Computational Models - Lecture 13[1]

Handout Mode

Iftach Haitner and Yishay Mansour.

Tel Aviv University.

June 9/11, 2014

[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Talk Outline

- Time Hierarchy Theorem (*Sipser*, 9.1)
- NP Hardness (*Sipser*, 10.1, 10.2)
  - Decision, search, and optimization problems
  - Approximation
  - Randomization
  - Fixed parameter algorithms
  - Heuristics
- Concluding Remarks

Part I

Time Hierarchy Theorem

Time Hierarchy Theorem

Definition 1 (time-constructible function)

Function $t : \mathbb{N} \mapsto \mathbb{N}$ is time constructible, if it is computable in time $O(t(n))$.

Theorem 2 (time hierarchy theorem)

For any time constructible function t with $t(n) \geq n \log n$, $\exists$ language L that is decidable in time $O(t(n))$, but not in time $o(t(n)/\log t(n))$.

Corollary 3

For any $1 \leq \varepsilon_1 < \varepsilon_2$, it holds that $\text{DTIME}(n^{\varepsilon_1}) \subsetneq \text{DTIME}(n^{\varepsilon_2})$.

Corollary 4

$\mathcal{P} \subsetneq \text{DTIME}(n^{\log n})$ *and* $\mathcal{NP} \subsetneq \text{DTIME}(2^{n^{\log n}})$.

Proof: $\mathcal{P} := \bigcup_{c \geq 0} \text{DTIME}(n^c) \subseteq \text{DTIME}(n^{\frac{1}{2} \cdot \log n}) \subsetneq \text{DTIME}(n^{\log n})$

Proving the Time Hierarchy Theorem

Fix a time-constructible function t with $t(n) \geq n \log n$, and consider the following language L:

Algorithm 5 (D)

On input w:

1. Let $n = |w|$.
2. Store $Max = \lceil t(n)/\log t(n) \rceil$ in a binary counter.
3. `Reject` if w is not of the form $\langle M \rangle 10^*$, or if $|\langle M \rangle| > \log t(n)$.
4. Emulate M on w:
   - `Reject` if M makes more then Max steps (use the counter).
   - `Reject` if M accepts; `Accept` if M rejects.

Let $L = L(D)$ — the language decided by D.

Analyzing D's Running Time

Algorithm 6 (D)

On input w:

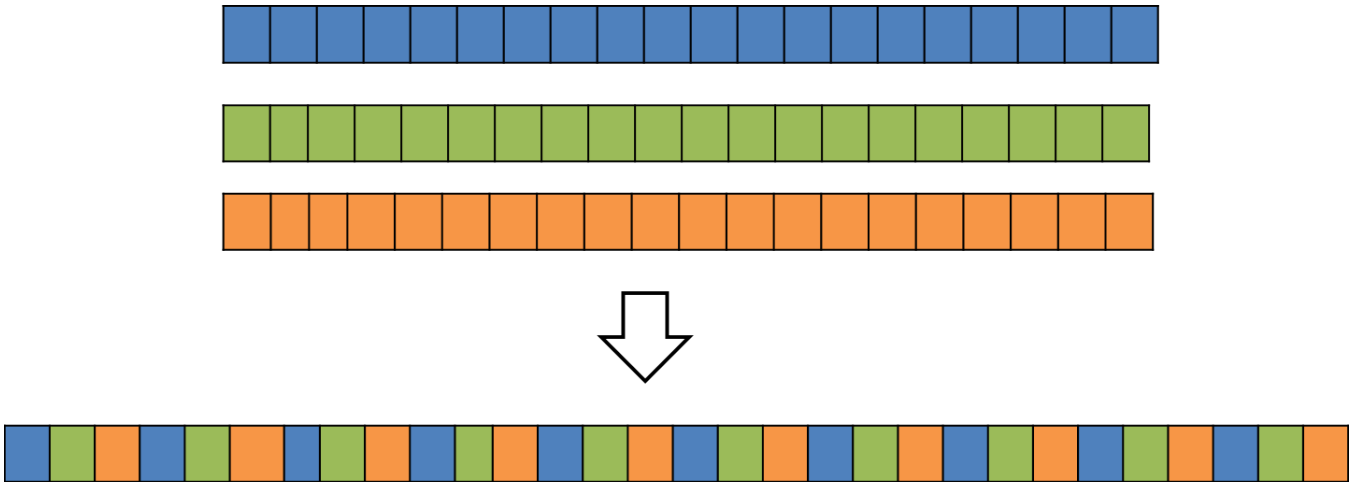
1. Let $n = |w|$.
2. Store $Max = \lceil t(n)/\log t(n) \rceil$ in a binary counter.
3. `Reject` if w is not of the form $\langle M \rangle 10^*$, or if $|\langle M \rangle| > \log t(n)$.
4. Emulate M on w:
   - `Reject` if M makes more then Max steps (use the counter).
   - `Reject` if M accepts; `Accept` if M rejects.

