Complexity Theory

University of Cambridge

Ashwin Ahuja

Computer Science Tripos Part IB
Paper 6

May 2019

Complexity Theory

Algorithms And Problems

① Aim: understand what makes certain problems difficult to solve algorithmically - requires inordinate time and memory space.

Sorting

$f = O(g)$ if $\exists\, n_0 \in \mathbb{N}$, const c st $n > n_0,\ f(n) \leq cg(n)$

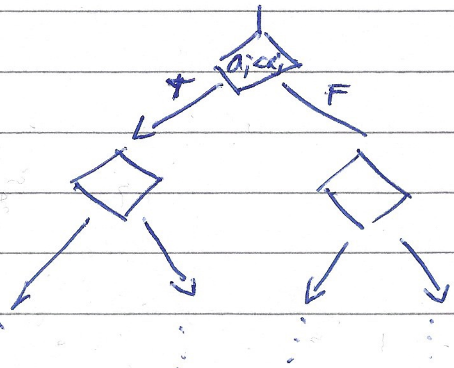
$f = \Omega(g)$ if $\exists\, n_0 \in \mathbb{N}$, const c st $n > n_0,\ f(n) \geq cg(n)$

$f = \Theta(g)$ if $f = O(g)$ and $f = \Omega(g)$

Establishing lower bound:

↳ Assume numbers all distinct $(a_1, \ldots, a_n)$

↳ At each branch point, boolean decision - can be represented by computation tree



↳ $n!$ different ways that initial collections presented.

Therefore, $n!$ leaves

↳ binary tree with $n!$ leaves has ordering $\log_2(n!)$

$$\log(n!) = \log(n) + \log(n-1) + \ldots + \log(1)$$

$$< \log(n) + \log(n) + \ldots + \log(n) = n\log n = O(n\log n)$$

AND $> \log(n/2) + \log(n/2) + \ldots \log(n/2) = n/2 \log(n/2) = O(n\log n)$

$$\therefore \log(n!) = \Theta(n\log n)$$

Travelling Salesmen Problem

Given set V of vertices, along with cost matrix $c: V \times V \to \mathbb{N}$, giving foreach pair of vertices a positive integer cost, in order to minimise the total cost

(2) $$c(v_n, v_1) + \sum_{i=1}^{n-1} c(v_i, v_{i+1})$$

In same way as sorting, trying to find possible ordering $\Rightarrow \Omega(n \log n)$. However, best known upper bound is $O(2^n n^2)$

Turing Machines

Q - finite state of states

Σ - finite state of symbols (disjoint from Q) - contains blank - $\sqcup$ and left end marker $\triangleright$

$s \in Q$ - initial state

$\delta : (Q \times \Sigma) \rightarrow Q \cup \{acc, rej\} \times \Sigma \times \underbrace{\{L, R, S\}}_{\text{left, right, stationary}}$ - transition funct

(q, w, u) - configuration = machine in state q ~~in state~~ with $q \in Q$ $w, u \in \Sigma^*$ | string wu on tape and head pointing to last symbol in w.

Computation: proceeding through series of configurations - specified by transition function δ:

$$(q, w, u) \longrightarrow_M (q', w', u')$$

if (1) $w = va$; (2) $\delta(q, a) = (q', b, D)$ and (3) $D = L \wedge w' = v, u' = bu$ OR $D = S$ and $w' = vb$ and $u' = u$ OR $D = R$ and $w' = vbc$ and $u' = x$ where $u = cx$. If u empty $w' = vb\sqcup$ and u' empty

$\longrightarrow^*_M$ is reflexive and transitive closure of $\longrightarrow_M$

Each machine M defines language $L(M) \subseteq \Sigma^*$ which it accepts

↳ $L(M) = \{x \mid (s, \triangleright, x) \longrightarrow^*_M (acc, w, u)$ for some w and $u\}$

↳ strings excluded in $L(M)$ include those which reach rej and those that lead to machine running forever

③

① Recursively Enumerable : $L \subseteq \Sigma^*$ is if it is $L(M)$ for some M.

↳ also are semi-decidable - may not halt on strings not in language

② Decidable : L decidable if $L(M)$ for some machine M which halts on every input

③ Computable : function $f : \Sigma^* \to \Sigma^*$ computable if machine M st $\forall x \quad (s, \triangleright, x) \xrightarrow{*}_{M} (acc, f(x), \epsilon)$

Example: Halting Problem is recursively enumerable but not decidable. If $[M]$ denotes string representing machine M, then problem H is defined:

$$H = \{ [M], x \mid M \text{ halts on input } x \}$$

Can also have Turing Machines with multiple tapes

Complexity

For machine M, running time of M is $r : \mathbb{N} \to \mathbb{N}$ st $r(n)$ is length of longest halting computation of M on input of length n. $r(n) = 0$ if M does not ~~halt~~ halt on input of length n.

For function $f : \mathbb{N} \to \mathbb{N}$, say that language $L \in TIME(f(n))$ if $\exists$ machine M st: (1) $L = L(M)$; (2) Running time of M is $O(f(n))$

SPACE$(f(n))$: languages accepted by a machine which uses at most $O(f(n))$ tape cells on inputs of length n.

↳ tape cells of the work tape

Can be shown two tape (or 1 tape) (read-only and work) Turing Machine simulates all other forms of computation - ~~at~~ at worst, polynomial factor increase in time and space complexity

Church-Turing thesis: Any two reasonable models of computation are polynomially equivalent

④

<u>Decidability and Complexity</u>: <u>every decidable language</u> has <u>a</u> time complexity

↳ Machine M st $L = L(M)$ and st M halts on all inputs and f is function mapping n to max number of steps taken by M on all poss string of length n.
Can construct algorithm given n that simulates M on all possible ~~strengths~~ strings of length n. - since M halts on all inputs, this terminates

But for <u>semi-decidable language</u> this is not true as f is not a computable function ($\nexists$ computable function st $f = O(g)$)
cannot be bounded above any computable func.

