ARTIFICIAL INTELLIGENCE

Lec 08

WHAT WE STUDIED BEFORE!

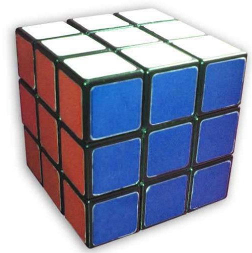
Explore whole state space graph systematically.

This systematicity is achieved by

- keeping one or more paths in memory and
- by recording which alternatives have been explored at each point along the path.

When a goal is found, the *path* to that goal also constitutes a *solution* to the problem

LOCAL SEARCH



In many problems, however, the path to the goal is irrelevant. e.g., Rubik's cube.

For such cases, we do a local search e.g., instead of memorizing nodes, we only expand the current node, find the best child, expand it, then find its best child, then expand it and so on.

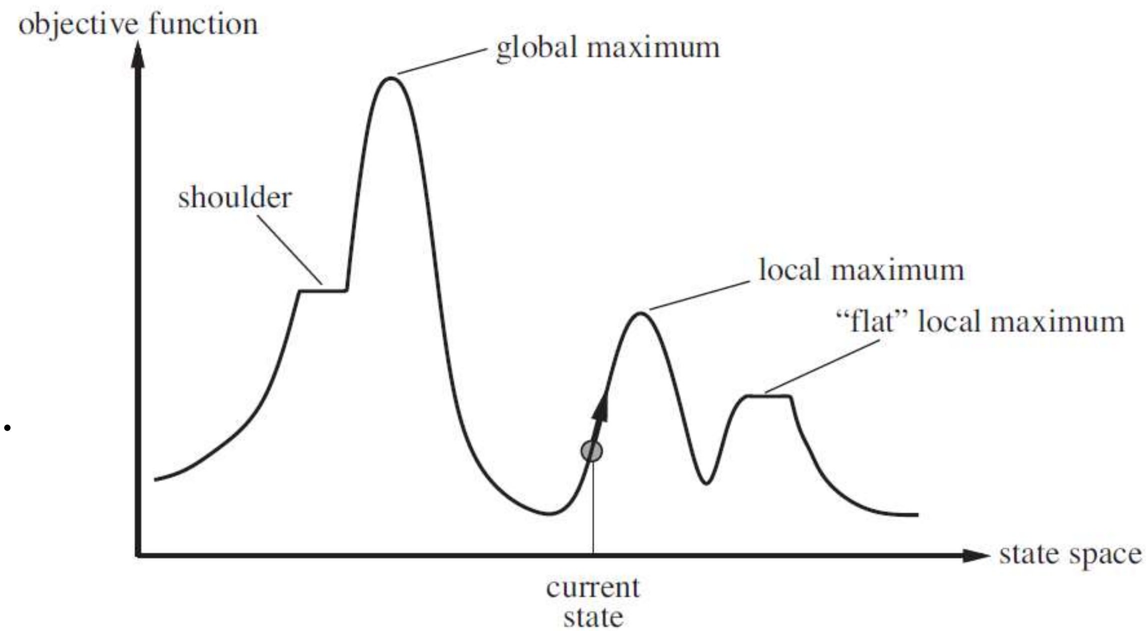
Advantages: low memory and working in a large state space graph (large value of m and d)

STATE SPACE LANDSCAPE

A landscape has

“location” (defined by the state) and

“elevation” (defined by the value of the heuristic cost function or objective function).



OPTIMIZATION

In optimization problems, aim is to find the best state according to an **objective function**

Example: n-queens problem: Put n queens on a n X n board with no two queens on the same row, column or diagonal.

Initial state (not given, randomly generated)

Solution 1

Solution 2

HILL CLIMBING / GREEDY LOCAL SEARCH

Objective Function: h(n)

$\sum_{i=1}^{4}$ number of queens in same row, column or diagonal as that of i^{th} queen

2 + 3 + 3 + 2 = 10

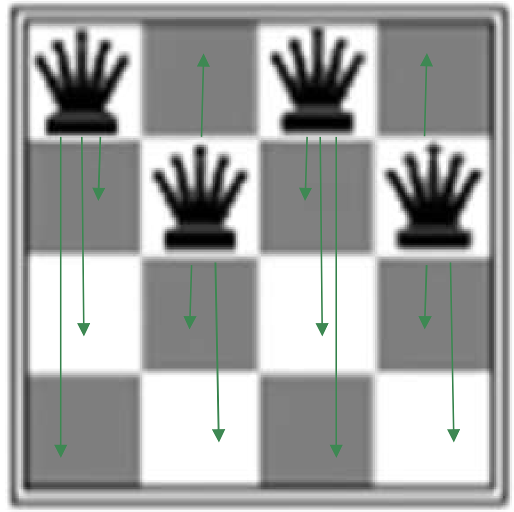
Find the state n for which this value is minimum
(in this case minimum is 0)

