

Computational Models Spring 09: Lecture 14

- $3SAT \leq_P$ SUBSET-SUM
- Decision, search, and **optimization** problems
- Coping with NP completeness: Approximation
- Coping with hard computational tasks: Randomization
- Coping with hard computational tasks: Heuristics
- Concluding Remarks

- Sipser, 7.5, 10.1, 10.2

Reminder: SUBSET-SUM

An instance of the problem

- A collection of numbers in binary/decimal, r_1, \dots, r_k
- Target number t
- Question: does some subcollection add up to t ?

$$\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S = \{r_1, \dots, r_k\} \\ \exists \{s_1, \dots, s_\ell\} \subseteq \{r_1, \dots, r_k\}, \sum_{y_j} = t \}$$

Collections are sets: repetitions not allowed.

Subset Sum

Theorem: SUBSET-SUM is NP-complete.

Must demonstrate

- **SUBSET-SUM** $\in NP$
- every $A \in NP$ poly-time reducible to **SUBSET-SUM**.

Note that

- we have already seen the first (the subset is the certificate),
- we will now show that **3SAT** \leq_P **SUBSET-SUM**.

Subset Sum

For every 3-CNF formula, ϕ , we generate a SUBSET-SUM instance:

- Construct very large numbers in decimal notation (binary would work as well, but not unary).
- For each variable x_i of the formula ϕ , the set S contains a pair of numbers y_i, z_i .
- To “hit” the target, t , we will have to choose exactly one of these y_i, z_i :
Take y_i if x_i is assigned **T**, take z_i if **F**.

Subset Sum

For every 3-CNF formula, ϕ , we generate a SUBSET-SUM instance:

- Construct very large numbers in decimal notation (binary would work as well, but not unary).
- For each clause C_j of the formula ϕ , the set S contains a pair of equal numbers g_j, h_j .
- If ϕ has ℓ variables x_1, \dots, x_ℓ and k clauses C_1, \dots, C_k , all numbers y_i, z_i, g_j, h_j are $\ell + k$ digits long.
- Clauses will also correspond to distinct positions in the decimal representations of these numbers.

Subset Sum

Let ϕ be a boolean formula with

- variables x_1, \dots, x_ℓ
- clauses C_1, \dots, C_k .

We construct a table whose rows are $2\ell + 2k$ numbers in decimal notation.

Subset Sum

Rows of table are labeled with

- variable encodings: $y_1, z_1, y_2, z_2, \dots, y_\ell, z_\ell,$
- clause encodings: $g_1, h_1, g_2, h_2, \dots, g_k, h_k,$
- goal t

Subset Sum

Each x_i encoded as y_i, z_i .

Decimal representation has two parts.

- left-hand part: digit i is 1, rest 0
- right-hand part: one digit for each clause
 - j^{th} digit of y_i is 1 if C_j contains x_i
 - j^{th} digit of z_i is 1 if C_j contains $\overline{x_i}$

Subset Sum Mapping: Example

$$(x_1 \vee \overline{x_2} \vee \dots) \wedge \dots \wedge (\overline{x_1} \vee \overline{x_2} \vee \dots)$$

	1	2	...	ℓ	C_1	...	C_k
y_1	1	0	...	0	1	...	0
z_1	1	0	...	0	0	...	1
y_2	0	1	...	0	0	...	0
z_2	0	1	...	0	1	...	1

Subset Sum

S also contains g_j and h_j for each clause C_j

- they are equal
- digit $\ell + j$ is 1, rest are 0.

	1	...	ℓ	C_1	C_2	...	C_k
\vdots							
g_1	0	...	0	1	0	...	0
h_1	0	...	0	1	0	...	0
g_2	0	...	0	0	1	...	0
h_2	0	...	0	0	1	...	0

Subset Sum

Bottom line of table is goal t

- first ℓ digits are 1
- last k digits are 3 (three!)

	1	...	ℓ	C_1	C_2	...	C_k
\vdots							
t	1	...	1	3	3	...	3

Subset Sum

	1	2	...	ℓ	C_1	C_2	...	C_k
y_1	1	0	...	0	1	0	...	0
z_1	1	0	...	0	0	0	...	1
y_2	0	1	...	0	0	0	...	0
z_2	0	1	...	0	1	0	...	1
\vdots								
g_1	0	0	...	0	1	0	...	0
h_1	0	0	...	0	1	0	...	0
g_2	0	0	...	0	0	1	...	0
h_2	0	0	...	0	0	1	...	0
\vdots								
t	1	1	...	1	3	3	...	3

Subset Sum

Claim: If ϕ is satisfiable, then some subset of S sums to t .

- select y_i if x_i is **true**.
- select z_i if x_i is **false**.

Adding them up

- yields a **1** in the **first** ℓ digits
- matching t so far

Subset Sum

Claim: If ϕ is satisfiable, then some subset of S sums to t .

- select y_i if x_i is **true**.
- select z_i if x_i is **false**.

Each of the **last** k digits

- **at least one** literal is **true**,
- number of true literals is between **1** and **3**,
- sum so far: at least **1**, at most **3**.
- so pick enough h_i, g_i to bring this digit up to 3.
- this can be done **separately** for each clause.

This completes one direction.

Subset Sum

Claim: If some subset of S sums to t , then ϕ is satisfiable.

- All digits in S are 0 or 1.
- Because ϕ is in **3CNF**, each column contains at most **five 1s** (why?).
- Therefore, summation causes **no carries** between columns.
- Subset must contain one of y_i or z_i , but not both.

(As an aside, how would you cope if encoding were **binary** rather than decimal?)

Subset Sum

Here is the satisfying assignment.

- if subset includes y_i , then x_i is **true**
- if subset includes z_i , then x_i is **false**

This assignment must satisfy ϕ because

- each of final k columns sums to 3.
- at most 2 come from g_i, h_i
- so at least one must come from **true literal**

Final point: transformation takes $O(n^2)$ stages.



