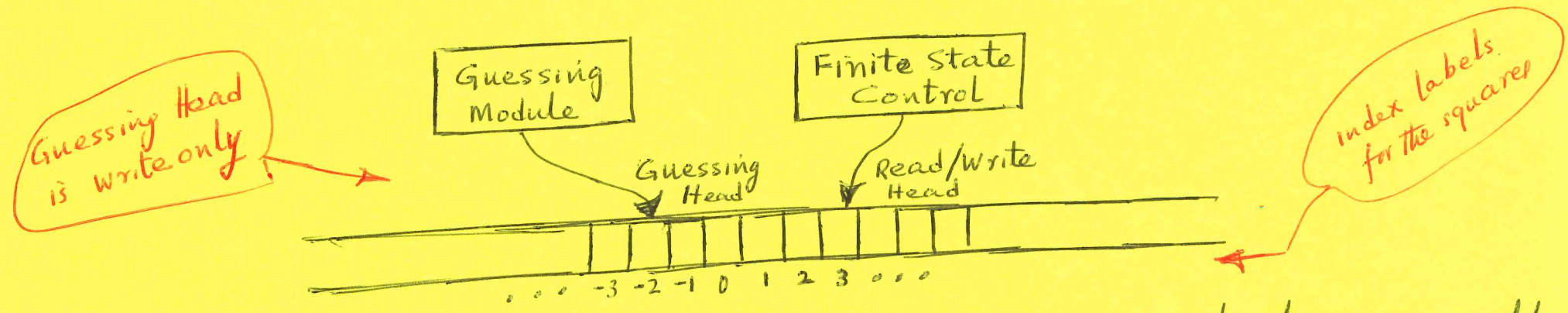
Lecture 26
ECE 664
Avi Kak
Purdue Univ.

A More Formal Definition of Class NP

- In Lecture 25, NP was defined as the class of all decision problems that can be solved by polynomial time nondeterministic algorithms.
- We will now use the notion of a nondeterministic Turing Machine (NDTM) to present a more formal definition for class NP.

The Structure of an NDTM:



The operation of an NDTM is specified by a set of quintuples in exactly the same manner as the operation of a DTM. However, there is a fundamental difference between the two with regard to how computations are carried out on an input string x.

- To solve a problem instance I by an NDTM, we first encode the instance into a string $x \in \Sigma^*$ over some alphabet Σ.
- We place the string x in squares 1 through $|x|$, and we place the symbol b (for blank) in all other squares. Note that $b \in \Gamma - \Sigma$ where Γ is the tape alphabet.
- Next we place the read/write head of the Finite State Control over the square indexed 1.
- Finally, before the actual computation can begin, we place the write-only guessing head on the square indexed -1.
- With the setup work as described above completed, the NDTM can start computing.
- The computation by NDTM takes place in <u>two stages</u>:

STAGE 1 (Guessing Stage): (i) Place the Finite-State Control in inactive mode.

(ii) Let the Guessing Module direct its write only head, one step at a time, either to write some symbol from Γ, or to move one square to the left, or to halt. (The choice of what symbol to write and the decision as to whether the guess construction should come to an end are completely random.)

(iii) If the Guessing Module directs its write-only head to a stop, the Guessing Module becomes inactive and, at the same time, the Finite State Control is activated in state q_0.

STAGE 2 (Checking Stage): (i) From this point on, the Checking Stage carries out its computations exactly as a DTM would.

(ii) The computation by the Checking Stage is said to be an accepting computation if it halts in state q_y.

(iii) All other computations, halting or not, are considered to be nonaccepting computations.

- A more precise definition of what constitutes an accepting computation by a given NDTM for input string x :

> Since an NDTM program will have an infinite number of possible computations for a given input x, one for each possible guessed string from Γ^*, we say that the NDTM program M accepts x if at least one of these is an accepting computation in the sense of the checking stage halting in state q_y.

- The language recognized by an NDTM program M is defined as :

$$\boxed{L_M = \{ x \in \Sigma^* \mid M \text{ accepts } x \}}$$

- Definition of the time required for accepting a given string x :

> The time required by an NDTM to accept a given string $x \in L_M$ is defined to be the minimum, over all accepting computations of M for x, of the number of steps occurring in the guessing and the checking stages up until the halt state q_y is reached.

