Theory of computation **Fall 2015**

Lecture 21: Computational complexity based on TMs

Faculty: K.R. Chowdhary *: Professor of CS*

21.1 Computational Complexity based on TMs

The problems which can be solved in principle by TMs, there exists problems that require one to run a modern supercomputer for the life time of earth or even more. This fact leads us to classify the real-world problems into two classes: 1) *tractable* problems which computed in feasible amount of time, and 2) *intractable* which cannot be computed in feasible amount of time. For this we make use of TM dues to two reasons: 1) time required to run a typical computer is related to run an equivalent TM , and 2) the time required to solve a problem for real computers are technology dependent.

The tractable problems take time proportional to polynomial of the size of input to the machine. The other type of the problems, i.e., intractable problems, require the time for their solution, which is proportional to exponential of the size of the input. The interesting fact about the later is, given the solution, its correctness can be verified in polynomial of the input size. For example, it is hard to find the factors of given any large integer number, but given the factors, it can be easily verified by multiplying them together, whether it results to the original number.

Some examples of Complexity of computation are given in the following:

Adding two n-digit numbers: Usually its takes $n+1$ steps to multiply two n-digit numbers. But if we look at minor steps then $5n+1$ steps are required as follows: n additions of digits, n additions of carry, n comparisons if sum of two digits is greater than 10, n steps to print lower digit, and n steps to save carry. The last step is for carry save from last sum. When further smaller steps are considered, it comes out to be $an+b$ steps, where a, b are constants, not dependent on n. Thus time complexity for adding two numbers is $\theta(n)$.

Multiplication: To multiply x and y, one approach is add x to 0, y times. If both numbers are n digit long, then total number of steps are $\theta(n.10^n)$. Yet Other method exists for multiplication, as per this the number of steps are: $\theta(n^2)$ complexity. The best known algorithm for multiplication is $\theta(n^{1.1})$.

Factoring: Factoring of n digit number is some times not well defined, when a number has different set of factors. For example, $1001 = 77 \times 13$, and also 91×11. To factor an integer Z, we need to divide it by range 2 to $Z-1$. If $|Z| = n$, complexity is 10^n. However, no solution like, $\theta(n)$ or $\theta(n^c)$ exists, where c is constant.

Following are some Complexity terms:

$T(n)$: It is *Time complexity* of standard Turing Machine. The function $T(n)$ is called *time-constructible* if there exists a *time-bounded* Deterministic Turing Machine (DTM) that with input $|w| = n$ makes n moves. The same term $T(n)$ is also used to refer nondeterministic Turing machine's Time complexity.

$S(n)$: It is used to refer to *Space complexity* of standard Turing Machine. Function $S(n)$ is called space-

constructible, if there exists a *space-bounded* standard TM, that for each input of length n, requires exactly $S(n)$ space.

$DTIME(T(n))$: It is class of languages that have deterministic time complexity of $O(T(n))$.

$NTIME(T(n))$: It is class of languages that have nondeterministic time complexity of $O(T(n))$.

Definition 21.1 *(Time complexity of a TM) The time complexity of a TM takes the form $T(n) : \mathbb{N} \mapsto \mathbb{N}$, where $\mathbb{N}$ is set of integers and $T(n)$ is number of steps in the of TM, and n is length of input. For complexity purpose, we take maximum value of this n, which is worst-case analysis.*

Example 21.2 *Consider the problem of recognition of language $L = \{a^m b^m \mid m \geq 0\}$, which we had solved earlier using standard TM. For $|w| = |a^m b^m| = n$. In each to and fro journey the R/W head marks one a and one b, hence total number of journeys are $n/2$. Total number of transition required for this language are:*

$$(\frac{n}{2} + \frac{n}{2}) + (\frac{n}{2} + \frac{n}{2}) + \ldots, n/2 \text{ times}$$
$$= \frac{n^2}{2}$$

At the end, when all a's and b's are marked, it makes $n/2$ transitions to check all b's are marked, and makes n transitions to reach to begin of tape. Thus, total number of transitions are: $n^2/2 + n/2 + n$, which gives polynomial time complexity of $O(n^2)$.

