Computer Science 366: Intensive Algorithms
Lecture 26

Business
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

The final is on May 10, 10:00am – 12:00

location TBD

It will cover material from class and

homework, but not the last 3 lectures.

I’ll hold office hours some time next week.

365 and 366 will need ULAs next year.

Please sign up.

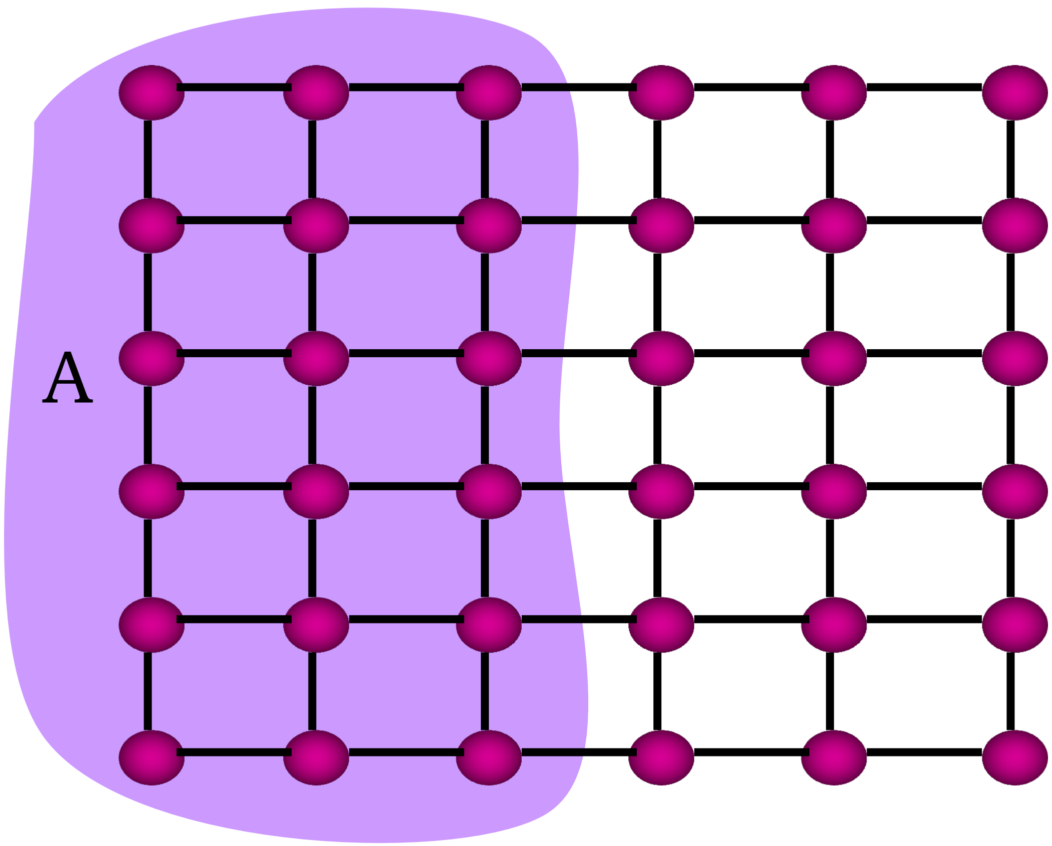
Favorite Problem(s):

Graph bisection:

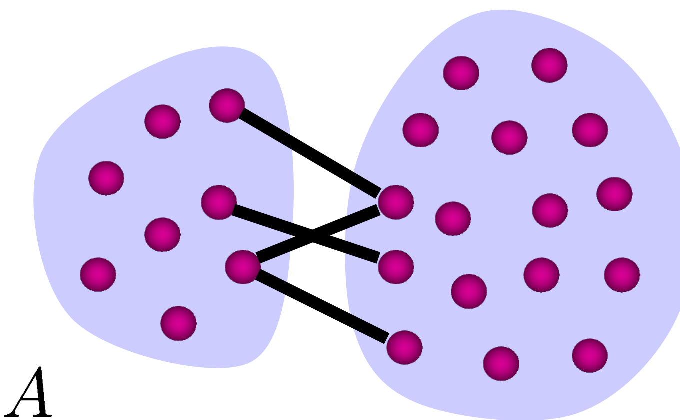
divide vertices V into two sets A and B

of equal size,

cutting as few edges as possible



Favorite Problem: Sparsest Cut



Sparsity of cut (A,B)

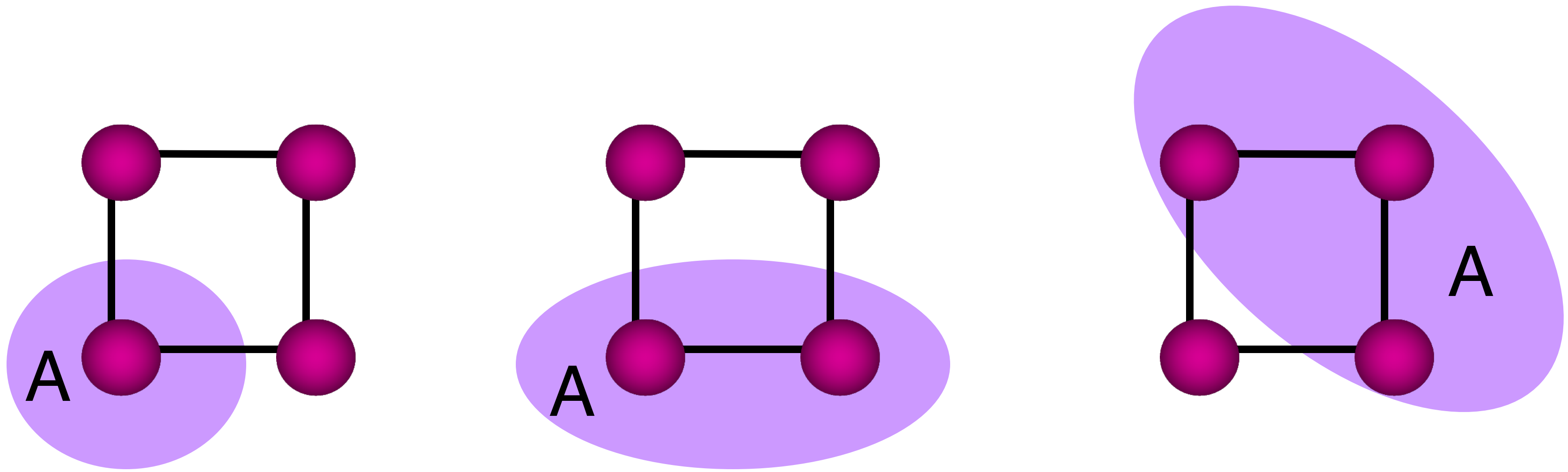
$$\Phi(A) \stackrel{\text{def}}{=} \frac{\#\ \text{edges leaving}\ A}{\min\left(|A|, |B|\right)}$$