- The Time Complexity Function $T_M : Z^+ \to Z^+$ for an NDTM program M is given by

$$T_M(n) = \max \{ \text{accepting times required for all strings of length } n \}$$

- M is a polynomial time NDTM program if there exists a polynomial p such that

$$T_M(n) \leq p(n) \qquad \text{for all } n \geq 1$$

- We can now define the class NP more formally as follows :

$$\boxed{NP = \left\{ L \;\middle|\; \begin{array}{l}\text{There exists a polynomial time} \\ \text{NDTM program } M \text{ for which } L_M = L\end{array} \right\}}$$

- A decision problem Π is said to belong to class NP under encoding scheme e if

$$L(\Pi, e) \in NP$$

An Important Theorem : The theorem that is shown below is important because it establishes the fact that <u>every</u> NP problem can be solved with a deterministic algorithm but in exponential time.

<u>THEOREM</u> : If $\Pi \in NP$, then there exists a polynomial p such that Π can be solved by a deterministic algorithm having time complexity $O(2^{p(n)})$.

<u>PROOF</u> : Let the Time Complexity Function of the NDTM program that solves Π be given by the polynomial $q(n)$. Since the scanhead of the Finite State Control can move <u>at most</u> one square in one time step, it must be the case that the guess put down by the guessing module for the acceptable accepting computation is no longer than $q(n)$ for any given n. But if the guesses are limited in length to $q(n)$ squares, that places a bound on the total number of different possible guesses that can exist : $k^{q(n)}$ where $k = |\Gamma|$ is the size of the tape alphabet. A deterministic algorithm can check through all these guesses in time $q(n) \times k^{q(n)}$. This is easily shown to be bounded by $O(2^{p(n)})$ for some $p(n)$.

An NDTM Example for Solving the TS Decision Problem:

INSTANCE: $C = \{1, 2, 3\}$ $d(1,2) = d(1,3) = d(2,3) = 100$, and $B = 10$

QUESTION: Does there exist a tour of the three cities of length B or less?

(This is obviously a no instance of TS. That is, it belongs to the set $D_{TS} - Y_{TS}$. See Lecture 25 for the notation D_{Π} and Y_{Π})

- To create an NDTM solution, we first need to specify an input alphabet Σ that can be used by an encoding scheme e to translate the instance into a representation that would be appropriate for the tape of the NDTM. We choose

$$\Sigma = \{0, 1\}$$

which implies that the encoder will use a binary code representation for all the numbers in the instance shown above. Next, we need to specify the tape alphabet Γ. Let's set

$$\Gamma = \Sigma \cup \{M, N, P\}$$

As a marker symbol, M will be used to delimit the binary representations of the city ID's, N to delimit the binary code words for the inter-city distances, and P for doing the same for the bound B.

- NDTM Operation: We start by laying down the TS instance on the tape, starting at the square indexed 0 and writing to the right:

Each square is a "short-hand" for the multiple squares needed for the binary code words for the numbers shown

								M	1	M	2	M	3	M	N	100	N	100	N	100	N	P	10	P							
								-3	-2	-1	0	1	2	3	4	5	→														

index values for the squares

Next, the guessing module kicks in. It lays down a guess for the solution (which, in our case, is a tour) on the tape starting at the square indexed -1 and writing to the left. Let the guess be the randomly generated sequence of cities (3, 1, 2). So the guessing module writes writes these numbers down as shown below:

	M	2	M	1	M	3	M	M	1	M	2	M	3	M	N	100	N	100	N	100	N	P	10	P							
	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5																		

where again each number inside a square stands for the multiple squares needed by the binary code for that number. Finally, the Finite-State control kicks in to verify whether the sum of the consecutive distances in the guessed tour is <u>no</u> greater than 10. For the guess shown the bound B=10 is not satisfied by the sum of the three distances. Let's now place this computation within the framework

of the nondeterministic computer presented in Lecture 24. We will use the NDTM described above in each thread of the nondeterministic computer. Obviously, the guess in each thread will be made independently.

- In order to answer the question stated at the top of the previous page, we now keep in mind the following:
  - For a no instance (which is the case in our example), there must not exist any guesses that yield a yes answer for the question.
  - For a yes instance, there must exist at least one guess for which the answer to the decision question is yes.
- For a problem to belong to class NP, the time taken by the NDTM to construct a guess and to verify it, for a yes instance of the problem, must be bounded by a polynomial in the size of the problem.
- But what about the no instances?
- Obviously, for the example on the previous page, the computations for even the no instances would come to a halt in polynomial time. However, in general, that may not be the case if our guesses are completely random.
- An important issue: The example on the previous page shows a very "structured" guess. Can the guessing module write on the tape a completely random stream of symbols from T as a guess? Yes, that is exactly what the guessing module is supposed to do. But note that the verification stage will implicitly impose a structure on a random guess. For example, if the verification stage cannot recognize the city ID's in a purely random guess the NDTM will abort by, say, entering into an infinite loop.