↳ Say $\exists f$ that is computable that $\forall x \in L$. $len(x) = n$, M accepts x in at most $f(n)$ steps. Can then construct machine M' that accepts L and always halts: M'(input x) takes length n of x and computes $f(n)$ - then simulates M $f(n)$ times - if acceptance, M' accepts. If not rejected. Hence M' halts for all inputs and accepts the same as M - therefore contradiction as L is not decidable

<u>Nondeterminism</u>: If make δ not a function but an arbitrary relation (multiple poss outputs) obtain nondeterministic Turing machine. Can be pictured as a tree of successive configuration. A deterministic TM can simulate a nondeterministic TM by carrying out BFS on computation tree until accepting configuration is found. <u>However, not</u> clear that simulation can be carried out in polynomial time

↳ need to find that height of computation tree on input string x is bounded by polynomial $p(|x|)$ - in actuality it is $O(2^{cp(x)})$ for some constant c.

Complexity Classes

Collection of languages - specified by three things

↳ ① model of computation

↳ ② resource (time, space, etc)

↳ ③ set of bounds

⑤ If considering deterministic machines, if choose functions to be broad ~~e~~nough languages included in complexity class does not depend on model of computation

↳ true for polynomial but not linear

$$\boxed{P = \bigcup_{k=1}^{\infty} TIME(n^k)}$$ → problems decidable in polynomial time.

Tractable = in P (for languages) - P is problems that are feasibly computable

EXAMPLE PROBLEMS

Reachability: Given directed graph $G = (V, E)$ and two nodes $a, b \in V$ want to know if there is a path from a to b in G. Algorithm is as follows:

↳ ① mark node a, all other nodes unmarked and initialise set S to $\{a\}$

↳ ② while S not empty: (1) choose node i in S, (2) remove i from S and (3) $\forall j$ st $\exists$edge $(i,j) \wedge j$ is unmarked, (4) mark j and (5) add j to S

↳ ③ if b is marked, accept, else reject

Time: This is $O(n^2)$ - each edge examined once → in polynomial time. (P)

Space: Only need two sets in workspace - S and marked vertices - each n bits therefore $O(n)$ $\{SPACE(n)\}$

Euclid's Algorithm:

RelPrime = $\{(x,y) \mid gcd(x,y) = 1\}$ | length of input $\overset{n \text{ bits}}{=} \log x + \log y$

Algorithm:

↳ Input (x,y)

↳ ~~Re~~ while $y \neq 0$:

$x \leftarrow x \bmod y$

swap x and y

↳ if $x = 1$, accept else reject

— repeated $2 \log x$ times ∴ in P {it is polynomial in length of input}

⑥ Prime Numbers: PRIME = $\{x \in \{0,1\}^* \mid x$ is bin representation of a prime number$\}$

There is a polynomial time algorithm for solving this (AKS method): embeds problem of checking primality into that of factoring polynomials. If a and p are co-prime then:

$$(x-a)^p = (x^p - a) \bmod p \iff p \in P \text{ (primes)}$$

To check this requires exponential time but AKS showed it was suitable to check it modulo a polynomial $x^r - 1$ for suitable small values of r.

Boolean Expressions: set of expressions formed from infinite set $X = \{x_1, x_2, \ldots\}$ of variables and $\{true, false\}$ by following rules:

1. const or variable is an expression
2. ϕ is boolean expression $\iff$ so is $\neg\phi$
3. if ϕ and φ ~~booth~~ both boolean expressions so are $(\phi \wedge \varphi)$ and $(\phi \vee \varphi)$

If contains variables, true or false for a given truth assignment

Evaluation Algorithm: scan input (each scan is $O(n)$) looking for subexpressions that match LHS of set of rules and replace with what it maps to $- O(n^2)$
↳ max of n times

Circuit Value Problem: directed graph $G = (V,E)$ with $V = \{1, \ldots, n\}$ with labelling $l: V \to \{true, false, \vee, \wedge, \neg\}$, satisfying:

1. If edge $(i,j) \Rightarrow i < j$
2. Every node in V has indegree at most 2
3. Node v has: indegree $0 \iff l(v) \in \{true, false\}$
   indegree $1 \iff l(v) = \neg$
   indegree $2 \iff l(v) \in \{\wedge, \vee\}$

Can be used to represent arbitrary Boolean expressions. The problem (find val of result node) is solvable in polynomial time.

7

<u>Satisfiability</u>: is there a truth assignment that makes a Boolean expression true.

SAT: set of all satisfiable Boolean expressions - language has time complexity $O(2^n n^2)$

2^n: n of possible assignments

n^2: see if resulting expression is true

VAL: set of valid expressions (all truth assignments make it true) - $O(2^n n^2)$

<u>Hamiltonian Graphs</u>: G is Hamiltonian if it has a Hamiltonian cycle - path starting and ending at same vertex and every node appears on cycle exactly once - this is an NP problem.

<u>Graph Isomorphism</u>: Given two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, they are isomorphic if $\exists$ bijection $\iota : V_1 \to V_2$ st $\forall u, v \in V$:

$$(u, v) \in E_1 \iff (\iota(u), \iota(v)) \in E_2$$

At a first glance - using naive method of trying all possible bijections, would take $O(n!)$ where n is number of vertices in each graph

Nondeterministic Polynomial Time

Language L in NP is one that can be solved by algorithm in two steps: (1) Prover and (2) Verifier (Generate and Test)

Verifier: deterministic algorithm st

$$L = \{x \mid (x, c) \text{ accepted by } V \text{ for some } c\}$$

If V runs in time polynomial in length of x, L is polynomially verifiable.

