Local Search

Learning Goals.

- Introduce Local Search.
- Metropolis Algorithm and Simulated Annealing.
- Max-Cut Problem.
- Local Search in Shape from Texture.

Readings: Reading Chp. 12, up to end of Section 12.5.

Based on Chapter 12, Kleinberg and Tardos

Coping With NP-Hardness

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you're unlikely to find poly-time algorithm.

Must sacrifice one of three desired features.

- Solve problem to optimality.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.

Here we use iterative local search, finding a slightly better solution each step, until we cannot locally improve the solution.

Blind hill-climbing (or descent), usually with weak or no guarantees.

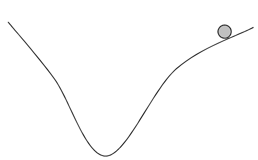
A key component here is how the local search neighbourhood is defined.

Local Search and Local Minima

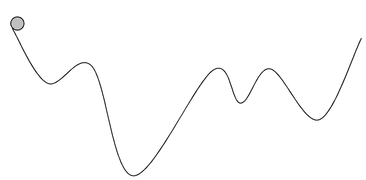
Local search. Algorithm that explores the space of possible solutions in sequential fashion, moving from a current solution to a "nearby" one.

Neighbour relation. Let $S \sim S'$ be a neighbor relation for the problem. E.g., a unit-length left/right move for the ball below.

Greedy descent. Let S denote current solution. If there is a neighbor S' of S with strictly lower cost, replace S with the neighbor whose cost is as small as possible. Otherwise, terminate the algorithm.



A funnel



A jagged funnel

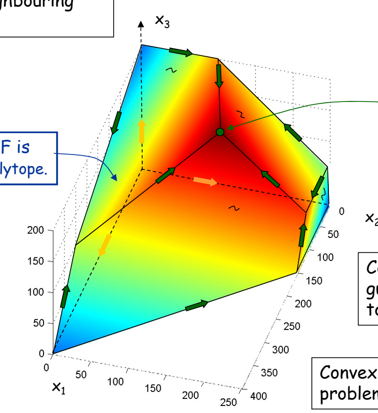
Simplex Method as Local Search

Maximize LP by switching between neighbouring vertices.

Problem Constraints:
$x_1 + 3x_3 \le 600$
$x_2 + x_3 \le 300$
$x_1 + x_2 + x_3 \le 400$
$x_2 \le 250$

Feasible set F is this closed polytope.

Maximizes $c^T x$, $c^T = (2, 3, 4)$, $c^T x$ gives shading.



Convexity of problem guarantees convergence to optimal solution.

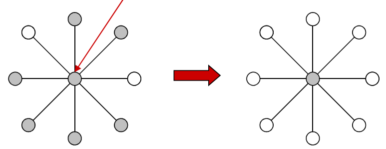
Convex (and "pseudo"-convex) problems are like funnels.

Greedy Descent: Vertex Cover

Vertex Cover Problem: Given a graph G = (V, E), find a subset of nodes S of minimal cardinality such that for each (u, v) in E, either u or v (or both) are in S.

Example: S initially V.

Every neighbour S′ of this cover must have this vertex in S′.



○ - denotes vertex is in cover S

Neighbour relation. Vertex cover S ~ S' if S' can be obtained from S by adding or deleting a single vertex and S′ itself is a vertex cover. Each vertex cover S has at most n-1 neighbors (as usual, |V| = n).

Greedy Descent: Vertex Cover

Vertex Cover Greedy Algorithm:

```
Greedy_Vertex_Cover(V, E) {
   S ← V
   while (true) {
      Nhbrs = {S′ s.t. S′~ S, |S′| < |S| and
                    S′ is a vertex cover}.
      if Nhbrs is not empty
        S ← an element of Nhbrs
      else
        return S
   }
}
```

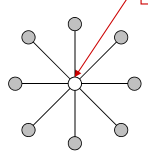
Remark. Algorithm terminates after at most n steps since each update decreases the size of the cover by one.

Key Question: How close to optimal is the resulting vertex cover?

Greedy Descent: Vertex Cover

Example: S initially V.

