Computational Models Spring 11: Lecture 14

- Decision, search, and optimization problems
- Coping with NP completeness: Search
- Coping with NP completeness: Approximation
- Coping with hard computational tasks: Randomization
- Coping with NP completeness: Fix Parameter Algorithms
- Coping with hard computational tasks: Heuristics
- Concluding Remarks

- Sipser, 10.1, 10.2

NP Hardness

A language $\mathcal{B}$ is NP-hard if it satisfies

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

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

Example: The language

$$\{\langle \Phi_1, \Phi_2 \rangle \mid \Phi_1 \in SAT, \Phi_2 \notin SAT\}$$

is NP-hard but apparently not NP-complete (why?).

coNP Hardness

A language $\mathcal{B}$ is coNP-hard if it satisfies

- Every $\mathcal{A}$ in coNP is polynomial time reducible to $\mathcal{B}$

We do not require $\mathcal{B} \in coNP$ (membership).

Example: The language

$$\{\langle \Phi_1, \Phi_2 \rangle \mid \Phi_1 \in SAT, \Phi_2 \notin SAT\}$$

is coNP-hard but apparently not coNP-complete (why?).

#3SAT

The #3SAT is the problem: given a 3CNF formula Φ, and a number k, is the number of satisfying assignment of Φ is k.

- #3SAT is NP-hard
- #3SAT is coNP-Hard
- In fact it is likely to be much harder.

Decision, Search, Optimization Problems

Let $R(\cdot, \cdot)$ be a poly time computable predicate.

- Decision Problem: Given input x, decide if there is some y satisfying $R(x, y)$?
- Using the "certificate" characterization of languages in NP, the decision problem is the same as deciding membership $x \in L$ for $L \in NP$.
- Search Problem: Given input x, find some y satisfying $R(x, y)$, or declare that none exist.
- The search problem seems harder to solve than the decision problem.

Decision, Search, Optimization Problems

- Search Problem: Given input x, find some y satisfying $R(x, y)$, or declare that none exist.
- Turns out that for NP complete languages, search and decision have the "same difficulty".
- Specifically, given access to an oracle for L (the decision problem), we can solve the search problem in poly time.
- When oracle is successively accessed with queries of decreasing sizes, this technique is known as self reduction.
- Examples: SAT and Clique (on board).
- This is applicable to NPC problems but not to all of NP.

Decision, Search, 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-Completeness

- Search Intelligent exhaustive search
- 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.

Intelligent Exhaustive Search: Backtracking

Example SAT

- Need to enumerate all assignment, in the worse case
- can direct the search,
- Helps many times.

Search Algorithm:

- Fix a variable ($x_i = 0$ or $x_i = 1$)
- decide which variable to fix next (an important decision)
- Terminate a recursive call if reaches a contradiction
- Example 2CNF and 3CNF (on board)

Intelligent Exhaustive Search: Branch and Bound

General Methodology:

- Compute the cost of the partial solution
- compute a lower bound on the remaining solution
- If the sum is more than the threshold, terminate this branch
- If the sum is less than the threshold, extend the partial solution.

Intelligent Exhaustive Search: Branch and Bound

Example TRAVELING-SALESMAN:

