

Computational Models - Lecture 13¹

Handout Mode

Iftach Haitner and Yishay Mansour.

Tel Aviv University.

June 18/20, 2012

¹Based 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)

$t: \mathbb{N} \mapsto \mathbb{N}$ with $t(n) \geq n \log n$ is **time constructible**, if the mapping from 1^n to the **binary** representation of $t(n)$ is computable in time $O(t(n))$.

Theorem 2 (time hierarchy theorem)

For any time constructible function $t: \mathbb{N} \mapsto \mathbb{N}$, exists a 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: $\mathcal{NP} := \bigcup_{c \geq 0} \text{NTIME}(n^c) \subseteq \bigcup_{c \geq 0} \text{DTIME}(2^{n^c}) \subseteq \text{DTIME}(2^{n^{\frac{1}{2} \cdot \log n}}) \subsetneq \text{DTIME}(2^{n^{\log n}})$.

L and Its $O(t(n))$ -Time Decider

For a fixed constructible function $t: \mathbb{N} \mapsto \mathbb{N}$, consider the following $L \in \text{DTIME}(O(t(n)))$:

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 $|\langle M \rangle| > \log t(n)$.
4. Emulate M on w :
 - ▶ **Reject** if M makes more than Max steps (use the counter).
 - ▶ If M accepts, **Reject**; If M rejects, **Accept**.

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

Analyzing D's Running Time

Algorithm 6 (D)

On input w :

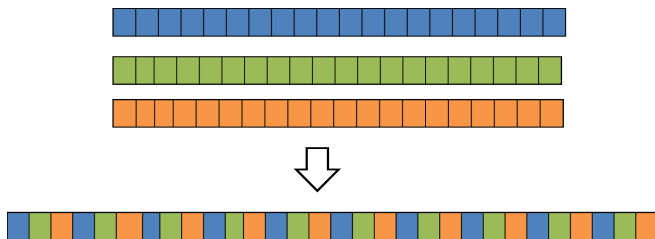
1. Let $n = |w|$.
2. Store the value $Max = \lceil t(n) / \log t(n) \rceil$ in a binary counter.
3. **Reject** if w is **not** of the form $\langle M \rangle 10^*$ or $|\langle M \rangle| > \log t(n)$.
4. Emulate M on w :
 - ▶ **Reject** if M makes more than Max steps (use the counter).
 - ▶ If M accepts,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

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 $O(t(n) / \log t(n))$ 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 **$\log t(n)$ (why?)**, 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 decider of running time $g(n) \in o(t(n)/\log t(n))$.
- $\exists n_0 \in \mathbb{N}$ such that $g(|w|) \leq t(|w|)/\log t(|w|)$ and $|\langle M \rangle| \leq \log t(n)$, where $w = \langle M \rangle 10^{n_0}$.
- $D(w)$ emulates $M(w)$ to its **completion**.
- Hence $D(w) \neq M(w)$, $\implies M$ 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).

Example 9

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

Definition 10 (coNP-hard languages)

A language B is **coNP-hard**, if it satisfies

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

We do **not** require $B \in \text{co-}\mathcal{NP}$ (membership).

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

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 $x \in \Sigma^*$,

$$T(x) = y \implies (x, y) \in T$$

$$T(x) = \perp \implies \nexists y \in \Sigma^* \text{ such that } (x, y) \in T.$$

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

NP-Hard Problems, 2

Definition 12 (Karp reductions)

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

Definition 13 (NP-hard problems)

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

Note that any NP-hard problem, is also coNP-hard.

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 that “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:** Solving the problem $T = R_L := (x, y) : x \in L \wedge y$ is a certificate for x — given input x , **find** some y satisfying $R_L(x, y)$, or declare that none exist.
The search problem seems **harder to solve** than the decision problem.
- Turns out that for **NP complete** languages, search and decision have the “same difficulty” — Karp reduced to each other.

Decision to Search – SAT

Let D be a **decider** for **SAT**. The following algorithm is a **searcher** for **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: a satisfied formula ϕ of variables $x_1 \dots, x_\ell$.

If $D(\phi_{x_1=1}) = 1$, return $(1, \text{Searcher}(\phi_{x_1=1}))$;

Otherwise, return $(0, \text{Searcher}(\phi_{x_1=0}))$

Decision to Search – CLIQUE

Let D be a **decider** for **CLIQUE**. The following algorithm is a **searcher** for **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 16 (Searcher)

Input: an integer k , and graph G containing a clique of size k , with vertices v_1, \dots, v_ℓ .

If $D(G_{-v_1}, k) = 1$, return $\text{Searcher}(G_{-\{v_1\}}, k)$;

Otherwise, return $\{v_1\} \cup \text{Searcher}(G_{-N(v_1)}, k - 1)$.

Remark 17

Such reductions (known as **downward self reductions**), exist for all NP-Complete languages (**why?**), but seemingly not to all of NP.