Only neighbour which is a vertex cover is V, which is bigger. Alg. terminates!



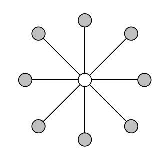
○ - denotes vertex is in cover S

Remark. Algorithm can get stuck in a "local minimum".
Might consider a larger neighbourhood to try to avoid this.
E.g., Swap up to two vertices into or out of S.
Then greedy descent might need to check $O(|V|^2)$ pairs.

Might consider randomly selecting amongst ties. Running several times.

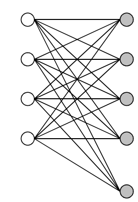
Greedy Descent: Vertex Cover

Local optimum. No neighbour (a cover) is strictly better.

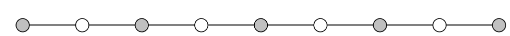


○ - denotes vertex is in cover S

optimum = center node only
local optimum = all other nodes



optimum = all nodes on left side
local optimum = all nodes on right side



optimum = even nodes |S| = 4
local optimum = odd nodes |S| = 5

Metropolis Algorithm

Metropolis algorithm. [Metropolis, Rosenbluth, Rosenbluth, Teller, Teller 1953]

- Simulate behavior of a physical system according to principles of statistical mechanics.
- Globally biased toward "downhill" steps, but occasionally makes "uphill" steps to break out of local minima.

Gibbs-Boltzmann function. The probability of finding a physical system in a state with energy E is proportional to $e^{-E/(kT)}$, where $T > 0$ is temperature and k is a constant.

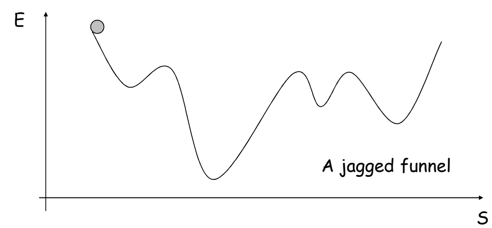
- For any temperature $T > 0$, function is monotone decreasing function of energy E.
- System more likely to be in a lower energy state than higher one.
  - T large: high and low energy states have roughly same probability
  - T small: low energy states are much more probable

For the vertex cover problem: We could define $E = |S|$.

Metropolis Algorithm

Metropolis algorithm.

- Given a fixed temperature T, maintain current state S.
- Randomly perturb current state S to new state $S' \in N(S)$.
- If $E(S') \le E(S)$, update current state to S'
  Otherwise, update current state to S' with probability $e^{-\Delta E/(kT)}$, where $\Delta E = E(S') - E(S) > 0$.

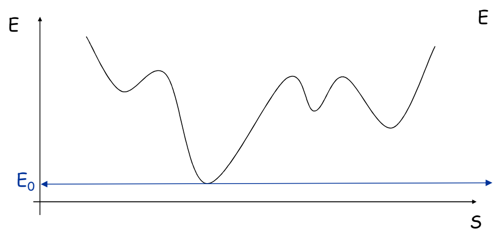


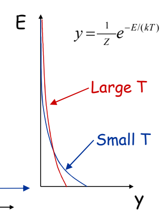
Metropolis Algorithm

Theorem. Let $f_S(t)$ be fraction of first t steps in which simulation is in state S. Then, assuming some technical conditions, with probability 1:

$$\lim_{t\to\infty} f_S(t) = \frac{1}{Z} e^{-E(S)/(kT)},$$

$$\text{where } Z = \sum_{S \in \text{Universe}} e^{-E(S)/(kT)}.$$





Intuition. Simulation spends roughly the right amount of time in each state, according to Gibbs-Boltzmann equation.

Simulated Annealing

Simulated annealing.

- T large $\Rightarrow$ probability of accepting an uphill move is large.
  State samples explore the space (often computational molasses).
- T small $\Rightarrow$ uphill moves are almost never accepted.
  State samples remain near local minimum.
- Idea: turn knob to control T.
- Cooling schedule: T = T(i) at iteration i.

Physical analog.