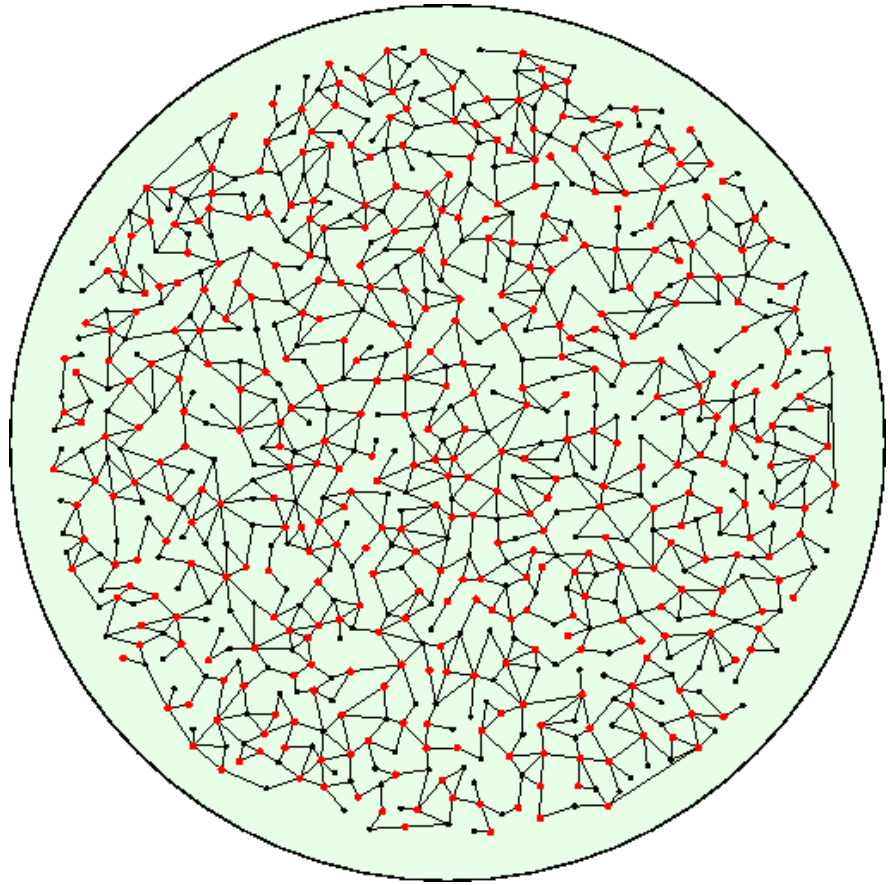
- Given a path P from s to v.
- Compute a cost of the path.
- Lower bound the cost of a path from v to s using only $V \setminus P$.
- Lower bound (example) suing the sum of:
  (1) minimal weight edge from v to $V \setminus P$,
  (2) minimal weight edge from $V \setminus P$ to s, and
  (3) MST in $V \setminus P$.

Approximation Algorithms

- Give a guarantee on the solution quality
- Approximation ratio:
- For minimization problem, $\frac{cost(approx)}{cost(opt)}$
- For maximization problem, $\frac{cost(opt)}{cost(approx)}$
- Fully Polynomial Approximation: $(1+\epsilon)$-approximation. Running time depends on ϵ.

Vertex Cover

The decision version of this problem is NP-complete by a reduction from Independent Set (proved in recitation).



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

Approx. Algorithm for VC (Gavril '74)

- $C := \emptyset$
- while there are edges in G
  - choose any edge (u, v) in G
  - add u and v to C
  - remove them from G

- **Claim:** This algorithm is a 2-approximation algorithm for vertex cover.
- Meaning C is at most twice as large as a minimum vertex cover.

Gavril's Approximation Algorithm

Claim: This is a 2-approximation algorithm.

- Cover C constructed from $|C|/2$ edges of G
- no two edges of these share a vertex
- any vertex cover, including the optimum, contains at least one node from each of these edges (otherwise an edge would not be covered).
- It follows that $OPT(G) \geq |C|/2$ (so $|C|/OPT \leq 2$) ♠

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

Set Cover

- Input: A universe U, and a collection of m sets $S_i \subseteq U$.
- Problem: Are there k indexes $\mathcal{I} \subseteq \{1, \ldots, k\}$ which cover U, i.e., $\cup_{j \in \mathcal{I}} S_j = U$.

Define

SET-COVER $=$
$\left\{ \langle S_1, \ldots, S_m, k \rangle | \exists I \subseteq \{1, \ldots, k\} \text{ s.t. } \cup_{j \in I} S_j = U \right\}$

- SET-COVER $\in$ NP
- VertexCover $\leq_P$ SET-COVER
- SET-COVER is NP-complete.