Wait a Minute

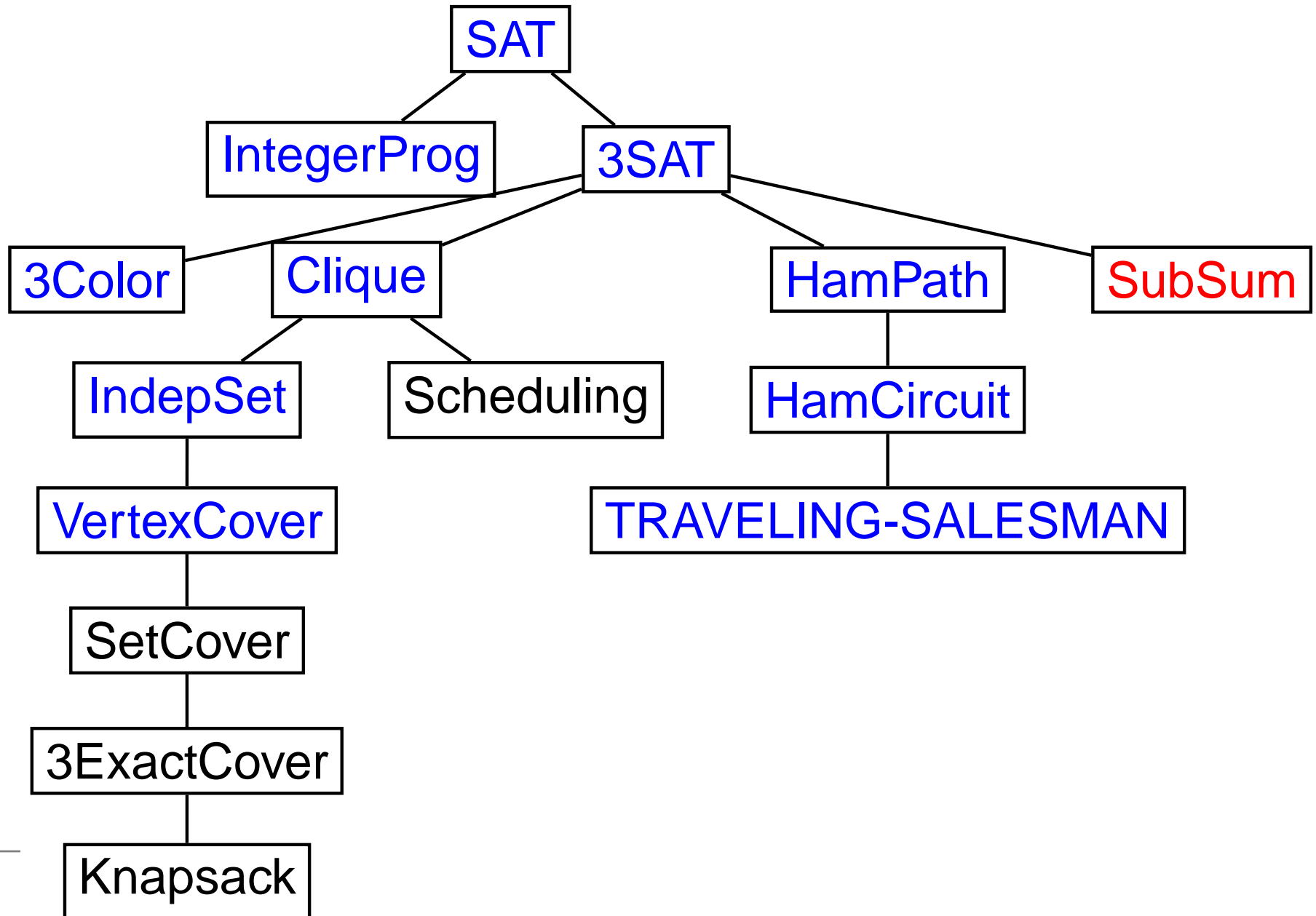
Those of you taking the algorithms course just saw a **polynomial time**, dynamic programming algorithm for **Subset Sum**.

- So either the teacher of the algorithms course made some embarrassing error.
- Or the teacher of this course did.
- Or both :-)
- Or **P=NP** but nobody bothered to tell you :-))

The “Paradox”

- To resolve this “paradox”, look at the **fine print**.
- The algorithm is **polynomial time** in **what**?
- What do you mean? **Polynomial time** in the input length, of course!
- But the inputs are integers. How are they encoded?
- In the NP completeness proof, they are encoded in **binary/decimal** (for a ϕ with 100 clauses and 50 variables, numbers will be about 10^{150}).
- In the algorithm, numbers are encoded in **unary**. So the algorithm can handle numbers around 150 comfortably, but nothing like 10^{150} !

Chains of Reductions: NPC Problems



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 $\mathcal{B} \in \text{coNP}$ (membership).

Example: The language

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

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

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 $R(x, y)$?

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 $L \in NP$.

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.

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.

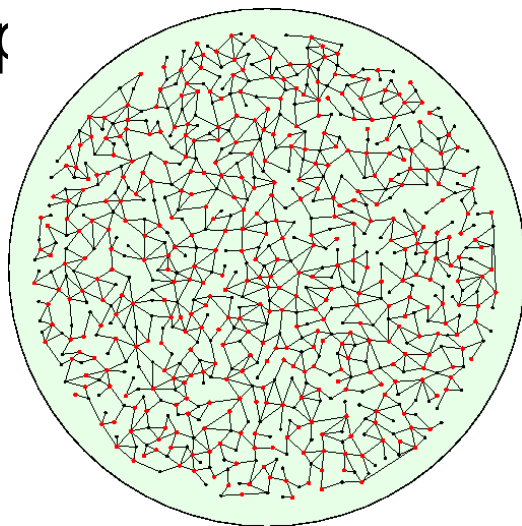
Decision, Search, Optimization Problems

- **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 legal coloring that **minimizes** number of colors used.
- **Example 2:** Given a graph G , find a clique of **largest** size possible.

Coping with NP-Completeness

- **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 advanced courses.



(figure from <http://www.brauer.in.tum.de/gruppen/theorie/hard/vc1.png>)

Example: Vertex Cover

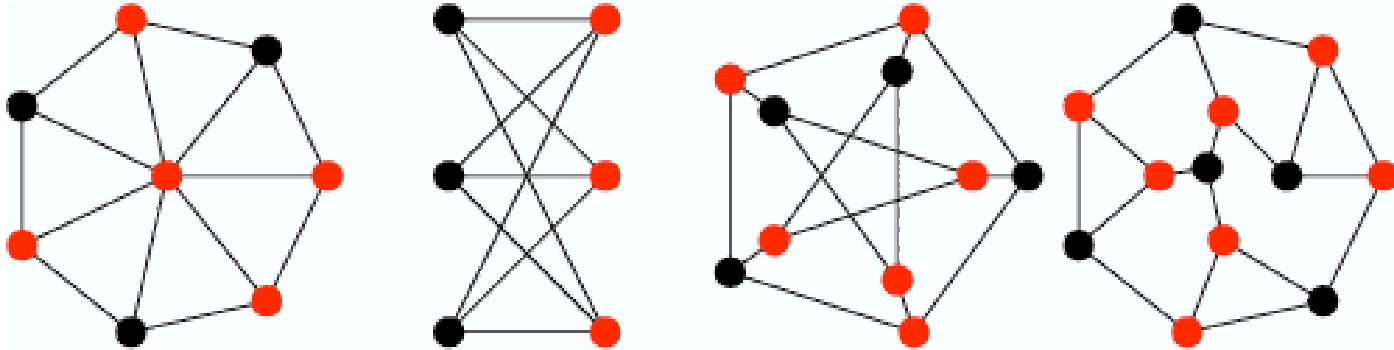
Given a graph (V, E)

