Local Search

The Maximum Cut Problem

Algorithms: Design and Analysis, Part II

The Maximum Cut Problem

Input: An undirected graph $G = (V, E)$.

Goal: A cut (A, B) – a partition of V into two non-empty sets – that maximizes the number of crossing edges.

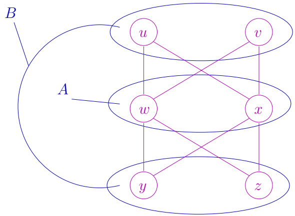
Sad fact: NP-complete.

Computationally tractable special case: Bipartite graphs (i.e., where there is a cut such that all edges are crossing)

Exercise: Solve in linear time via breadth-first search

Quiz

Question: What is the value of a maximum cut in the following graph?



A) 4

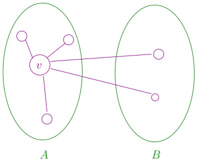
B) 6

C) 8

D) 10

A Local Search Algorithm

Notation: For a cut (A, B) and a vertex v, define
$c_v(A, B) = \#$ of edges incident on v that cross (A, B)
$d_v(A, B) = \#$ of edges incident on v that don't cross (A, B)



Local search algorithm:

(1) Let (A, B) be an arbitrary cut of G.

(2) While there is a vertex v with $d_v(A, B) > c_v(A, B)$:
- Move v to other side of the cut

[key point: increases number of crossing edges by $d_v(A, B) - c_v(A, B) > 0$]

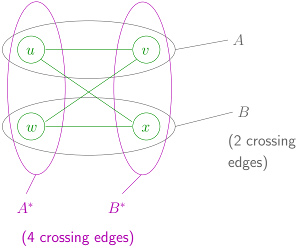
(3) Return final cut (A, B)

Note: Terminates within $\binom{n}{2}$ iterations [+ hence in polynomial time].

Performance Guarantees

Theorem: This local search algorithm always outputs a cut in which the number of crossing edges is at least 50% of the maximum possible. (Even 50% of $|E|$)

Tight example:



Cautionary point: Expected number of crossing edges of a random cut already is $\frac{1}{2}|E|$.

Proof: Consider a random cut (A, B). For edge $e \in E$, define

$X_e = \begin{cases} 1 & \text{if } e \text{ crosses } (A, B) \\ 0 & \text{otherwise} \end{cases}$. We have $\mathrm{E}[X_e] = \Pr[X_e = 1] = 1/2$.

So $\mathrm{E}[\#\text{ crossing edges}] = \mathrm{E}[\sum_e X_e] = \sum_e \mathrm{E}[X_e] = |E|/2$. QED

Proof of Performance Guarantee

Let (A, B) be a locally optimal cut. Then, for every vertex v, $d_v(A, B) \leq c_v(A, B)$. Summing over all $v \in V$:

$$\boxed{\sum_{v \in V} d_v(A, B)} \leq \boxed{\sum_{v \in V} c_v(A, B)}$$

counts each non-crossing edge twice

counts each crossing edge twice

So:

$2 \cdot$[# of non-crossing edges] $\leq 2 \cdot$[# of crossing edges]

$2 \cdot |E| \leq 4 \cdot$[# of crossing edges]

# of crossing edges $\geq \frac{1}{2}|E|$ QED!

The Weighted Maximum Cut Problem

Generalization: Each edge $e \in E$ has a nonnegative weight w_e, want to maximize total weight of crossing edges.

Notes:

(1) Local search still well defined

(2) Performance guarantee of 50% still holds for locally optimal cuts [you check!] (also for a random cut)

(3) No longer guaranteed to converge in polynomial time [non-trivial exercise]

Algorithms: Design and Analysis, Part II

Local Search

Principles of Local Search

Neighborhoods

Let X = set of candidate solutions to a problem.

Examples: Cuts of a graph, TSP tours, CSP variable assignments

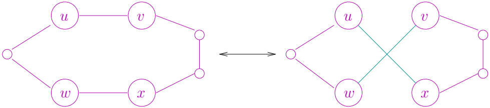
Key ingredient: Neighborhoods

- For each $x \in X$, specify which $y \in X$ are its "neighbors"

Examples: x, y are neighboring cuts $\iff$ Differ by moving one vertex

x, y are neighboring variable assignments $\iff$ Differ in the value of a single variable

x, y are neighboring TSP tours $\iff$ Differ by 2 edges



A Generic Local Search Algorithm

(1) Let $x =$ some initial solution.