Set Cover: Greedy Approximation

Greedy Algorithm:

- $t = 0$; $V_t = U$
- While $V_t \neq \emptyset$
- Let $i_t = \arg\max_i |S_i \cap V_t|$
- $V_{t+1} = V_t \setminus S_{i_t}$
- $t = t + 1$
- END WHILE

OUTPUT: $C = \{i_1, \ldots, i_{t-1}\}$.

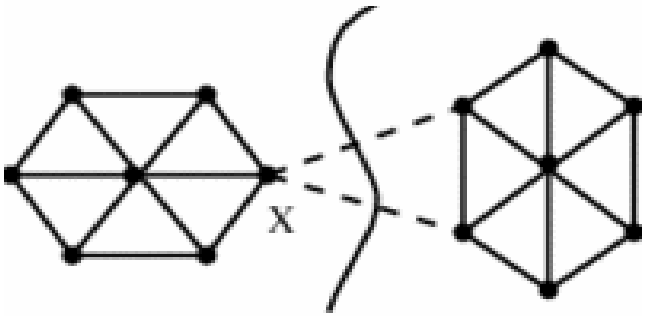
Set Cover: Greedy Approximation

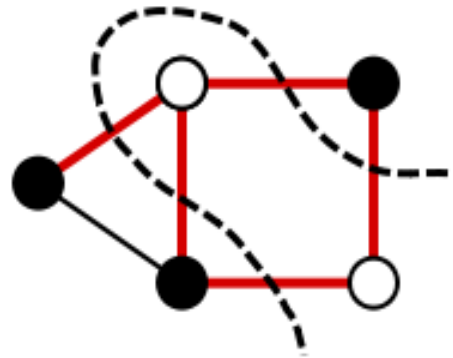
Analysis

- Assume there are k subsets that cover U.
- Each iteration, one of the OPT subsets covers at least $|V_t|/k$ items.
- Let $m_t = |V_t|$.
- Number of remaining items:
  $m_t \leq m_{t-1}(1 - \frac{1}{k}) = |U|(1 - \frac{1}{k})^t$.
- For $t \geq k \log |U|$, we have $m_t < 1$, and hence we terminate.
- Approximation ratio $\log |U|$.

Cuts in Graphs

Definition Let $G = (V, E)$ 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 $V - S$ is called a cut .





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

Cuts in Graphs

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 $V - S$.
2. Max Cut: Find a partition that maximizes the number of edges between S and $V - S$.

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-complete.

Max Cut Algorithm

Consider the following local improvement strategy

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

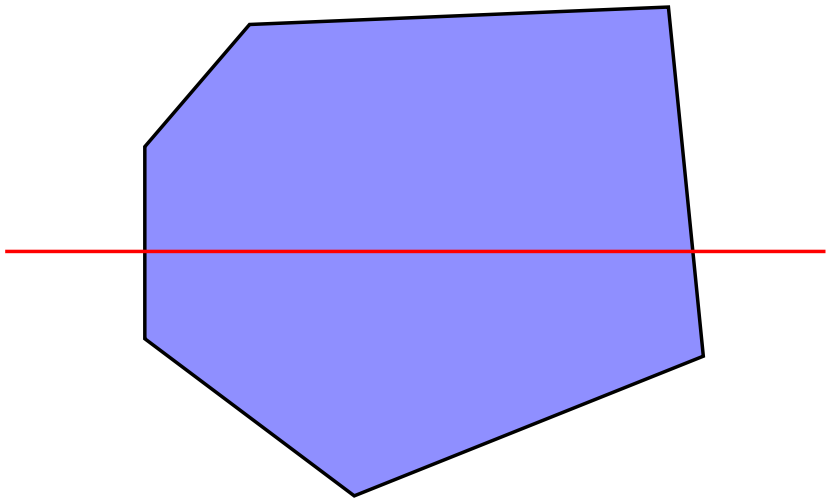
Running time

- Any cut has at most $|E|$ edges,
- thus at most $|E|$ improvements possible,

$\Longrightarrow$ algorithm is polynomial time.