Initial state (n)

POSSIBLE SUCCESSORS / NEIGHBORS

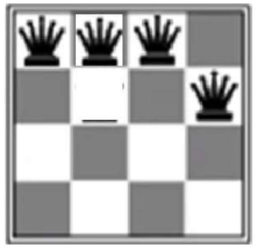
You can move a queen anywhere within that column

Therefor each state has 4 * 3 = 12 successor states.

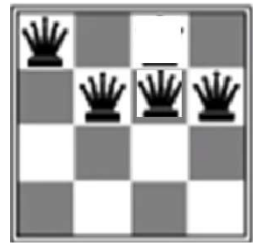


Among these 12 successor states, find the one with lowest value of h and set that successor as the current state; continue until h=0 !

h = 2+3+2+3=10



h = 2+2+3+1=8



h = 1+3+2+2=8

h = 3+2+3+2=10

h = 2+3+3+2=10

h = 1+0+2+1=4

h = 2+3+3+2=10

h = 2+2+2+0=6

h = 0+2+2+2=6

h = 1+1+2+1=5

h = 1+2+0+1=4

h = 3+3+2+2=10

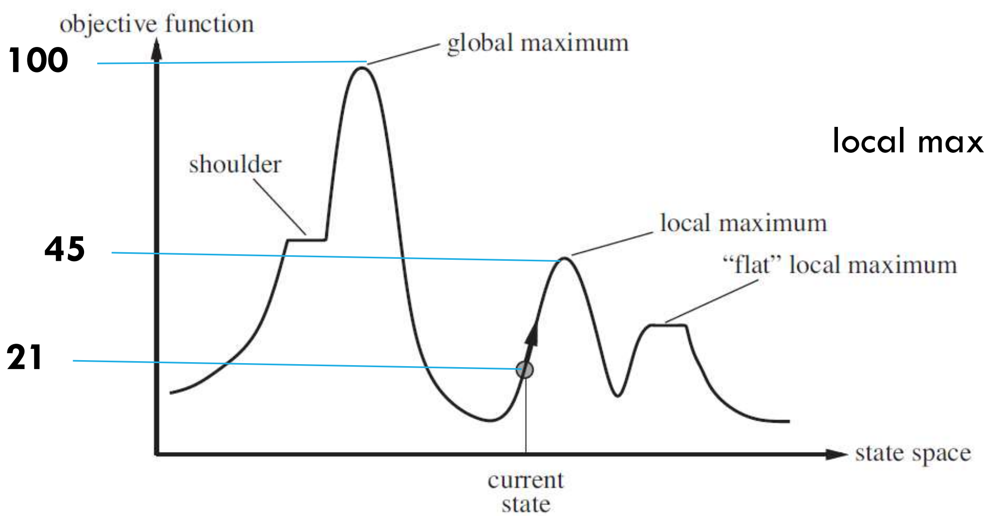
HILL CLIMBING - PSEUDOCODE

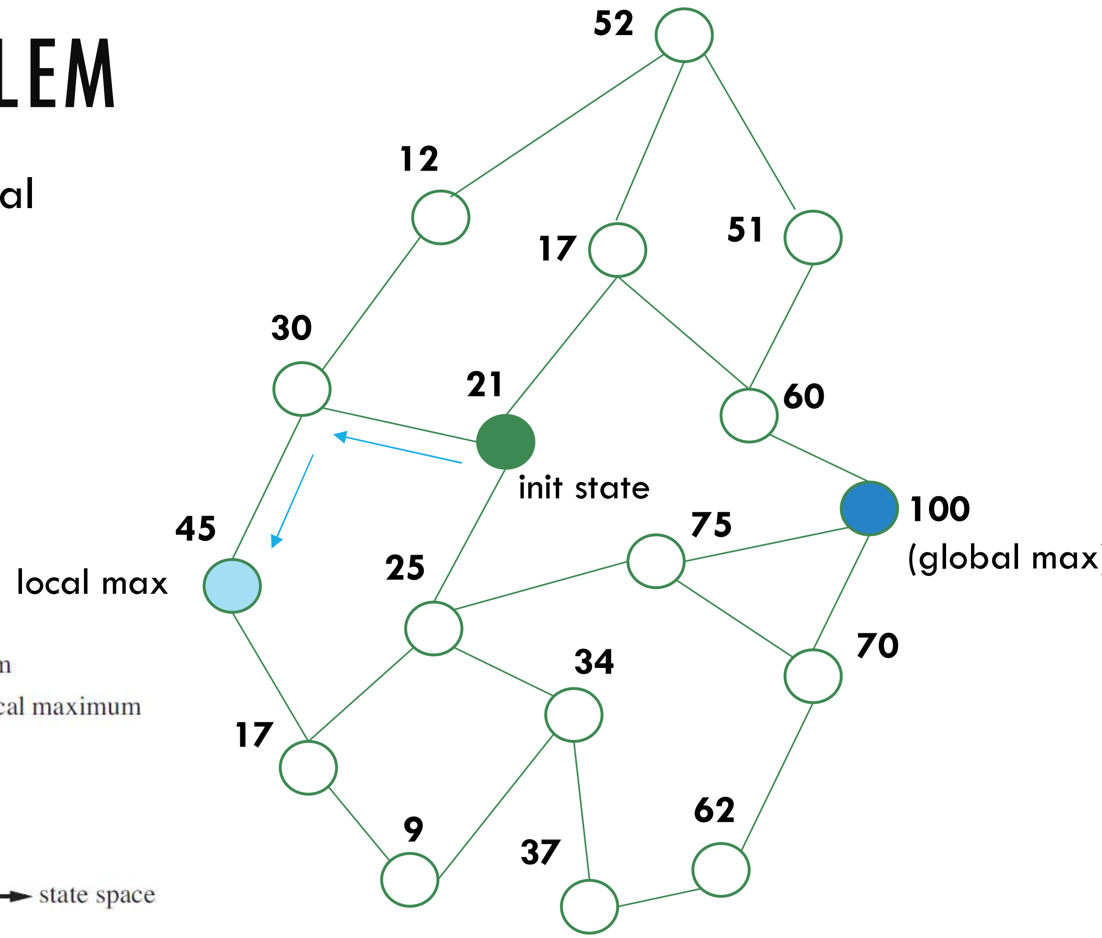
```
function HILL-CLIMBING(problem) returns a state that is a local maximum

  current ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
      neighbor ← a highest-valued successor of current
      if neighbor.VALUE ≤ current.VALUE then return current.STATE
      current ← neighbor
```

HILL CLIMBING - PROBLEM

Normal hill climbing can get stuck at local maximum/minimum depending upon start/initial/current state.