- find the **smallest** set of vertices C
- such that for each edge in the graph,
- C contains at least one endpoint.
- Consider a **star** first, then the following examples:

Example: Vertex Cover

Given a graph (V, E)

- find the **smallest** set of vertices C
- such that for each edge in the graph,
- C contains at least one endpoint.
- Consider a **star** first, then the following examples:



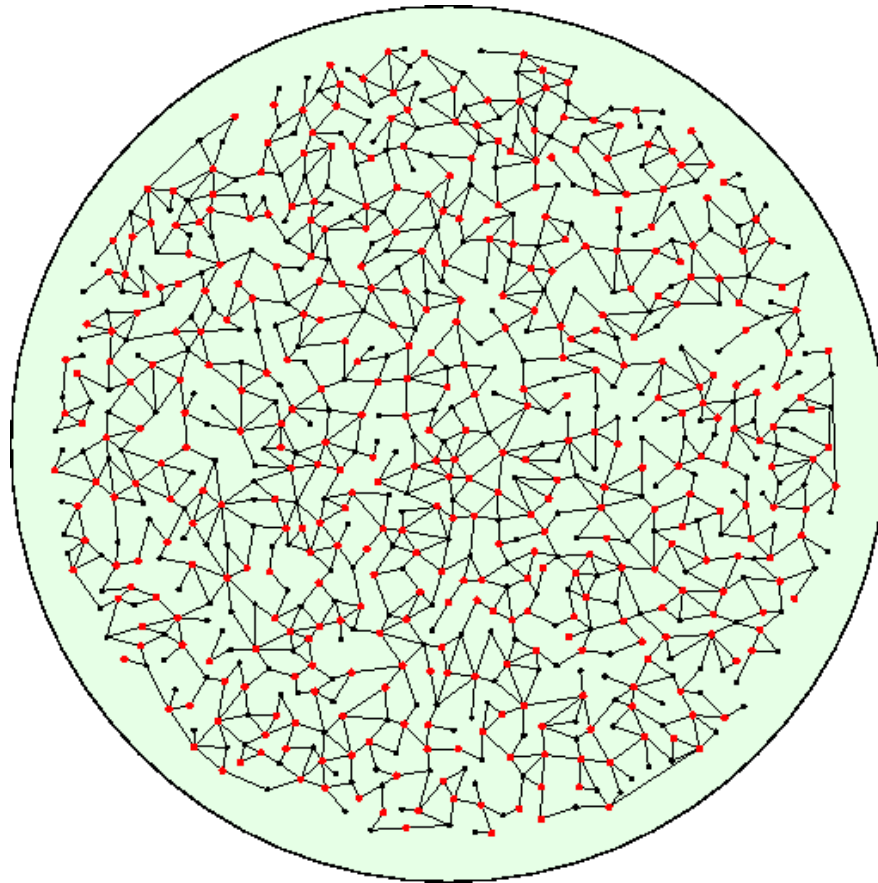
(figure from www.cc.ioc.ee/jus/gtglossary/assets/vertex_cover.gif)

Vertex Cover

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

Vertex Cover

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



(figure from <http://www.brauer.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

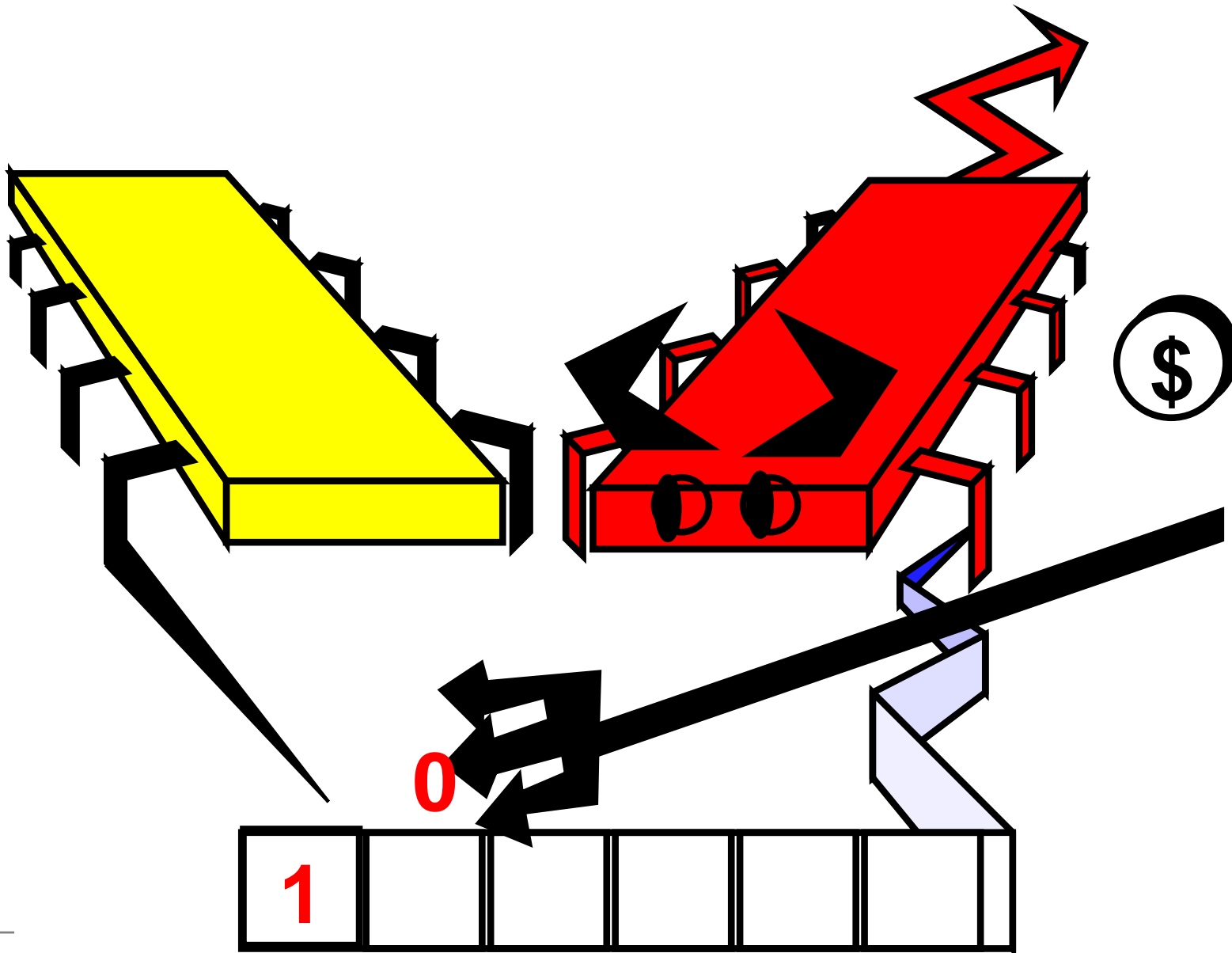
Claim: This is a 2-approximation algorithm.