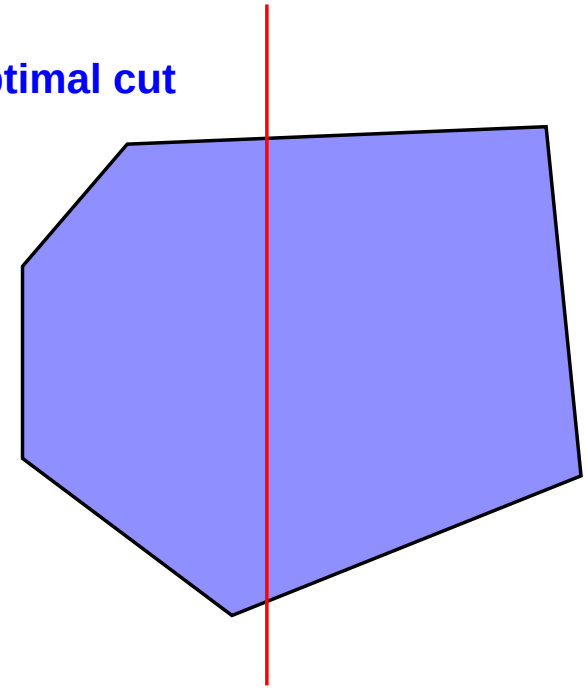
Claim: This is a 2-approximation algorithm!