Sparsity of G

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

Sparsity

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

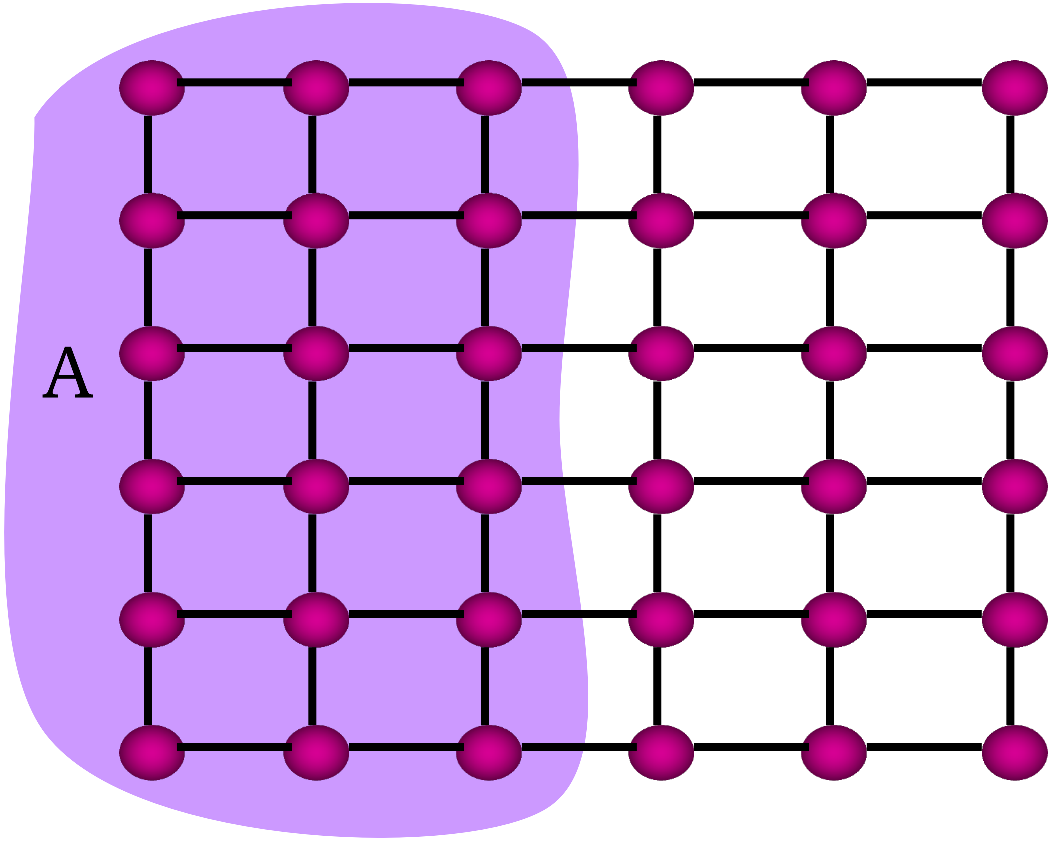


$\Phi(A) = 2$ $\Phi(A) = 1$ $\Phi(A) = 2$

Sparsity

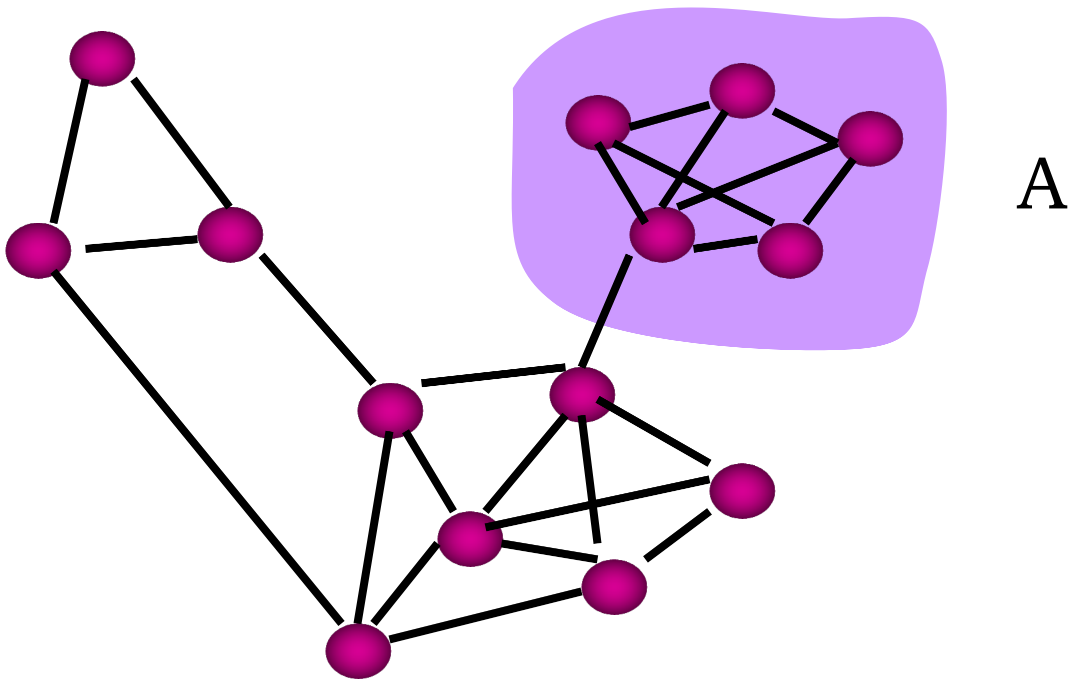
$$\Phi(A) \stackrel{\text{def}}{=} \frac{\#\text{ edges leaving } A}{\min\left(|A|, |B|\right)}$$

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



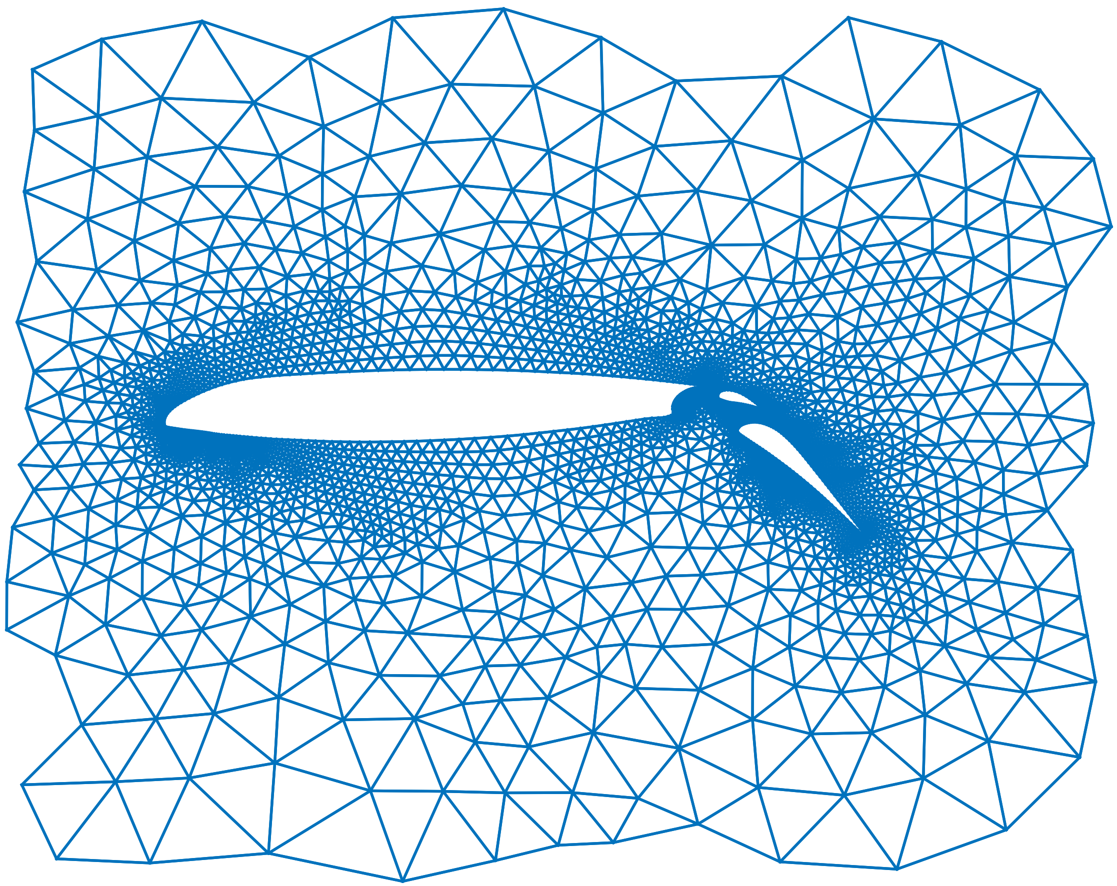
Why Sparsest Cut:

Clustering in graphs
Organizing data



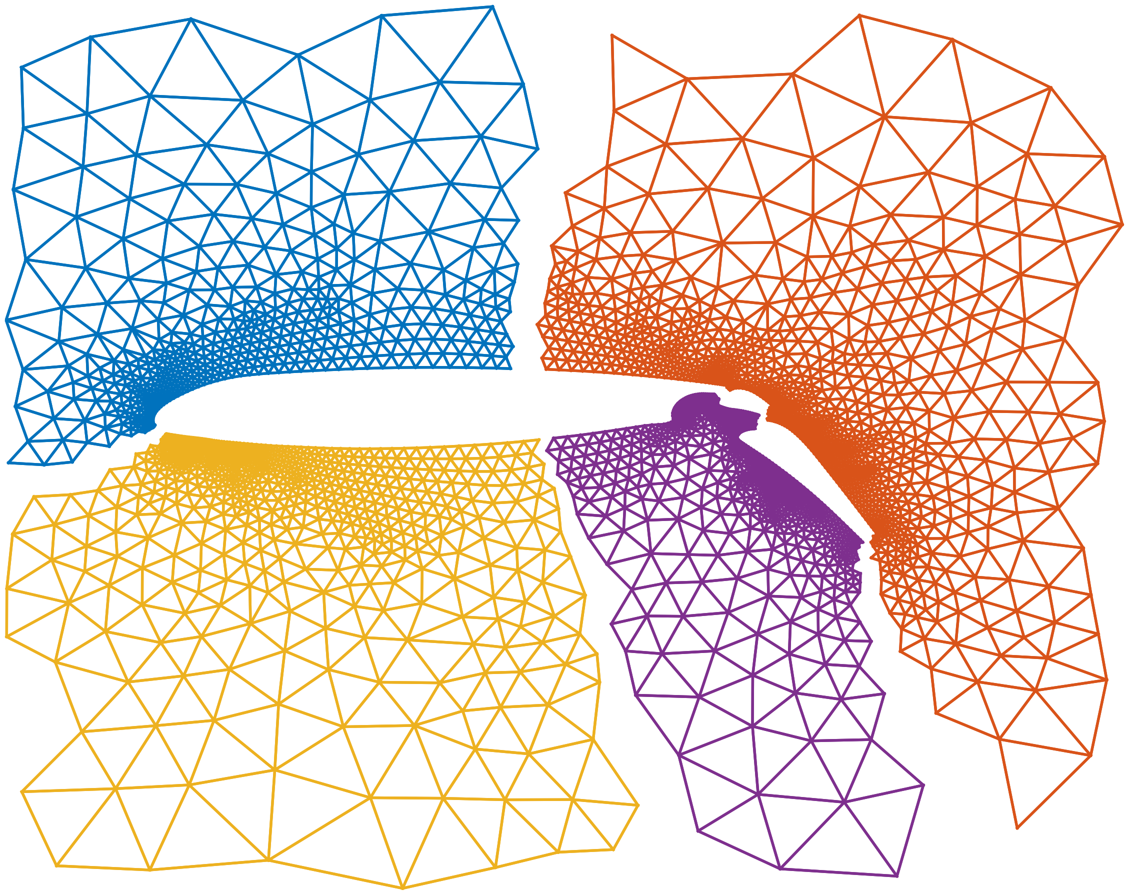
Why Sparsest Cut:

Dividing computation over processors.



Why Sparsest Cut:

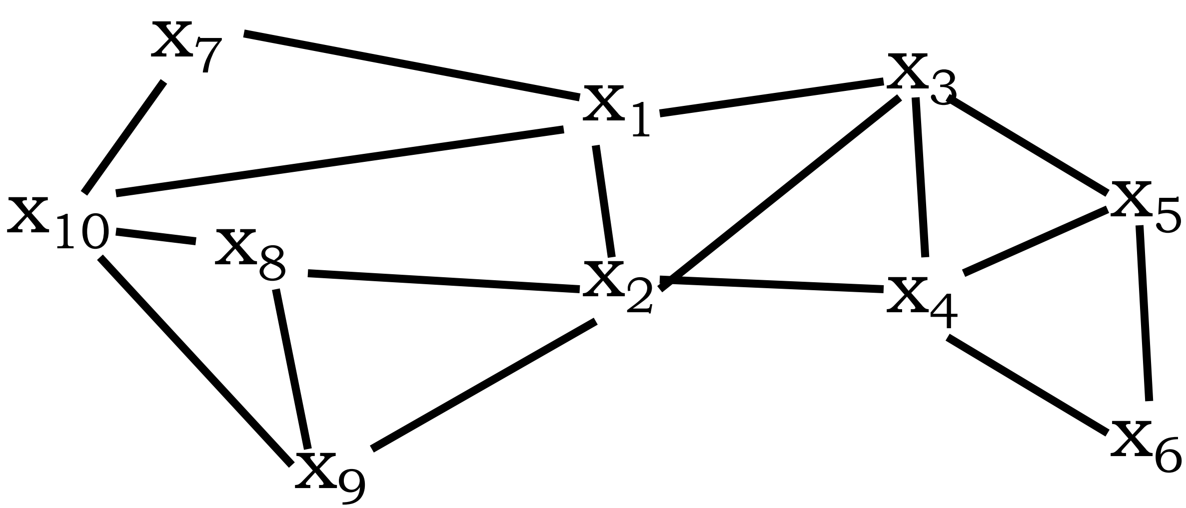
Dividing computation over processors.



