CSE 421
Introduction to Algorithms

Lecture 29:

What not to do to deal with NP-completeness

Wrap Up

Reminder/Announcement

- The Final Exam is Monday December 9, 2:30-4:45 pm here
- See the pinned Edstem posts for Final Exam Information.
- I will run a Zoom review session on Sunday starting at 2:00 pm.
  - PLEASE BRING YOUR QUESTIONS!
- The reference sheet you will receive on the exam will be available later today.

Other Heuristic Algorithms you might hear about

Genetic algorithms:

- View each solution as a **string** (analogy with DNA)
- Maintain a **population of good solutions**
- Allow **random mutations** of single characters of individual solutions
- **Combine two solutions** by taking part of one and part of another (analogy with crossover in sexual reproduction)
- Get rid of solutions that have the worst values and make multiple copies of solutions that have the best values (analogy with natural selection -- survival of the fittest).

*Usually very slow. In the rare cases when they produce answers with better objective function values than other methods they tend to produce very **brittle** solutions – that are very bad with respect to small changes to the requirements.*

Deep Neural Nets and NP-hardness?

- **Artificial neural networks**
  - based on very elementary model of human neurons
  - **Set up a circuit of artificial neurons**
    - each artificial neuron is an analog circuit gate whose computation depends on a set of **connection strengths**
  - **Train the circuit**
    - Adjust the connection strengths of the neurons by giving many positive & negative training examples and seeing if it behaves correctly
  - **The network is now ready to use**

Despite their wide array of applications, they have not been shown to be useful for NP-hard problems.

Quantum Computing and NP-hardness?

Use physical processes at the quantum level to implement "weird" kinds of circuit gates based on unitary transformations

- Quantum objects can be in a "superposition" of many pure states at once
  - Can have n objects together in a superposition of 2^n states
- Each quantum circuit gate operates on the whole superposition of states at once
  - Inherent parallelism but classical randomized algorithms have a similar parallelism: *not enough on its own*
  - Advantage over classical: copies interfere with each other.
- Exciting direction - theoretically able to factor efficiently.
  Major practical problems wrt errors, decoherence to be overcome.
- *Small brute force improvement but unlikely to produce exponential advantage for NP.*

Where we have been…

Problems and major solution paradigms

- Stable Matching
- Graph Traversal
- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
- Network Flow
- Linear Programming
  - Focus on encoding as LPs as opposed to how to solve them.
- NP-completeness
- Approximation algorithms & other ways of side-stepping NP-hardness

How to use these ideas

- 1st : See if your problem is a special case/close to/reminds you of one of the problems we have considered in the class
- 2nd : Try these:
  - Graph traversal
  - Greedy algorithms
    - Be **skeptical**! Your first greedy idea is probably wrong; maybe all greedy approaches are wrong. Proving correctness is critical.
- 3rd: Try to solve it recursively w/ smaller subproblems of the same type
  - If subproblems are a constant factor smaller: Divide and Conquer
  - If same subproblems show up repeatedly: Dynamic Programming
    - See how the pattern of recursion/subproblems matches example patterns you already know. Maybe try a new one.

How to use these ideas

- 4th: See if you can express your problem as a Flow/Cut/Matching.
- 5th : Try Linear Programming
- 6th : Maybe your problem is NP-hard. Check out lists of NP-hard problems to see if yours is there, or try to show it directly.
- 7th : If your problem is NP-hard, try to side-step that hardness.

There are other methods we have barely touched on and

getting some polynomial-time algorithm isn't the end of the story…

… we have barely touched on the subject.

Methods: Randomized Algorithms

Algorithms that make random choices can often be made simpler than deterministic ones. We considered QuickSelect but there is MUCH more than that:

Really neat example:

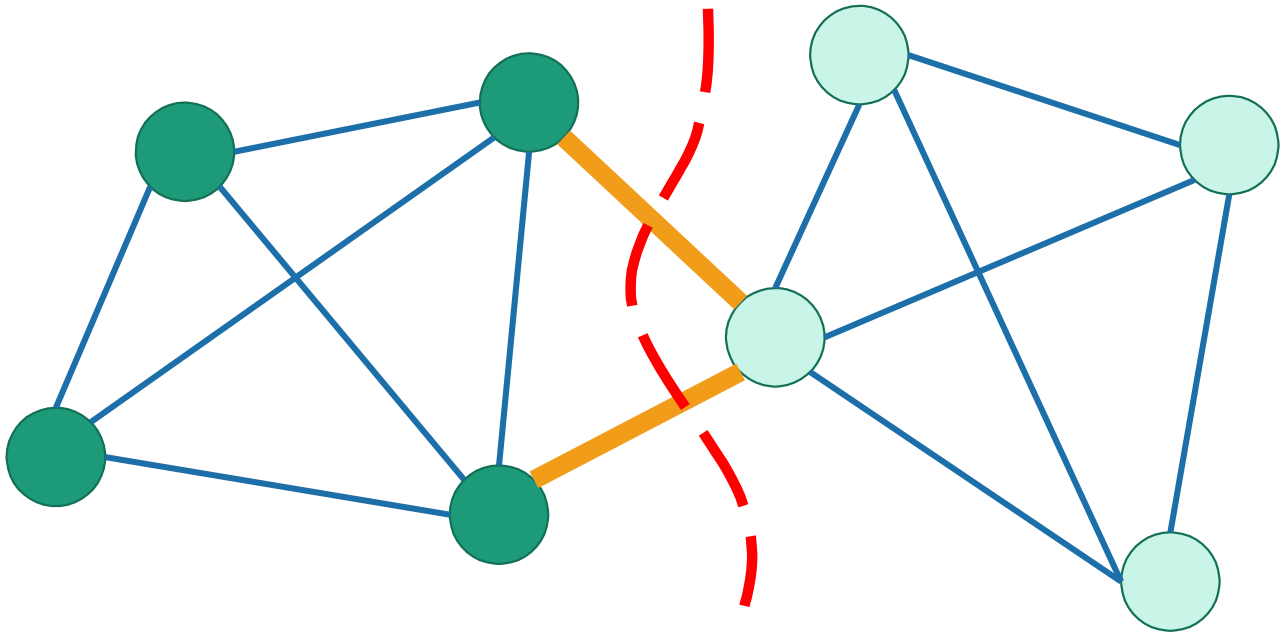
Global Minimum Cut: Find a minimum size set of edges that disconnects an undirected graph. (Even well-defined for multigraphs with parallel edges.)

A solution: Run MaxFlow/MinCut algorithm for every st-pair. Slow and complicated…

Related problem: Find *all* global minimum cuts.

- Not even clear how many there might be or how to find them all!