Max Cut Algorithm

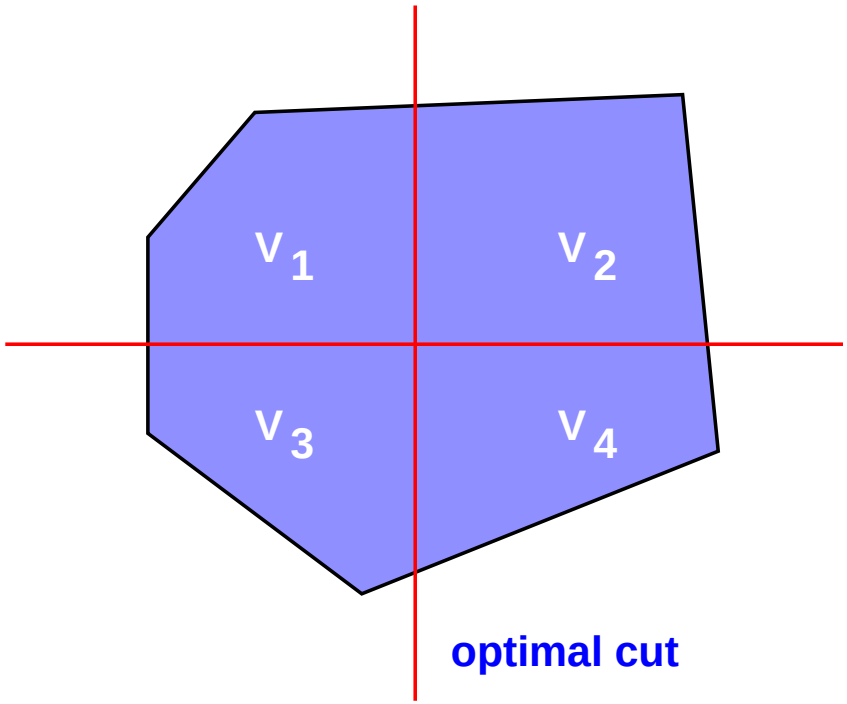


heuristic cut

optimal cut

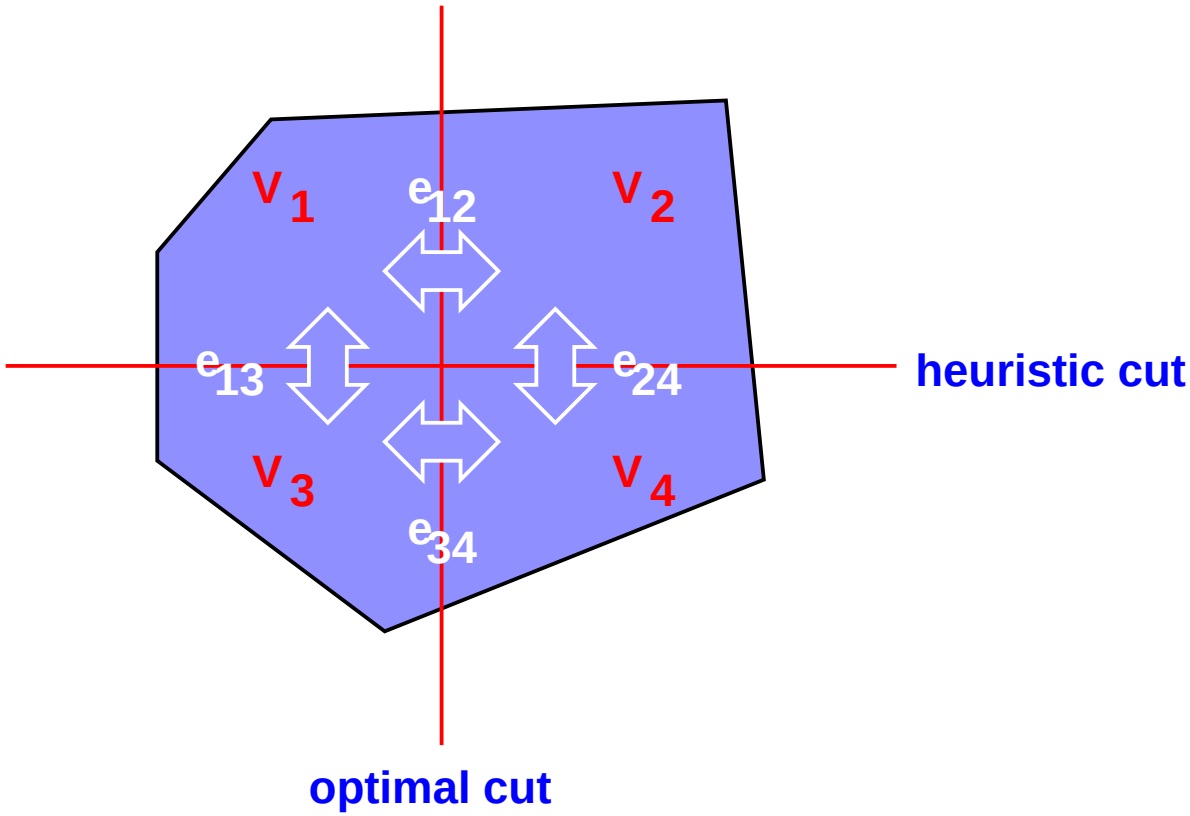


Max Cut Algorithm



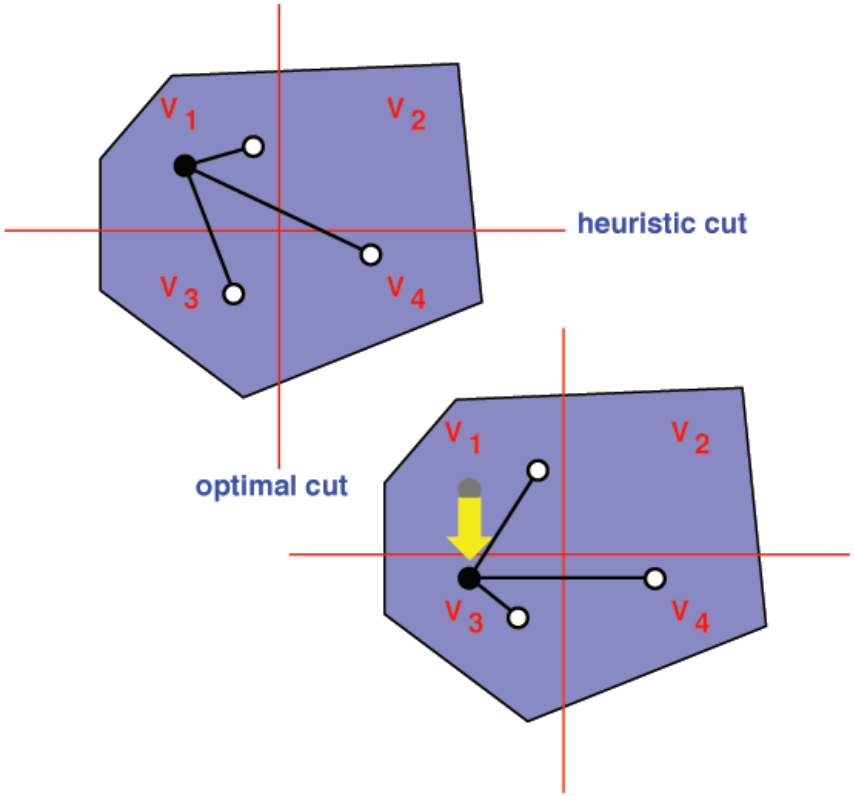
- Heuristic yields $V_1 \cup V_2$, $V_3 \cup V_4$
- Optimal yields $V_1 \cup V_3$, $V_2 \cup V_4$

Max Cut Algorithm



- Every 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.