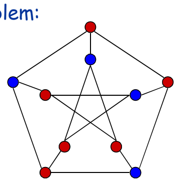
- Take solid and raise it to high temperature, we do not expect it to maintain a nice crystal structure.
- Take a molten solid and freeze it very abruptly, we do not expect to get a perfect crystal either.
- Annealing: cool material gradually from high temperature, allowing it to reach statistical equilibrium at succession of intermediate lower temperatures.

Maximum Cut

Maximum cut. Given an undirected graph $G = (V, E)$ with positive integer edge weights w_e, find a node partition (A, B) such that the total weight of edges crossing the cut is maximized.

$$w(A,B) := \sum_{u \in A,\ v \in B} w_{uv}$$

Example problem:



All edge weights 1.
A = red's B = blue's
Note: Recolour any single vertex and w(A,B) decreases.
Maximal?

Real applications. Scheduling problems, circuit layout, statistical physics.

Scheduling Two Machines

Problem Statement. Given jobs $J_n = (p_n, t_n)$, for n=1, …, N, where t_n is the processing time and p_n is the priority. We wish to assign each job to one of two machines M_1 and M_2 (cannot split a job). Use m(n) = 1 or 2 to denote the assignment of job J_n to machine $M_{m(n)}$.

We assume each p_n and t_n are positive integers.

On machine m we sort the jobs, and process them in order of increasing $s_m(n)$. Given this order, the finish time T_n of each job on machine m is:

$$T_n \equiv t_n + \sum_{k:s_m(k)<s_m(n)} t_k.$$

We wish to choose a schedule to minimize the priority weighted finish time:

$$F \equiv \sum_{n=1}^{N} p_n T_n.$$

Warm-Up and Review: Scheduling One Machine

If we only had one machine,

$$F_1 \equiv \sum_{n=1}^{N} p_n T_n = \sum_{n=1}^{N} \left[p_n t_n + \sum_{k:s(k)<s(n)} p_n t_k \right].$$

A natural greedy rule is to sort the jobs in non-increasing order of priority per unit time, p_n/t_n. Thus, $p_k/t_k > p_n/t_n$ iff $s(k) < s(n)$.

Can prove this job ordering minimizes F_1 (for one machine).

Using this sorting rule we can rewrite F_1 simply as

$$F_1 \equiv \sum_{n=1}^{N} p_n T_n = \sum_{n=1}^{N} \left[p_n t_n + \sum_{k<n} w_{kn} \right],$$

where $w_{kn} = \min\{ p_k t_n, p_n t_k \}$.

Scheduling Two Machines: Graph Formulation

Consider the graph G with vertices V = {1, … , N} representing each job. The undirected edges E consist of all pairs of vertices with edge weights $w_{kn} = \min\{ p_k t_n, p_n t_k \}$.

Suppose (A,B) with $B = V \setminus A$, is a cut corresponding to jobs to be assigned machines M_1 and M_2 respectively. Then summing up the costs for each machine separately we find

$$\begin{aligned} F &\equiv \sum_{n \in A} p_n T_n + \sum_{n \in B} p_n T_n, \\ &= \sum_{n=1}^{N} p_n t_n + \sum_{k<n,\{k,n\} \subseteq A} w_{kn} + \sum_{k<n,\{k,n\} \subseteq B} w_{kn}, \\ &= \underbrace{\boxed{\sum_{n=1}^{N} p_n t_n + \sum_{k<n} w_{kn}}}_{\text{Constant}} - \underbrace{\boxed{\sum_{j \in A, k \in B} w_{jk},}}_{\text{Cut Weight}} \\ &= C - w(A,B) \end{aligned}$$

Minimizing objective function F is equivalent to solving max-cut on G!

Maximum Cut: Single Flip Neighbourhood

Greedy single-flip. Given a partition (A, B), move one node from A to B, or one from B to A if it improves the solution.

```
Max-Cut-Local (G, w) {
   Pick a random node partition (A, B)

   while (∃ improving node v) {
      if (v is in A) move v to B
      else           move v to A
   }

   return (A, B)
}
```

Local improvement: Suppose $u \in A$, then switch u to B iff

Weight of cut edges (u,v), if u moved to B.

$$\sum_{v\in A} w_{uv} > \sum_{v\in B} w_{uv}$$

Current weight of cut edges (u,v), since $u \in A$.

