Computer Science 365: Design and Analysis of Algorithms Last lecture

What we did
My favorite problem
My favorite heuristic
Exciting developments
Topics we've neglected
Lies I've told
What else to take
Where to learn more

Business

We will need ULAs next year.

Please consider signing up.

What we did

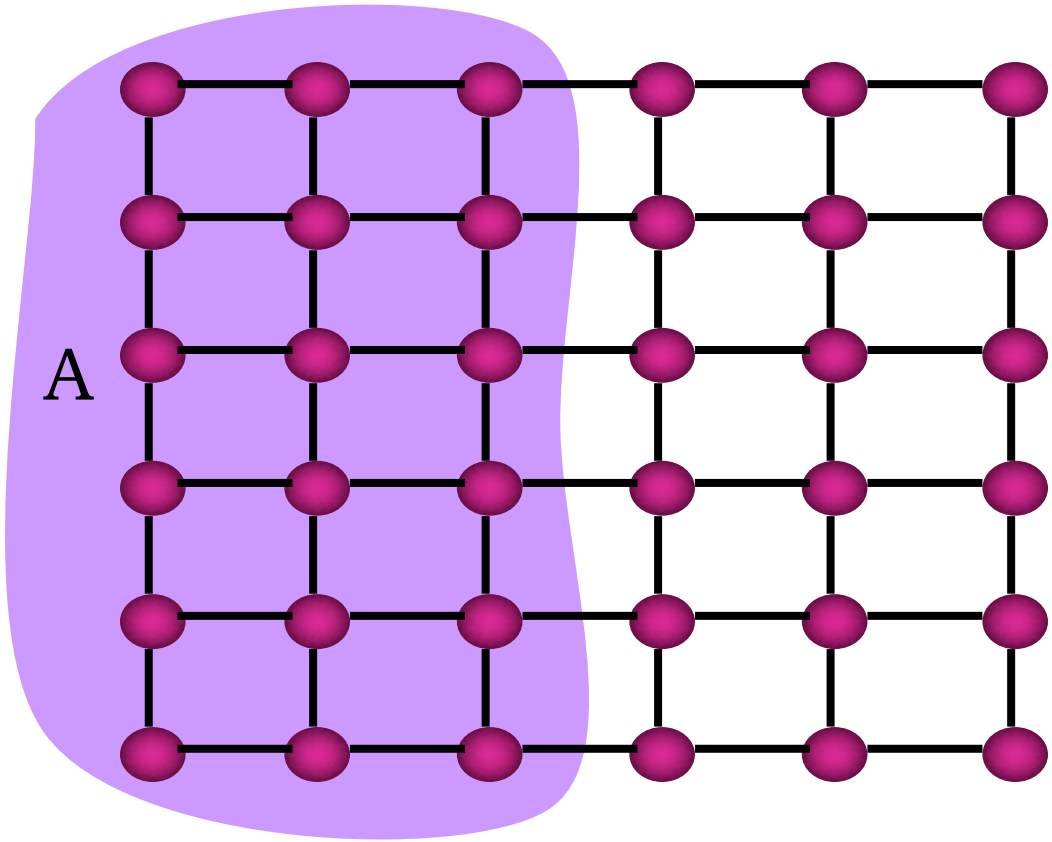
Stable Matching
Greedy Algorithms
Dynamic Programming
Divide and Conquer
Flow
NP-Completeness
Approximation Algorithms
Randomized Algorithms

Patience and Perseverance

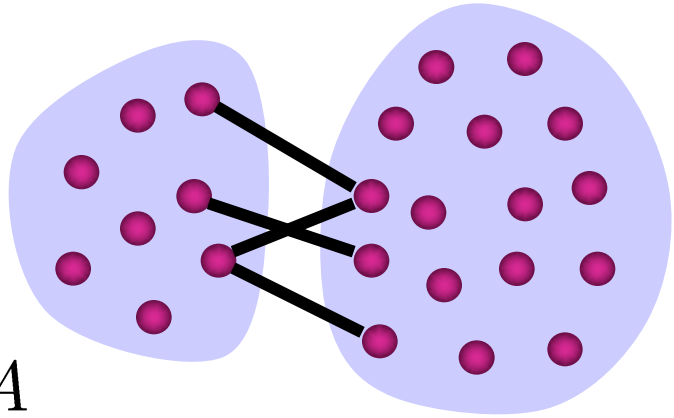
Favorite Problem(s):

Graph bisection:

divide vertices V into two sets A and B
of equal size,
cutting as few edges as possible



Favorite Problem: Sparsest Cut



Sparsity of cut (A,B)

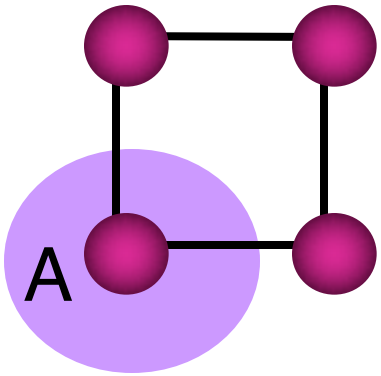
$$\Phi(A) \stackrel{\text{def}}{=} \frac{\#\text{ edges leaving } A}{\min(|A|, |B|)}$$

Sparsity of G

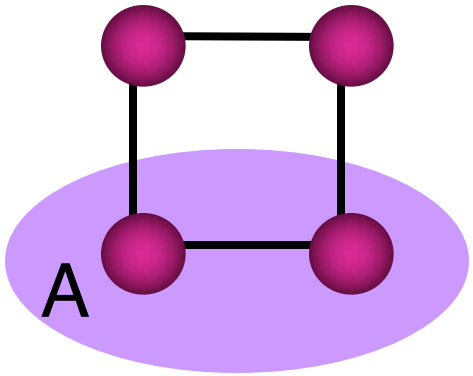
$$\Phi_G \stackrel{\text{def}}{=} \min_A \Phi(A)$$

Sparsity

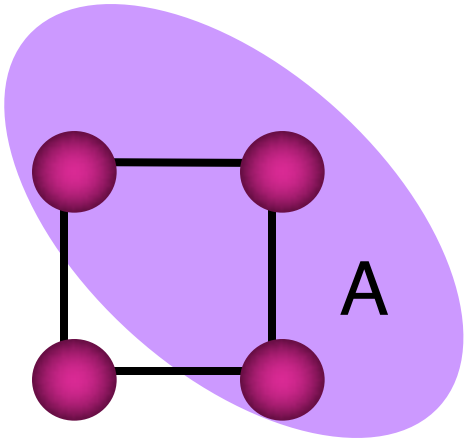
$$\Phi(A) \stackrel{\text{def}}{=} \frac{\#\text{ edges leaving } A}{\min\left(|A|, |B|\right)}$$



$\Phi(A) = 2$



$\Phi(A) = 1$

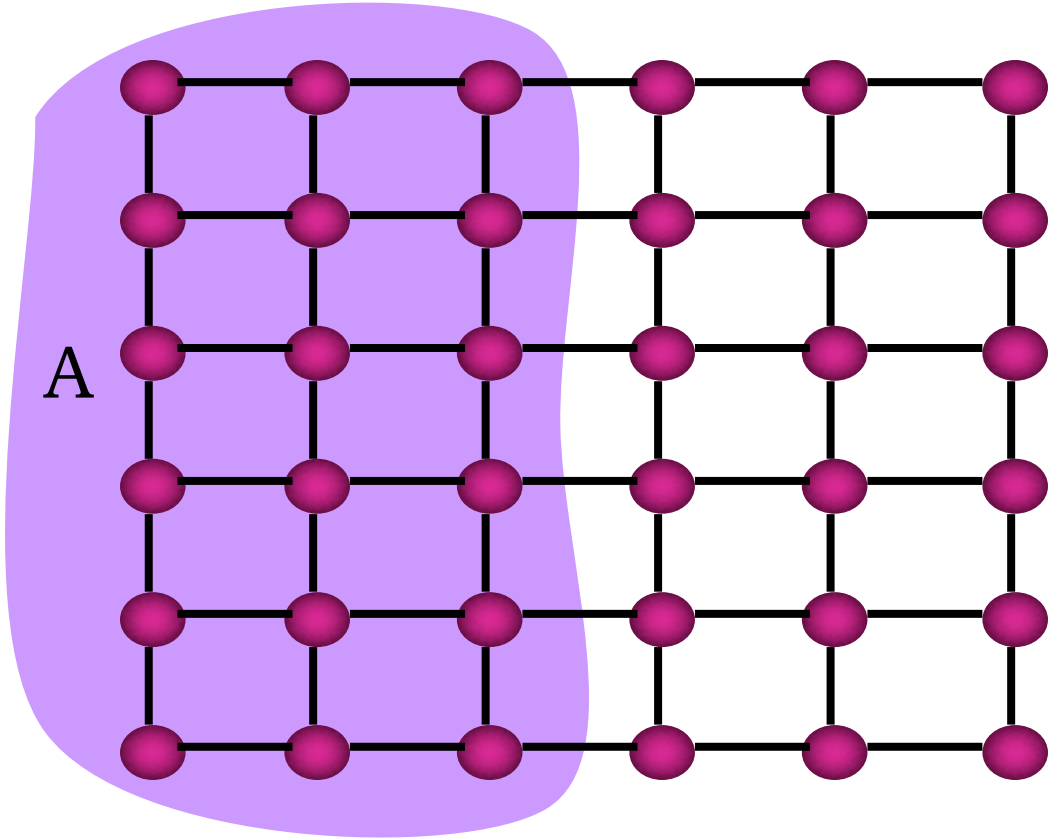


$\Phi(A) = 2$

Sparsity

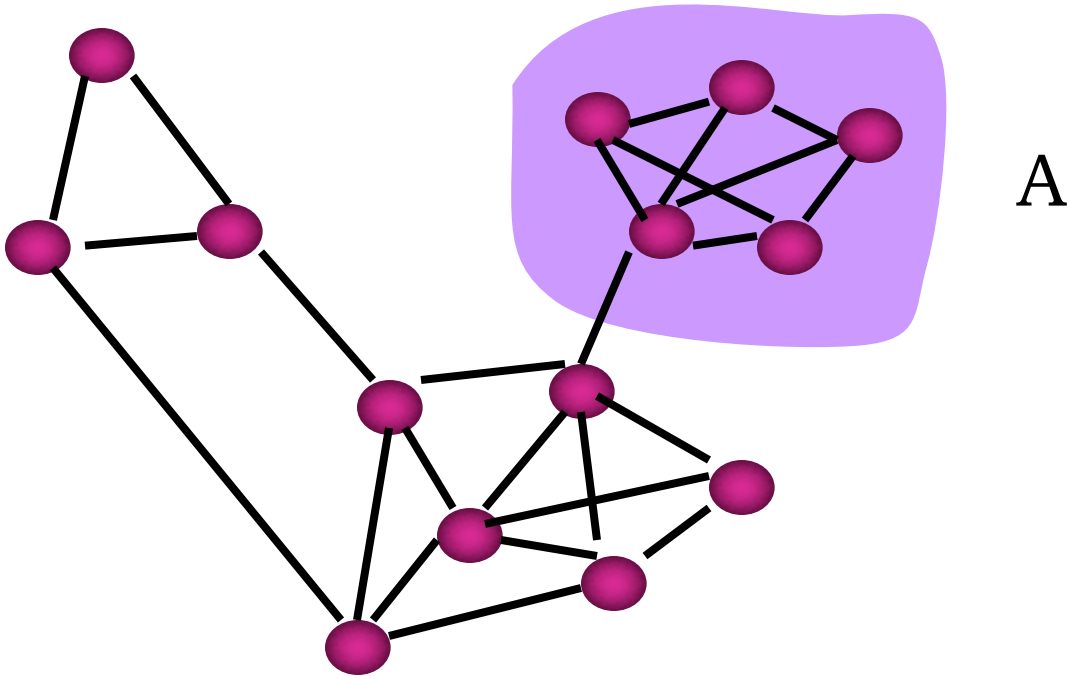
$$\Phi(A) \stackrel{\text{def}}{=} \frac{\#\text{ edges leaving } A}{\min(|A|, |B|)}$$