HILL CLIMBING - PROBLEM

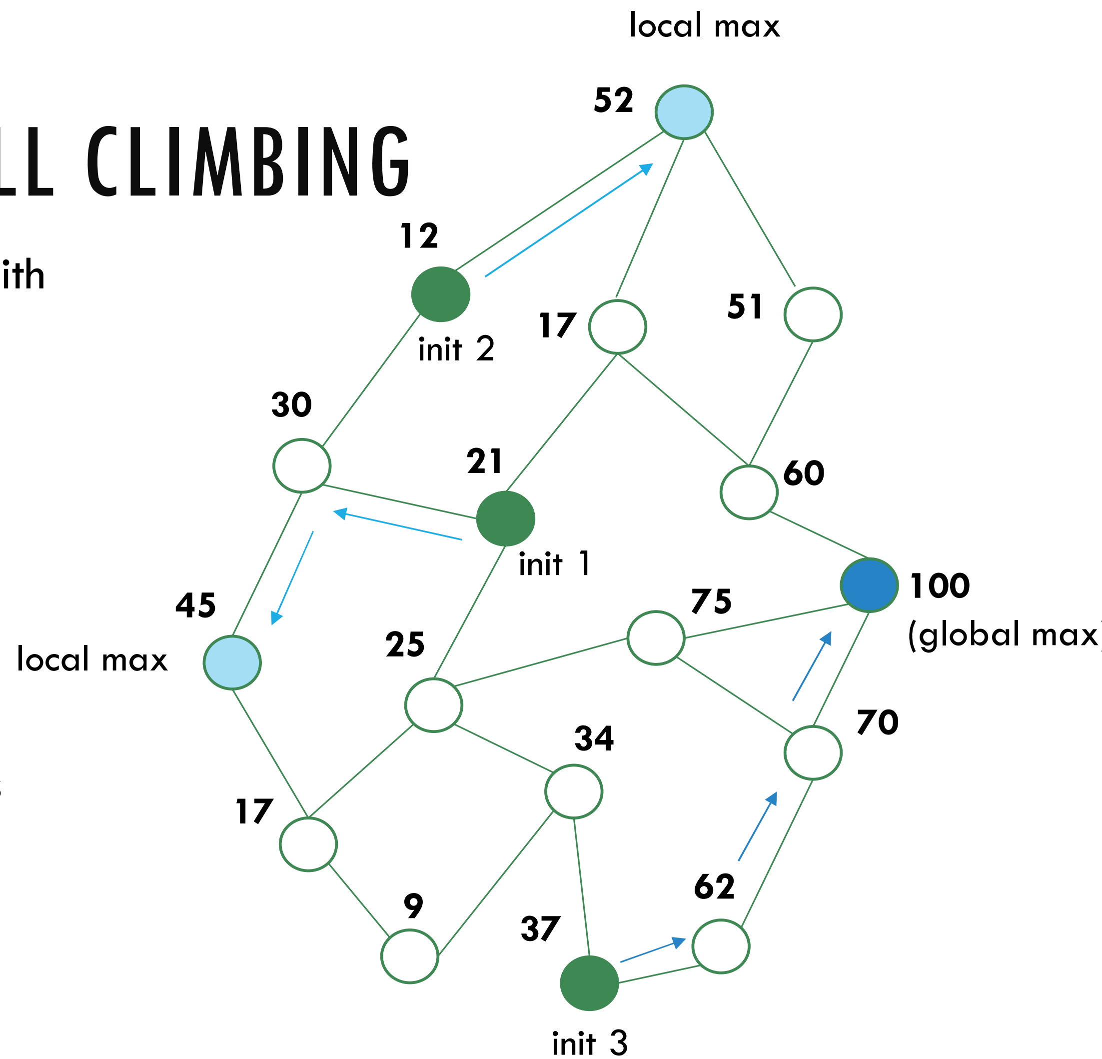
Solution(s)

- Random Restart Hill Climbing
- Local Beam Search
- Genetic Algorithms
- Simulated Annealing

RANDOM RESTART HILL CLIMBING

In case of failure in 1st search, restart with a different initial state. Keep repeating the procedure until h==100 state is reached.

If each hill-climbing search has a probability p of success, then the expected number of restarts required is 1/p.