NTIME(f) denotes class of language L are accepted by nondeterministic TM M st $\forall x \in L$ of length n, $\exists$ computation that is accepted of M on x of length $> f(n)$

$$\boxed{NP = \bigcup_{k=1}^{\infty} NTIME(n^k)}$$

equiv to L being polynomially verifiable

(8) Nondeterministic algorithm that accepts language L.

① input x of length n

② nondeterministically guess c (algorithm writes string of length $p(n)$ on tape - nondeterministic choice at each step of which symbol to write on tape) of length $\leq p(n)$

③ run V on (x, c)

For every string c of length at most $p(n)$ $\exists$ computation sequence that results in c being written on string

Suppose (1) M is nondeterministic machine that accepts L and runs in time $p(n)$ and (2) in any configuration, $\exists \leq k$ possible next configuration.

$\exists$ Deterministic algorithm V that takes input (x,c) and does the follows: at i^{th} nondeterministic choice point, V looks ($\hookrightarrow$ c is string of length $p(n)$ over k-letter alphabet) at i^{th} character in c to decide which branch to follow. If M accepts, then V accepts, else it rejects. $\therefore$ V is polynomial verifier for L.

<u>Reductions</u> (used to establish undecidability of languages)

Given two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, a reduction of L_1 to L_2 is function (computable) $f: \Sigma_1^* \to \Sigma_2^*$ s.t. $\forall$ string $x \in \Sigma_1^*$, $f(x) \in L_2 \iff x \in L_1$

① If there is a reduction from L_1 to L_2 and L_2 is decidable, then L_1 is as well. / if L_2 is undecidable, then L_1 is also undecidable

② ~~If~~

<u>Resource-Bounded Reductions</u>: When concerned about polynomial time computability rather than be computed within bounded resources. If f is reduction from L_1 to L_2 and f is computable by algorithm running in polynomial time, L_1 is polynomial time reducible to L_2:

$L_1 \leq_p L_2 \Rightarrow L_2$ is at least as hard as L_1 ($L_1 \leq_p L_2 \wedge L_2 \in P \Rightarrow L_1 \in P$)

Can compose algorithm computing reduction with decision procedure for L_2

(9) to get a polynomial time decision procedure for L_1, string $f(x)$ produced by reduction f on input x ~~bound~~ must be bounded in length by polynomial in length of x - hence L_2 decision procedure runs on input ~~#~~ $f(x)$ runs in time polynomial in length of x.

NP-Completeness

NP-Hard: language L is NP-hard if for every language $A \in NP$, $A \leq_p L$ [N.B. $\leq_p$ is transitive]

NP-Complete: language L is NP-complete if in NP and is NP-hard

Showing SAT is NP-complete: RTP: $\forall$ languages $L \in NP$, $\exists$ polynomial time reduction from L to SAT.

L ~~is~~ ~~in~~ NP so $\exists$ machine $M = (Q, \Sigma, s, \delta)$ and polynomial p st x is in L if accepted by M within ~~n^k~~ $p(|x|)$ steps $\equiv n^k$

Construct $f(x)$ using following variables

STATE $\hookrightarrow S_{i,q}$ for each $i \leq n^k$ and $q \in Q$: true if machine at time i in state q

TAPE $\hookrightarrow T_{i,j,\sigma}$ for each $i,j \leq n^k$ and $\sigma \in \Sigma$: true if at time i, symbol at position j is σ

HEAD $\hookrightarrow H_{i,j}$ for each $i,j \leq n^k$. true if head pointing at j at time i.

Total number of variables $= |Q| n^k + |\Sigma| n^{2k} + n^{2k}$

$f(x)$ built as conjunction of

1. $S_{1,s} \wedge H_{1,1}$ (Start)
2. $\bigwedge_i \bigwedge_j (H_{i,j} \longrightarrow \bigwedge_{j' \neq j} (\neg H_{i,j'}))$ (head never in two places)
3. $\bigwedge_i \bigwedge_j \bigwedge_\sigma (T_{i,j,\sigma} \longrightarrow \bigwedge_{\sigma' \neq \sigma} (\neg T_{i,j,\sigma'}))$ (each tape cell ~~never in two~~ contains ~~two~~ 1 one symbol)
4. $\bigwedge_i \bigwedge_j \bigwedge_{\cancel{q}} (\cancel{\exists} S_{i,q} \longrightarrow \bigwedge_{q' \neq q} (\neg S_{i,q'}))$ (never in two states)
5. $\bigwedge_{j \leq n} T_{1,j,x_j} \wedge \bigwedge_{n < j} T_{1,j,\sqcup}$ (at time 1, tape contains string x in first n cells and is blank after that)

10

⑥ $\bigwedge_i \bigwedge_j \bigwedge_{j' \neq j} \bigwedge_\sigma (H_{i,j} \wedge T_{i,j',\sigma}) \rightarrow T_{i+1,j',\sigma}$

(tape only changes under the head)

If head at time i is at position j and at the same position j' on the tape ($j \neq j'$) contains σ, then j' still contains σ at time $i+1$

⑦ Δ = set of triples (q', σ', D) st $((q,\sigma),(q',\sigma',D)) \in \delta$ and $j' = \begin{cases} j & \text{if } D=S \\ j-1 & \text{if } D=L \\ j+1 & \text{if } D=R \end{cases}$

$$\bigwedge_i \bigwedge_j \bigwedge_\sigma \bigwedge_q (H_{i,j} \wedge S_{i,q} \wedge T_{i,j,\sigma}) \rightarrow \bigvee_\Delta (H_{i+1,j'} \wedge S_{i+1,q'} \wedge T_{i+1,j,\sigma'})$$

asserts that change from time step i to $i+1$ for each i according to transition relation δ. If at time i, head position j, state q and symbol σ then state q' and head position j' at time $i+1$ and symbol at j at time $i+1$ is obtainable from one of transitions allowed by δ

