Computational Complexity Theory: Quick reference

vishvAs vAsuki

July 23, 2010

See [2], [1]. Under construction.

Part I

Introduction

1 Notation

Open problem: OP.

2 Research effort

2.1 Themes

2.1.1 Motivation: complexity of problems

Give algorithms researchers and engineers better understanding of the difficulty of the problem, if not practical algorithms.

Efficient Computability: Complexity class P. Upper and lower bounds on efficiency of algorithms for various problems; classification of problems into complexity classes. Efficient computation with randomness. Relationship between complexity classes. Approximation algorithms.

2.1.1.1 Limitations of worst case results

Many algorithms which are exponential time for pathological cases, turn out to be highly efficient in practice: Eg: Simplex method for solving linear programs. Usually, can make these costly cases unlikely by the use of randomization: eg: quicksort.

2.1.2 Generating and checking mathematical proofs

With interaction. Proof complexity: Minimum length of proof for a statement.

2.1.3 Computing with exotic physics

Quantum/ String theory.

2.2 Characterization of research effort

Read many papers/ books, talk to others, know many theorems, understand proof strategies, digest core results into short sentences, note open problems.

Attack open problems by trying out various ideas and strategies, fail many times, get a result, see what it implies, use the same strategy to get more results.

Models: Both the strength and the weakness of theory: sometimes they don't change fast enough.

3 Models of computation to solve problems

Decision problems. Function problems.

For grammars and automata, see Boolean Functions ref.

3.1 Turing machine

3.1.1 k-tape Turing Machine (TM)

States (Q), Alphabet (S) with start and blank symbols, Transition function (δ); 1 input tape, k-1 R/W tapes with 1 output tape, termination states. In practice, values assigned to registers represents state.

3.1.1.1 Directional restrictions: irrelevance

Bidirectionally ∞ TM of time T(n) can be simulated by unidirecionally ∞ TM in O(T(n)).

3.1.2 Measuring complexity of problems

Given input X, time taken by TM to solve it is T(X); practically, this is the number of basic operations, which are commonly taken to be scalar ops: +, -, *, /. $T(n) = \max_X T(X)$.

3.1.3 Hypercomputation

Aka super-Turing computation. Computation of non Turing compatible functions. Eg: Precise computation of integrals and limits. Not known to be physically realizable.

3.1.4 Random Access Machines (RAM)

Can be simulated by a TM in time $O(T(n)^2)$. Turing's thesis. Usual assumption: word size is $\Omega(\log n)$.

3.1.5 Time and space constructible functions

Time constructible function (Proper complexity fn): $T(n) > n$, $T(|x|)$ computable from $|x|$ in $T(|x|)$.

3.1.6 The Universal TM (UTM)

TM's as strings. 3-tape UTM can simulate TM of time T(n) in $O(T(n)^2)$ naively.

3.1.6.1 Power of 3 tapes

3-tape UTM can simulate TM of time T(n) in $O(T(n) \log T)$: Put contents of k tapes (initially blank / input symbols) in 1 tape, each cell is k sim-cells; make zones L_i, R_i of size 2^i; zones always with 0 or 2^{i-1} (half) or 2^i (full) contents; $|L_i \cup R_i| \leq 2^i$; cursor always at L_0; tape contents moved to simulate head movement; initially zones half full with contents (blank / input symbols). To simulate right head movement: Find non-empty R_i; move contents to half fill $R_0 \dots R_{i-1}$; adjust $L_0 \dots L_i$. So, min 2^{i-1} shifts required before 2^i chars in R_i shifted; So, UTM needs $O(\sum^{\log T} \frac{T}{2^{i-1}} 2^i) = O(T(n) \log T)$.

