Theory of Computation: Time Complexity classes

Complexity theory

- Classify problems according to the computational resources required to solve them. In this course:
  Running time – time complexity
  Storage space – space complexity
- Attempt to answer: what is computationally feasible with limited resources?
- Complexity class: A set of functions that can be computed within given resource bounds – could be time bounds/space bounds.
- Contrast with decidability – what is computable? But more importantly we care about the resources.
- Classification of Decision problems – we will briefly look at why this is a good enough motivation to study complexity theory. Search problems can be related to decision problems in a natural way.

Central Questions

- Is finding a solution as easy as recognizing one?
  P = NP?
- If I have a non-deterministic TM using polynomially many cells to solve a problem, can I design a deterministic TM using polynomially many cells to solve the problem?
  PSPACE = NPSPACE
- If I have a non-deterministic TM using $f(n)$ many cells to solve a problem, can I design a non-deterministic TM using $f(n)$ many cells to solve the complement problem?
  Eg: NL = co-NL
- Many more central problems – computer scientists have an idea of what the answer should be but they do not have a proof yet.
- What are the consequences if one of the problems gets solved? What happens if the answer is opposite to the widely held beliefs?

Time complexity

- DTIME: Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A language L is in $DTIME(T(n))$ if and only if there is a deterministic TM that for some constant $c > 0$, given an n-length input runs in time $c \cdot T(n)$ and decides L.
- Recall that a deterministic TM on any given input can proceed in exactly one way.
- The class P: $\bigcup_{c \geq 1} DTIME(n^c)$. Thus the set of all problems that can be solved in polynomial time.

The class P

Problems in P, Decision versions of:

- Search algorithms
- Sorting
- BFS tree
- DFS tree
- Graph connectivity
- Binary search

Nondeterminism

So far we considered decision problems, where algorithms were designed in deterministic Turing Machines and where the total running time was polynomial.
What happens if we introduce non-determinism?

Nondeterminism

- Recall a simple problem: You are given a TM with an infinite tape (infinite in both directions, but this is still equivalent to a normal TM), where the tape head starts at a particular cell with a special symbol $\vdash$ (not used anywhere else on the tape), and there is at most one cell that contains a 1 - all other cells contain blanks. The aim is to design an algorithm to find if there is a cell with a 1.
- Let $\mathcal{C}$ be a cell with 1. Non-deterministic machine – will guess in which direction $\mathcal{C}$ with 1 is and take time linear in the distance between that cell and $\vdash$.
  Deterministic machine – will have to check in both directions. If the distance between $\mathcal{C}$ and $\vdash$ is n, then the deterministic machine could take as much as $O(n^2)$ time.
- If no such $\mathcal{C}$ exists – nondeterministic machine will guess that no such cell exists.

The class NP

This class contains problems that can be verified in polynomial time – as opposed to solved (class P).
If there is some form of evidence (certificate) given along with the input itself, then a deterministic TM can be designed to verify if the certificate indeed shows that the given instance is an instance of the problem.

The class NP: Formal definition

A language $L \in 0, 1^*$ is in NP if:

- there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and
  A polynomial time deterministic Turing Machine M
- such that for every $x \in 0, 1^*$,
  $x \in L \iff \exists u \in \{0,1\}^{p(|x|)}$ s.t $M(x, u) = 1$.
- $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfies $M(x, u) = 1$: u is called a certificate of x.

Example

- Independent set: In a graph, an independent set I is a set of vertices such that no pair in I have an edge between them.
- Independent Set Problem: Given a graph G and an integer k, is there an independent set of size st least k in G?
- Certificate: An independent set of size at least k in G!
- Given a vertex set of the graph G, it is easy to verify whether it is an independent set of size at least k. Hence this problem is in NP.

NTIME

For every function $T : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$, L is said to be in $NTIME(T(n))$ if there is a constant $c > 0$ and a NDTM M that runs in $c \cdot T(n)$-time such that for every $x \in \{0,1\}^*$, $x \in L \iff M(x) = 1$.

NP and NTIME

Theorem: $NP = \bigcup_{c \in \mathbb{N}} NTIME(n^c)$.

NP and NTIME contd.

Proof:

- The main idea is that the sequence of nondeterministic choices made by an accepting computation of an NDTM can be thought to be a certificate that the input is in the language and vice versa.
- $\bigcup_{c \in \mathbb{N}} NTIME(n^c) \subseteq NP$: Suppose p is a polynomial and L is decided by NDTM N with running time $p(n)$.
- For an $x \in L$ consider the sequence of nondeterministic choices in order to reach accept state t. The sequence has length $p(|x|)$ - can be thought of as a certificate.
- Given the certificate let M a deterministic TM that simulates N using the certificate - M is the verifier.

NP and NTIME contd.

Proof:

- $NP \subseteq \bigcup_{c \in \mathbb{N}} NTIME(n^c)$: Suppose p is a polynomial and $L \in NP$ with certificate sizes bounded by p and deterministic TM M as verifier.
- For an $x \in L$ design a NDTM N that runs in polynomial time in the following way.
- N nondeterministically guesses the certificate for x - the certificate is of polynomial length, so polynomial guesses.
- It then runs the polynomial time verifier M to verify using the guessed certificate if $x \in L$.