$\Phi(A) \sim 2/\sqrt{n}$



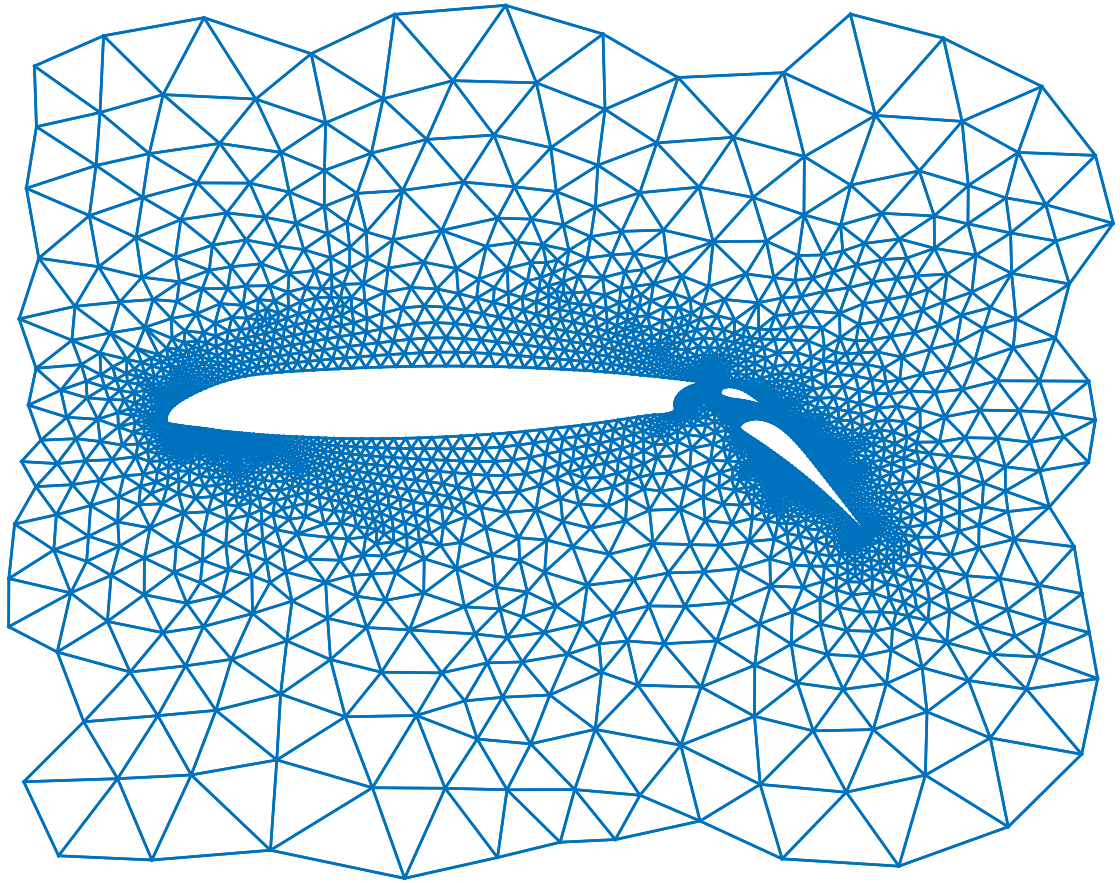
Why Sparsest Cut:

Clustering in graphs
Organizing data



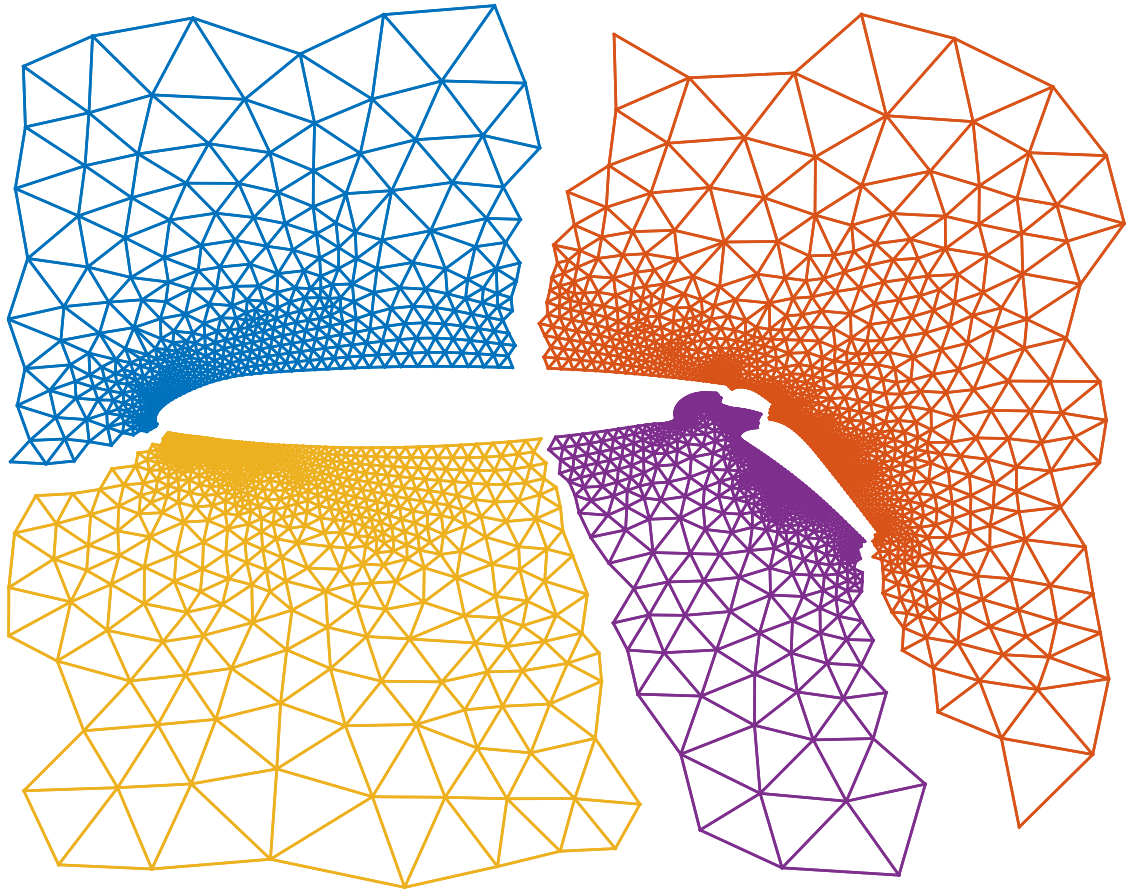
Why Sparsest Cut:

Dividing computation over processors.



Why Sparsest Cut:

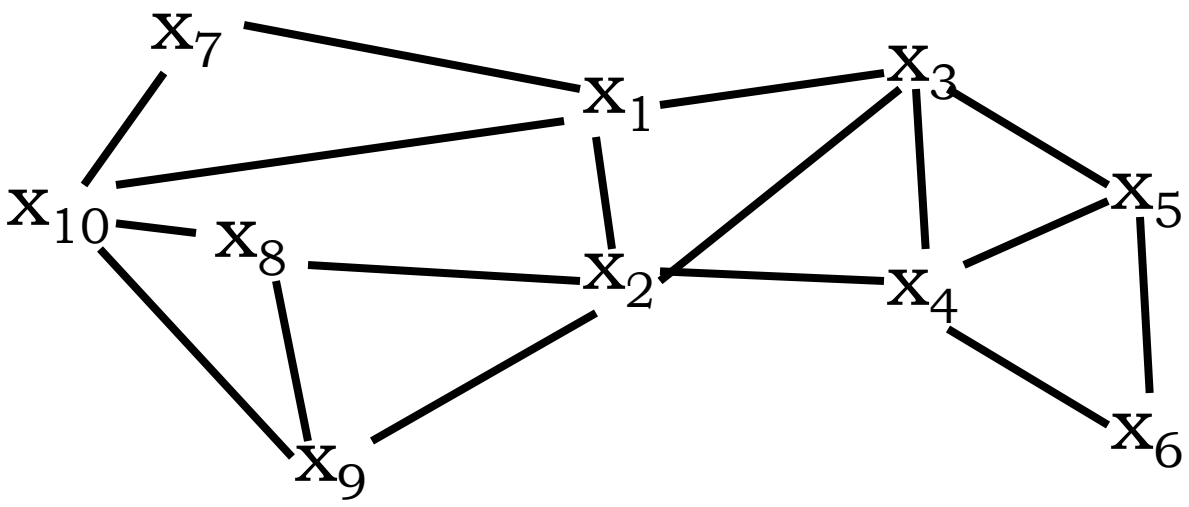
Dividing computation over processors.



Why Sparsest Cut:

Generalized Divide and Conquer (3-SAT)

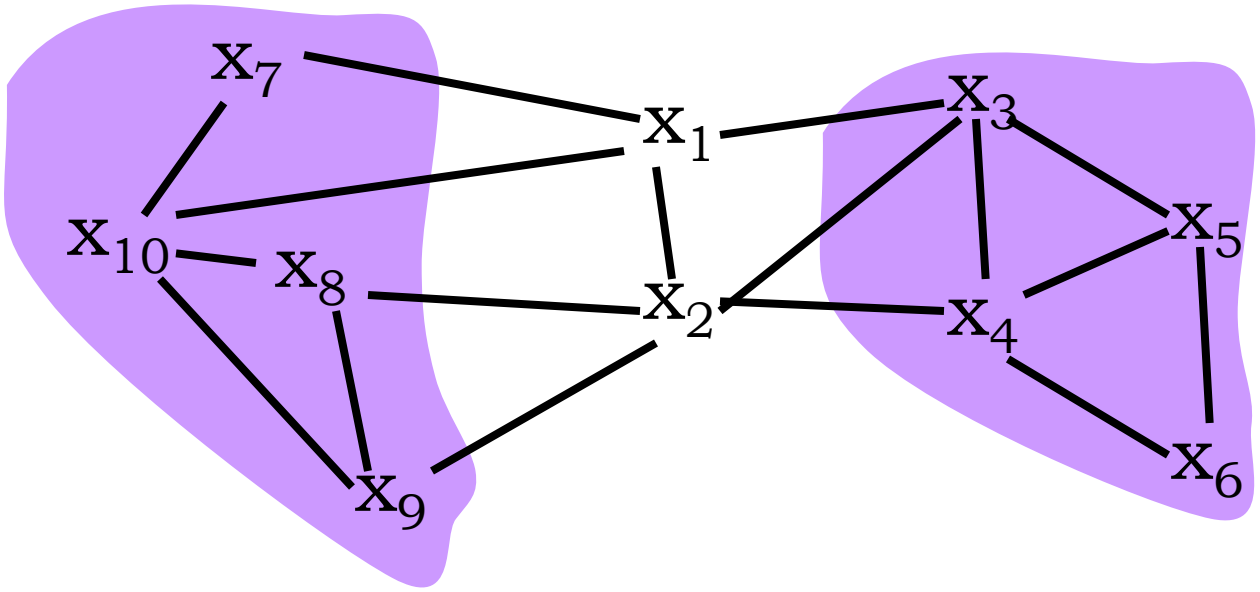
Variables -> vertices
Clauses -> cliques (edges on all pairs)



Why Sparsest Cut:

Generalized Divide and Conquer (3-SAT)

Variables -> vertices
Clauses -> cliques (edges on all pairs)



Cut vertices instead of edges

Complexity of Sparsest Cut

Exact solution NP hard.