Search Vs. Optimization Problems

- **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 satisfying $R(x, y)$, that is the **largest** among all solutions (or **smallest**), or declare that none exist.
- Example 1: Given a graph G , find a vertex cover of **smallest** size possible.
- Example 2: 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 three or four 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 $\frac{\text{cost}(\text{approx})}{\text{cost}(\text{opt})}$
- For maximization problem $\frac{\text{cost}(\text{opt})}{\text{cost}(\text{approx})}$

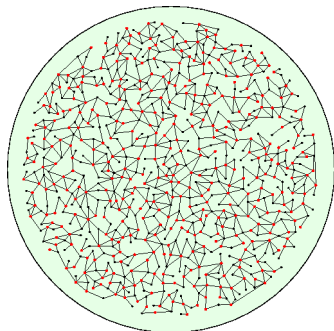
Fully Polynomial Approximation: $(1 + \epsilon)$ -approximation, where running time depends on ϵ .

Vertex Cover

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

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

Note that $\text{VertexCover} \in \mathcal{NPC}$ 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 18 (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 19

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) contain) **at least** one node from each $e \in E_C$ (otherwise an edge would not be covered).
 $\implies OPT(G) \geq |C|/2$ (or $\frac{|C|}{OPT} \leq 2$)



Remark 20

Under some plausible complexity assumption, this factor 2-approximation **cannot** be improved.

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

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

Note that

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

Approximating Algorithm for Set Cover

Algorithm 21 (Appx)

Input: Sets U, S_1, \dots, S_m .

- 1 Set $V_0 = U$ and $t = 1$.
- 2 While $V_t \neq \emptyset$
 - 1 Let $i_t = \arg \max_j |S_j \cap V_t|$
 - 2 $V_{t+1} = V_t \setminus S_{i_t}$
 - 3 $t = t + 1$
- 3 Output $C = \{i_1, \dots, i_{t-1}\}$.

Claim 22

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

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

Proof:

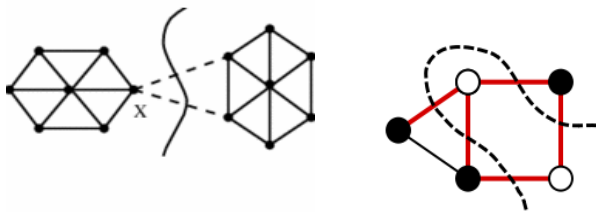
- Assume there are k subsets that cover U , and let \mathcal{I}^* be their index set.
- In the t 'th iteration of Appx, one of the subsets in \mathcal{I}^* covers at least $|V_t|/k$ items in V_t .
- Hence, $|V_t| \leq (1 - \frac{1}{k}) \cdot |V_{t-1}| \leq (1 - \frac{1}{k})^{t-1} \cdot |U|$.
- For $t > k \cdot \log |U|$ it holds that $|V_t| < 1$, and hence Appx terminates.
 \implies Appx approximation ratio $\log |U|$.



Cuts in Graphs

Definition 23

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



left pic 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 24 (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
 - ⇒ **at most** $|E|$ improvements are possible
 - ⇒ **Appx** is polynomial time.

Claim 25

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

Cut Edges and Non-Cut Edges

We use the following definitions with respect to a fixed cut C ,

- The cut partitions the edges into **cut edges** E_C , and **non-cut** edges E_N .
- Let c_v be the number of **cut edges** from node v .
- Let n_v be the number of **non-cut edges** from v .

Appx is a 2-Approximation for MAX-CUT

Proof: When Appx terminates:

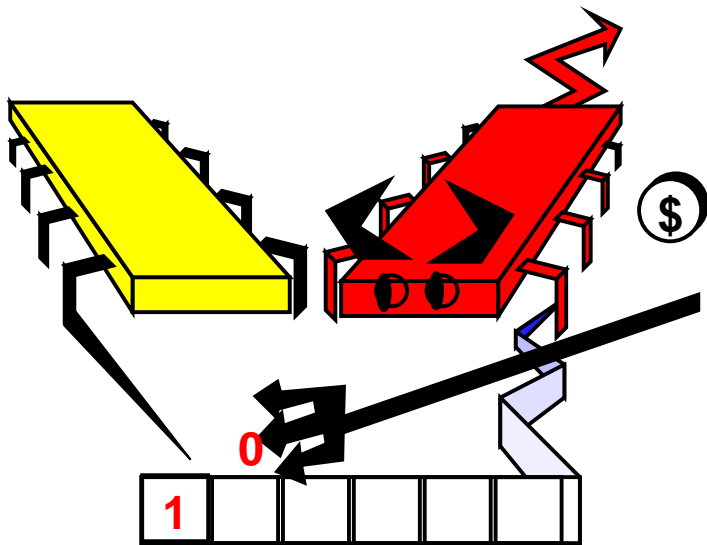
- $c_v \geq n_v$ for every node v (why?)
- $\implies \sum_{v \in V} c_v \geq \sum_{v \in V} n_v$
- $\sum_{v \in V} c_v = 2|E_C|$ and $\sum_{v \in V} n_v = 2|E_N|$ (each edge is counted twice).
- $\implies |E_C| \geq |E_N|$.
- $\implies 2|E_C| \geq |E_N| + |E_C| = |E|$
- $\implies |E_C| \geq |E|/2$.
- 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 – Randomization

Randomized TMs



Randomized Computation

We can sometimes use **randomization** to solve problems that are difficult to solve **deterministically**.