Class EXP

$EXP = \bigcup_{c>1} DTIME(2^{n^c})$.
$NEXP = \bigcup_{c>1} NTIME(2^{n^c})$ - problems that can be verified in exponential time. It is to EXP what NP is to P.

$P \subseteq NP \subseteq EXP$

- $P \subseteq NP$: If L is in P then there is a Turing Machine N solving it. This can be thought of as a Turing machine with the empty string as a certificate.
- $NP \subseteq EXP$: Suppose $L \in NP$ has certificates and defined by the polynomial function $p()$ and is verified by a machine M. Then for an input x, design a TM that enumerates all certificates in $\{0,1\}^{p(|x|)}$ and then for each such certificate runs the TM M.
  There can be $2^{O(p(n))}$ certificates for an n-length input string.

EXP and NEXP

If $EXP \neq NEXP$, then $P \neq NP$

- If $P \neq NP$ then $EXP = NEXP$ could still hold!
- We assume that $P = NP$ and show that then $EXP = NEXP$
- Suppose $L \in NTIME(2^{n^c})$ recognized by a machine M. For each string $z \in L$ create $z1^{2^{|z|^c}}$: new language L_{pad}.
- If z is an input of L, in a new TM N, we take an input $z1^{2^{|z|^c}}$ of L_{pad}. This makes the length of the input string approximately $2^{|z|^c}$.
- N first tries to see if the input is of the form $z1^{2^{|z|^c}}$ and extracts z. It runs M on z.
- So, $L_{\text{pad}} \in NP \implies L_{\text{pad}} \in P \implies L \in EXP$.

$P \neq NP$

- It is widely believed that $P \neq NP$: there is a difference in complexity of solving in polynomial time and verifying in polynomial time.
- Some techniques have been developed to determine which problems are "hard" in NP: NP-completeness. It is believed that if NP-complete problems are in P then $NP = P$.
- Many intermediate classes have also been discovered under the assumption of $P \neq NP$: Ladner's theorem constructs a language that is neither in P nor is NP-complete.

The class coNP

$coNP = \{L | \overline{L} \in NP\}$. A language $\overline{L} \in 0, 1^*$ is in coNP if:

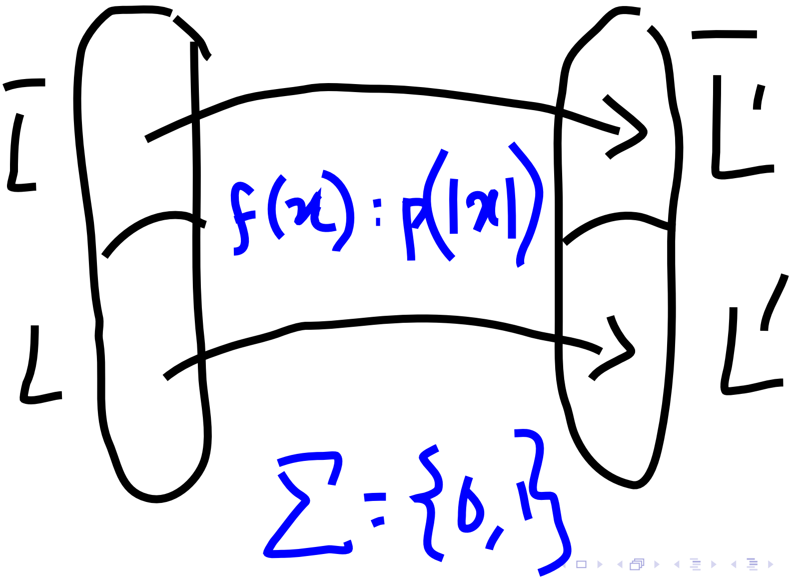
- there exists a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$ and
  A polynomial time deterministic Turing Machine
- such that for every $x \in 0, 1^*$,
  $x \in \overline{L} \iff \forall u \in \{0, 1\}^{p(|x|)}$ s.t $M(x, u) = 1$.

$P \in NP \cap coNP$! If $P = NP$ then $NP = coNP$!

Hard problems in NP: NP-Completeness

A language $L \subseteq \{0,1\}^*$ is polynomial-time Karp reducible to a language $L' \subseteq \{0,1\}^*$, denoted by $L \leq_p L'$, if there is a polynomial-time computable function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $x \in L$ if and only if $f(x) \in L'$.
We say that L' is NP-hard if $L \leq_p L'$ for every $L \in NP$. L' is NP-complete if L' os NP-hard and $L' \in NP$.

Karp Reductions



Properties of reductions

Theorem:

1. (Transitivity) If $L \leq_p L'$ and $L' \leq_p L''$, then $L \leq_p L''$.
2. If language L is NP-hard and $L \in P$, then $P = NP$.
3. If language L is NP-complete, then $L \in P$ if and only if $P = NP$.