Why Sparsest Cut:

Generalized Divide and Conquer (3-SAT)

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

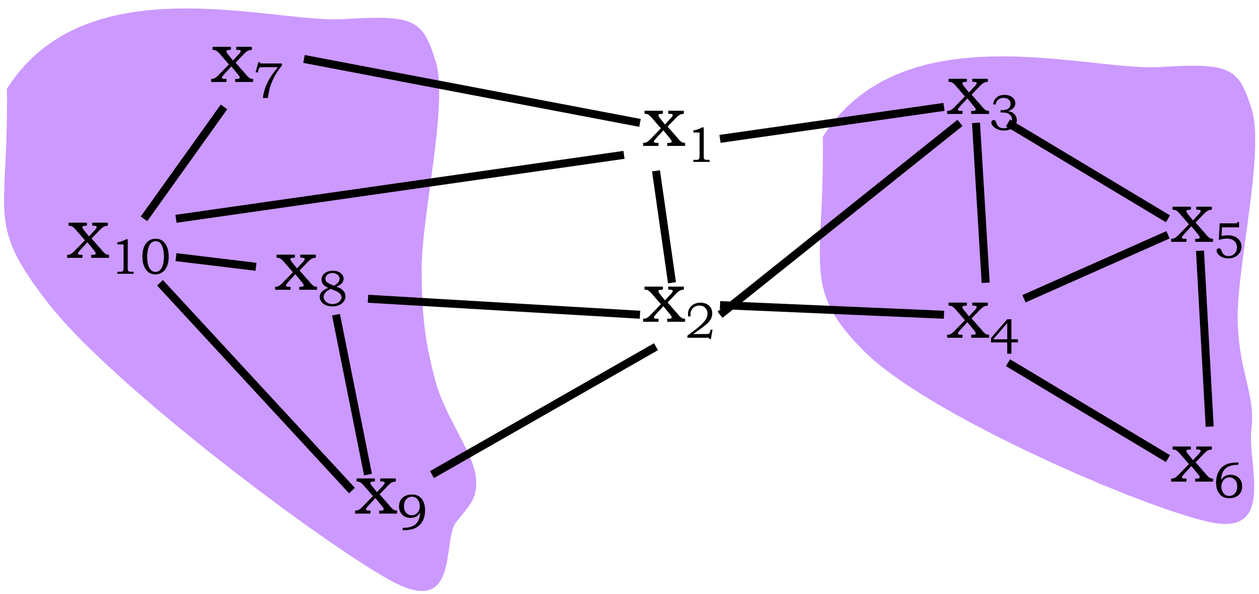


Why Sparsest Cut:

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.

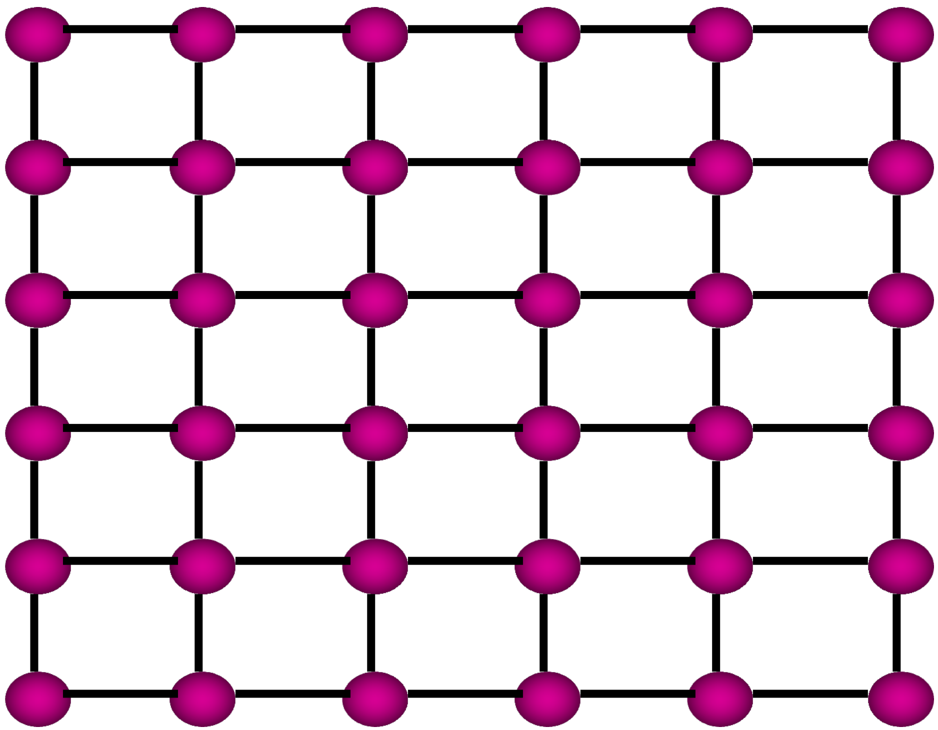
If (1+ε)-approximation,
then SAT is in time $2^{O(n^{\alpha})}$, α < 1

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

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

Multilevel methods

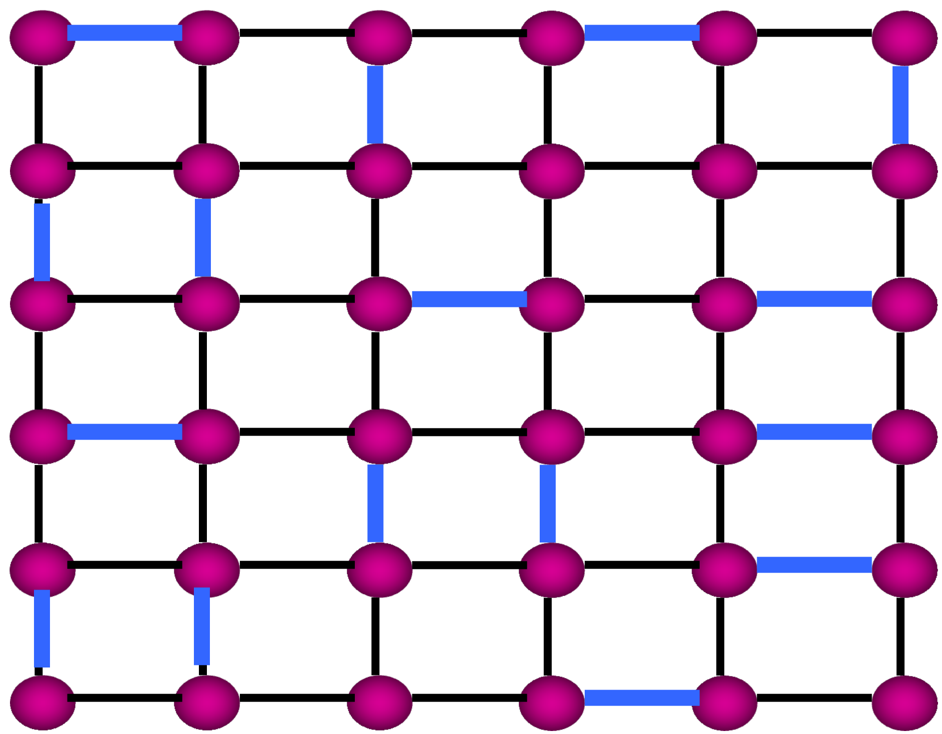
1. Approximate problem by smaller problem.
2. Solve smaller problem (recursively).
3. Lift solution of smaller problem to original.
4. Refine lifted solution.



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.

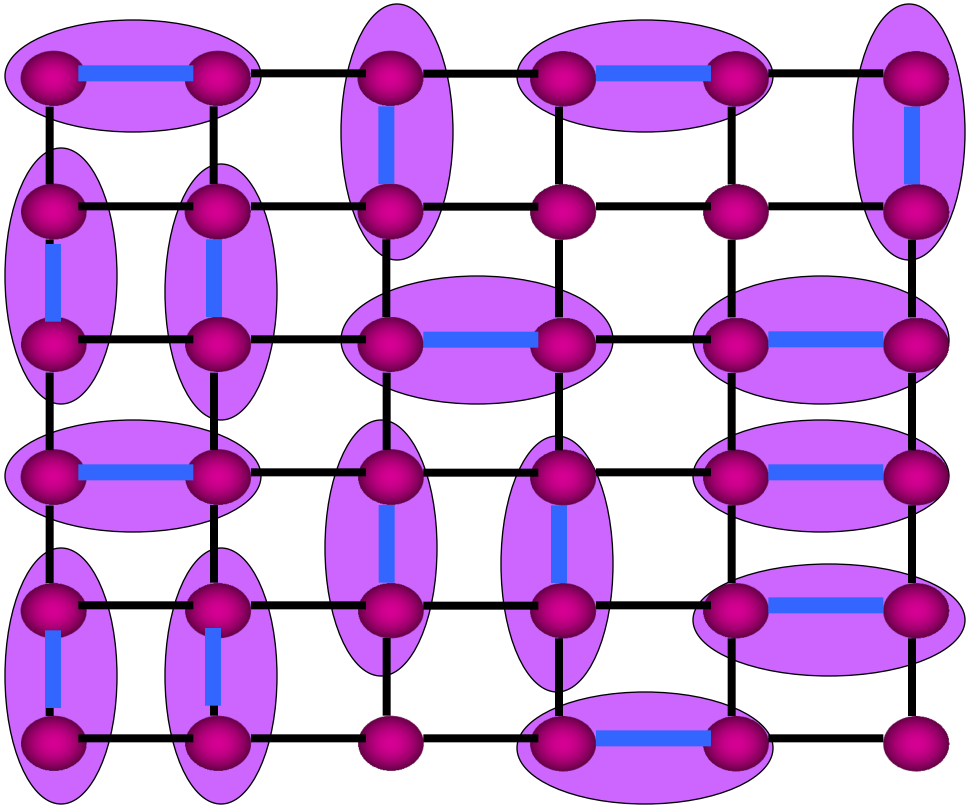
1. Match nodes at random, and merge



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.

