Local Search Algorithms

K. LEKSHMI
SSNCE

Objectives

- To explain Hill Climbing search strategies

Outcomes

- Solve problem using Hill Climbing Search strategies

- Best-first search
- Greedy best-first search
- A^* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Local beam search

Local search algorithms

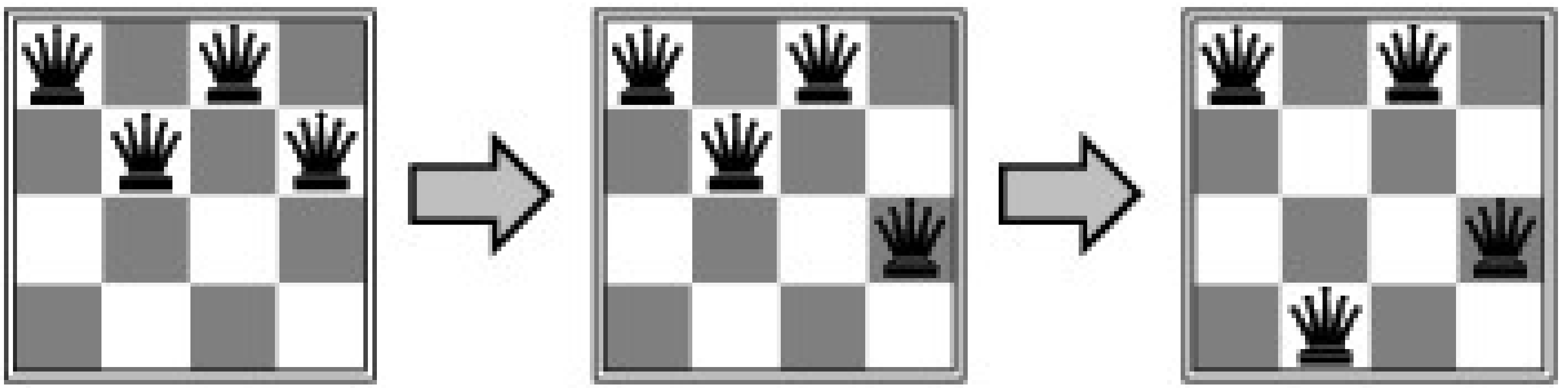
- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens

- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it

Example: *n*-queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal
-



Generate-And-Test

Generate-And-Test

- Generate current state as initial state
- Generate possible solution by applying operator
- Compare newly generated solution with goal state
- If solution is found, quit else return to step 2

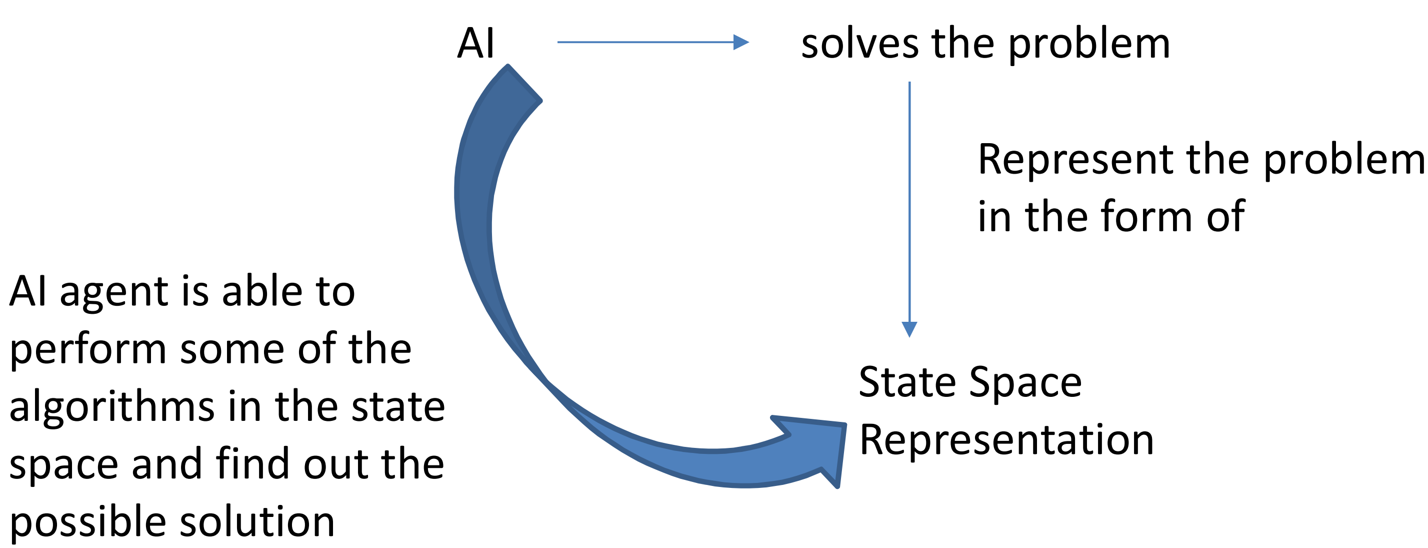
AI $\longrightarrow$ solves the problem

$\downarrow$ Represent the problem in the form of

State Space Represenation

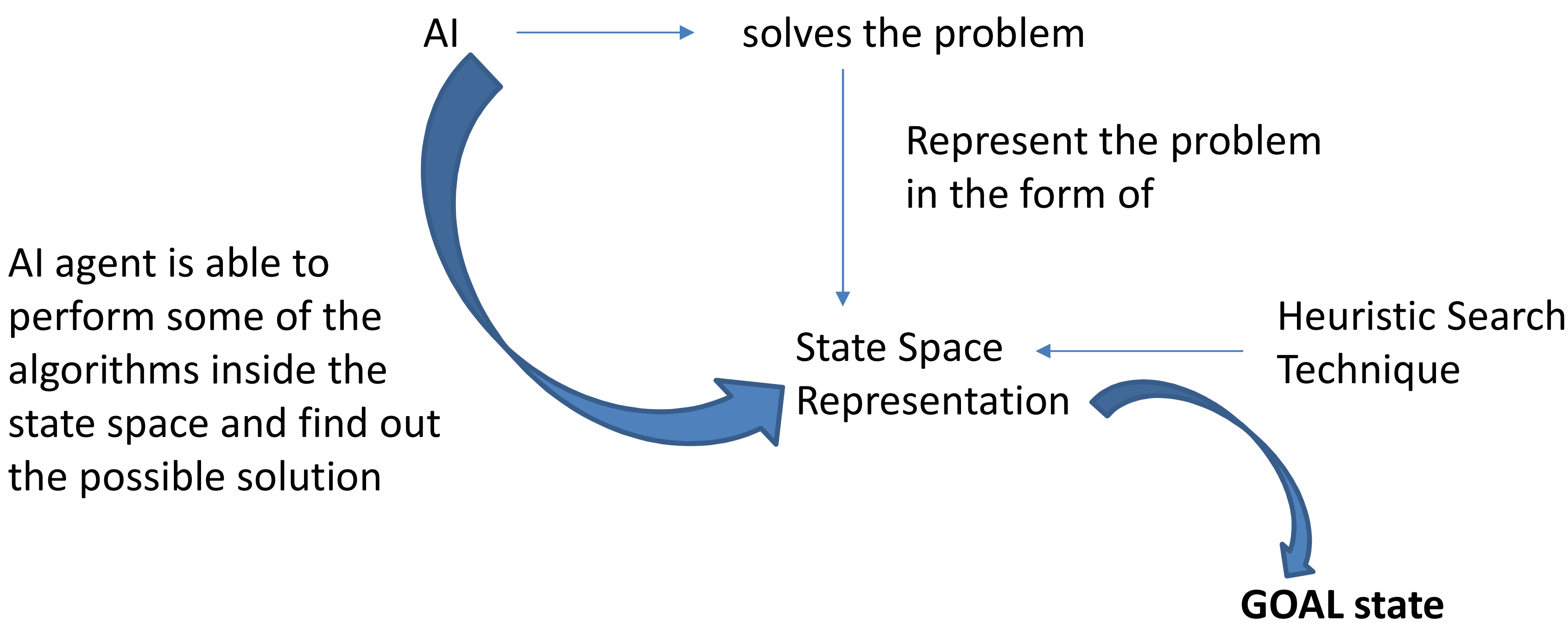
Generate-And-Test

- Generate current state as initial state
- Generate possible solution by applying operator
- Compare newly generated solution with goal state
- If solution is found, quit else return to step 2



Generate-And-Test

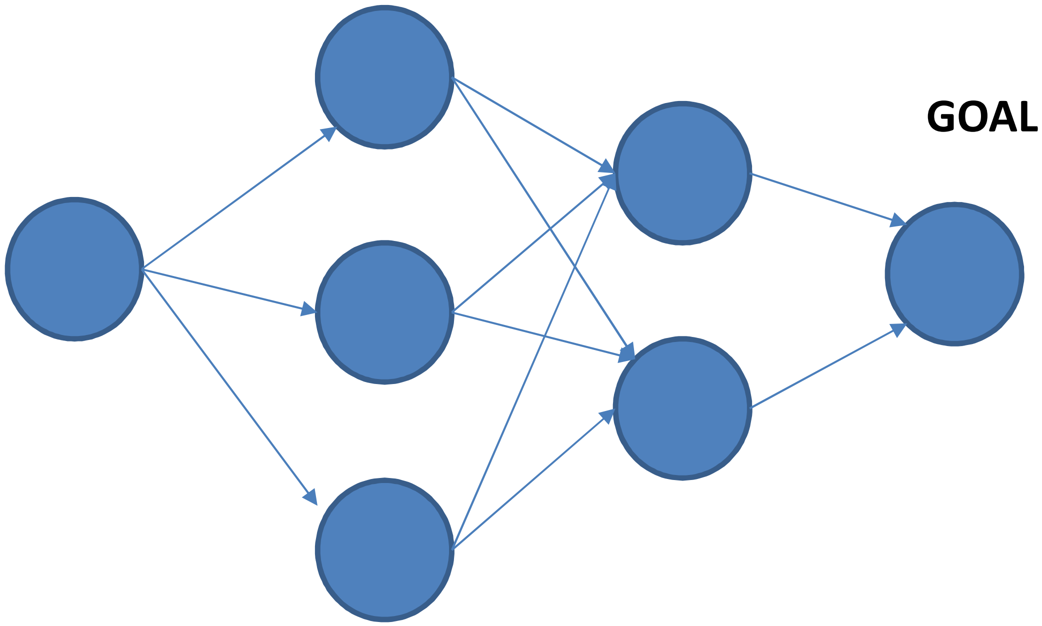
- Generate current state as initial state
- Generate possible solution by applying operator
- Compare newly generated solution with goal state
- If solution is found, quit else return to step 2



- Heuristic is GUI to the state space searching algorithms
- It makes AI agent to reach the solution to the problem in an optimized way

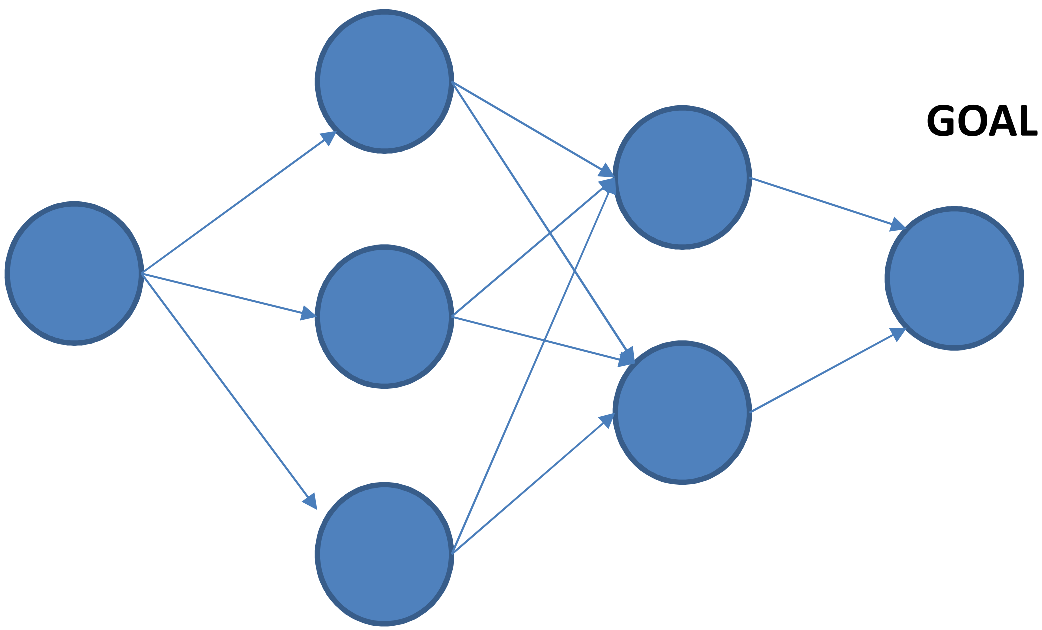
Generate-And-Test

- State Space Representation of the Problem



Generate-And-Test

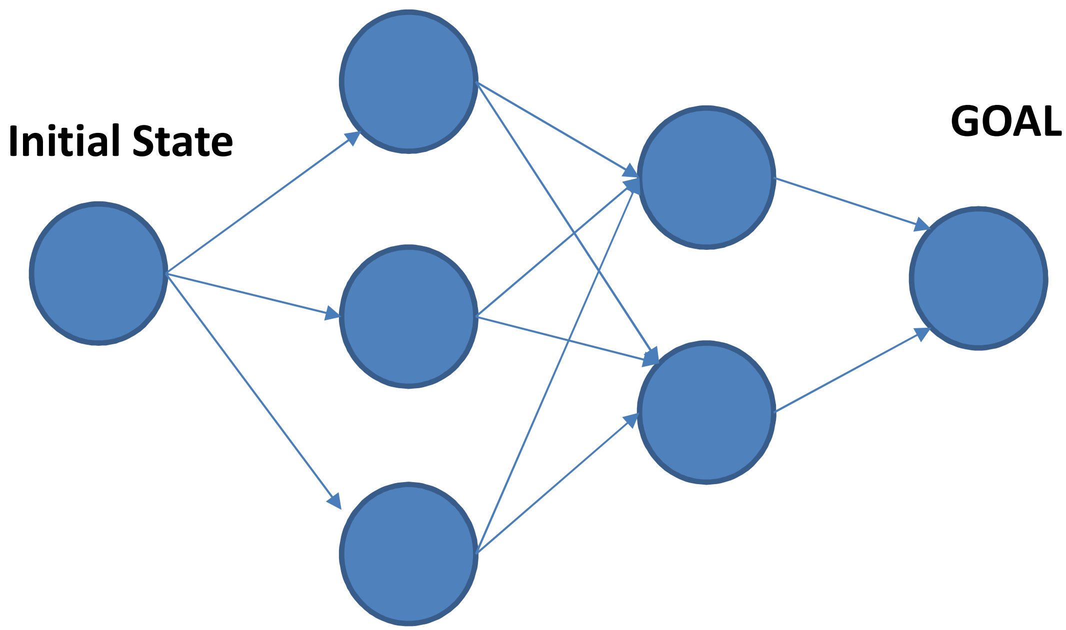
- State Space Representation of the Problem



1. **Generate current state as initial state**
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. If solution is found, quit else return to step 2

Generate-And-Test

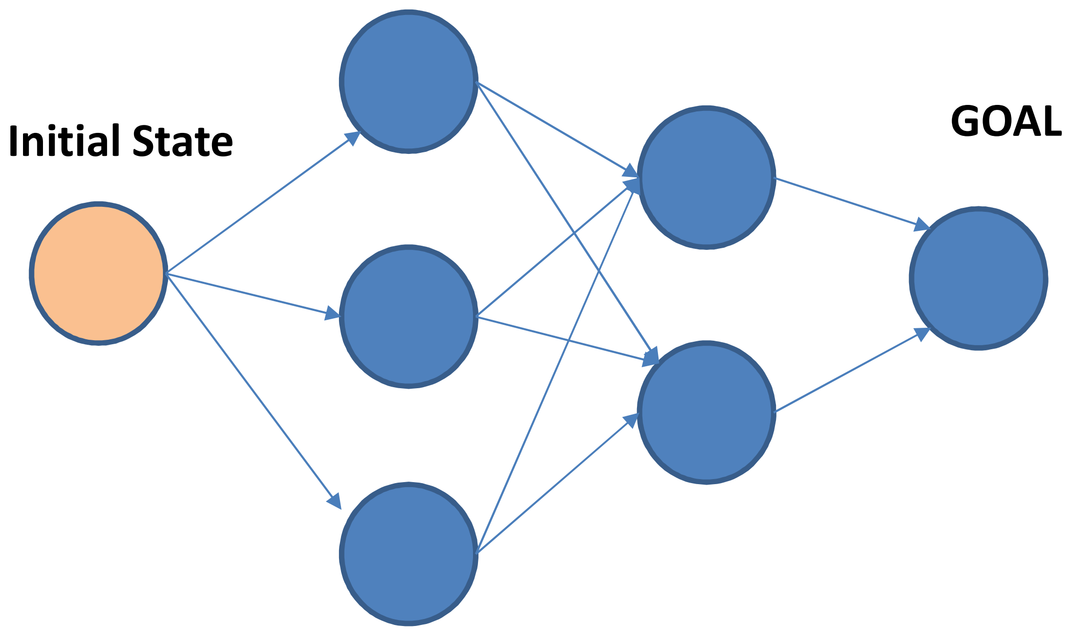
- State Space Representation of the Problem