Global Minimum Cut



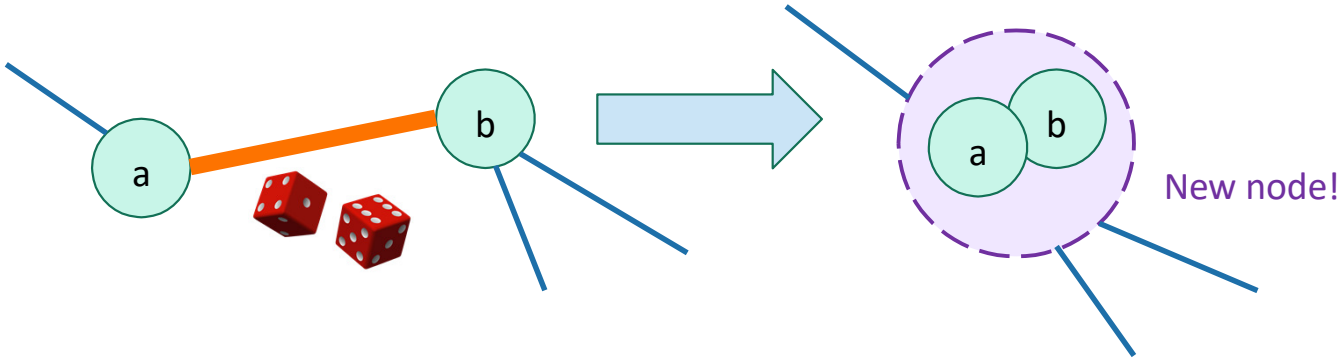
Karger's Randomized Algorithm

- Finds **global minimum cuts** in undirected graphs
- Randomized algorithm
- Karger's algorithm **might be wrong**.
  - Compare to QuickSelect, which just might be slow.

- Its success depends on the random choices:
  - **A single run won't succeed in getting a true minimum that often, but it succeeds often enough that we can run it many times and take the smallest cut we find and be almost surely right.**
- It is really simple!

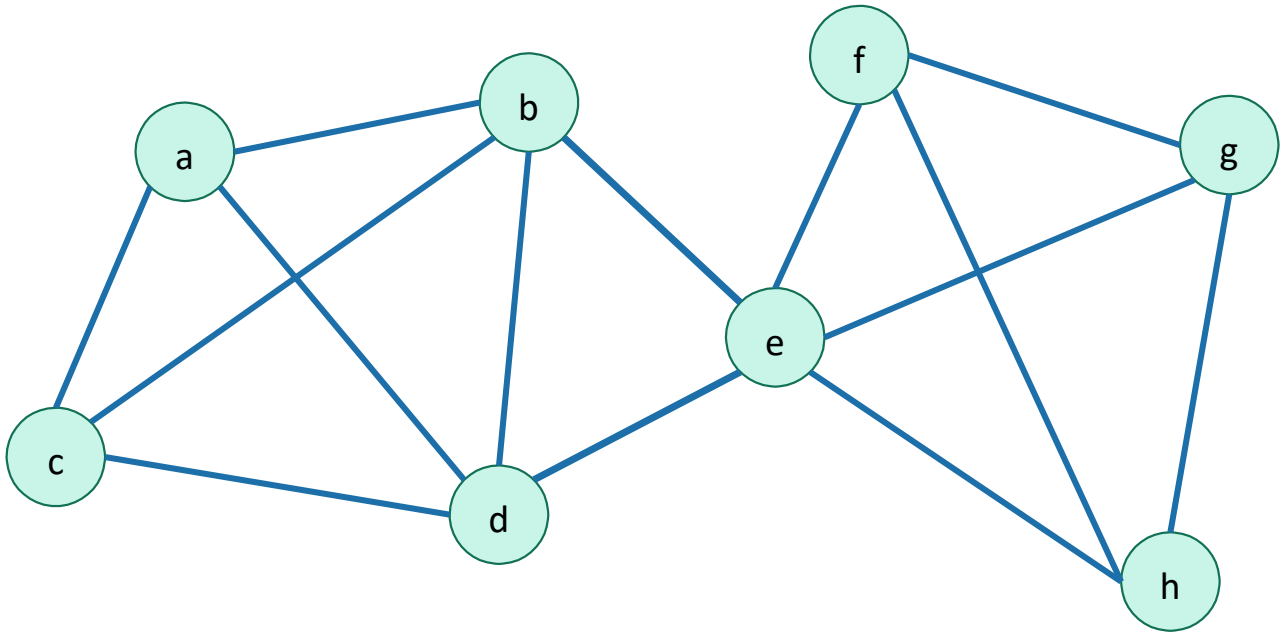
Karger's Algorithm

- Pick a random edge.
- **Contract** it.
- Repeat until you only have two vertices left.
- Use those two vertices to define a cut.