⑧ $\bigvee_i S_{i,acc}$ (at some point at time i, accepting state is reached)

Given all of these, can always write Boolean expr that is satisfiable iff any nondeterministic machine M accepts x in time $p(|x|)$

Therefore straightforward to check that expr can be constructed in time polynomial in $|x|$

Conjunctive Normal Form if conjunction of set of clauses, each a disjunction of literals (variable or negation of variable). Each Boolean Expression is equivalent to CNF form (through repeated laws of distribution, ~~and~~ DeMorgan's Laws and law of double negation)

Algorithm for converting boolean expression to CNF is non-polynomial - because there are (as $n \to \infty$) expressions ϕ of length n st the shortest CNF expression equivalent to ϕ has length $\Omega(2^n)$. However, can use above reduction from any language in NP to SAT there is already a polynomial time algorithm to convert into CNF expressions - rewrite above rules in CNF and done.

(ii) Hence for every language in NP, $\exists$ a polynomial time computable function f st $f(x)$ is a CNF expression for all x and $f(x)$ is satisfiable $\iff x \in L$

↳ CNF-SAT is NP-complete

3CNF - in CNF and each clause is disjunction no more than 3 literals

$\forall$ CNF expression ϕ, $\exists$ expression ϕ' in 3CNF st ϕ' is satisfiable iff ϕ is (and $\exists$ algorithm (in polynomial time) to convert ϕ to ϕ')

Hence ~~3SAT $\le_p$~~ CNF-SAT $\le_p$ 3SAT $\Rightarrow$ 3SAT is NP-complete

NP-Complete Problems

Graph Problems

IND = the set of pairs (G, K) where G is a graph and K is an integer st G contains an independent set with $>$ K vertices - converted into problem by setting target size in input.

Independent set: $X \subseteq V$ is independent set if no edges (u, v) in E for any $u, v \in X$

This is in NP - nondeterministically generate an arbitrary subset of vertices and in polynomial time check (1) $>$ K vertices and (2) independent set.

Can show NP-complete by finding reduction from 3SAT to IND: map Boolean expression ϕ in 3CNF with m clauses to pair (G, m) where G is a graph and m is target size:

↳ G contains m triangles (one per clause) with each node representing one literal in clause - edge between two nodes of different triangles if they represent negated literals of one another

Need to show: (1) transformation with polynomial time algorithm, (this is reasonably easy to show) (2) G has independent set $\iff \phi$ is satisfiable

(12) ($\Leftarrow$) Assume ϕ is satisfiable

Let t be satisfying truth assignment. For each clause in ϕ, pick one ~~so~~ literal satisfied by t, with x being corresponding vertices – this has no two edges in a triangle and nothing from a to $\neg a$ $\therefore$ this is an independent set

($\Rightarrow$) G has independent set with m vertices. Let X be an independent set. Must contain exactly one vertex from each triangle. Generate truth assignment ~~here~~ using the vertices ~~&~~ $\therefore$ done

<u>Clique</u>: subset $X \subseteq V$ if $\forall u,v \in X$ $(u,v) \in E$. Can make this into a decision problem, called CLIQUE: set of pairs (G,K), G is a graph, K is an integer st G contains a clique with $\geq K$ vertices

$\exists$ algorithm that takes (G,K) guesses subset X of vertices of G containing K elements and then verifies that it forms a clique

Can show $IND \leq_p CLIQUE$ by reduction that maps (G,K) to $(\bar{G},K)$ – any clique is now an independent set $\therefore$ NP-complete.

<u>Graph Colourability</u>: if G is h-colourable $\exists$ function

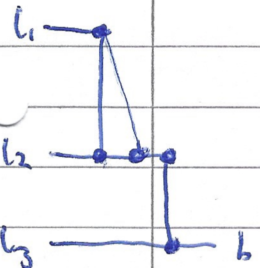
$$\chi : V \to \{1,\ldots,h\} \text{ st } \forall u,v \in E$$

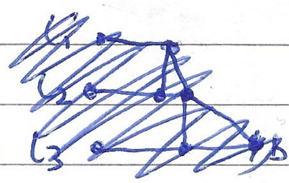
(no neighbouring edges are the same colour) $\quad \chi(u) \neq \chi(v)$

Therefore, decision problem for each h: given $G=(V,E)$, is it h-colourable. For $h>2$, this is NP-complete (for example 3-colourability is NP-complete) - checking is clearly polynomial while generation is nonpolynomial so in NP

Can show $3SAT \leq_p$ 3-colourability : map as follows:

1. 2 special vertices: a and b
2. vertex for each x and $\neg x$
3. triangle of edges between x, $\neg x$ and a
3. a and b connected
4. For each clause, five new vertices as follows





13

Hence, RTP G is 3-colourable $\iff \phi$ is satisfiable

$\Rightarrow$ ↳ a must be assigned a colour (say R) $\Rightarrow$ b is a different colour. x and ~~$\neg$~~ $\neg x$ are B and G

↳ Valid truth assignment if x is B everywhere

$\Leftarrow$ ↳ Can obtain valid 3-colouring by setting all vertices B if true and G if false.

↳ Show by showing cases on the gadget on previous page

Hamiltonian Graphs: since verification is clearly polynomial - it is in NP

3SAT $\leq_p$ HAM - this can be shown by showing $\phi \equiv$ Graph G st every satisfying assignment corresponds to a hamiltonian cycle

TSP: first need to make this a decision problem by setting target for the cost of the tour - show it as NP hard by

HAM $\leq_p$ TSP through the following reduction:

Maps graph $G = (V, E) \to (V, c: V \times V \to \mathbb{N}, n)$ where n is number of vertices in V and cost matrix is:

$$c(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E \\ 2 & \text{otherwise} \end{cases}$$

If total cost ~~$\leq n$~~ $= n$ then within budget - hence can show Hamiltonian cycle and conversely if hamiltonian cycle in graph G, clearly a way of touring with total cost n.