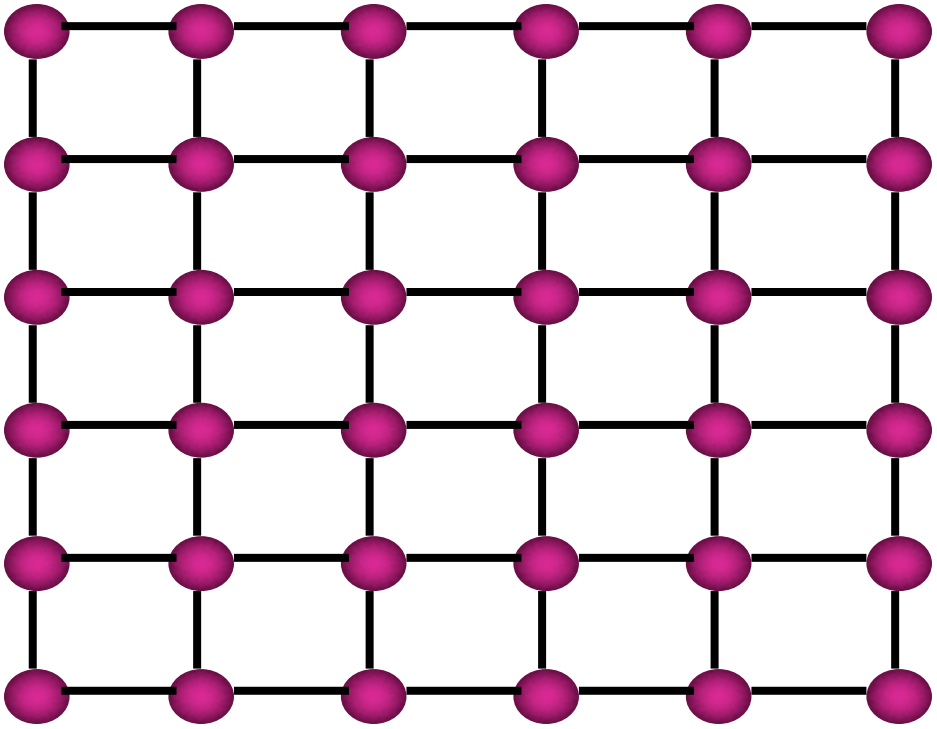
If (1+ε)-approximation,
then SAT is in time $2^{O(n^{\alpha})}$, $\alpha < 1$

Can approximate within $O(\sqrt{\log n})$

Really good heuristics (Chaco, Metis, Scotch)
by multilevel methods.

Multilevel methods

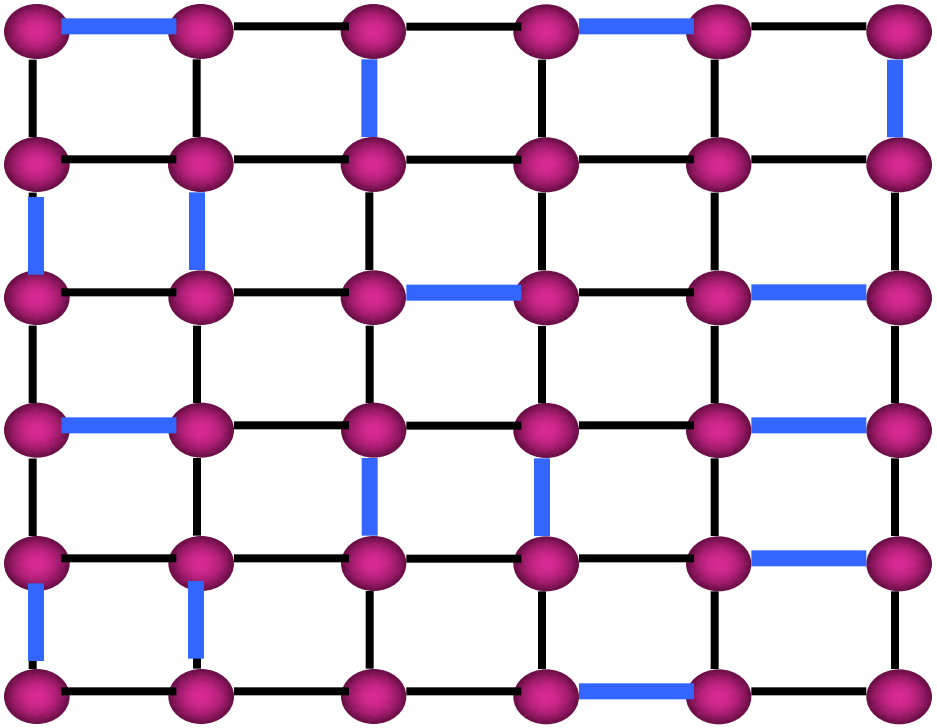
1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.