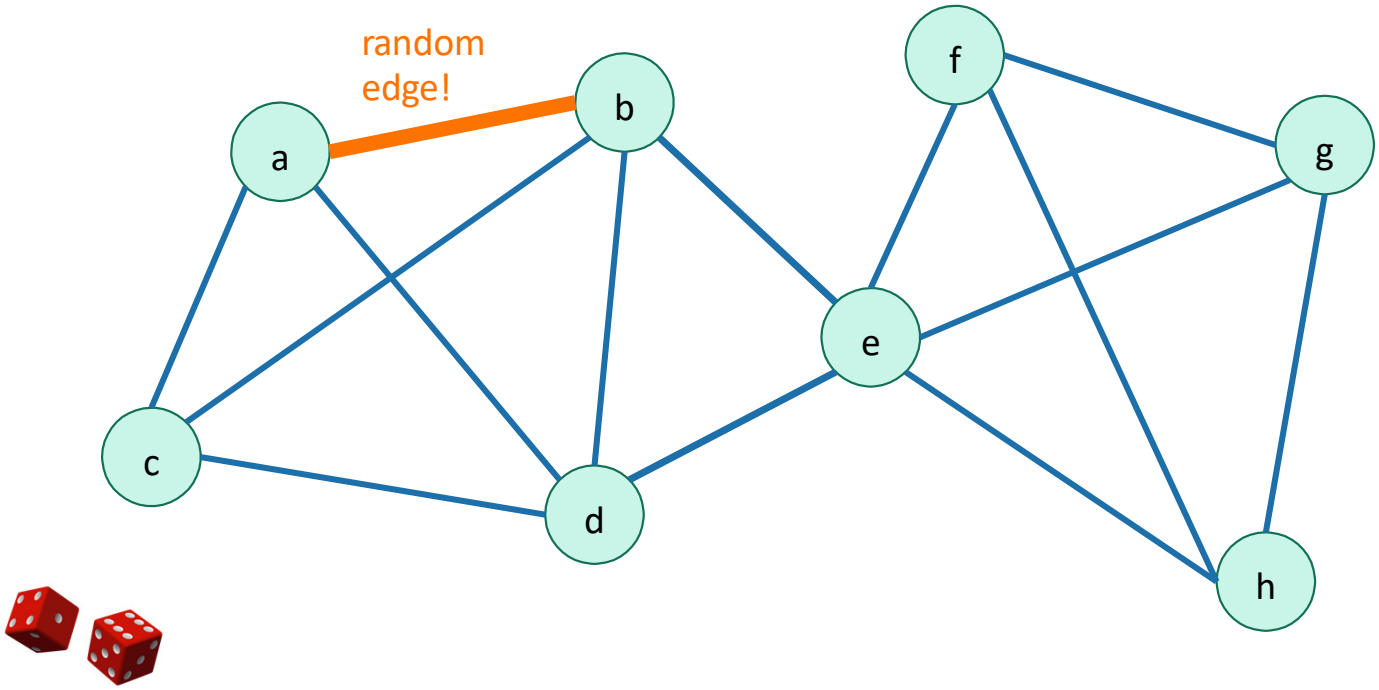


Why is this a good idea? We'll see shortly.