In the following examples we Consider three problems with variable time complexities.

1. *Reachability* problem: Given a directed graph $G = (V, E)$, determine whether there exists a path from node s to node t in the graph.
2. *Eulerian path problem*: To determine whether exists a closed path that passes through all the edges of an undirected graph $G = (V, E)$, going through each edge exactly once only.
3. *Hamiltonian path problem:* To determine whether there exists a closed path passing through all the nodes of a directed graph $G = (V, E)$, with each node once only.

Out of these, the 1st and 2nd can be solved in polynomial time. But, yet it is not known whether any one can solve the 3rd in polynomial time, as the solutions available so far are exponential.

The question now is whether there exists any real life problem that can be transformed into these graph related problems. One example is, the membership problem for *context-free languages* can be interpreted to decide whether whether or not there exists a sequence of rules such that applying these successively from the start symbol ultimately leads to generation of string $|w|$, which has been input to a TM. If TM can generate it, then prints Yes else No.

The halting problem can be thought of as deciding whether or not there exists a permissible sequence of configurations from start configuration up to the configuration with the halting state. The difficulty or number of steps we need for these problems is, whether we can use brute force method or some other.

21.2 Verifier

We introduce a verifier that can decide whether or not a given object satisfies a given condition. Let this verifier is V, which is a DTM with two tapes. Given a graph $\langle G \rangle$ on tape 1 together with potential Hamiltonain path u on tape 2, the verifier checks in polynomial time, if u is Hamiltonian path of $\langle G \rangle$. On the other hand, since verifier is just a DTM with polynomial time, the output of verifier, initially given on tape 1 as $\langle G \rangle$ and tape 2 as u is,

$$V(\langle G \rangle, u) = \begin{cases} Yes, \text{ if } V \text{ goes to accept state} \\ No, \text{ if } V \text{ goes to reject state} \end{cases} \tag{21.1}$$

The string u on tape 2 is called *certificate*. From above we say that if potential path u on tape 2 is a polynomial time computation, then we accept the graph $\langle G \rangle$ as a graph having hamiltonian path. Hence,

$$HAMPATH = \{\langle G \rangle \mid \text{ there exists a } u \text{ such that } V(\langle G \rangle, u) = Yes\}.$$

Verifying existence, given a candidate path is much easier than determining existence without having given the candidate. For example, checking fitness of given candidate for a certain job is much easier than searching if a fit candidate exists at all. The verifier can check it in polynomial time.

Since, the verifier exists to check if the given solution (Hamiltonian path) is a solution, in polynomial time, it appears that all problems can be checked in a similar way. Let us consider the problem:

$$\overline{HAMPATH} = \{\langle G \rangle \mid V(\langle G \rangle, u) = \text{ No, for any certificate } u\},$$

which is complement of the set of the Hamiltonian graph. Verifying that a graph does not have Hamiltonain path seems to be more difficult than verifying that a graph has Hamiltonain path. Although it is yet not known whether it can be verified, that a graph does not have Hamiltonain path, in polynomial time, but there is yet no short certificate for this, i.e., of finding it in polynomial time.

Definition 21.3 Verifier. *A verifier V is a deterministic two-tape TM. The verifier V verifies the language L if following conditions are satisfied:*

1. *For any $w \in L$ there exists $u \in \Sigma^*$ so that $V(w, u) = Yes$.*
2. *For any $w \notin L$, there is $V(w, u) = No$ for any $u \in \Sigma^*$, where w is on tape 1 and u is on tape 2.*

The language that is verified by V is denoted by $L(V)$, such that,

$$L(V) = \{w \mid \text{ there exists } u \in \Sigma^*, \text{ and } V(w, u) = Yes\}.$$

The class of languages verified by the verifier V in polynomail time are denoted by NP. □

21.3 Classes P

A language L is decidable in polynomial time if there is standard TM M that accepts L with time complexity $tc_M \in O(n^r)$, where $|w| = n$ is length of input string, and r is an integer constant not dependent on n. The family of languages decidable in polynomial time is denoted by **P**.

A language accepted by multi-tape TM in time $O(n^r)$ is accepted by a standard TM in time $O(n^{2r})$, which is also polynomial. This invariant property of TM shows the robustness of TM.

