CMSC 451: Local Search & Randomized Algorithms

Slides By: Carl Kingsford

Department of Computer Science
University of Maryland, College Park

Based on §12.1,12.2,13.2 of *Algorithm Design* by Kleinberg & Tardos.

Local Search

What if we have a hard problem but we can't find an approximation algorithm for it?

Local search is a general class of algorithms that is often useful in practice.

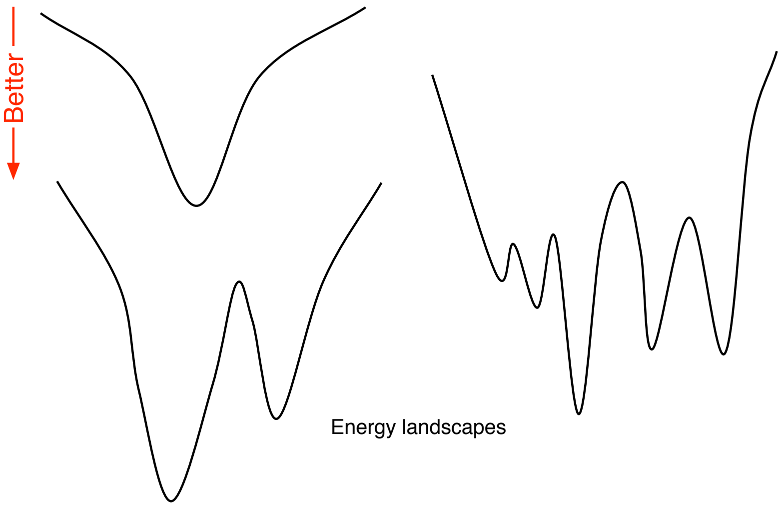
Unfortunately, we can almost never prove that they will return a good solution.

Optimization Problems

Optimization problems

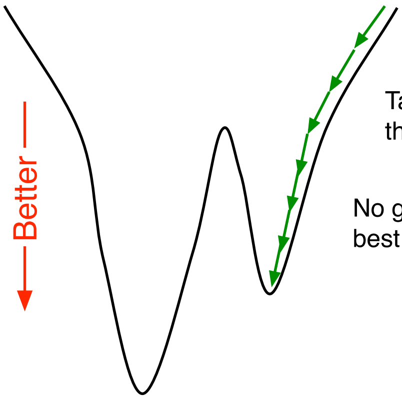
- A set $\mathcal{C}$ of possible solutions
- A cost $\mathbf{cost}(c)$ for each $c \in \mathcal{C}$.
- We're looking for a minimum/maximal cost $c \in C$.

Energy Landscapes



Energy landscapes

Gradient Descent



Take a series of "steps," improving the solution a little bit each time.

No guarantee you'll end up at the best solution...

A Little More Formal

Local Search:

- A set of "feasible solutions": $\mathcal{C}$.
- A neighbor relation between some of these solutions: $S \sim S'$ for some pairs $S, S' \in \mathcal{C}$.
- $\mathcal{N}(S) = \{S' : S \sim S'\}$: the neighbors of solution S.

Local Search

Local Search Algorithm Schema:

1. Define a set of feasible solutions $\mathcal{C}$.
2. Define a neighbor relation $\sim$ on these sets.
3. Let S_0 be some feasible solution.
4. Let $S = S_0$.
5. Repeatedly choose some $S' \in \mathcal{N}(S)$, and let $S = S'$.

Example: Vertex Cover

Vertex Cover

Find the minimum size vertex cover for graph G.

Define a state S as a set of vertices that is a vertex cover.

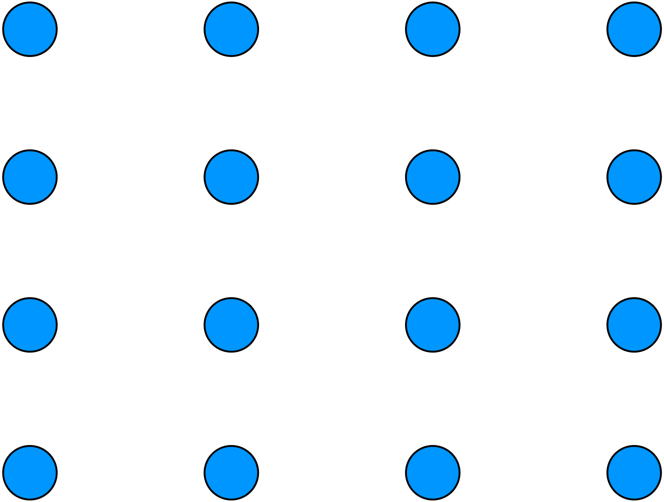
Two states are neighbors if they differ by adding or deleting a vertex.