1. **Generate current state as initial state**
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. If solution is found, quit else return to step 2

Generate-And-Test

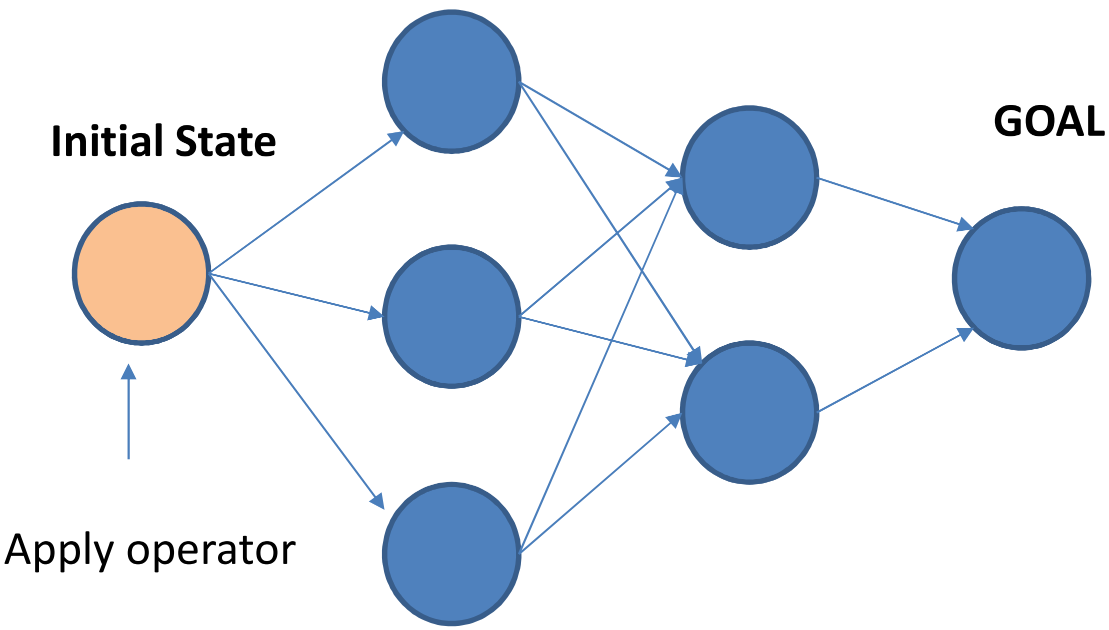
- State Space Representation of the Problem



1. **Generate current state as initial state**
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. If solution is found, quit else return to step 2

Generate-And-Test

- State Space Representation of the Problem

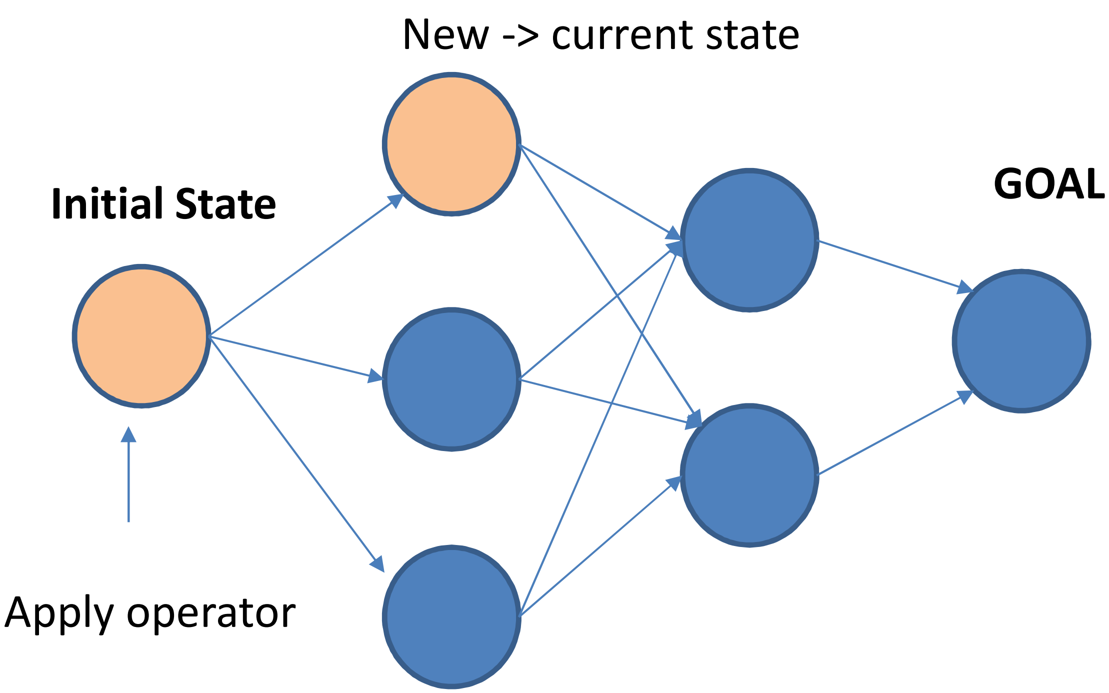


- Just apply some possible operator to reach a possible solution
- Generate any possible solution without even considering
  - Whether it can goal or not (or)
  - Whether it is an optimized choice or not

1. Generate current state as initial state
2. **Generate possible solution by applying operator**
3. Compare newly generated solution with goal state
4. If solution is found, quit else return to step 2

Generate-And-Test

- State Space Representation of the Problem

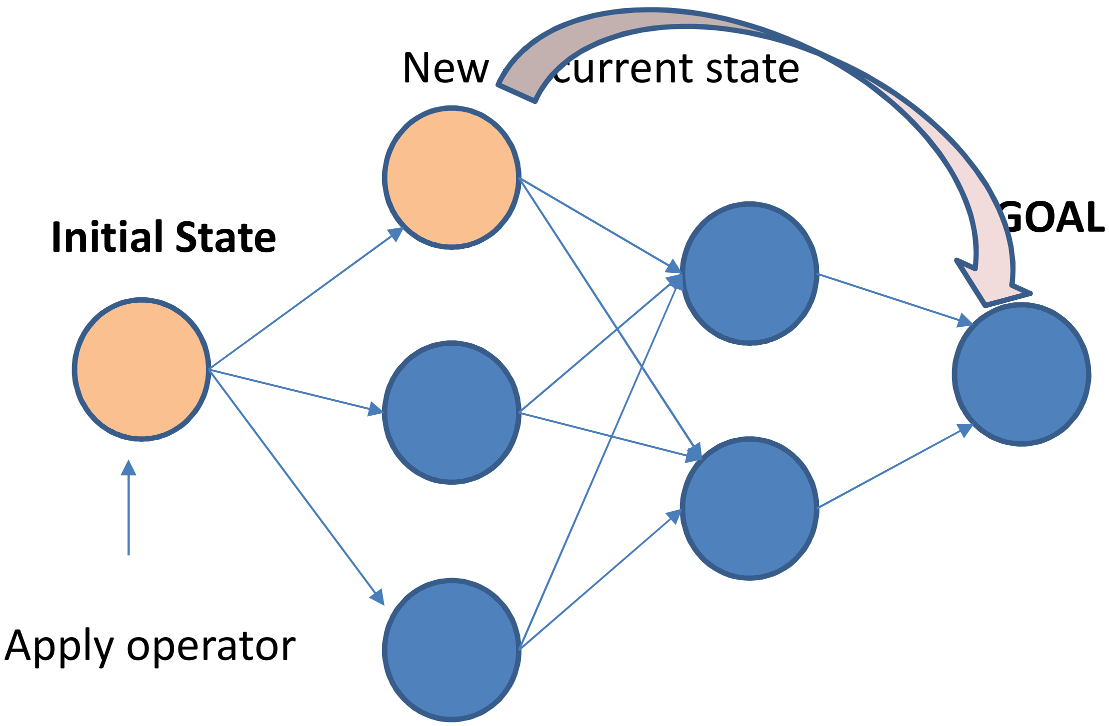


- New represents newly generated state which now becomes current state

1. Generate current state as initial state
2. **Generate possible solution by applying operator**
3. Compare newly generated solution with goal state
4. If solution is found, quit else return to step 2

Generate-And-Test

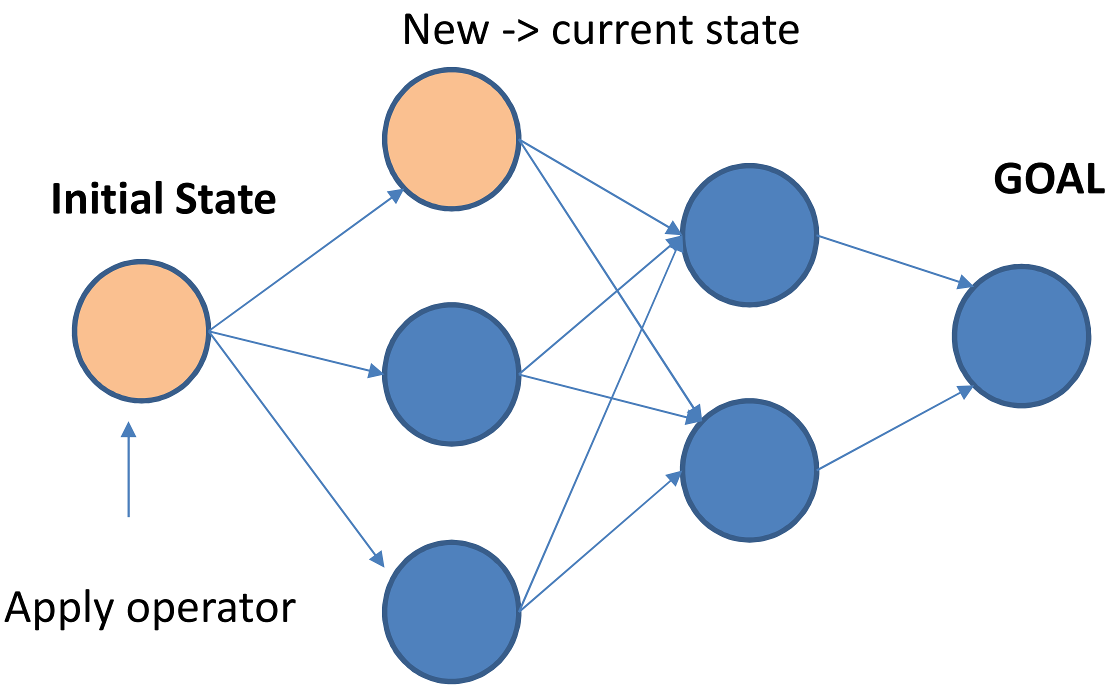
- State Space Representation of the Problem



1. Generate current state as initial state
2. Generate possible solution by applying operator
3. **Compare newly generated solution with goal state**
4. If solution is found, quit else return to step 2

Generate-And-Test

- State Space Representation of the Problem



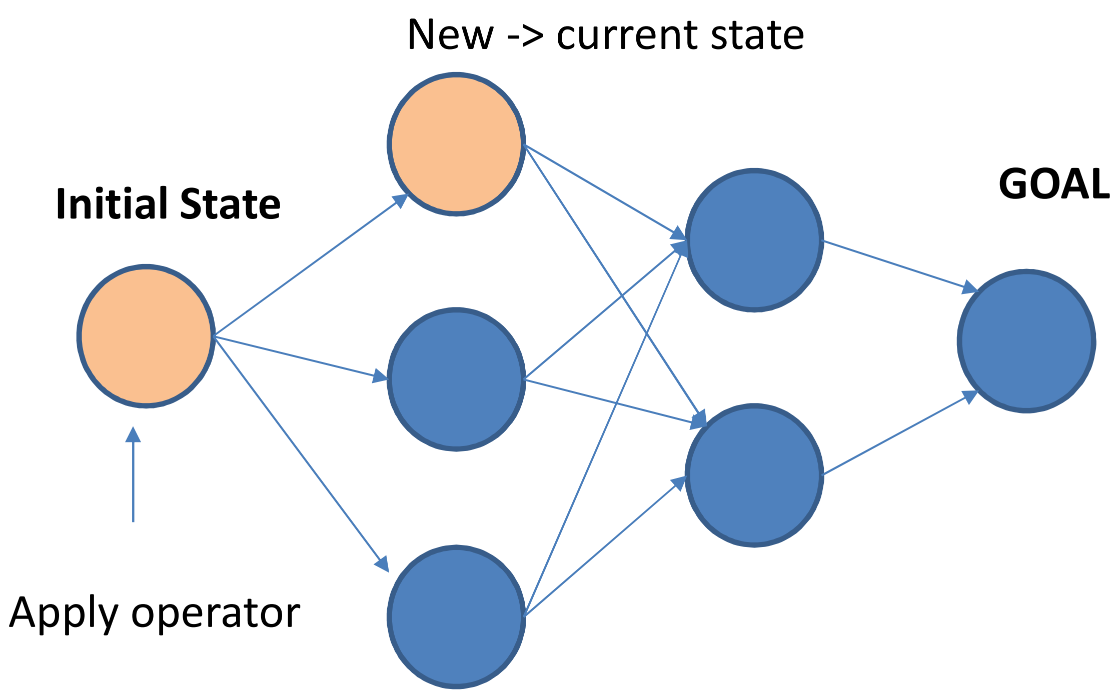
- Its not GOAL state.
- No QUIT.

- *will QUIT only when the generated state is the GOAL state*

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

Generate-And-Test

- State Space Representation of the Problem



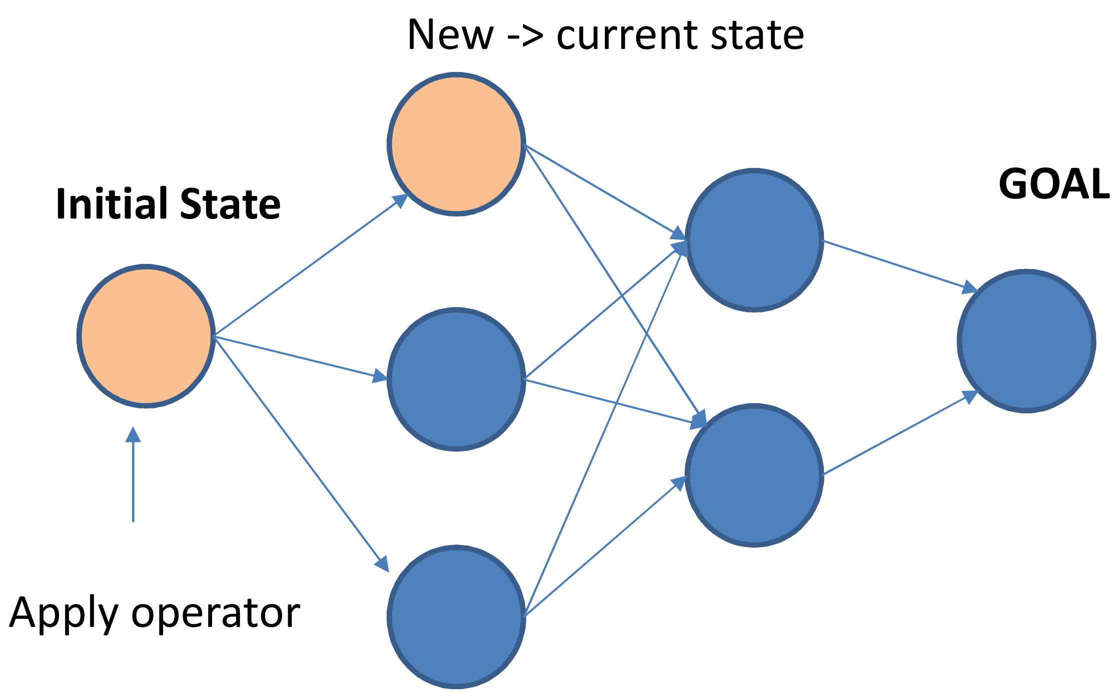
- Its not GOAL state.
- No QUIT.