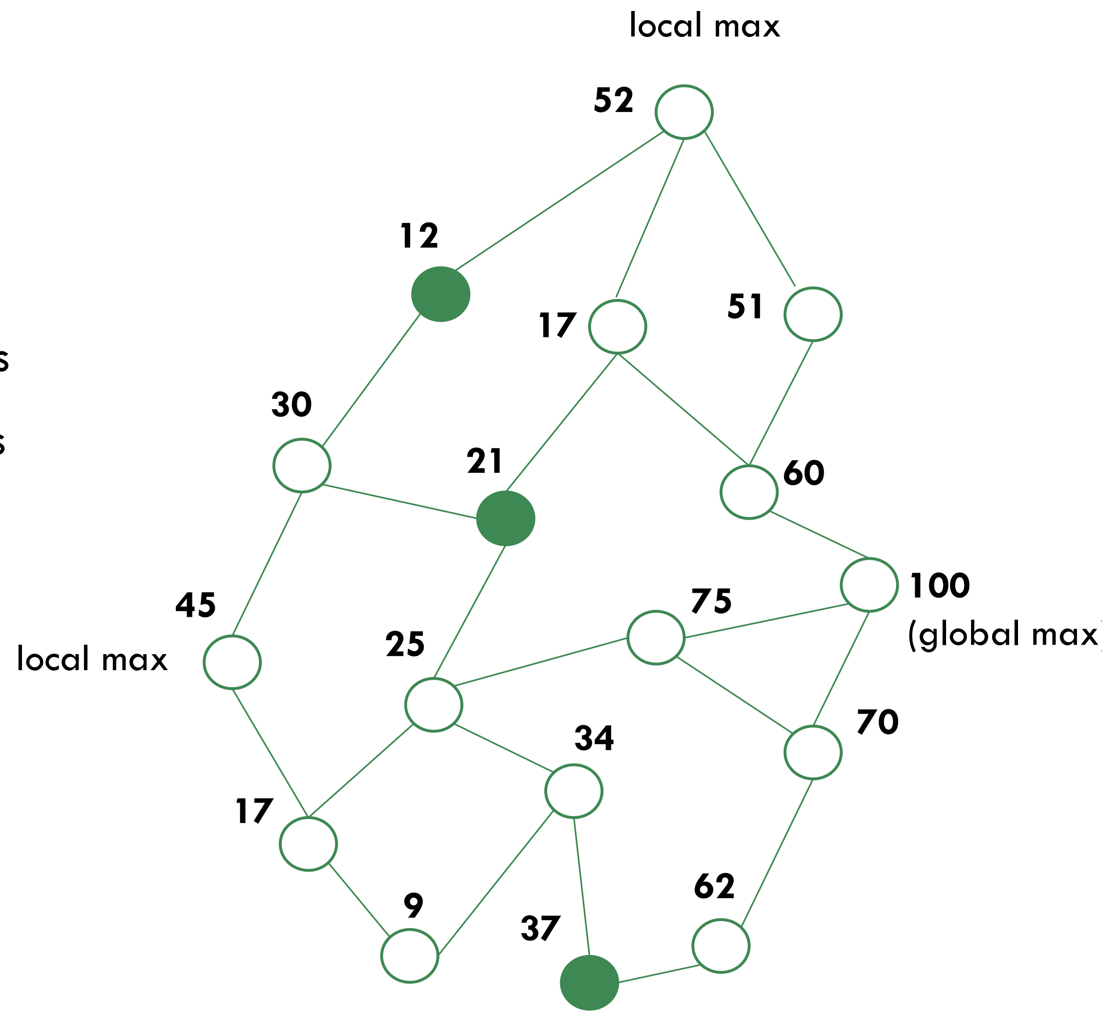
LOCAL BEAM SEARCH

Start with k randomly generated states

Generate all successors of those k states

If no goal found, select best k successors and repeat.

