Heuristics: Local Search Algorithms

- Coping with NP-Complete Problem
- Local Search
- Local Search for Max-cut
- Gradient Descent
- Metropolis Algorithm
- Simulated Annealing

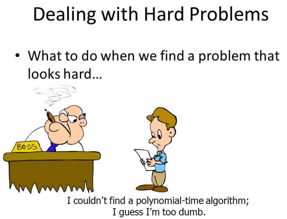
Imdad ullah Khan

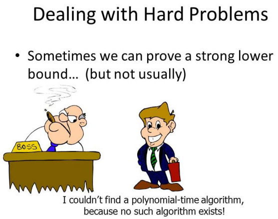
Intractable Problems in Practice

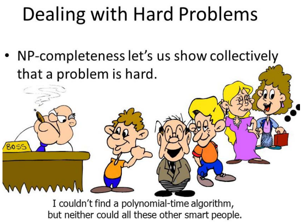
Try to solve a problem through some design paradigm

If fruitless, try to prove that your problem is NP-Hard

You can tell your boss one of the three things!







source: slideplayer.com via Google images

Good theoretical exercise, but the problem doesn't go away

In this lecture we briefly explore what to do in this case

NP-Completeness is not a death certificate, it is the beginning of a fascinating adventure

NP-Hardness

When you prove a problem X to be NP-Hard, then as per the almost consensus opinion of $\mathrm{P} \neq \mathrm{NP}$, it essentially means

1. There is no polynomial time
2. deterministic algorithm
3. to exactly/optimally solve the problem X
4. for all possible input instances

What are the option? Things to consider when your problem is NP-Hard

Coping with NP-HARDNESS

- Do I need to solve the problem for all valid input instances?
  - Sometimes just need to solve a restricted version of the problem -
    ▷ (special cases) that include realistic instances
- Is exponential-time OK for my instances?
  - Exponential-time algorithms are "not slow" ▷ they don't scale well
  - If relevant instances are small, then they may be acceptable
  - Can bring exponent/base of runtime down ▷ $2^n \to 2^{\sqrt{n}}$ or $2^n \to 1.5^n$
- Is non-optimality OK?
  - What if our algorithm is better than others ▷ faster than bruteforce

Coping with NP-Hardness

To cope with NP-Hardness, sacrifice one of these features

Poly-time	Deterministic	Exact/Opt Solution	All cases/ Parameters	Algorithmic Paradigm
✓	✓	✓	✗	Special Cases Algorithms Fixed Parameter Tractability
✓	✓	✗	✓	Approximation Algorithms Heuristic Algorithms
✗	✓	✓	✓	Intelligent Exhaustive Search
✓	✗	$\mathbb{E}$(✓)	✓	Mote Carlo Randomized Algorithm
$\mathbb{E}$(✓)	✗	✓	✓	Las Vegas Randomized Algorithm

- Special cases of input instances (based on structure of a range of parameter(s))
- Approximation algorithms guarantee a bound on suboptimality
- Heuristics algorithms do not have any guarantee
- Randomized algorithms generally used for problems in class P

Coping with NP-Hardness

Approaches to tackle hard problems

1. Special Cases: Relevant structure on which the problem is easy
   - Exact results in poly-time only for special cases or a range of parameters
2. Intelligent Exhaustive Search: Exponential time in worst case
   - The base and/or exponent are usually smaller
   - Could be efficient on typical more realistic instances
   - Backtracking, Brand-and-Bound
3. Nearly exact solutions: Output is *'close'* to exact (optimal) solution
   - Approximation Algorithms: Solutions of guaranteed quality in poly-time
   - Heuristic: Solutions hopefully good in poly-time
4. Randomized Algorithms: Use coin flips for making decisions
   - Typically used for approximation, also used problems in P

Local Search Algorithms

A widely used method to estimate solution of optimization problems

▷ Very rarely there is a guarantee on the quality of solution

1. A local search algorithm begins with a feasible solution
2. At each step it "slightly" modifies the current solution to "improve" it
3. "Slight" modification in the solution means to move to a better solution in the "local neighborhood"
4. A fundamental ingredient is definition of neighborhood in solution space