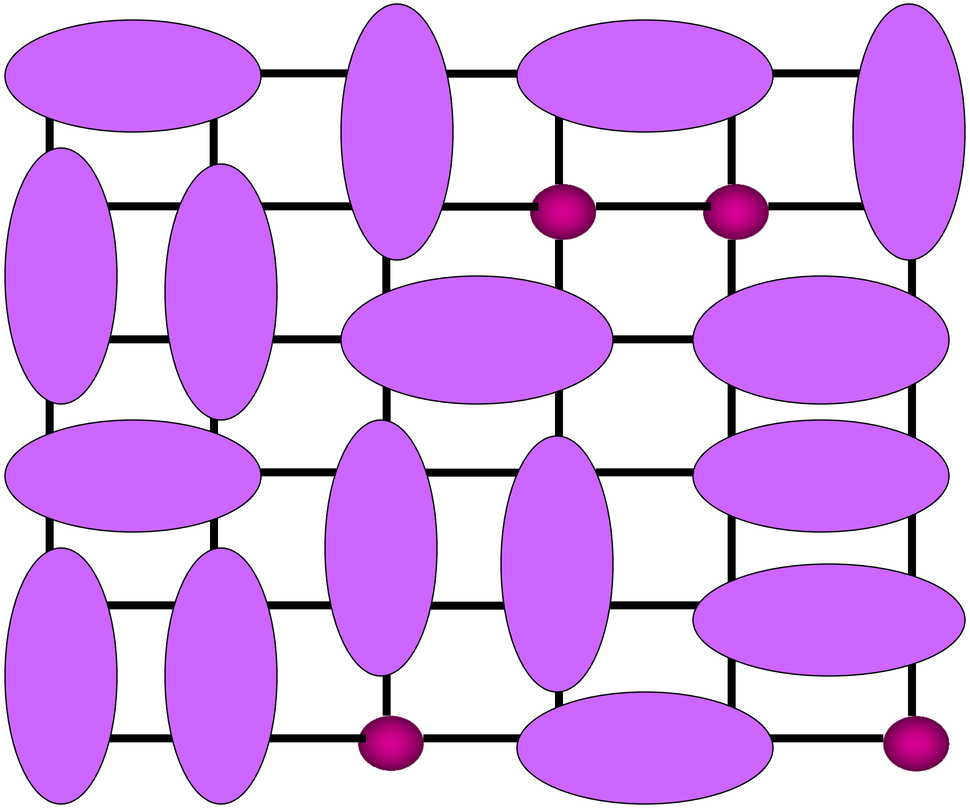
1. Match nodes at random, and merge



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.

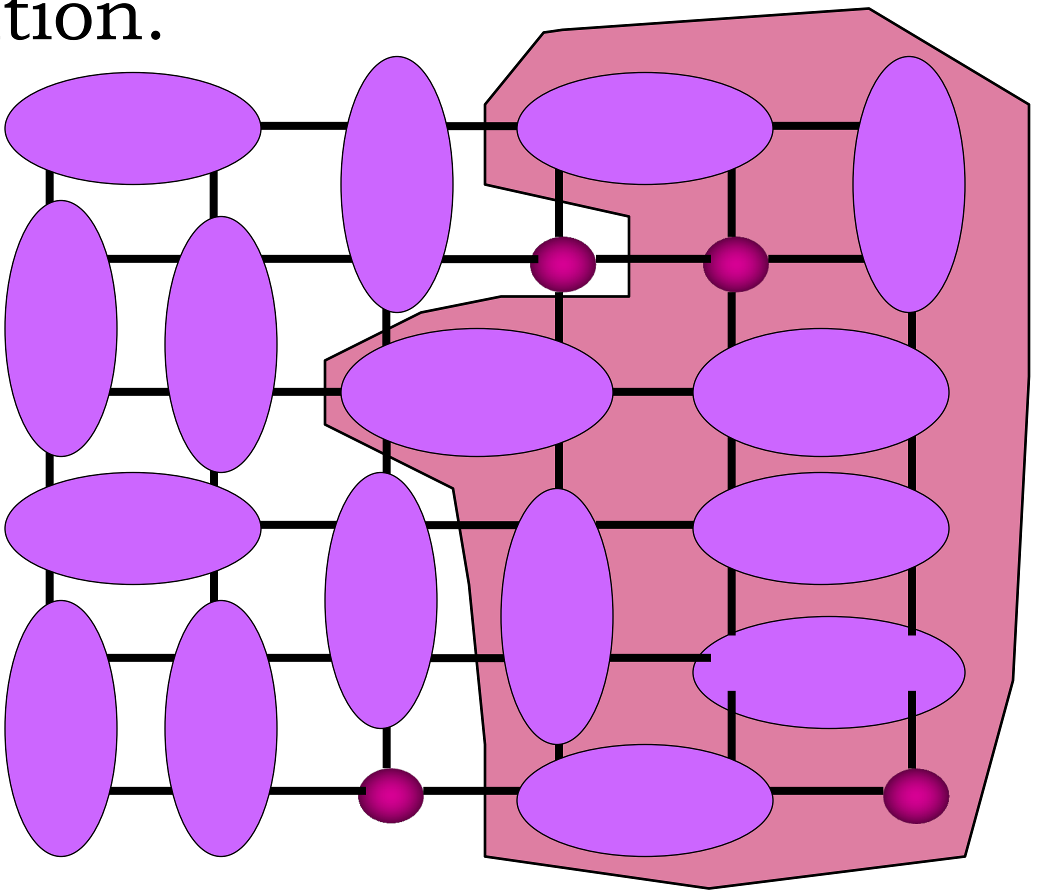
1. Match nodes at random, and merge



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.

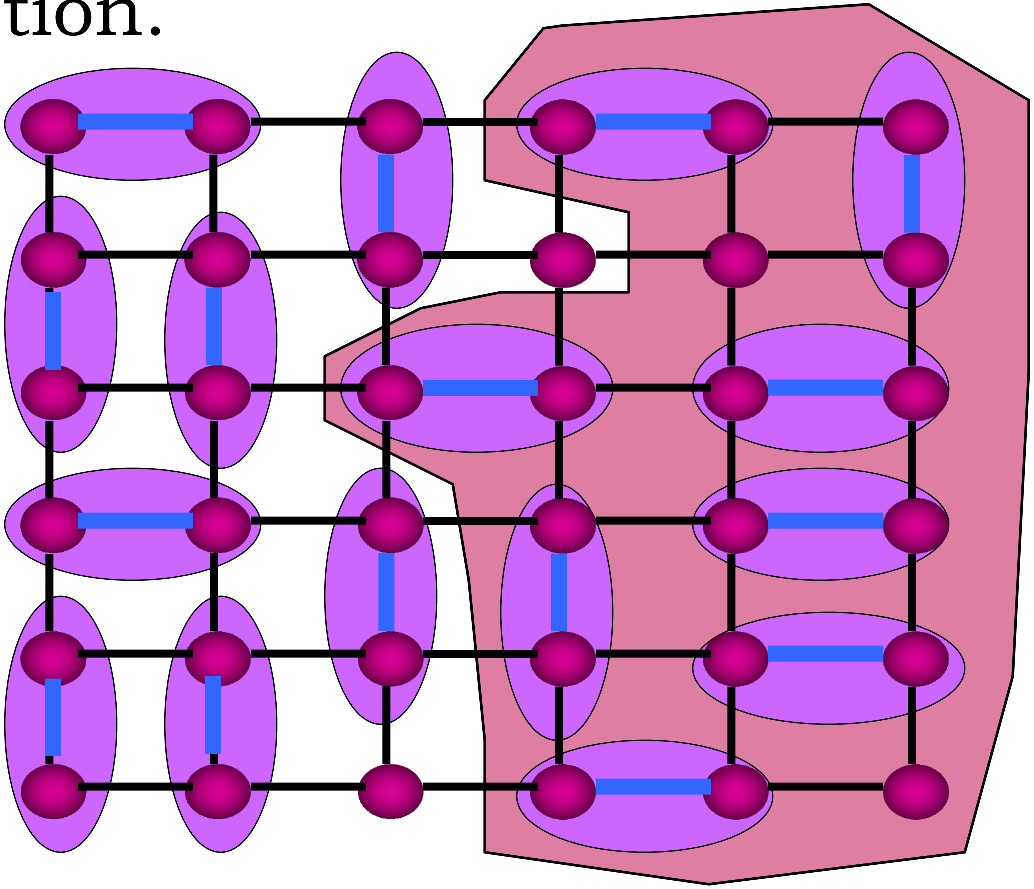
2. Solve sub-problem



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.

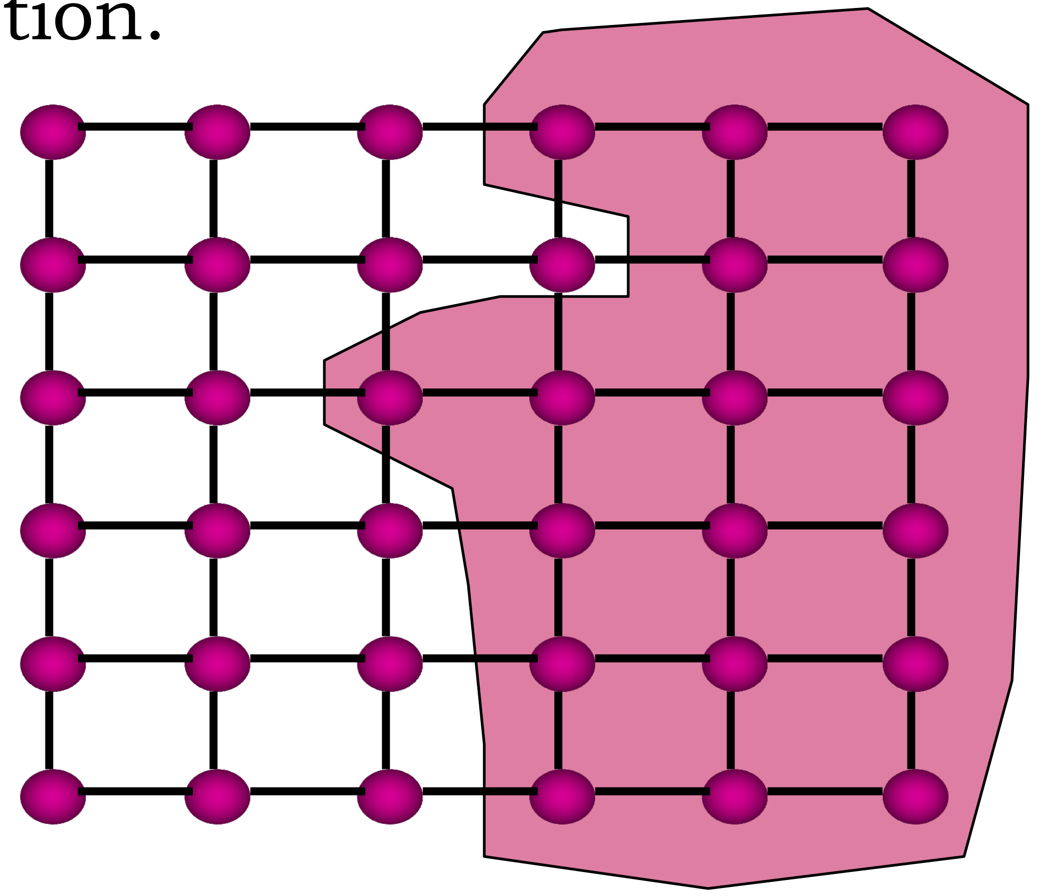
3. Lift solution



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.

