ITCS:CCT09 : Computational Complexity Theory Mar 2, 2009

Lecture 1

Lecturer: Jayalal Sarma M.N. *Scribe: Jayalal Sarma M.N.*

1 Introduction to the course

This course aims to provide a (slightly advanced) exposure to the theory of computational complexity. Computational complexity theory is the study of efficient computation and its fundamental limitations. The theory helps us to state questions about this in a precise way and provide qualified answers to them. To state some examples, how much resources is required to solve concrete computational problems? Does the use of randomness or parallelism make computations significantly faster? If not, can we prove the same with respect to some explicit examples of problems.

However, it turns out that the hardest thing to prove about the models is that they cannot solve a given problem within particular resource constraints. Such "lower bounds" are currently obtainable only when the model is very specialized or the constraints are very severe. Inherent limitations of the currently known techniques is also something that is currently explored on the research frontier of complexity theory.

The course will give a quick introduction to the classical results and techniques in complexity theory and thereafter cover (selected) newer results, techniques and areas. The choice of topics is based on the view that the course should provide a platform for the current topics of research in the area of complexity theory. We might possibly be skipping some of the basic material in a standard complexity theory course. But we will mention and give pointers to extra reading whenever necessary. In general, we hope to touch upon most of the topics below roughly divided into 30 lectures in total. We may choose to deviate from this plan as the course progresses.

- Classical structural complexity theory : Notion of a resource. Time and space. Hierarchy theorems. Oracle Turing Machines, relativisation. Reductions and Completeness. Alternation. Ladner's Theorem. Polynomial Time Hierarchy. Karp-Lipton type Collapse results. Counting Complexity. Permanent problem. Toda's Theorem (5 lectures).
- Nonuniform models of computation. Boolean circuits. Uniformity. Circuit value problem, connection to standard Turing model. The world *inside* P. Branching Programs, Word problems. Algebraic Characterisation of circuit classes NC^1 and ACC^0 (Barrington and Barrington-Therein). (4 lectures).

- Bounds in circuits : Asymptotic circuit size bounds. Shannon, Lypanov Theorems. Elimination Method and the Polynomial method. The Switching Lemma. Parity lower bound for constant-depth circuits. Proof that AC^0 with mod p gates can't compute mod q. The "fusion method" for proving circuit lower bounds. Application of the "fusion method" to prove a lower bound on monotone circuit size required to compute 3-clique. (4 lectures).

- Randomized Complexity theory : Use of randomness. Classes BPP and RP. Deranodmization goals. Error reduction using expanders. Explicit constructions of expanders. Extractors, Hitting set generators, PRGs. Nisan-Wigderson generator. Conditional derandomization results. Average case complexity: Levin's theory. Hardness amplification. (7 Lectures)

- Interactive Proofs (outline): $\mathsf{IP} = \mathsf{PSPACE}$, the PCP-Theorem(statement). Graph Products and Reingold's USTCON algorithm. Dinur's proof of the PCP Theorem. (3 Lectures).

- Recap of counting. Arithmetic circuits. Straight line programs. Polynomial identity testing. Permanent vs Determinant Problem. Impagliazzo-Kabanets theorem. Algebraic Complexity Theory (basics and directions). (4 lectures)

- Limitations of current techniques : relativization (recap), Natural proofs, Algebrization. Geometric Complexity Theory (intro, if we get to it). (3 lectures).

2 Basic Models of Computation

The first lecture is rather introductory and the notes are being written for the sake of completeness. The material below are (better) explained in most of the standard textbooks. In addition, we will fix up some notation and concepts which we will use throughout the course.

Turing machines are simple, yet powerful model of computation and almost all reasonable, general-purpose models are known to be equivalent to the Turing machines; in the sense that they define the same set of computational functions. We will first talk about deterministic Turing machines. Perhaps the best way is to view it as a generalisation of finite automata which you have seen in the previous courses.