- 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.

Randomized (Coin Flipping) TMs



Randomized Computation

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

Examples:

- 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).

Randomized Computation

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

Randomized Computation

We are given a multivariate polynomial $\pi(x_1, \dots, x_m)$.

We wish to know if $\pi(x_1, \dots, x_m)$ is **identically zero**.

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

Randomized Computation

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

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

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 Computation

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

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

This strategy may not be very efficient.
For example, consider

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

A second example deals with the determinant of a **symbolic** matrix (a matrix containing both constants and variables' symbols).

Randomized Computation

Recall the **determinant** of a matrix:

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

$$\det A = \sum_{\pi} \sigma(\pi) \prod_{i=1}^n a_{i,\pi(i)}$$

Where

- π ranges over all permutations
- σ is 1 if π is product of even number of transpositions
- σ is -1 otherwise

Randomized Computation

Can compute determinants by **Gaussian Elimination**

$$\begin{pmatrix} 1 & 3 & 2 & 5 \\ 1 & 7 & -2 & 4 \\ -1 & -3 & -2 & 2 \\ 0 & 1 & 6 & 2 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 3 & 2 & 5 \\ 0 & 4 & -4 & -1 \\ 0 & 0 & 0 & 7 \\ 0 & 1 & 6 & 2 \end{pmatrix}$$

Gaussian Elimination

- subtract multiples of first row from $2, \dots, n$
- to make first column element zero
- continue with subsequent rows ...

$$\begin{pmatrix} 1 & 3 & 2 & 5 \\ 0 & 4 & -4 & -1 \\ 0 & 0 & 0 & 7 \\ 0 & 1 & 6 & 2 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 3 & 2 & 5 \\ 0 & 4 & -4 & -1 \\ 0 & 0 & 0 & 7 \\ 0 & 0 & 6 & 2\frac{1}{4} \end{pmatrix}$$

Randomized Computation

If **pivot** element is zero:

$$\begin{pmatrix} 1 & 3 & 2 & 5 \\ 0 & 4 & -4 & -1 \\ 0 & 0 & 0 & 7 \\ 0 & 0 & 6 & 2\frac{1}{4} \end{pmatrix}$$

Randomized Computation

Then

- transpose rows
- multiply determinant by -1
- if this is impossible, determinant is zero.

$$\begin{pmatrix} 1 & 3 & 2 & 5 \\ 0 & 4 & -4 & -1 \\ 0 & 0 & 6 & 2\frac{1}{4} \\ 0 & 0 & 0 & 7 \end{pmatrix}$$

Randomized Computation

At the end, matrix has **upper triangular form**:

$$\begin{pmatrix} 1 & 3 & 2 & 5 \\ 0 & 4 & -4 & -1 \\ 0 & 0 & 6 & 2\frac{1}{4} \\ 0 & 0 & 0 & 7 \end{pmatrix}$$

Determinant is

- product of diagonal elements: 196
- adjusted for row interchanges: -196

Randomized Computation

Claim: We can calculate the determinant of an $n \times n$ matrix in polynomial time.

Must check

- if entries are integers of b bits,
- result not exponentially long in b

This isn't difficult.

Randomized Computation

What about determinants of **symbolic** matrices?

Randomized Computation

What about determinants of **symbolic** matrices?

$$\begin{aligned}
 \det \begin{pmatrix} x & w & z \\ z & x & w \\ y & z & w \end{pmatrix} &\implies \det \begin{pmatrix} x & w & z \\ 0 & \frac{x^2 - zw}{x} & \frac{wx - z^2}{x} \\ 0 & \frac{zx - wy}{x} & -\frac{zy}{x} \end{pmatrix} \implies \\
 \det \begin{pmatrix} x & w & z \\ 0 & \frac{x^2 - zw}{x} & \frac{wx - z^2}{x} \\ 0 & 0 & -\frac{yz(xz - xw) + (zx - wy)(wx - x^2)}{x(x^2 - xw)} \end{pmatrix}
 \end{aligned}$$

Randomized Computation

What about determinants of **symbolic** matrices?

$$\begin{aligned} \det \begin{pmatrix} x & w & z \\ z & x & w \\ y & z & w \end{pmatrix} &\implies \det \begin{pmatrix} x & w & z \\ 0 & \frac{x^2-zw}{x} & \frac{wx-z^2}{x} \\ 0 & \frac{zx-wy}{x} & -\frac{zy}{x} \end{pmatrix} \implies \\ \det \begin{pmatrix} x & w & z \\ 0 & \frac{x^2-zw}{x} & \frac{wx-z^2}{x} \\ 0 & 0 & -\frac{yz(xz-xw)+(zx-wy)(wx-x^2)}{x(x^2-xw)} \end{pmatrix} \end{aligned}$$

Is there a problem here?

Randomized Computation

We do have a problem.

Applying Gaussian elimination to symbolic matrices works, but

- intermediate terms are rational functions
- it can be shown that their size become exponentially large!

bad news!

Randomization to the Rescue

Schwartz' Lemma

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

Randomization to the Rescue

Schwartz' Lemma

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

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

Randomization to the Rescue

Schwartz' Lemma

Let $\pi(x_1, \dots, x_m)$ 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 $(x_1, \dots, x_m) \in \{0, 1, \dots, M\}$
- satisfying $\pi(x_1, \dots, x_m) = 0$
- is at most mdM^{m-1}

Randomization to the Rescue

Using Schwartz' Lemma, choose $M > 0$

large enough so that $mdM^{m-1}/M^m = md/M < \varepsilon$.

Randomization to the Rescue

Using Schwartz' Lemma, choose $M > 0$

large enough so that $mdM^{m-1}/M^m = md/M < \varepsilon$.

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

Randomization to the Rescue

Using Schwartz' Lemma, choose $M > 0$

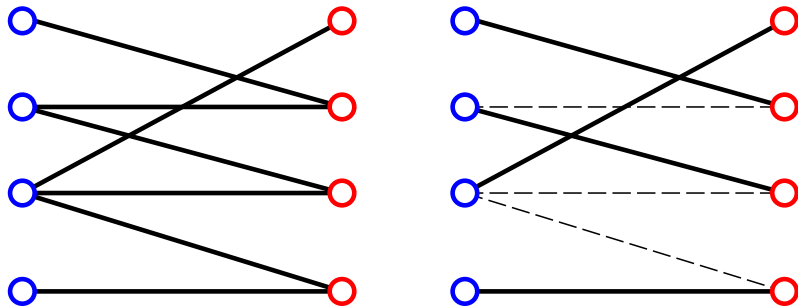
large enough so that $mdM^{m-1}/M^m = md/M < \varepsilon$.