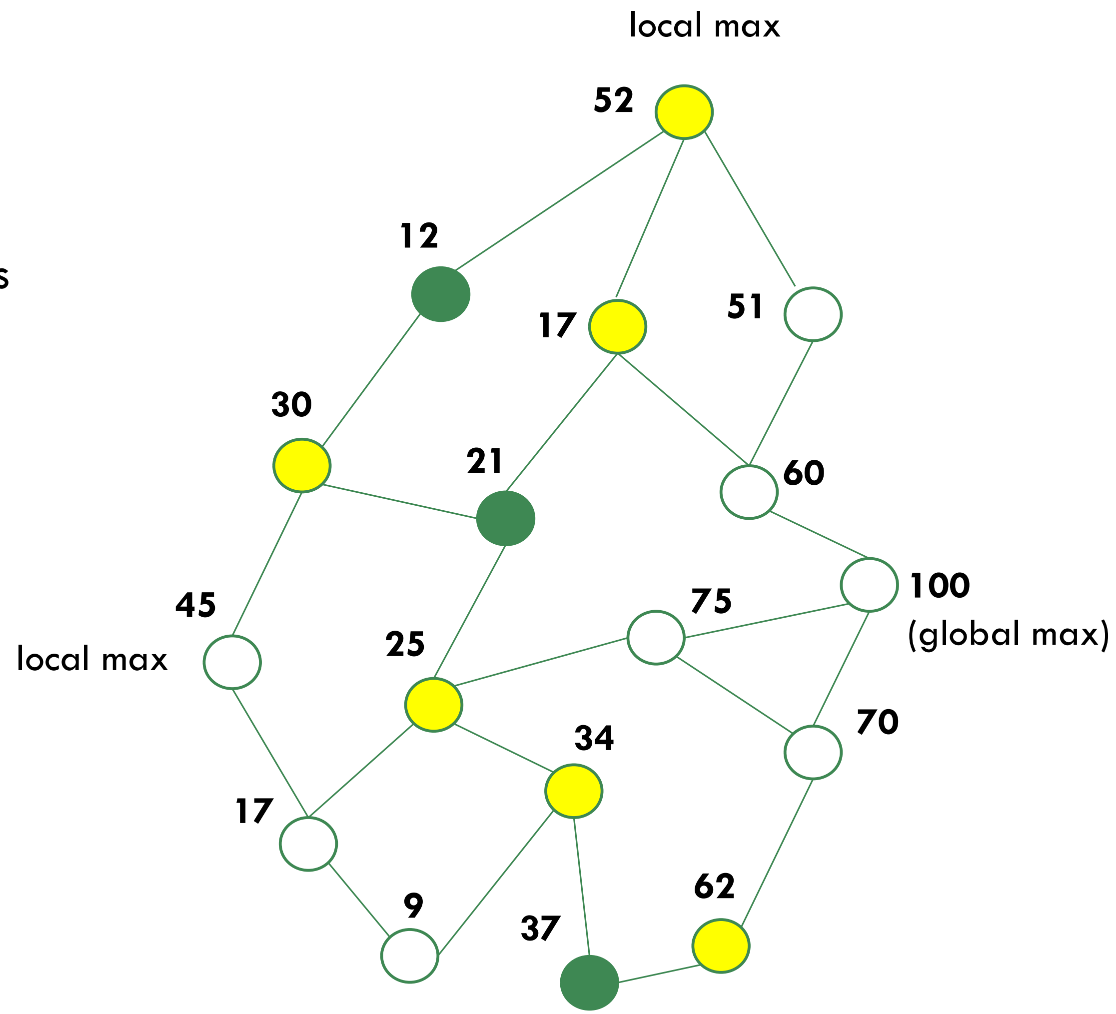
For the tree on the right, k =3



LOCAL BEAM SEARCH