Multilevel methods

1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

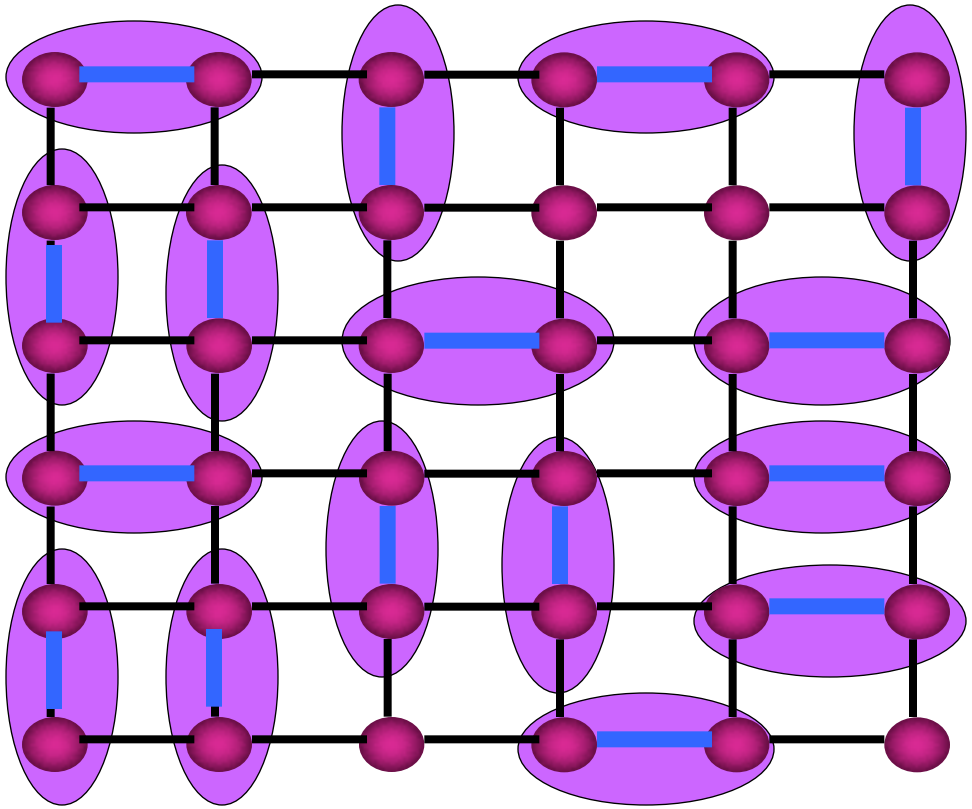
1. Match nodes at random, and merge



Multilevel methods

1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

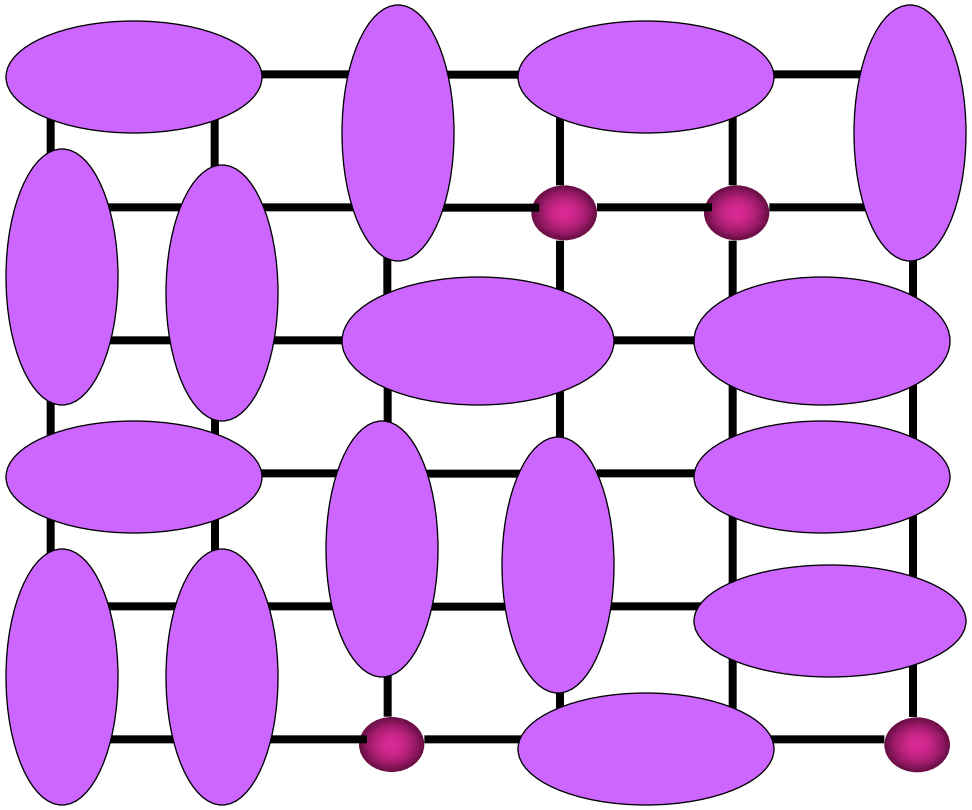
1. Match nodes at random, and merge



Multilevel methods

1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

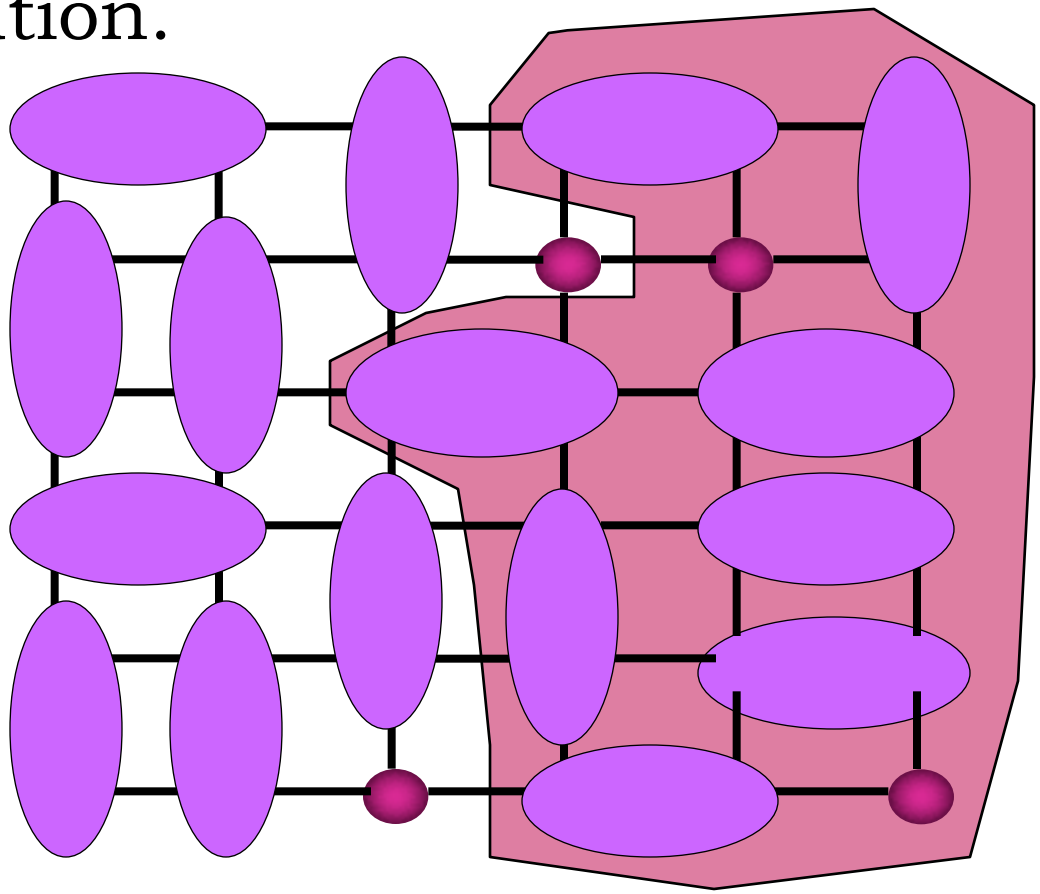
1. Match nodes at random, and merge



Multilevel methods

1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

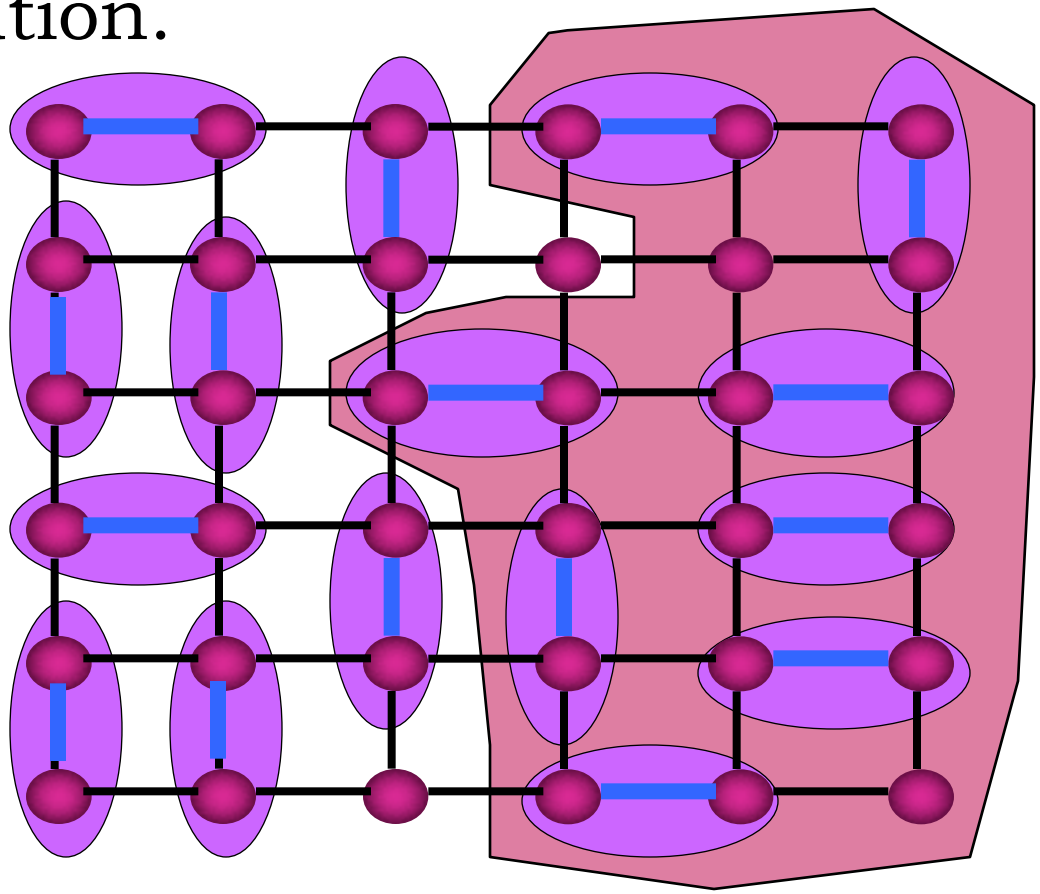
2. Solve sub-problem



Multilevel methods

1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

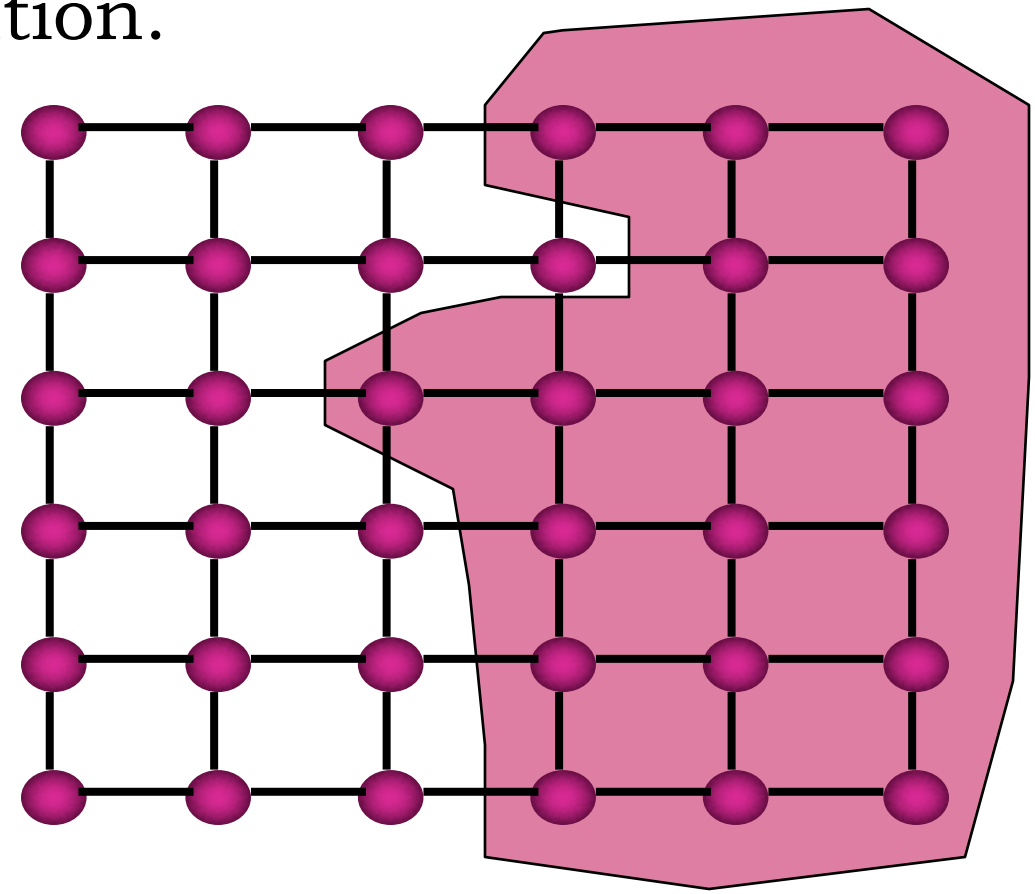
3. Lift solution



Multilevel methods