3.1.7 Oblivious TM (OTM)

Head movements same for all inputs of same size, actions may differ. Make 1 work-tape Obl TM of time $O(T(n)^2)$ from k tape TM of time T(n): Reserve T(n) cells for each of k (unidirecionally ∞) tapes; in simulation, force a visit to each cell, its left and right neighbor; overwrite cell, move simulated cursor as necessary. Make 2 work-tape Obl TM of time $O(T(n) \log T)$ from k tape TM of time T(n): Use the $O(T(n) \log T)$ UTM; mime the worst possible (constant leftward) movement for each sim-tape, but implement useful movement only.

3.1.8 Variant TM's

The probabilistic turing machine : has a fair coin.

3.1.8.1 Non deterministic turing machine

Configuration of turing machine. Configuration trees and computation paths. Given an input, visualize a computational tree.

3.2 Oracle machines

TM with a subroutine call to Oracle. Eg: P^{SAT}. $P^{NP} = NP$. NP^{NP} not trivially NP: Can't take NP^{NP} alg L and make new config tree for NP alg L': can't know when all branches of L's config tree yield false.

Part II

Classes of decision problems

4 Computability and efficiency of problems

4.1 Uncomputability by TM

Tabulate all TM strings vs all inputs; mark output (0, 1 or *); **diagonalize** (reverse output in diagonal) to get function UC no TM can solve. **Halting problem**: Uncomputable, else UC computable; Even model checking cannot predict halting of the twin primes (p, p+2) problem.

4.2 Decidability by TM

TM decides 'recursive languages'. Given query $x \in L$?, TM gives coorect answer eventually.

4.3 Acceptance by TM/ Recursive enumerability

Given query $x \in L$?, TM gives coorect answer eventually if $x \in L$. It may run for ever otherwise. TM accepts 'recursively enumerable lanugages'.

4.3.1 Enumerability

Strings in L can be enumerated. Create $S = \left\{(x, n) : x \in \{0,1\}^*, n \in N\right\}$; S is countable as $\{0,1\}^*, N$ are countable. Generate one string after another from S, check to see if the simulated TM accepts x in exactly n steps, if so print x, else continue.

4.4 Hierarchy

Time and space hierarchy theorems. [**Incomplete**]

4.5 Decision problems vs Function prolems

Efficiently convering decision problem alg (Eg: INDSET) to function problem alg with binary search. FP.

4.6 Reduction

Polynomial time / log space reduction from A to B $\implies A \preceq B$ (B atleast as hard as A). Reduction is transitive.

5 P and NP

5.1 P

P=coP. P completeness. CIRCUIT VALUE is P complete: In P by Spira and $DSPACE(log^k n) \subseteq P$; . (Find out more.)

5.2 Non determinism

5.2.1 NP

Evaluatable by a non-deterministic turning machine. All computation paths end in polytime.

5.2.1.1 Acceptance of x

If $x \in L$, there is at least 1 path which leads to acceptance of x. Else, all paths lead to rejection of x. Thus, assymetry in Acceptance/ Rejection criteria.

5.2.1.2 Other views

Membership queries $x \in L$? have poly-size certificates verifiable in poly time; there is a polynomial sized proof of membership. **OP**: Show $P \neq NP$:-).

5.2.2 co-NP

NP: $\exists$ problems (SAT), coNP: $\forall$ problems (UNSAT). $NP \cap coNP$: $\exists$ both membership and disqualifier cert.

5.2.3 NP completeness

NP hardness vs completeness. TMSAT(TM string, input, time limit) NP compl.

5.2.3.1 SAT(b) is NP compl

(Cook - Levin): $SAT \in NP$; Take L in NP; for $x \in L$, $\exists$ cert u, oblivious 1 work-tape poly-time TM M with M(x,u)=1 of time $T'(n) = O((T(n))^2)$; make vars for all T'(n) work-tape and input-tape cell visits: if cell is visited t times, there be t+1 vars to track t+1 values over time; make formula f from M!: (state, cells at T=0) $\wedge$ (deduction of state, cell visit at T=1) ...; u and state sequence is certificate for satisfying f; size of formula is $O((T(n))^2)$; f efficiently convertible to CNF: each clause takes 2^c time for constant c. For formula of size $O(T(n) \log T)$, use $O(T(n) \log T)$ OTM with similar trick, cert.