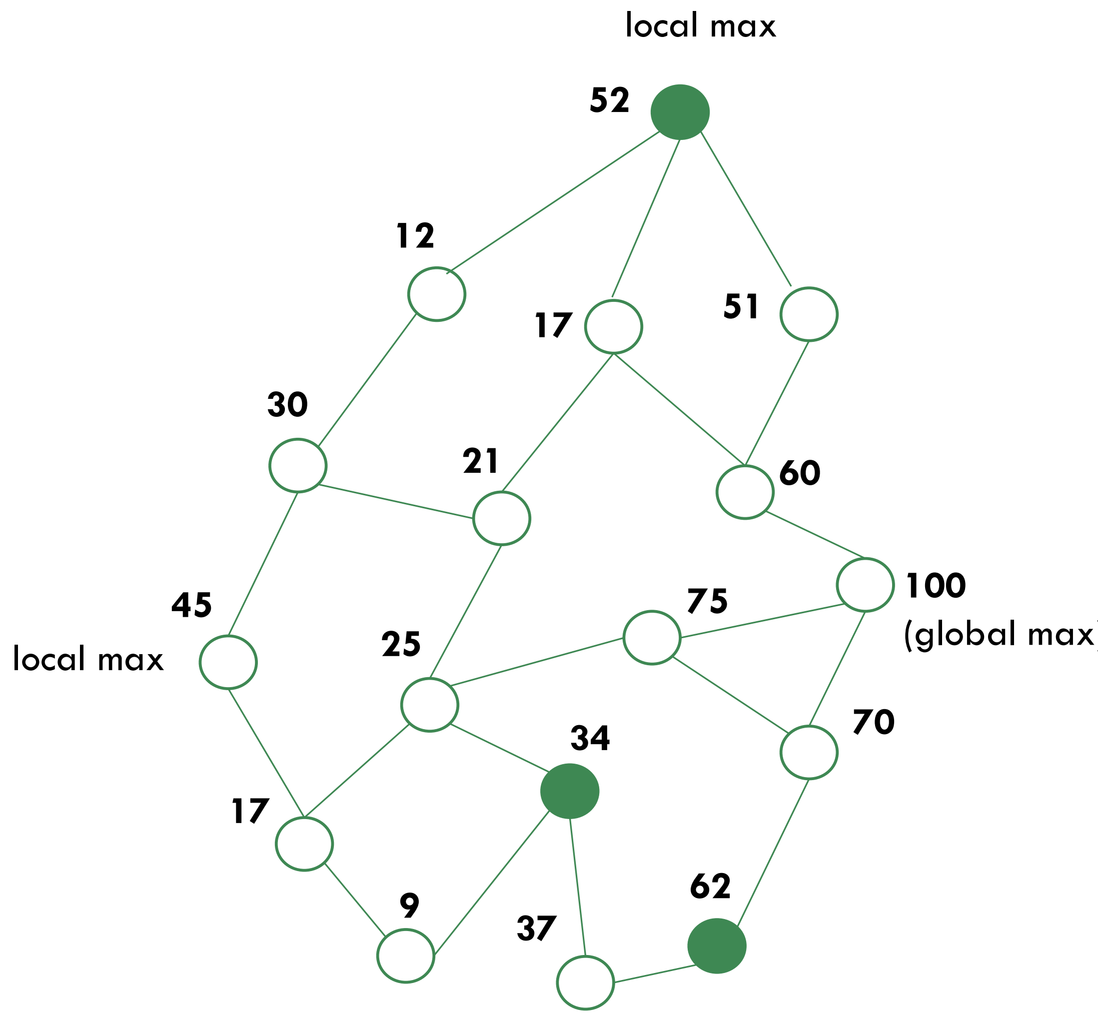
Generate all successors of those k states