1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

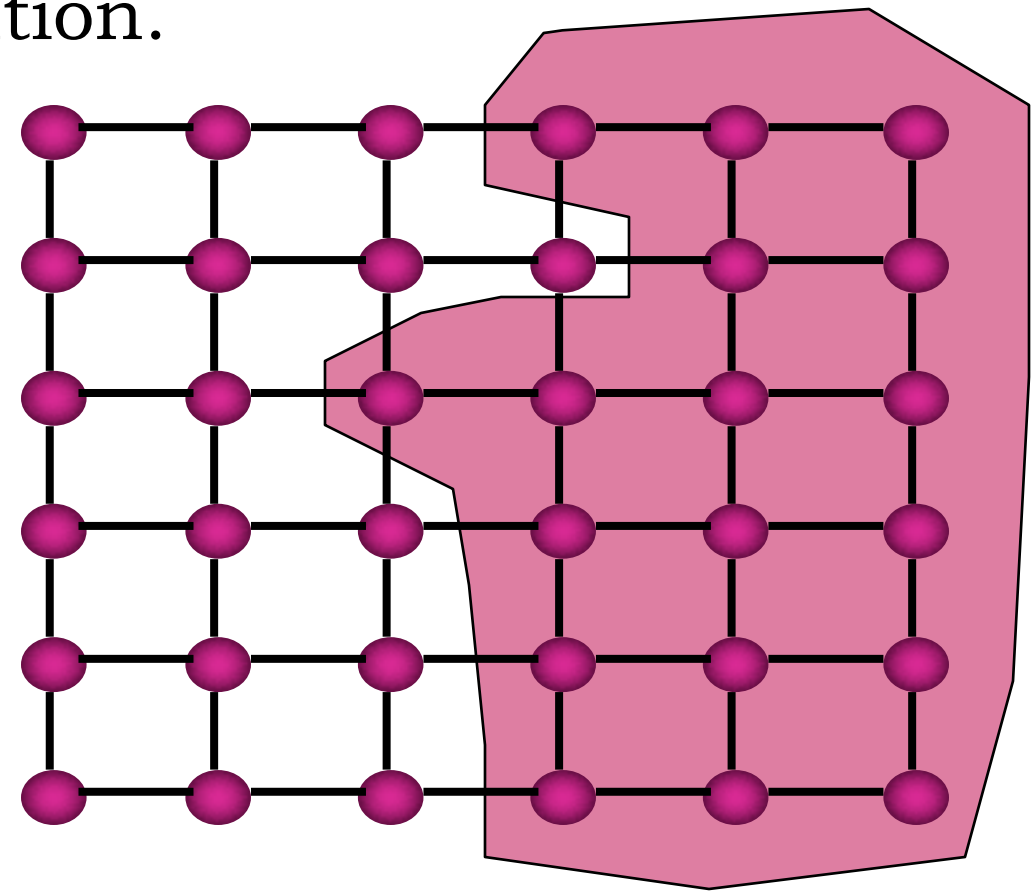
3. Lift solution



Multilevel methods

1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

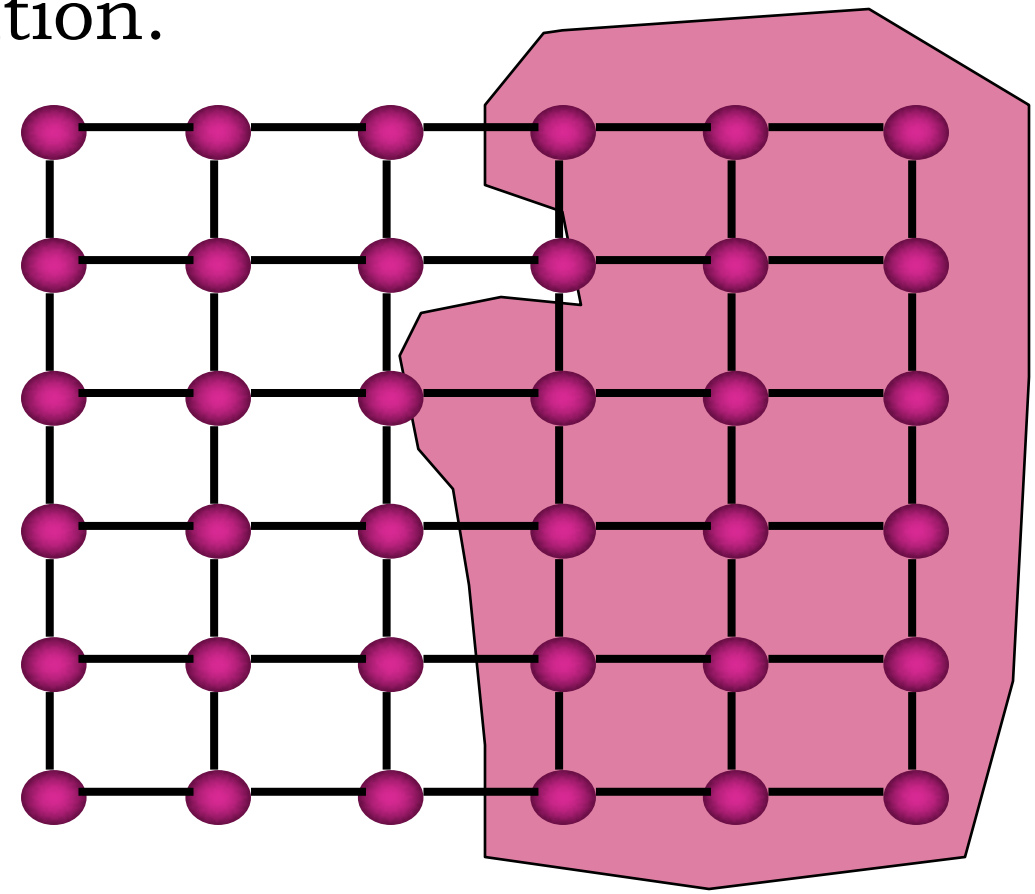
4. Refine solution



Multilevel methods

1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

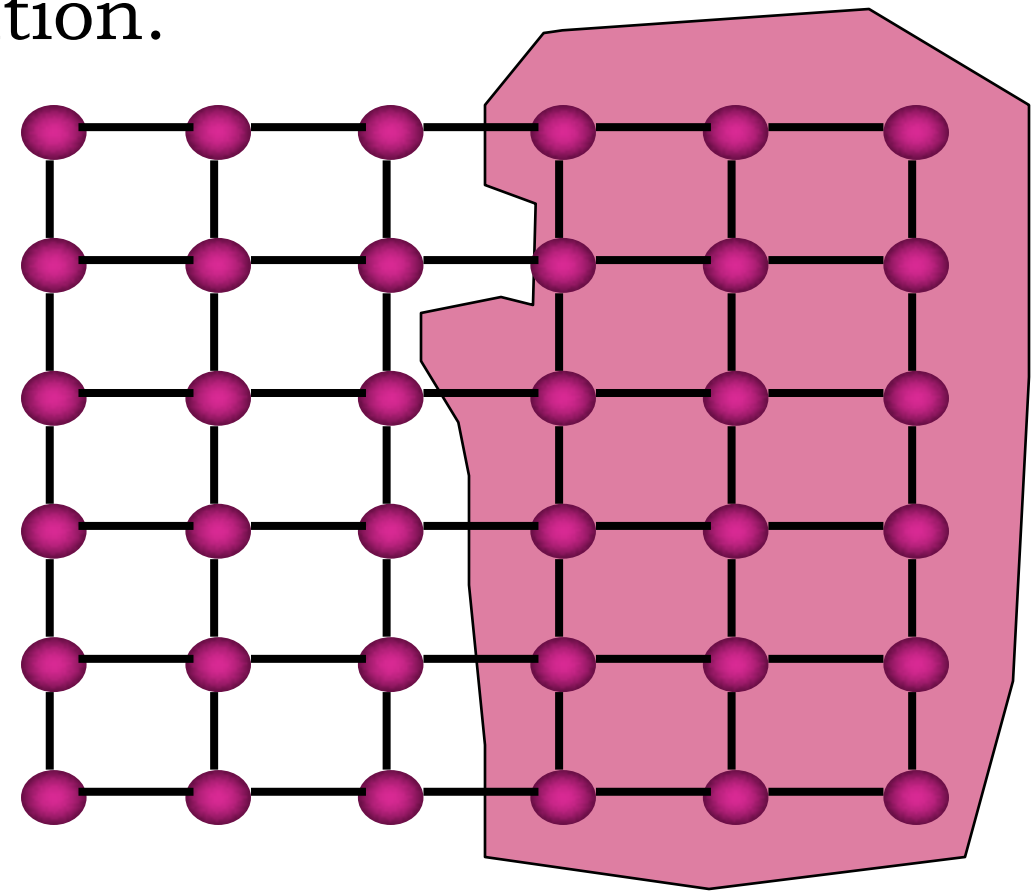
4. Refine solution



Multilevel methods

1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

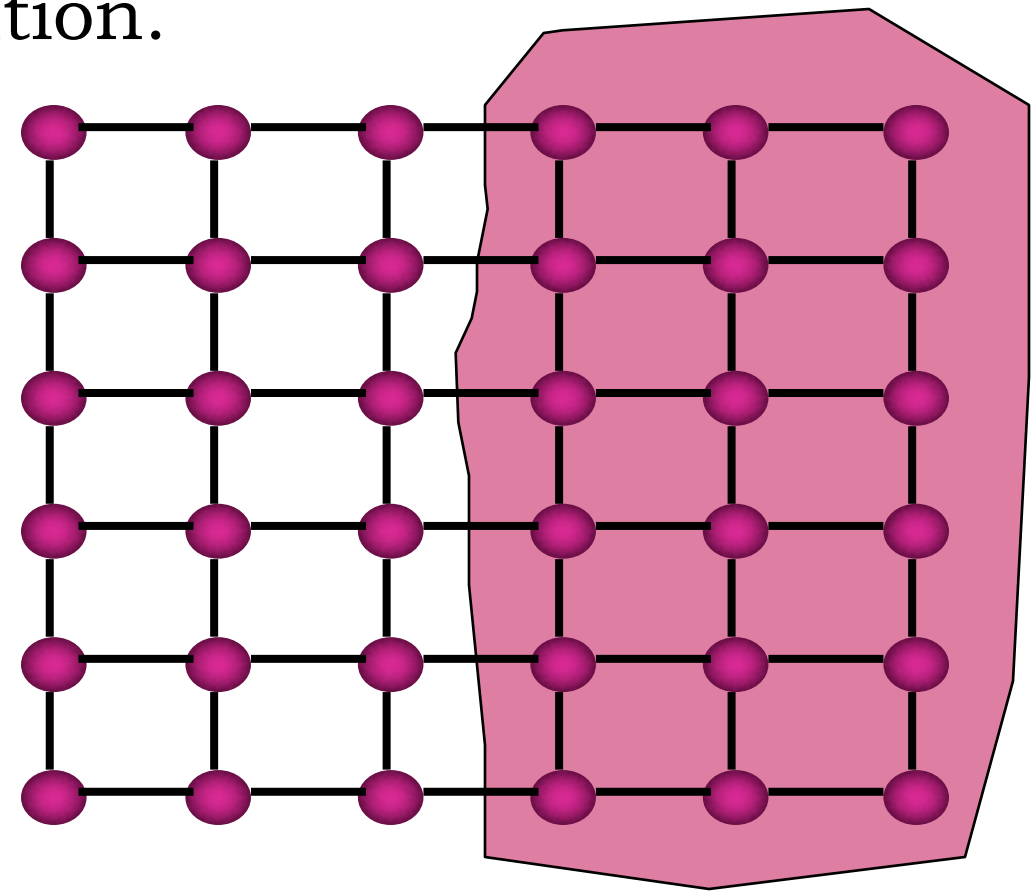
4. Refine solution



Multilevel methods

1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

4. Refine solution



Multilevel methods

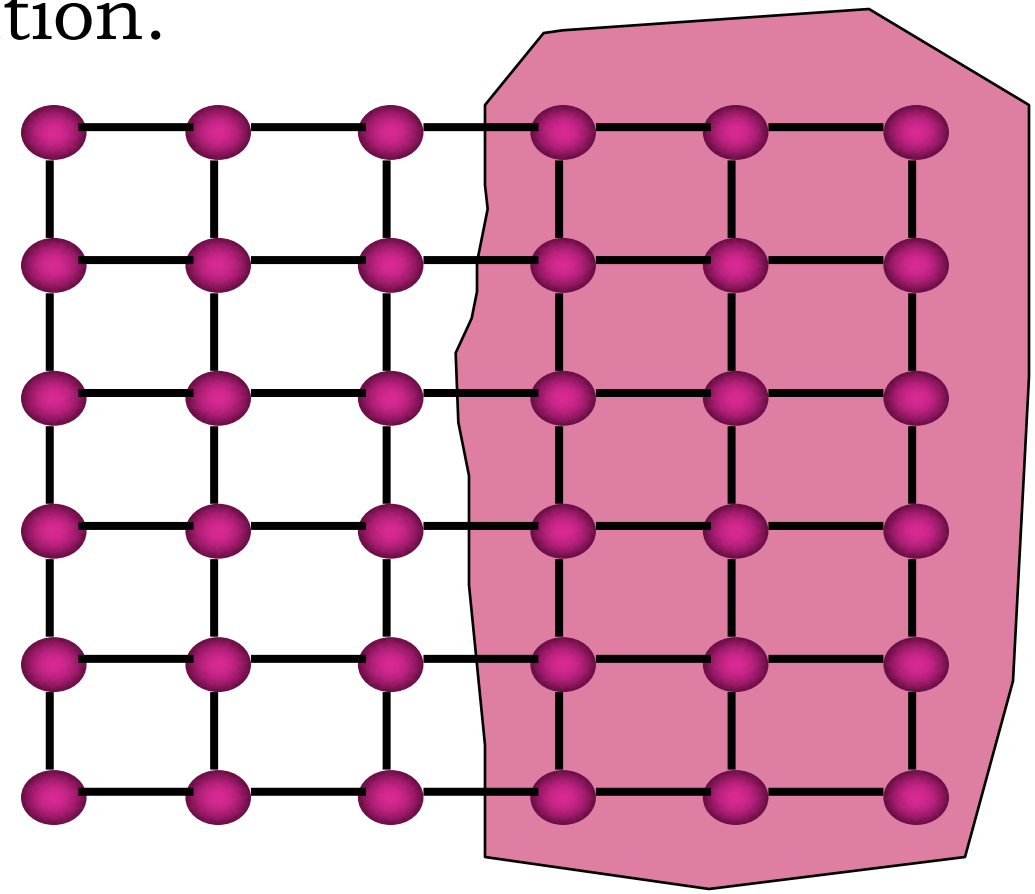
1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.