- *will QUIT only when the generated state is the GOAL state*

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

Generate-And-Test

- State Space Representation of the Problem



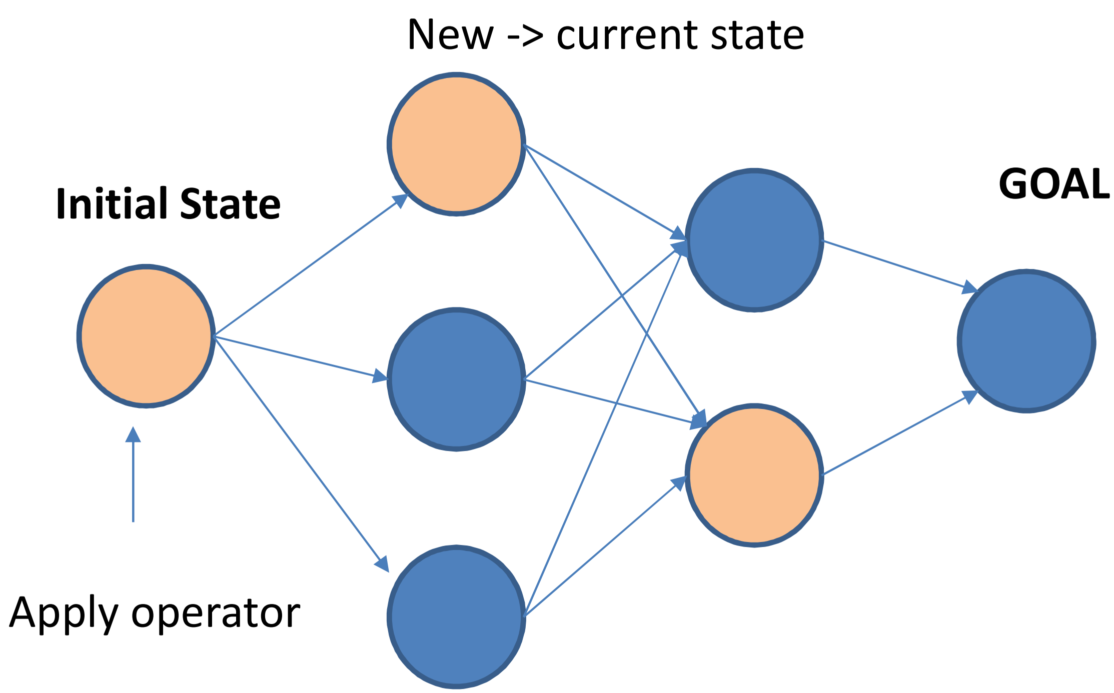
- Its not GOAL state.
- No QUIT.

- *will QUIT only when the generated state is the GOAL state*

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

Generate-And-Test

- State Space Representation of the Problem

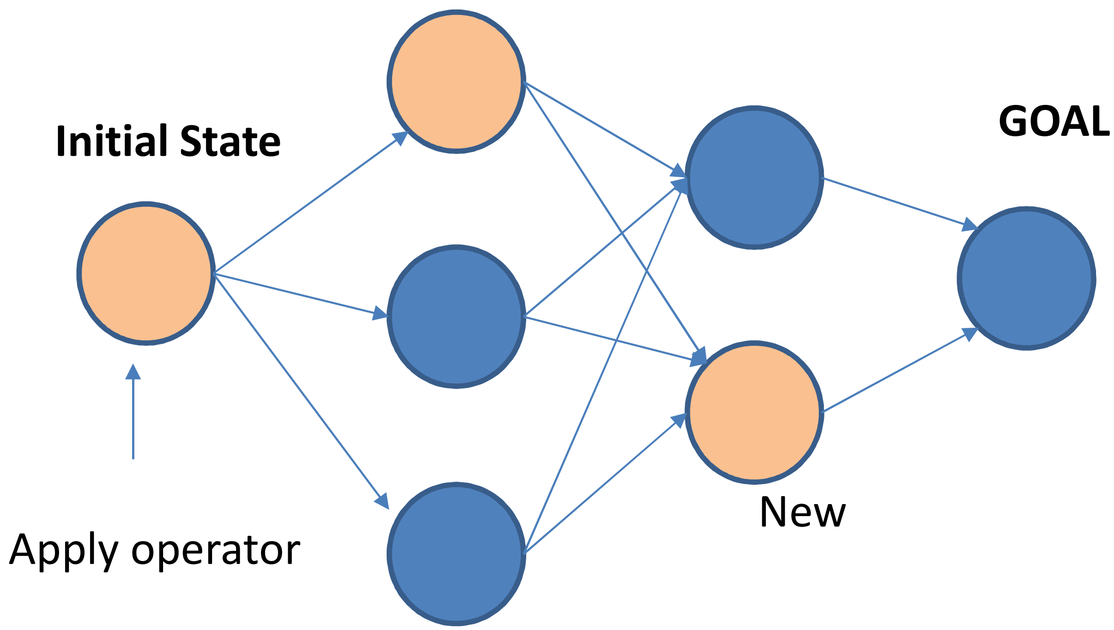


- Step 2: Apply operator from the CURRENT state and get the possible solution

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

Generate-And-Test

- State Space Representation of the Problem

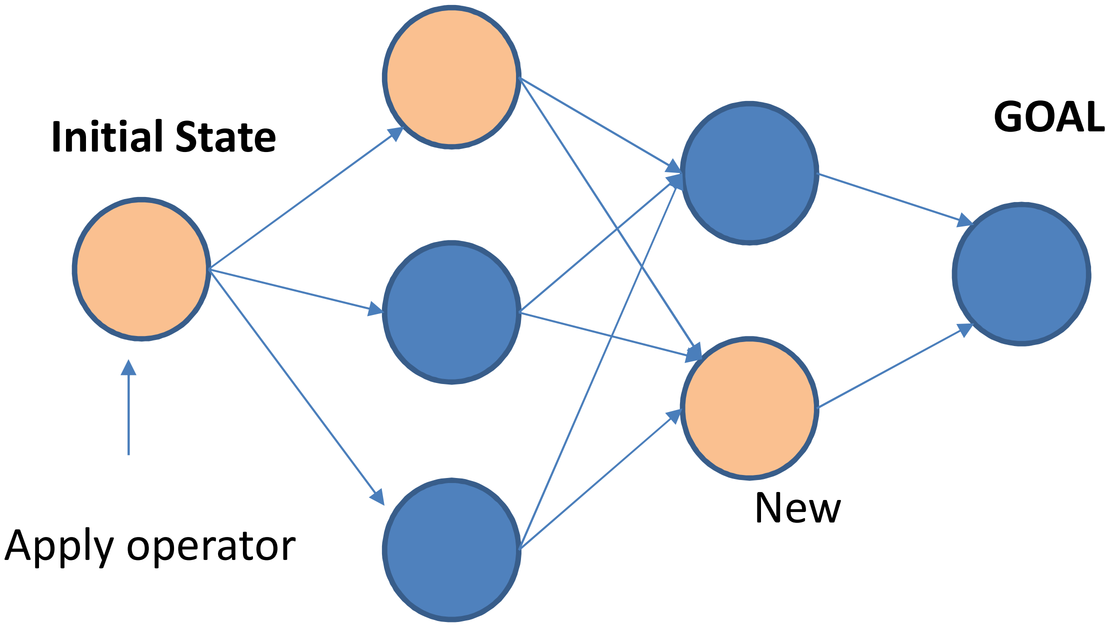


- Step 2: Apply operator from the CURRENT state and get the possible solution
- New node gets generated.
- Again, its not GOAL state

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

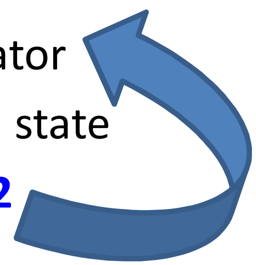
Generate-And-Test

- State Space Representation of the Problem



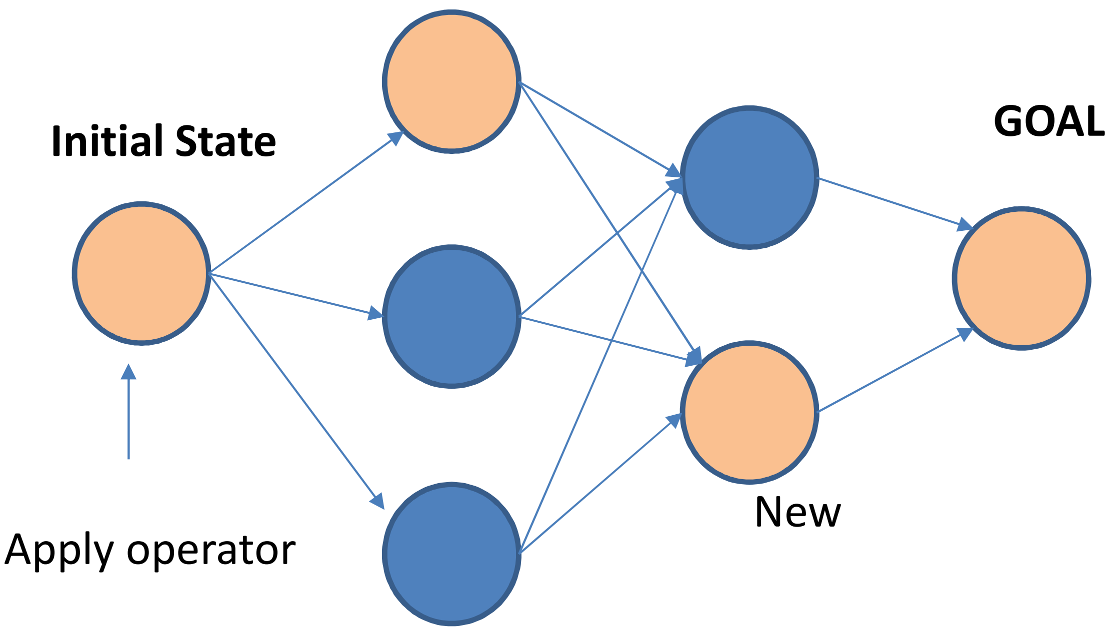
- Step 2: Apply operator from the CURRENT state and get the possible solution
- New node gets generated.
- Again, its not GOAL state
- **Repeat STEP 2**

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**



Generate-And-Test

- State Space Representation of the Problem

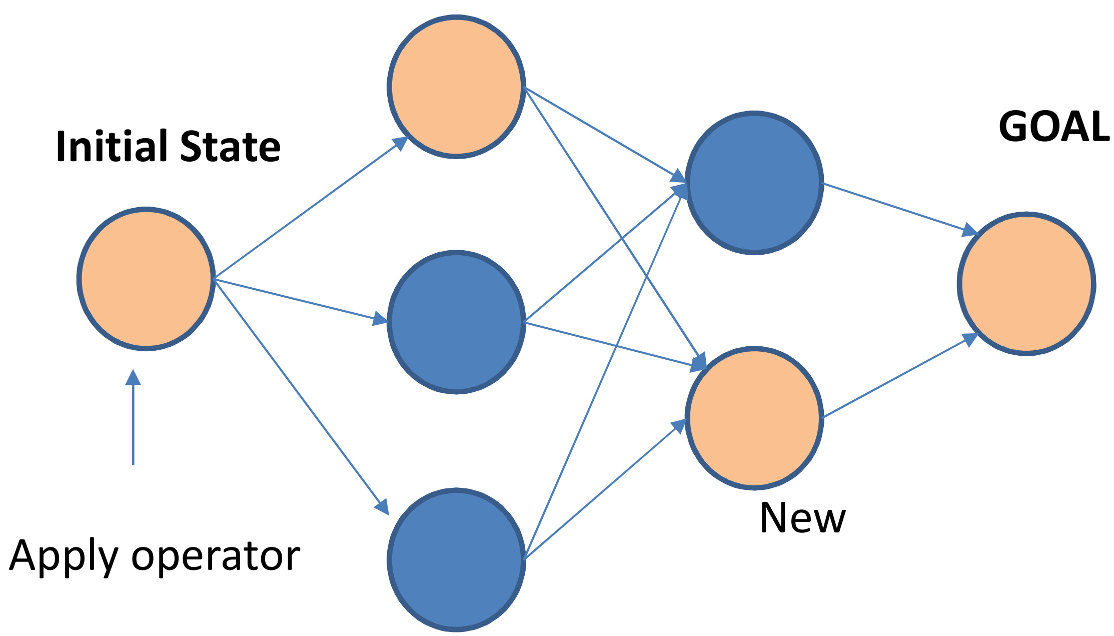


- Step 2: **Apply operator** from the CURRENT state and get the possible solution
- New node gets generated.
- Now, its a **GOAL** state
- Algorithm will **QUIT** now

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

Generate-And-Test

- State Space Representation of the Problem

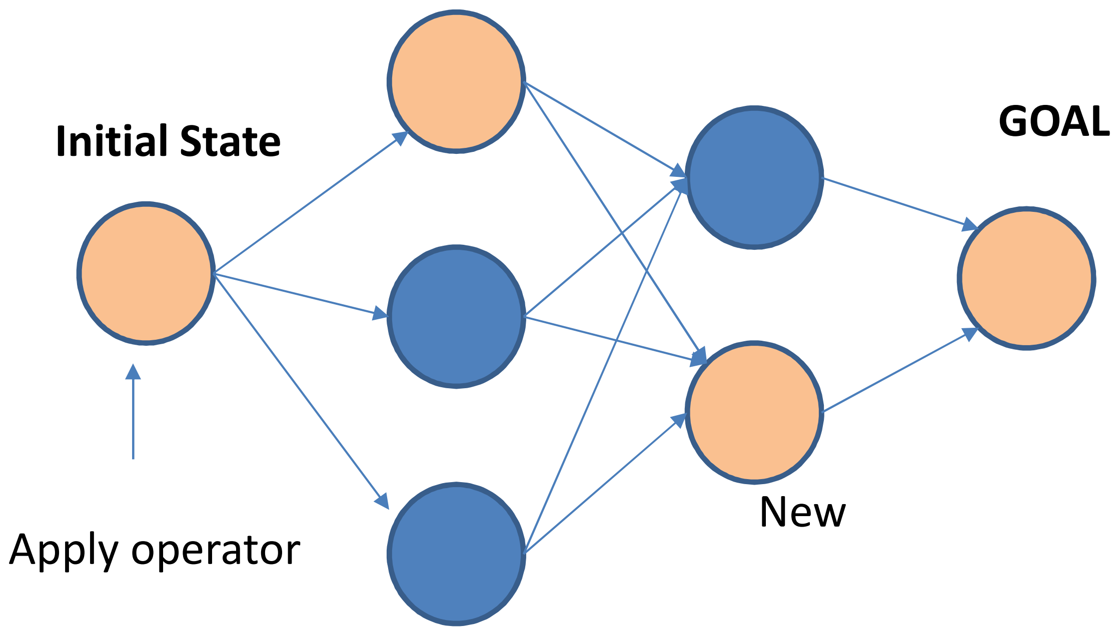


- It is a simple example of how a GENERATE AND TEST algorithm is able to reach the goal state
- At the INITIAL state (we generate), keep on generating possible solution without even considering whether it will **reach the goal** state or whether it is the **optimized choice** or not.