Among these successors (yellow) choose, top 3 since k=3.



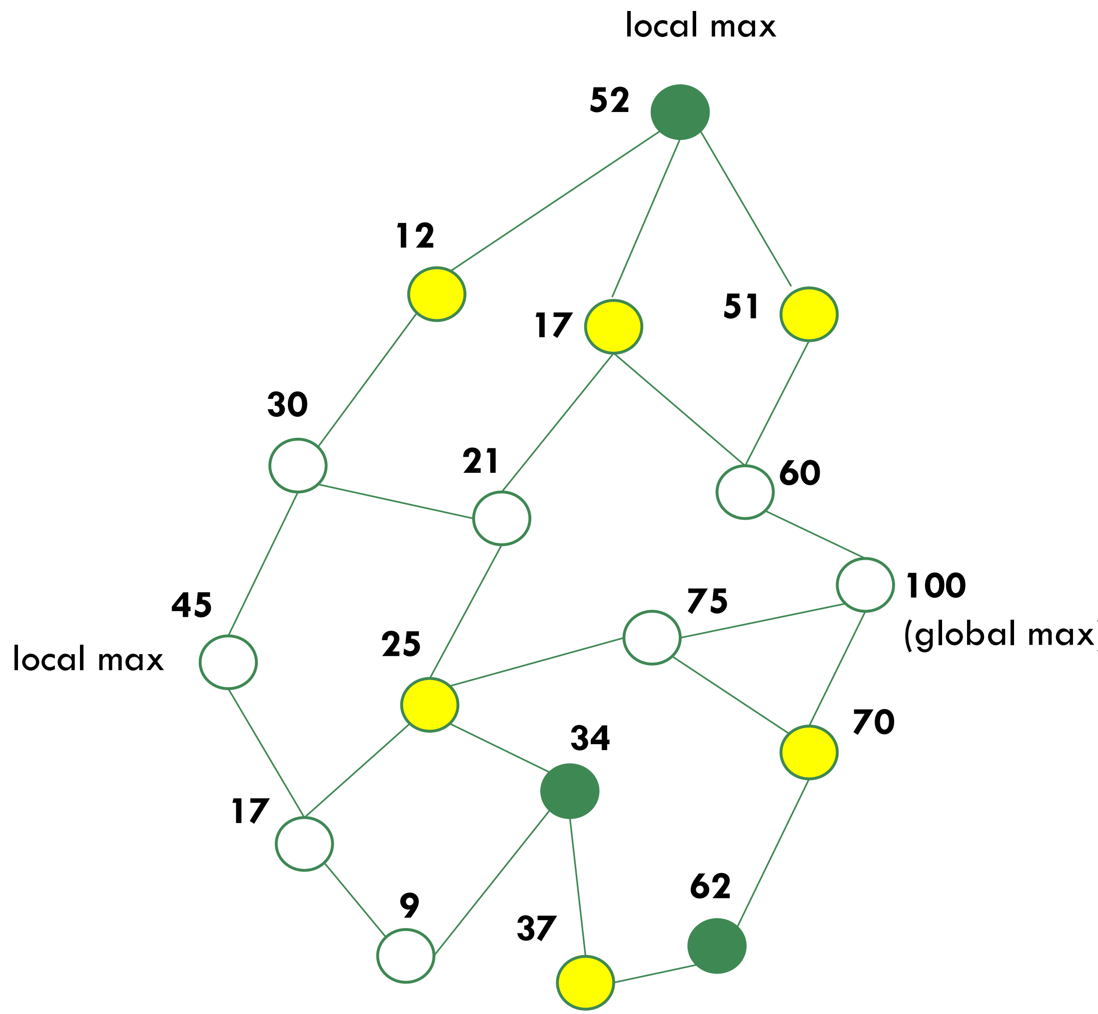
LOCAL BEAM SEARCH

For these 3, repeat the same procedure until state with h=100 is reached.