Reduction from k-SAT to 3-SAT: Use this trick repeatedly: $(a \vee b \vee c \vee d) = (a \vee b \vee z) \wedge (\sim z \vee c \vee d)$.

5.2.3.2 coNP completeness

UNSAT is in coNP; Also coNP complete : for any L in coNP, use reduction to SAT used by $\bar{L} \in NP$ to get UNSAT version of L. **OP**: Is $coNP \neq NP$?

5.3 Approximation algorithms

Approximate optimal solution to NP hard problem. Performance ratio of approx alg wrt optimal alg. For approximation algs using approximation of IP problem using LP, see randomized algs ref.

6 Other classes

6.1 Space-classes

$NTIME(f(n)) \subseteq DSPACE(f(n))$.
L, NL, polylog space, $PSPACE = NSPACE$. $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$: One of these is $\subset$: from space hierarchy theorem.

Reachability method on configuration graph:
$NSPACE(f(n)) \subseteq DTIME(2^{f(n)})$.

Directed reachability in $DSPACE(log^2 n)$ (Savitch) [**Check**]. So, by reachability method: $NSPACE(f(n)) = DSPACE((f(n))^2)$. Reachability is NL complete. [**Find proof**]

Immerman-Szelepcsenyi: $Unreachability \in NL$.
So, coNSPACE(f(n)) = NSPACE(f(n)).

6.2 Boolean circuits

See Boolean fn ref.

6.2.1 Crictuit families and uniformity

Circuit family $\{C_n\}$: Circuits for various input size. Uniformity: TM can efficiently construct ckt given input size n (on input 1^n). Log space uniform ckt family.

6.2.2 P/Poly : Non uniform

$P/Poly$ inlcudes non uniform ckts ('advice'). $P \subseteq P/Poly$: Take OTM for alg; Make ckt to sync with head movements and state change.

6.2.3 NC and AC

NC_k: (Nick's class) polylog depth, uniform, bounded fan-in. Also polylog space from defn.

AC_k: NC_k with unbounded fanin.

6.2.3.1 Their relationship

$AC_k \subseteq NC_{k+1}$. $AC_0 \subset NC_1$: $PARITY \notin AC_0$ [**Find proof**].

$NC_k \subseteq L^k$: just need to store current path.

6.2.4 PRAM

RAM model with parallel processors, shared mem. Logspace uniform family of PRAMs. Parallel computation time: f(ckt depth). AC_k = Languages L decided by concurrent read/ concurrent write PRAM programs with polynomial procs and $O(\log^k n)$ time: Take ckt for L; 1 proc for each edge, 1 mem location for each gate; AND gate initialized with 1, OR gates initialized with 0.

6.2.5 Arithmatic

a+b $\in AC_0 \subset NC_1$: trivial ckt.

6.2.5.1 Multiplication

Find ab. $O(n^2)$ school alg; $O(n \log n)$ FFT alg:
$a = \sum a_i 2^i = p(2), b = q(2)$: p and q are polynomials.

$a * b \in AC_1$.

6.2.5.2 Matrix multiplication

$c_{i,j} = \sum_{k=1}^{n} A_{ik} B_{kj}$: both * and $\sum$ doable in $O(\log n)$ ckt: $\in NC_1$.

Boolean multiplication: $c_{i,j} = \vee A_{ik} B_{kj} \in AC_0$.

Matrix powering by repeated squaring: $\in NC_2$.

$DETERMINANT \in NC_2$: do gaussian elimination, multiply.

Boolean powering $\in AC_1$: repeated squaring. So, $REACHABILITY \in AC_1$. So, $NL \subseteq AC_1$.