Sets, Numbers And Scheduling

3D-Matching: 3D extension of (bipartite matching problem): defined as problem of determining given two sets B and G of equal size and set $M \subseteq B \times G$ of pairs whether $\exists$ matching (subset $M' \subseteq M$ st each element of B and G appear on a single pair) - solvable by polynomial time

(14) 3D matching defined by:

↳ Given three disjoint sets X, Y and Z and set of triples $M \subseteq X \times Y \times Z$, does M contain a matching - is there a subset M' st each element of X, Y and Z appears in one triplet of M'

Can prove NP-~~complete~~ hard by reduction from 3SAT

↳ Given ϕ in 3CNF with m clauses and n variables.

↳ For each variable v, we include in the set X, m distinct elements $x_{v_1}, x_{v_2}, \dots, x_{v_m}$ and in Y; $y_{v_1}, \dots, y_{v_m}$. Also, include in Z $2m$ elements for each variable v: $z_{v_1}, \dots, z_{v_m}$, $\bar{z}_{v_1}, \dots, \bar{z}_{v_m}$

↳ Hence, triples in M are $(x_{v_i}, y_{v_i}, z_{v_i})$ and $(x_{v_i}, y_{v(i+1)}, \bar{z}_{v_i})$ for $i < m$ (and for $i = m$ $y_{v(i+1)}$ is set as y_v)

↳ For each clause c of ϕ, we have elements $x_c \in X$ and $y_c \in Y$.

↳ If for some variable v, the literal occurs in c, include the triple (x_c, y_c, z_{vc}) in M

↳ If $\neg v \in c$ $(x_c, y_c, \bar{z}_{vc})$

↳ Add $m(n-1)$ additional dummy elements to X and Y and include (x, y, z) in M

RTP: matching $\iff$ ϕ is satisfiable

$\Leftarrow$ ~~$\Rightarrow$~~ For matching variable v choose one of $(x_{v_1}, y_{v_1}, z_{v_1})$ or $(x_{v_1}, y_{v_2}, \bar{z}_{v_1})$ which constrain choice for all other x_{v_i} - only two ways of matching - use all z_{v_m} or $\bar{z}_{v_m}$

~~$\Leftarrow$~~ For variable set to true in satisfying truth assignment select all triples of form $(x_{v_i}, y_{v(i+1)}, \bar{z}_{v_i})$ and for false $(x_{v_i}, y_{v_i}, z_{v_i})$ (1) Hence if v true, elements z_i can satisfy x_c and y_c for clauses where v is positive literal (2) If v false, $\bar{z}_{v_i}$ available to satisfy $\neg v$ $\Rightarrow$ hence we have a matching

(15) ~~Exact Cover by 3-sets~~

Exact Cover by 3-sets

Set Covering: Given set U with $3n$ elements and collection $S = \{S_1, \ldots, S_m\}$ of three element subsets is there a ~~small~~ sub collection containing exactly n of these sets whose union is all of U.

Reduction from 3DM (atching), mapping instance of (X, Y, Z, M) of 3DM to pair (U, S) where $U = X \cup Y \cup Z$ and S consists of three element sets $\{x, y, z\}$ where $(x, y, z) \in M$

Set Covering: Given set U, collection of $S = \{S_1, \ldots, S_m\}$ subset of ~~the~~ U and integer budget B is there a collection of B sets in S whose union is U

↳ solved by reduction from exact-cover by 3-sets, mapping (U, S) to (U, S, n) where $3n$ is number of elements in U

Knapsack: given n items each with positive integer value (v_i) and weight w_i. Can we select subset of items whose weight does not exceed some max weight W and value > min value V

Reduction from exact cover by 3-sets, mapping $U = \{1, \ldots, 3n\}$ and $S = \{S_1, \ldots, S_m\} \longrightarrow m$ elements each corresponding to one S_i and having weight and value

$$\sum_{j \in S_i} (m+1)^{j-1}$$

and set target weight and value as

$$\sum_{j=0}^{3n-1} (m+1)^j$$

Represent subsets of U as strings of 0s and 1s ~~of~~ of length $3n$ - treat as representations of numbers in base $m+1$. Hence only way we can get target number (1 in all positions) ~~is~~ is if union of the sets chosen is all of U and no element more than once - this is a specific instance of Knapsack where weight and val are equal - Subset Sum Problem

(16)

Scheduling : Knapsack can be used to show scheduling problems as NP-complete

① Timetable Design
- ↳ H is set of work periods
- ↳ W is set of workers each with subset of H
- ↳ T is set of tasks and assignment $r: W \times T \to \mathbb{N}$ of required work
- ↳ is there a mapping $f: W \times T \times H \to \{0,1\}$ which completes all tasks

② Sequencing with Deadlines
- ↳ T is set of tasks
- ↳ Length $l \in \mathbb{N}$ for each task, release time $r \in \mathbb{N}$ and a deadline $d \in \mathbb{N}$
- ↳ is there a work schedule which completes task between release time and deadline
  - ↳ assignment to each task $t \in T$ ~~for which $f(w,h,t)=1$ and there is one only if w is available~~, a start time $s(t)$ st $s \geq r(t)$, $d(t) \geq s(t) + l(t)$ and st for any other t', $s(t') \geq s(t) + l(t)$

③ Job Scheduling
- ↳ Given T is set of tasks + length $l \in \mathbb{N}$ for each task
- ↳ number ($m \in \mathbb{N}$) processors
- ↳ is there a multi-processor schedule which completes all tasks by the deadline

N.B. the following are no longer if only considering planar graphs (graphs with no intersecting edges): CLIQUE, 4-Colourability (3-colourability is still NP-complete)

Approximation Algorithms : not guaranteed to be the best solution but will produce solution within known factor.

