SIMULATED ANNEALING

Advanced Algorithms Course

Motivation

- Finding a good solution to an optimization problem.
- Good does not mean perfect.
- Trying to minimize the cost of travelling is a good example

Hill climbing

- Our first approach will be the rather naïve **Hill Climbing** algorithm.
- The basic idea of Hill Climbing is:
  1. Choose a starting point.
  2. Try to improve solution
  3. If no further improvements are possible then stop.

Hill climbing

- It is like the algorithm is climbing a hill and tries to find the top, where the best solution is.
- It can only take steps to take it uphill.
- This means it can stop before finding the best solution.

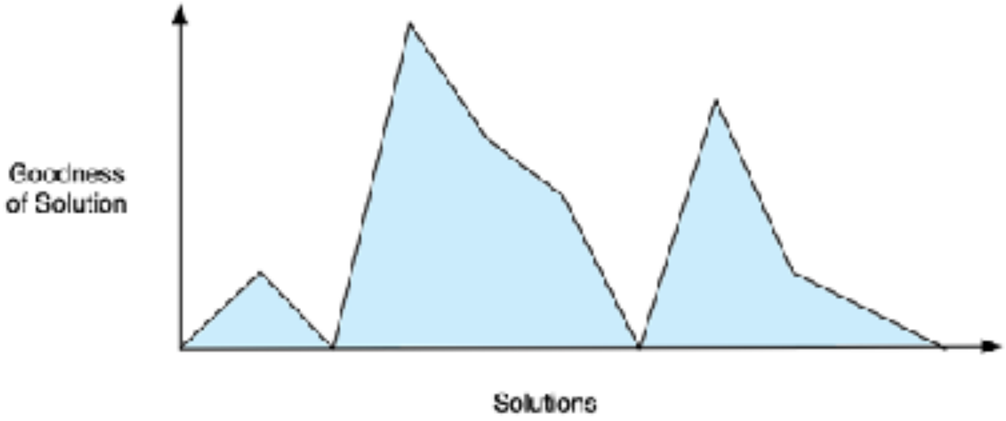
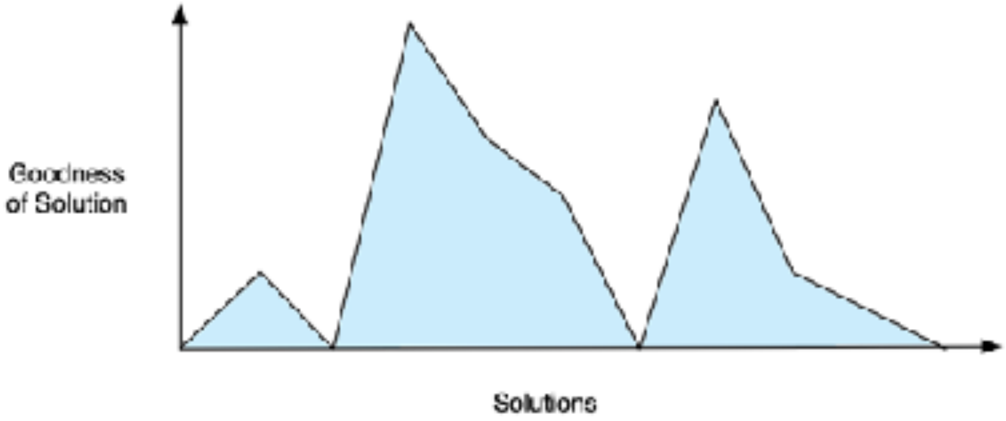


Image source: http://katrinaeg.com/simulated-annealing

Hill climbing

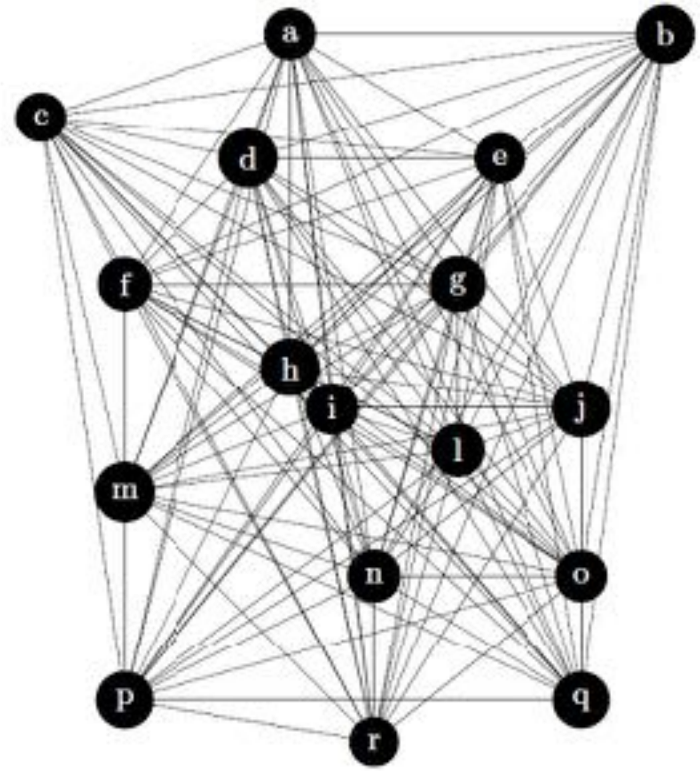
- The biggest hill is the **global maximum.**
- The top of any other hill is a **local maximum.**



Hill climbing

- We need to define the initial solution:
  - Randomly generate a sequence o cities.

```
city(a,45,95).
city(b,90,95).
city(c,15,85).
city(d,40,80).
city(e,70,80).
city(f,25,65).
city(g,65,65).
city(h,45,55).
city(i,5,50).
city(j,80,50).
city(l,65,45).
city(m,25,40).
city(n,55,30).
city(o,80,30).
city(p,25,15).
city(q,80,15).
city(r,55,10).
```