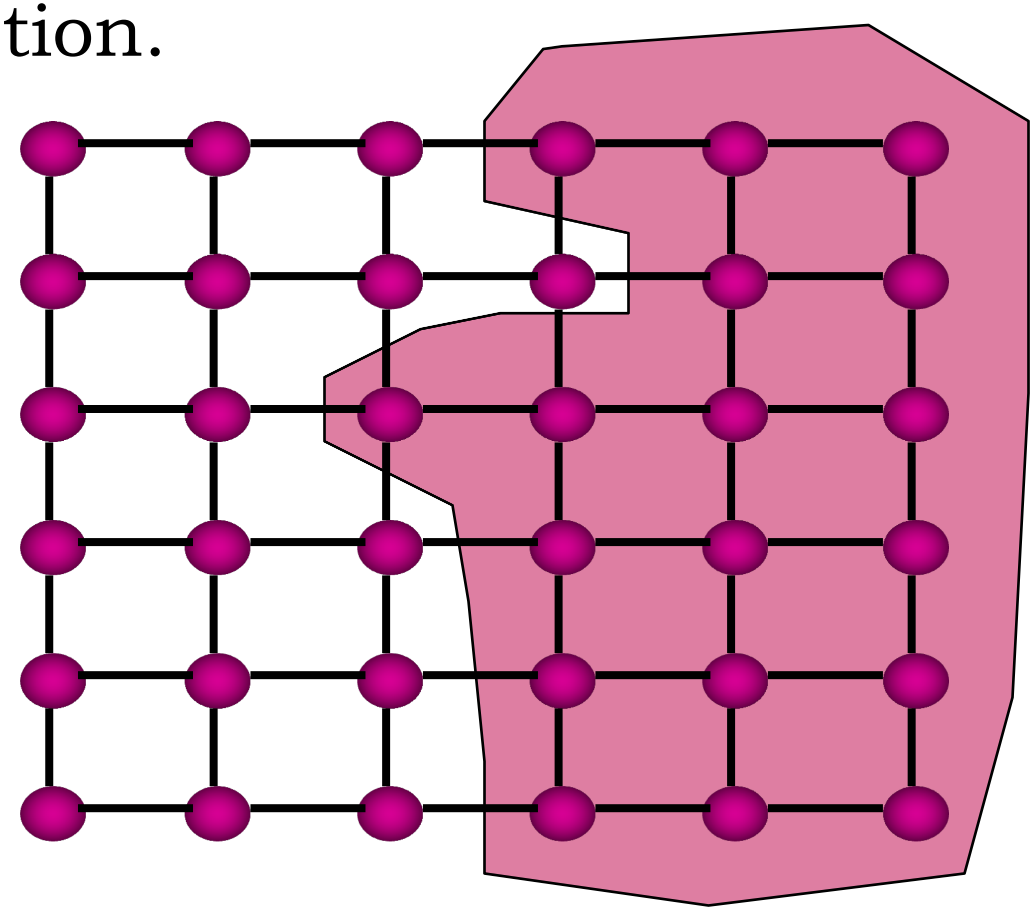
3. Lift solution



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.

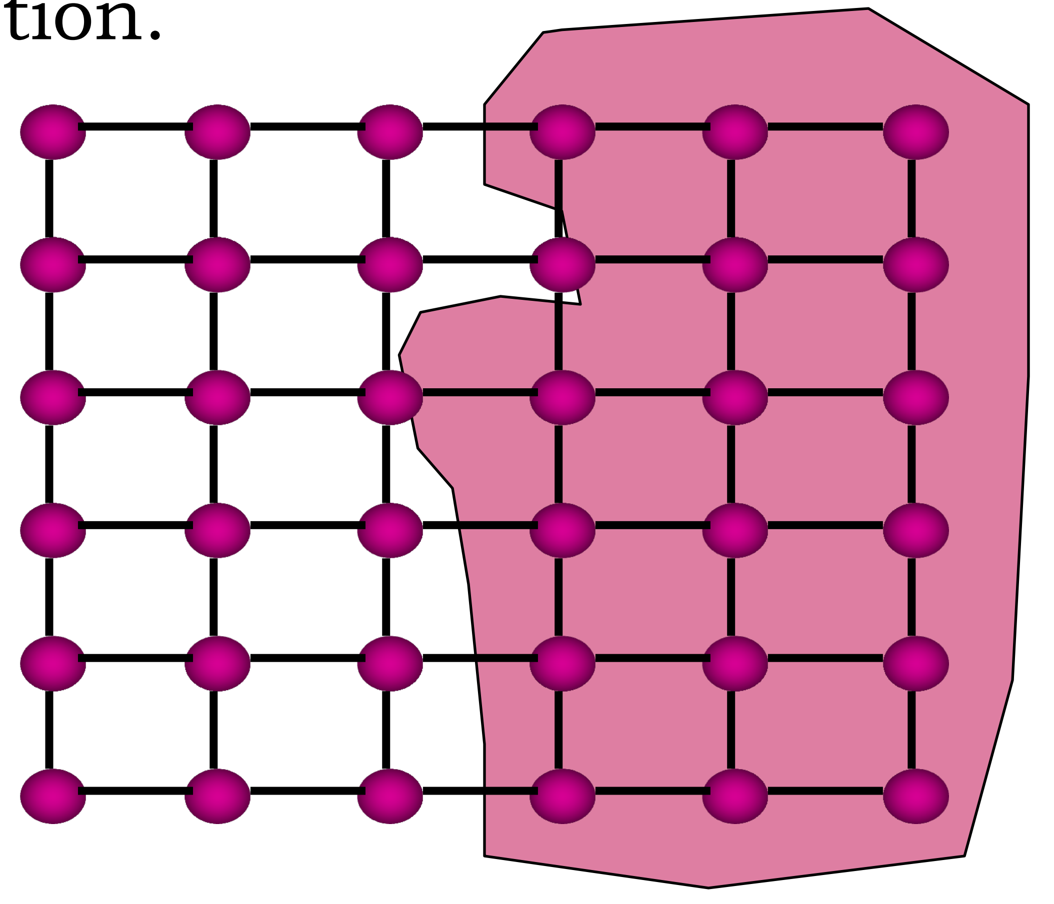
4. Refine solution



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.

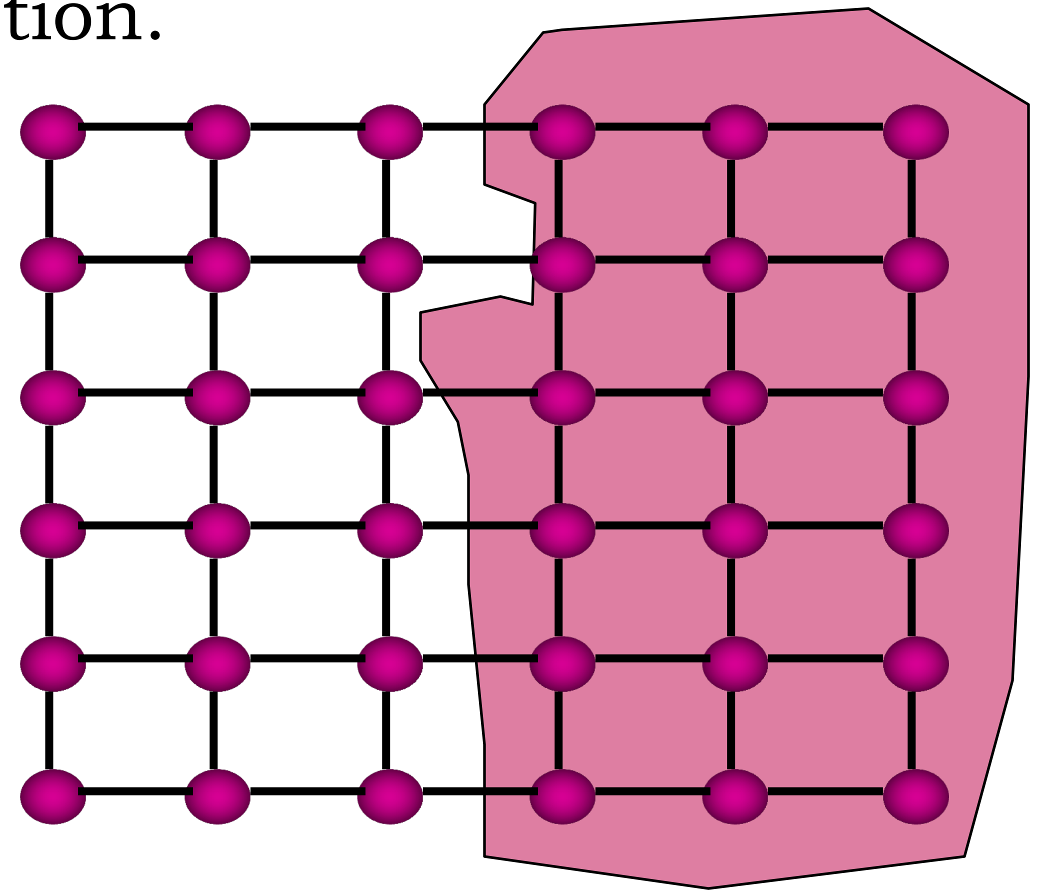
4. Refine solution



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.

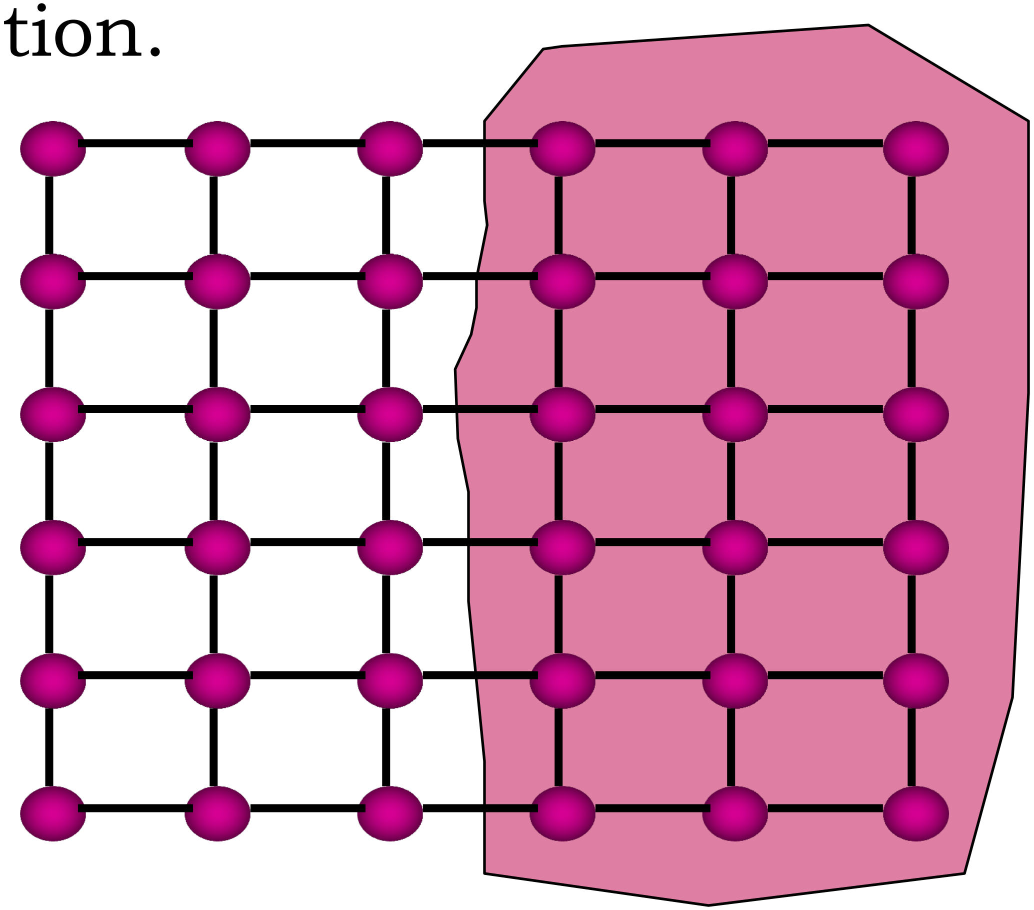
4. Refine solution



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.