   ▷ For a candidate solution $x \in X$, identify solutions $y \in X$ as neighbors

The Generic Local Search Algorithm

Explore the solution space in sequential fashion

Move step by step from current solution to a “nearby” one

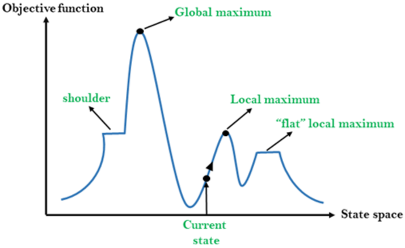
Algorithm Local Search

$s \leftarrow$ some initial solution

while a solution s' in the neighborhood of s is better than s : **do**

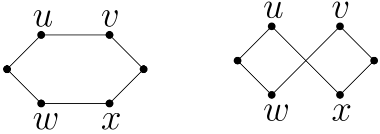
$s \leftarrow s'$ ▷ Key ingredient

return s ▷ Locally optimal solution



Neighborhood for some NP-HARD Problems

1. TSP: two neighboring tours differ from each other by 2 edges



   - Cannot differ in just one edge. why?

2. MAX-CUT: A neighboring CUT can be obtained by moving one vertex from one side to the other in the current CUT

3. K-SAT: Two assignments are neighbors if they differ in the value of a single variable i.e. one can be obtained by flipping just one variable

4. VERTEX-COVER: A neighboring VERTEX-COVER can be obtained by adding or deleting one vertex

Key Characteristics of Local Search Algorithm

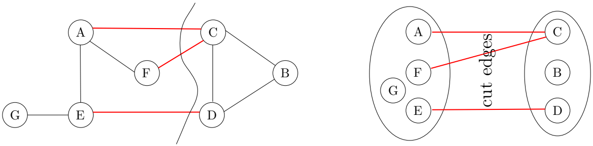
- Unlike greedy algorithms, we do not require maintaining a feasible solution all the time
  - Greedy algorithm typically build the solution bottom-up
- We need to have a solution so we can compute its value in order to determine whether or not to make a move to a neighboring solution
- Easy to design an algorithm
- Generally, No provable guarantees on the quality of the solution
- Can get a local optimum instead of a global optimum
- The larger the neighborhood, the better the resulting solution and the higher the running time

Cuts in Graphs

- Cuts in graphs are very useful structures
- Application in network flows, statistical physics, circuit design, complexity and approximation theory

A cut in G is a subset $S \subset V$

- Denoted as $[S, \overline{S}]$
- $S = \emptyset$ and $S = V$ are trivial cuts, we assume that $\emptyset \neq S \neq V$
- A graph on n vertices has 2^n cuts
- An edge (u, v) is crossing the cut $[S, \overline{S}]$, if $u \in S$ and $v \in \overline{S}$

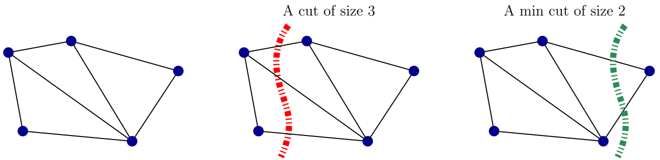


The MAX-CUT(G) problem

A cut in G is a subset $S \subset V$

- Denoted as $[S, \overline{S}]$
- An edge (u, v) is crossing the cut $[S, \overline{S}]$, if $u \in S$ and $v \in \overline{S}$

Size (or cost) of a cut in the number of crossing edges



- In weighted graphs size of cut is the sum of weights of crossing edges

The MAX-CUT(G) problem: Find a cut in G of maximum size?

The MAX-CUT(G) problem

A cut in G is a subset $S \subset V$

- Denoted as $[S, \overline{S}]$
- An edge (u, v) is crossing the cut $[S, \overline{S}]$, if $u \in S$ and $v \in \overline{S}$

Size (or cost) of a cut in the number of crossing edges

- In weighted graph size of cut is the sum of weights of crossing edges

The MAX-CUT(G) problem: Find a cut in G of maximum size?