(2) While the current solution x has a superior neighboring solution y:

    Set $x := y$

(3) Return the final (locally optimal) solution x

FAQ

Question: How to pick initial solution x?

Answer #1: Use a random solution.
$\Rightarrow$ Run many independent trials of local search, return the best locally optimal solution found.

Answer #2: Use your best heuristics
(i.e., use local search as a postprocessing step to make your solution even better).

Question #2: If there are superior neighboring y, which to choose?

Possible answers: (1) Choose at random, (2) biggest improvement, (3) more complex heuristics.

Question #3: How to define neighborhoods?

Note bigger neighborhoods $\Rightarrow$ slower to verify local optimality, but fewer (bad) local optima

Answer: Find "sweet spot" between solution quality and efficient searchability.

FAQ II

Question: Is local search guaranteed to terminate (eventually)?

Answer: If X is finite and every local step improves some objective function, then yes.

Question: Is local search guaranteed to converge quickly?

Answer: Usually not. [though it often does in practice] (see "smoothed analysis")

Question: Are locally optimal solutions generally good approximations to globally optimal ones?

Answer: No. [To mitigate, run randomized local search many times, remember the best locally optimal solution found]

Local Search

The 2-SAT Problem

Algorithms: Design and Analysis, Part II

2-SAT

Input:

(1) n Boolean variables $x_1, x_2, \ldots, x_n$. (Can be set to TRUE or FALSE)

(2) m clauses of 2 literals each ("literal" $= x_i$ or $\neg x_i$)

Example: $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_4)$

Output: "Yes" if there is an assignment that simultaneously satisfies every clause, "no" otherwise.

Example: "yes", via (e.g.) $x_1 = x_3 =$TRUE and $x_2 = x_4 =$FALSE

(In)Tractability of SAT

2-SAT: Can be solved in polynomial time!

- Reduction to computing strongly connected components (nontrivial exercise)
- "Backtracking" works in polynomial time (nontrivial exercise)
- Randomized local search (next)

3-SAT: Canonical NP-complete

- Brute-force search $\approx 2^n$ time
- Can get time $\approx \left(\frac{4}{3}\right)^n$ via randomized local search [Schöning '02]

Papadimitriou's 2-SAT Algorithm

Repeat $\log_2 n$ times:

- Choose random initial assignment
- Repeat $2n^2$ times:
  - If current assignment satisfies all clauses, halt + report this
  - Else, pick arbitrary unsatisfied clause and flip the value of one of its variables [choose between the two uniformly at random]

Report "unsatisfiable"

Key question: If there's a satisfying assignment, will the algorithm find one (with probability close to 1)?

Obvious good points:

(1) Runs in polynomial time

(2) Always correct on unsatisfiable instances

Algorithms: Design and Analysis, Part II

Local Search

Random Walks on a Line

Random Walks

Key to analyzing Papadimitriou's algorithm:

Random walks on the nonnegative integers (trust me!)

Setup: Initially (at time 0), at position 0.

At each time step, your position goes up or down by 1, with 50/50 probability.

[Except if at position 0, in which case you move to position 1 with 100% probability]

Quiz

Notation: For an integer $n \geq 0$, let T_n = number of steps until random walk reaches position n.

[A random variable, sample space = coin flips at all time steps]

Question: What is $E[T_n]$? (your best guess)

A) $\Theta(n)$

B) $\Theta(n^2)$

C) $\Theta(n^3)$

D) $\Theta(2^n)$

Coming up: $E[T_n] = n^2$.

Analysis of T_n

Let Z_i = number of random walk steps to get to n from i. (Note $Z_0 = T_n$)

Edge cases: $E[Z_n]=0$, $E[Z_0]=1+E[Z_1]$

For $i \in \{1, 2, \ldots, n-1\}$

$$E[Z_i] = \underbrace{\Pr[\text{go left}]}_{1/2} \; \underbrace{E[Z_i \mid \text{go left}]}_{(1+E[Z_{i-1}])} + \underbrace{\Pr[\text{go right}]}_{1/2} \; \underbrace{E[Z_i \mid \text{go right}]}_{(1+E[Z_{i+1}])}$$

$$= 1 + \tfrac{1}{2}E[Z_{i+1}] + \tfrac{1}{2}E[Z_{i-1}]$$

Rearranging: $E[Z_i] - E[Z_{i+1}] = E[Z_{i-1}] - E[Z_i] + 2$

Finishing the Proof of Claim

So:

$$
\begin{array}{rl}
 & \mathsf{E}[Z_0] - \mathsf{E}[\cancel{Z_1}] = 1 \\
 & \mathsf{E}[\cancel{Z_1}] - \mathsf{E}[\cancel{Z_2}] = 3 \\
 & \mathsf{E}[\cancel{Z_2}] - \mathsf{E}[\cancel{Z_3}] = 5 \\
 & \vdots \\
+ & \mathsf{E}[\cancel{Z_{n-1}}] - \mathsf{E}[\cancel{Z_n}^{\,0}] = 2n-1 \\
\hline
 & \mathsf{E}[Z_0] = n^2 \\
 & \quad || \\
 & \mathsf{E}[T_n]
\end{array}
$$

$\frac{n}{2}$ pairs of numbers, each sums to $2n$

QED!

A Corollary

Corollary: $\Pr[T_n > 2n^2] \leq \frac{1}{2}$. (Special case of Markov's inequality)

Proof: Let p denote $\Pr[T_n > 2n^2]$.

$$\begin{aligned}
\text{We have } n^2 &= E[T_n] \\
\text{by last claim} &= \underbrace{\sum_{k=0}^{2n^2} k \Pr[T_n = k]}_{\geq 0} + \sum_{k=2n^2+1}^{\infty} \underbrace{k}_{\geq 2n^2} \Pr[T_n = k] \\
&\geq 2n^2 \Pr[T_n > 2n^2] \\
&= 2n^2 p.
\end{aligned}$$

$\Rightarrow p \leq \frac{1}{2}$ QED!

Algorithms: Design and Analysis, Part II

Local Search

Analysis of Papadimitriou's Algorithm

Papadimitriou's Algorithm

n = number of variables

Repeat $\log_2 n$ times:

- Choose random initial assignment
- Repeat $2n^2$ times:
  - If current assignment satisfies all clauses, halt + report this
  - Else, pick arbitrary unsatisfied clause and flip the value of one of its variables [choose between the two uniformly at random]

Report "unsatisfiable"

Obvious good points:

(1) Runs in polynomial time

(2) Always correct on unsatisfiable instances

Satisfiable Instances

Theorem: For a satisfiable 2-SAT instance with n variables, Papadimitriou's algorithm produces a satisfying assignment with probability $\geq 1 - \frac{1}{n}$.

Proof: First focus on a single iteration of the outer for loop.

Fix an arbitrary satisfying assignment a^*.

Let a_t = algorithm's assignment after inner iteration t ($t = 0, 1, \ldots, 2n^2$) [a random variable]

Let X_t = number of variables on which a_t and a^* agree. ($X_t \in \{0, 1, \ldots, n\}$)

Note: If $X_t = n$, algorithm halts with satisfying assignment a^*.

Proof of Theorem (con'd)

Key point: Suppose a_t not a satisfying assignment and algorithm picks unsatisfied clause with variables x_i, x_j.

Note: Since a^* is satisfying, it makes a different assignment than x_i or x_j (or both).

Consequence of algorithm's random variable flip:

(1) If a^* and a_t differ on <u>both</u> x_i & x_j, then $X_{t+1} = X_t + 1$ (100% probability)

(2) If a^* and a_t differ on exactly one of x_i, x_j, then

$$X_{t+1} = \begin{cases} X_t + 1 & \text{(50\% probability)} \\ X_t - 1 & \text{(50\% probability)} \end{cases}$$

Quiz: Connection to Random Walks

Question: The random variables $X_0, X_1, \ldots, X_{2n^2}$ behave just like a random walk of the nonnegative integers except that:

A) Sometimes move right with 100% probability (instead of 50%)

B) Might have $X_0 > 0$ instead of $X_0 = 0$

C) Might stop early, before $X_t = n$

D) All of the above

Completing the Proof

Consequence: Probability that a single iteration of the outer for loop finds a satisfying assignment is $\geq \Pr[T_n \leq 2n^2] \geq 1/2$

from last video

Thus:

$$
\begin{aligned}
\Pr[\text{algorithm fails}] &\leq \Pr[\text{all } \log_2 n \text{ independent trials fail}] \\
&\leq \left(\tfrac{1}{2}\right)^{\log_2 n} \\
&= \tfrac{1}{n}.
\end{aligned}
$$

QED!