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

Generate-And-Test

- State Space Representation of the Problem

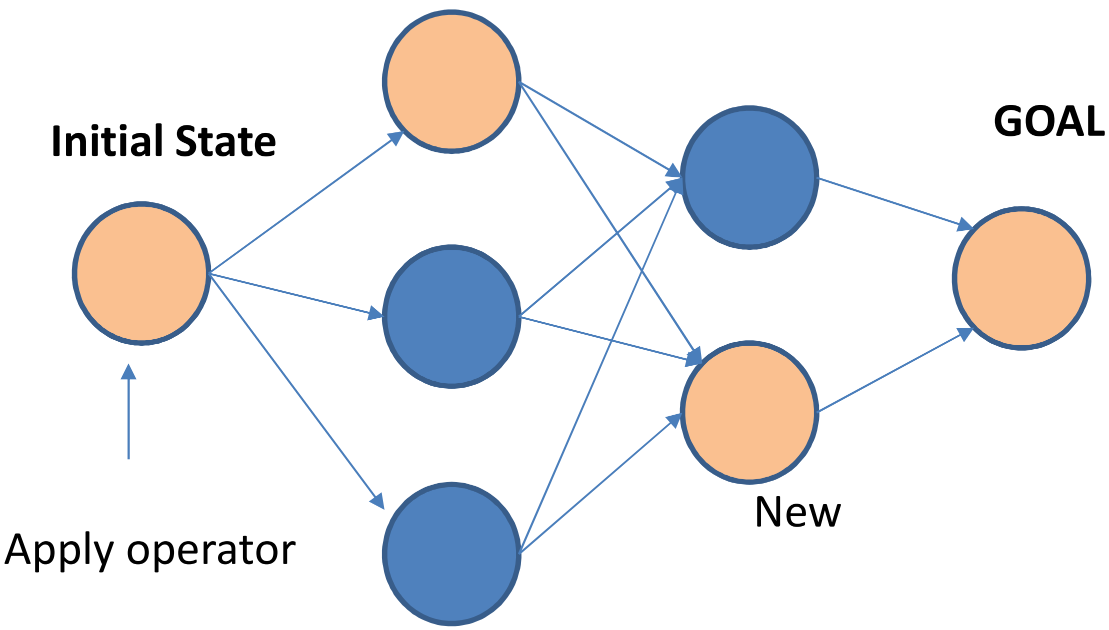


- But we could optimize it using HEURISTIC
- GENERATE AND TEST is not a heuristic technique
- If we will put HEURISTIC functions to make the choices of the possible solutions – then **?**

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

Generate-And-Test

- State Space Representation of the Problem

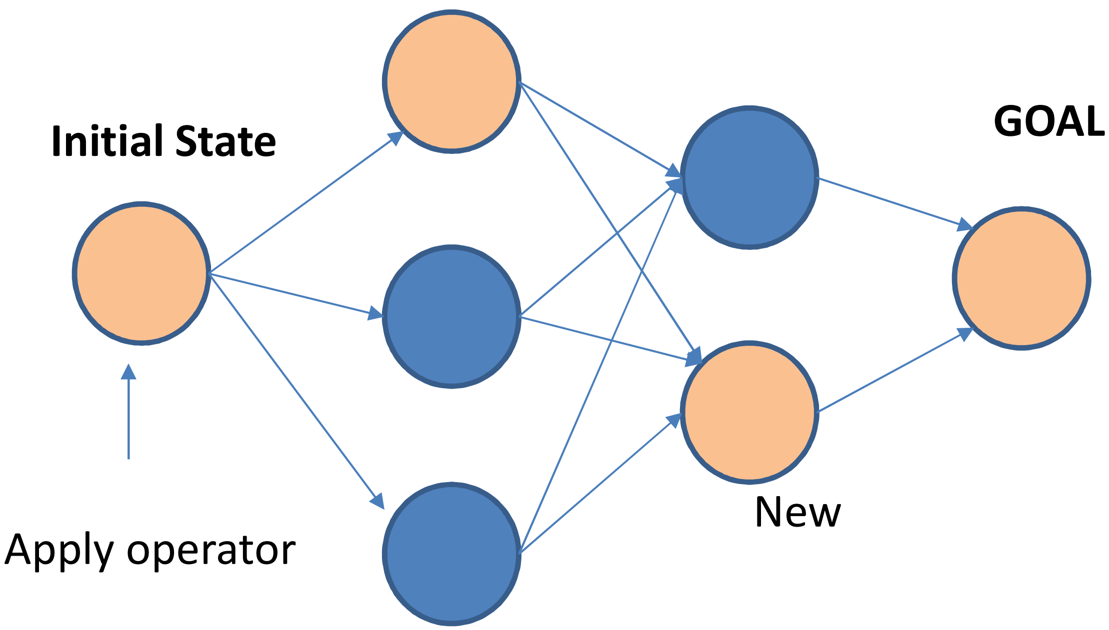


- But we could optimize it using HEURISTIC
- GENERATE AND TEST is not a heuristic technique
- If we will put HEURISTIC functions to make the choices of the possible solutions – then it is called as SIMPLE HILL CLIMBING Algorithm

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

Generate-And-Test

- State Space Representation of the Problem

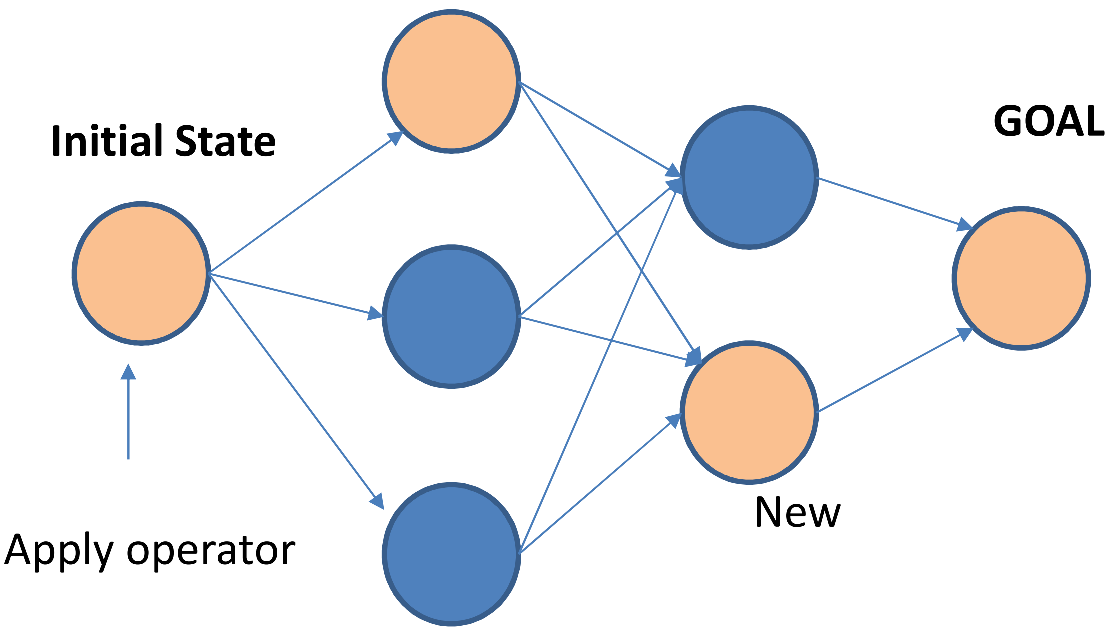


- This type of search technique is known as **EXHAUSTIVE SEARCH**
- Otherwise known as **DEPTH FIRST SEARCH**
- **Reason**: Because it is going to take a LOT of steps or a LOT of random states to reach GOAL state
- It leads to a **LOT of TIME** to reach the GOAL state

1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

Generate-And-Test

- State Space Representation of the Problem



- If we will use the HEURISTIC function in the DECISION Making of which state should we choose next, then we can **optimize** it

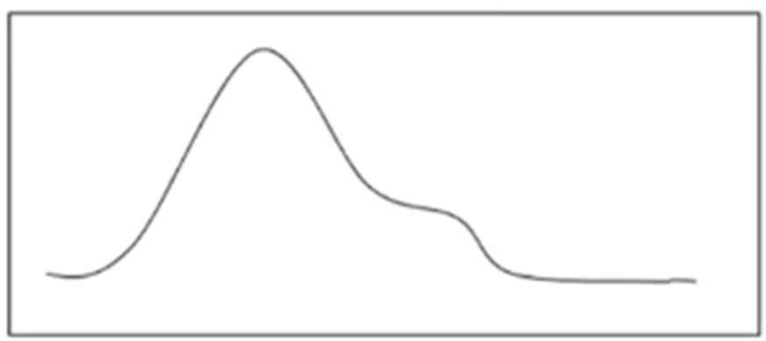
1. Generate current state as initial state
2. Generate possible solution by applying operator
3. Compare newly generated solution with goal state
4. **If solution is found, quit else return to step 2**

Simple Hill Climbing Algorithm

Generate-And-Test + Heuristic

Hill Climbing - Introduction

- Local search method
- uses an iterative improvement strategy
- continuously moves in the direction of increasing elevation
- used for optimizing the mathematical problems



Steps in Hill Climbing

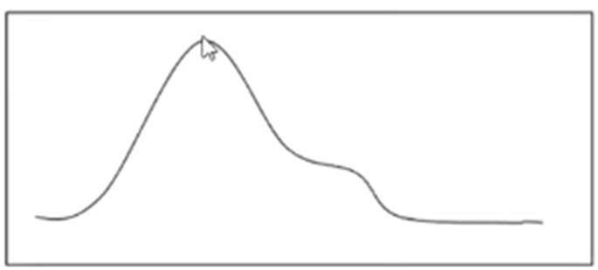
- Applied to a single point - the current point (or state) - in the search space.
- At each iteration, a new point x' is selected by performing a small displacement or perturbation in the current point x, i.e., the new point is selected in the neighbourhood of the current point: $x' \in N(x)$.

Steps in Hill Climbing

- Depending on the representation used for x, this can be implemented by simply adding a small random number, Δx, to the current value of x: $x' = x + \Delta x$.
- If that new point provides a better value for the evaluation function, then the new point becomes the current point.
- Else, some other displacement is promoted in the current point (a new neighbour is chosen) and tested against its previous value.

Stopping Criteria – Hill Climbing

- **No Further improvement can be made**
  - ✓ No near-by points to the optimal solution are better
- **A fixed number of iteration have been performed**
- **A goal point is attained**



Algorithm – Hill Climbing

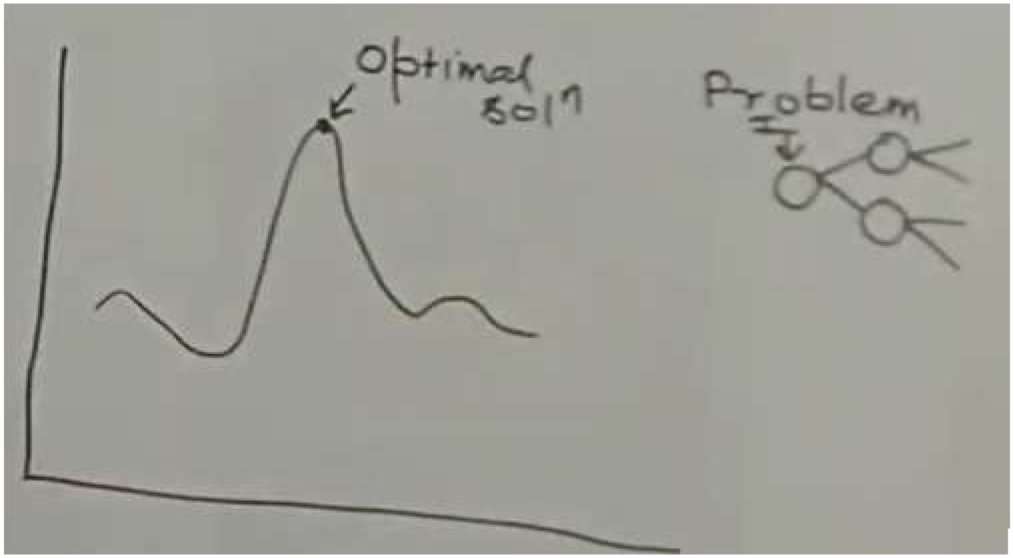
```
procedure [x] = hill-climbing(max_it,g)
  initialize x
  eval(x)
  t ← 1
  while t < max_it & x != g & no_improvement do,
      x' ← perturb(x)
      eval(x')
      if eval(x') is better than eval(x),
            then x ← x'
      end if
      t ← t + 1
  end while
end procedure
```

A standard (simple) hill-climbing procedure

eval(x) – Objective point => gives the cost factor of a particular state
t – no. of iterations, the algorithm will take
perturb () – small **changes** in input and made it as x’

Simple Hill Climbing Algorithm

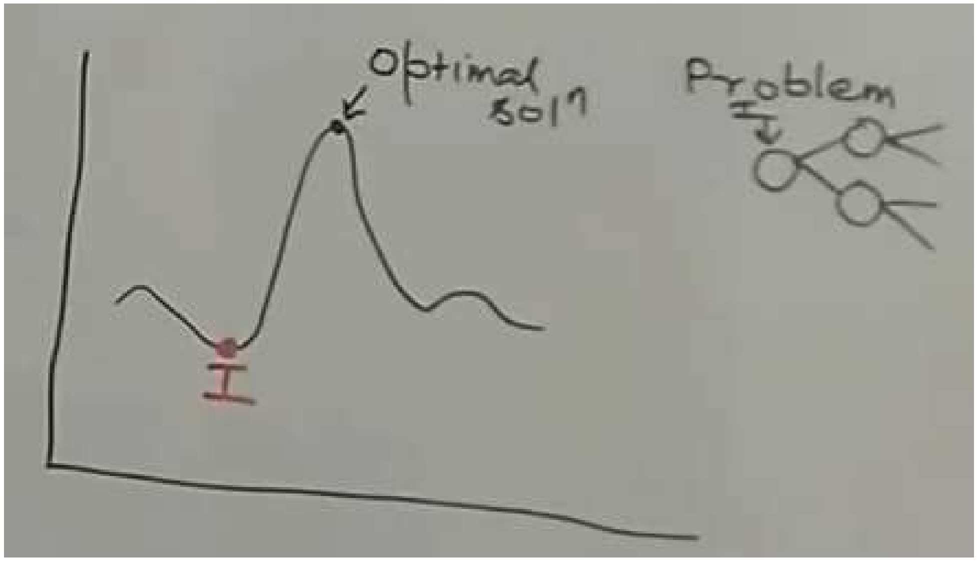
1. Evaluate initial state, if GOAL → **QUIT**
2. LOOP until solution is found or no new operators are left to be applied on the current state
   a) Select and apply an operator (not applied yet) to produce next state
   b) Evaluate new state
      i. If GOAL → **QUIT**
      ii. If it is better than the current state, assign it as current state
      iii. If not better continue in LOOP



This is a graph of a particular problem. It will vary according to the different problems.