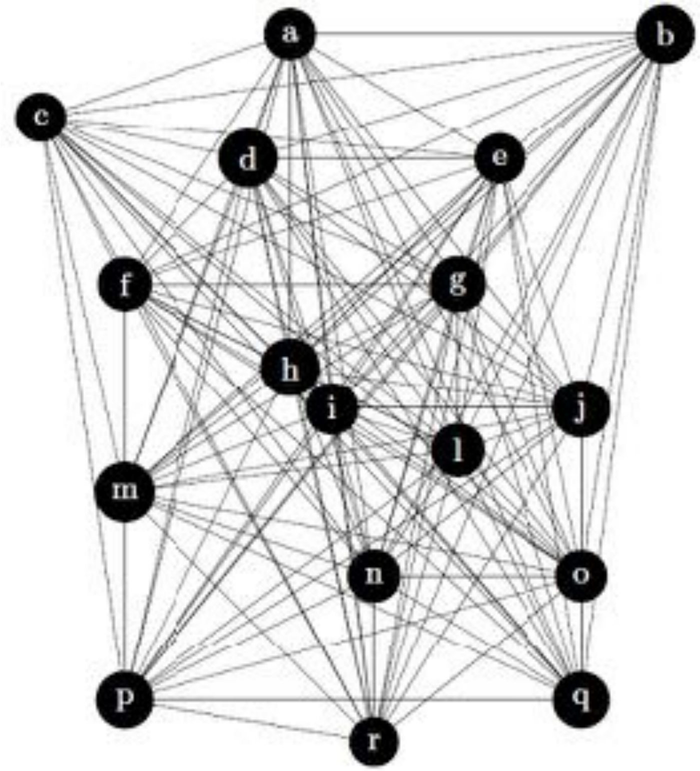


Hill climbing

- We need to define the initial solution:
  - Randomly generate a sequence of cities.

```
initialSolution(Orig,L):-
        findall(X,(city(X,_,_),X\=Orig),L1),
        random_permutation(L1,L).
```

```
?-initialSolution(a,L)
```



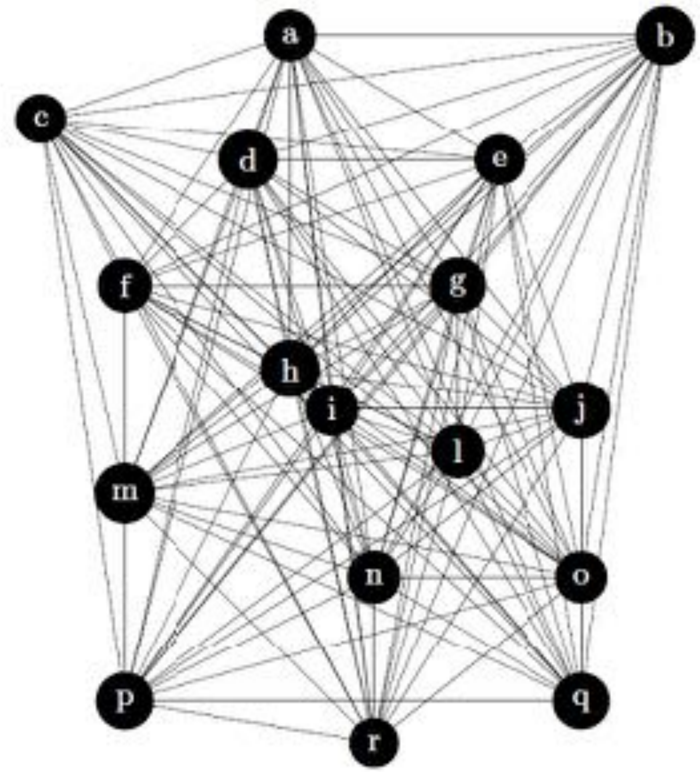
Hill climbing

- We need to define the initial solution:
  - Randomly generate a sequence of cities.

```
initialSolution(Orig,L):-
            findall(X,(city(X,_,_),X\=Orig),L1),
            random_permutation(L1,L).
```

```
?-initialSolution(a,L)
```

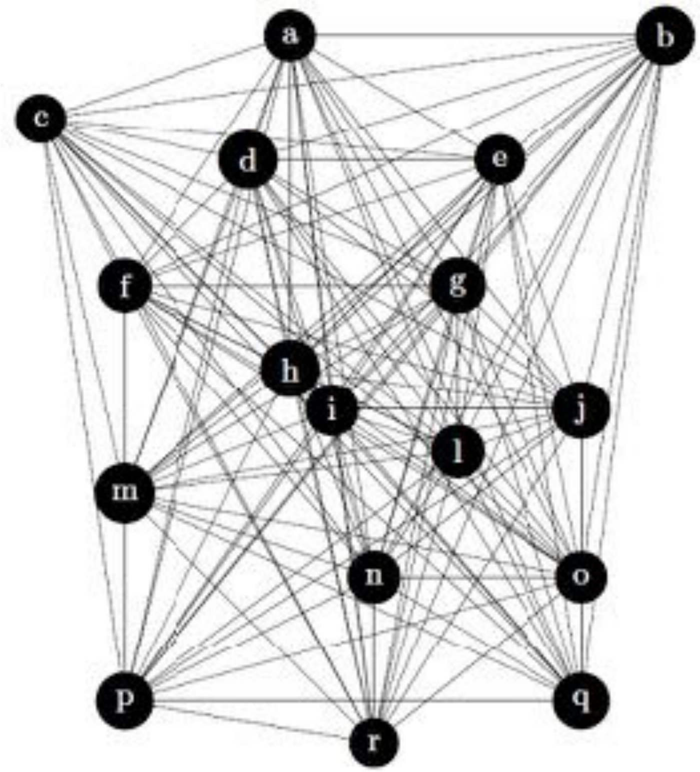
```
L = [b,r,c,d,f,l,g,p,n,e,i,m,h,j,q,o]
```



