Local Search

Local Search

1. Introduction
2. The gradient descent Algorithm
3. The Metropolis Algorithm and Simulated Annealing
4. Application of Local Search to the Hopfield Neural Networks
5. Maximum-Cut Approximation
6. Best Response Dynamics and Nash Equilibrium

1.Introduction

Local Search describes any algorithm that explores the space of possible solutions in a sequential way, moving in one step from a current solution to a nearby one.

+

- It is not difficult to design a local search approch to almost any hard problem.

--

- It is often very difficult to say anything provable about the solutions that it finds.

1.Introduction

Local Search approch is used as an approch to solve **Computational optimization problems.**

In a typical such problem we have :

- Large set S of possible solutions
- A cost function c(s) that measures the quality of a solution s
- The goal is to find s* for which c(s*) is as small/large as possible

- **A neighbor relation on the set of solutions** :

s’ is a neighboring solution of s (s’ ≈ s) if s’ can be obtained with a small modification of s.

1.Introduction

A Local Search approch follows this general scheme :

- It maintains **a current solution** s at all times.
- **Chooses a neighbor** solution s' to s and sets it as the new current solution.
- It saves the best solution cost and try compare it to the current one.

The crux of a Local Search Algorithm is in two crucial points :

- **The choice of the neighbor relation**
- **The rule for choosing a neighboring solution**

2.The gradient Descent Algorithm

The **Gradient Descent Algorithm** is a very simple algorithm.

It starts with a trivial current solution and do the following :
Let s denote the current solution. If there is a neighbor s' of s with strictly lower cost, then choose the neighbor whose cost is as small as possible. Otherwise terminate the algorithm.

This algorithm terminates precisely at solutions that are **Local Minimum**

2.The gradient Descent Algorithm

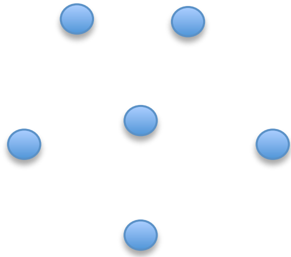
Application of this algorithm to a concrete problem **: Vertex Cover Problem** :

We are given a graph G = (V, E) , The set of possible solutions S is the set of all subset s of V that form Vertex covers.

- For any s of S , the cost C(s) is defined as the size of s
- s ≈ s' if s' can be obtained from s by adding or deleting a single node

In this problem, the algorithm starts with the trivial solution V.

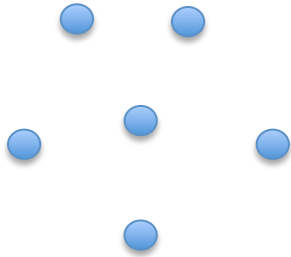
2.The gradient Descent Algorithm



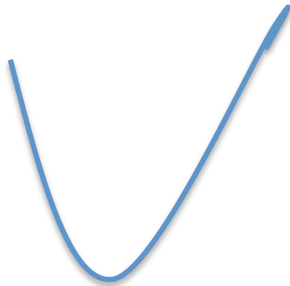
In a graph with no edges, the best solution is the empty set.

$C(s) = 0$

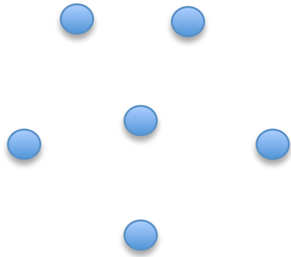
2.The gradient Descent Algorithm



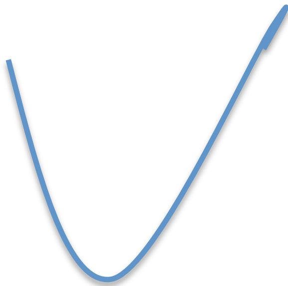
The algorithm finds the empty set.
In this case, the local minimum is
also a global minimum.

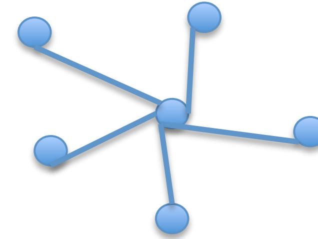


2.The gradient Descent Algorithm



The algorithm finds the empty set. In this case, the local minimum is also a global minimum.