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

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

Bipartite Matching

Surprisingly enough, this approach can be applied to find **perfect matching** in bipartite graphs.



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**;

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.

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.

Heuristics

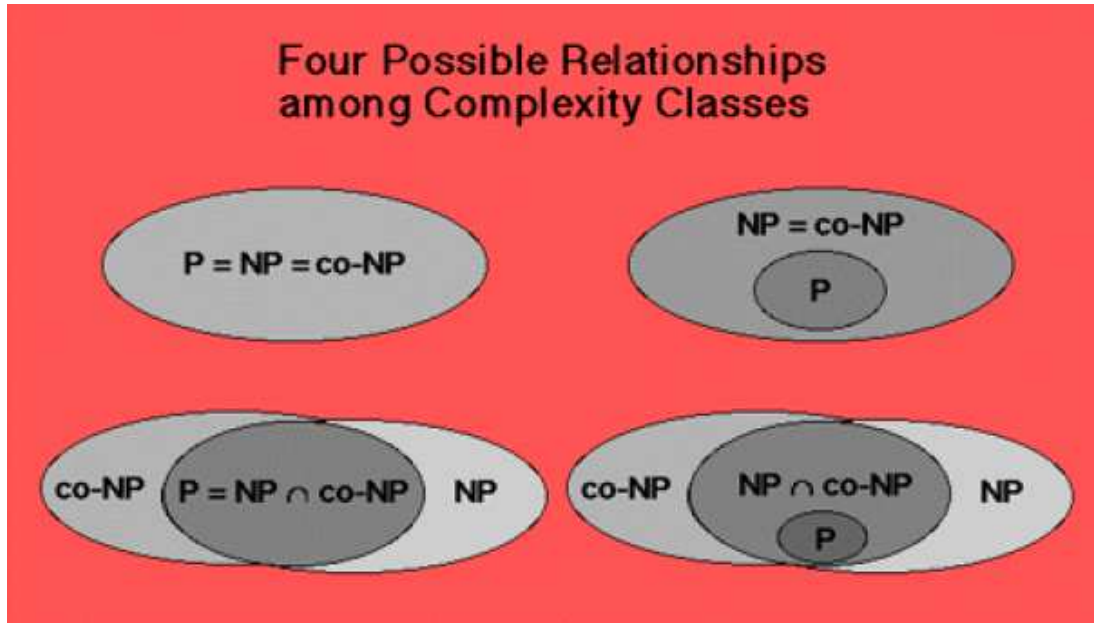
Heuristics are widely used in almost every area with hard optimization problems. They are typically based on solid **intuition**, but their run-time analysis "in practice" is beyond current knowledge.

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**.
- A list with specific topics will shortly be published on course site.
- Both multiple choice ("closed") and "open" questions.
- You can bring and use **two** double sided A4 (normal size) pages.
- Piece of cake.



Some Self Propaganda

- Next year, I'm teaching the "intro to cryptography" course (semester A).
- An interesting, but not too easy course (IMHO).
- I'll be happy to see **many of you** there.
- I'll also be teaching the computational models course (semester B).
- But hope **not** to see **any** of you there.

Famous Last Words

You have brains in your head.

You have feet in your shoes.

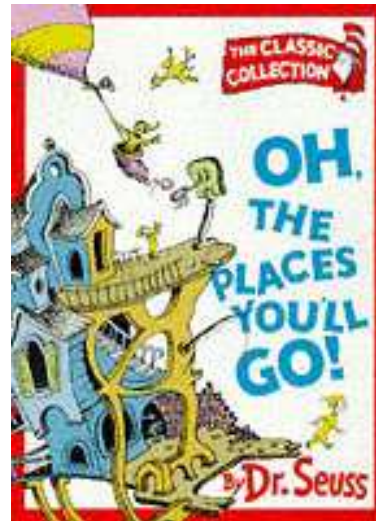
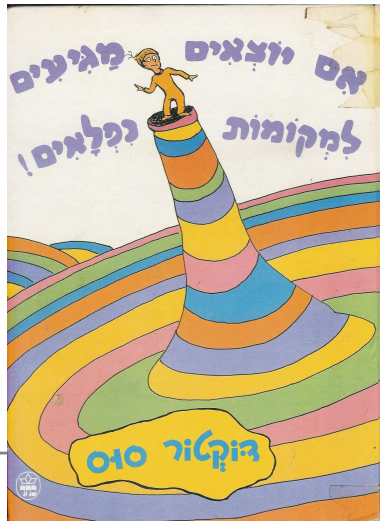
You can steer yourself
any direction you choose.

You're on your own. And you know what you know.

And *YOU* are the guy who'll decide where to go.

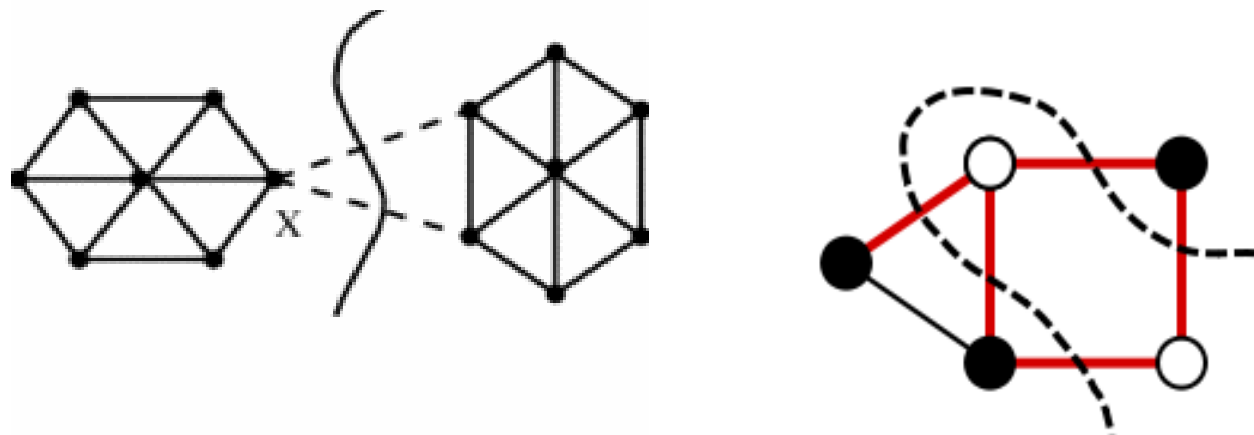
By Theodor Seuss Geisel (aka Dr. Seuss, 1904–1991).

Hebrew translation by Leah Naor.



Back to Approximations: 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/~algorithm/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 $V - S$.

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).

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,
- ⇒ algorithm is polynomial time.