Hill climbing

- Now, we need to evaluate the solution:
  - Compute the total distance.

```
distance(C1,C2,Dist):-
            city(C1,X1,Y1),
            city(C2,X2,Y2),
            DX is X1-X2,
            DY is Y1-Y2,
            Dist is sqrt(DX*DX+DY*DY).
```

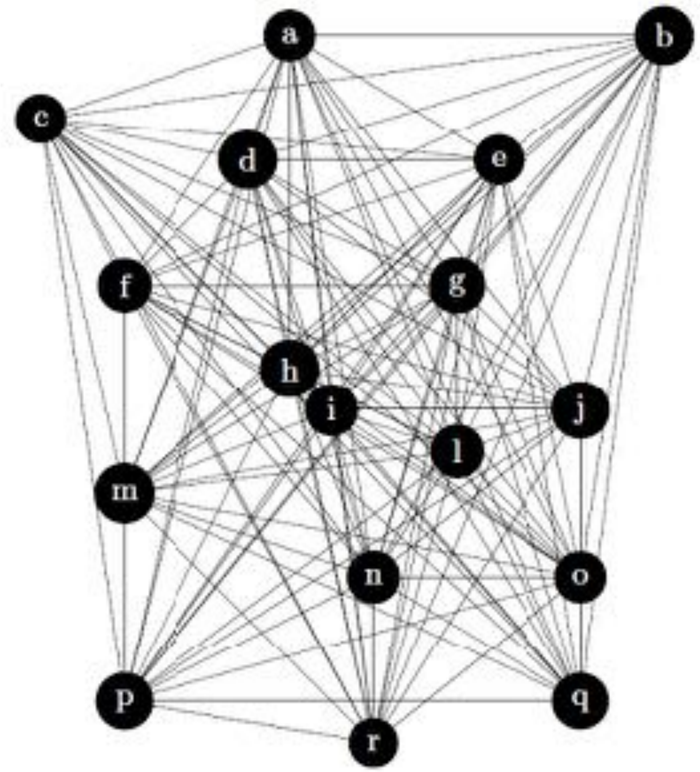


Hill climbing

- Now, we need to evaluate the solution:
  - Compute the total distance.

```
distance(C1,C2,Dist):-
            city(C1,X1,Y1),
            city(C2,X2,Y2),
            DX is X1-X2,
            DY is Y1-Y2,
            Dist is sqrt(DX*DX+DY*DY).
```

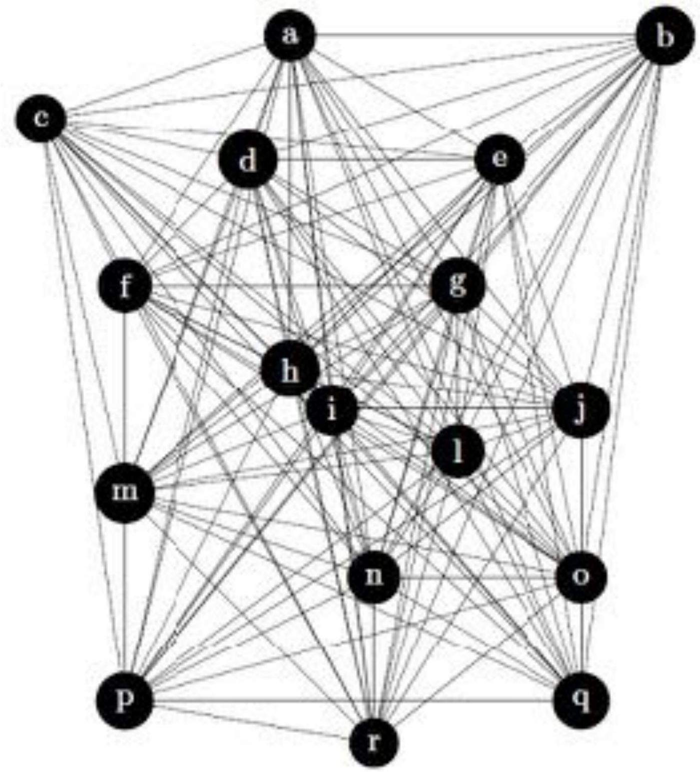
```
?-distance(a,n,D).
D = 65.76
```



Hill climbing

- Now, we need to evaluate the solution:
    - Compute the total distance.

```
totalDistance([],0).
totalDistance([_],0).
totalDistance([X,Y | L],T):-
              distance(X,Y,Cl),
              totalDistance([Y | L],T1),
              T is T1 + Cl.
```