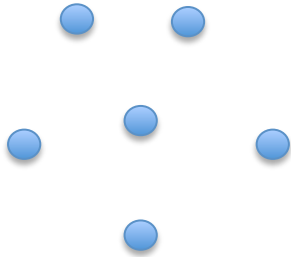




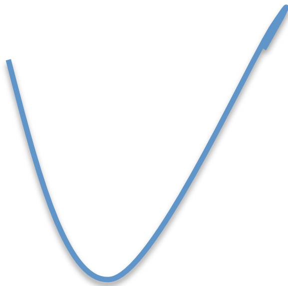
In a star graph , the best solution is the {x}.

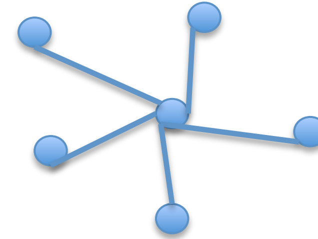
C(s) = 1

2.The gradient Descent Algorithm

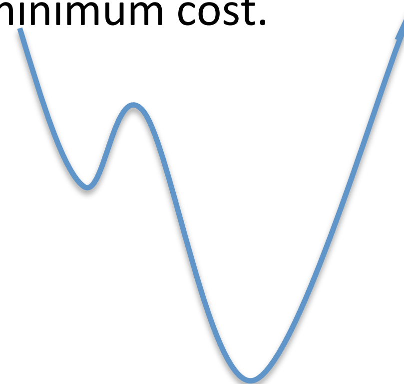


The algorithm finds the empty set.
In this case, the local minimum is also a global minimum.





The solution found depends on the first deleted node.
A local minimum has a high cost relative to the global minimum cost.



3. Metropolis Algorithm and Simulated Annealing

The Metropolis Algorithm keeps the same neighbor relation, but changes the rule of choosing a neighboring solution.

The Metropolis algorithm is biased toward downhill moves , but will also accept uphill moves with certain probablity.
→ So it can correct some wrong choices it took.

3. Metropolis Algorithm and Simulated Annealing

Start with a solution s_0 and constants k and T

In one step :

let s be the current solution

let s' be chosen uniformly at random among the neighbors of s.

if $c(s') \leq c(s)$ then update s := s'

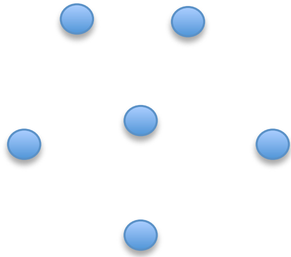
else

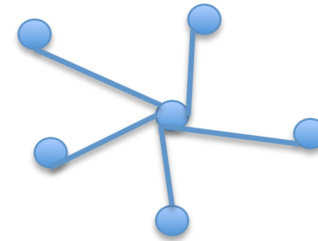
with probablity $\exp(- (c(s')-c(s)) / k.T)$ update s:=s'

otherwise leave s unchanged

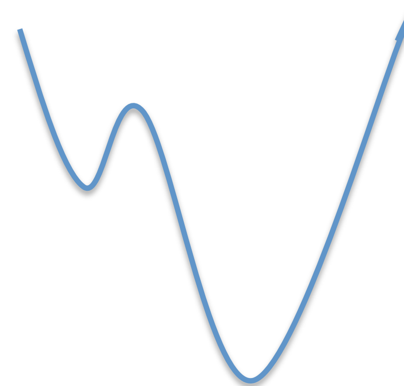
end if

3. Metropolis Algorithm and Simulated Annealing

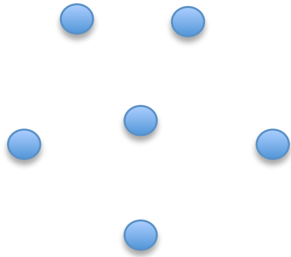




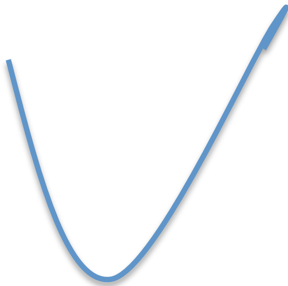
If the central node is deleted, it can be put back with positive probability