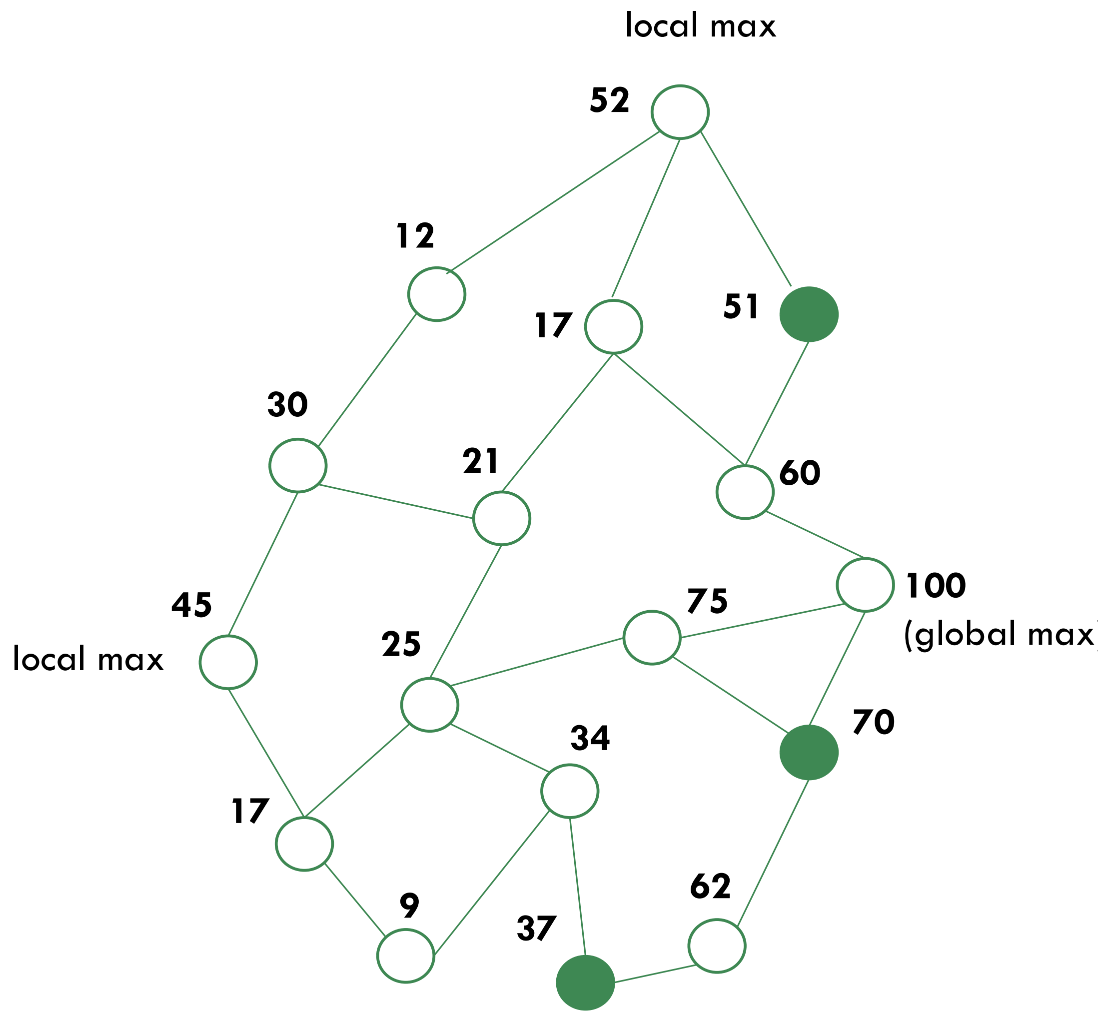
LOCAL BEAM SEARCH

New children are generated



LOCAL BEAM SEARCH

Best 3 are selected.



LOCAL BEAM SEARCH

One of the successors is a goal (global max) !