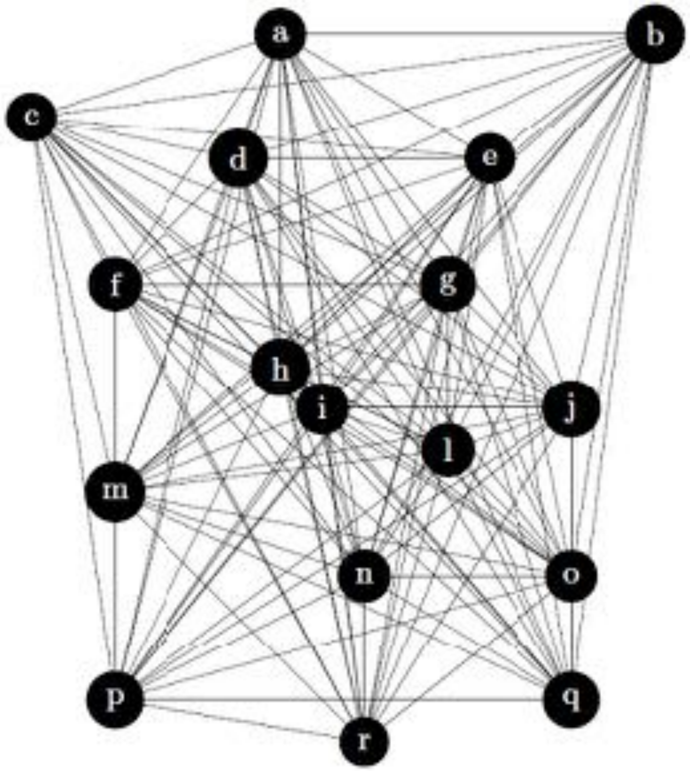


Hill climbing

- Now, we need to evaluate the solution:
  - Compute the total distance.

```
totalDistance([],0).
totalDistance([_],0).
totalDistance([X,Y | L],T):-
            distance(X,Y,Cl),
            totalDistance([Y | L],T1),
            T is T1 + Cl.
```

```
?-totalDistance([a,b,r,c,d,f,l,g,p,n,e,i,m,h,j,q,o,a],D).
D = 761
```



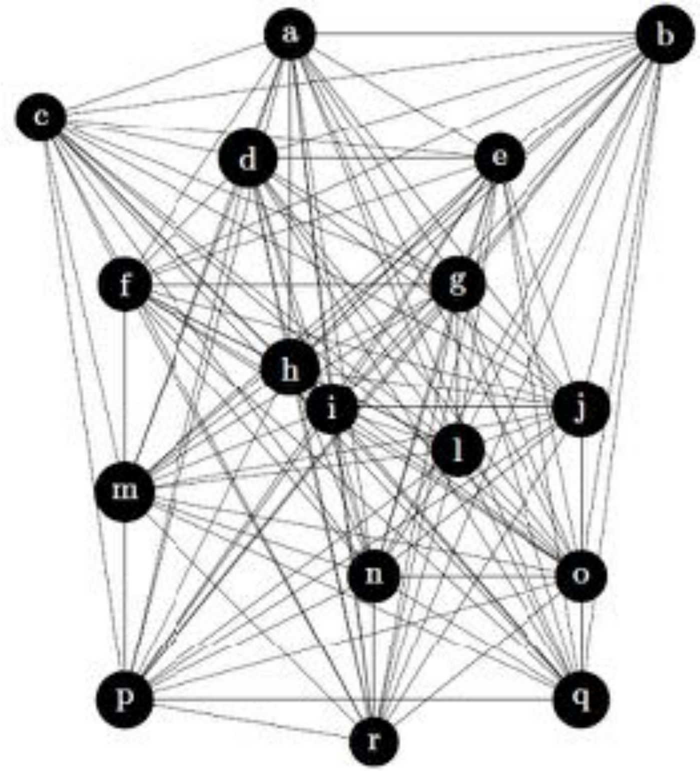
Hill climbing

- Now, we need to evaluate the solution:
  - Compute the total distance.

```
totalDistance([],0).
totalDistance([_],0).
totalDistance([X,Y|L],T):-
          distance(X,Y,Cl),
          totalDistance([Y|L],T1),
          T is T1 + Cl.
```

```
?-totalDistance([a,b,r,c,d,f,l,g,p,n,e,i,m,h,j,        D).
D = 761
```

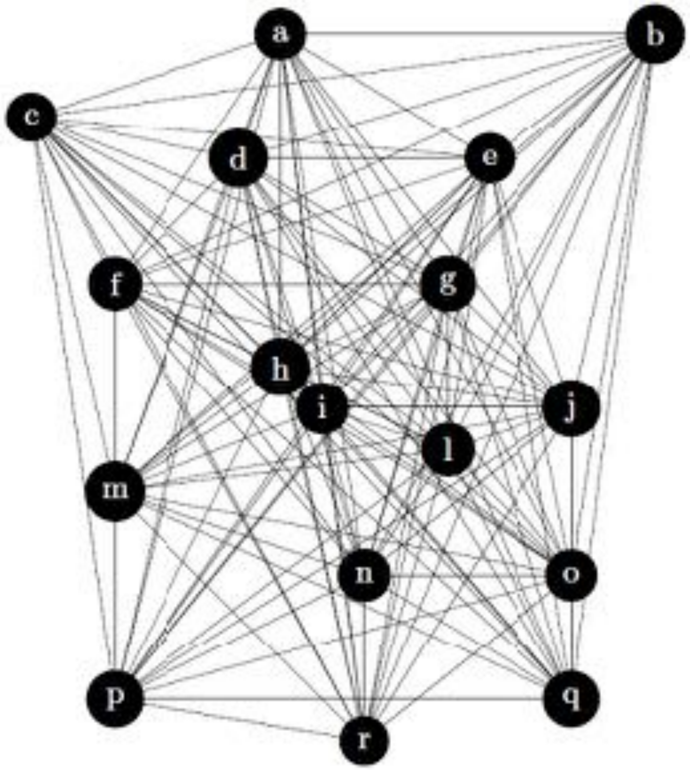
Goal: Minimize this!