4. Refine solution

Very fast.

Works well.

No analysis, yet.



Hardness of Approximation

Max 3-SAT: Satisfy as many clauses as possible.

Hard to do better than $7/8+\varepsilon$

Set-Cover: Hard to do better than $(1-\varepsilon) \log n$

Max IS: Hard to do better than $n^{1-\varepsilon}$

VC: Hard to do better than 1.36 (and maybe $2-\varepsilon$)

Unique Games Conjecture (Khot):

Hard to approximate the fraction of solvable linear equations modulo p.

Implies many tight hardness results:

VC is hard to approximate better than 2.

Semidefinite relaxations are optimal for many other problems.

The Strong Exponential Time Hypothesis (Impagliazzo, Paturi, Zane '01)

There is no algorithm that solves SAT on n variables and m clauses in time

$$O(m^c(2-\epsilon)^n)$$

Implies no sub-quadratic algorithms for:

- Edit Distance (Bakurs, Indyk '15)
- Fréchet Distance (Bringmann '14)

The Strong Exponential Time Hypothesis (Impagliazzo, Paturi, Zane '01)

There is no algorithm that solves SAT on n variables and m clauses in time

$$O(m^c(2-\epsilon)^n)$$

Implies hardness of approximation:

ln n – c ln ln n for set cover (Harris '16)

Densest Subgraph (BKRW '15)

The Strong Exponential Time Hypothesis (Impagliazzo, Paturi, Zane '01)

There is no algorithm that solves SAT on n variables and m clauses in time

$$O(m^c(2-\epsilon)^n)$$

k-Sum : (Pătraşcu-Williams '10)

Given n numbers, are there k that sum to 0?

Best algs: $O(n^{\lceil k/2 \rceil})$

The Strong Exponential Time Hypothesis (Impagliazzo, Paturi, Zane '01)

Given N vectors in d dimensions,

can you test if two are orthogonal in time $O(N^{2-\epsilon})$?

Given m clauses on n variables,

create $N = 2^{n/2}$ vectors in m dimensions so that

two are orthogonal if and only if clauses satisfiable

Williams '05

The Strong Exponential Time Hypothesis (Impagliazzo, Paturi, Zane '01)

Given m clauses on n variables,

create $N = 2^{n/2}$ vectors in m dimensions so that

two are orthogonal if and only if clauses satisfiable

Partition variables in half, each with n/2.

For all $2^{n/2}$ truth assignments on these, create a vector with a 0 for clauses it satisfies, 1 for the others.

Two are orthogonal if their 0s span the clauses.

Williams '05

Faster Maximum Flow

With n vertices and m edges, can compute an ϵ-approximate maximum flow in time

$O(m \log^c n/\epsilon^3)$	Peng '15
$O(m^{10/7} \log^c n \log 1/\epsilon)$	Madry '13
$O(mn^{1/2} \log^c n \log 1/\epsilon)$	Lee, Sidford '15

By numerical optimization algorithms

Faster Shortest Paths (with negative edge lengths)

$$O(m^{10/7} \log^c n \log 1/\epsilon)$$

Cohen, Sankowski, Madry and Vladu ‘17

By numerical optimization algorithms

Neglected topic: Data structures

Fancy data structures enable fast algorithms.

Hashing

Splay trees (Sleator-Tarjan)
practical binary search trees

Bloom filters

Geometric data structures

Cache efficiency

Neglected topic: Dynamic Algorithms

Maintain answers on changing inputs.

Recent breakthroughs:

Maintaining components in dynamic graphs.
Time $O(\log^5 n)$ per edge insertion and deletion.
[Kapron-King-Mountjoy '13]

Cannot beat time $n^{1/3}$ for node insertions
unless are faster algorithms for 3SUM
[Abboud-Vassilevska '14]

Neglected topic: Geometric Algorithms

Convex Hulls

Voronoi Diagrams and Delaunay Triangulations

Meshing

Visibility

Point Location

Geometric Data Structures

Neglected topic: Primality Testing

Miller ‘76: in polynomial time,
if Extended Riemann Hypothesis true

Rabin ‘80: In randomized polynomial time,
detect composite with probability ½.

Adelman-Huang ’92: Expected polynomial time,
when stops zero probability of error.

Agarwal-Kayal-Saxena ’04: Polynomial time,
by derandomization.

Hard Problems? Factoring Integers

For b-bit integers, best-known complexity is

$$2^{O(b^{1/3}\log^{2/3} b)}$$

(assuming conjectures in number theory)

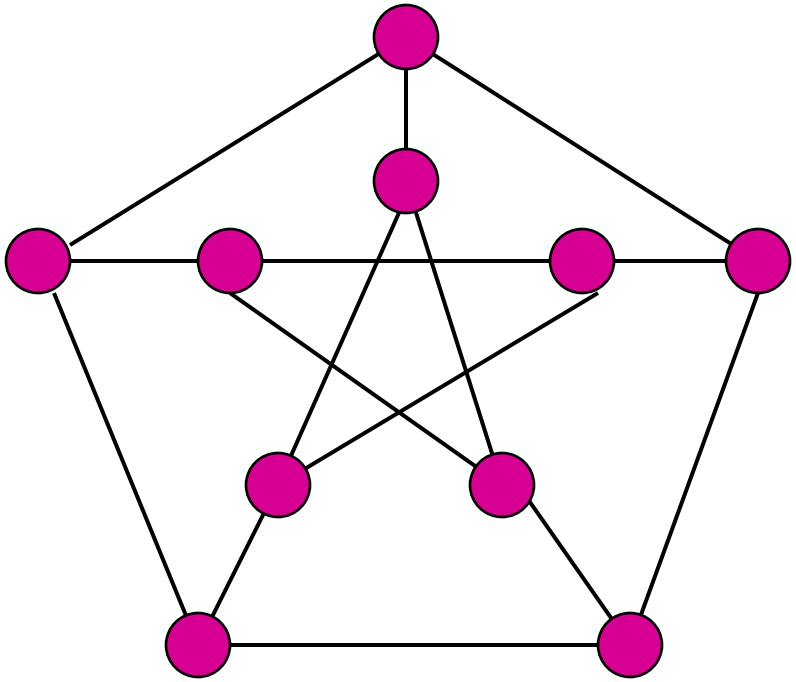
Pollard, Arjen and Hendrik Lenstra,

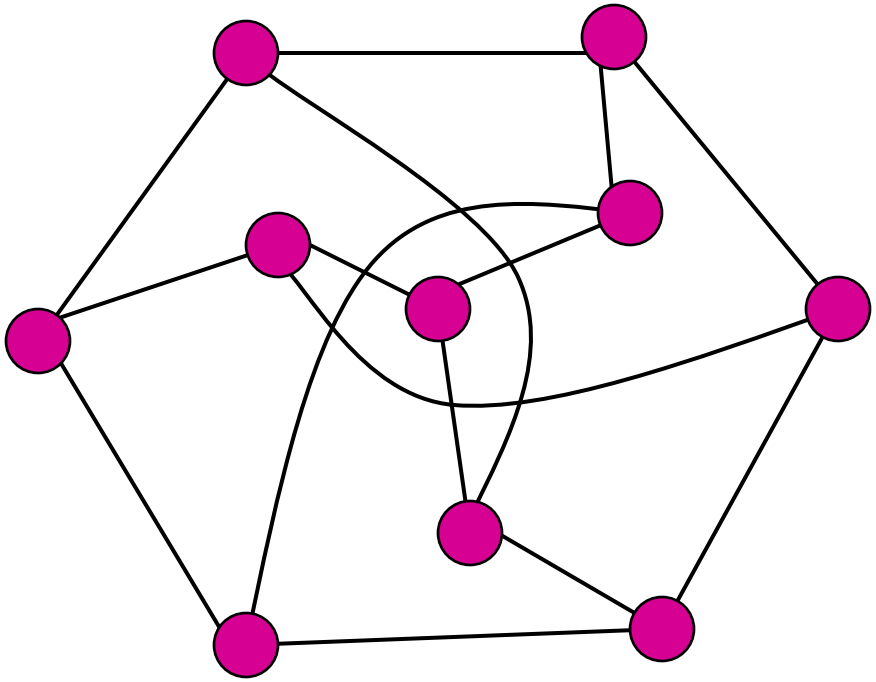
Manasse, Odlyzko, Adelman, Pomerance

If NP-hard, would have NP = co-NP.

Hard Problems? Graph Isomorphism.

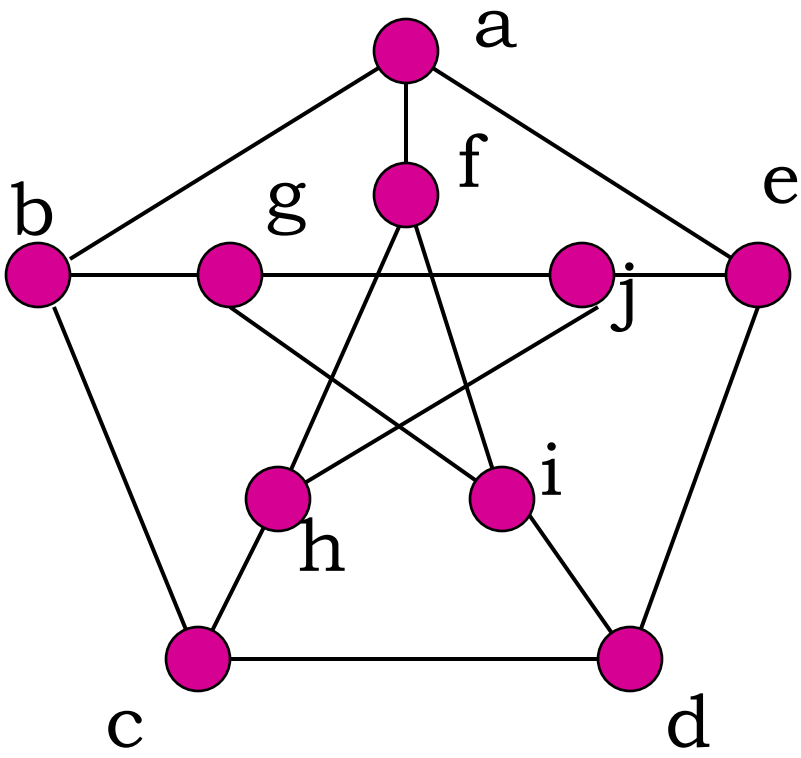
Given two labeled graphs, can vertex sets be relabeled so are same?