Properties of reductions

Proof:

- If function p is $O(n^c)$ and q is $O(n^d)$. Composition function $p \cdot q$ is $O(n^{cd})$ - also a polynomial.
- 1. If f_1 is a polynomial-time reduction from L to L' and f_2 is a polynomial-time reduction from L' to L'', then $f_2 \cdot f_1$ is a polynomial-time reduction from L to L'' such that $x \in L \iff f_1(x) \in L' \iff f_2(f_1(x)) \in L''$.

Properties of reductions

Proof:

- 2. If L is NP-hard, then for all $L' \in NP$ there is a polynomial time reduction to L. Now, if $L \in P$ then there is a deterministic TM M running in $cp(n)$-time, for some polynomial p, such that $L = L(M)$. Build a deterministic polynomial time machine $M_{L'}$ such that $L' = L(M_{L'})$: On input x, first $M_{L'}$ reduces it to an instance x' of L. Then it runs M on x' and outputs the answer of M.
  So $NP \subseteq P \subseteq NP \implies P = NP$
- 3. If L is NP-complete and $L \in P$ then by its NP-hardness and the previous point, $P = NP$.
  If L is NP-complete and $P = NP$, then since $L \in NP$, $L \in P$.

NP-complete problems

- Are there NP-complete problems?
- NP-complete problems are problems in NP that are at least as hard as any other problem in NP.
- As seen above, if we can give a polynomial time algorithm for an NP-complete problem, then we will solve $P = NP$ (and win 1 million dollars!)

Boolean Formulas

- Boolean formula ϕ: over a set of variables $\{u_1, u_2, \ldots, u_n\}$, logical operators $\wedge, \vee, \neg$.
- Eg: $(u_1 \wedge u_2) \vee (u_3 \wedge u_4)$.
- The set $\{u_1, \neg u_1, u_2, \neg u_2, \ldots, u_n, \neg u_n\}$ are called literals for the Boolean formula.
- Take $z = (z_1, z_2, \ldots, z_n) \in \{0, 1\}^n$. The string z is called an assignment of ϕ: $\phi(z)$ denotes the evaluation of ϕ under the assignment $u_1 = z_1, u_2 = z_2, \ldots, u_n = z_n$.
  If $u_i = z_i$ then $\neg u_i = 1 - z_i$ (also denoted as $\overline{u_i}$).
- If $\phi(z) = 1$ then z is a satisfying assignment of ϕ.
- The size of a formula ϕ, denoted by $|\phi|$, is the length of the formula (in terms of the literals).

CNF formulas and SAT

- CNF formula: Boolean formula of the form $\wedge_i(\vee_j v_{i_j})$, where v_{i_j} is either a variable u_k or its negation $\overline{u_k}$.
- Clause: $(\vee_j v_{i_j})$.
- SAT: Determine if there is a satisfying assignment for an input CNF formula ϕ.
  A satisfying assignment has to satisfy each clause: so at least one literal in each clause.

SAT is NP-complete

- $SAT \in NP$: A satisfying assignment z is a polynomial length certificate for the problem.
- SAT is NP-hard by Cook-Levin Theorem:
  For each input x of a language $L \in NP$ accepted by a TM M, a SAT formula ϕ_x is constructed based on the configurations of M such that $x \in L$ if and only if ϕ_x is satisfiable.
  Suppose $T : \mathbb{N} \to \mathbb{N}$ is the function denoting the maximum number of steps M executes on any instance - on input instance x, the time taken is $T(|x|)$.
  If $|x| = n$, then $|\phi_x|$ is $T(n) \log n$.

3-SAT is NP-complete

- 3-SAT: Each clause has at most 3 literals.
- 3-SAT in NP as it is a special case of SAT.
- 3-SAT is NP-hard as $SAT \leq_p 3 - SAT$: For each CNF formula ϕ we construct a 3-CNF formula ψ such that ϕ is satisfiable if and only if ψ is satisfiable.
  Take a clause C, say $C = (\ell_1 \vee \ell_2 \vee \ldots \vee \ell_k)$, where ℓ_is are literals.
  Delete C, introduce a new variable z
  Introduce new clauses $C_1 = (\ell_1 \vee \ell_2 \ldots \ell_{k-2} \vee z)$ and $C_2 = (\ell_{k-1} \vee \ell_k \vee \overline{z})$.
  There is an assignment satisfying C if and only if there is an assignment of $\{u_1, u_2, \ldots, u_n\} \cup \{z\}$ that satisfies $C_1 \wedge C_2$
  Repeat this strategy of reducing the size of clauses by introducing new variables till all clauses have at most 3 literals – this is ψ.

coNP-completeness:

coNP-completeness for language $L \in coNP$: Any language $L' \in coNP$ is such that $L' \leq_p L$.

- TAUTOLOGY $= \{\phi | \forall z \in \{0,1\}^n, \phi(z) = 1\}$.
- TAUTOLOGY is the complement language of SAT: in coNP.
- SAT is NP-complete: For any language $L \in NP$, $L \leq_p SAT$. $\implies \overline{L} \leq_p \overline{SAT} = TAUTOLOGY$.
- TAUTOLOGY is coNP-complete.