Hill climbing

- Compute adjacent solutions:
  - Lets swap two random elements

```
newAdjacent(S1,Sn):-
            length(S1,T1),
            random_between(1,T1,Pos1),
            random_between(1,T1,Pos2),
            nth1(Pos1,S1,E1),
            nth1(Pos2,S1,E2),
            removeElementPos(Pos1,S1,S2),
            insertElementPos(Pos1,E2,S2,S3),
            removeElementPos(Pos2,S3,S4),
            insertElementPos(Pos2,E1,S4,Sn).
```

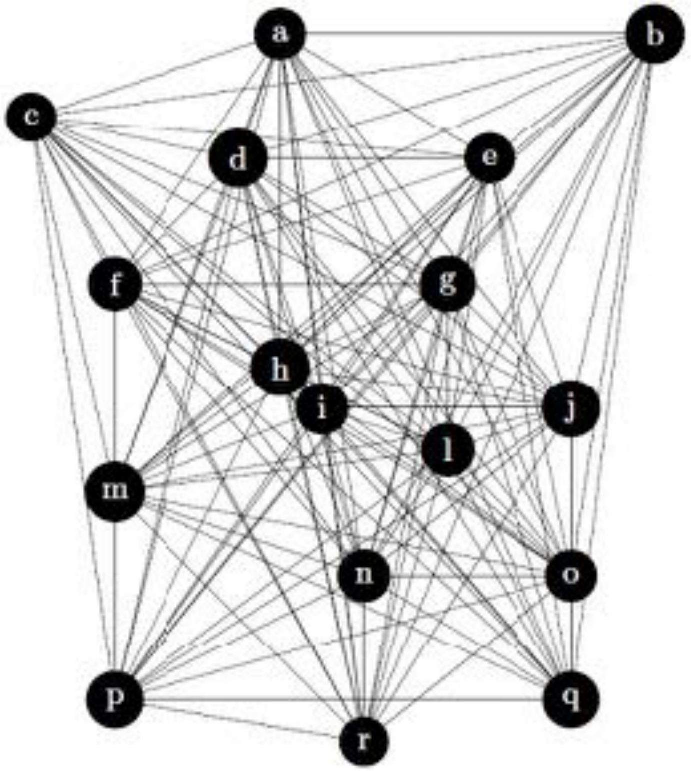


Hill climbing

- Compute adjacent solutions:
  - Lets swap two random elements

```
newAdjacent(S1,Sn):-
                length(S1,T1),
                random_between(1,T1,Pos1),
                random_between(1,T1,Pos2),
                nth1(Pos1,S1,E1),
                nth1(Pos2,S1,E2),
                removeElementPos(Pos1,S1,S2),
                insertElementPos(Pos1,E2,S2,S3),
                removeElementPos(Pos2,S3,S4),
                insertElementPos(Pos2,E1,S4,Sn).
```

```
?-newAdjacent([b,r,c,d,f,l,g,p,n,e,i,m,h,j,q,o],NA).
NA = [b,r,c,d,h,l,g,p,n,e,i,m,f,j,q,o]
```

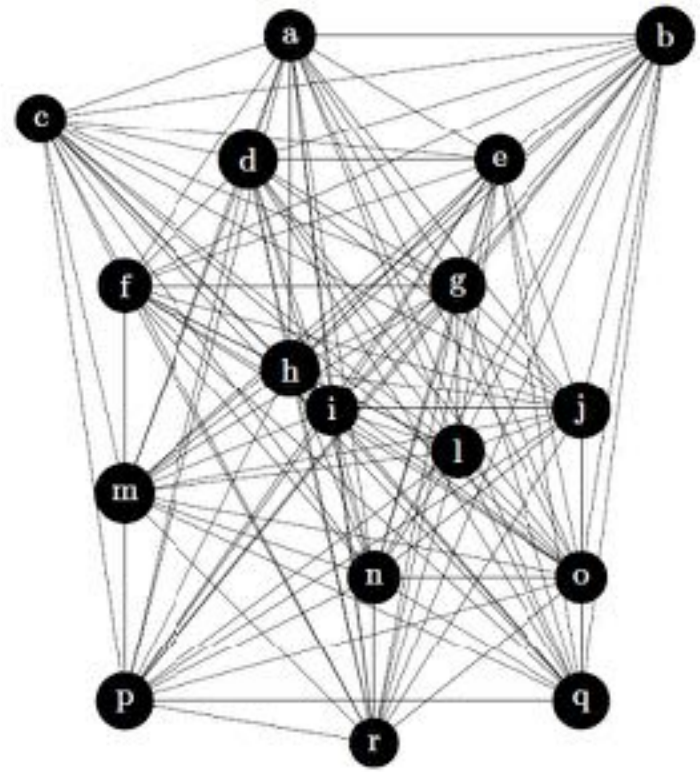


Hill climbing

- Compute adjacent solutions:
  - Lets swap two random elements

```
newAdjacent(S1,Sn):-
                length(S1,T1),
                random_between(1,T1,Pos1),
                random_between(1,T1,Pos2),
                nth1(Pos1,S1,E1),
                nth1(Pos2,S1,E2),
                removeElementPos(Pos1,S1,S2),
                insertElementPos(Pos1,E2,S2,S3),
                removeElementPos(Pos2,S3,S4),
                insertElementPos(Pos2,E1,S4,Sn).
```

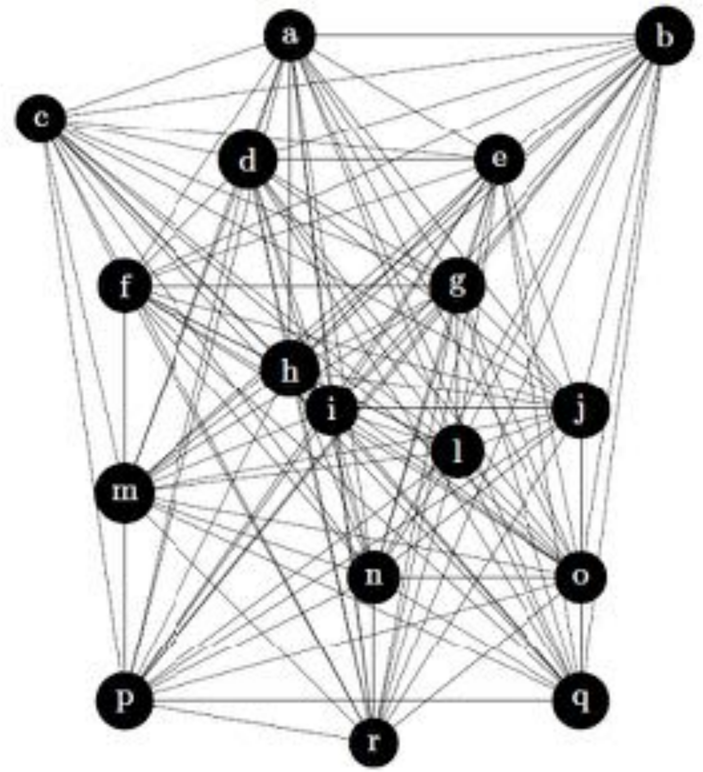
```
?-newAdjacent([b,r,c,d,f,l,g,p,n,e,i,m,h,j,q,o],NA).
        NA = [b,r,c,d,h,l,g,p,n,e,i,m,f,j,q,o]
```