Greedy Single Flip Runtime

Greedy ascent: On each iteration, `Max-Cut-Local` chooses a single flip which increases w(A, B) (otherwise it stops).

Each iteration can be implemented to run in O(m+n) time, where |V| = n, |E| = m. (Can simply update the sum of weights for the vertices sharing edges with the one that was switched.)

Lemma: `Max-Cut-Local` runs in O((n+m)W) time.
Pf: Note w(A,B) must increase by at least 1 each iteration (integer edge weights). An upper bound for w(A, B) is $W = \sum_e w_e$, i.e., the sum of all edge weights in G. And a lower bound is 0. Therefore at most W iterations are required. ▪

So the runtime is pseudo-polynomial (depends on W not log(W)).

What about the weight of the cut w(A,B) that it produces?

Greedy Single Flip Approximation Properties

Theorem. Let (A, B) be a locally optimal partition and let (A*, B*) be optimal partition. Then $w(A, B) \geq \frac{1}{2} \sum_e w_e \geq \frac{1}{2} w(A^*, B^*)$.

weights are nonnegative

Pf.

- Local optimality implies that for all $u \in A$: $\sum_{v \in A} w_{uv} \leq \sum_{v \in B} w_{uv}$
  Adding up all these inequalities yields:

  negation of local improvement rule

$$2 \sum_{\substack{e\in E,\\ e \text{ is } A-A}} w_e \leq \sum_{u\in A,\, v\in B} w_{uv} = w(A,B)$$

- Similarly

$$2 \sum_{\substack{e\in E,\\ e \text{ is } B-B}} w_e \leq \sum_{u\in A,\, v\in B} w_{uv} = w(A,B)$$

- Now,

$$\sum_{e\in E} w_e = \underbrace{\sum_{e \text{ is } A-A} w_e}_{\leq \frac{1}{2}w(A,B)} + \underbrace{\sum_{e \text{ is } B-B} w_e}_{\leq \frac{1}{2}w(A,B)} + \underbrace{\sum_{e \text{ is } A-B} w_e}_{w(A,B)} \leq 2w(A,B) \quad ▪$$

each edge counted once

Maximum Cut: Big Improvement Flips

Local search. Has the appropriate a $\frac{1}{2}$-approximation bound for MAX-CUT, but is not poly-time! (So it is not a $\frac{1}{2}$-approximation algorithm).

Big-improvement-flip algorithm. Only choose a node which, when flipped, increases the cut value by at least $\frac{2\varepsilon}{n} w(A, B)$. That is, suppose $u \in A$, then switch u to B iff

Current weight of cut edges involving u.

Cut edges if u moved to B.

$$\sum_{v\in A} w_{uv} \geq \left(\sum_{v\in B} w_{uv}\right) + \frac{2\varepsilon}{n} w(A,B)$$

Claim. Upon termination, big-improvement-flip algorithm returns a cut (A, B) with $(2+\varepsilon)\, w(A, B) \geq w(A^*, B^*)$, i.e. a $1/(2+\varepsilon)$-approximation.

Pf idea. Add $\frac{2\varepsilon}{n} w(A, B)$ to each inequality in original proof, find

$$\sum_{e\in E} w_e = \sum_{e \text{ is } A-A} w_e + \sum_{e \text{ is } B-B} w_e + \sum_{e \text{ is } A-B} w_e \leq (2+\varepsilon) w(A, B)$$

Maximum Cut: Big Improvement Flips

Claim. Big-improvement-flip algorithm terminates after $O(\varepsilon^{-1} n \log W)$ flips, where $W = \Sigma_e w_e$.

That is, big-improvement-flip is a $1/(2+\varepsilon)$-approximation algorithm.

Pf:
- Each flip improves cut value by at least a factor of $(1 + \varepsilon/n)$.
- After n/ε iterations the cut value improves by a factor of
  $m = (1 + \varepsilon/n)^{(n/\varepsilon)}$
- For $x \geq 1$, $f(x) = (1+1/x)^x \geq 2$. [Blackboard]
- So for $n/\varepsilon \geq 1$, the cut value is at least doubled in n/ε iterations.
- But integer cut value can be doubled at most $O(\log(W))$ times.
- Total number of iterations is therefore $O((n/\varepsilon)\log(W))$.