Algorithm does not know where / what is the optimum solution

Problem: To find the shortest path from one node to another

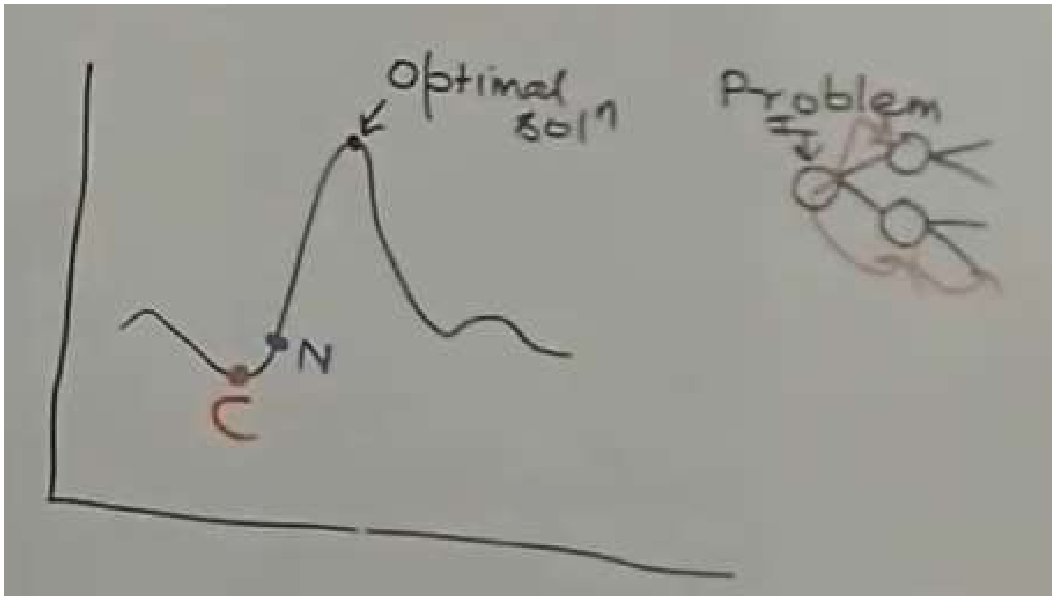


- I = Initial state

Problem: To find the shortest path from one node to another

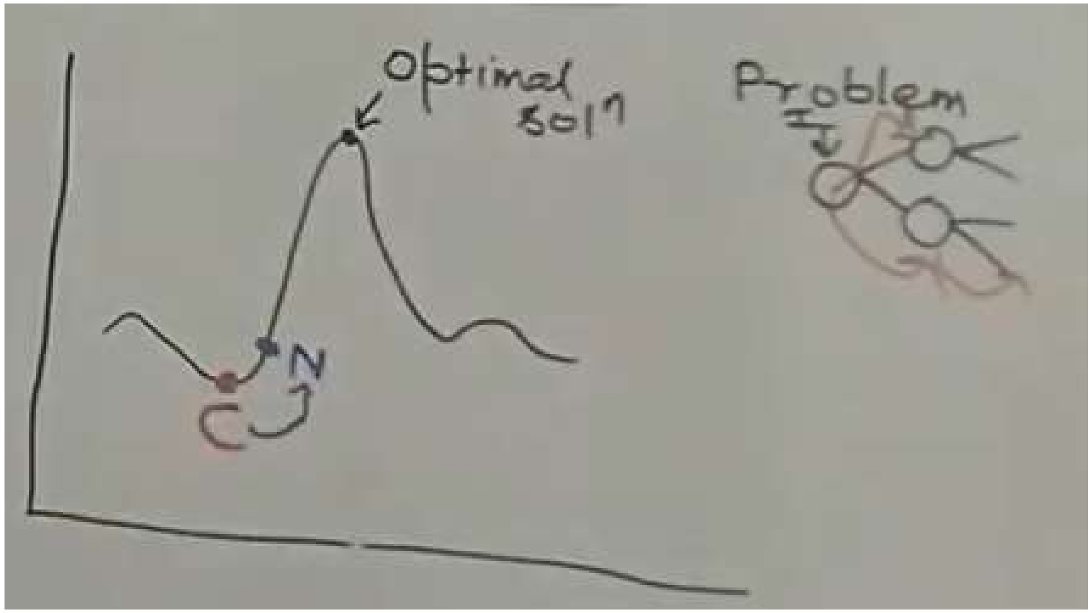
Let I be the INITIAL state

Objective: Need to make sure that I should reach the OPTIMAL solution



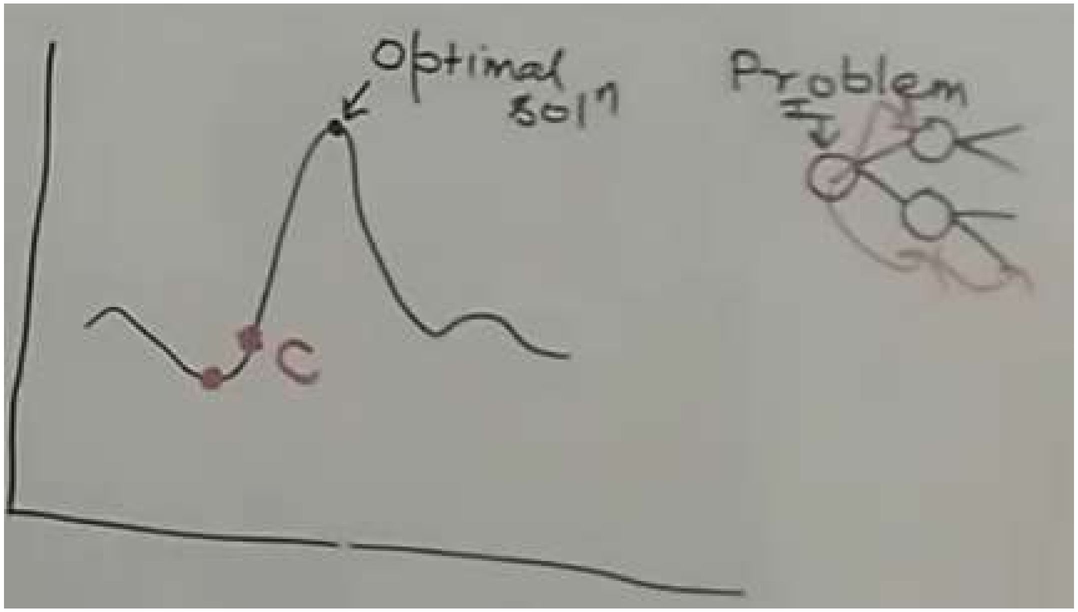
- C represents Current state
- C = I
- N represents **New** state that is generated

1. Evaluate initial state, if GOAL $\longrightarrow$ **QUIT**
2. LOOP until solution is found or no new operators are left to be applied on the current state
   a) **Select and apply an operator (not applied yet) to produce next state**
   b) Evaluate new state
      i. If GOAL $\longrightarrow$ **QUIT**
      ii. If it is better than the current state, assign it as current state
      iii. If not better continue in LOOP



- C represents Current state
- C = I
- N represents **New** state that is generated

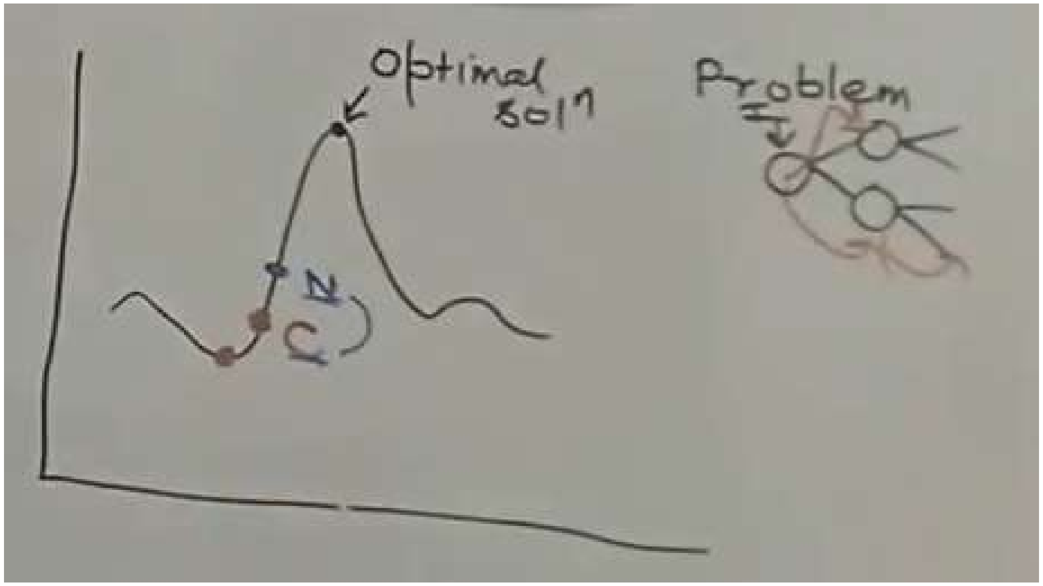
1. Evaluate initial state, if GOAL → **QUIT**
2. LOOP until solution is found or no new operators are left to be applied on the current state
   a) Select and apply an operator (not applied yet) to produce next state
   b) Evaluate new state
      i. If GOAL → **QUIT**
      ii. If it is better than the current state, assign it as current state
      iii. If not better continue in LOOP



- Make the Re-Assignment as

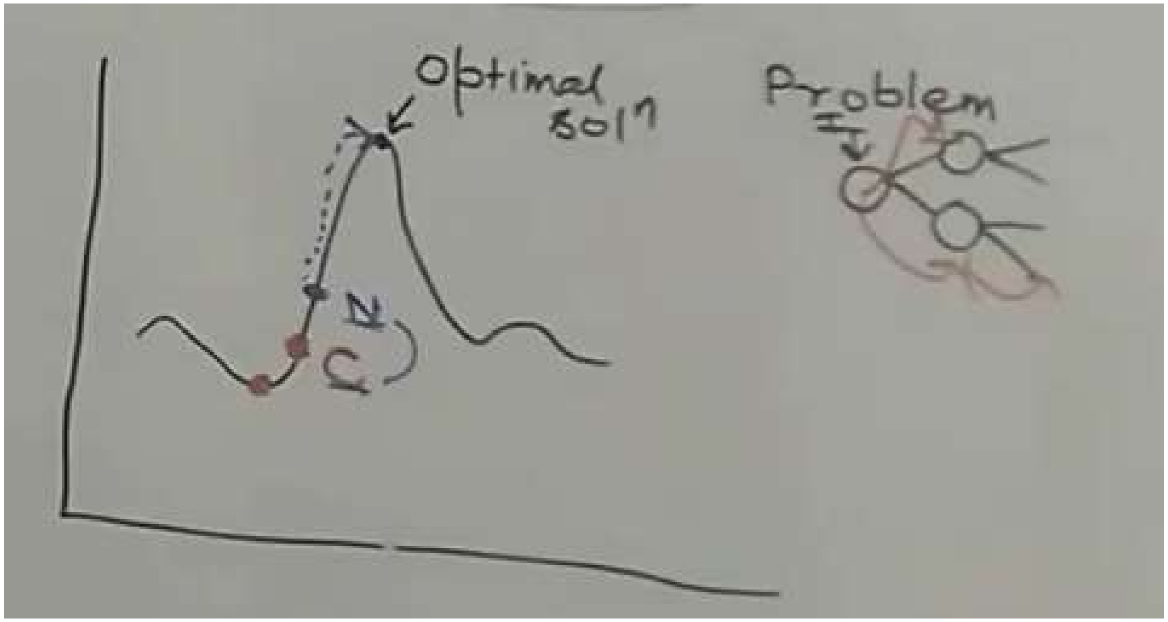
  C = New

1. Evaluate initial state, if GOAL → **QUIT**
2. LOOP until solution is found or no new operators are left to be applied on the current state
   a) Select and apply an operator (not applied yet) to produce next state
   b) Evaluate new state
      i. If GOAL → **QUIT**
      ii. If it is better than the current state, assign it as current state
      iii. If not better continue in LOOP



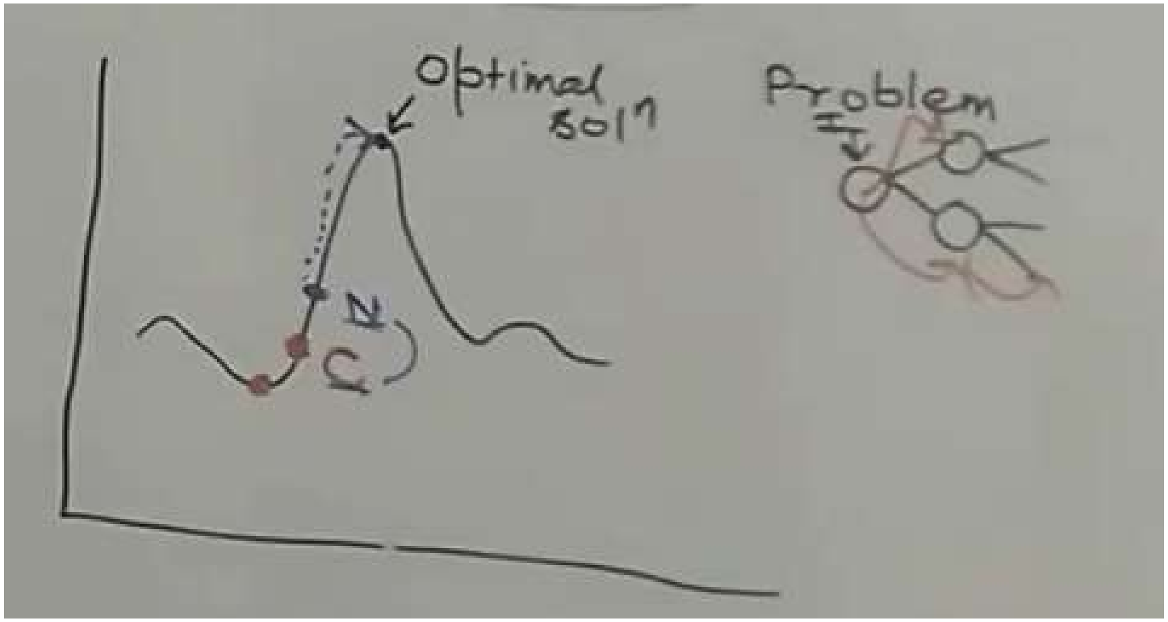
- N represents **New** state that is generated

1. Evaluate initial state, if GOAL ⟶ **QUIT**
2. LOOP until solution is found or no new operators are left to be applied on the current state
   a) **Select and apply an operator (not applied yet) to produce next state**
   b) Evaluate new state
      i. If GOAL ⟶ **QUIT**
      ii. If it is better than the current state, assign it as current state
      iii. If not better continue in LOOP



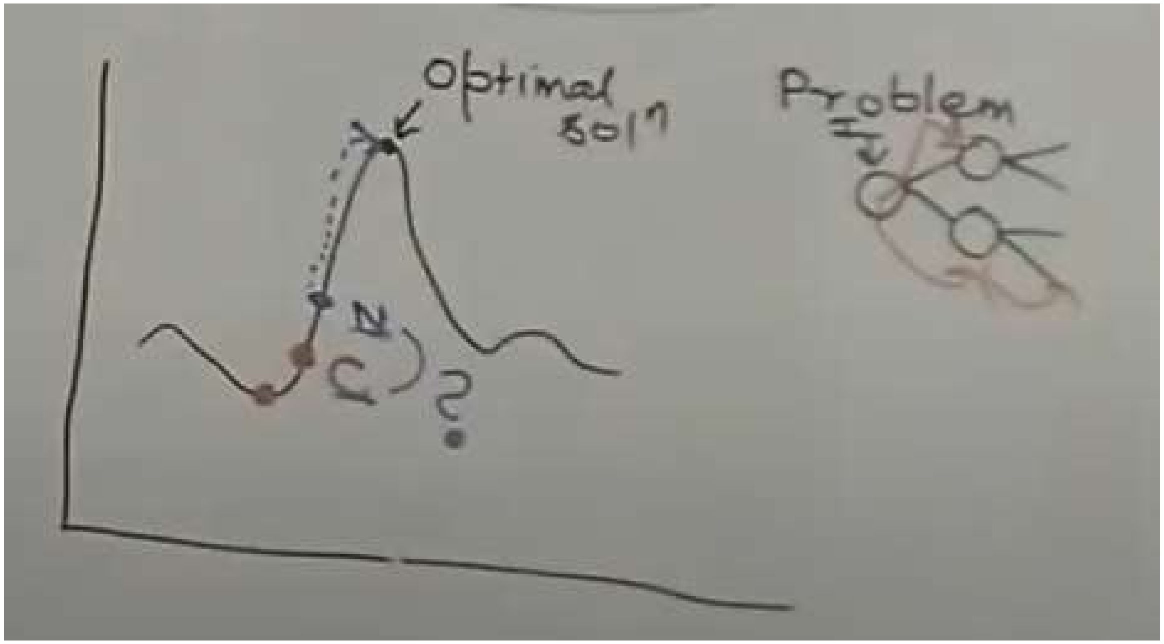
- Re-Assignment as
  C = N
- This will happen again and again until and unless we reach the OPTIMAL SOLUTION
- On reaching the optimal solution, i.e., GOAL, then **QUIT**

1. Evaluate initial state, if GOAL ⟶ **QUIT**
2. **LOOP until solution is found or no new operators are left to be applied on the current state**
   a) Select and apply an operator (not applied yet) to produce next state
   b) Evaluate new state
      i. If GOAL ⟶ **QUIT**
      ii. If it is better than the current state, assign it as current state
      iii. If not better continue in LOOP



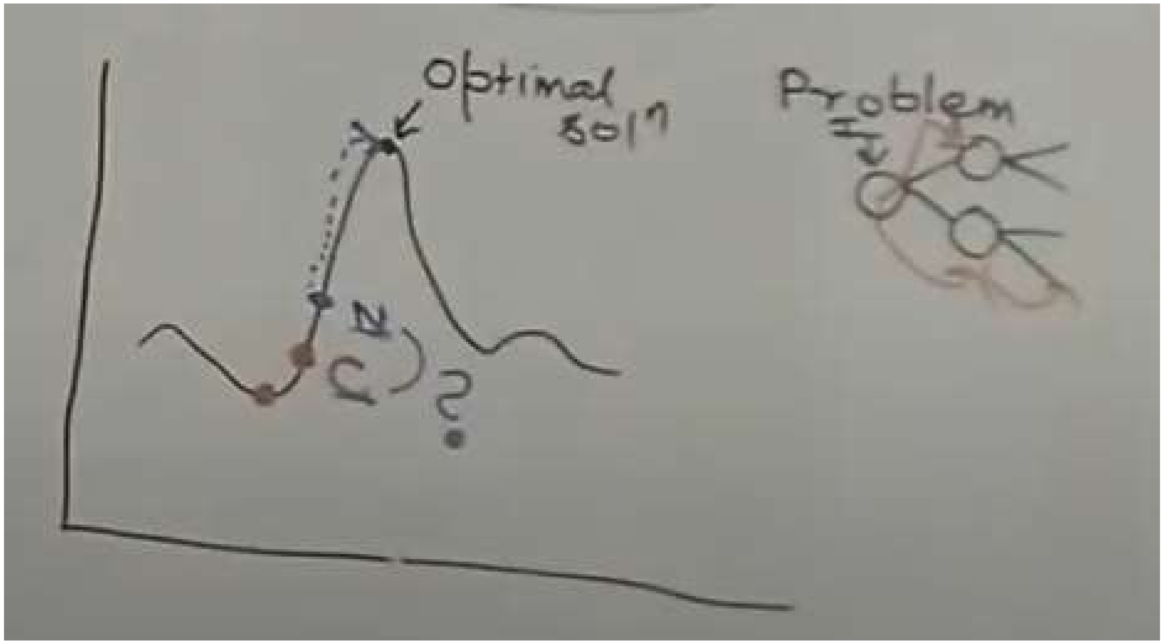
- This is a simple Hill Climbing algorithm
- This curve is the **HILL**
- Trying to climb the HILL by doing the comparisons between current state and new state

1. Evaluate initial state, if GOAL → **QUIT**
2. LOOP until solution is found or no new operators are left to be applied on the current state
   a) Select and apply an operator (not applied yet) to produce next state
   b) Evaluate new state
      i. If GOAL → **QUIT**
      ii. If it is better than the current state, assign it as current state
      iii. If not better continue in LOOP