Max Cut Algorithm

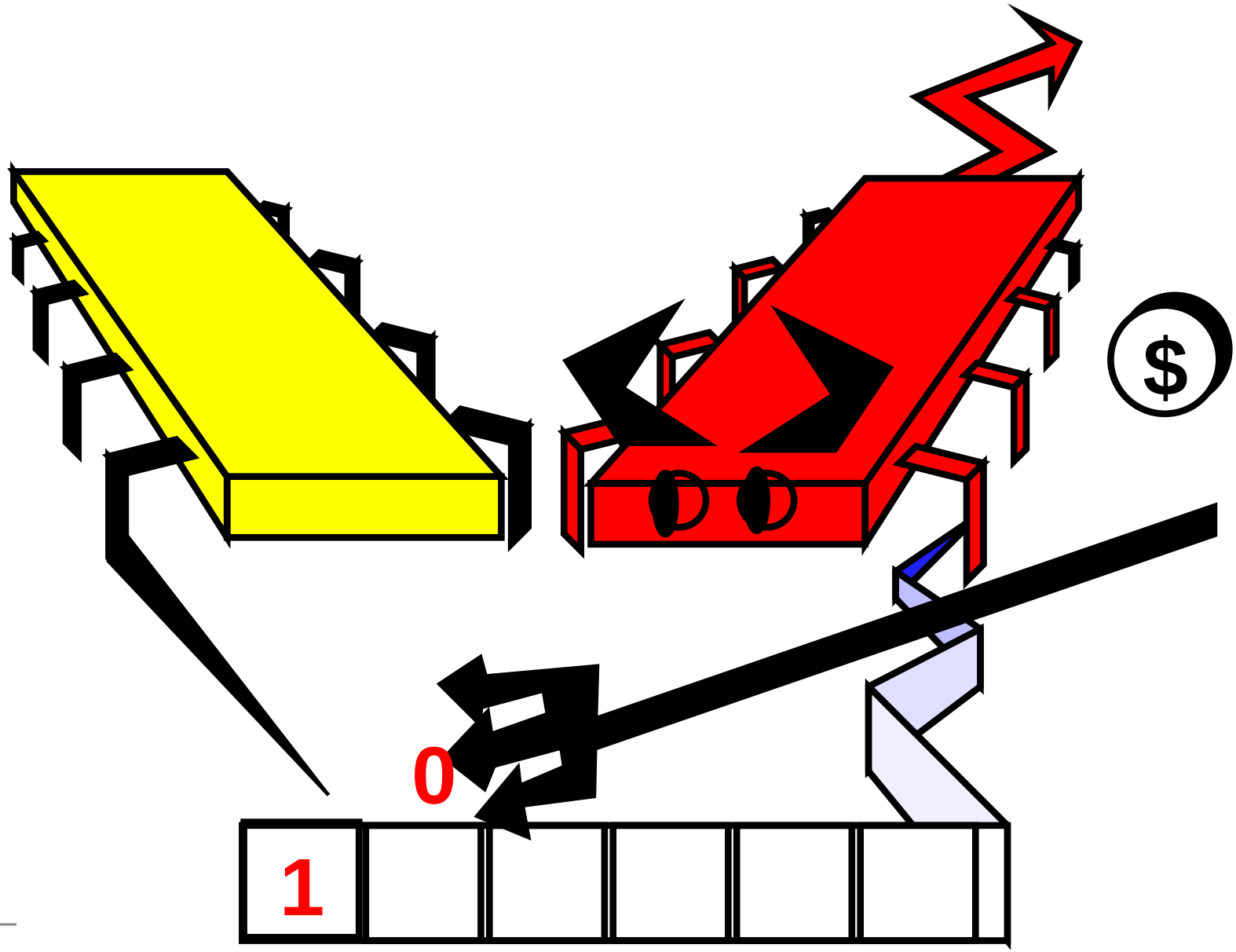


- When algorithm terminates, for every node v, the number of cut edges is greater or equal than the number of non-cut edges, $c_v \geq n_v$.
- Otherwise, switching the node v would increase the size of the cut produced by the algorithm.
- Summing over all nodes in V: $\sum_v c_v \geq \sum_v n_v$

Max Cut Algorithm

- Summing over all nodes in V: $\sum_v c_v \geq \sum_v n_v$.
- But $\sum_v c_v = 2|E_C|$, $\sum_v n_v = 2|E_N|$
  (each edge is counted twice).
- Thus $|E_C| \geq |E_N|$.
- $\Longrightarrow 2|E_C| \geq |E_N| + |E_C| = |E|$
- So $|E_C| \geq |E|/2$.
- Clearly $|E| \geq OPT$ (any cut is set of edges).
- Thus $c(\mathcal{A}) \geq OPT/2$, *i.e.* algorithm is 2-MaxCut approximation ($c(A)/OPT \geq 1 - 1/2$) . ♣

Randomized (Coin Flipping) TMs



Randomized Computation

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

Examples:

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

MAX-CUT: Randomized Computation

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.

#3SAT: Randomized Computation

We give a randomized $8/7$-approximation for #3SAT

strategy: Each variable x_i is with probability $\frac{1}{2}$ set $x_i = 1$ (and otherwise set $x_i = 0$).

Analysis: For each clause C with probability $\frac{7}{8}$ it is satisfied.

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.

Randomized Computation

We are given a multivariant polynomial $\pi(x_1, \ldots, x_m)$.
We wish to know if $\pi(x_1, \ldots, x_m)$ is identically zero.

One strategy: Fully expand $\pi(x_1, \ldots, x_m)$, and check if all resulting coefficients are zero.

This strategy may not be very efficient.
For example, consider
$\pi(x_1, \ldots, x_m) = (x_1 - y_1)(x_2 - y_2) \ldots (x_n - y_n)$

Randomization to the Rescue

Schwartz' Lemma

Let $\pi(x_1, \ldots, x_m)$ be a polynomial

- not identically zero
- in m variables
- of degree at most d in each variable
- and let $M > 0$ be an integer.