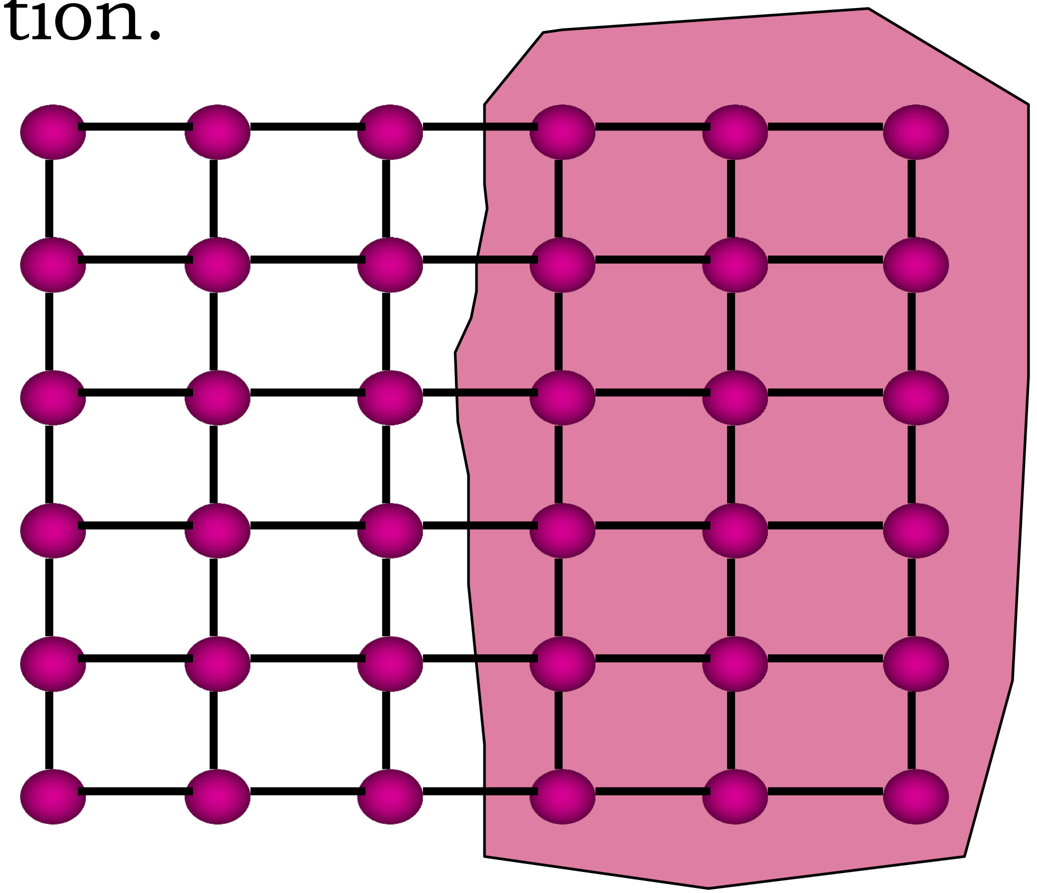
4. Refine solution



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.

Neglected topic: Geometric Algorithms

Convex Hulls

Voronoi Diagrams and Delaunay Triangulations

Meshing

Visibility

Point Location

Geometric Data Structures

Neglected topic: Data structures

Fancy data structures enable fast algorithms.

Hashing

Splay trees (Sleator-Tarjan)
practical binary search trees

Bloom filters

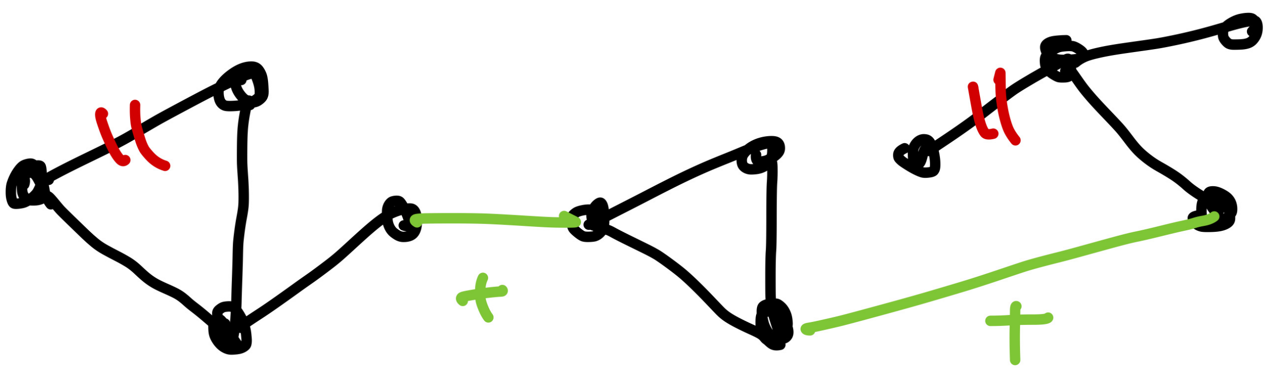
Geometric data structures

Cache efficiency

Neglected topic: Dynamic Algorithms

Maintain answers on changing inputs.

Maintaining components in dynamic graphs.
Time $O(\log^4 n)$ per edge insertion and deletion.
[Kapron-King-Mountjoy '13,
Gibb-Kapron-King-Thorn '14]



Neglected topic: Dynamic Algorithms

Maintain answers on changing inputs.

Maintaining components in dynamic graphs.
Time $O(\log^4 n)$ per edge insertion and deletion.
[Kapron-King-Mountjoy ’13,
Gibb-Kapron-King-Thorn ‘14]

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

Neglected topic: Dynamic Algorithms

Shortest s-t paths (fixed s, all t),
under edge deletions in undirected graphs in
total time $m^{1+o(1)}$

$o(1)$ = decreasing below any positive constant

[Bernstein, Gutenberg, Saranurak '21]

Continuous and Discrete

Maximum Flow in time $m^{1+o(1)}$
[Chen, Kyng, Liu, Peng, Gutenberg, Sachdeva ‘22]

Minimize

$$\Phi(\boldsymbol{f}) \stackrel{\text{def}}{=} 20m \log(\boldsymbol{c}^\top \boldsymbol{f} - F^*) + \sum_{e \in E} \Big((\boldsymbol{u}_e^+ - \boldsymbol{f}_e)^{-\alpha} + (\boldsymbol{f}_e - \boldsymbol{u}_e^-)^{-\alpha} \Big)$$

Uses dynamic 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.

Agrawal-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)

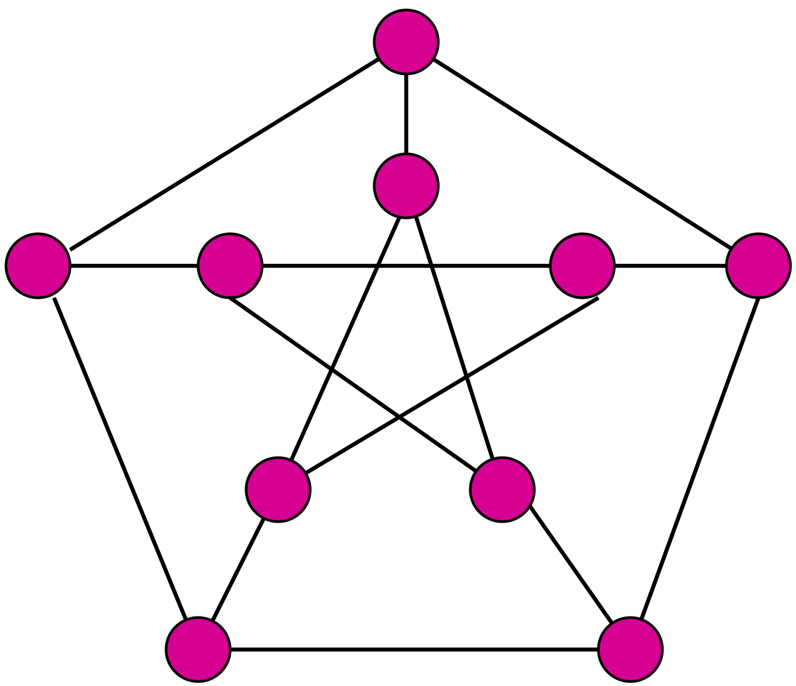
Pollard, Arjen and Hendrik Lenstra,

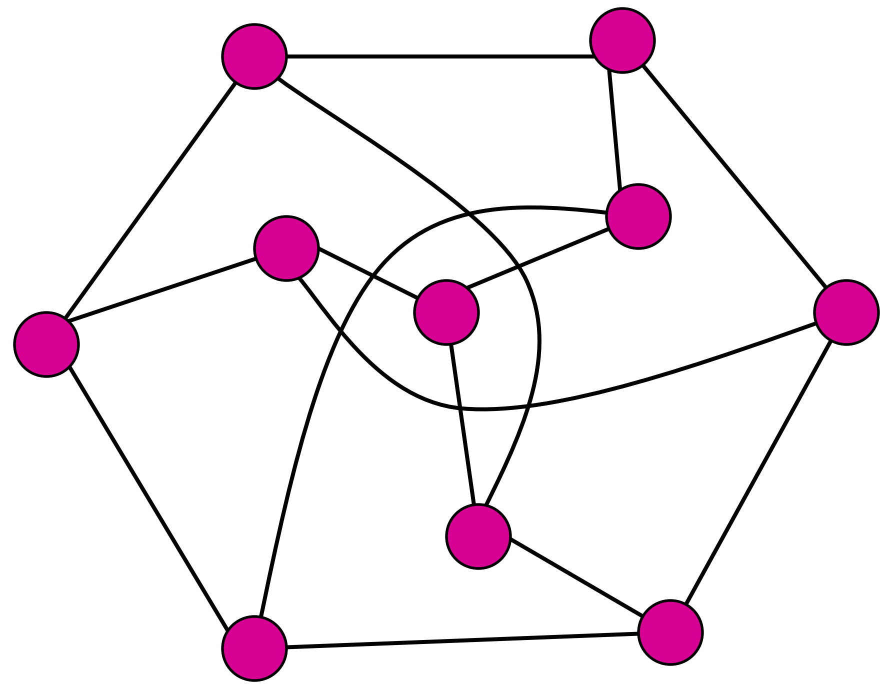
Manasse, Odlyzko, Adelman, Pomerance

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

Hard Problems? Graph Isomorphism.