Claim 7

D runs in time $O(t(n))$.

- Clearly, steps $1, 2, 3$ can be performed in time $O(t(n))$.(?)
- Step 4 could be performed in time $O(t(n))$ on a three-tape TM.
- Emulating $M(w)$ in time $O(t(n))$ on a single-tape TM (while keeping track on the counter), is more challenging...

Emulating M(w) in Time $O(t(n))$



1. We use 3 tracks stored interchangeably on the (single) tape.
   1. M's emulated tape
   2. Description of M's transition function
   3. The binary counter
2. We maintain the content of tracks $2, 3$ near the head of the first track
3. Since the content of tracks $2, 3$ is of length at most $\log t(n)$ (?), the whole emulation + counter updates can be done in time $O(\log t(n) \cdot t(n)/\log t(n)) = O(t(n))$.

L is Not Decided in time $o(t(n)/\log t(n))$

Proof:

- Let M be a TM of running time $g(n) \in o(t(n)/\log t(n))$.
- For long enough w — $g(|w|) \leq t(|w|)/\log t(|w|)$.
- For large enough k — $|\langle M \rangle| \leq \log t(|\langle M \rangle 10^k|)$.

  $\Longrightarrow$ $\exists k \in \mathbb{N}$ such that for $w = \langle M \rangle 10^k$ —
  $g(|w|) \leq t(|w|)/\log t(|w|)$ and $|\langle M \rangle| \leq \log t\,|w|$.
- $D(w)$ emulates $M(w)$ to its completion.
- Hence $D(w) \neq M(w) \Longrightarrow$ M is not a decider for L.

♣

Part II

NP Hardness

NP-Hard Languages

Definition 8 (NP-hard languages)

A language B is NP-hard, if it satisfies

- Every A $\in \mathcal{NP}$ is poly-time reducible to B,

We do not require B $\in \mathcal{NP}$ (membership).
To prove B is NP-hard, suffices to prove L $\leq_P$ B, for some L $\in \mathcal{NPC}$.

Example 9

The language $L = \{\langle \Phi_1, \Phi_2 \rangle : \Phi_1 \in SAT, \Phi_2 \notin SAT\}$ is NP-hard, but apparently not NP-complete. (?)

Definition 10 (coNP-hard languages)

A language B is coNP-hard, if it satisfies

- Every A $\in$ *co*-$\mathcal{NP}$ is poly-time reducible to B

The language L from above is coNP-hard but apparently not coNP-complete (?).

NP-Hard Problems

Definition 11 (problems)

A problem is a set $T \subseteq \Sigma^* \times \Sigma^*$.
A (deterministic) algorithm S is a solver for T, if for all $x \in \Sigma^*$,
$S(x) = y \implies (x, y) \in T$
$S(x) = \perp \implies \nexists y \in \Sigma^*$ such that $(x, y) \in T$.

Deciding a language L is equivalent of solving the (decision) problem $L \times \{1\}$.

NP-Hard Problems, cont.

Definition 12 (Karp reductions)

A language L is Karp reducible to a problem T, denoted $L \leq_P^{Karp} T$, if $\exists$ poly-time oracle-aided algorithm D, such that for any solver S for T, it holds that D^S is a decider for L.[a]

[a] We do not count the oracle running time, as part of D's running time.

Definition 13 (NP-hard problems)

A problem T is NP-hard, if for every $L \in \mathcal{NP}$ is holds that $L \leq_P^{Karp} T$.

To prove T is NP-hard, suffices to prove $L \leq_P^{Karp} T$, for some $L \in \mathcal{NPC}$.