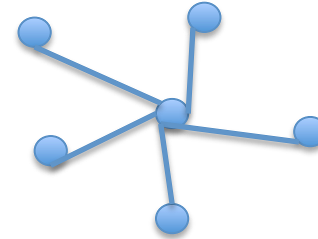


3. Metropolis Algorithm and Simulated Annealing

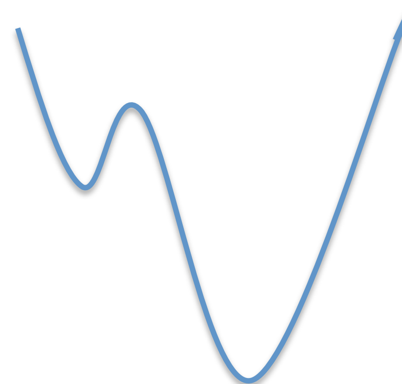


When it reaches a solution with $c(s) = c << n$, the neighboring solution will have a higher cost and can be accepted with a positive probability.





If the central node is deleted, it can be put back with positive probability



3. Metropolis Algorithm and Simulated Annealing

The probability of having an uphill move is $\exp(- (c(s')-c(s)) / k.T)$

- if T is small , uphill moves are never accepted : Gradient Descent Algorithm
- if T is too large, Metropolis is having random walk , independently from the cost.

3. Metropolis Algorithm and Simulated Annealing

The probability of having an uphill move is $\exp(- (c(s')-c(s)) / k.T)$

- if T is small , uphill moves are never accepted : Gradient Descent Algorithm
- if T is too large, Metropolis is having random walk , independently from the cost.

Simulated Annealing :

It executes the Metropois Algorithm but decreases T as the algorithm.

- At the begenning, T is large : helps escaping from local minimum
- At the end, T is small : helps stucking in a minimum.

4. Hopfield Neural Networks

We are given a graph $G = (V,E)$, every edge e has a weight w_e.
A configuration s of the network is assigning a value s_u for each node u.
s_u can be -1 or 1.

An edge $e = (u,v)$ is good either : $w_e < 0$ and $s_u = s_v$
or : $w_e > 0$ and $s_u \neq s_v$ ($w_e . s_u . s_v < 0$)
Otherwise e is bad.

4. Hopfield Neural Networks

We are given a graph $G = (V,E)$, every edge e has a weight w_e.
A configuration s of the network is assigning a value s_u for each node u.
s_u can be -1 or 1.

An edge $e = (u,v)$ is good if: $w_e < 0$ and $s_u = s_v$
or : $w_e > 0$ and $s_u \neq s_v$ $\quad (w_e . s_u . s_v < 0)$
Otherwise e is bad.

A node u is satisfied if : $\sum_{e=(u,v)} w_e . s_u . s_v \leq 0 \quad \left(\sum_{e \text{ good}} |w_e| > \sum_{e \text{ bad}} |w_e| \right)$

Finally, we say a configuration is stable if all nodes are satisfied

Result :

Every Hopfield network has a stable configuration, and which can be found in time polynomial in n and W = $\sum$ |we|

Result :

Every Hopfield network has a stable configuration, and which can be found in time polynomial in n and W = $\sum$ |we|

The stable configuration in fact arise as the local optimum of a certain local search procedure.

The State-Flipping Algorithm :

```
While the current configuration is not stable
        There must be an unsatisfied node
        Choose an unsatisfied node u
        Flip the state of u
End while
```

The State-Flipping Algorithm :

While the current configuration is not stable
    There must be an unsatisfied node
    Choose an unsatisfied node u
    Flip the state of u
End while

- If the algorithm terminates, we will have a stable configuration.

- To proove the algorithm terminates, we will look for a measure (progress) that increases in every flipping step and is upper bounded.