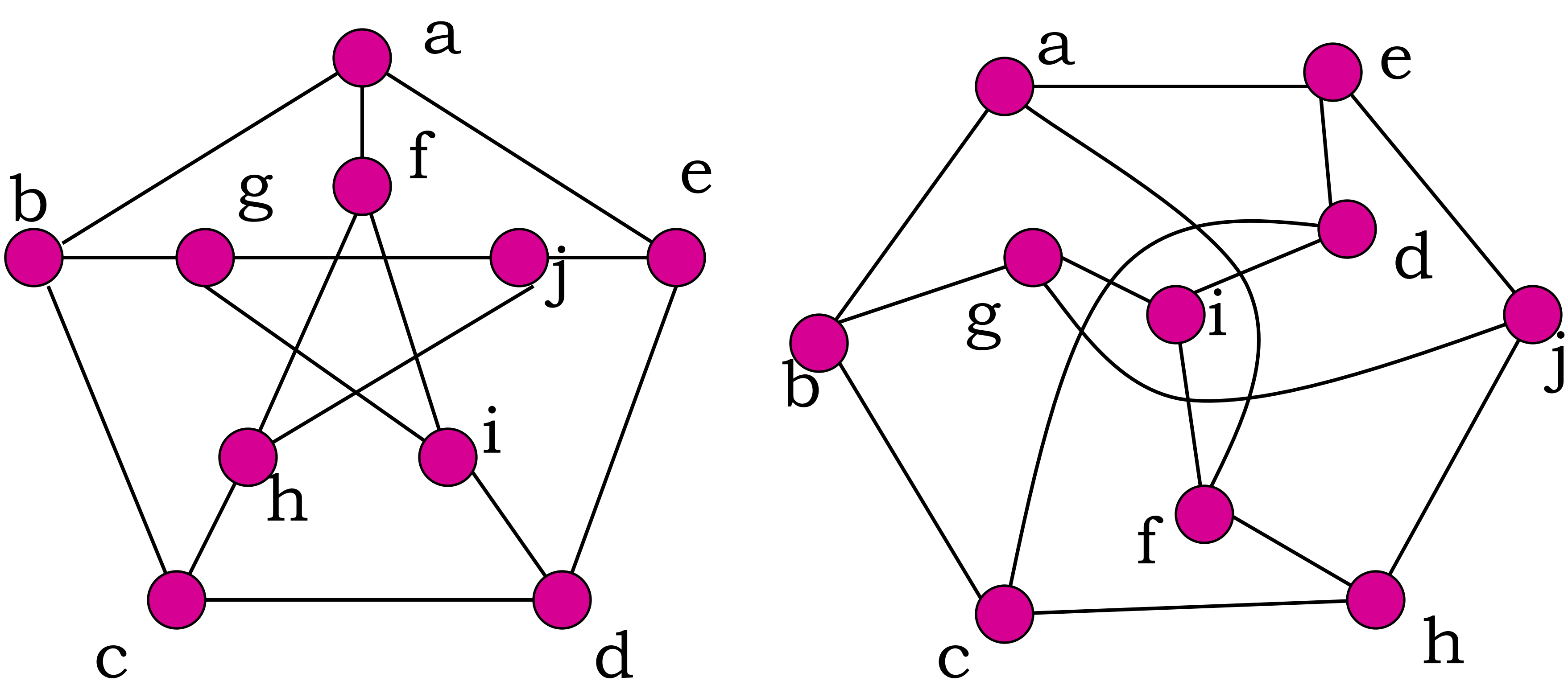
Given two labeled graphs, can vertex sets be relabeled so are same?





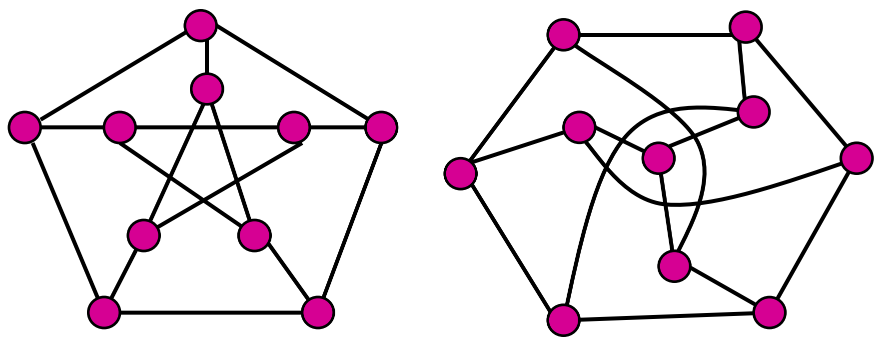
Hard Problems? Graph Isomorphism.

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

But, check out theory of Machine Learning.

Related Courses

366 is the intro to

Theoretical Computer Science

a.k.a. Theory of Computing

More Algorithms

CPSC 464: Algorithms and their Societal Implications (Vishnoi)

CPSC 465: Theory of Distributed Systems (Aspnes)

CPSC 469: Randomized Algorithms (Aspnes)

Continuous Algorithms:

S&DS 431: Optimization and Computation (Yang)

S&DS 432: Advanced Optimization Techniques (Tatikonda)

ENAS 440: Applied Numerical Methods (Bennett)

CPSC 367?

Numerical Linear Algebra (Gilbert)?

More Theory and Theory Adjacent

CPSC 447: Introduction to Quantum Computation (Ding)

CPSC 455: Economics and Computation (Cai)

CPSC 467: Cryptography and Security (Papamanthou)

CPSC 468: Complexity Theory

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:

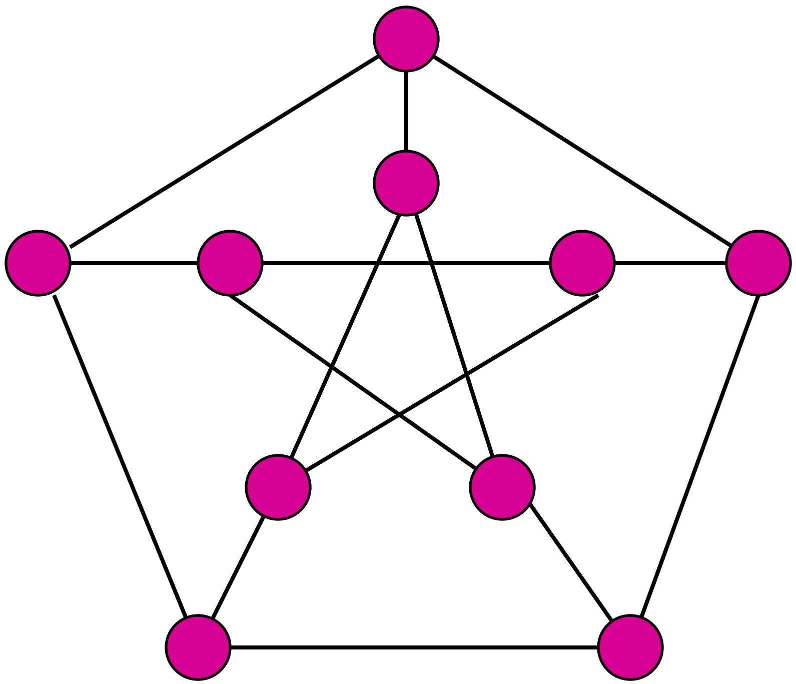
pick one at random,

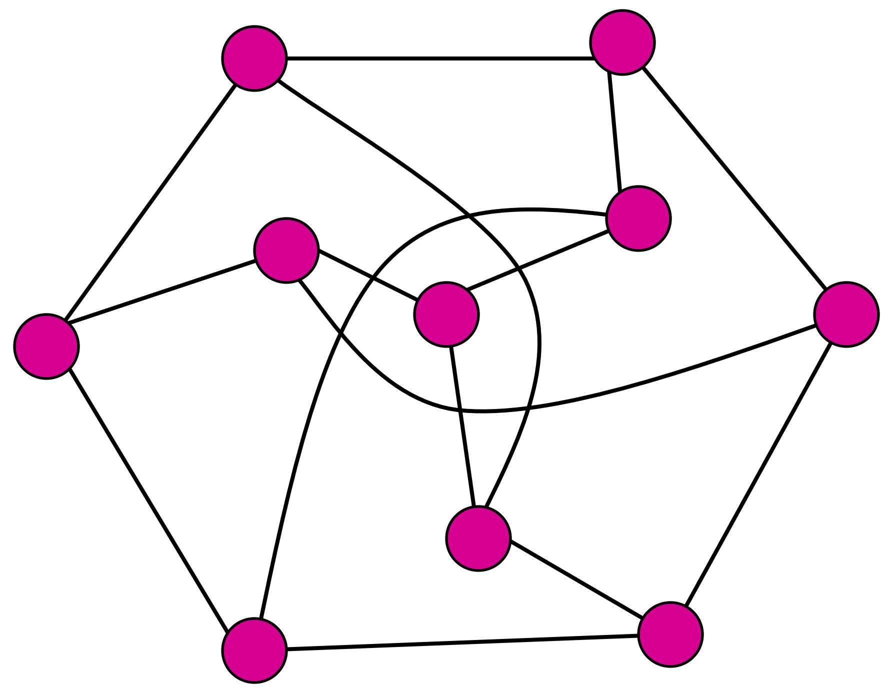
draw it at random,

can I tell you which it was?

Graph non-isomorphism.