Example 14 (#3SAT)

The #3SAT problem is: given a 3CNF formula ϕ as input, compute the number of satisfying assignment for ϕ.

- #3SAT is NP-hard.
- It is likely "much harder" than NP.

Section 1

Decision, Search, and Optimization Problems

Decision Vs. Search Problems

Let $L \in \mathcal{NP}$.

- **Decision Problem:** Solving the problem $T = L \times \{1\}$ — given and input x, decide if $x \in L$.
- **Search Problem:** Solve $R_L := \{(x, y) : V_L(x, y) = 1\}$, where V_L is the canonical verifier for L.[2]
- For NP complete languages, search and decision have the "same difficulty": Karp reduced to each other.
- Clearly, $L \leq_P^{\text{Karp}} R_L$ for any $L \in \mathcal{NP}$. (?)

  Such reduction are called: Decision to Search reductions.
- Showing that $R_L \leq_P^{\text{Karp}} L$ for any $L \in \mathcal{NPC}$, is slightly more challenging.

  Such reduction are called: Search to Decision reductions.
- We show for SAT and CLIQUE.

[2] $V_L(x, y)$ returns the output of M on input x, with non-deterministic choices set to y, where M is the smallest (description wise) poly-time NTM for deciding L.

$R_{SAT} \leq_P^{\text{Karp}} SAT$

Given a formula ϕ and a Boolean variable x in ϕ, let $\phi_{x=b}$ be the formula implied by fixing x to b.

Algorithm 15 (Searcher)

Input: formula ϕ of variables $x_1 \ldots, x_\ell$. Oracle: D

1. If $D(\phi) = 0$, return $\perp$.
2. If $\phi \equiv 1$, return $\emptyset$.
3. If $D(\phi_{x_1=1}) = 1$, return $(1, \text{Searcher}(\phi_{x_1=1}))$.
   Otherwise, return $(0, \text{Searcher}(\phi_{x_1=0}))$.

Claim 16

Searcher runs in poly-time, and Searcher^D is a searcher for SAT, for any decider D for SAT.

$R_{\text{CLIQUE}} \leq_P^{\textbf{Karp}} \text{CLIQUE}$

Given a graph $G = (V, E)$:

- For $V' \subseteq v$, let $G_{-V'}$ be the graph implied by removing V' from G.
- For $v \in V$, let $N(v)$ be the neighbours of v in G (including v).

Algorithm 17 (Searcher)

Input: an integer k and graph G with vertices $V = \{v_1, \ldots, v_\ell\}$.

1. If $D(G, k) = 0$, return $\perp$.
2. If $k = 0$, return $\emptyset$.
3. If $D(G_{-v_1}, k) = 1$, return $\text{Searcher}(G_{-\{v_1\}}, k)$.
   Otherwise, return $\{v_1\} \cup \text{Searcher}(G_{-(V \setminus N(v_1))}, k - 1)$.

Claim 18

Searcher runs in poly-time, and Searcher^D is a searcher for CLIQUE, for any decider D for CLIQUE.

Search and Optimization Problems

Let $L \in \mathcal{NP}$.

- Search Problem: Given input x, find some y satisfying $R(x, y)$, or declare that none exist.
- Optimization Problem: Given input x, find some y with $(x, y) \in T$, that is the largest among all solutions (or smallest), or declare that none exist.

  Example: Given a graph G, find a clique of largest size possible.

Coping with NP-Hardness

- Approximation algorithms for hard optimization problems.
- Randomized (coin flipping) algorithms.
- Fixed parameter algorithms.
- Heuristics.

These stand in the forefront of current algorithmic research, and could easily fill up quite a few advanced courses.

Section 2

Coping with NP-Harness – Approximation

Approximation Algorithms

Solve efficiently, while giving a guarantee on the solution quality.

