CS-E4530

Computational Complexity Theory

Janne H. Korhonen
Aalto University
Spring 2018

Lecture 16:

Fine-grained Complexity, Counting and Closing Remarks

Random-acccess Machines

Limitations of Turing Machines

- **Turing machines are unwieldy for discussing fine-grained complexity**
  - Turing machines are very mechanical and don't reflect all capabilities of modern computers
  - Indeed, Turing machines predate modern computers

- **One key limitation: no random access**
  - For example, reading the ith entry of an array takes at least i time just due to moving the head

RAM Models

- **Various random-access machine models address this limitation**

- **A random access machine has the following features (informally):**
  - An infinite number of **registers**, each capable of storing a single number
  - A finite **instruction set** (think assembly language)
  - Certain **addressing** instructions allowing direct access to a register specified by a value of other register
  - A specific instruction for **halting**

Register Values

- **One can define different RAM models based on what values the registers can hold:**
  - **Real RAM**: registers can hold arbitrary real numbers
  - **Integer RAM**: registers can hold arbitrary positive integers
  - **Word RAM**: registers can hold integers of size $O(\log n)$, where n is the length of the input

- **The first two models are very powerful, but useful for discussing upper bounds**

Time and Space for RAMs

- **Time for RAM models:**
  - Number of elementary instructions executed
  - Addressing and number operations ($+$, $-$, $=$, $\leq$) are assumed to be constant-time operations

- **Space for RAM models:**
  - Number of registers used
  - Note that real and integer RAMs can solve lots of problems in *constant* space by exploiting unbounded register balues

Fine-grained Complexity

- **Application of RAM models: understanding complexity landscape inside P**

- **Let $L \in$ P be a language**
  - What is the smallest constant $c \geq 1$ such that L can be solved in time $O(n^c)$ with random-access machines?
  - This gives rise to **fine-grained complexity**

Fine-grained Complexity

- **Typical question: what is the relative complexity of problems L_1 and L_2?**
  - **Typical result:** if L_1 can be solved in time $O(n^{c-\varepsilon})$ for some specific constant c, then problem L_2 can also be solved in time $O(n^{c-\varepsilon})$
  - Here $O(n^c)$ is usually the best known upper bound for L_1 and L_2
  - This means working with reductions that
    - can be computed in significantly faster than $O(n^c)$
    - increase the instance size sub-linearly
  - Though other variations on the theme are possible

Hard Problems in P?

Hard Problems in P?

- **Recent work in fine-grained complexity has identified certain problems in P that seem to be 'canonically expressive' in some sense:**
  - **Best known algorithm**: $O(n^c)$ for some constant c (up to sub-polynomial factors)
  - Used as a subroutine in best known algorithms for many other problems
  - Lower bound $\Omega(n^c)$ would imply that many known algorithms for other problems are optimal

- **This is not hardness in a structural complexity sense**
  - Useful for identifying relationships inside P
  - Tells us that we are facing the same algorithmic challenge for many problems

Three-sum

3SUM problem

Given a set S of n numbers, decide if there are distinct numbers $x, y, z \in S$ such that $z + y + z = 0$.

- **Trivial algorithm:** $O(n^3)$
- **Best upper bound:** roughly $O(n^2)$

- **Open:** is there a $O(n^{2-\varepsilon})$ algorithm for any $\varepsilon > 0$?

Matrix Multiplication

Matrix multiplication

Given two matrices A and B, compute the matrix product $C = AB$, where

$$C_{ik} = \sum_j A_{ij} B_{jk} \, .$$

- **Trivial algorithm:** $O(n^3)$
- **Best upper bound:** roughly $O(n^{2.373})$ (**???**)
- **Open:** what is the real complexity of matrix multiplication?
- **Matrix multiplication is very expressive problem with lots of applications**

Min-Sum Matrix Multiplication

Min-sum (semiring) matrix multiplication

Given two matrices A and B, compute the min-sum matrix product $C = AB$, where

$$C_{ik} = \min_j (A_{ij} + B_{jk}) .$$

- **Trivial algorithm:** $O(n^3)$
- **Best upper bound:** roughly $O(n^3)$

- **Open:** is there a $O(n^{3-\varepsilon})$ algorithm for any $\varepsilon > 0$?

All-pairs Shortest Paths

APSP

Given a weighted undirected/directed graph $G = (V, E)$, compute the distance $d(v, u)$ for all pairs of vertices $u, v \in V$.