Max Cut Algorithm

Consider the following **local improvement** strategy

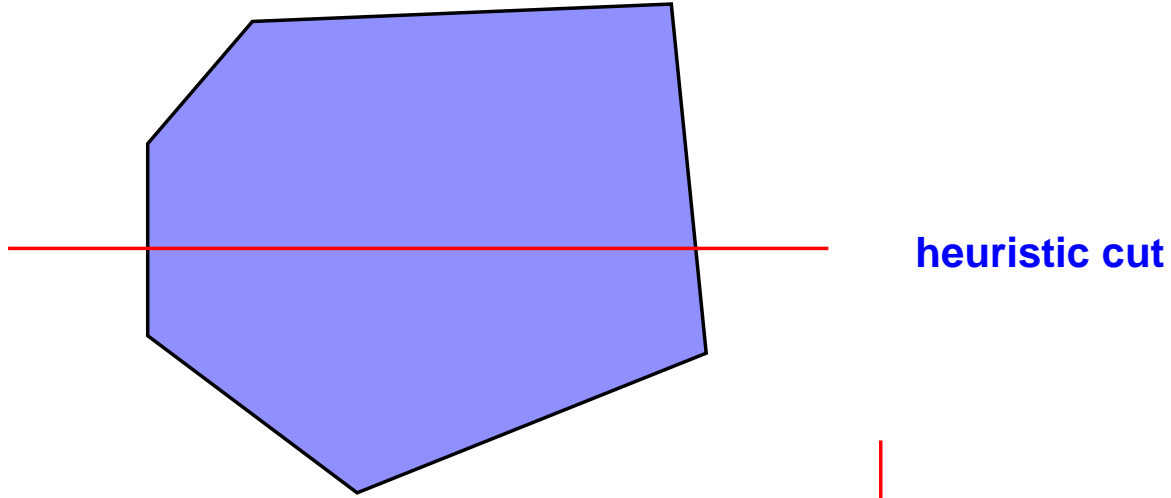
- Pick any partition S and $V - S$
- If the cut can be improved by moving any vertex from $V - S$ to S , or vice-versa, do so.
- Quit when no improvement is possible (**local** maximum reached).

Running time

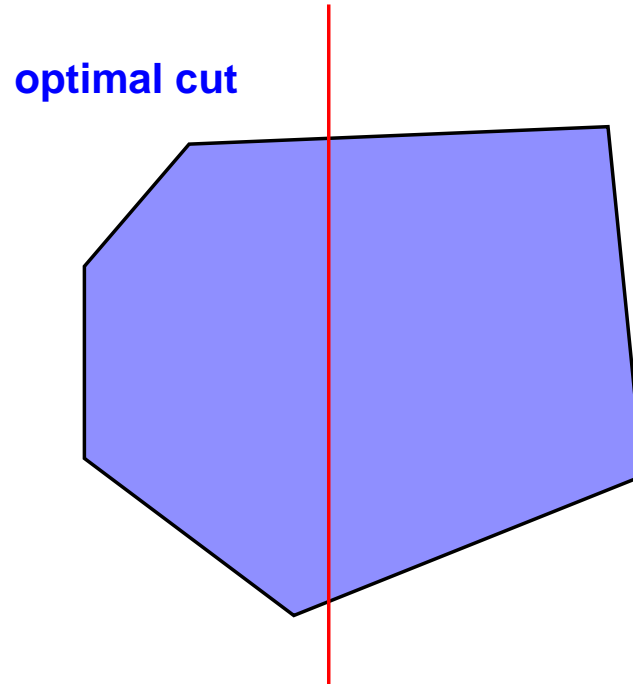
- Any cut has at most $|E|$ edges,
 - thus at most $|E|$ improvements possible,
- ⇒ algorithm is polynomial time.

Claim: This is a $\frac{1}{2}$ -approximation algorithm!