(17) Heuristics: Arise from limitations of application area and are used to cut down the search space

CERTIFICATES, FUNCTION CLASSES & CRYPTOGRAPHY

Complexity classes defined in terms of nondeterministic machine models are not necessarily closed under complementation of languages (Eg $\overline{VAL}$)

Certificates: Language $L \subseteq \Sigma^*$ in NP iff expressed as $L = \{x \mid \exists y R(x,y)\}$ where R is a relation on strings satisfying two conditions

↳ ① R is decidable in polynomial time by deterministic machine

↳ ② R is polynomially balanced - $\exists$ polynomial p st $R(x,y) \wedge$ length$(x) = n \Rightarrow$ length$(y) \leq p(n)$

If $R(x,y)$ holds, say that y is a certificate of membership of x in L - it is a solution to x.

Eg. L = SAT and x is a Boolean expression, y is assignment of truth values to variables of x and $R(x,y)$ is relation that holds if y makes x true.

co-NP

Defined as the complements of languages in NP - hence, if L is in co-NP, $\exists$ relation S which is polynomial time decidable and polynomically balanced st $\overline{L} = \{x \mid \exists y\, S(x,y)\} \equiv L = \{x \mid \forall y\, \neg S(x,y)\}$

Since P closed under complementation $\neg S(x,y)$ is decidable in polynomial time $\therefore$ co-NP is set of languages L for which $\exists$ polynomial-time decidable relation R st:

$$L = \{x \mid \forall y\, |y| < p(|x|) \longrightarrow R(x,y)\}$$

↳ co-NP is class of problem that is polynomially falsiable

$P \subseteq NP \cap$ co-NP

Language L is ~~NP~~ co-NP-complete if L is in co-NP and for every language $A \in$ co-NP, $A \leq_P L$

(18) ↳ Complement of NP-complete language is co-NP-complete

↳ if f is reduction from L_1 to L_2, then also the reduction of $\overline{L_1}$ to $\overline{L_2}$ as $x \in \overline{L_1} \iff x \notin L_1 \iff f(x) \notin L_2 \iff f(x) \in \overline{L_2}$

Factorisation:

Decision problem Factor consisting of pairs (x,k) st x has factor y with $1 < y < k$ - If problem decidable in polynomial - implies we have an algorithm for constructing the prime factorisation of any number.

Factor is in NP ∩ co-NP: a factor of x less than k is a certificate that (x,k) is a member of Factor. And in co-NP because succint certificate of disqualification

Graph Isomorphism

Easy to see if problem is in NP - since ι is a succint certificate. Has been shown as in quasi-polynomial time - ~~[illegible]~~ $O(n^{k \log n})$

Function Classes

Have generally considered decision problems as just looking for the lower bound. ~~Pa~~ However, useful to consider functions. While functions for deterministic machines are understood - harder for nondeterministic functions are harder since cannot determine the output string. Instead talk about complexity of the witness functions in NP

Witness Function for language L ($=\{x \mid \exists y\ R(x,y)\}$)

↳ Any function f st

↳ if $x \in L \Rightarrow f(x) = y \wedge R(x,y)$

↳ $f(x)$ = "no" otherwise

FNP is collection of all witness functions for languages in NP. (If NP-complete problem had a polynomial time witness function, then P=NP) Eg. Witness function for SAT would be one returning truth assignment or "no"

19 Factorisation function maps n to tuple : $(2, k_1; 3, k_2; \dots ; p_m, k_m)$ where $n = 2^{k_1} 3^{k_2} \dots p_m^{k_m}$ - is in FNP since witness function for trivial problem in NP - set of all positive integers

CRYPTOGRAPHY

Aim: enable Alice to Bob to communicate without Eve being able to eavesdrop

private key system relies on keeping e and d secret.

↳ e - encryption key ; $E(x, e)$ - encryption function

↳ d - decryption key ; $D(x, d)$ - decryption function

Can say $d = e$ and say D and E are $\oplus$ function - One Time Pad - this is provably secure

Public Key Cryptography: key e is made public while d is kept secret. D and E must be computable in polynomial time. However function that maps $E(x,e)$ to x without knowing d not computable in polynomial time. However, must be in FNP as witness function for $\{ y \mid \exists x \; E(x,e) = y \}$

Hence, not provably secure, relies on unproved hypothesis that $\exists$ function in FNP not in FP

One Way Functions required for public-key cryptography, requires:

↳ f is one-to-one - don't want two distinct plaintext to one encrypted text

↳ for each x, $|x|^{1/k} \leq |f(x)| \leq |x|^k$ for some k ↳ cannot result in more than polynomial increase in length of string

↳ $f \in FP$

↳ $f^{-1} \notin FP$

in order to have these two need to show $P \neq NP$

RSA: however, this function $f(x, e, p, q) = (x^e \bmod pq, pq, e)$ is good (public key = (pq, e))

Unambiguous machine is one st for any input x, there is at most one accepting computation of the machine. UP is the class of

(20) languages accepted by unambiguous machines in polynomial time

$UP = \{x \mid \exists_y \underbrace{R(x,y)}\}$

↳ ① polynomially time computable
↳ ② polynomially balanced
↳ ③ $\forall x\ \exists$ at most one y st $R(x,y)$ (R is a partial function)

$P \subseteq UP \subseteq NP$: in general, difficult to think of natural problems that are in UP but not in P. But, existence of one-way functions is equivalent to the statement that $P \neq$ ~~NP~~ UP

Proof : assume one-way function f

$$L_f = \{(x,y) \mid \exists z\ (z \leq x \wedge f(z) = y\}$$

L_f is clearly in UP since f is one-to-one and there is a non-deterministic machine that recognises it by guessing value for z then checking $f(z) = y$

But, not in P as could then compute f^{-1} using binary search in P time

↳ given a y, $\exists z$ st $f(z) = y$ then $z \leq 2^{k \log y}$ by ② (polynomial length increase)