The decision version of the MAX-CUT(G) problem is NP-COMPLETE

Max-Cut: Local Search Algorithm

Local Search Idea for Max Cut

Begin with a cut and while possible improve it in each step

Algorithm Max-Cut-Local-Search($G = (V, E)$)

$[A, B] \leftarrow$ an arbitrary partition of V

while some node v has higher degree in the other side **do**

MOVE v to the other side ▷ move to neighboring cut

return $[A, B]$

Neighboring cut differ by one vertex

Max-Cut: Local Search Algorithm

Local Search Idea for Max Cut

Begin with a cut and while possible improve it in each step

Algorithm Max-Cut-Local-Search($G = (V, E)$)

$[A, B] \leftarrow$ an arbitrary partition of V

while some node v has higher degree in the other side **do**

MOVE v to the other side ▷ move to neighboring cut

return $[A, B]$

For $v \in V$

$deg_{int}(v)$: the number of neighbors of v in its own part

$deg_{crs}(v)$: the number of neighbors of v in the other part

$$\text{Size of cut } [A, B] = \sum_{v \in A} deg_{crs}(v) = \sum_{v \in B} deg_{crs}(v)$$

MAX-CUT: Local Search Algorithm

Algorithm Max-Cut-Local-Search($G = (V, E)$)

$[A, B] \leftarrow$ an arbitrary partition of V
while some node v has higher degree in the other side **do**
  MOVE v to the other side ▷ move to neighboring cut
return $[A, B]$

- In every iteration the size of cut increase by at least 1
- Hence the algorithm terminates (in $O(|E|)$ steps)

$$f(\text{OPT}(G)) \leq |E| = \frac{1}{2}\sum_{v} deg(v) = \sum_{v} deg_{int}(v) + deg_{crs}(v) \qquad \triangleright \textbf{UB}$$

At end of execution for every $v \in V$ $\quad deg_{crs}(v) \geq deg_{int}(v)$

$$f([A, B]) = \frac{1}{2}\sum_{v} deg_{crs}(v) \geq \frac{1}{2}\sum_{v} \frac{1}{2} deg(v) \geq \frac{|E|}{2} \geq \frac{1}{2} f(\text{OPT}(G))$$

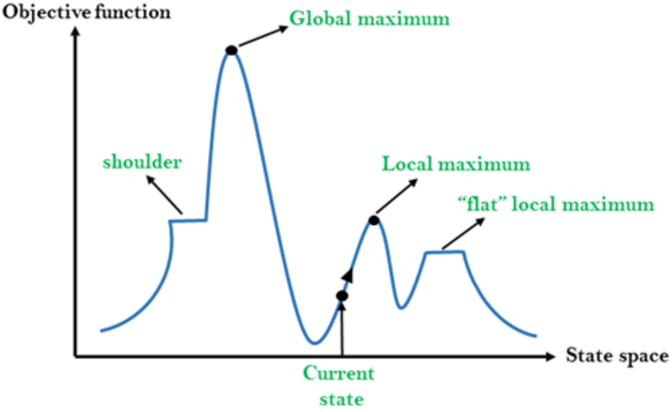
MAX-CUT-LOCAL-SEARCH is a 2-approximate algorithm

Local Search: Issues

- How to pick the initial solution s?
  - Pick a random solution
  - Use best heuristics
- If there several better neighbors s', which one to choose?
  - Choose s' at random
  - Choose the best of s'
- How to define the neighborhoods?
  - The larger the neighborhood the longer it takes
  - Tradeoff between solution quality vs computational resources required
- Is local search guaranteed to converge (eventually)?
  - Yes, if solution space is finite as in MAX-CUT and TSP
- Is local search guaranteed to converge quickly (polynomial time)?
  - Usually not : "Smoothed Analysis"- to estimate runtime
- Are local optima generally good approximations to global optima?
  - No. To mitigate, random (re)start, choose best of many local optima

