Heaps of topics we didn't cover!

- **Randomized Algorithms**: Instead of providing an optimal solution <u>always</u>, provide an optimal solution with some arbitrarily high (and controllable) probability.

E.g. Global Min Cut: Could solve via N-1 Max Flow computations. Randomized Algo is trivial

While G contains > 2 nodes:
  Choose an edge uniformly at random
  Contract e, replacing endpoints with new node w
Return the weight of the final cut

* note: Here, we maintain the multiplicity of each edge when we collapse endpoints.

The probability that this algo finds the global min cut is $\binom{n}{2}^{-1}$. If we run this algo $\binom{n}{2}\lg(n)$ times and keep the minimum result, the probability we fail to find the global min cut is

$$\left[1-\binom{n}{2}^{-1}\right]^{\binom{n}{2}\lg(n)} \le \frac{1}{e^{\lg(n)}} = \frac{1}{n}$$

- Because the failure prob is small and the algo is randomized, we can repeat many times to find the solution with high probability (whp).
- Randomized Algorithms are their own field of research with many interesting results.
- Eliminating or reducing the degree of randomness in a randomized algorithm is known as "derandomization". Can be used to find non-randomized algos that would otherwise be difficult to discover.

• Local Search

- Start with a set of "feasible" solutions C
- Find a neighbor relation between solutions : $S \sim S'$ for $S, S' \in C$
- $N(S) = \{S' : S \sim S'\}$ are neighboring feasible solutions of S.

Alg: ① define C (may be implicit)
② define $N(\cdot)$ (may be implicit)
③ Let S_0 be some feasible solution
④ Let $S = S_0$
⑤ Repeatedly choose some $S' \in N(S)$ and, based on a rule, set $S = S'$

Intuition: Global optimization may be difficult, but we can often improve our existing solution locally.

E.g. Vertex Cover

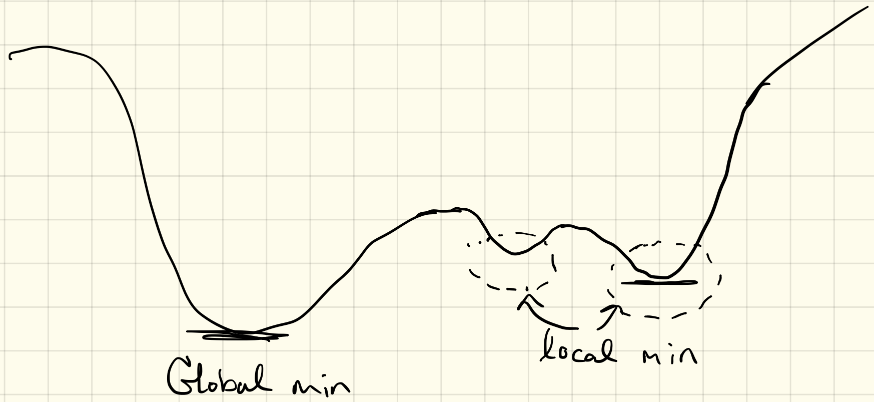
Define a state S as a set of vertices that is <u>a</u> vertex cover.

$S \sim S'$ if S' can be obtained from S by adding & deleting a single vertex

<u>Algorithm</u>: While there is $S' \in N(S)$ with a lower cost (smaller cardinality),
let $S = S'$

Pros: Often trivial to implement & understand
Cons: Often difficult to prove the quality of the solution.

Can get trapped in local optima



Some strategies to avoid this

- Simulated Annealing
    - only accept a move with probability proportional to its benefit
    - Sometimes "skip" optimal moves
- Run many instances of local search & keep the best (similar to randomized algos.)

• Advanced Dynamic Programming

- DP over hypergraphs instead of DAGs
    - Solution relies on multiple optimal sub-problems simultaneously

    E.g. in NLP, finding the optimal parse in a Context-free grammar
    in comp bio some RNA-structure problems, network history inf.
    - Efficiently enumerating k-best solutions (cube pruning)
    - Efficiently summing over / aggregating exponentially many sols.

- Numerical algorithms

E.g. ① Solving a linear system of equations
- Gauss - Seidel

② Computing Eigenvectors of a matrix $Av = \lambda v$

- 1^{st} eigenvector - power method
- all or top-K eigenvectors - Lanczos algorithm

③ Regression
- Least squares

Given n observations (x_i, y_i) and a model of the form $f(x, \beta)$, find the params β to minimize

$$\sum_{i=1}^{n} \left(y_i - f(x_i, \beta)\right)^2$$

- Probabilistic Inference

E.g. Given a set of observations (e.g. X = points in the plane) and a model (points coming from k Gaussian distributions), find the "best" parameters.

- Find $\vec{u}, \vec{\Sigma}$, the mean and covariance params that maximize

$$P(\vec{u}, \vec{\Sigma} \mid X) \propto \mathcal{L}(X \mid \vec{u}, \vec{\Sigma})$$

Many very cool algorithms & ideas

1. Expectation Maximization
2. Full Bayesian Inference (don't just find params, but full posterior distributions)
3. Variational Methods

Final Exam: This room on

- Monday, May 20. 5:30 – 8:00 PM

- Comprehensive, and covers topics from all semester
- Will have material from complexity / approx algos.
- Will not be proportionally longer than midterms (more time per problem)