↳ So can find z using binary search making $(\log y)^k$ calls - polynomial time

Can also show if language L in UP but not in P, and U is machine accepting L, exists function f_U : if x is string that encodes computation of U, then $f_U(x) = 1y$ where y is input string accepted by this computation else $f_U(x) = 0x$

↳ f_U is one-to-one because machine is unambiguous => given y has one accepting combination

↳ f_U is in FP and $f_U^{-1} \in FP \Rightarrow L \in P$ hence $f_U^{-1} \notin NP$

(21) ## SPACE COMPLEXITY

SPACE($f(n)$) : languages accepted by machine which uses at most $O(f(n))$ tape cells (of work tape)

NSPACE($f(n)$) : class of languages accepted by nondeterministic machine that uses at most $O(f(n))$ tape cells on input of length n.

$L = SPACE(\log n)$
$NL = NSPACE(\log n)$
$PSPACE = \bigcup_{k=1}^{\infty} SPACE(n^k)$
$NPSPACE = \bigcup_{k=1}^{\infty} NSPACE(n^k)$

Inclusions

$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq NPSPACE$

and since L, P and PSPACE are all closed under complementation

$L \subseteq NL \cap co\text{-}NL$, $P \subseteq NP \cap co\text{-}NP$

and $PSPACE \subseteq NPSPACE \cap co\text{-}NPSPACE$

Constructible Functions : functions which can be used for bounds
Function $f : \mathbb{N} \to \mathbb{N}$ is constructible if
↳ ① f is monotonically increasing
↳ ② $\exists$ deterministic machine which on any input length n replaces input with string $0^{f(n)}$ and runs in time $O(n + f(n))$ and uses $O(f(n))$ work space

f should not require more resources than limit exposed by f itself – allows us to compose computation of f with any other computation taking $O(f(n))$ time and space

If f and g are constructible, so are : $f+g$, $f \cdot g$, 2^f and $f \circ g$

More Inclusions
① $SPACE(f(n)) \subseteq NSPACE(f(n))$
② $TIME(f(n)) \subseteq NTIME(f(n))$
③ $NTIME(f(n)) \subseteq SPACE(f(n))$
④ $NSPACE(f(n)) \subseteq TIME(k^{\log n + f(n)})$ for constant k

→ Deterministic machine can simulate non-deterministic machine M by backtracking as well as keeping track of current config of M as well as choices for nondeterministic points. Space reqd < constant multiple of length of computation – hence space < number of steps so far – $O(f)$ where f is time bound on the machine

22 ④ Reachability ($\Rightarrow NL \subseteq P$)

↳ as let $f(n) = c\log n$

$NSPACE(\log n) \subseteq TIME(k^{(c+1)\log n}) = TIME(n^{(c+1)\log k})$

$\in P$

↳ Given directed graph $G = (V, E)$

↳ two nodes $a, b \in V$

↳ decide whether path from a to b in G

Algorithm

① write index of node a in work space

② If i is index currently on work space

(a) if $i = b$ then accept

else guess index j ($\log n$ bits) and write on work space

perform $\log n$ steps each doing the nondeterministic choice of writing 0 or 1 on work tape and moving right

(b) if (i,j) is not an edge reject

else replace i by j and return to (2)

For every j, there is a computation path that results in j being written on the tape.

Stores two indices, each $\log n$ bits $- O(\log n)$ space. Hence, reachability is in NL. Can be used to show that all problems in NL are in P. In general, trying to show: $NSPACE(f(n)) \subseteq TIME(k^{2f(n)})$

↳ M is nondeterministic machine with workspace bounded by $f(n)$ for input length n

↳ For x input (of length n) there are finite fixed configurations of M that are possible.

↳ Finite state control can be in any of q states and work tape can have any of $s^{f(n)}$ strings on it (s is number of distinct symbols) Head in one of $f(n)$ positions

↳ Total distinct configurations $= qnf(n)s^{f(n)} < n c^{f(n)}$ for const c

↳ Configuration graph is graph whose nodes are all possible configurations of M and work tape having at most $f(|x|)$ and edge between i and j iff $i \rightarrow_M j$

(23) ↳ Hence, M accepts x iff path from starting config to accepting config – reachability – $O(n^2) = O(g^2)$ where g is size of config graph

Hence time $\leq c'(nc^{f(n)})^2$ for some c' which is $k^{\log n + f(n)}$ for some $k \leq c'c^2$

Hence, $NL \subseteq P$ {$NSPACE \subseteq EXP$}

Savitch's Theorem

Can show reachability solvable by deterministic algorithm in $O((\log n)^2)$ space.

Algorithm for determining if path from a to b of length i or less

```
Path(a, b, i):
  if i = 1 and a ≠ b and no edge (a, b)
      then reject
  else if edge (a, b) or a = b
      then accept
  else for each vertex x
      if Path(a, x, floor(i/2)) and Path(x, b, ceil(i/2)) then
         accept
```

↳ Recursion can be implemented by keeping a stack of records, each a triple (a, b, i). x can be implemented as a counter (using $\log n$ bits)

↳ Each activation record on stack representing $3 \log n$ bits ($\log n$ for each of three components)

↳ Max depth of recursion is $\log n$ since val of i halved at each nested recursive call – for each, at most two activation records placed on the stack → $\underline{2 \log n \text{ records} = 6(\log n)^2 \text{ bits}} = O((\log n)^2)$

↳

Hence, for any constructible function f st $f(n) \geq \log n$, $NSPACE(f(n)) \subseteq SPACE(f(n)^2)$ → can solve config graph of nondeterministic machine which has $O(f(n))$ having $g = c^{\log n + f(n)}$ nodes

(24) hence reachability can be solved using space:

$$O((\log g)^2) = O((\log n + f(n))^2) = O(\underbrace{f(n)^2)}_{\text{since } f(n) \geq \log n}$$

However, must first produce configuration on tape - has $c^{\log n + f(n)}$ takes $> f(n)^2$ space but fix by not storing config graph on tape but instead check on the machine if configuration change is acceptable

Hence $NSPACE(f(n)) \subseteq SPACE(f(n)^2)$

Hence, Savitch's Theorem: PSPACE = NPSPACE

↳ NPSPACE = co-NPSPACE since PSPACE closed under complementation

Has also been proven that for constructible function f with $f(n) \geq \log n$,

$NSPACE(f(n)) = co\text{-}NPSPACE(f(n))$

Provable Intractability: Proof of NP-completeness does not mean problem not in P unless we can prove $P \neq NP$, but can show problem is not in P for a specific problem.