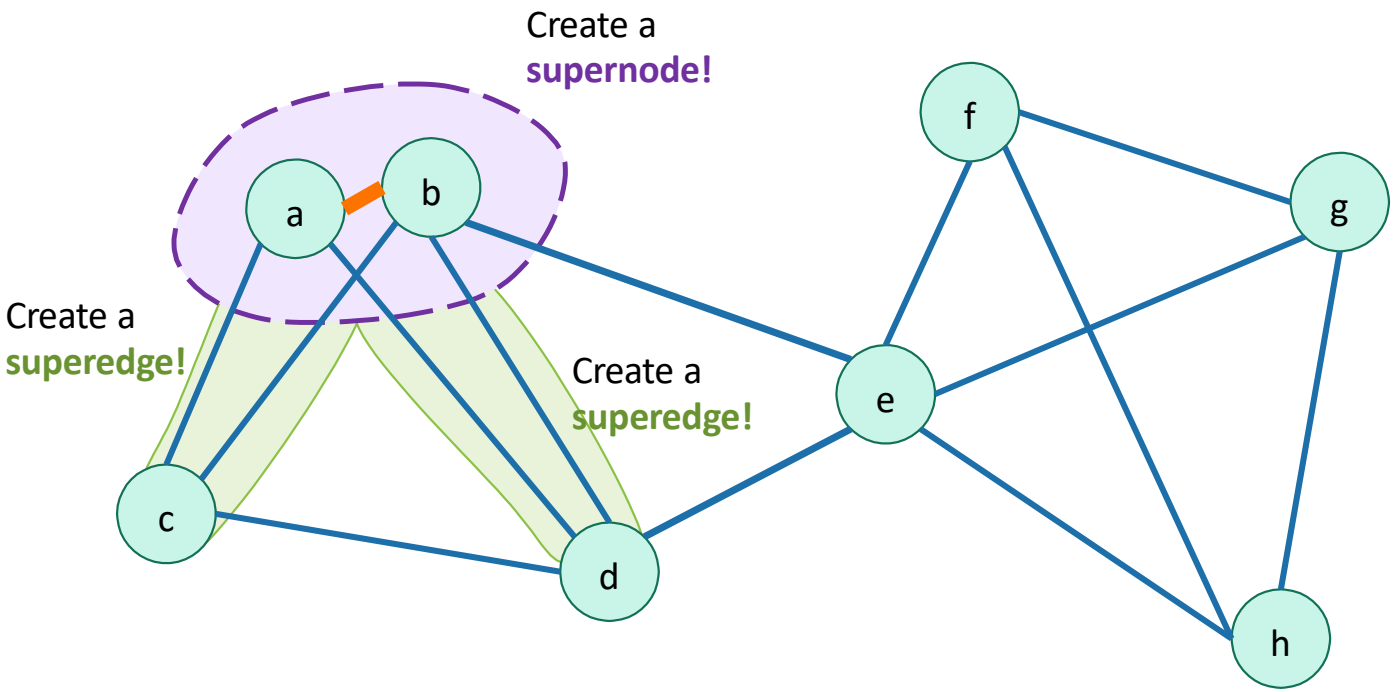
Karger's Algorithm



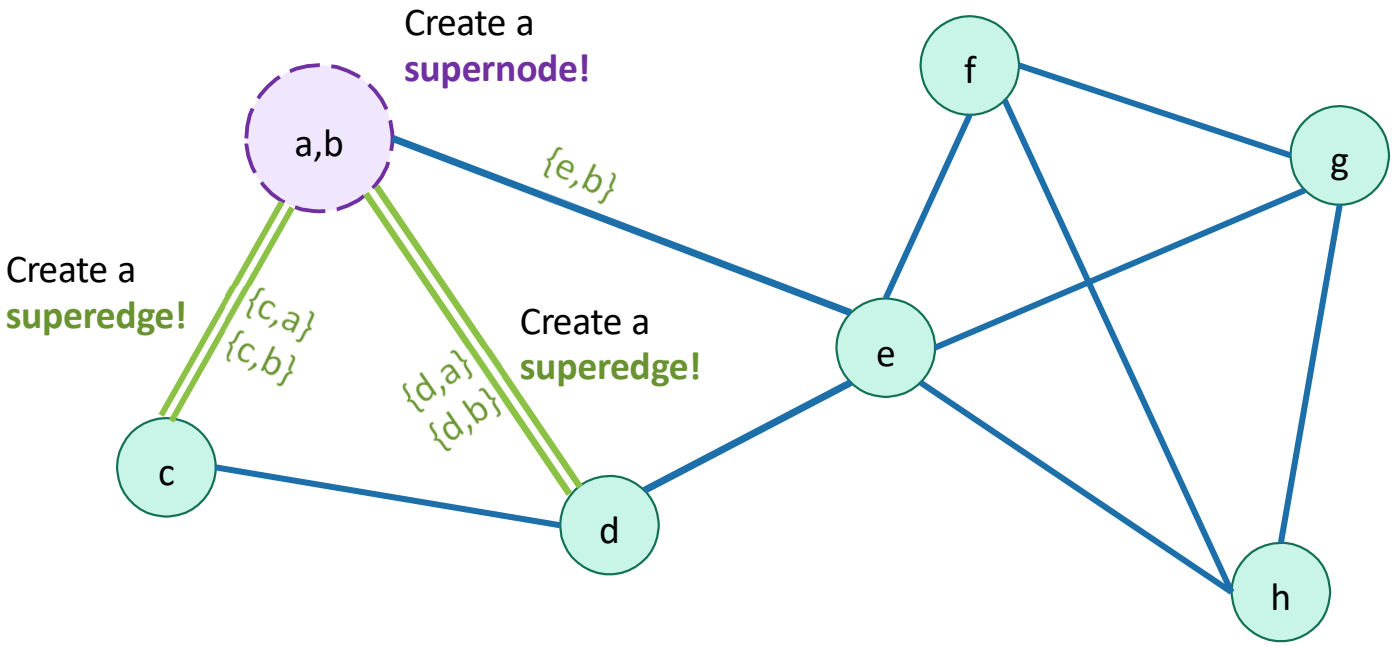
Karger's Algorithm



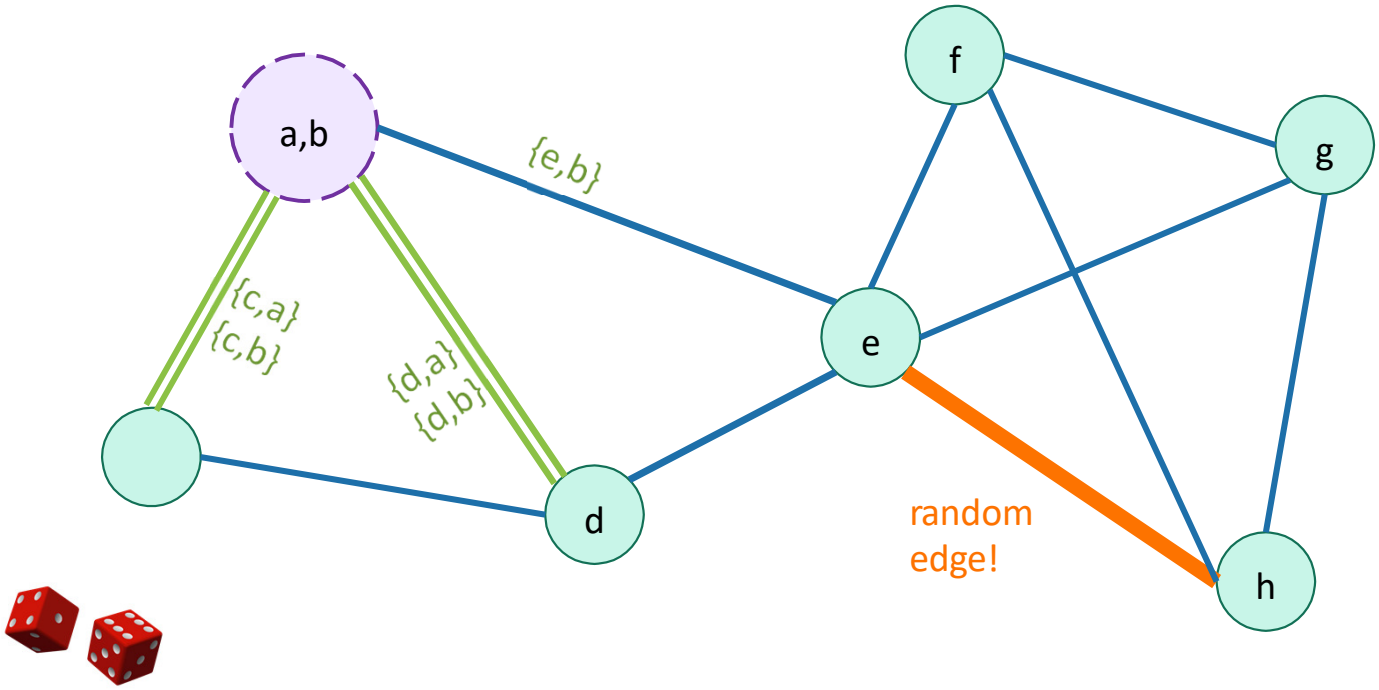
Karger's Algorithm



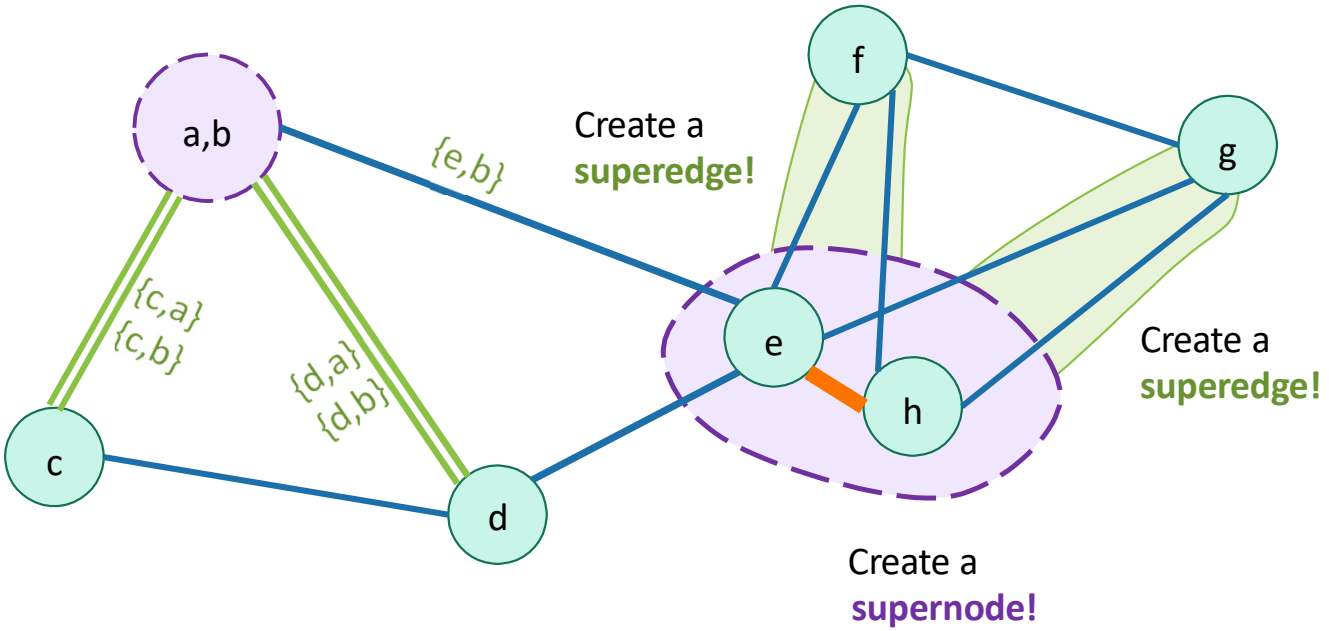
Karger's Algorithm



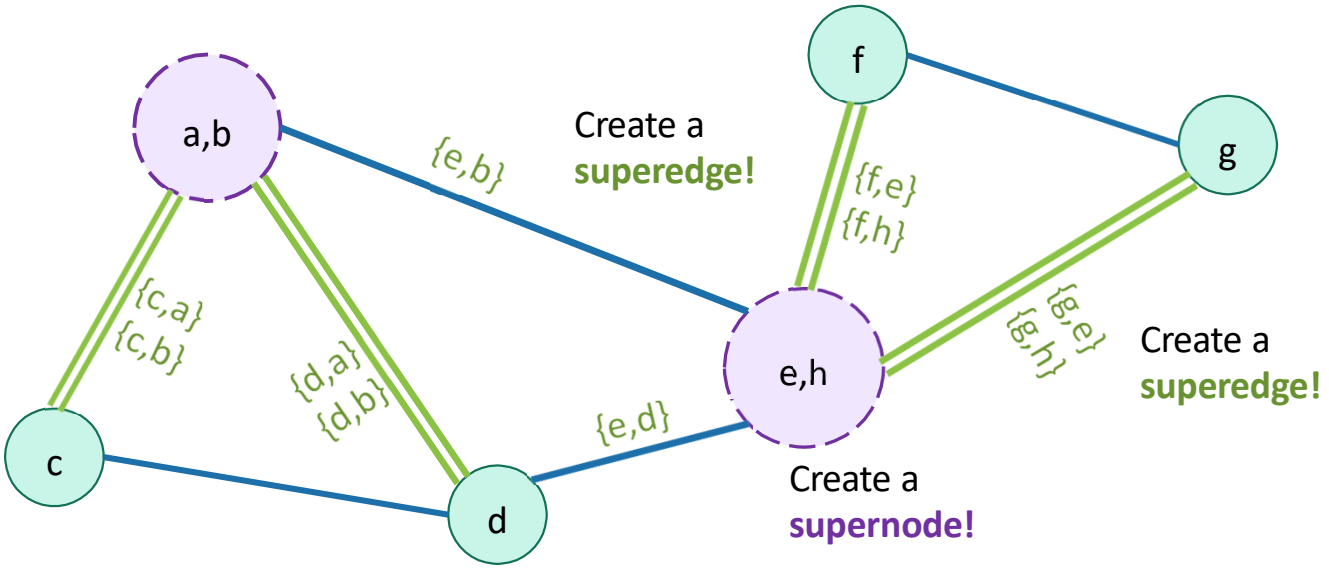
Karger's Algorithm



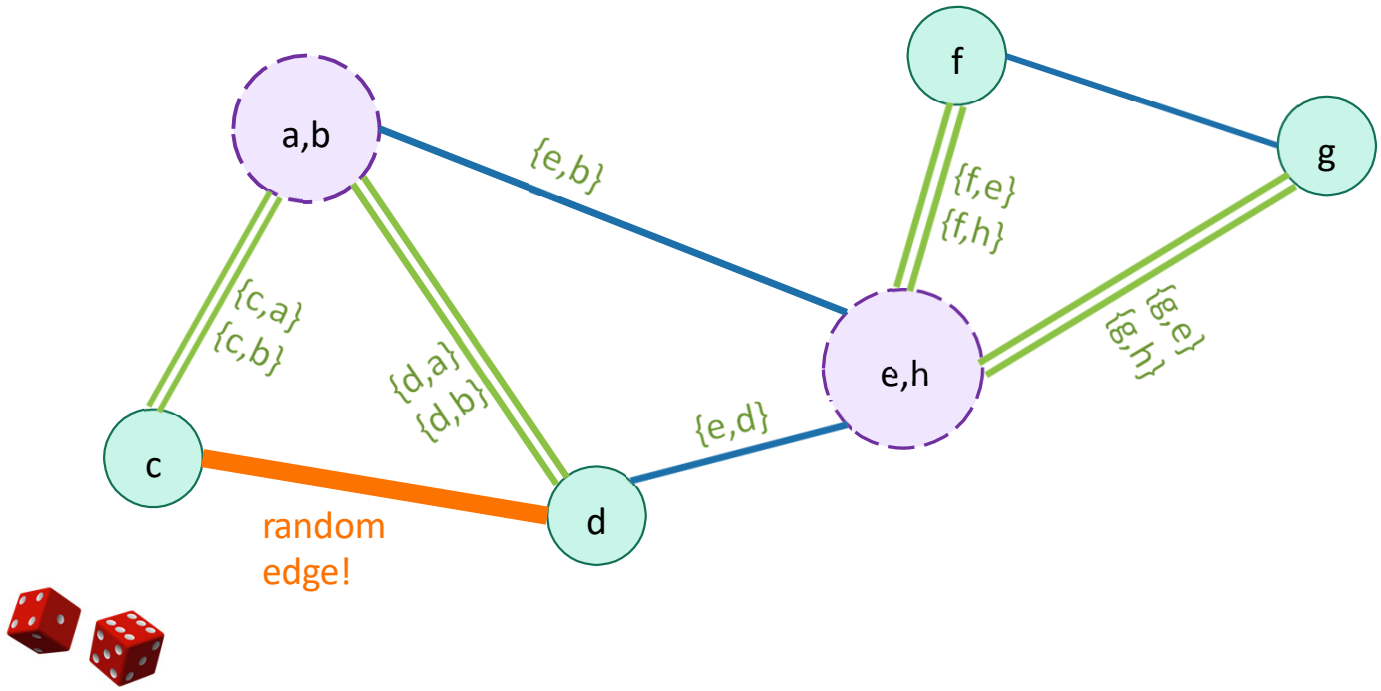
Karger's Algorithm



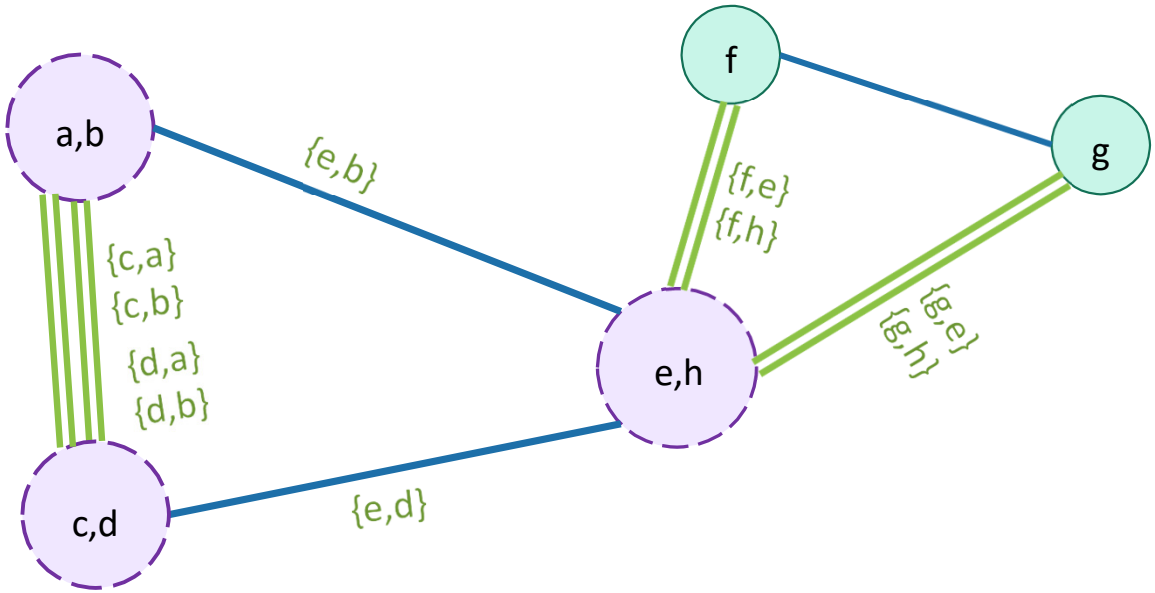
Karger's Algorithm



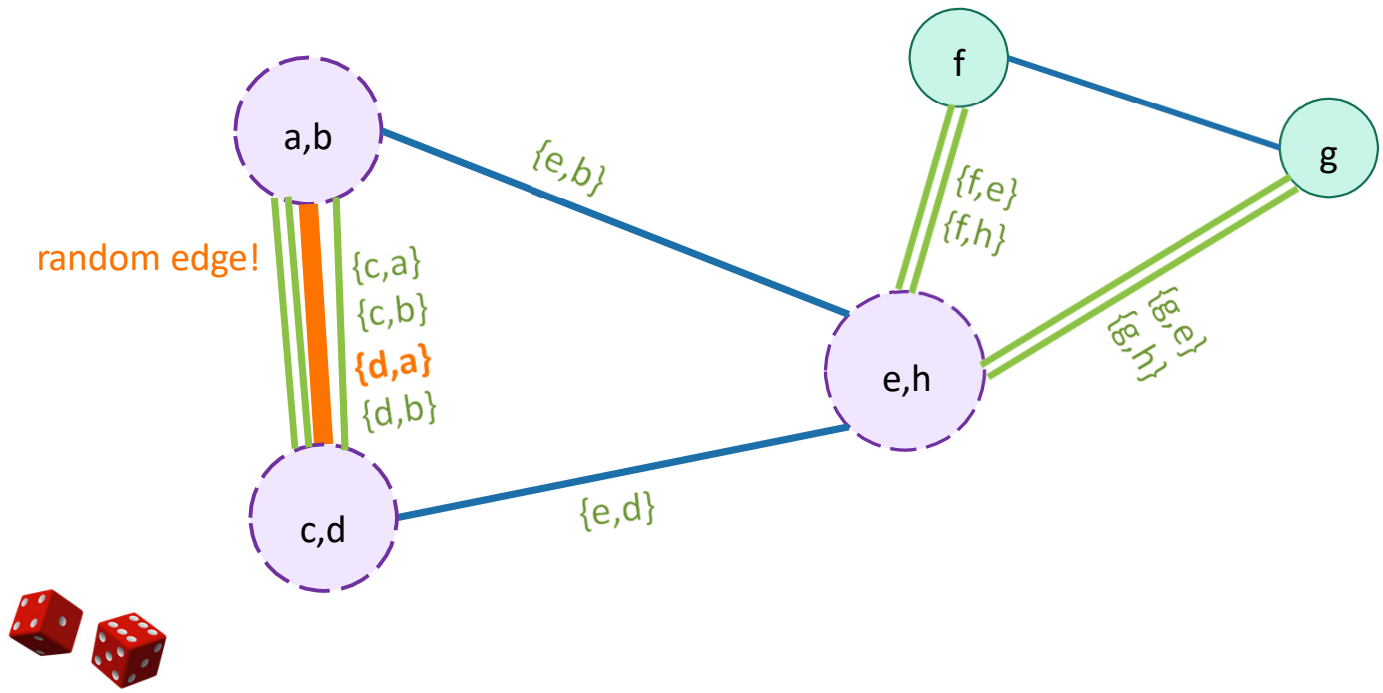
Karger's Algorithm



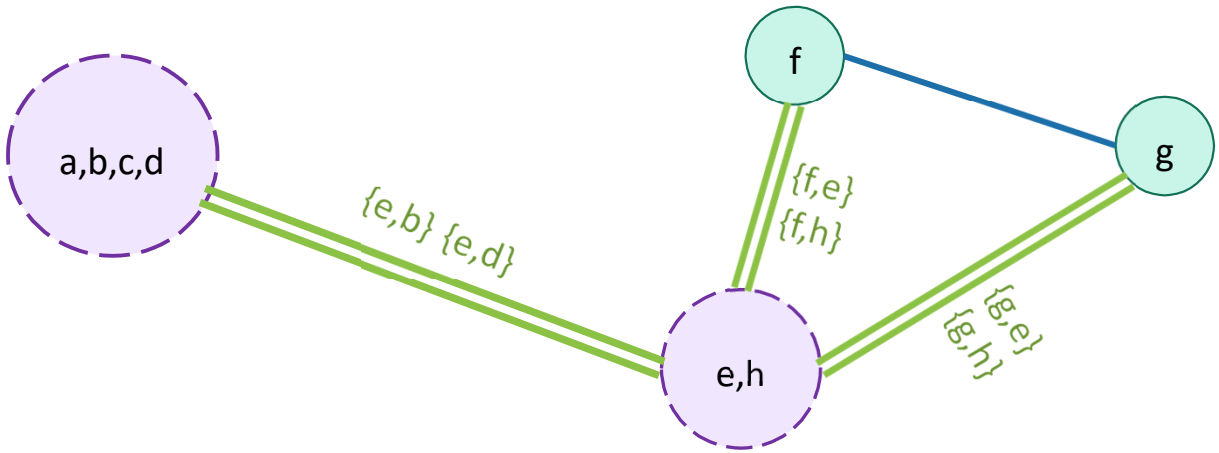
Karger's Algorithm



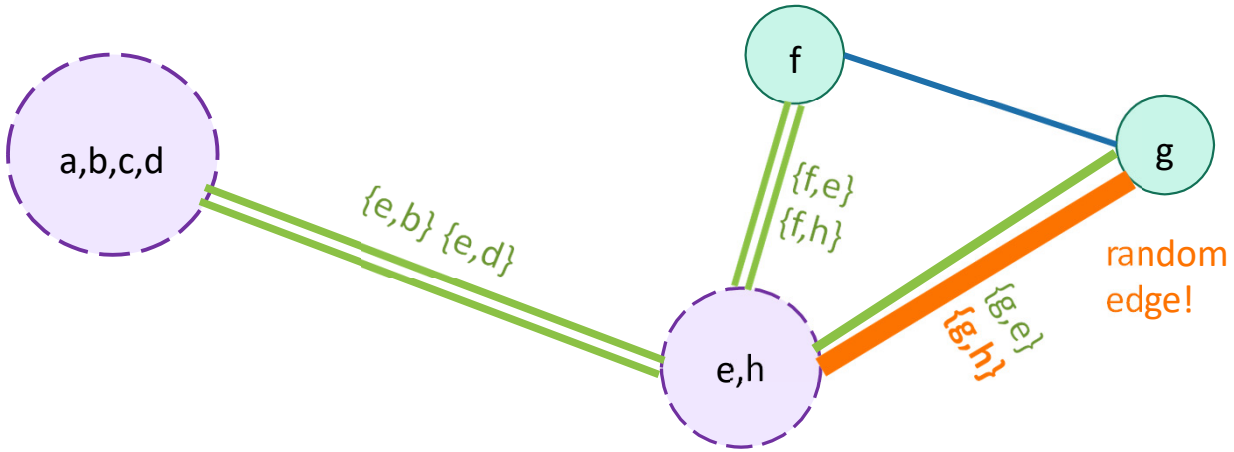
Karger's Algorithm



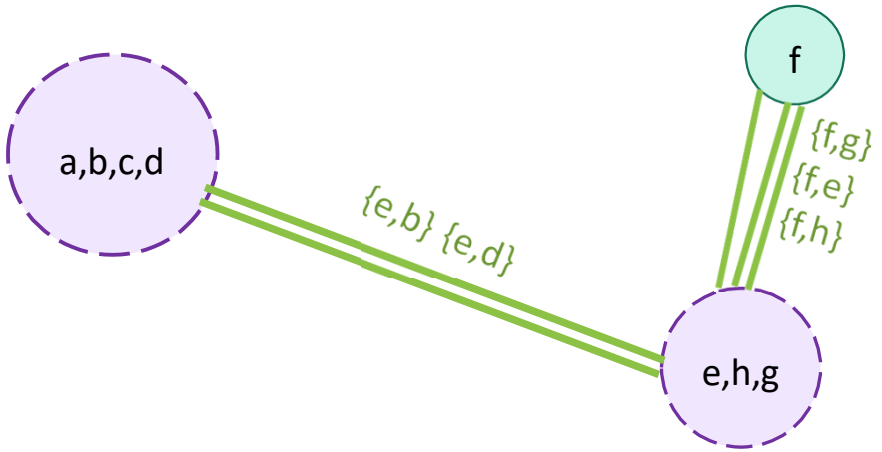
Karger's Algorithm



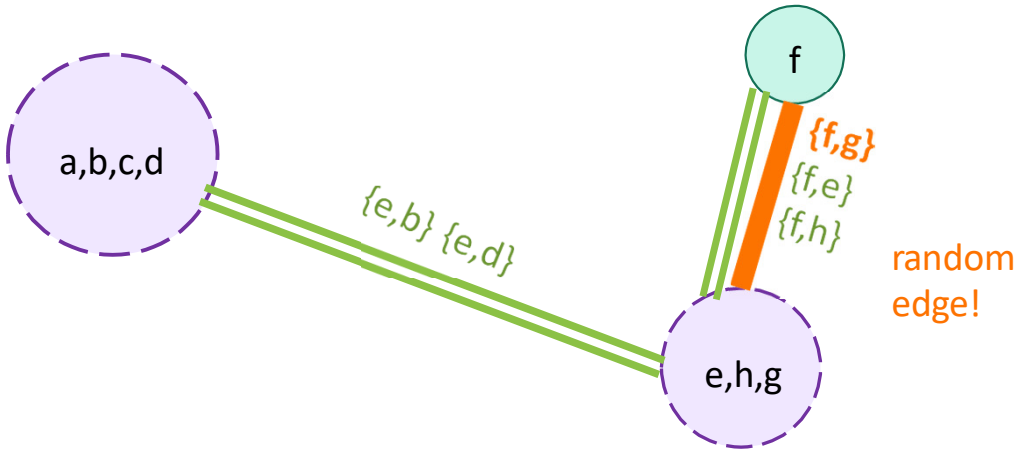
Karger's Algorithm



Karger's Algorithm

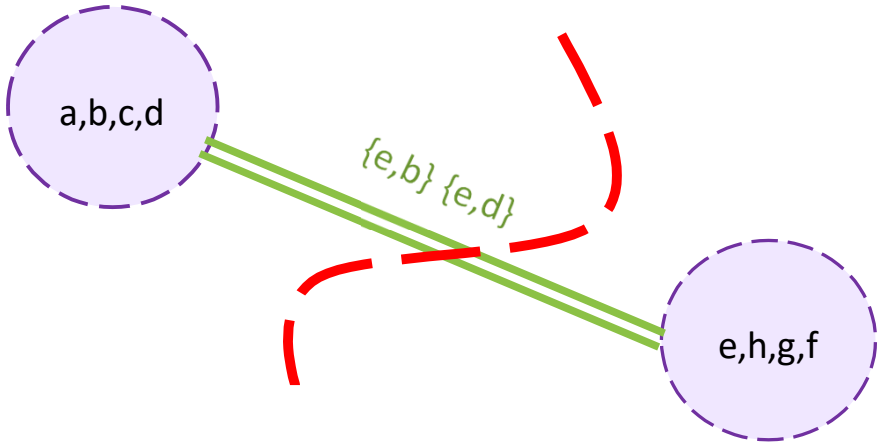


Karger's Algorithm

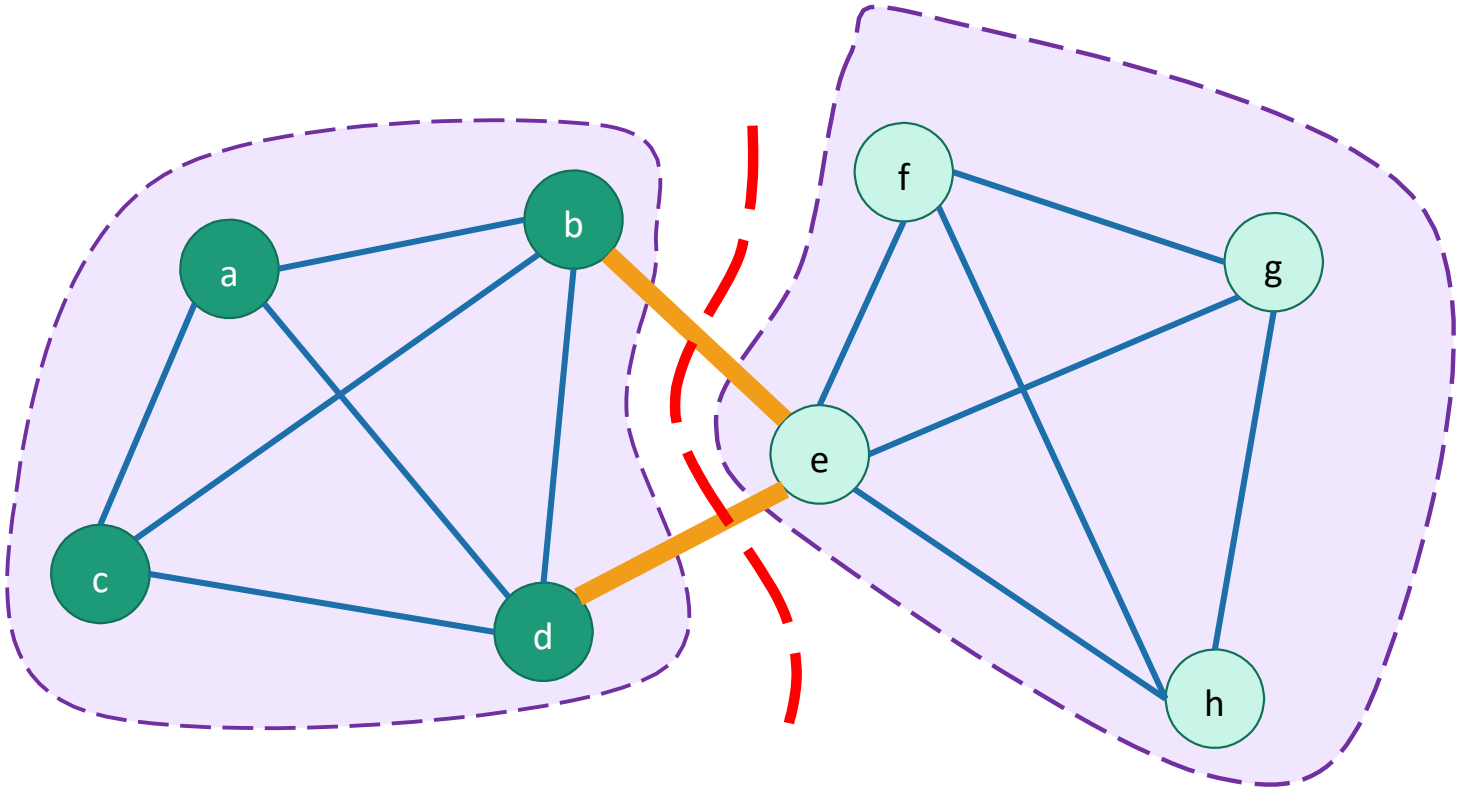


Karger's Algorithm

Produces cut {a,b,c,d}, {e,h,g,f}



Karger's Algorithm



Karger's Algorithm Properties

Even though G has 2^{n-1} global cuts overall...