A deterministic Turing machine consists of three basic units: a finite control, a working tape, a read-only input tape, a write-only output tape. The finite control is can be in finite number of states. All the tapes extend to infinity on both ends, are divided into cells and contained one taperead-head each. Each tape cell can store one of the finite tape-symbols or alphabets. The control unit determines the next state of the Turing machine by looking at the current state, and the symbol read on each of the tapes.

More formally, a Turing machine M is defined by the following information.

- A finite set Q of states.
- An initial state $q_0 \in Q$.
- A subset $F \subset Q$ of accepting states.
- A finite set Σ of input symbols.
- A finite set Γ containing Σ of tape symbols, including a special blank symbol B.
- A partial transition function $\delta : (Q - F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$.

A configuration of a Turing machine is a record of all the information of the computation of M at a specific moment, which includes the current state, the current symbols on the tapes. Formally it is an element (q, x, y) of $Q \times \Gamma^* \times \Gamma^*$ such that the leftmost symbol of x and the rightmost symbol y are not B. This denotes that the current state is q and the current nonblank symbols on the tape are the string xy and the tape head is scanning the leftmost symbol of y. If the TM has more than one tape the same encoding could be repeated to include each of them.

Fixing a way to index into the tape to represent the read-head for each tape, defines a set of configurations of the TM. Now the notion of *transition* in the finite control can be generalised to the set of configurations to define a *next configuration* function. We will not do this formally. Computation of the TM M is the sequence of configurations $\alpha_0, \alpha_1, \ldots, \alpha_n$ such that α_0 is the initial configuration, and $\alpha_i \to_1 \alpha_{i+1}$ for $1 \leq i \leq n-1$. TM on input x accepts if M halts on x and the halting state is in F.

It is also important to note that there set of all turing machines are countable because of the standard encoding one can assign to each Turing machine. We call this enumeration as the *standard enumeration* of the Turing machines. We will end this section of the review from the previous courses by the the well known Church-Turing thesis states: *A function computable in any reasonable computational model is also computable by a Turing machine.*

2.1 Notion of a Complexity Measure

To do complexity theory, which as we said is the study of the resources required to perform various computations, we need the notion of a resourse more formalised. To start with examples, the classic notions of resources (or the measure of complexity) arising in the study of algorithms are time and space. However, it is clear that one needs to ask a more fundamental question. What is resource, and when is a quantity associated with the TM computation considered to be a resource.

For now, the resource is a function which depends on the size of the representation of the input rather than the input itself.

It is clear that the measure should be a function of the input itself and the Turing machine that we are looking at. Fixing a the standard enumeration of the Turing machine, measure

should be a 2-place function from $\mathbb{N}^2 \to \mathbb{N}$. Intuitively, the function must have some basic properties too : (1) something that a machine can test. Under the Church-Turing hypothesis this would mean that there is a Turing machine which given the description of the Turing machine can test whether the resource considered is indeed the give value. In addition the resource function should be allowed to be a partial function which needs to take a value whenever the machine halts on the particular input.

Manuel Blum, back in 1967, formalised the above intuition into two axioms known as the *Blum axioms* to define a resource formally.

Definition 1 (Blum Axioms) *Let $\phi(i,x)$ be a function $\mathbb{N}^2 \to \mathbb{N}$. We say that ϕ is a* **complexity measure** *if it satisfies the following two axioms.*

- *$\phi(i,x)$ is defined if and only the i^{th} Turing machine Φ_i halts on input x.*
- *Given i, x and y, the relation $\phi(i,x) \leq y$ should be computable where $\phi(i,x) \leq y$ is defined to be false if $\phi(i,x)$ is undefined.*

Another thought process will lead us to the question of what should be the measure a function of : should it be the input itself or should it just depend on the size of the representation of the input. The former choice is driven by the fact that that given two inputs of the same length the Turing machine might be able to solve one better(in terms of the use of the resources) than the other. However, the latter choice is simpler since it makes the function significantly easier to analyse and could be thought of as the worst case resource bound for each input length. One could look at the average complexity for each input length and we shall return to this issue later in the course. For now we will fix the latter choice for the resource measure.