Hill climbing

- Compute adjacent solutions:
  - Now we can compute some adjacents (let's do it 10 times)

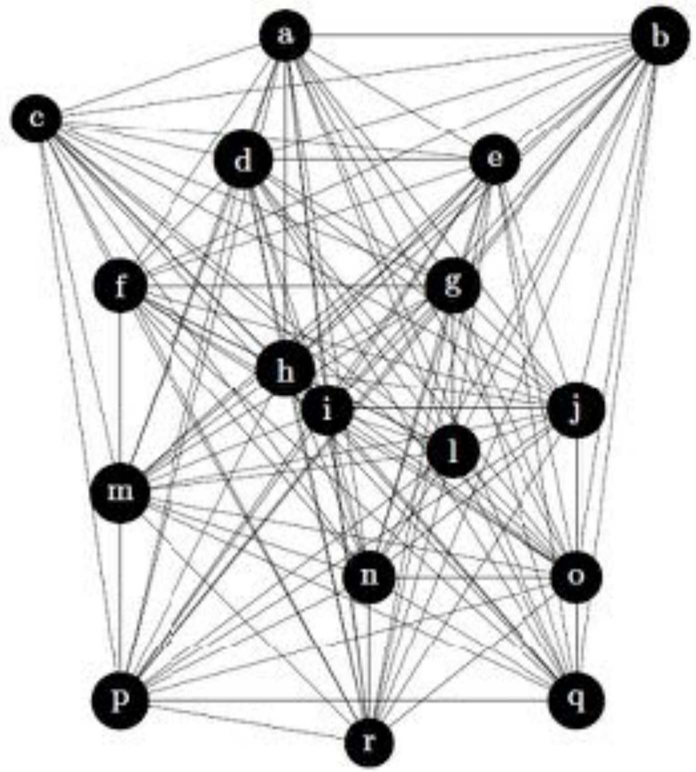
[b,r,c,d,h,l,g,p,n,e,i,m,f,j,q,o]
[b,r,c,d,l,f,g,p,n,e,i,m,h,j,q,o]
[b,r,c,d,f,l,g,p,n,e,i,h,m,j,q,o]
[b,r,c,d,f,q,g,p,n,e,i,m,h,j,l,o]
[b,r,l,d,f,c,g,p,n,e,i,m,h,j,q,o]
[b,r,c,d,f,l,g,p,n,e,i,m,h,j,q,o]
[b,r,c,d,f,l,g,h,n,e,i,m,p,j,q,o]
[b,r,c,g,f,l,d,p,n,e,i,m,h,j,q,o]
[b,h,c,d,f,l,g,p,n,e,i,m,r,j,q,o]
[b,r,c,d,n,l,g,p,f,e,i,m,h,j,q,o]



Hill climbing

- Compute adjacent solutions:
  - And add the origin/destination

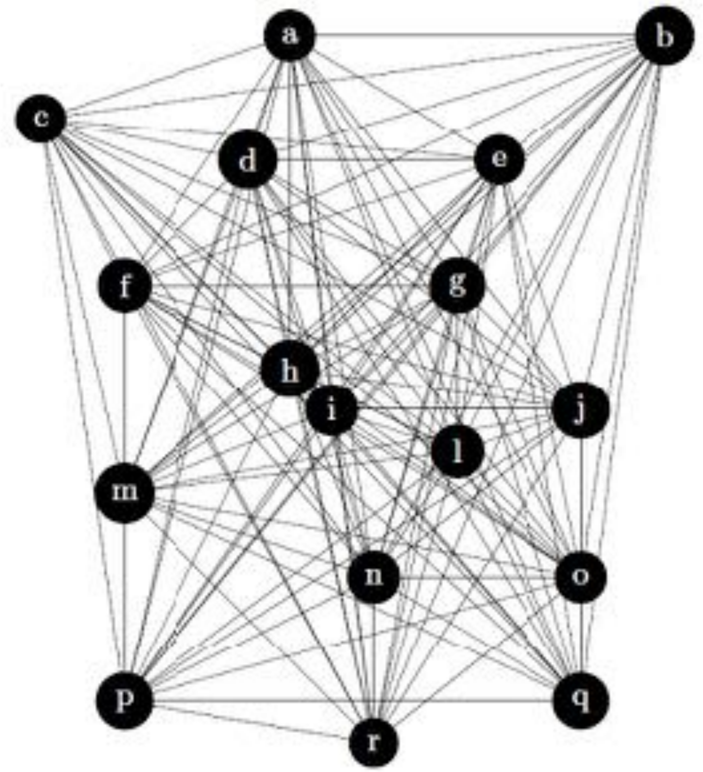
[a,b,r,c,d,h,l,g,p,n,e,i,m,f,j,q,o,a]
[a,b,r,c,d,l,f,g,p,n,e,i,m,h,j,q,o,a]
[a,b,r,c,d,f,l,g,p,n,e,i,h,m,j,q,o,a]
[a,b,r,c,d,f,q,g,p,n,e,i,m,h,j,l,o,a]
[a,b,r,l,d,f,c,g,p,n,e,i,m,h,j,q,o,a]
[a,b,r,c,d,f,l,g,p,n,e,i,m,h,j,q,o,a]
[a,b,r,c,d,f,l,g,h,n,e,i,m,p,j,q,o,a]
[a,b,r,c,g,f,l,d,p,n,e,i,m,h,j,q,o,a]
[a,b,h,c,d,f,l,g,p,n,e,i,m,r,j,q,o,a]
[a,b,r,c,d,n,l,g,p,f,e,i,m,h,j,q,o,a]



Hill climbing

- Compute adjacent solutions:
  - Are they better than my original path (**761**)?