- **Floyd-Warshall:** $O(n^3)$
- **Using min-sum MM:** $\log n$ applications of min-sum MM

- **Open:** is there a $O(n^{3-\varepsilon})$ algorithm for any $\varepsilon > 0$?
- Closely connected to the complexity of min-sum matrix multiplication

Set Cover with Two Sets

Set Cover with Two Sets

Given a set family $\mathcal{S}$ of size n over universe U of size m, decide if there are two sets $S_1, S_2 \in \mathcal{S}$ such that $S_1 \cup S_2 = U$.

- **Trivial algorithm:** $O(n^2 m)$
- **Open:** is there a $n^{2-\varepsilon}$ poly(m) algorithm for any $\varepsilon > 0$?

- **This question connects polynomial-time algorithms to exponential-time algorithms:**
  - If set cover with two sets can be solved in time $n^{2-\varepsilon}$ poly(m), then CNF-SAT has an algorithm with running time $2^{\delta n}$ poly(m) for some $\delta < 2$
  - That is, the **strong exponential time hypothesis** has consequences for problems in P

Counting Complexity

Counting and Enumeration

- **Problems in P and NP can be viewed as decision problems of specific type:**
  - Problem is defined by a polynomial-time Turing machine M accepting two inputs x and y
  - Question is whether for given $x \in \{0,1\}^*$, is there a $y \in \{0,1\}^*$ with length polynomial in $|x|$ such that $M(x,y) = 1$

- **We can similarly ask related counting and enumeration questions:**
  - **Counting**: count the number of y such that $M(x,y) = 1$
  - **Enumerate**: list all y such that $M(x,y) = 1$

Counting and Enumeration

- **Enumeration can clearly be very difficult**
  - The number of certificates y can be exponential in $|x|$
- **What about counting?**
  - If the decision problem is in P, what does this imply about counting?
  - Turns out counting is often more difficult than decision

Perfect Matching

Definition (Perfect matching)

- **Instance:** Bipartite graph $B = (U, V, E)$, where $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, $E \subseteq U \times V$.
- **Question:** Is there a set $E' \subseteq E$ of n edges such that for any two distinct edges $(u, v), (u', v') \in E'$, $u \neq u'$ and $v \neq v'$ (i.e., is there a **perfect matching**)?

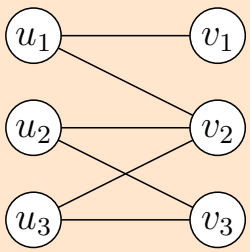
- We saw a polynomial-time randomised algorithm finding a perfect matching
- The related counting problem #MATCHING ask us to count the number of perfect matchings in a bipartite graph

Permanent and #MATCHING

- Perfect matching problem is related to computing the **determinant** of the adjacency matrix A^G
- The counting version is related to the problem of computing the **permanent** of the matrix adjacency matrix:

$$\text{perm}(A^G) = \sum_{\pi} \prod_{i=1}^{n} A^G_{i,\pi(i)} = \sum_{\pi} A^G_{1,\pi(1)} A^G_{2,\pi(2)} \cdots A^G_{n,\pi(n)}$$

Example



$$A^G = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

$$\begin{aligned} &\text{perm}(A^G) \\ &= A^G_{1,1} A^G_{2,2} A^G_{3,3} + A^G_{1,1} A^G_{2,3} A^G_{3,2} + \\ &= A^G_{1,2} A^G_{2,1} A^G_{3,3} + A^G_{1,2} A^G_{2,3} A^G_{3,1} + \\ &= A^G_{1,3} A^G_{2,1} A^G_{3,2} + A^G_{1,3} A^G_{2,2} A^G_{3,1} \end{aligned}$$

Counting and Probability

Definition (Graph reliability)

- **Instance:** An undirected graph $G = (V, E)$, vertices $s, t \in V$.
- **Question:** Compute the probability that there remains a s–t-path if all edges of G fail (i.e. are deleted) independently with probability $1/2$.