Definition 2 (Time & Space) *Let n denote the size of the (representation of) the input x denoted by $|x|$. Time measure t is defined to be*

$$t(i,n) = \max_{x:|x|=n} \{ \text{ the least } \ell \text{ such that } \Phi_i(x) \text{ halts in at most } \ell \text{ transitions.} \}$$

Similarly, we define space measure s to be:

$$s(i,n) = \max_{x:|x|=n} \{ \text{ the least } \ell \text{ such that } \Phi_i(x) \text{ uses at most } \ell \text{ worktape cells.} \}$$

For simplicity, we will be considering functions that does not depend on i and hence we will denote call the resources $t(n)$ and $s(n)$ respectively.

It is an exercise to verify that the above resources indeed satisfies the Blum axioms stated above. As an example of a measure which is not a complexity measure:

$v(i,x) =$ the least ℓ such that $\Phi_i(x)$ visits its accepting configuration at most ℓ times.

We claim that $v(i,x)$ does not satisfy Blum axioms.

Definition 3 $\mathsf{DTIME}(t)$ *is the class of all languages L that are accepted by determinstic Turing machines M running in time $t(n)$. For a class of functions $\mathcal{F}$:*

$$\mathsf{DTIME}(\mathcal{F}) = \bigcup_{t \in \mathcal{F}} \mathsf{DTIME}(t)$$

$\mathsf{DSPACE}(s)$ *and* $\mathsf{DSPACE}(\mathcal{F})$ *are defined analgously.*

3 Elementary Theorems about TMs

3.1 Tape reduction

Theorem 4 *For any k-tape Turing machine M which runs in time $t(s)$ and space $s(n)$, there is a 1-tape Turing machine computing the same function as M in time $O(t(n)^2)$ and space $O(s(n))$. In fact, there is a 2-tape Turing machine computing the same function as M in time $O(t(n)\log(t(n)))$ and space $O(s(n))$*

Proof Let M be the k-tape Turing machine and let the tape alphabet be Γ. We will describe a new machine M' which simulates M using only one tape in time $(t(n)^2)$. Fix some arbitrary reference for numbering the cells in the two way infinite tapes of M. The idea is to view the one tape cell as k "sub-cells" and each subcell having two parts. That is, i^{th} subcell (of the j^{th} cell of the tape of M') has two parts; one storing the content of the j^{th} cell of the i^{th} tape, and the other storing a bit (say the head-bit) indicating whether the tape head of the i^{th} tape is currently positioned at j^{th} cell.

Simulation of M is done in a straightforward way. M' has only one tape-head (because it has only one tape!). To simulate one step of M it needs to get the information about all the k-tapes of M which are stored in one tape of M' as per the above encoding. Once it scans and has all the information it makes the transition of M and writes up the new tape contents of M' according to that of M via the above encoding. It is easy to see that this simulation is indeed correct.

To analyse the time overhead for M'. The overhead comes from the fact that while simulating each step of M, to get the information about heads of M, M' will have to linearly scan over all the tape cells to searching for the 1s stored in the "head-bits". These bits could be atmost $t(n)$ away from each other. Thus in the worst case M' needs to spend atmost $t(n)$ time for each step of M. Thus the bound $O(t(n)^2)$.

It is easy to notice that the space used by M' is within a factor of $2k$ from that of M. The above proof can be modified a little to prove the stronger claim. We will skip that to the excercises. ■

3.2 Space Bounded TMs and Halting

Theorem 5 *Let M be a Turing machine with k-tapes, working over an alphabet of size σ, with c states in the control unit, then :*
If M runs for more than $c\sigma^{k\ell}\ell^k$ time without touching more than ℓ tape cells, then M does not halt on input x. In this case, we say $s(|x|) \leq \ell$.