Theorem: For any global minimum cut (A, B) in G, Karger's Algorithm returns (A, B) with probability $1/\binom{n}{2}$.

Corollary: Any undirected graph G has at most $\binom{n}{2}$ global mincuts.

Analysis idea for Karger's Algorithm

- Suppose # of edges crossing global mincut (A, B) is k.
- If (A, B) is returned can't ever contract any of those k edges.
- Since mincut is size k, min-degree $\geq k$ so $\geq kn/2$ edges to start and probability of choosing a crossing edge is

$$\leq \frac{k}{kn/2} = \frac{2}{n}$$

- After t contractions, $n - t$ nodes and $2/(n-t)$ probability of crossing.
- Probability a crossing edge is never chosen is then:
- $\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right)\left(1 - \frac{2}{n-2}\right)\cdots\left(1 - \frac{2}{3}\right) = \frac{\cancel{n-2}}{n} \cdot \frac{\cancel{n-3}}{n-1} \cdot \frac{\cancel{n-4}}{\cancel{n-2}} \cdots \frac{\cancel{3}}{5} \cdot \frac{2}{\cancel{4}} \cdot \frac{1}{\cancel{3}}$

Methods: Spectral Graph Algorithms

Apply linear algebra methods like computing eigenvalues and eigenvectors to the adjacency matrix of the graph!