[(764,[a,b,r,c,d,h,l,g,p,n,e,i,m,f,j,q,o,a]),
(803,[a,b,r,c,d,l,f,g,p,n,e,i,m,h,j,q,o,a]),
(799,[a,b,r,c,d,f,l,g,p,n,e,i,h,m,j,q,o,a]),
(810,[a,b,r,c,d,f,q,g,p,n,e,i,m,h,j,l,o,a]),
(741,[a,b,r,l,d,f,c,g,p,n,e,i,m,h,j,q,o,a]),
(761,[a,b,r,c,d,f,l,g,p,n,e,i,m,h,j,q,o,a]),
(742,[a,b,r,c,d,f,l,g,h,n,e,i,m,p,j,q,o,a]),
(834,[a,b,r,c,g,f,l,d,p,n,e,i,m,h,j,q,o,a]),
(716,[a,b,h,c,d,f,l,g,p,n,e,i,m,r,j,q,o,a]),
(777,[a,b,r,c,d,n,l,g,p,f,e,i,m,h,j,q,o,a])]

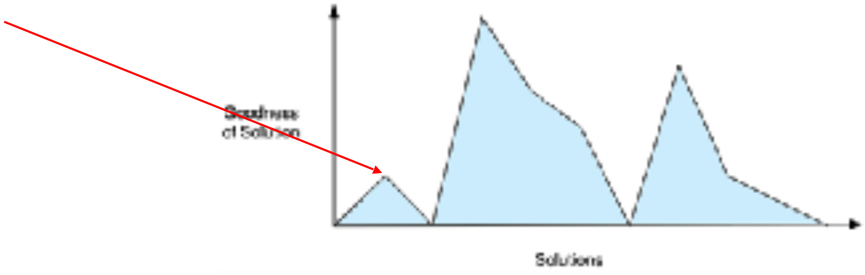


Hill climbing

- If we have a better solution we move.

(716,[a,b,h,c,d,f,l,g,p,n,e,i,m,r,j,q,o,a])

- We define a total number of iterations to stop or we stop when <u>no neighbor is a better solution</u>!
- But not finding a better solution in the neighborhood doesn't mean it does not exist.



Simulated Annealing

- Tries to overcome this problem by (sometimes) not accepting to move to best neighbors.
- The basic idea:
  1. Generate an initial random solution.
  2. Calculate its cost c_{old}.
  3. Generate a random neighbor
  4. Calculate the new solution's cost c_{new}
  5. Compare them:
     1. If $c_{new} < c_{old}$: move to the new solution
     2. If $c_{new} > c_{old}$: ***maybe*** move to the new solution
  6. Repeat steps 3-5 above until an acceptable solution is found or you reach some maximum number of iterations.

Simulated Annealing

1. Generate an initial random solution.
2. Calculate its cost c_{old}.
3. Generate a random neighbor
4. Calculate the new solution's cost c_{new}
5. Compare them:
    1. If $c_{new} < c_{old}$: move to the new solution
    2. If $c_{new} > c_{old}$: ***maybe*** move to the new solution
6. Repeat steps 3-5 above until an acceptable solution is found or you reach some maximum number of iterations.

Similar to hill climbing

Different from hill climbing and requires further details

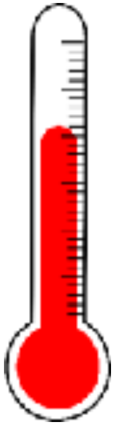
Simulated Annealing

If $c_{new} > c_{old}$: ***maybe*** move to the new solution

- Hill climbing can get caught at local maxima.
- To avoid that problem, Simulated Annealing sometimes chooses to keep the worse solution.
- To decide, the algorithm calculates an **acceptance probability** and then compares it to a **random number.**

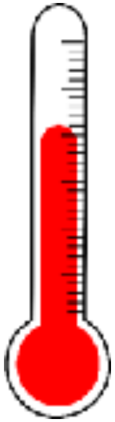
Simulated Annealing

- The acceptance probability function takes c_{old}, c_{new} and a temperature *T*.
- Temperature?
  - Yes, Simulated Annealing is based on metalworking.
  - Temperature is usually started at 1.0
  - It decreases at the end of each iteration by multiplying by a constant α (usually a value between 0,80 and 0,99)
  - Experience shows that higher is better!



Simulated Annealing

- Temperature?
  - We also need to decide how many neighbour generations and comparisons we make <u>at each temperature</u>.
  - One can use a fixed value (the higher the better – between 100 and 1000)
  - An alternative is to dynamically change the number of iterations as the algorithm progresses.
    - At lower temperatures it is important that a large number of iterations are done so that the local optimum can be fully explored.
    - At higher temperatures, the number of iterations can be less.



Simulated Annealing

- Going back…
- The acceptance probability function takes c_{old}, c_{new} and a temperature *T*.
- The acceptance probability generates a number between 0 and 1, which is a sort of recommendation on whether or not to jump to the new solution. For example:
  - 1.0: switch (the new solution is better)
  - 0.0: do not switch (the new solution is infinitely worse)
- Once the acceptance probability is calculated, it's compared to a randomly-generated number between 0 and 1.
- If the acceptance probability is larger than the random number, switch to the new solution.