6.2.6 Randomized ckts

Random bits r, $C_n(x,r)$. Contains BPP. RNC: RP for NC ckts.

6.2.6.1 Non-uniformity stronger than randomness: $BPP \subseteq P/Poly$

Take BPP alg; get randomized ckt $C_n(x,r)$; reduce Pr ($C_n(x,r)$ wrong for fixed x, rand r) to 2^{-n-1}; so Union bound: Pr (rand r bad) $\leq 2^{-1} < 1$; so $\exists$ r for which $C_n(x,r)$ correct; get deterministic ckt.

Can make $log_{\frac{3}{2}} n$ depth tree from any tree like ckt. So, polynomial size boolean formulae in NC_1.

6.3 Probabilistic computation

6.3.1 One sided error: RP

Aka Monte Carlo alg h. RP: $Pr(h(x) = 1 | x \in L) \geq 0.5$, $Pr(h(x) = 1 | x \notin L) = 0$. Visualize the possible execution of an RP alg with computation tree: 2 types of leaves: yes or no.

Also see randomized algorithms ref. Bounding error prob p. Boosting confidence: $(1-p)^{\frac{l}{p}} \leq e^{-l}$. $RP \subseteq NP$ (NP has one sided error too).

6.3.2 2 sided error: BPP

PP: $Pr(accept(x) | x \notin L) < 0.5$. No good if $Pr(accept(x) | x \notin L) = 0.5 - 2^{-O(n)}$

Also see randomized algorithms ref. $BPP \subseteq PP$: $Pr(accept(x) | x \in L) \geq 0.5$, $Pr(accept(x) | x \notin L) \leq 0.5 - \frac{1}{q(.)}$; 2-sided bounded error; Boosting accuracy: Run many trials, take majority; use Chernoff. $RP \subseteq BPP$.

6.3.3 ZPP

$RP \cap coRP = ZPP$: Las Vegas alg by combining RP, coRP algs : both unsure when RP alg says $x \in L$ and coRP alg says $x \notin L$; (check).

7 Other notions of complexity

7.1 Communication complexity

See Information and coding theory ref.

7.2 Komogrov complexity

Of a computational object: Resourses required to specify it. Eg: If P is a program which outputs a string x, then P is a description of x. The length of the description is just the length of P as a character string.

7.3 Smoothed complexity

[**Incomplete**]

8 Other topics

8.1 Explore further

Interactive proofs. Transperent proofs: a small error shows up every where - make combinations of atomic results.

8.2 Lower bounds

Crossing sequence: an important trick. [**Incomplete**]Adverserial technique: Let adversary make an alg, you make an input to screw it up. [**Incomplete**]Use information theory. Reduce to other hard problems. Construct input for which minimum processing is required.

$P \neq NP$ proof is hard: natural proofs (satisfy constructivity and largeness) won't work: else you'll end up with an alg =f(proof) to solve NP prob in Poly time. [**Find proof**]

8.3 Problems

8.3.1 Graph Problems

Graph G.

NP complete: Is S the largest independent set in G? Does there exist a route for the travelling salesman in G, of cost c? Graph Isomorphism (G1, G2). Is G 3-colorable?

Connectivity (G, u, v).

8.3.2 SAT Problems

TMSAT: (TM string M, input x, time limit) | M accepts (x, u) with cert u. SAT: satisfiable (cnf). 3-SAT: satisfiable (3-cnf). UNSAT: unsatisfiable (b). Diophontine equations (Polynomial eqns with integer coeff) has integer solns?. Subset sum: (S,n). Linear programming (Soln to linear inequalities). Integer programming (Find integer soln).

8.3.3 Number Theory problems

Factoring(n). IsPrime(n).

Bibliography

[1] Sanjeev Arora and Boaz Barak. *Complexity Theory: A Modern Approach*. Princeton University Computer network, http://www.cs.princeton.edu/theory/complexity/frontmatter.pdf, January 2007.

[2] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, November 1993.