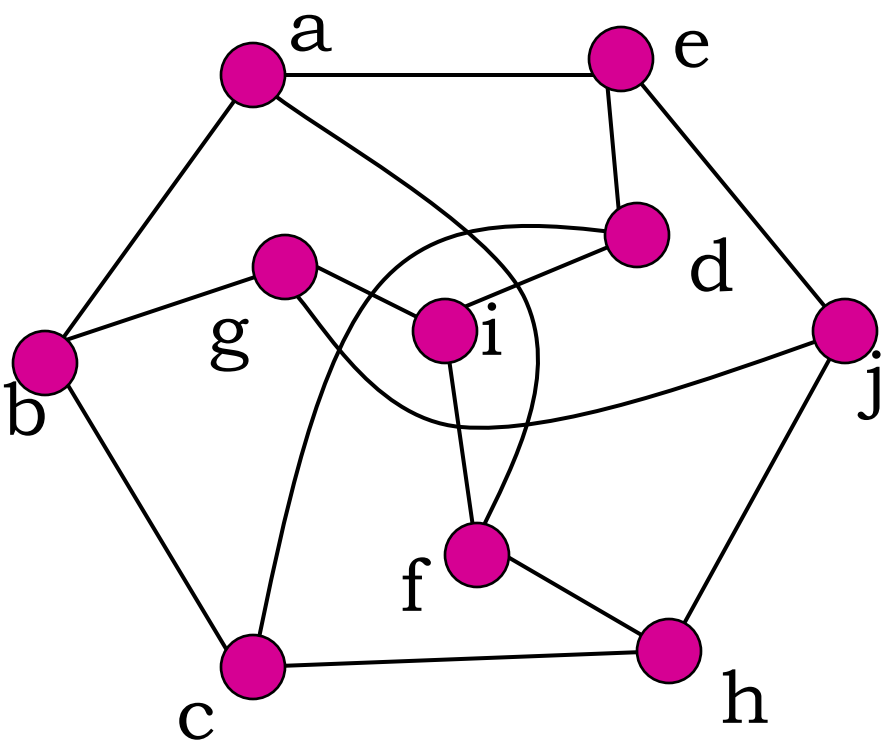




Hard Problems? Graph Isomorphism.

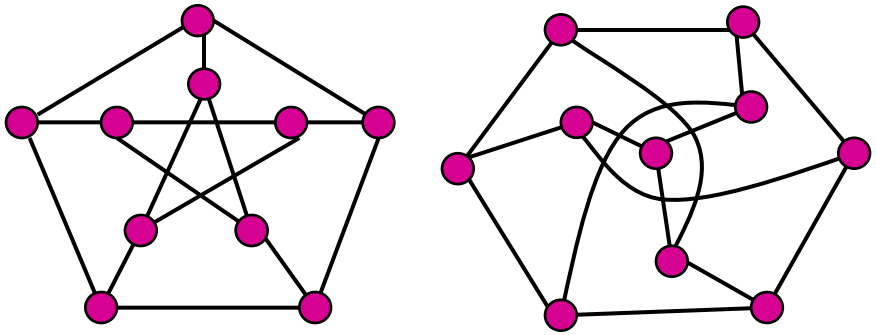
Given two labeled graphs, can vertex sets be relabeled so are same?





Hard Problems? Graph Isomorphism.

Given two labeled graphs, can vertex sets be relabeled so are same?



Complexity $2^{(\log n)^c}$ [Babai '15]

Polynomial time if constant degree

Are counter-examples to most naïve algorithms

Quantum Computing

Quantum computers have different operations

Factoring in polynomial time (Shor ’94)

and breaking Diffie-Helman (‘76) key exchange

Can they solve NP hard problems?

Graph isomorphism?

Numerical Algorithms

Solving systems of linear equations, quickly.

Doing Gaussian elimination correctly.

Graph algorithms and data structures.

Lies

Lies

Polynomial-time = efficient

Big-O notation.

Worst-case analysis.

Lies

Very few algorithms are
both elegant and useful.

Most algorithms papers are merely
evidence for the existence of useful algorithms.

Most problem we want to solve
do not have mathematically precise formulations

Related Courses

More Algorithms (Jim Aspnes)

CPSC 465: Theory of Distributed Systems

CPSC 469: Randomized Algorithms

Others?

Optimization:
S&DS 430 (Sekhar Tatikonda)

Linear programming.

Convex programming.

Gradient descent.

Newton's method.

Maximize something subject to constraints.

Applied Machine Learning: STAT 365

John Lafferty, Susan Wang

Prediction

Classification

Neural Nets

Deep Learning

Random Forests

Data Mining: CPSC 445

Guy Wolf

Principal Component Analysis

Multidimensional Scaling

Support Vector Machines

K-means clustering

Machine Learning for Biology: 453

Smita Krishnaswami

Computational Data Science: 262

Sahand Negahban & Me?

Assumes Calc and Linear Algebra

Databases, clusters, and parallel computing

Large scale optimization

SVD, ICA, Matrix completion

Streaming Algorithms

Numerical Methods:

CPSC 440 (Eisenstat or Rokhlin)
ENAS 440/441 (Beth Bennett)

Numerical problems that arise in sciences.

Solving linear equations.

Computing eigenvectors and eigenvalues.

Solving differential equations

Interpolation.

Integration.

CPSC 467: Cryptography and Security
Michael Fischer

RSA

One-way functions

Zero-knowledge proofs

Digital signatures

Playing poker over the phone

CPSC 468: Complexity Theory

(Dana Angluin)

P = NP?

NP = co-NP? short proofs of unsatisfiability?

Poly Time = Poly Space?

Interactive proofs.

Probabilistically checkable proofs.

Hardness of approximation.

Pseudo-random generators, and

Derandomizaton.

Interactive proofs:

Colors exist:

give colorblind two balls of different colors

shown one, we can always tell which it was

Interactive proofs:

Colors exist:

give colorblind two balls of different colors

shown one, we can always tell which it was

Graphs non-isomorphic:

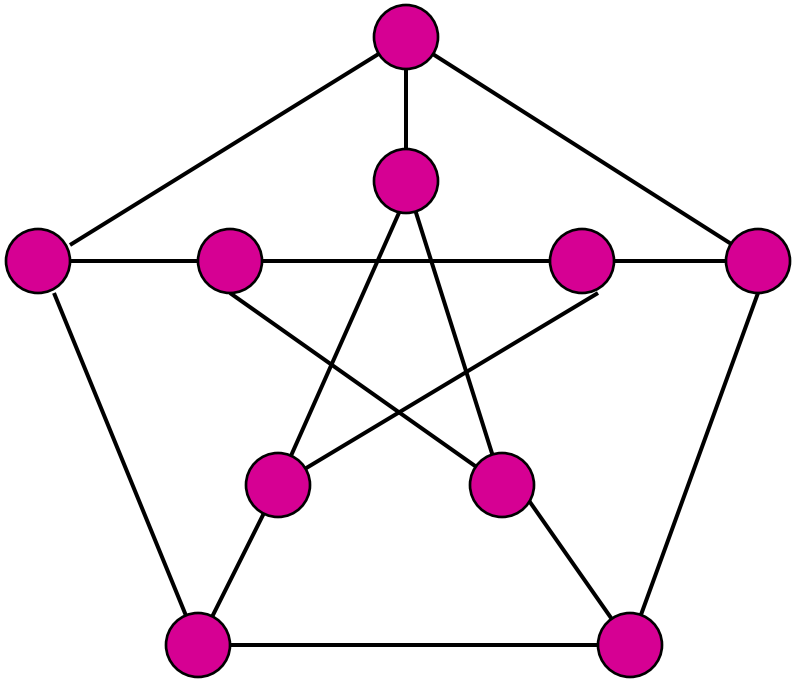
pick one at random,

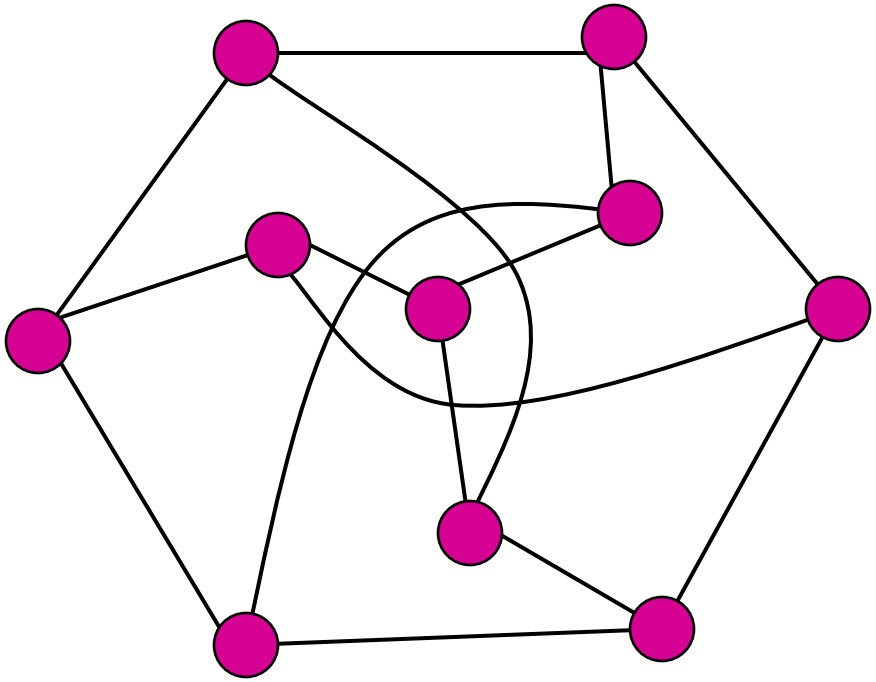
draw it at random,

can I tell you which it was?

Graph non-isomorphism.