Neighbor Relations for Max Cut

1-flip neighborhood. (A, B) and (A', B') differ in exactly one node.

k-flip neighborhood. (A, B) and (A', B') differ in at most k nodes.
- $\Theta(n^k)$ neighbors.

KL-neighborhood. [Kernighan-Lin, 1970]

cut value of (A_1, B_1) may be worse than (A, B)

- To form neighborhood of (A, B):
  - Iteration 1: flip node from (A, B) that results in best cut value (A_1, B_1), and mark that node.
  - Iteration i: flip node from (A_{i-1}, B_{i-1}) that results in best cut value (A_i, B_i) among all nodes not yet marked.
- Neighborhood of (A, B) = $(A_1, B_1), \ldots, (A_{n-1}, B_{n-1})$.
- Neighborhood includes some very long sequences of flips, but without the computational overhead of a k-flip neighborhood.
- Practice: powerful and useful framework.
- Theory: explain and understand its success in practice.

Maximum Cut: Context

Theorem. [Shani-Gonzales, 1976] There exists a $\frac{1}{2}$-approximation algorithm (polytime by defn) for MAX-CUT.

Theorem. [Goemans-Williamson, 1995] There exists an 0.878567-approximation algorithm for MAX-CUT.

$$\min_{0 \leq \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1-\cos\theta}$$

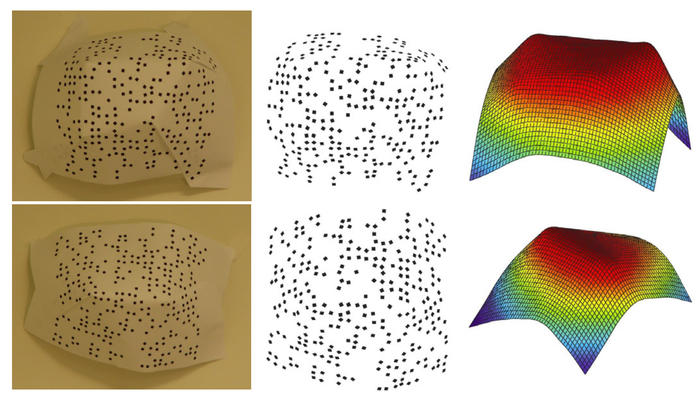
The G-W approach uses semi-definite programming (SDP) and provides a useful rounding heuristic in practice.

Theorem. [Håstad, 2001] Unless P = NP, no 16/17 approximation algorithm for MAX-CUT.

0.941176

Example: Local Search in Shape from Texture

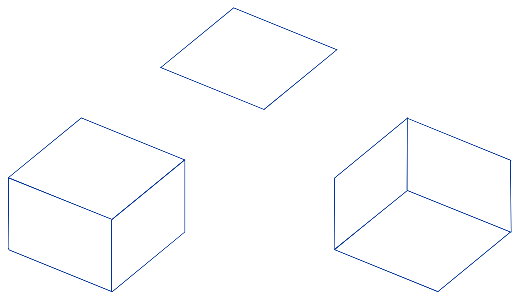
Humans can perceive shape from image texture. Can a machine?



Original Images   Texture Only   Reconstructed Shape

Shape From Projected Squares

Depth-flip ambiguity:

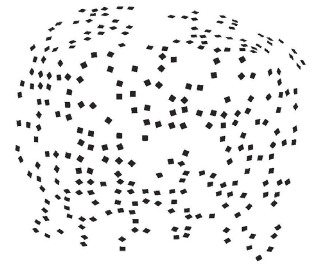


Given the image of a small square, there are two possible 3D orientations.

Given a mathematical model for image projection, we can solve for these orientations. We refer to them by $d = \pm 1$.

Merging Local Depth Flips

Every image parallelogram has a depth-flip ambiguity $d_k = \pm 1$.



We wish to find a set of flips that can be fit with a smooth surface.