Time Hierarchy Theorem

Can use diagonalisation to construct a language with a specific lower bound - hence by increasing bounds, can show that there are more languages

For constructible function f with $f(n) \geq n$, $TIME(f(n))$ properly constrained in $TIME(f(2n+1)^2)$

↳ Halting Problem with time bound f

$H_f = \{[M], x \mid M \text{ accepts in } f(|x|) \text{ steps}\}$

(i) $H_f \in TIME(f(n)^2)$

(first compute $f(|x|)$ then have counter and simulate M $f(|x|)$ times)

25

② $H_f \notin TIME(f(\lfloor n/2 \rfloor))$

Proof: Assume true, construct machine N which accepts [M] iff [M], [M] $\notin H_f$. Machine copies [M], inserting comma between two copies and then runs machine that accepts H_f

Running time of N $= f(\lfloor (2n+1)/2 \rfloor) = f(n)$. Ask whether N accepts input [N] and get contradiction either way.

Consequences: ① $\exists$ no fixed k s.t. all languages in P can be decided in $O(n^k)$

② $EXP = \bigcup_{k=1}^{\infty} TIME(2^{n^k})$ is an extension of P

$\Downarrow$

$P \subseteq EXP$ ($EXP \nsubseteq P$)

Descriptive Complexity

Describe complexity of describing a problem.

① What kind of formal language can decision problems be formalised. For example, triangle problem (Given graph $G=(V,E)$, does it ~~create~~ ~~a~~ contain a triangle?) can be described in first-order logic, but reachability problem and 3-colourability cannot be.

First-Order Predicate Logic: $E(x,y) \mid \phi \wedge \varphi \mid \phi \vee \varphi \mid \neg\phi \mid \exists x \phi$
$\mid \forall x \phi$

if ϕ is sentence (no free variables), ~~on~~ ϕ defines a collection of graphs for which ϕ is true.

Any property of graphs definable in this way is in L.

$G \models \phi \rightarrow$ time $O(\boxed{ln^m})$ and $O(\boxed{m \log n})$ space where l is length of ϕ ~~and~~, m is nesting depth of quantifiers and n is number of vertices

(does G satisfy ϕ)

① if φ is atomic $G \models_\sigma \varphi$ by directly looking up edges

② if φ is $\theta_1 \wedge (\vee, \neg$ etc) θ_2 do each half seperately then recombine

③ if φ is $\exists x \theta$ then ~~$\exists$~~ $\forall v \in G$ $G \models_{\sigma[x/v]} \theta$

at most m nested for loops and 1 for each connective

at most m free variables and must store pointers for each = log n bits

26

To go beyond L, need to go beyond first-order logic and introduce second-order quantifiers

<u>3-Colourability</u> (R, G, B)

$$\exists R \subseteq V \exists B \subseteq V \exists G \subseteq V$$
$$\forall x (Rx \lor Bx \lor Gx) \land$$
$$\forall x (\neg (Rx \land Bx) \land \neg (Bx \land Gx) \land \neg (Rx \land Gx)) \land$$
$$\forall x \forall y (Exy \to (\neg (Rx \land Ry) \land \neg (Bx \land By) \land \neg (Gx \land Gy)))$$

<u>Reachability</u>

$$\forall S \subseteq V (a \in S \land \forall x \forall y ((x \in S \land E(x,y)) \to y \in S) \to b \in S)$$

Any set S of vertices which contains a and which is closed under the edge relation E (if contains x, will contain y if $E(x,y)$ holds) must contain b.

<u>Second-Order Logic</u> = first-order logic + collection of second-order variables : X, Y, .. where each variable has associated arity a. Two added rules

↳ ① Have atomic formulas $X(t_1, .., t_a)$ where X is a second-order variable of arity a and $t_1, ..., t_a$ are first-order terms

↳ ② If ϕ is formula, so is $\exists X \phi$ and $\forall X \phi$

<u>Existential Second-Order Logic</u> : formulas of form $\exists X_1 ... \exists X_k \phi$ where ϕ is a first-order formula. 3-colourability is ESO, but reachibility is not.

<u>Fagin Theorem</u>

ESO <=> NP - hence NP has a natural characterisation not mentioning Turing machines, nondeterminism, polynomial or time

⇒ Can show for any ESO sentence $\exists X_1 ... \exists X_k \phi$ can define a nondeterministic machine which takes graph G and determines (in time polynomial) whether $G \models \phi$

27

↳ Nondeterministically guess interpretation for $X_1, \dots, X_k$ and then checks whether ϕ holds with this interpretation

↳ time = $n^{a_1} + \dots + n^{a_k}$ where $a_1, \dots, a_k$ are the arities of variables $X_1, \dots, X_k$

↳ bounded by polynomial in n

Hence, total running time is bounded by a polynomial

⇐ this direction requires proof similar to Cook-Levin theorem - can show, given nondeterministic Turing machine M and polynomial p can write sentence ϕ_M of ESO that is true in graph G iff accepting computation of M on G of length at most $p(n)$

number of vertices in G