Algorithm:

While there is a neighbor S' of S with lower cost, let the new S be the lowest cost neighbor.

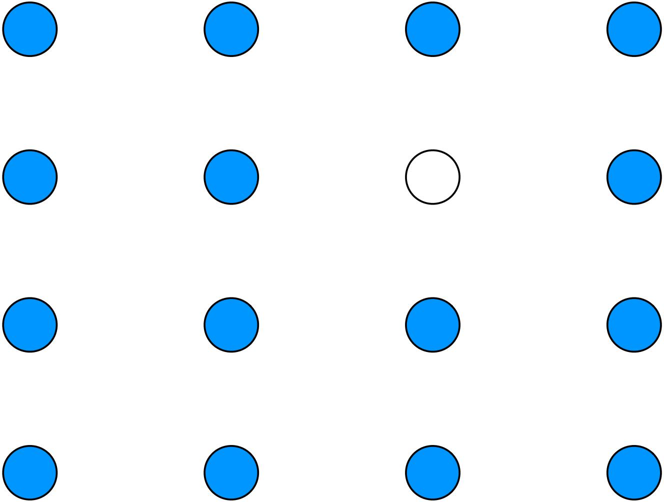
Gradient Descent: Vertex Cover

Empty Graph:



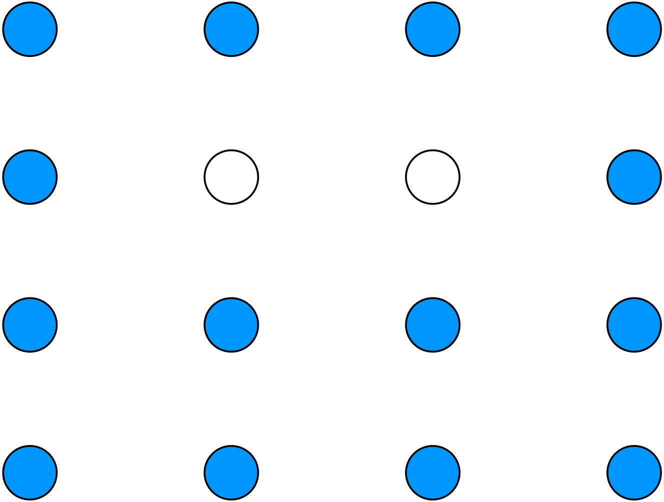
Gradient Descent: Vertex Cover

Empty Graph:



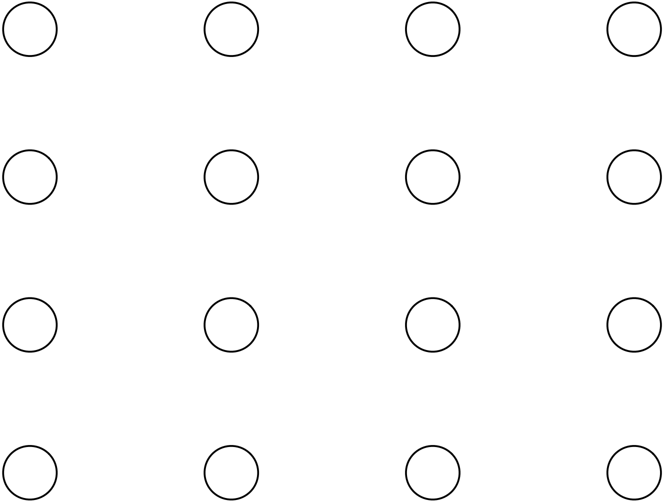
Gradient Descent: Vertex Cover

Empty Graph:

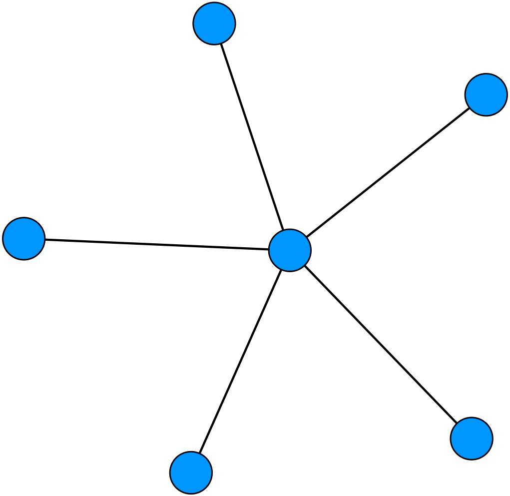


Gradient Descent: Vertex Cover

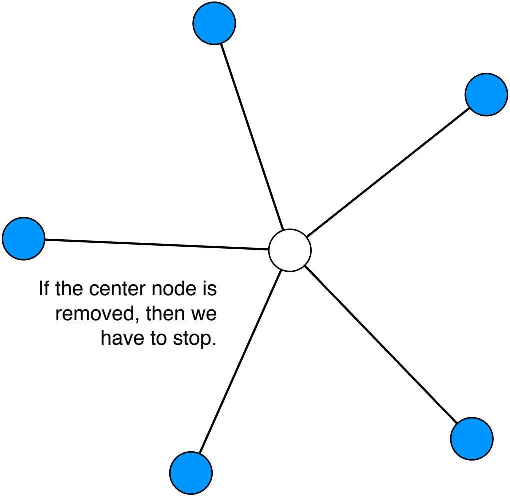
Empty Graph:



Gradient Descent: Vertex Cover 2



Gradient Descent: Vertex Cover 2



Metropolis Algorithm

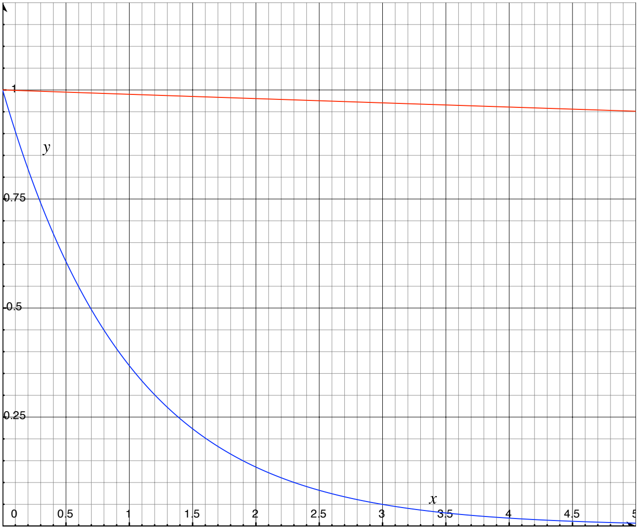
Physical systems also typically have an "energy."

The Gibbs-Boltzmann function says the probability of a system of being in a state of energy E is:

$$e^{-E/(kT)}$$

where $T > 0$ is the temperature of the system.

Gibbs-Boltzmann



Metropolis Algorithm, 2

Metropolis Algorithm:

```
S = an initial solution
While not done:
  Choose S' ∈ 𝒩(S)
  If cost(S') ≤ cost(S):
    Set S = S'
  Else:
    Delta = cost(S') - cost(S)
    With probability exp(-delta / (kT)):
      Let S = S'
  EndIf
EndWhile
```

Simulated Annealing

High values of T means you jump around a lot.

Low values of T mean you never increase the cost.

Simulated Annealing: **Idea:** Start with high T and reduce it as time progresses.

Some cooling schedule determines how to change T.

Lot of work into finding cooling schedules that work in practice...

Simulated Annealing, 2

- Simulated annealing used all the time in practice
- No guarantees, but often gets good solutions
- (Often does not, too, though.)

Randomized Algorithms

Randomized Algorithms

- Allow our algorithms to flip some random coins to make their choices.
- May require that the optimum solution is found with expected good runtime.
- Or may require that we always run in polynomial time, and we find the optimum solution with high probability.
- Often run in expectation faster than

Quicksort

You've probably seen probablistic algorithms of the first type: quicksort.

When the list of numbers is already sorted, a naive deterministic algorithm performs very bad ($O(n^2)$).

A solution to this: randomly permute the input numbers.

Then the chance that you are in a "bad" case is small.