Dealing with Local Optimum

1. Randomization and restarts
2. Gradient Descent
3. Simulated Annealing



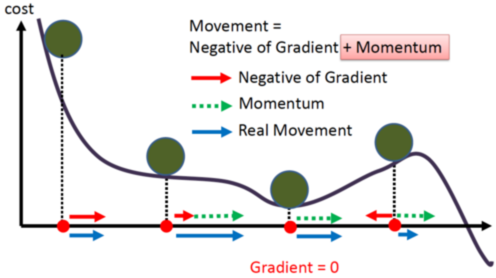
Randomization and Restarts

Randomization and Restarts

- Pick a random initial solution e.g: a random TSP tour, random CUT
- When there are many local optima, randomization make sure there is at least some probability of getting to the better local optima
- Repeat local search several times, each time with random initial solution and return the best solution
- How many iterations to run the local search?
- Your choice: Solution quality vs computational resources required

Gradient Descent or Hill-Climbing Algorithm

- Local search can easily get stuck in a not so good local optima
- In continuous solution spaces, 'infinitely' many local neighbors
- Step-size (displacement of current and next solution) determines
    - Convergence rate
    - and quality of end solution
- When the value function is differentiable both the step-size and direction are determined by the derivative (gradient)



Gradient Descent

Algorithm Gradient-Descent

$x \leftarrow$ INITIAL-SOLUTION

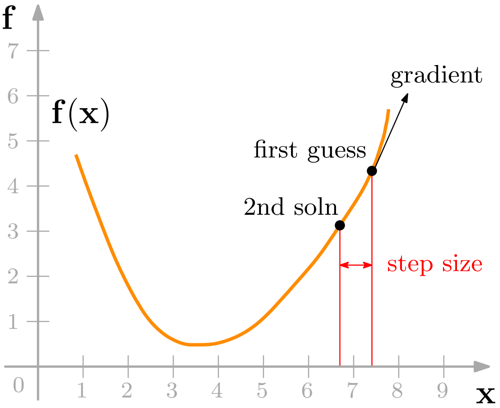
$h \leftarrow$ STEP-SIZE

while $f'(x) \not\approx 0$ **do**

  DIRECTION $= -f'(x)$

  $x \leftarrow x - h\,f'(x)$

  return x



Gradient Descent

Algorithm Gradient-Descent

$x \leftarrow$ INITIAL-SOLUTION

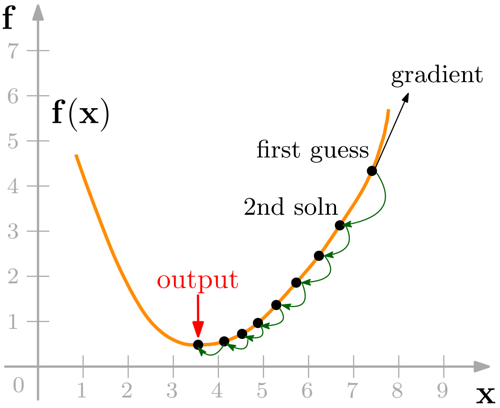
$h \leftarrow$ STEP-SIZE

while $f'(x) \not\approx 0$ **do**

  DIRECTION $= -f'(x)$

  $x \leftarrow x - h\, f'(x)$

  return x



For higher dimension gradient is vector of partial derivatives

$$\nabla f(x,y) = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

Gradient Descent: Step Size

Algorithm Gradient-Descent

$x \leftarrow$ INITIAL-SOLUTION

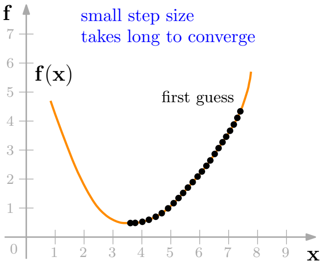
$h \leftarrow$ STEP-SIZE

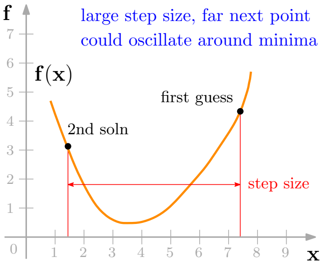
while $f'(x) \not\sim 0$ **do**

DIRECTION $= -f'(x)$

$x \leftarrow x - h\, f'(x)$

return x





Metropolis Algorithm and Simulated Annealing

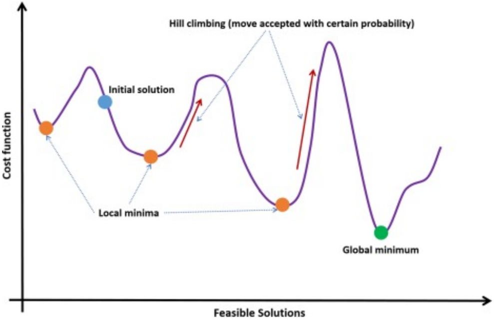
- For many problems, the ratio of number of bad to number of good local optima is large