Approximation ratio:

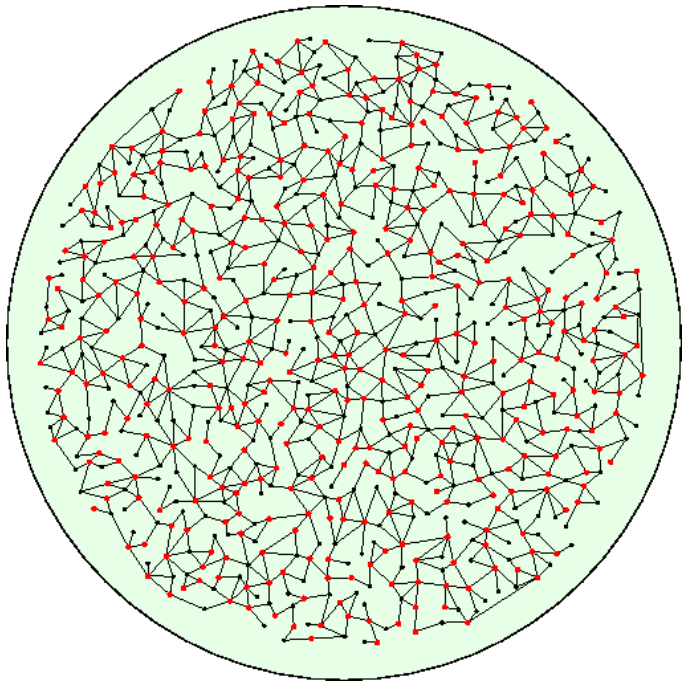
- For minimization" problem $\frac{cost(approx)}{cost(opt)}$.
- For maximization problem: $\frac{cost(opt)}{cost(approx)}$.

Vertex Cover

Given a graph $G = (V, E)$, find the smallest vertex cover $C \subseteq V$ (i.e., contains at least one endpoint for every $e \in E$).

$$\text{VertexCover} = \{\langle G, k\rangle : \exists \text{ a vertex cover for } G \text{ of size } k\}$$

Recall that VertexCover $\in \mathcal{NPC}$ by a reduction from Independent Set (proved in recitation).



(figure from http://wwwbrauer.in.tum.de/gruppen/theorie/hard/vc1.png)

Approximating Algorithm for Vertex Cover (Gavril '74)

Algorithm 19 (Appx)

Input: A graph $G = (V, E)$.

1. Set $C = \emptyset$.
2. While there are edges in E:
   1. Choose any edge $(u, v) \in E$.
   2. Add u and v to C.
   3. Remove u and v from G.

Claim 20

Appx is a 2-approximation algorithm for vertex cover – C is at most twice as large as a minimum vertex cover.

Appx is a 2-Approximation for Vertex Cover

Proof:

- The cover C constructed from $|C|/2$ edges E_C of G
- No two edges of E_C share a vertex
- Any vertex cover (including the optimum) contains at least one node from each $e \in E_C$ (otherwise e is not covered).

  $\Longrightarrow$ $OPT(G) \geq |C|/2$ (or, alternatively, $\frac{|C|}{OPT} \leq 2$)

♣

Remark 21

Under some plausible complexity assumption, 2-approximation is optimal for efferent algorithms.

Set Cover

Given a universe (i.e., set) U and a collection of m sets $S_i \subseteq U$, are there k indexes $\mathcal{I} \subseteq [k]$ that cover U (i.e., $\cup_{j \in \mathcal{I}} S_j = U$)?
Define

$$\text{SET-COVER} = \{\langle U, S_1, \ldots, S_m, k\rangle : \exists \mathcal{I} \subseteq [k] \text{ s.t. } \bigcup_{j \in \mathcal{I}} S_j = U\}$$

Note that

- SET-COVER $\in \mathcal{NP}$
- VertexCover $\leq_P$ SET-COVER

  $\Longrightarrow$ SET-COVER $\in \mathcal{NPC}$

Approximating Algorithm for Set Cover

Algorithm 22 (Appx)

Input: Sets $U, S_1, \ldots, S_m$.

1. Set $V_0 = U$ and $t = 0$.
2. While $V_t \neq \emptyset$:
   1. $t = t + 1$.
   2. Let $i_t = \arg\max_i |S_i \cap V_{t-1}|$
   3. $V_t = V_{t-1} \setminus S_{i_t}$
3. Output $C = \{i_1, \ldots, i_t\}$.

Claim 23

Appx is a $(\ln |U|)$-approximation for set cover – $\mathcal{I}$ is at most $\ln |U|$ as large as a minimum set cover.