- Good properties of Google's Pagerank algorithm for ranking search results are based on related quantities.

Other Algorithmic Scenarios

Massive Data Sets

When your input size is in the Megabyte or Terabyte range, polynomial time isn't fast enough; even $O(n^2)$ time algorithms will be too slow.

- Only linear or near-linear (e.g. $O(n \log n)$) time algorithms will do.
- May need to settle for approximate solutions using randomized algorithms
- Can even get sublinear algorithms based on random sampling.
- Though there is plenty of storage for the data, it only fits in slow storage. Our assumption of constant-time random-access to the data may not be correct, but it is easy to send that data as a high bandwidth sequential stream to the processor.

CSE 422 Toolbox for Modern Algorithms focuses on these and other related problems

Streaming and Online algorithms

So far, we have assumed that the entire input is available at the start.

- With data available only as a stream it may not make sense to assume that there is an "end" to the data - algorithms might need to produce answers or make irrevocable decisions continuously rather than produce just one answer and stop.
  - E.g. maintaining approximate website access statistics.
- How do we measure solution quality in this case?
  - Streaming algorithms: there is a single exact answer after each step and the algorithm's answer is close to that (maybe also with small probability of error).
  - Online algorithms: Sometimes it is impossible to be close to the best. Instead compare the *ratio* of the quality of the answer with the best online solution that could be obtained by an algorithm with full information from the start.

Parallel Algorithms...

Modest-level parallelism for multicore processors

At large scales in data centers we can have thousands of processors working on the same problem.

- MPC (Massively Parallel Computation) algorithms
  - e.g. using MapReduce as you might have used in CSE 344
  - How can you best design algorithms with primitives like these?

- Large Scale Simulation and Multigrid algorithms
  - Operations on massive sparse matrices.

Algorithmic Mechanism Design

We have assumed that the actual inputs to our algorithms are available.

- What if the inputs are each held by agents who might lie about their input values?
- e.g. n people know their value = maximum they are willing to spend to buy a painting.
- If they didn't lie, the seller could ask them all, compute the MAX and sell to the person with the largest value at that price.
- Instead we have algorithms we call an Auction.
  - e.g. English Auction: Winning bidder will pay a tiny bit more than the 2nd highest value
- Designing these algorithms/mechanisms well is hugely important economically:
  - e.g. Adwords auction for Google is a huge portion of their revenue.

Quantum Algorithms...

... and much much more!!

Course Evaluations

- We really want your feedback!
- Please take the time now to fill out the evaluations both for the lecture and quiz sections.
- I will be in my office for my regular office hour after class.