Then

- the number of m-tuples $(x_1, \ldots, x_m) \in \{0, 1, \ldots, M\}$
- satisfying $\pi(x_1, \ldots, x_m) = 0$
- is at most mdM^{m-1}

Randomization to the Rescue

Using Schwartz' Lemma, choose $M > 0$
large enough so that $mdM^{m-1}/M^m = md/M < \varepsilon$.

Pick $(x_1, \ldots, x_m) \in \{0, 1, \ldots, M\}$ at random.

Then, if π is not identically zero, $Pr(\pi(x_1, \ldots, x_m) = 0) < \varepsilon$

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 $O(f(r)poly(n))$.

Simple example SAT

- SAT has a natural parameter k which is the number of variables
- Easy to show $O(2^k n)$

Fixed Parameter: Vertex Cover

The parameter: k, the size of the optimal cover.

Fixed Parameter Algorithm: runs in $O(2^k|E|)$.

- Main Observation: For each edge (u, v) either u or v are in the cover.
- We need to consider only covers of size at most k.

Fixed Parameter: Vertex Cover

Algorithm

- Start with an empty cover
- while there is an uncovered edge (u, v).
- Search for a VertexCover of size $k-1$ in the graph G_u, where we delete u.
- Search for a VertexCover of size $k-1$ in the graph G_v, where we delete v

Analysis

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

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

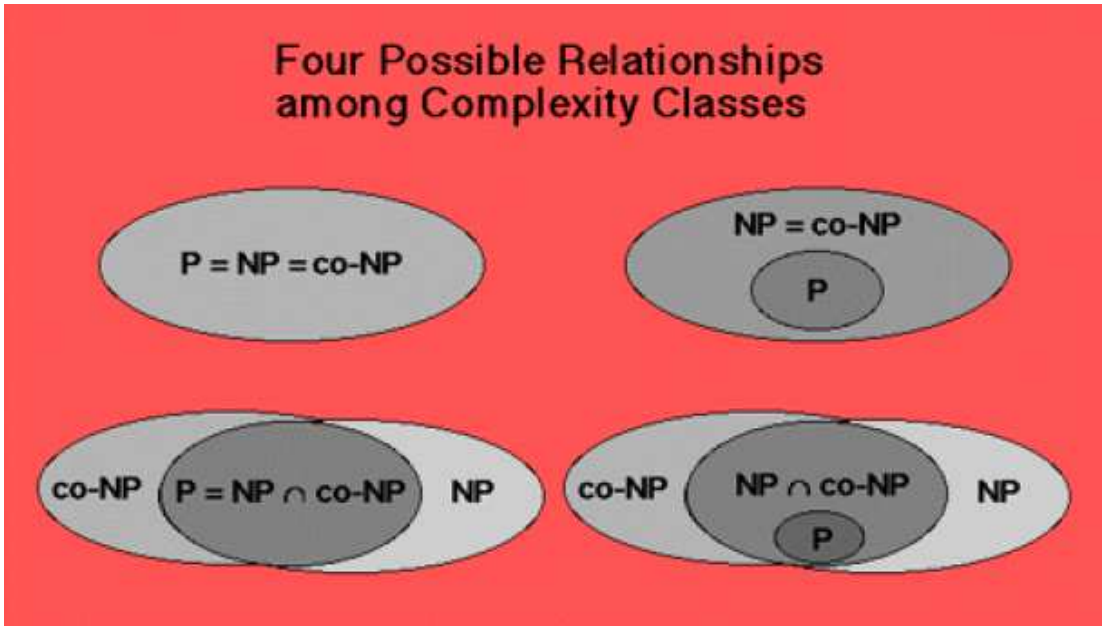
Heuristics are widely used in almost every area with hard optimization problems. They are typically based on solid intuition, but their run-time analysis "in practice" is beyond current knowledge.
Examples: Simulated annealing, genetic (evolutionary) algorithms, neural networks.

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

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.
- Only material taught in both classes.
- 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.
- Seker Horaa