Appx is a $(\log |U|)$-Approximation for Set Cover

Proof:

- Assume there are k subsets that cover U, and let $\mathcal{I}^*$ be their index set.
- In the t'th iteration of Appx, one of the subsets in $\mathcal{I}^*$ covers at least $|V_{t-1}|/k$ items in V_{t-1}.
- $|V_1| \leq (1 - \frac{1}{k}) \cdot |V_0|$.
- More generally, $|V_t| \leq (1 - \frac{1}{k}) \cdot |V_{t-1}| \leq \ldots \leq (1 - \frac{1}{k})^t \cdot |U|$.
- For $t \geq k \cdot \ln |U|$, it would hold that $|V_t| \leq (1 - \frac{1}{k})^{k \cdot \ln |U|} \cdot |U| \leq e^{-\ln |U|} \cdot |U| = 1$.

  $\Longrightarrow$ Appx terminates within t rounds.

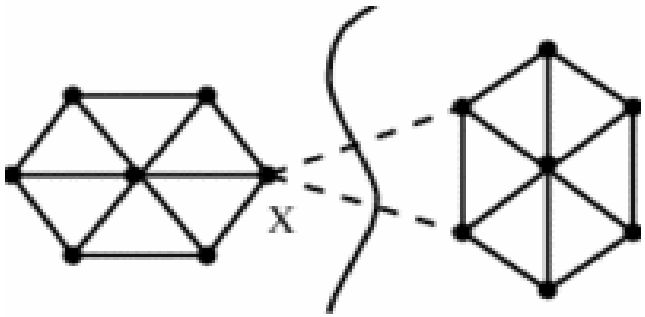
  $\Longrightarrow$ Appx approximation ratio $\ln |U|$.

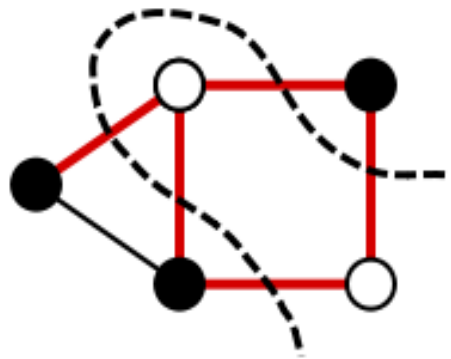
♣

Cuts in Graphs

Definition 24

$G = (V, E)$ be an undirected graph. For any partition of the nodes of into two sets, S and $V \setminus S$, the set of edges between S and $V \setminus S$ is called a cut .





left picture from http://www.cs.sunysb.edu/∼algorith/files/edge-vertex-connectivity.shtml, right from wikipedia

Cuts Optimization

For cuts, both optimization problems make sense (in different contexts):

1. MIN-CUT: Find a partition that minimizes the number of edges between S and $V \setminus S$.
2. MAX-CUT: Find a partition that maximizes the number of edges between S and $V \setminus S$.

The two optimization problems have very different complexities:

1. MIN-CUT is tightly related to *network flow*, and has polynomial time algorithms.
2. MAX-CUT is NP-hard.

Approximation Algorithm for MAX-CUT

Consider the following local improvement strategy

Algorithm 25 (Appx)

1. Pick any partition S and $V \setminus S$.
2. If the cut can be improved by moving any vertex from $V \setminus S$ to S, or vice-versa, do so.
3. Quit when no improvement is possible (local maximum reached).

Running time:

Any cut has at most $|E|$ edges.

$\Longrightarrow$ at most $|E|$ improvements are possible.

$\Longrightarrow$ Appx is polynomial time.

Claim 26

Appx is a 2-approximation algorithm for MAX-CUT.

Appx is a 2-Approximation for MAX-CUT

We use the following definitions with respect to a given cut C:

- C partitions the edges into cut edges E_C, and non-cut edges E_N.
- c_v — number of cut edges from node v.
- n_v — number of non-cut edges from v.

Proof: When Appx terminates:

- $c_v \geq n_v$ for every node v (?)
- $\Longrightarrow \sum_{v \in V} c_v \geq \sum_{v \in V} n_v$
- $\sum_{v \in V} c_v = 2|E_C|$ and $\sum_{v \in V} n_v = 2|E_N|$ (each edge is counted twice).