Max Cut Algorithm

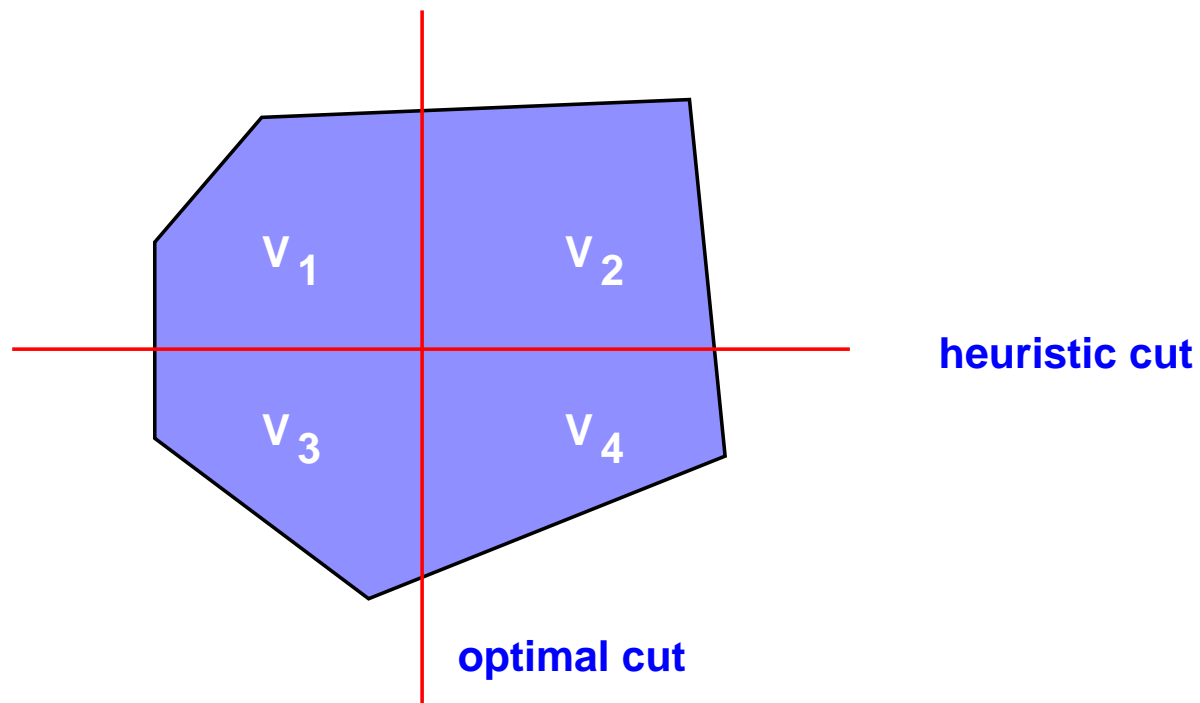


heuristic cut



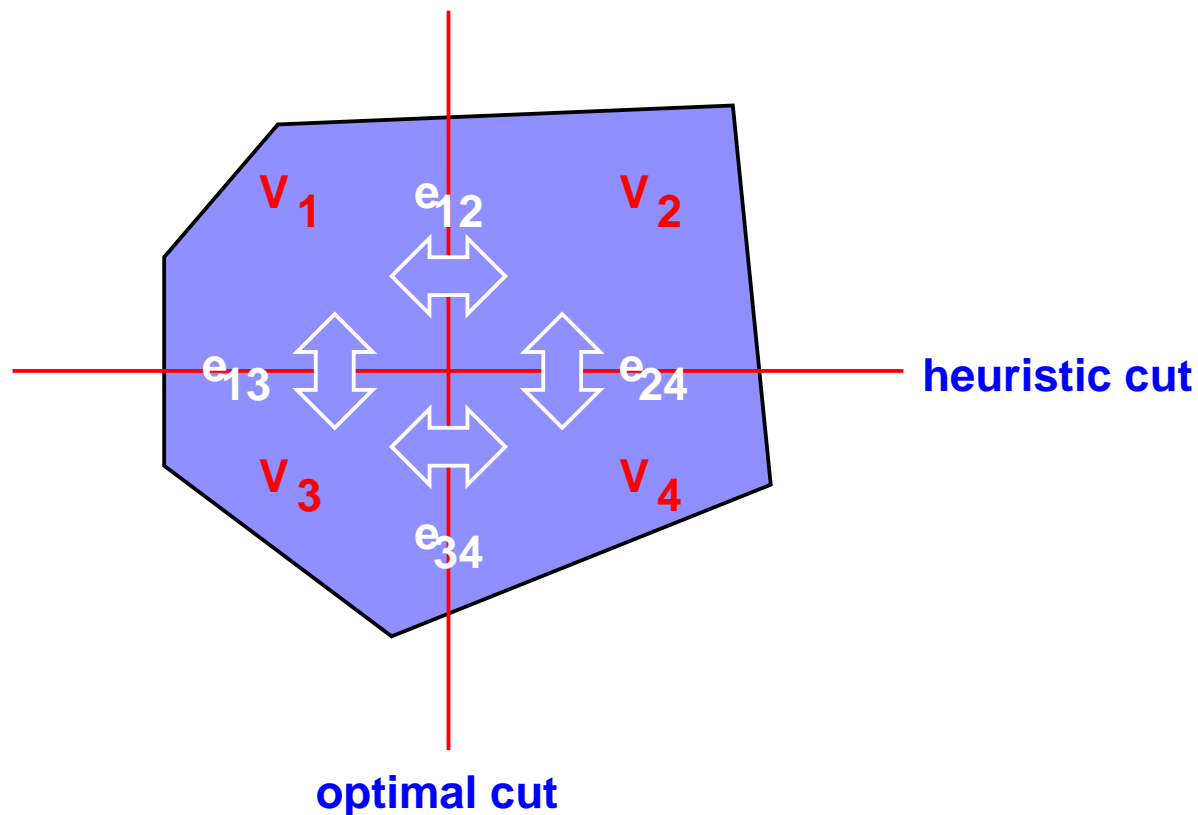
optimal cut

Max Cut Algorithm



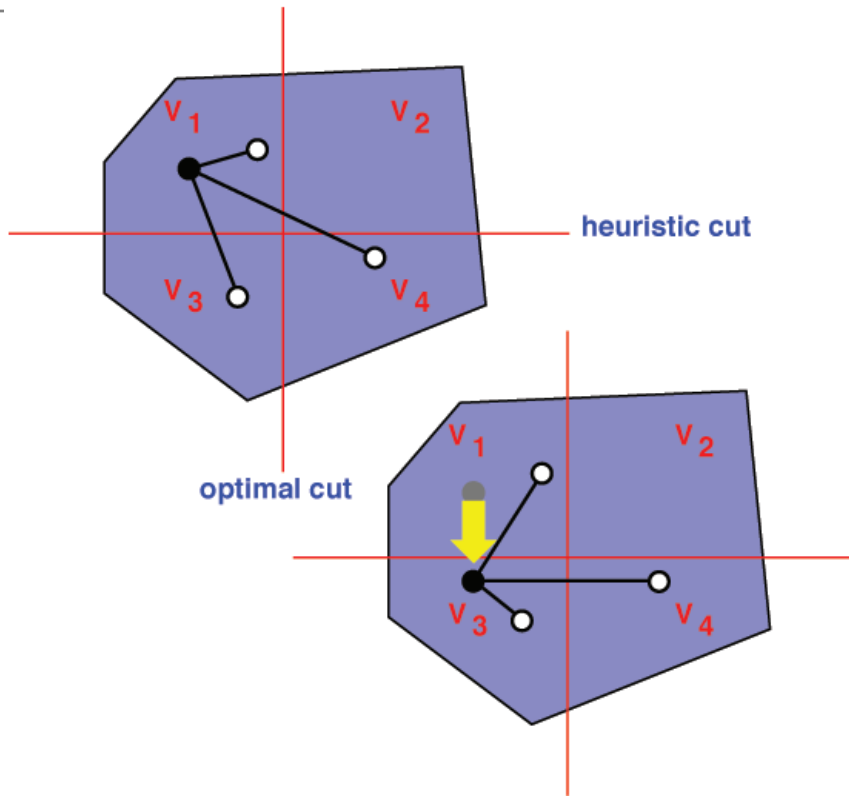
- Heuristic yields $V_1 \cup V_2, V_3 \cup V_4$
- Optimal yields $V_1 \cup V_3, V_2 \cup V_4$

Max Cut Algorithm



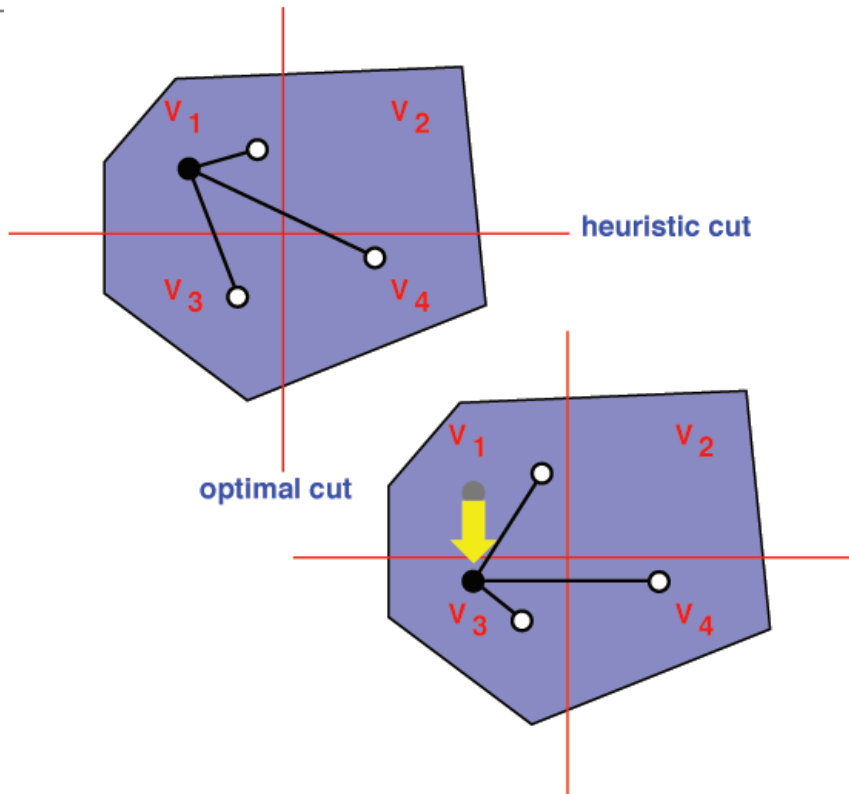
- Every cut partitions the edges into **cut edges**, E_C , and **non-cut edges**, E_N .
- Let c_v be the number of **cut edges** from node v .
- Let n_v be the number of **non-cut edges** from v .