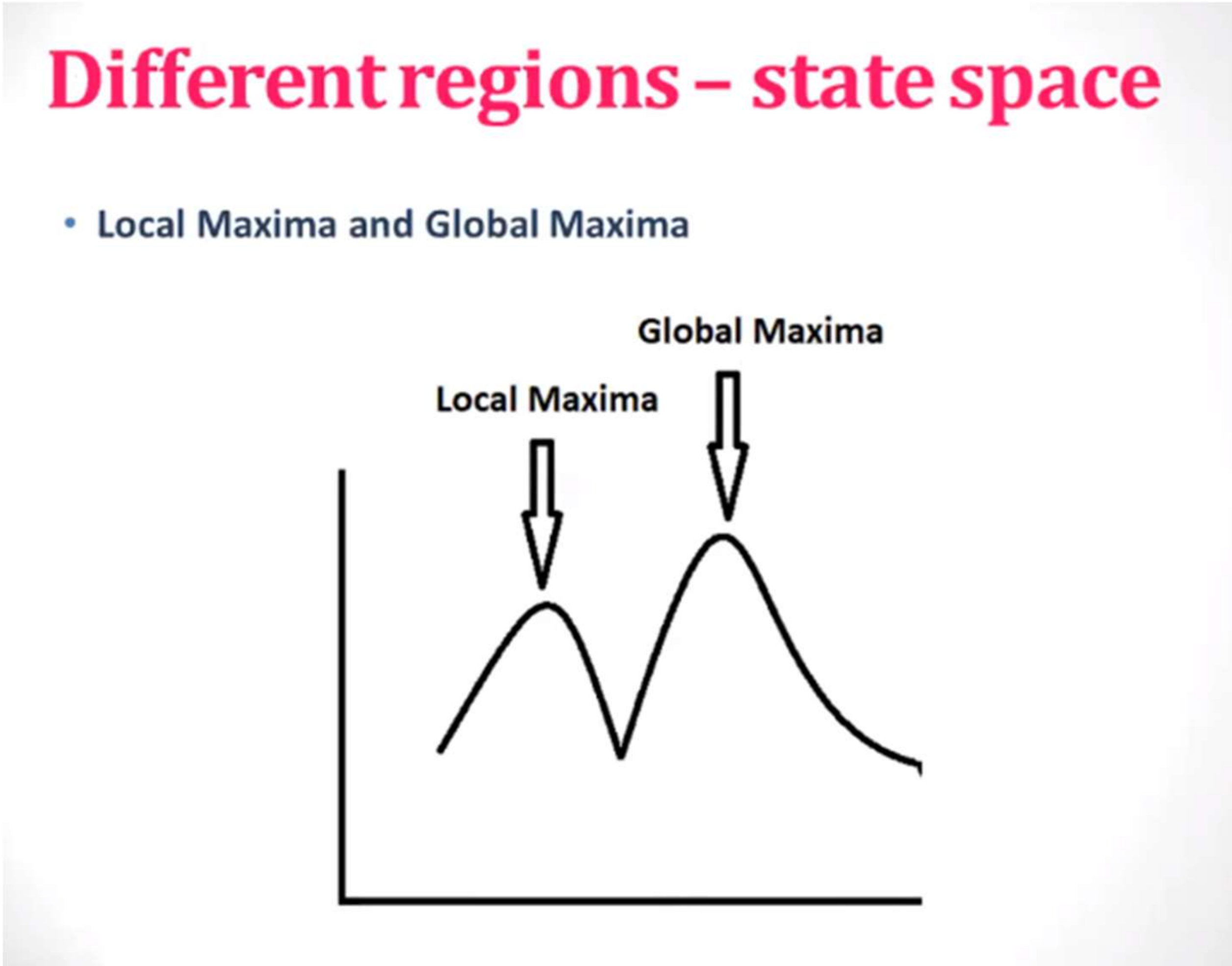
- How will you know the NEXT state is **BETTER** than the CURRENT state?

1. Evaluate initial state, if GOAL → **QUIT**
2. LOOP until solution is found or no new operators are left to be applied on the current state
   a) Select and apply an operator (not applied yet) to produce next state
   b) Evaluate new state
      i. If GOAL → **QUIT**
      ii. If it is better than the current state, assign it as current state
      iii. If not better continue in LOOP

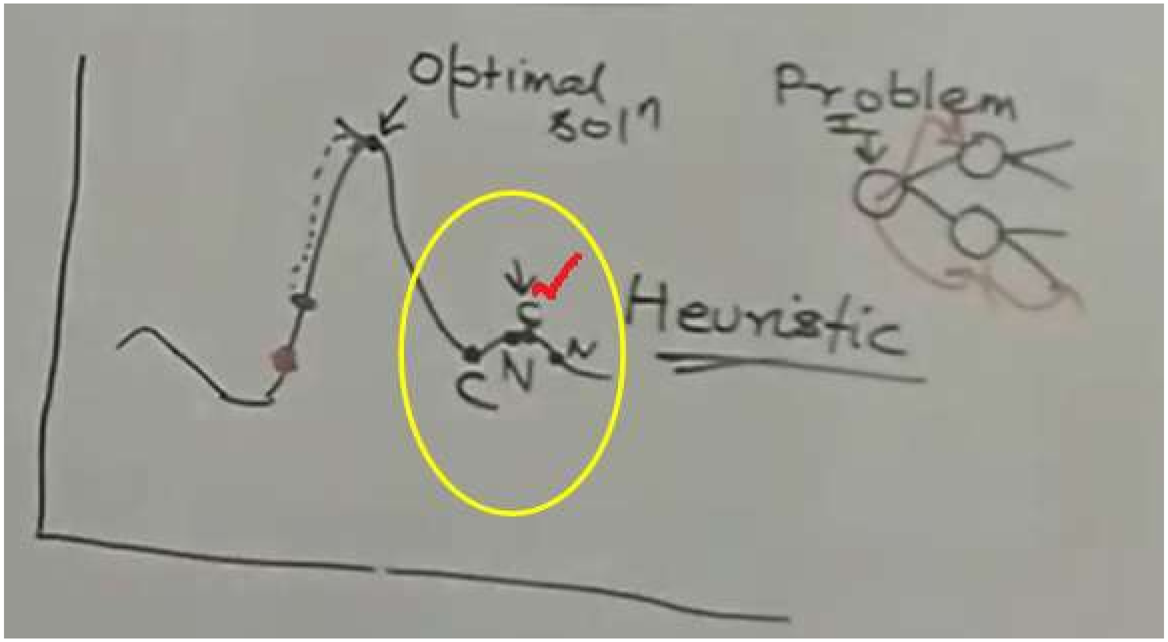


- **Using HEURISTIC function**
- Compare the values of the next state with the current state
- That's y, it is said to be G & T + Heuristic

1. Evaluate initial state, if GOAL ⟶ **QUIT**
2. LOOP until solution is found or no new operators are left to be applied on the current state
   a) Select and apply an operator (not applied yet) to produce next state
   b) Evaluate new state
      i. If GOAL ⟶ **QUIT**
      ii. If it is better than the current state, assign it as current state
      iii. If not better continue in LOOP



Case (i): Algorithm may get **stop** at **LOCAL maximum** point, thinking that no points are better than that point(**LOCAL maximum**)

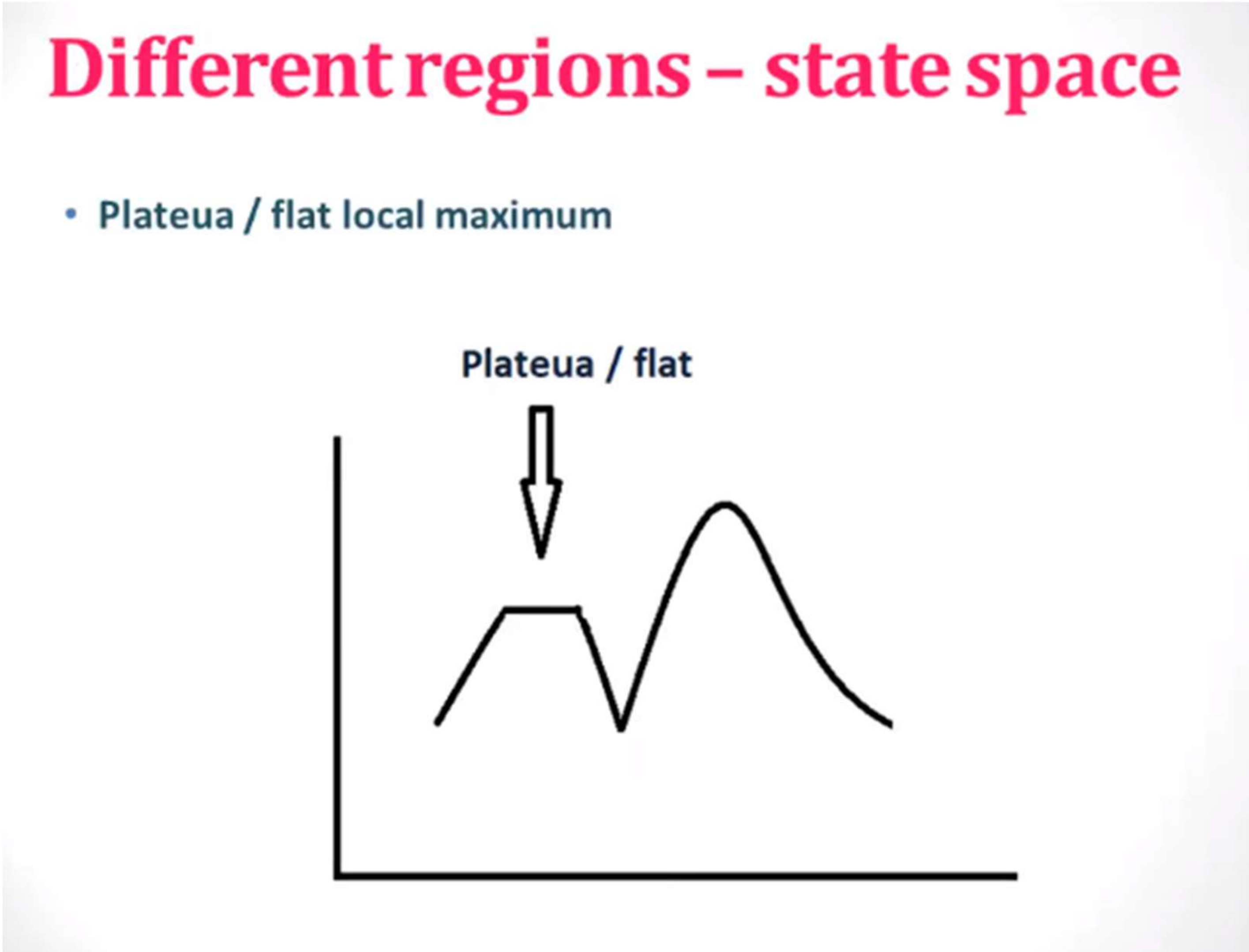


- **Using HEURISTIC function**
- Compare the values of the next state with the current state

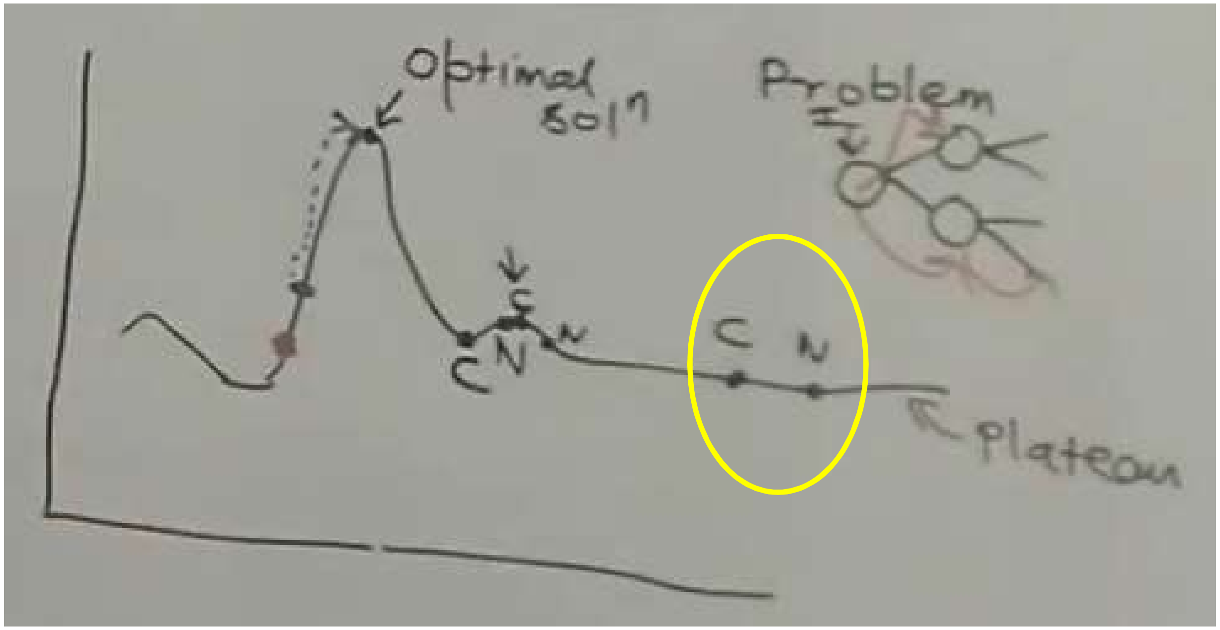
- That's y, it is said to be G & T + Heuristic

Drawbacks:

1. **Case (i)** If we choose the CURRENT point in the near-by hill, then New state that will generate, will be NEXT (Right) to it.
2. Again repeat the same process, with Reassignment as Current = New
3. Now, the New state that has generated will be lower (not better) than Current
4. So according to the ALGORITHM, the OPTIMAL solution is reached. Therefore, the Current state (**Red tick Mark**) would be declared as optimum solution which is not actual solution.
5. This problem is said to be **LOCAL OPTIMUM** problem
    1. Because, this algor. may get stop at LOCAL Optimum



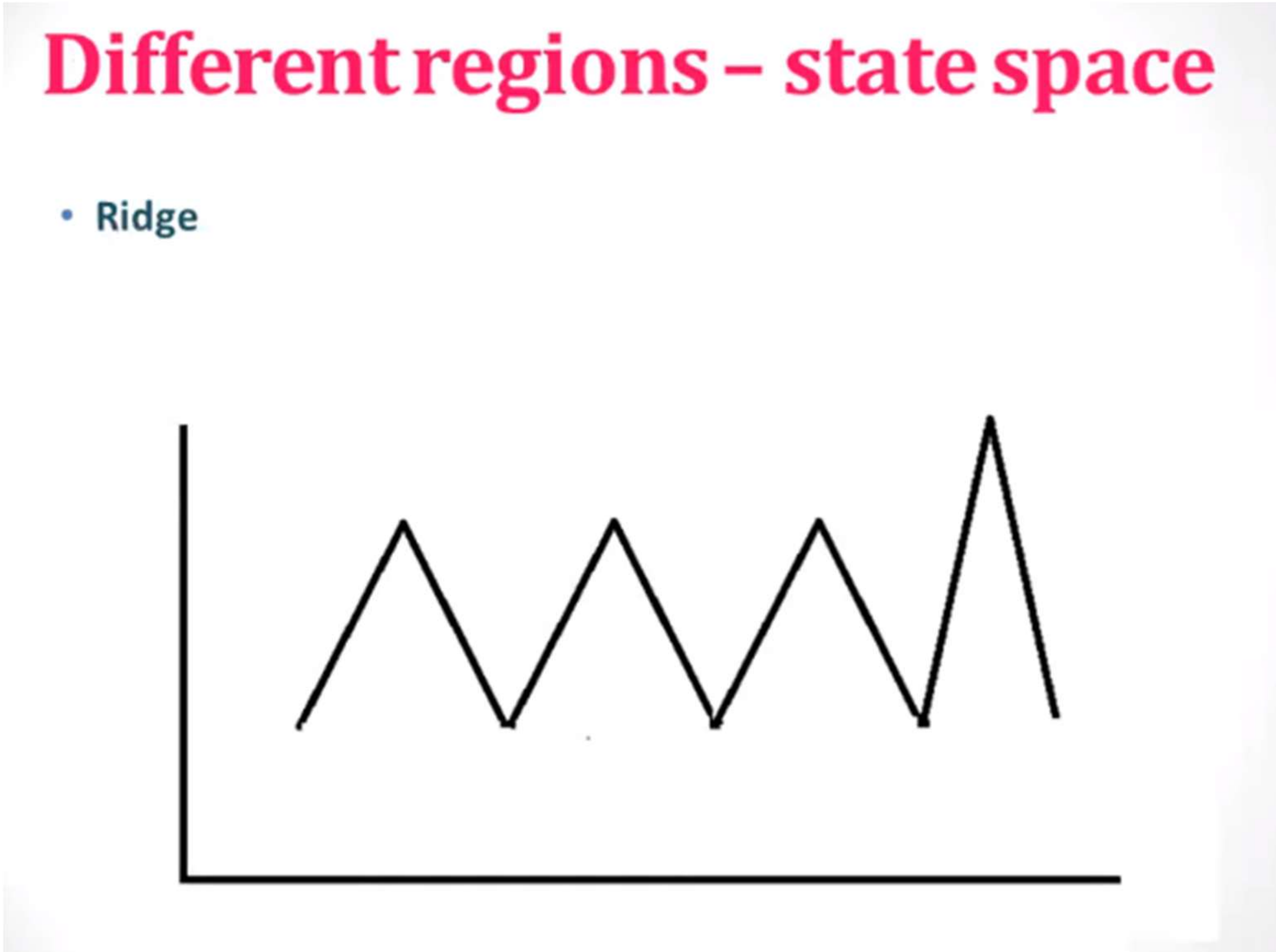
Case (ii): Algorithm may get **stop** at **Falt surface**, as X and X’ are more or less same vale.