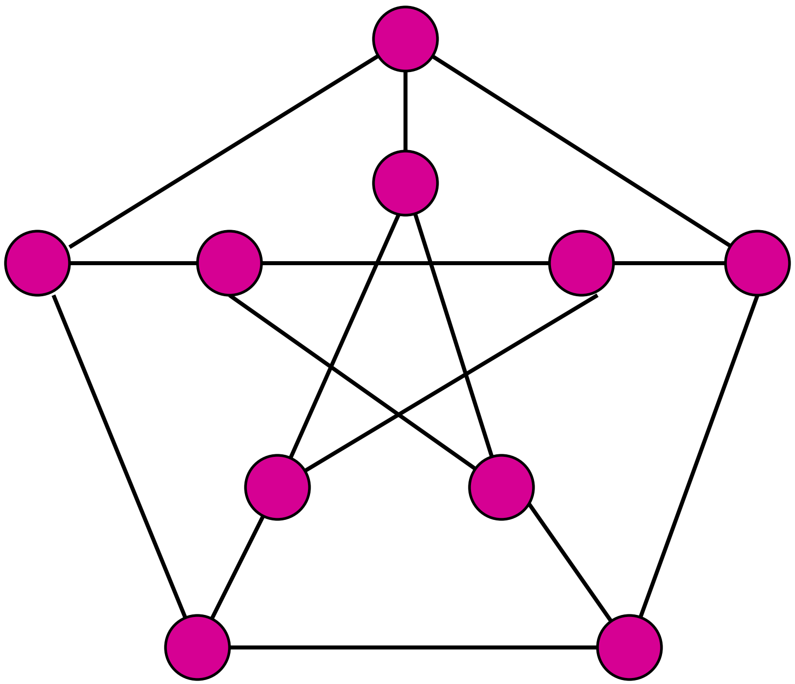
pick one





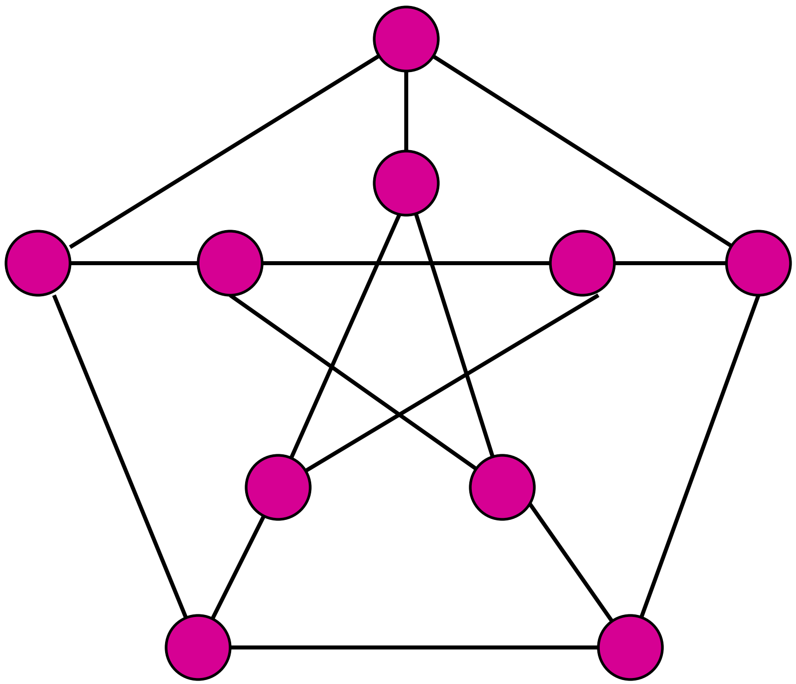
Graph non-isomorphism.

pick one



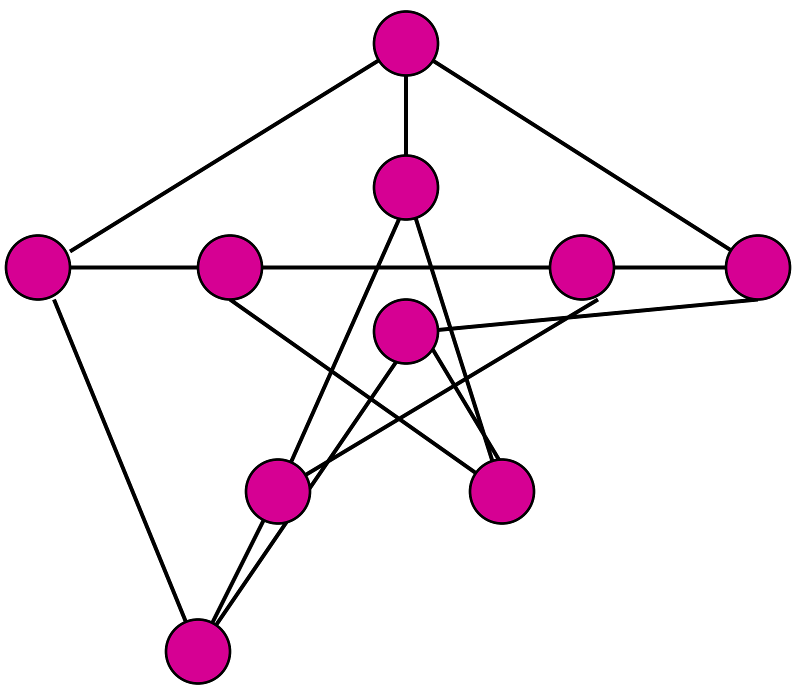
Graph non-isomorphism.

scramble it



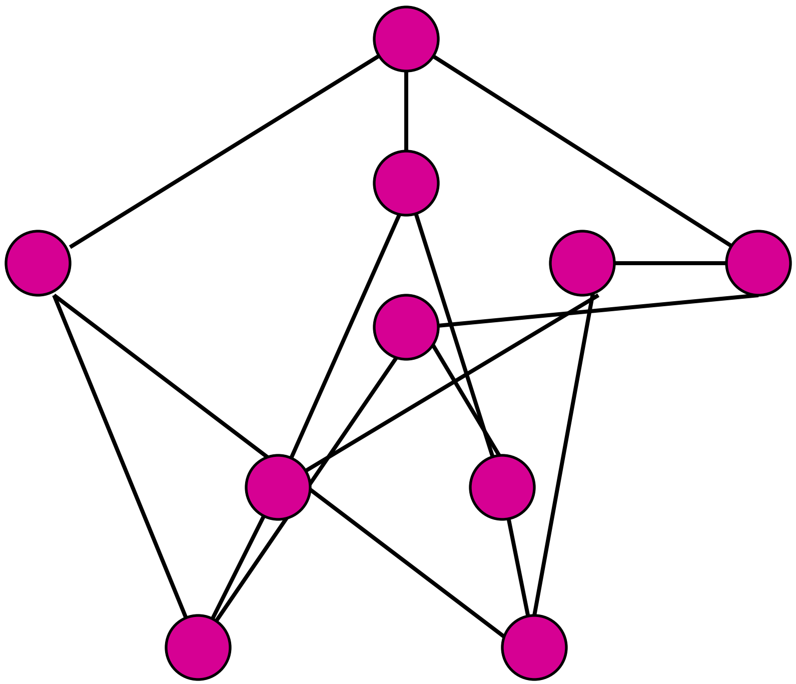
Graph non-isomorphism.

scramble it



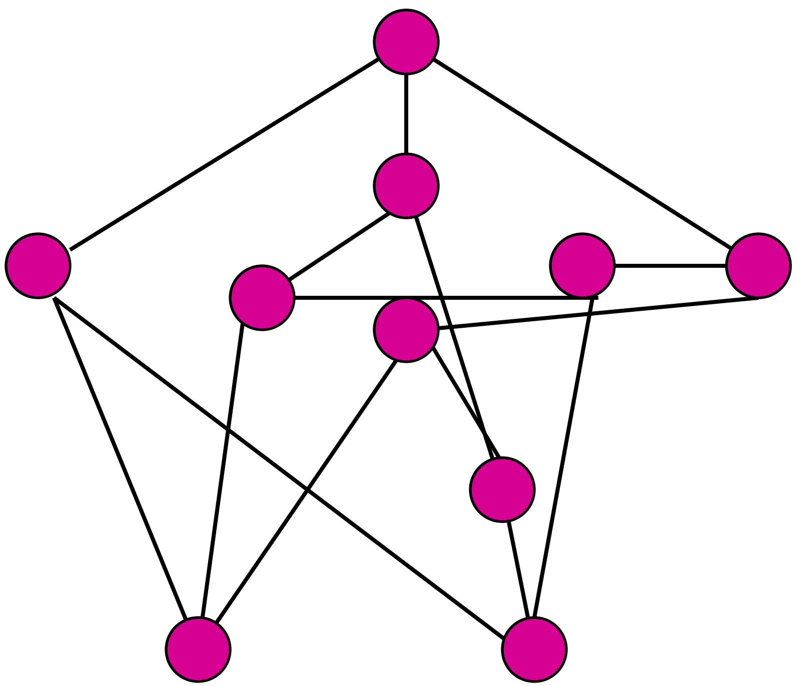
Graph non-isomorphism.

scramble it



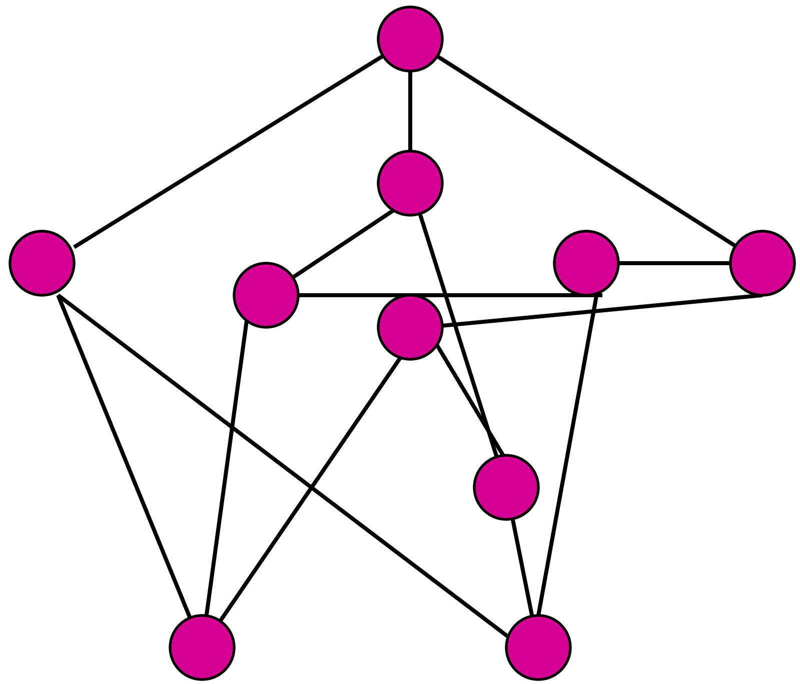
Graph non-isomorphism.

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

Algebraic Topology

Stochastic Calculus

…

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)
HALG (Highlights of Algorithms)

Where to learn more

arXiv and blogs: http://cstheory-feed.org/

Class lecture notes.

Video lectures: https://sites.google.com/view/tcsplus/

SIMONS INSTITUTE
for the Theory of Computing

Quanta Magazine
Simons Foundation MPS Articles
Communications of the ACM

Just keep learning