Proof As we noticed, the configuration of the Turing machine is completely specified by: (1) contents of used part of the tapes (2) state of the finite control (3) position of the tape heads. The next-move of the TM is completely determined by these. Hence, if the TM reaches the same configuration twice, it has to follow exactly the same computation path which it followed when it visited the configuration the previous time.

We give the proof by contradiction. Let us imagine that M does halt on input x in time $t > c\sigma^{k\ell}\ell^k$ steps but does not touch more than ℓ tape-cells. Since M has touched only ℓ tape cells, the number of possible configurations that the machine can be in is at most $c\sigma^{k\ell}\ell^k$. Since $t > c\sigma^{k\ell}\ell^k$ many computational steps, it has to visit the same configuration c again. But that is a contradiction since computation which took M from c to itself configuration will just repeat itself, and hence M runs into an infinite loop. ■

4 Basic Theorems on Space & Time

We saw previously that a k-tape Turing machines can be simulated by a 1-tape Turing machine with a loss of constant factor overhead in the space used by the TM. Now we will see that this is not really an overhead and that set of functions computable in space $s(n)$ is indeed same as those computable in space $O(s(n)$, thus showing a robustness of the model.

Theorem 6 (Tape Compression Theorem) *For any constant $c > 0$,* $\mathsf{DSPACE}(s(n)) = \mathsf{DSPACE}(c.s(n))$.

Proof The idea is again encoding the tape contents. Intuitively, saving on worktape space should be easier. This time we will again divide the cells of M into groups of size α. Let M be the machine be having q states, k tapes, and runs in space at most $s(n)$ with tape alphabet to be Γ. The machine M' works over and alphabet Γ^α and has $q\alpha^k$. M' works over a bigger alphabet and hence interpret each cell into a block of the tape cell in M's tape. Each state has information about state of M and for each tape, the position of the tape heads of M within each block to which the head of M' currently points to.

M' simulates M in the obvious way, namely that to simulate a move of M all that M' needs to know is the state and the contents of M' tape. But now since the alphabet size if bigger, M' head is coarser and points only to blocks and it needs positions of the tapeheads of M within the blocks that its coarser tapehead is poiting to). It is exactly this information

that is stored in the state. Thus it saves on the number of tape cells used by a facter of c although at the expense of increasing the alphabet size and the number of states of the finite control. This proves the theorem. ■

We will see a similar bound for the time as the resource using a technique which is similar, but not the same. Of course, in the case of time, sublinear time bounded TMs cant even look at the whole input, and hence we will restrict our attention to time bounds which is at least linear.

Theorem 7 (Linear Speed-up Theorem) *If $t(n) \geq n$, then for any constant $c > 0$,* $\mathsf{DTIME}(t(n)) = \mathsf{DTIME}(c.t(n))$.

Proof Let us say $c < 1$. Thus trivially $\mathsf{DTIME}(c.t(n)) \subseteq \mathsf{DTIME}(t(n))$. In that case, reverse inclusion is trivial. Let M be a TM running in time $t(n)$ with Γ as the tape alphabet. The idea is to come up with an M' which uses again encoding of tape contents to save on time. Divide the contents of the tape into blocks of size α each (imagine α to be large value to be chosen later). Each block could be thought of as the elements of a larger alphabet Γ^α.

For a block b, its left neighbour b_ℓ and the right neighbour b_r, denote by $\textsc{History}(b_\ell, b, b_r)$ as the history of the computation of M when it enters the block b and until it leaves b_ℓ or b_r. Notice that the size of History depends only on α (in fact it is at least α). But it represents a small part of computation of M. The idea then is intuitive. Encode History into fewer(say β, independent of α) transitions of M' by encoding them into the states of M'. Now choosing, $\alpha > \beta/c$ will show that for large enough n the time taken to be bounded by $c.t(n)$. We will skip the formal details of this calculation. ■