- **Using HEURISTIC function**
- Compare the values of the next state with the current state
- That's y, it is said to be G & T + Heuristic

Drawbacks:

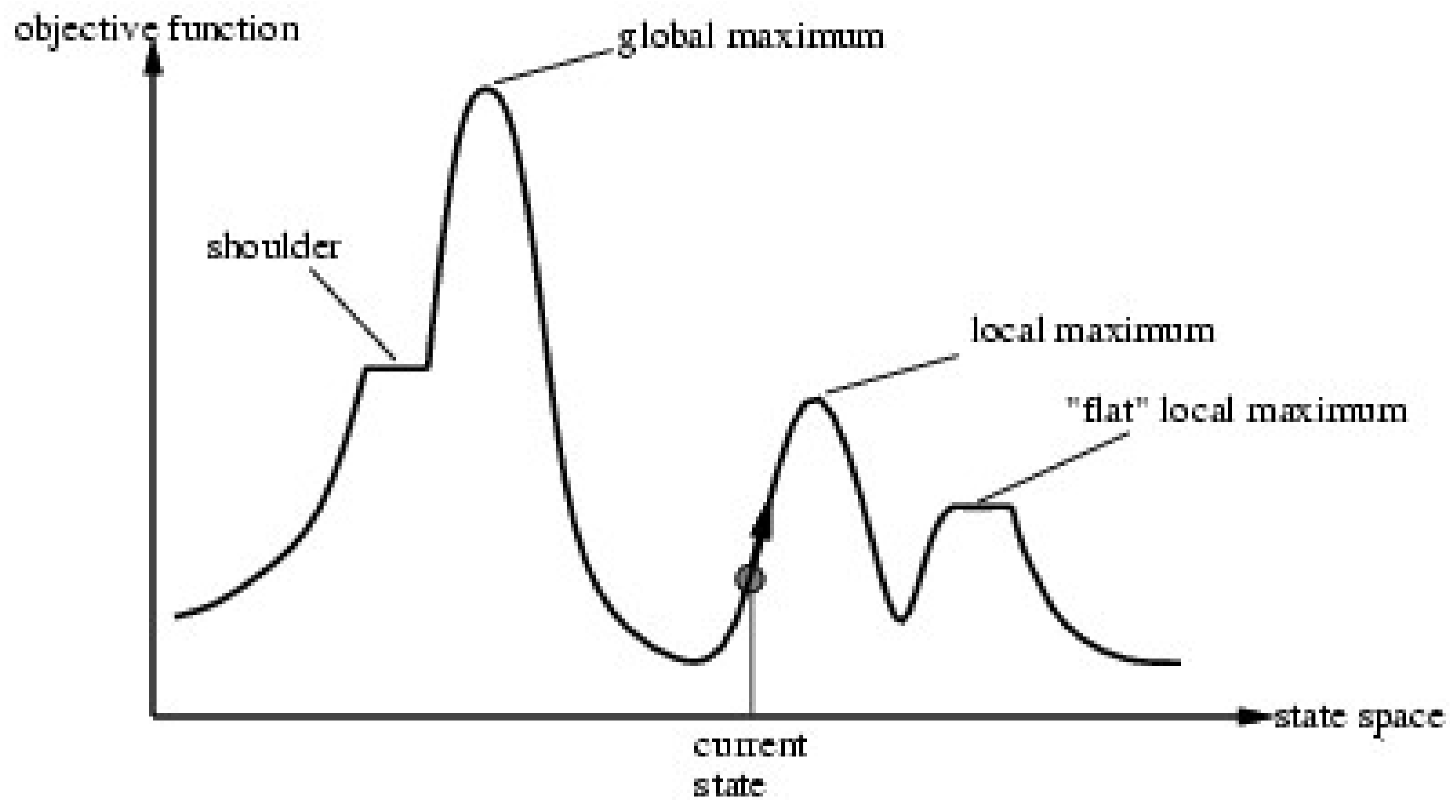
4. So according to the ALGORITHM, the OPTIMAL solution is reached. Therefore, the Current state (**Red tick Mark**) would be declared as optimum solution which is not actual solution.
5. This problem is said to be **LOCAL OPTIMUM** problem
    1. Because, this algor. may get stop at LOCAL Optimum
6. **Case (ii)** If the curve is PLATEAU, [FLAT / STRAIGHT line]
    1. Both Current and Next(New) state are of equal (important)
    2. We will not ne able to reach the Optimal solution in PLATEAU also.



Case (iii): **Ridges - small mounds which lead to multiple LOCAL optimum solutions / points.**

Algorithm may get **stop** at **these local optimum points**, as Global optimum may present farthest from them

Simple Hill-climbing search

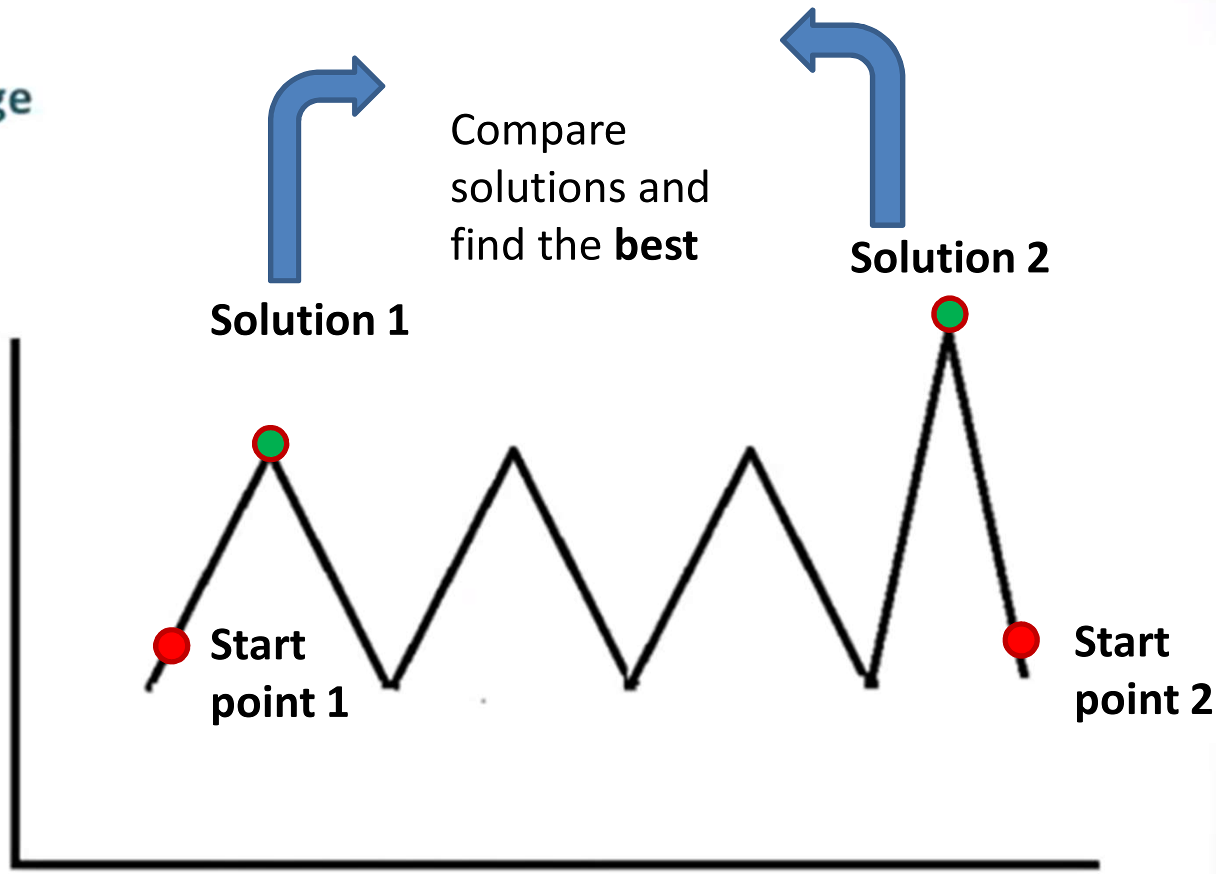
- Problem: depending on initial state, can get stuck in local maxima
- 

- How to resolve these drawbacks of Simple Hill Climbing Algorithm?

- Modify Hill Climbing algor. using Iterative Procedure
  - i.e., same algorithm can be applied with **different starting points**
- Steepest Ascent Hill Climbing Algorithm

Different regions – state space

- Ridge



Hill Climbing - Iterated

Starts from number of random points

```
procedure [best] = IHC(n_start,max_it,g)
  initialize best
  t1 ← 1
  while t1 < n_start & best != g do,
    initialize x
    eval(x)
    x ← hill-climbing(max_it,g) //Algorithm 1
    t1 ← t1 + 1
    if x is better than best,
           then best ← x
    end if
  end while
end procedure
```

An iterated hill-climbing procedure.

Hill Climbing - Iterated

Starts from number of random points

```
procedure [best] = IHC(n_start,max_it,g)
  initialize best
  t1 ← 1
  while t1 < n_start & best != g do,
    initialize x
    eval(x)
    x ← hill-climbing(max_it,g) //Algorithm 1
    t1 ← t1 + 1
    if x is better than best,
          then best ← x
    end if
  end while
end procedure
```

An iterated hill-climbing procedure.

Solve problems of : (i) Local Maxima
(ii) Ridge

Hill Climbing – Stochastic

Accepts x’ with some probability

```
procedure [x] = stochastic hill-climbing(max_it,g)
  initialize x
  eval(x)
  t ← 1
  while t < max_it & x != g do,
     x’ ← perturb(x)
    eval(x’)
    if random[0,1) < (1/(1+exp[(eval(x)-eval(x’))/T])),
           then x ← x’
    end if
    t ← t + 1
  end while
end procedure
```

A stochastic hill-climbing procedure

- **When eval(x’) is not better than eval(x), we could allow/accept x’ until it falls under certain probability.**
- probability function used here is **Gaussian distribution**
- **i.e.,** x’ can be taken as new point and search for any new solution
- **Also,** it has the probability of accepting the new points on the flat surface with certain limits

Hill Climbing – Stochastic

Accepts x’ with some probability

```
procedure [x] = stochastic hill-climbing(max_it,g)
  initialize x
  eval(x)
  t ← 1
  while t < max_it & x != g do,
     x’ ← perturb(x)
    eval(x’)
    if random[0,1) < (1/(1+exp[(eval(x)-eval(x’))/T])),
           then x ← x’
    end if
    t ← t + 1
  end while
end procedure
```

A stochastic hill-climbing procedure

Solve problems of : Plateau

Hill Climbing Method - Summary

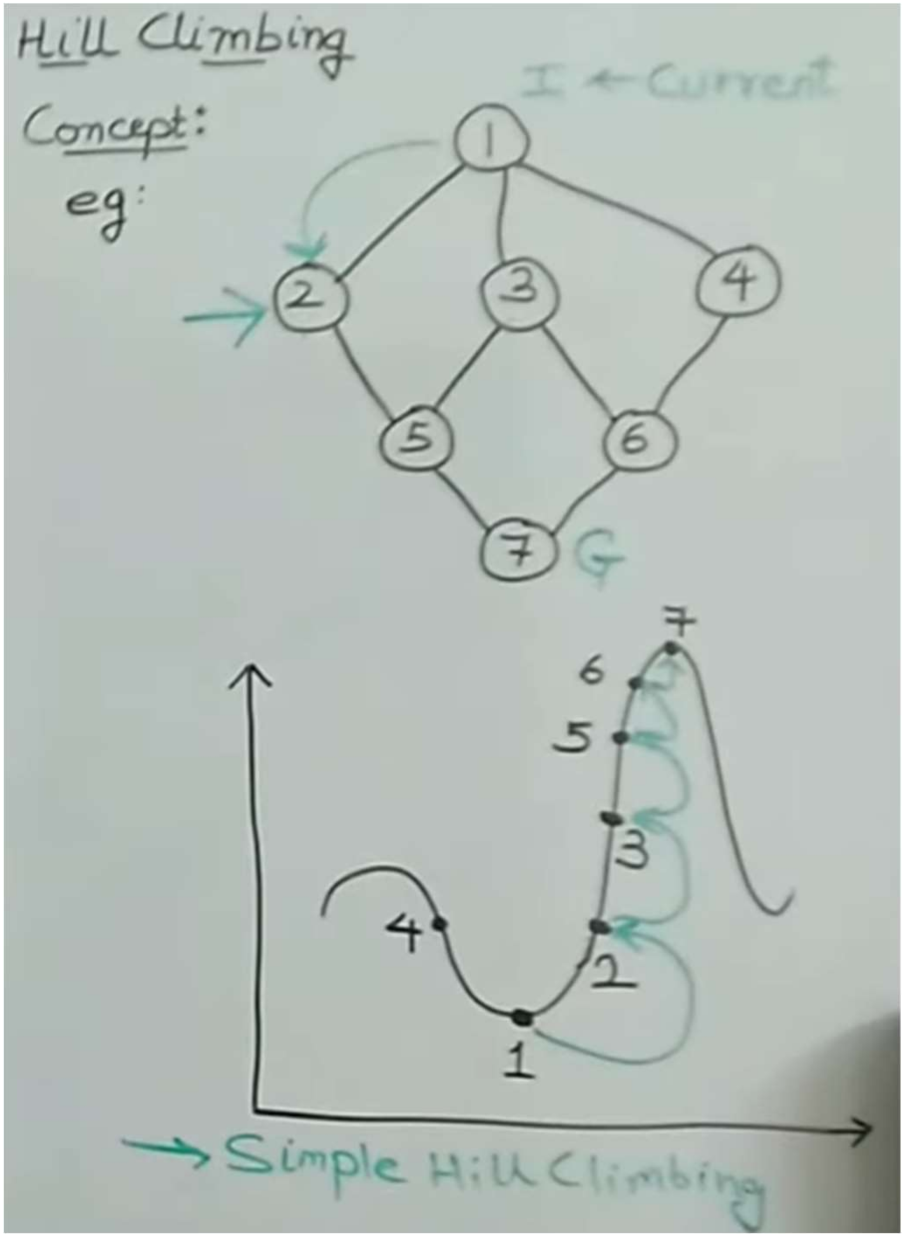
- Local search method
- It is used for optimization problem
  - Pick the best solution among the already existing solutions
- It can give the best solution every time
  - Need to work out to make it an optimized one