- Simple RANDOMIZED-RESTART may not be effective
- Metropolis Algorithm combines gradient descent with random walk
- Occasionally allows the move that increases the cost

  ▷ to avoid trapping into a bad local optima

Metropolis Algorithm

Algorithm Metropolis Algorithm

$s \leftarrow$ initial solution
while stopping condition is not met **do**
  $s' \leftarrow$ RANDOM() solution in neighborhood of s
  $\Delta \leftarrow f(s') - f(s)$
  if $\Delta < 0$ **then**
    $s \leftarrow s'$
  else
    $s \leftarrow s'$with pr. $e^{-\Delta/T}$
return s



Metropolis Algorithm

Takes input parameter T (temperature)

Algorithm Metropolis Algorithm

$s \leftarrow$ initial solution
while stopping condition is not met **do**
  $s' \leftarrow \textsc{Random}()$ solution in neighborhood of s
  if $\Delta = f(s') - f(s) < 0$ **then**
    $s \leftarrow s'$
  else
    $s \leftarrow s'$ with probablity $e^{-\Delta/T}$
return s

- If $T = 0$, this is almost gradient descent
- If T is moderately large, then uphill moves are occasionally accepted
- If T is too large, it just becomes random walk

Simulated Annealing

Simulated Annealing executes Metropolis Algorithm but decreases T as the algorithm proceeds

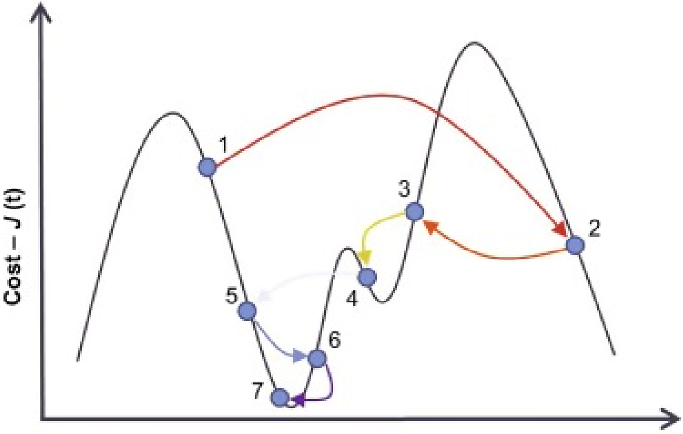
- Initially T is large
  - Solution varies a lot
  - helps escaping local optima
- In the end T is small
  - Solution are nearby
  - helps settles at a (hopefully good) local optima

If T decreases slowly enough, then simulated annealing is very likely to find a global optima

Widely used in VLSI layout, airline scheduling, etc.

Simulated Annealing

Simulated Annealing executes Metropolis Algorithm but decreases T as the algorithm proceeds



Simulated Annealing

- Inspired by the physics of crystallization
- Annealing is a heating method to produce metals and glass with desirable physical properties
- Metal is heated to temperature below its melting point, but high enough so the crystalline lattice structures within the metal break apart
- With gradual cooling the crystalline structures reform and grow larger
- These structures correspond to a low energy state