Some definitions related to class **P** are as follows.

Definition 21.4 **P** *is class of membership problems for the languages set expressed as,*

$$\bigcup_{P(n)} DTIME(P(n)); \tag{21.2}$$

where $P(n)$ is polynomial in n.

Definition 21.5 *(Acceptance of palindromes) Output is YES if $w \in \Sigma^*$ is palindrome, else NO. The Complexity Class is* **P**.

Definition 21.6 *(Path problem in directed graphs) Input is $G = (V, E)$. Output is YES if there is a path from v_i to v_j in the graph, else NO. Complexity class is* **P**, *as the complexity is $O(n^2)$ due to Dijkstra's algorithm.*

Definition 21.7 *(Deriviability in CNF) Input is $CNF(G, w)$, output is Yes, if $S \Rightarrow^* w$ else No. Complexity is* **P**.

The class **P** is defined in terms of time complexity of implementation of an algorithm on standard TM. Even if we choose multi-track or multi-tape machine as computational model on which algorithms are implemented, the complexity **P** is invariant. The time complexity of multi-track and standard TM are same. Also, we have seen that change from standard TM to multi-track and vice-versa, the category **P** is invariant.

Also, it has been found that the transition on a standard TM increases only polynomially with the number of instructions executed in a computer. The robustness of the class **P** under changes of machines and architectures provide support for its selection as defining the border between tractable and intractable problems.

21.4 The Class NP

Definition 21.8 *A language L is in nondeterministic polynomial time (***NP***) if there is a NDTM M that accepts L in polynomial time $tc_M \in O(n^r)$, where input $|w| = n$, and r is a natural number independent of n. The family of languages accepted in nondeterministic polynomial time is called* **NP**.

Note that NDTM guesses the alternatives.

The family **NP** is subset of *recursive languages*. This is because the polynomial bound (for NDTM) on the number of transitions is guarantee that all computations of M eventually terminate. Since, every deterministic machine is also a nondeterministic machine, but not vice-versa (yet!), hence $\mathbf{P} \subseteq \mathbf{NP}$.

Polynomial solution for the **NP** problems are not known to exist. In $NDTM$ the solution is selected nondeterministically rather than systematically examining all the possibilities. Also, the **NP** is defined as the class of membership problems for languages in,

$$\bigcup_{P(n)} NTIME(P(n)). \tag{21.3}$$

Following are the examples are of **NP** problems:

1. *SATISFIABILITY* problem: Input to TM M is Boolean expression u in CNF (Conjunctive Normal Form), and output is Yes if there is an assignment that satisfies u otherwise the output is No. The time complexity in **P** is unknown, but in **NP** is confirmed as Yes.
2. *Hamiltonian path problem:* Input to TM M is a directed graph $G = (V, E)$ and output is Yes if there is a single cycle that visits all nodes, and No other wise. Time complexity **P** is unknown, **NP** is Yes. Thus, Hamiltonian path problem is in **NP**, but its solution can be verified in time **P**.
3. *Subset sum problem:* The Input is set S, number k, and output Yes if there is $P \subseteq S$, and sum of each subset is k, else No. Complexity **P** is unknown, but **NP** is yes.

Primality test and Compositeness: The languages *PRIMES* and *COMPOSITS* are formally defined as:

$$PRIMES = \{x \mid x \text{ is prime}\}, \tag{21.4}$$

and,

$$COMPOSITS = \{y \mid y \text{ is Composite number}\}. \tag{21.5}$$

Hence, $PRIMES = \overline{COMPOSITS}$. Therefore, if $COMPOSITS$ is **NP** then $PRIMES$ is $Co - \mathbf{NP}$ (Complement of NP). Compositness can be determined by NDTM through guessing nondterministically. COMPOSITNESS is in NP but its solution can be verified in P time.

Fermat's Little theorem is tool for primality test. It states that, if p is prime and a is an integer then $a^p \equiv a(mod\ p)$, i.e., $a^p - a$ is evenly divisible by p. This problem is in **NP** because of the exponential component a^p.