Simulated Annealing

- The **acceptance probability function** takes c_{old}, c_{new} and a temperature *T*.
- And how do we compute it?
- Usually with the following formula:

$$Ap = e^{\frac{c_{old} - c_{new}}{T}}$$

(Stuart Russell and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall Press, Upper Saddle River, NJ, USA.)

Simulated Annealing

$$Ap = e^{\frac{c_{old} - c_{new}}{T}}$$

- The probability decreases exponentially with the "badness" of the move. $(c_{old} - c_{new})$.
- The probability also decreases as the "temperature" T goes down:
  - "bad" moves are more likely to be allowed at the start when T is high, and they become more unlikely as T decreases.
- If the algorithm lowers T slowly enough, it will find a global optimum with probability approaching 1.

Simulated Annealing

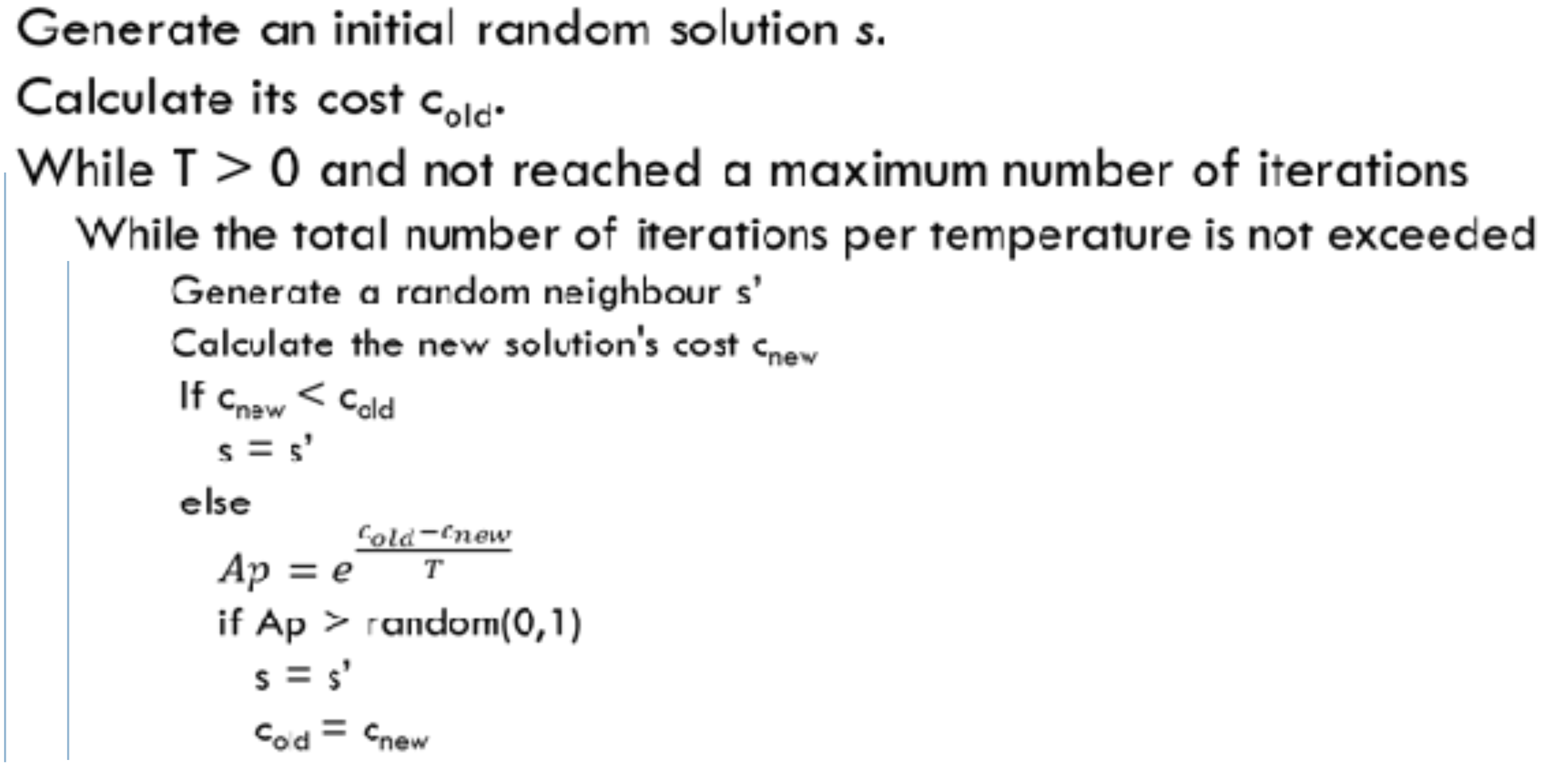
- $$Ap = e^{\frac{c_{old} - c_{new}}{T}}$$

- Examples:
  - T = 1, c_{old} = 99 and c_{new} = 100, Ap ≈ 0,38 (38%)
  - T = 0.9, c_{old} = 100 and c_{new} = 98, Ap ≈ 9 (more than 1, we consider 100%)
  - T = 0.9, c_{old} = 98 and c_{new} = 100, Ap ≈ 0,11 (11%)

(Recall $e = 2{,}71828$)

Simulated Annealing

- The complete idea:



```
Generate an initial random solution s.
Calculate its cost $c_{old}$.
While T > 0 and not reached a maximum number of iterations
    While the total number of iterations per temperature is not exceeded
        Generate a random neighbour s'
        Calculate the new solution's cost $c_{new}$
        If $c_{new} < c_{old}$
            s = s'
        else
            $Ap = e^{\frac{c_{old} - c_{new}}{T}}$
            if Ap > random(0,1)
                s = s'
                $c_{old} = c_{new}$
```