Global Minimum Cut

Global Minimum Cut

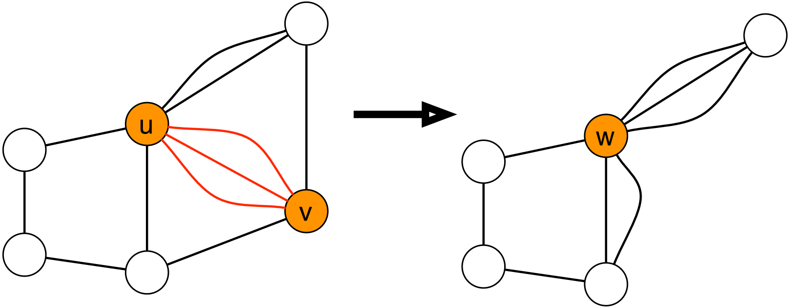
Given an undirected graph G find a partition of the nodes of G into two non-empty sets A and B such that the number of edges that have 1 endpoint in A and 1 endpoint in B is minimized.

Like minimum cut, but in an undirected graph, and we don't specify s and t.

Using Network Flow

- Let s be some node in G.
- In the global minimum cut, s must be separated from something.
- Try all $n-1$ choices for that something as t, and use network flow to compute the minimum $s-t$ cut.
- (Replace each undirected edge by two, anti-parallel, directed edges.)
- Cost is $n-1$ network flow computations.

Contraction



Contraction step: choose a edge and merge its endpoints.

Contraction Algorithm

Contraction Algorithm:

```
While G contains more than 2 nodes:
   Choose an edge e uniformly at random
   Contract e, replacing its endpoints
      with a new node w
```

Each new node is really a *supernode* that "contains" a number of original nodes.

Once we have a graph with only 2 supernodes, the supernodes define the cut.

Proving Correctness

Let F be a global minimum cut.

Suppose $|F| = k$.

Every node in G must have degree $\geq k$. Why?

Therefore, $|E| \geq \frac{1}{2}kn$.

The chance that we contract an edge in F in the first step is at most:

$$\frac{k}{\frac{1}{2}kn} = \frac{2}{n}$$

After j contractions

After j contractions there are $n - j$ supernodes.

Each super node has degree $\geq k$. Why?

There are at least $\frac{1}{2}k(n-j)$ edges, and the probability that we choose one from F to contract is:

$$\frac{k}{\frac{1}{2}k(n-j)} = \frac{2}{n-j}\,.$$

Proof, cont.

The contraction algorithm stops after $n-2$ iterations.

It will return the global minimum cut if none of the $n-2$ contractions picked one of the edges in F.

Def. $\mathcal{E}_i =$ even that an edge of F was not contracted in step i.

- $\Pr[\mathcal{E}_1] \geq 1 - \frac{2}{n}$
- $\Pr[\mathcal{E}_{j+1} \mid \mathcal{E}_1 \cap \mathcal{E}_2 \cap \cdots \cap \mathcal{E}_j] \geq 1 - \frac{2}{n-j}$

Probability of Success: $\Pr[\mathcal{E}_1 \cap \cdots \cap \mathcal{E}_{n-2}]$

Unravel Conditional Expectations

Theorem

The probability that the contraction algorithm returns the minimum cut is $\geq 1/\binom{n}{2}$.

$$
\begin{aligned}
&\Pr[\mathcal{E}_1 \cap \cdots \cap \mathcal{E}_{n-2}] \\
&\quad = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdots \Pr[\mathcal{E}_{j+1} \mid \mathcal{E}_1 \cap \cdots \cap \mathcal{E}_j] \cdots \\
&\quad \geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \cdots \left(1 - \frac{2}{n-j}\right) \cdots \left(1 - \frac{2}{3}\right) \\
&\quad = \left(\frac{n-2}{n}\right) \cdots \left(\frac{n-3}{n-1}\right) \cdots \left(\frac{1}{3}\right) \\
&\quad = \frac{2}{n(n-1)} = \binom{n}{2}^{-1} \square
\end{aligned}
$$

Repeating the contraction algorithm

Repeat the algorithm $\binom{n}{2} \ln n$ times.

The probability that we fail to find the global minimum cut every time is:

$$\left(1 - \frac{1}{\binom{n}{2}}\right)^{\binom{n}{2} \ln n} \leq \frac{1}{n}.$$

Summary

- Local Search often simple and works well in practice, despite it being hard to prove anything about.
- Randomization often yields simpler, faster algorithms.