  $\Longrightarrow |E_C| \geq |E_N|$.

  $\Longrightarrow 2|E_C| \geq |E_N| + |E_C| = |E|$

  $\Longrightarrow |E_C| \geq |E|/2$.
- Since $|E| \geq OPT$, it holds that $|E_C| \geq OPT/2$

  Hence, Appx is 2-approximation (i.e., $\frac{OPT}{|E_C|} \leq 2$)

♣

Section 3

Coping with NP-Hardness – Heuristics

Heuristics

http://image.examiner.com/images/blog/wysiwyg/image/beast.jpg

Heuristics

Definition:

Main Entry heuristic
Pronunciation: hyu-'ris-tik
Function: adjective
Etymology: German heuristisch, from New Latin heuristicus, from Greek heuriskein – to discover;

Involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods (heuristic techniques; a heuristic assumption);
also: of or relating to exploratory problem-solving techniques that utilize self-educating techniques (as the evaluation of feedback) to improve performance (a heuristic computer program)

Heuristics

Widely used in almost every area with hard problems, e.g.,
http://dai.fmph.uniba.sk/~simko/satsolver/
http://www.boolsat.com/

Solutions are typically based on solid intuition, but their run-time analysis "in practice" is beyond current knowledge.

Examples: Simulated annealing, genetic (evolutionary) algorithms, neural networks.

When all else fails, a smart heuristic may do wonders.

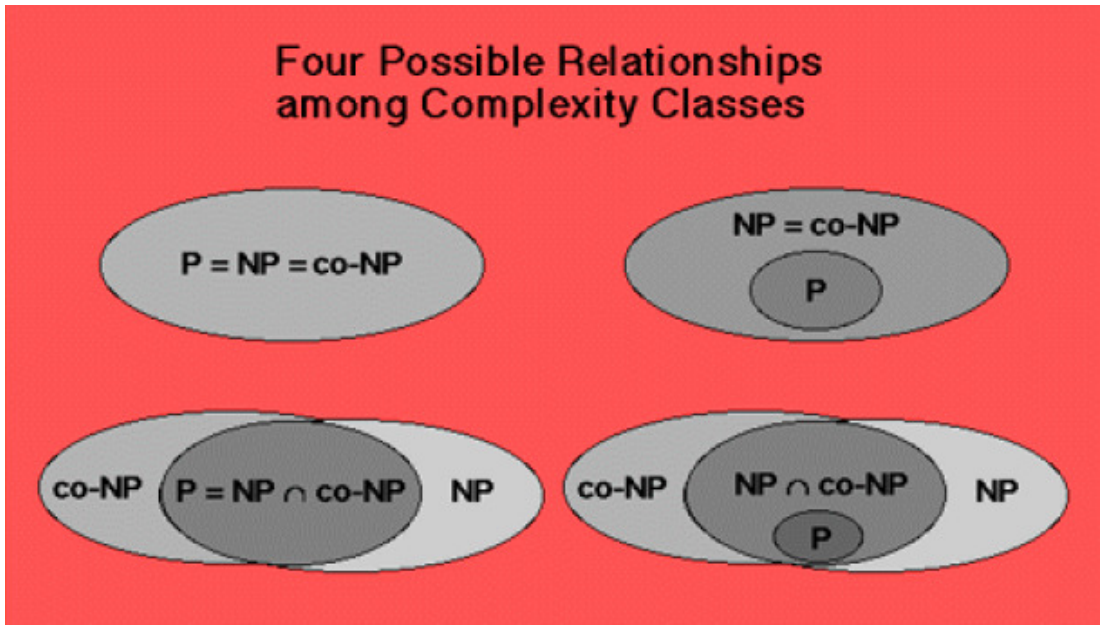
Part III

Concluding Remarks

Ladies and Gentlemen, Boys and Girls

We have just ended the third part of the course:

Introduction to Computational Complexity (no more).



And with it, naturally, we've ended the whole course.

We hope you have enjoyed your flight, and look forward to meeting you in the future (but not in Moed B :-).

The Dreaded Exam

- All material covered in class and recitations, from all parts of course.
- Both multiple choice ("closed") and "open" questions.
- You can bring and use two double sided A4 (normal size) pages marked with your name
- Piece of cake.