Aalto University
School of Science

CS-E4530 Computational Complexity Theory

Lecture 17: Fine-Grained Complexity, Counting and Beyond

Aalto University
School of Science
Department of Computer Science

Spring 2019

Agenda

- Random-acccess machines
- Hard problems in P?
- Counting complexity
- Towards lower bounds

Limitations of Turing Machines

- **Turing machines are impractical for discussing *fine-grained complexity***
  - Turing machines are don't reflect all characteristics of modern computers
  - Indeed, Turing machines predate modern computers
  - Perfect for computability and coarse-grained complexity (P, NP etc), not so much for fine-grained complexity (e.g. inside P)

- **One key limitation: *no random access***
  - For example, reading the ith entry of an array takes at least i steps just due to moving the tape 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)
  - A set of *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-number 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, yet useful for discussing upper bounds**
  - Algorithm design without considering low-level implementation details (which however may be significant in the very-large-n limit)

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
  - Caveat: real-number and integer RAMs can solve lots of problems in *constant* space by exploiting unbounded register values; this is not reasonable in practice

Fine-grained Complexity

- **Application of RAM models: understanding the complexity landscape inside** P
- **Let** $L \in \mathsf{P}$ **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 problem L_1 can be solved in time $O(n^{c-\varepsilon})$ for some specific constant c, then also problem L_2 can be solved in time $O(n^{c-\varepsilon})$
  - Here $O(n^c)$ is usually the best currently 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?

- **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 (this is often denoted $\widetilde{O}(n^c)$).
  - 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 in many problems

The Three-sum Problem

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

- **Trivial algorithm:** $O(n^3)$
- **Easy algorithm:** $O(n^2)$ (by sorting and testing pairs)
- **Best known algorithm:** $O(n^2 (\log\log n)^{O(1)} / \log^2 n)$
- **Open:** Is there an $O(n^{2-\varepsilon})$ algorithm for any $\varepsilon > 0$?

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:** $O(n^3)$
- **Classic algorithm:** $O(n^{2.81})$ (V. Strassen 1969)
- **Best known algorithm:** $O(n^{2.373})$ (V. Williams 2013, F. Le Gall 2014)

- **Open:** What is the real-number complexity of matrix multiplication?

- **Matrix multiplication is a very expressive problem with lots of applications**

Min-Sum 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 known algorithm:** $\widetilde{O}(n^3/2^{c\sqrt{\log n}})$ for some $c > 0$ (R. Williams 2014)

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

All-pairs Shortest Paths

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

- **Classic algorithm:** $O(n^3)$ (R. Floyd, S. Warshall 1962)
- **Best known algorithm:** $\widetilde{O}(n^3/2^{d\sqrt{\log n}})$ for some $d > 0$; by $\log n$ applications of min-sum MM

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

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 an $n^{2-\varepsilon}\,\mathrm{poly}(m)$ 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 $n^{2-\varepsilon}\,\mathrm{poly}(m)$, then CNF-SAT has an algorithm with running time $2^{\delta n}\,\mathrm{poly}(m)$ for some $\delta < 2$
  - That is, the *strong exponential time hypothesis*[1] has consequences for problems in P

[1] Exponential time hypothesis (ETH) $\approx$ CNF-SAT cannot be solved in time $2^{o(n)}$. Strong ETH (SETH): there is no constant $c < 1$ such that CNF-SAT can be solved in time 2^{cn}. Here n is the number of variables in the given CNF-SAT instance.

Counting and Enumeration

- **Problems in** P **and** NP **can be viewed as decision problems of a specific type:**
  - The problem is defined by a polynomial-time Turing machine M with two inputs x and y
  - The question is whether for a given $x \in \{0,1\}^*$, there is some $y \in \{0,1\}^*$ 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 $M(x,y) = 1$
  - *Enumeration*: 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 $G = (U, V, E)$, 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*)?

- Polynomial-time algorithms for determining the existence of perfect matchings are well known (a randomised one was presented in an earlier example)
- The related counting problem #MATCHING is to count the number of perfect matchings in a bipartite graph

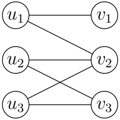
Permanent and #MATCHING

- The perfect matching problem is related to computing the *determinant* of the (symbolic) adjacency matrix A^G. (Is $\det(A^G) \equiv 0$?)
- The counting version is related to the problem of computing the *permanent* of the adjacency matrix:

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

(Number of nonzero terms in $\operatorname{perm}(A^G)$.)

Example



$$A^G = \begin{pmatrix} x_{1,1} & x_{1,2} & 0 \\ 0 & x_{2,2} & x_{2,3} \\ 0 & x_{3,2} & x_{3,3} \end{pmatrix}$$

$$\operatorname{perm}(A^G) = x_{1,1}x_{2,2}x_{3,3} + x_{1,1}x_{2,3}x_{3,2}$$

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 an s–t path if all edges of G fail (i.e. are deleted) simultaneously and independently with probability $1/2$.

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

Class #P

Definition

A function $f\colon \{0,1\}^* \to \mathbb{N}$ is in #P (pronounced '*sharp-p*' or '*number-p*') if there exists a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ 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?**
  - In other words, is #P $=$ FP?
  - *FP* is the class of functions $f\colon \{0,1\}^* \to \{0,1\}^*$ 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 a value $f(x)$ in a single time step, assuming it has computed x
  - For any function $f\colon \{0,1\}^* \to \{0,1\}^*$, 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 (note that the decision version of 2-SAT is in P)
  - *#MATCHING* and *PERMANENT* (again the decision version of MATCHING is in P)
  - *#HAMILTONIAN-CYCLE*

- **In particular, counting versions of many problems in P are #P-complete**
  - Not everything, though: counting spanning trees is in FP by the 'matrix-tree theorem' from algebraic graph theory

Toda's Theorem

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

Theorem (Toda's theorem, 1991)

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

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

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 input bits we need to check in order to determine the answer
  - *Communication complexity*:
    - Alice and Bob are both holding n-bit strings, and want to compute a function $f\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$
    - How many bits do 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*:**
  - The parity function (that is, counting the number of 1's in the input modulo 2) is not in AC^0 (J. Håstad 1987)
  - NEXP $\not\subseteq$ ACC (R. Williams 2010)
  - NQP $\not\subseteq$ ACC (C. Murray & R. Williams 2018)[2]

[2] NQP $\sim$ 'nondeterministic quasi-polynomial time', NQP $= \cup_{c>0} NTIME(n^{\log^c n})$

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

- **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 for complexity classes defined in terms of *oracle* Turing machines
  - This implies a limitation for diagonalisation

Theorem (T. Baker, J. Gill, R. Solovay 1975)

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

Barriers: Natural Proofs

- **Why are *circuit lower bounds* difficult?**
- **One can define a notion of *natural proof* for circuit lower bounds (S. Rudich & A. Razborov 1994)**
  - 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 $\mathsf{P}_{/\mathsf{poly}}$
- **In summary:** there is non-trivial amount of research explaining why certain 'obvious' proof techniques do not work for proving lower bounds

Lecture 17: 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