We define $\Phi(s) = \sum_{e\ good} |w_e|$

- Φ in increasing
- Φ in upper bounded by W

We define $\Phi(s) = \sum_{e\ good} |w_e|$

- Φ in increasing
- Φ in upper bounded by W

→ The algorithm terminates in at most W iteration, every iteration take a number of operations polynomial in n.

4. Hopfield Neural Networks

We define $\Phi(s) = \sum_{e\ good} |w_e|$

- Φ in increasing
- Φ in upper bounded by W

→ The algorithm terminates in at most W iteration, every iteration take a number of operations polynomial in n.

→ The existance proof for stable configurations was really about local search

- We set up a function Φ to maximize
- Configurations are the possible solutions for the problem
- We define a neighbor relation between solutions.
- We identified that every local maximum for Φ is a stable configuration.

5. Maximum-Cut Approximation

We are given a graph $G = (V,E)$, every edge e has a weight $w_e>0$.

For a partition (A,B) of V, we denote W(A,B) the total weight of edges with one end in A and one end in B.

The goal is to find a partition that maximizes W(A,B)

5. Maximum-Cut Approximation

We are given a graph G = (V,E) , every edge e has a weight w_e>0.

For a partition (A,B) of V, we denote W(A,B) the total weight of edges with one end in A and one end in B.

The goal is to find a partition that maximizes W(A,B)

→ **A close relation with Hopefield Neural Networks**

- Configuration in nodes state corresponds to a partition (A,B)
- A node is in A iff its state is -1
- A node is in B iff its state is 1

5. Maximum-Cut Approximation

We are given a graph G = (V,E) , every edge e has a weight $w_e>0$.

For a partition (A,B) of V, we denote W(A,B) the total weight of edges with one end in A and one end in B.

The goal is to find a partition that maximizes W(A,B)

→ **A close relation with Hopefield Neural Networks**

- Configuration in nodes state corresponds to a partition (A,B)
- A node is in A iff its state is -1
- A node is in B iff its state is 1
- W(A,B) is exactly Φ
- A flipping step corresponds to moving a node from A to B or from B to A

5. Maximum-Cut Approximation

We are given a graph G = (V,E) , every edge e has a weight $w_e>0$.

For a partition (A,B) of V, we denote W(A,B) the total weight of edges with one end in A and one end in B.

The goal is to find a partition that maximizes W(A,B)

→ **A close relation with Hopefield Neural Networks**

- Configuration in nodes state corresponds to a partition (A,B)
- A node is in A iff its state is -1
- A node is in B iff its state is 1
- W(A,B) is exactly Φ
- A flipping step corresponds to moving a node from A to B or from B to A

→ Single-flip neighborhood algorithm finds a local maximum of W(A,B)

Result :

Let (A,B) be a local maximum for Maximum-cut obtained by the single-flip neighborhood , and (A*,B*) be a global maximum.
Then $W(A,B) \geq W(A^*,B^*) / 2$

6. Best Response Dynamics and Nash Equilibria

So far, we are considering Local Search approch to solve **a single objective**, but in some problems, we can consider it to solve **different objectives.**

→ Different agents, each one has his own objective

So far, we are considering Local Search approch to solve **a single objective**, but in some problems, we can consider it to solve **different objectives.**

→ Different agents, each one has his own objective

Multicast Rooting Problem :
Let a Graph G = (V,E) each edge e has a cost $c_e > 0$,
and let $t_1, t_2 \dots t_k$ k agents each agent residing in one node ,
a server s is located in one node.

→ Each agent wants to construct a path to the server

6. Best Response Dynamics and Nash Equilibria

So far, we are considering Local Search approch to solve **a single objective**, but in some problems, we can consider it to solve **different objectives.**

→ Different agents, each one has his own objective

Multicast Rooting Problem :

Let a Graph G = (V,E) each edge e has a cost $c_e>0$,
and let $t_{1,} t_2 \dots t_k$ k agents each agent residing in one node ,
a server s is located in one node.

→ Each agent wants to construct a path to the server

Each agent pays c_e/n instead of c_e , where n is the number of agents using the path.