Max Cut Algorithm



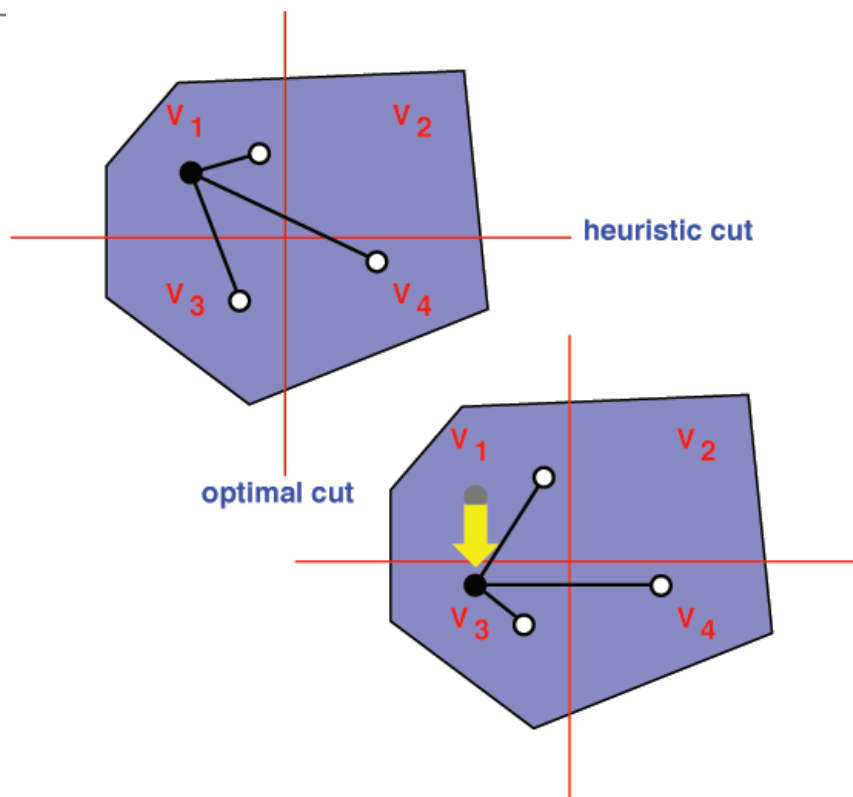
- When algorithm terminates, for every node v , the number of **cut edges** is **greater** or equal than the number of **non-cut edges**, $c_v \geq n_v$.

Max Cut Algorithm



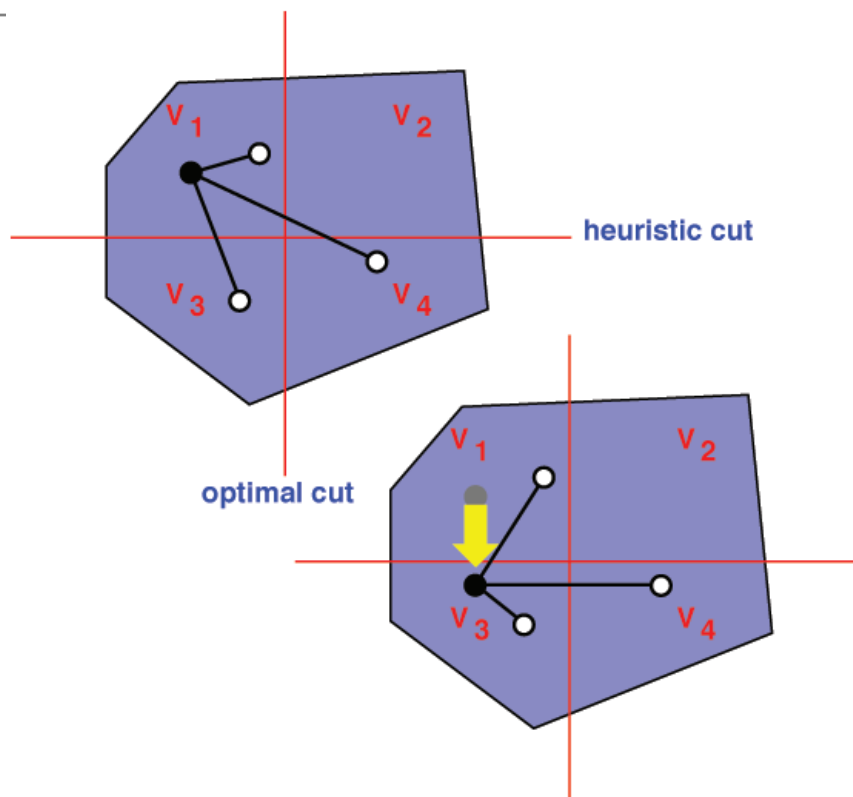
- When algorithm terminates, for every node v , the number of **cut edges** is **greater** or equal than the number of **non-cut edges**, $c_v \geq n_v$.
- Otherwise, switching the node v would increase the size of the cut produced by the algorithm.

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



- 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 : $\sum_v c_v \geq \sum_v n_v$.

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**).

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 $|E_C| \geq |E_N|$.

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 $|E_C| \geq |E_N|$.
- $\implies 2|E_C| \geq |E_N| + |E_C| = |E|$

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 $|E_C| \geq |E_N|$.
- $\implies 2|E_C| \geq |E_N| + |E_C| = |E|$
- So $|E_C| \geq |E|/2$.

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 $|E_C| \geq |E_N|$.
- $\implies 2|E_C| \geq |E_N| + |E_C| = |E|$
- So $|E_C| \geq |E|/2$.
- Clearly $|E| \geq OPT$ (any cut is set of edges).

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 $|E_C| \geq |E_N|$.
- $\implies 2|E_C| \geq |E_N| + |E_C| = |E|$
- 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 $\frac{1}{2}$ -MaxCut approximation ($c(\mathcal{A})/OPT \geq 1 - 1/2$). ♣