Example 21.9 *The expression* $2^{11} - 2$ *is divisible by* 11. *Sets of primes are in* **NP** *but not in NP-complete. Similar is case with the COMPOSITS. The language of PRIMES is* $NP \cap Co - NP$, *and hence of COMPOSITS also. Because, if that is not the case then* $NP = Co - NP$.

Theorem 21.10 *COMPOSITS are NP.*

Proof: Input to a NDTM M *is* p, *and* $|p| = n$. *Guess a factor* f *of at most* n *bits* ($f \neq 1, f \neq p$). *This part is non-deterministic. The time taken by any sequence of choice is* $O(n)$.

Divide p *by* f *and check if remainder is* 0, *accept if yes. This part is deterministic of complexity* $O(n^2)$ *on a 2-tape TM.* □

Definition 21.11 *(NP-Complete) A language* B *is* NP*-complete if it satisfies two-conditions: (1)* $B \in NP$, *(2) Every* $A \in NP$ *is polynomial time reducible to* B, *i.e.,* $B \in NP \wedge \forall A : A \in NP \wedge A \leq_P B \Rightarrow B \in$ *NP-Complete.*

Definition 21.12 *A language Q is **NP-hard** if every $L \in NP$ is polynomially reducible to Q, that is, $\forall L : L \in NP \wedge L \leq_P Q \Rightarrow Q \in NP - hard$.*

The NP-hard problem that is also NP is called NP-complete. *Co-NP* is complement of NP, therefore, *Co-NP* is set of all the complements of all the NP problems.

One can consider an NP-complete language as a Universal language in the class NP.

Definition 21.13 *If there is a polynomial time algorithm for one NP-problem, then all NP problems are solvable in P time, are called NP-complete.*

This is because, if A is NP-complete, then all NP-problems are reducible to it. And, if $A \in P$, then all those NP are P.

The benefit of the problem **N**-complete is that, if one can be solved, then all rest are automatically solved, hence, one may choose only one of the most appropriate **NP** problem for solution.

Satisfiability is NP-complete. A Boolean expression $\phi = \{\bar{x} \wedge y) \vee (x \wedge \bar{z})$ is satisfiable for $x = 0$, $y = 1$, $z = 0$, as it evaluates ϕ to 1 (TRUE). SAT is languages of all satisfiable formulas, hence, $SAT = \{\langle\phi\rangle | \phi\rangle$ is satisfiable Boolean formula$\}$. Cook-Levin theorem links the complexity of SAT problem to complexities of all problems in **NP**.

Cook-Levin Theorem: Following the proof idea of Cook-Levin.

Theorem 21.14 *SAT is NP-Complete.*

Proof Idea: It is easy to show that SAT *is NP, the hard part is to show that any language in NP is polynomially reducible to* SAT. *Therefore, we construct a polynomial time reduction for every $A \in NP$ to* SAT.

Reduction for a language A takes input w and produces Boolean formula ϕ that simulates the NP machine for A on input w.

If machine accepts, ϕ has a satisfying assignment, that corresponds to accepting computation, otherwise NO. Therefore, $w \in A$ iff ϕ is satisfiable.

Following are NP-Complete problems: 3-SAT, Hamiltonian path problem, and subset construction problem.

Complexity classes-Time: Following are the complexity classes for time complexity.

Time constraint	**Class**	**Machine**
$f(n)$	$DTIME(f(n))$	DTM
$poly(n)$	**P**	DTM
$f(n)$	$NTIME(f(n))$	$NDTM$
$poly(n)$	**NP**	$NDTM$
$2^{poly(n)}$	$EXPTIME$	DTM

The class of membership problems for the problem *EXPTIME* is languages,

$$\bigcup_{P(n)} DTIME(2^{P(n)}). \tag{21.6}$$

Following is a theorem based on NP-completeness.

class	**machine**	**Space constraint**
DSPACE(f(n))	DTM	f(n)
L	DTM	$O(log_n)$
PSPACE	DTM	poly(n)
EXPSPACE	DTM	$2^{poly(n)}$
NSPACE(f(n))	NDTM	f(n)
NL	NDTM	poly(n)
NEXPSPACE	NDTM	$2^{poly(n)}$