6. Best Response Dynamics and Nash Equilibria

We introduce new concepts , related to the context of multi-objective problems:

- **Best Response Dynamics** : Each agent iscontinually prepared to improve his solution in response to changes made by other(s) agents.
- **Nash Equilibrium** : a stable solution when all agent don't need to change their actual solutions.
-

6. Best Response Dynamics and Nash Equilibria

We introduce new concepts , related to the context of multi-objective problems:

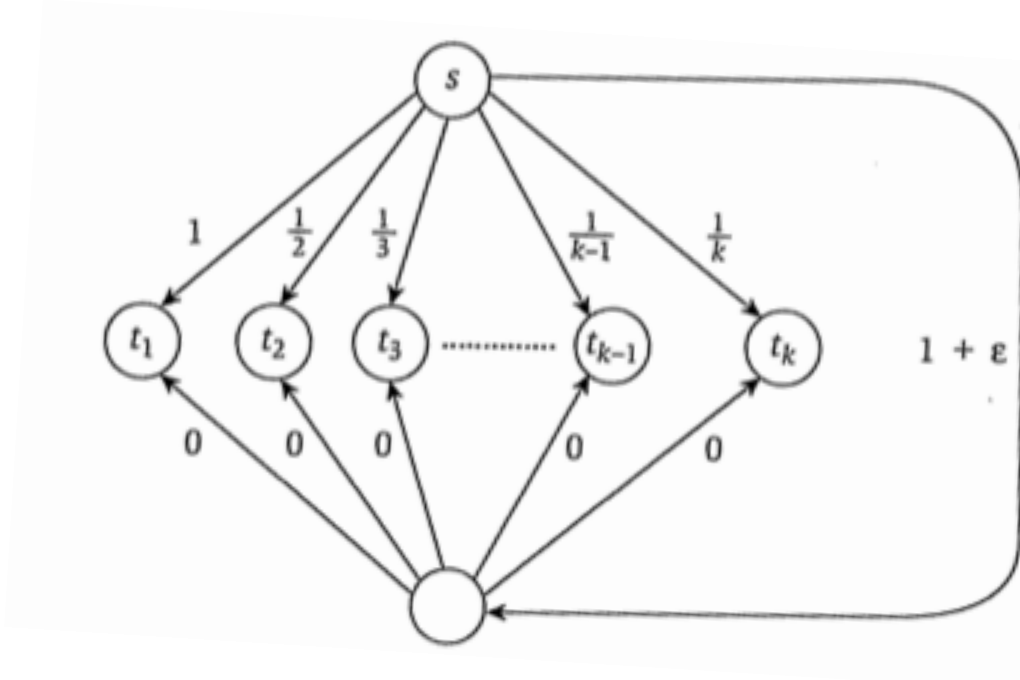
- **Best Response Dynamics** : Each agent iscontinually prepared to improve his solution in response to changes made by other(s) agents.
- **Nash Equilibrium** : a stable solution when all agent don't need to change their actual solutions.

→ **Social Optimum : **It is a solution that minimizes the total cost of all the agents.

6. Best Response Dynamics and Nash Equilibria

- Some Nash Equilibrium can have a total cost worse than the Social Optimum
- Question : is a Social Optimum always a Nash Equilibium solution ?

- Some Nash Equilibrium can have a total cost worse than the Social Optimum
- Question : is a Social Optimum necessarily a Nash Equilibium solution ?



The answer is no

→ **Price of stability** : cost of the best Nash Equilibrium/ cost of Social Optimum

6. Best Response Dynamics and Nash Equilibria

Question : Is a Nash Equilibrium always exist ?

Best Response Dynamics always leads to a set of paths that forms a Nash Equilibria.

6. Best Response Dynamics and Nash Equilibria

Question : Is a Nash Equilibrium always exist ?

Best Response Dynamics always leads to a set of paths that forms a Nash Equilibria.

$\Phi (P_1,P_2,\ldots,P_k) = \sum_e c_e \, h(x_e)$ where $h(x) = 1 + 1/2 + \ldots + 1/x$

Φ decreases strictly in a finite set of values and is bounded by 0.