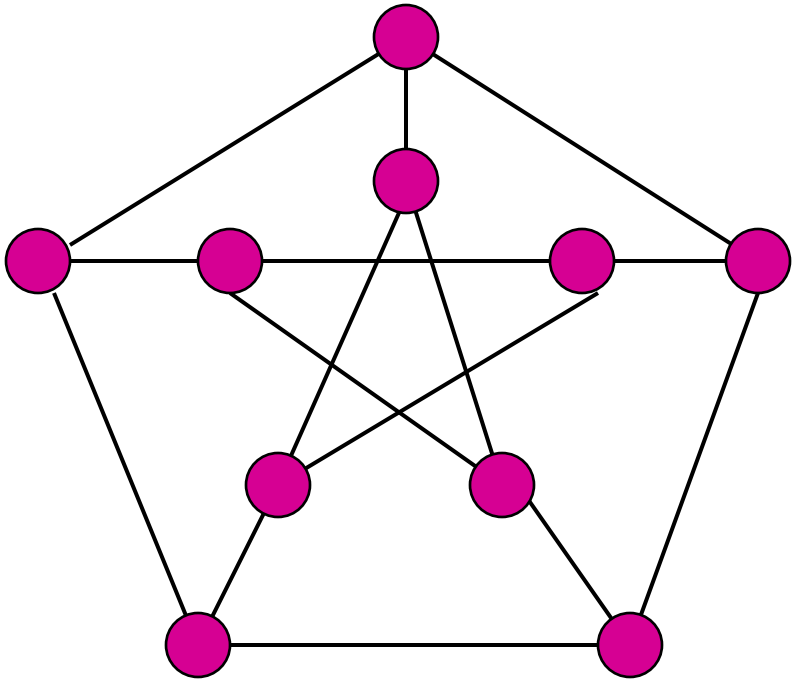
pick one





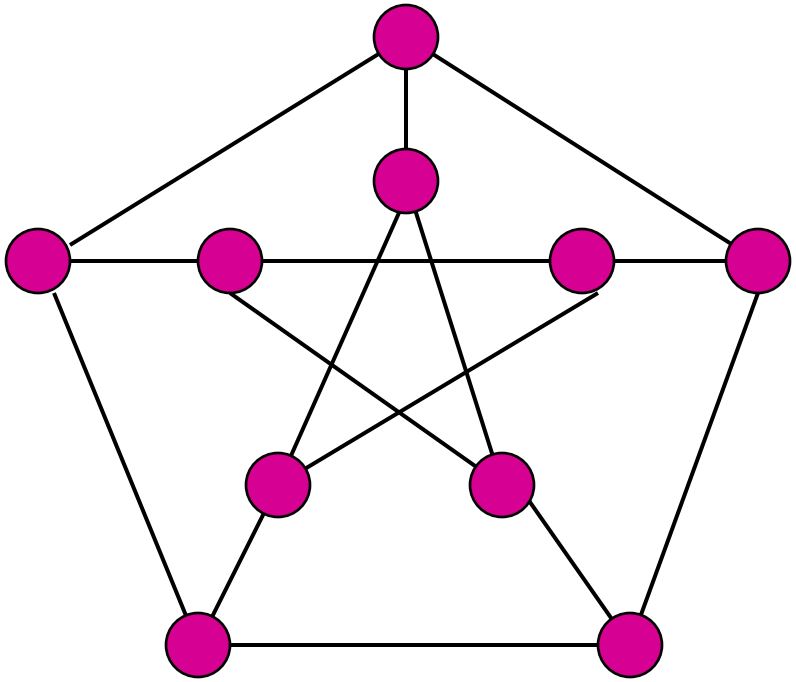
Graph non-isomorphism.

pick one



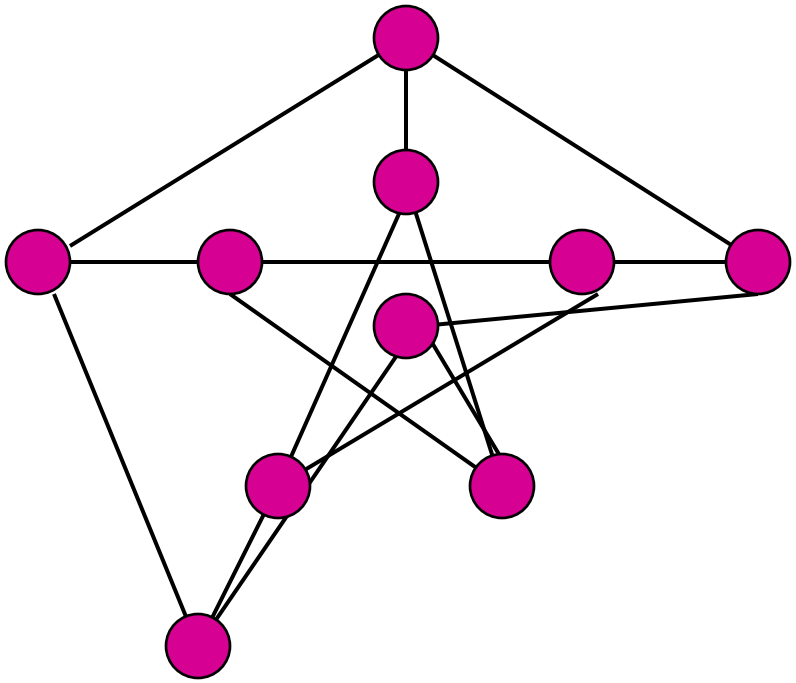
Graph non-isomorphism.

scramble it



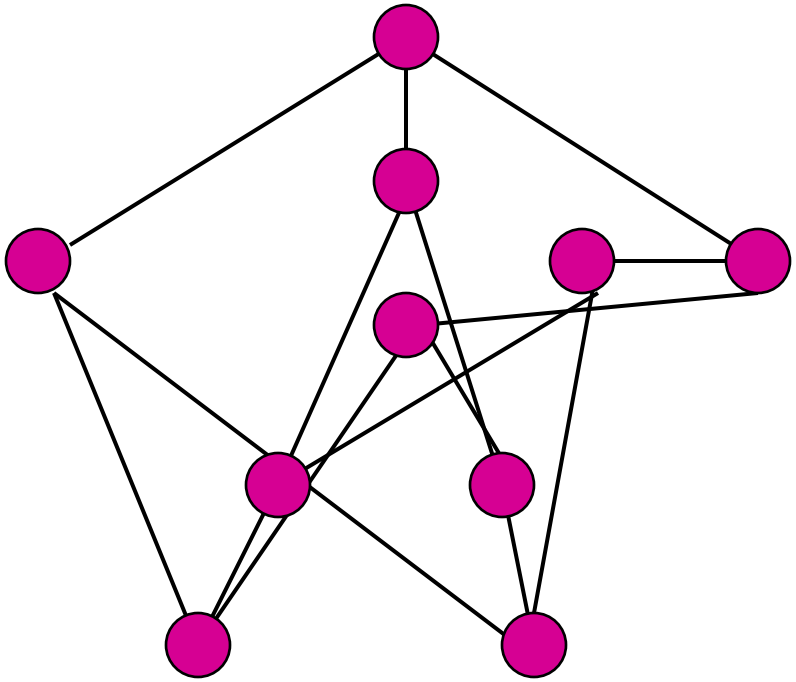
Graph non-isomorphism.

scramble it



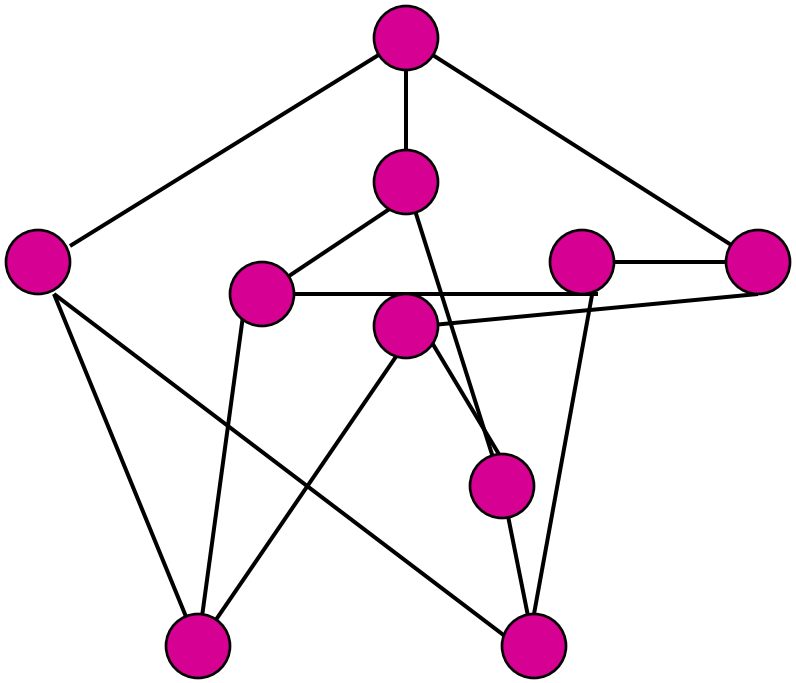
Graph non-isomorphism.

scramble it



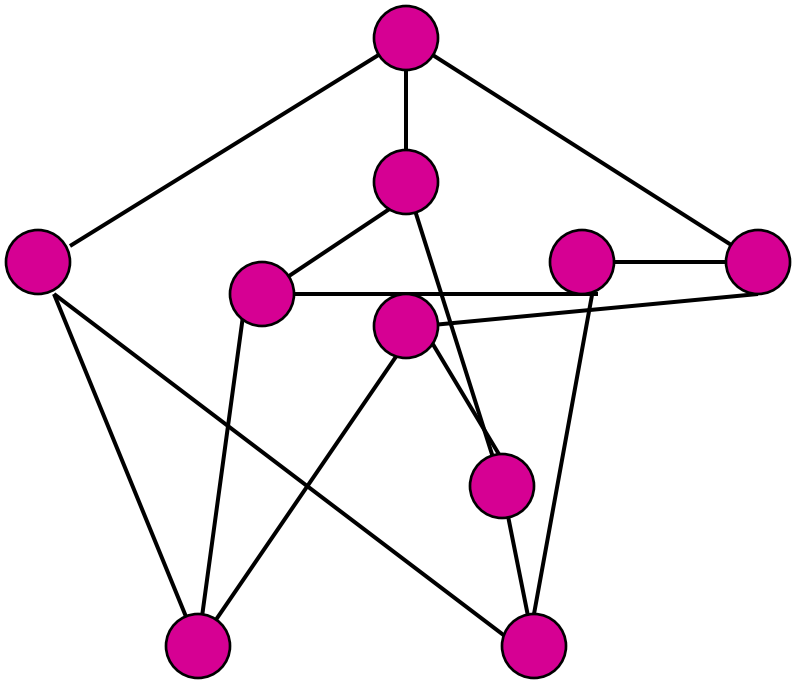
Graph non-isomorphism.

scramble it



Graph non-isomorphism.

randomly located vertices



If were same, no one can tell which it was

Probabilistically Checkable Proofs

A proof you can check by examining a few bits chosen carefully at random.

For every “yes” instance of every problem in NP, there is a PCP.

The source for most hardness of approximation results.

Math!

Combinatorics

Probability

Algebra: Group theory and finite fields

Fourier Analysis

Functional Analysis

Algebraic Geometry

Topology

etc.

Where to learn more

arXiv and blogs: http://feedworld.net/toc/

Class lecture notes.

Video lectures: https://sites.google.com/site/plustcs/
and the Simons Institute for the Theory of Computing

Surveys:

- Communications of the ACM
- Simons Foundation MPS Articles
- Quanta Magazine

Where to learn more

Major conferences:

- ACM STOC (Symposium on Theory of Computing)
- IEEE FOCS (Foundations of Computer Science)
- ACM/SIAM SODA (Symposium on Discrete Algorithms)
- ICALP (European Association for Theoretical CS)

- COLT (Computational Learning Theory)
- SOCG (Symposium on Computational Geometry)
- SPAA (Symposium on Parallelism in Algorithms and Architectures)
- ITCS (Innovations in Theoretical Computer Science)
- ALENEX (Algorithm Engineering and Experimentation)

Good luck with
the end of the semester!