Theorem 21.15 *If $B \in$ NP-Complete and $B \leq_P C$ for $C \in NP$, then $C \in$ NP-Complete.*

Proof: We must show that every $A \in NP$ is polynomially reducible to C. Because B is NP-Complete, therefore, every $A \in NP$ is polynomially reducible to B (as per property of NP-Completeness). And B in turn is polynomially reducible to C (given). Because the property of polynomial is closed under the composition, we conclude that every $A \in NP$ is polynomially reducible to C. Therefore C is NP-Complete. □

Space - Complexity classes: $S(n)$: The function $S(n)$ is called space constructible if there exists an $S(n)$ space-binded Deterministic TM that for each input $|w| = n$ requires exactly $S(n)$ space. Therefore, $S(n) =$ Space complexity of a Deterministic Turing Machine.

$DSPACE(S(n))$: It is class of languages that have deterministic space complexity of $O(S(n))$.

$PSPACE$: The class of membership problems for the languages decidable in polynomial space on deterministic TM,

$$PSPACE = \bigcup_k DSPACE(n^k) \tag{21.7}$$

$$DSPACE(f(n)) = \{L | L \text{ is decidable by O(f(n)) space on DTM}\}. \tag{21.8}$$

$$NSPACE(f(n)) = \{L | L \text{ is decidable by O(f(n)) space on NDTM}\} \tag{21.9}$$

Savitch's Theorem: If a $NDTM$ uses $f(n)$ space, it can be converted into a DTM that uses $f^2(n)$ space.

As per Savitch's theorem: $PSPACE = NSPACE$, $EXPSPACE = NEXPSPACE$.

For NDTM, if $f(n)$ is maximum number of tape-cells scan in any branch of computation, then its complexity if $f(n)$.

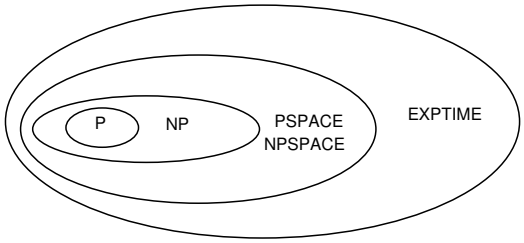
SAT which is $NP - Complete$ in time, is linear space. (because is reusable).

$PSPACE = NSPACE$, $P \subseteq PSPACE$. $NP \subseteq NSPACE$, therefore, $NP \subseteq PSPACE$.

Therefore, $P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$.

21.4.1 Is P = NP?

A language accepted by in polynomial time by a deterministic multi-track or multi-tape TM is in **P**. Construction of a standard TM equivalent to these preserves polynomial complexity.



A construction of an equivalent DTM for a NDTM does not preserve polynomial time complexity.

The NDTM solves the Hamiltonian path problem in time P (by guessing) but the DTM takes exponential time.

The question of $\mathbf{P} = \mathbf{NP}$ is whether constructing a solution is more difficult than checking it, to see if a single possibility satisfies the condition of the problem.

The $\mathbf{P} = \mathbf{NP}$ is, requirement of precisely formulated mathematical problem, and can be resolved only when two classes are equal or **P** is proper subset of **NP** is proved.

One approach to determining $\mathbf{P} = \mathbf{NP}$ is to explain the properties of each language on individual basis, for one can be Hamiltonian problem, ..., what is required is a universal solution of all **NP** problems, in **P** time.

If a polynomial-time is discovered that accepts an NP-complete language, can be used to construct machines to accept every language in NP in deterministic polynomial time. and, if that takes place, then P = NP.

Theorem 21.16 *If B is NP-Complete and $B \in P$, then $P = NP$.*

Proof: If B is NP-Complete then every problem in **NP** *is polynomially reducible to B. Since $B \in$* **P**, *therefore, every* **NP** *problem is polynomially reducible to to B, which is* **P**. *Hence, every* **NP** *is* **P**, *i.e.* $\mathbf{P} = \mathbf{NP}$. □

Once we get NP-Complete, other **NP** problems can be reduced to it. However, establishing first NP-Complete problem is difficult.