Examples:

- MAX-CUT
- MAX-3SAT
- Determining if a multivariate polynomial is identically zero
- Primality testing (not really **needed** anymore – known to be in \mathcal{P} by the AKS primality test of 2002).

Randomized Algorithm for MAX-CUT

- We give a randomized 2-approximation for MAX-CUT
- **Strategy:** Each node v with probability $\frac{1}{2}$ is in S .
- **Analysis:** For each edge (u, v) with probability $\frac{1}{2}$ is in the cut.
- Expected size of cut is $|E|/2$.
- Clearly, in OPT we cut at most $|E|$ edges.

Randomized Algorithm for MAX-3SAT

- We give a randomized $8/7$ -approximation for MAX-3SAT
- **Strategy:** Each variable x_i is with probability $\frac{1}{2}$ set $x_i = 1$ (and otherwise set $x_i = 0$).
- **Analysis:** Each clause C is satisfied with probability $\frac{7}{8}$.
- Expected number of satisfied clauses is at least $\frac{7}{8}m$, where m is the number of clauses.
- Clearly, in *OPT* we satisfy at most m clauses.

Polynomial Identity

- We are given a multivariate polynomial $\pi(x_1, \dots, x_m)$ and wish to know if $\pi(x_1, \dots, x_m)$ is **identically zero**.
- **Naive strategy**: Fully expand $\pi(x_1, \dots, x_m)$, and check if all resulting coefficients are 0.
- This strategy may not be very efficient.
For example, consider

$$\pi(x_1, \dots, x_m) = (x_1 - y_1)(x_2 - y_2) \dots (x_n - y_n)$$

Randomized Algorithm for Polynomial Identity

Lemma 26 (Schwartz' Lemma)

Let $\pi(x_1, \dots, x_m)$ be a **not identically zero** polynomial of degree (in each variable) at most d , and let $T \in \mathbb{N}$

Then the number of m -tuples $\bar{x} = (x_1, \dots, x_m) \in \{0, 1, \dots, T\}^m$ satisfying $\pi(\bar{x}) = 0$, is at most $md \cdot T^{m-1}$.

Corollary 27

For such π it holds that $\Pr_{\bar{x} \leftarrow \{0, 1, \dots, T\}^m} [\pi(\bar{x}) = 0] \leq \frac{md}{T}$.

For $T \in \mathbb{N}$, define the following **one-sided error** randomized algorithm

Algorithm 28 (Appx_T)

Input polynomial π as above.

- 1 Pick $\bar{x} \leftarrow \{0, 1, \dots, T\}^m$
- 2 If $\pi(\bar{x}) = 0$ return **TRUE**; otherwise return **FALSE**

If π is not identically zero, $\Pr[\text{Appx}_T(\pi) = \text{TRUE}] < md/T$

Section 4

Coping with NP-Hardness – Fixed Parameter

Fixed Parameter

- **Main idea:** Many NP-complete problems have a natural parameter r .
- Examples: **clique size** (CLIQUE), **number of variables** (SAT), **size of cover** (VertexCover), **magnitude of integers** (SUBSET-SUM).
- The hardness depends on the parameter r .
- Get an algorithm which runs in $O(f(r) \cdot \text{poly}(n))$, “caring not” about f .
- Simple example SAT:
 - ▶ SAT has a natural parameter k – the number of variables
 - ▶ Easy to show $O(2^k n)$ -time solution, where n is the formula size.

Fixed Parameter Algorithm for Vertex Cover

The parameter: k — the size of the optimal cover.
Fixed parameter algorithm runs in time $O(2^k \cdot |E|)$.

Algorithm 29

Input: Graph $K = (V, E)$ and integer k :

- 1 Start with an empty cover C
- 2 While there is an uncovered edge $(u, v) \in E$.
Search (via recursion) for a vertex-cover of size $(k - 1)$ in the graphs G_{-u} and G_{-v} .

Time analysis:

- The recursion depth is at most k .
- At least one of the recursive paths is successful.

Section 5

Coping with NP-Hardness – Heuristics

Heuristics



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

Heuristics

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

Heuristics are widely used in almost every area with hard optimization problems. They 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.

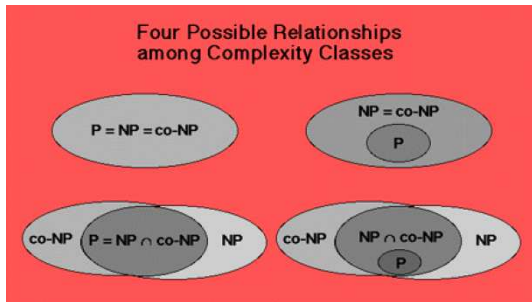
Part III

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 three 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.