- **Graph reliability can be solved by counting:**
  - After deletions, the remaining graph is any subgraph of G with equal probability
  - Solution is thus given by counting subgraphs of G where s and t are connected, and dividing it with the number of all subgraphs

Class #P

Definition

A function $f : \{0,1\}^* \to \mathbb{N}$ is in **#P** (pronounced '*sharp p*') if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time Turing machine M such that

$$f(x) = \left|\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\}\right|.$$

- **Are all functions in #P computable in polynomial time?**
  - That is, does it hold #P $=$ FP?
  - **FP** is the class of functions $f : \{0,1\}^* \to \{0,1\}^*$ computable in polynomial time

#P-completeness

- **Completeness for #P is defined in terms of oracle reductions**
  - Generalising prior definitions, a Turing machine with oracle access to function f can obtain value $f(x)$ in single time step, assuming it has computed x
  - For any function $f : \{0,1\}^* \to \{0,1\}^*$, we denote by FP^f the class of functions computable by polynomial-time Turing machines with oracle access to f

#P-completeness

Definition

A function $f : \{0,1\}^* \to \{0,1\}^*$ is **#P-complete** if $f \in$ #P and every function $g \in$ #P is in FP^f.

Theorem

If f is #P*-complete and $f \in$* FP*, then* #P $=$ FP*.*

#P-completeness

- **Some examples of #P-complete problems:**
  - **#SAT**: counting satisfying assignments for a CNF formula
  - **#2-SAT**: counting satisfying assignments for a 2-CNF formula (decision version in P)
  - **#MATCHING** and **PERMANENT**
  - **#HAMILTONIAN-CYCLE**

- **In particular, counting versions of many problems in P are #P-complete**
  - Not everything, though: counting spanning trees is in FP

Toda's Theorem

- **How powerful is counting exactly?**
  - Clearly #P $\subseteq$ PSPACE
  - Both PH and #P are generalisations of NP; what is the relationship between these classes?

Theorem (Toda's theorem)

$$\mathrm{PH} = \mathrm{P}^{\#\mathrm{SAT}}$$

- **That is, all problems in PH can be solved in polynomial time with oracle access to a #P-complete function**

Towards Lower Bounds

Concrete Lower Bounds?

- **Proving concrete lower bounds for Turing machines and circuits seems to be out of reach**

- **Two general lines of research related to this issue:**
  - Proving lower bounds for **restricted** models of computation
  - Understanding **why** general lower bounds are difficult

Concrete Lower Bounds

- **Examples of models with concrete lower bounds:**
  - **Decision trees**: understanding how many bits of the input we have to look to decide an answer
  - **Communication complexity**:
    - Alice and Bob are both holding n-bit strings, and want to compute a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$
    - How many bits they have to communicate?
  - **Monotone circuits**: complexity for circuits without NOT gates

Circuit Lower Bounds

- **There are also (fairly weak) circuit lower bounds known**
  - AC^0 is the class of problems solvable by polynomial-size, constant-depth, unbounded fan-in circuits
  - ACC is AC^0 with counters (up to an arbitrary constant m)

- **The following represent state of the art:**
  - Parity function (that is, the counting number of ones in input modulo 2) is not in AC^0 (1980s)
  - $\text{NTIME}(2^n) \subsetneq \text{ACC}$ (Williams 2010)

Barriers: Relativisation

- **Can diagonalisation be used to prove P $\neq$ NP?**
  - Diagonalisation works for undecidability and hierarchy theorems, why not for P $\neq$ NP?

- **Diagonalisation relies on specific properties of Turing machines**

  (I) Turing machines can be efficiently represented as strings

  (II) Turing machines can be simulated by Turing machines with small overhead

Barriers: Relativisation

- **The properties (I) and (II) also hold for oracle Turing machines**
  - Implies that any statement diagonalisation proves for complexity classes defined in terms of Turing machines, it also proves it for complexity classes defined in terms of oracle Turing machines
  - This implies a limitation for diagonalisation

Theorem (Baker, Gill, Solovay 1975)

There exist languages A *and* B *such that* $\mathrm{P}^A = \mathrm{NP}^A$ *and* $\mathrm{P}^B \neq \mathrm{NP}^B$.

Barriers: Natural Proofs

- **Why are circuit lower bounds difficult?**
- **One can define a notion of natural proof for circuit lower bounds**
  - This is a specific, technical notion!
  - Most known lower bounds are natural in this sense
  - It has been proven that if sufficiently strong one-way functions exist, then natural proofs cannot prove that an explicit function f is not in $\mathrm{P}_{/\mathrm{poly}}$
- **In summary:** there is non-trivial amount of research explaining why certain 'obvious' proof techniques do not work for proving lower bounds

Lecture 16: Summary

- RAM models
- Fine-grained complexity
- Counting complexity and #P

- Explicit lower bounds for weaker models are known
- Explicit lower bounds for circuits and Turing machines seem difficult to prove