Steepest Hill Climbing Algorithm

How to choose the next node using Heuristic Value

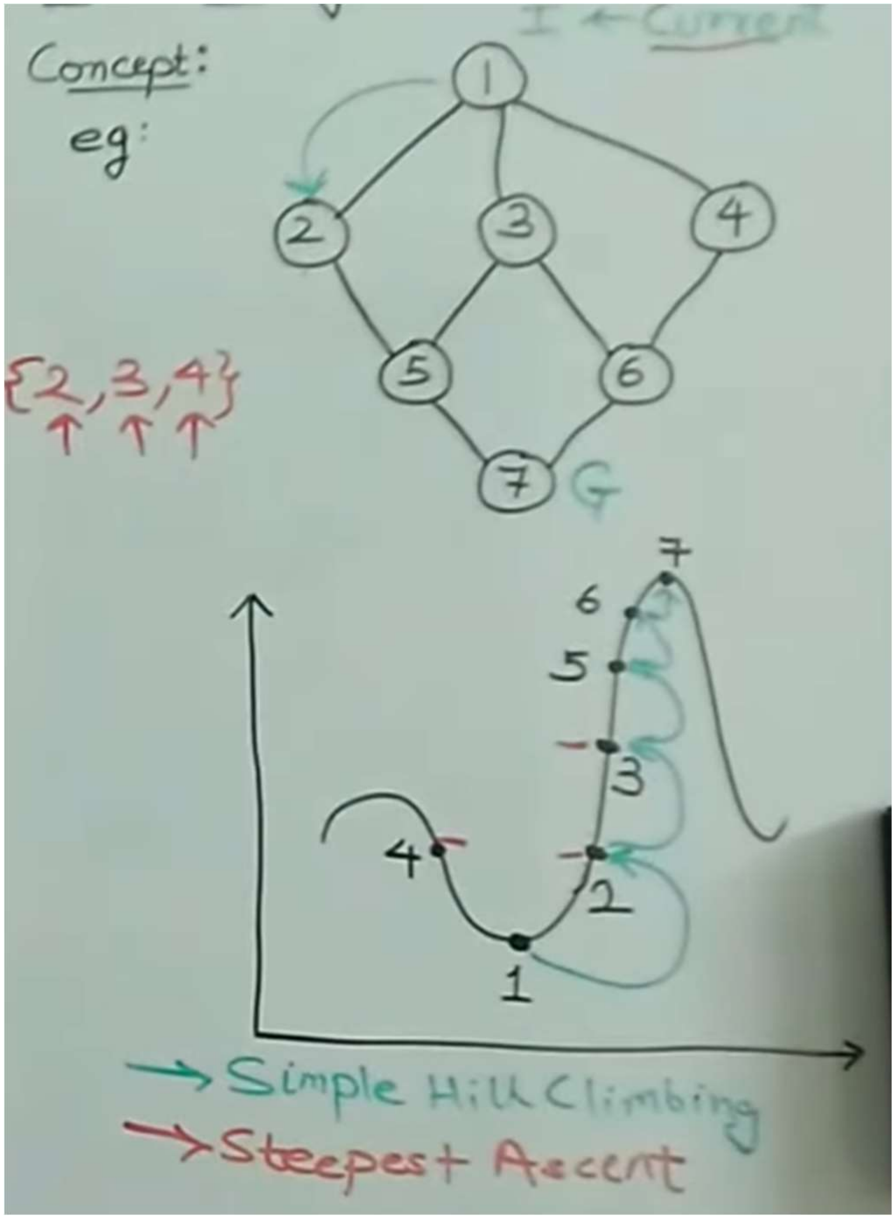
(**Logic**: Finding the maximum among n numbers)

Simple Hill Climbing



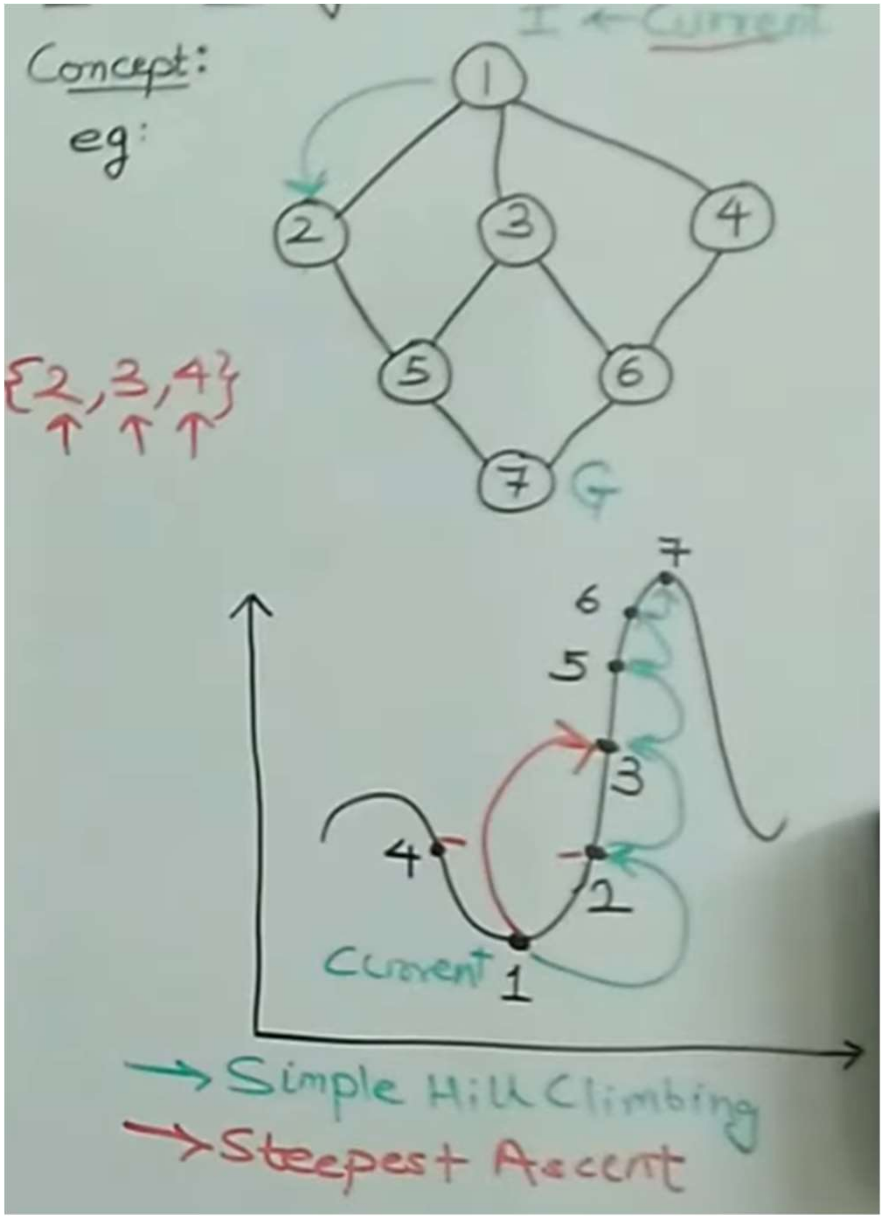
- Plot the curve according to its heuristic values
- At every step / iteration, we reassign the current state

Steepest Ascent Hill Climbing



- Explore all branches that are expanding from the CURRENT node
- Pick the best among all explored nodes
- From the graph, we see C is the BEST state

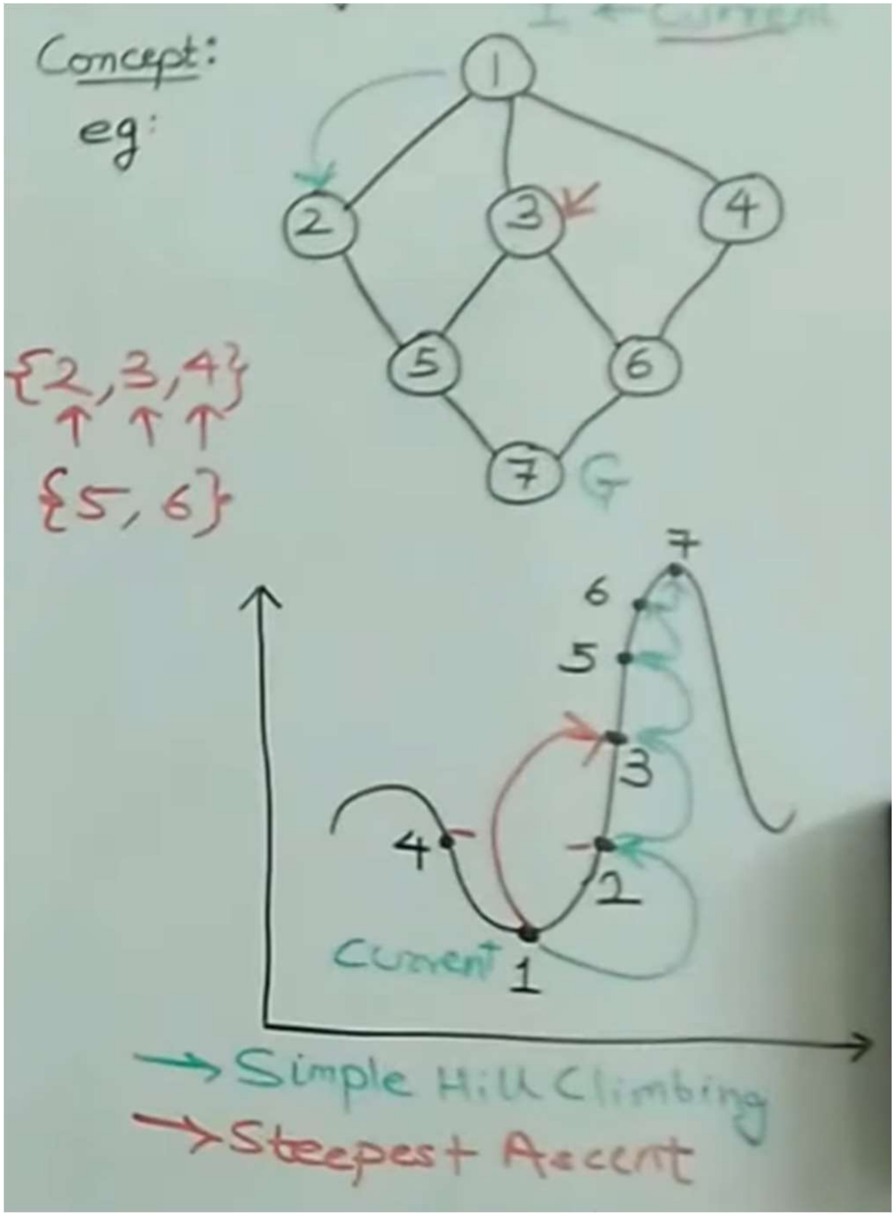
Steepest Ascent Hill Climbing



- Explore all branches that are expanding from the CURRENT node
- Pick the best among all explored nodes
- From the graph, we see 3 is the BEST state

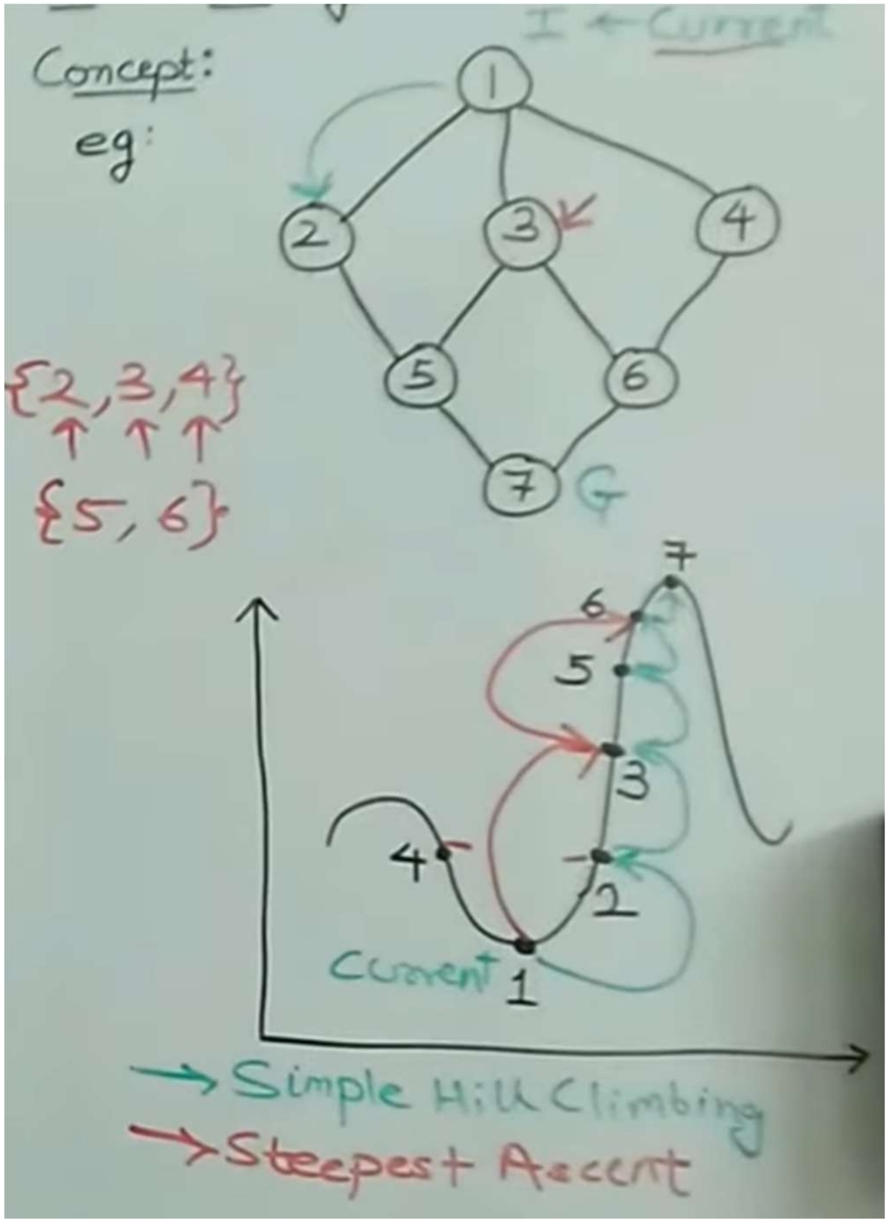
- Now the Current state is 3

Steepest Ascent Hill Climbing



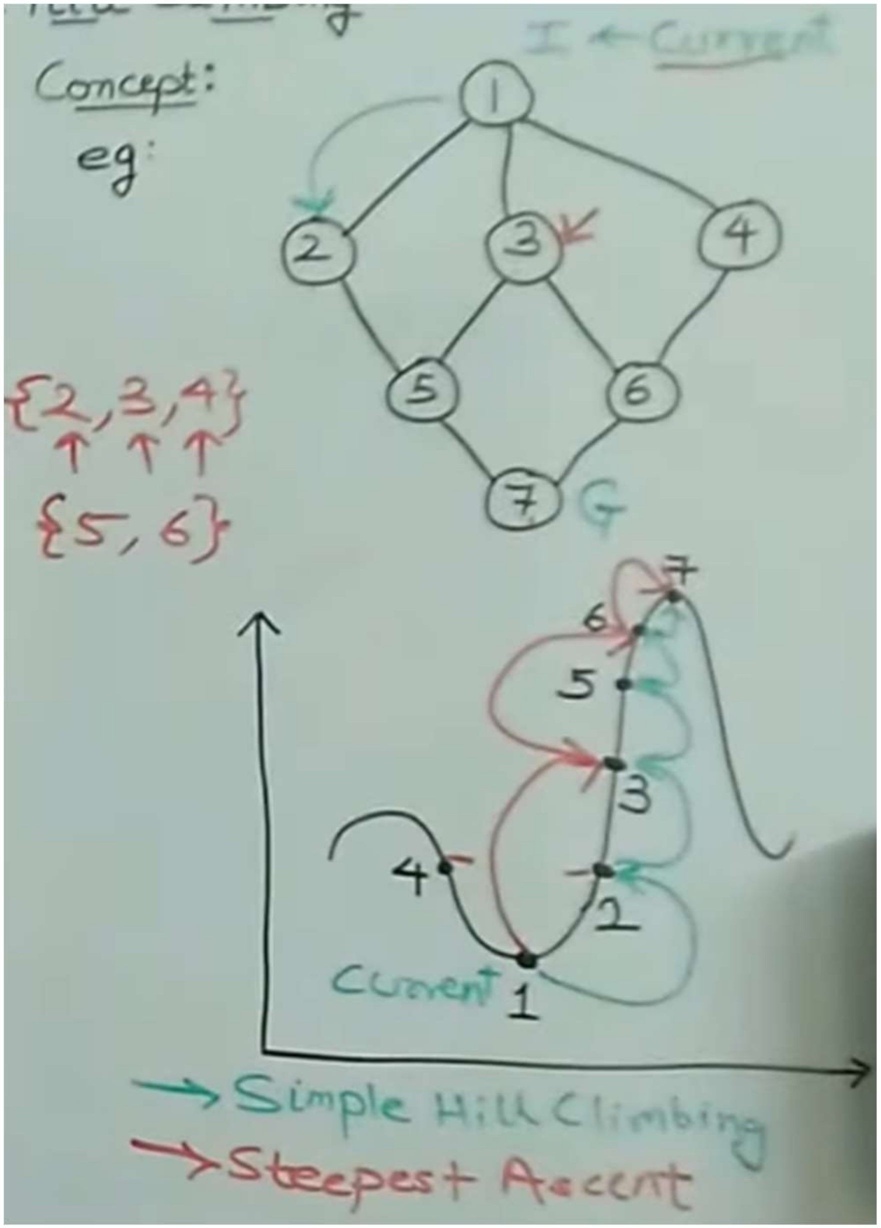
- Explore all branches (**5 and 6**) that are expanding from the CURRENT node 3
- Pick the best among all explored nodes
- From the graph, we see 6 is the BEST state

Steepest Ascent Hill Climbing



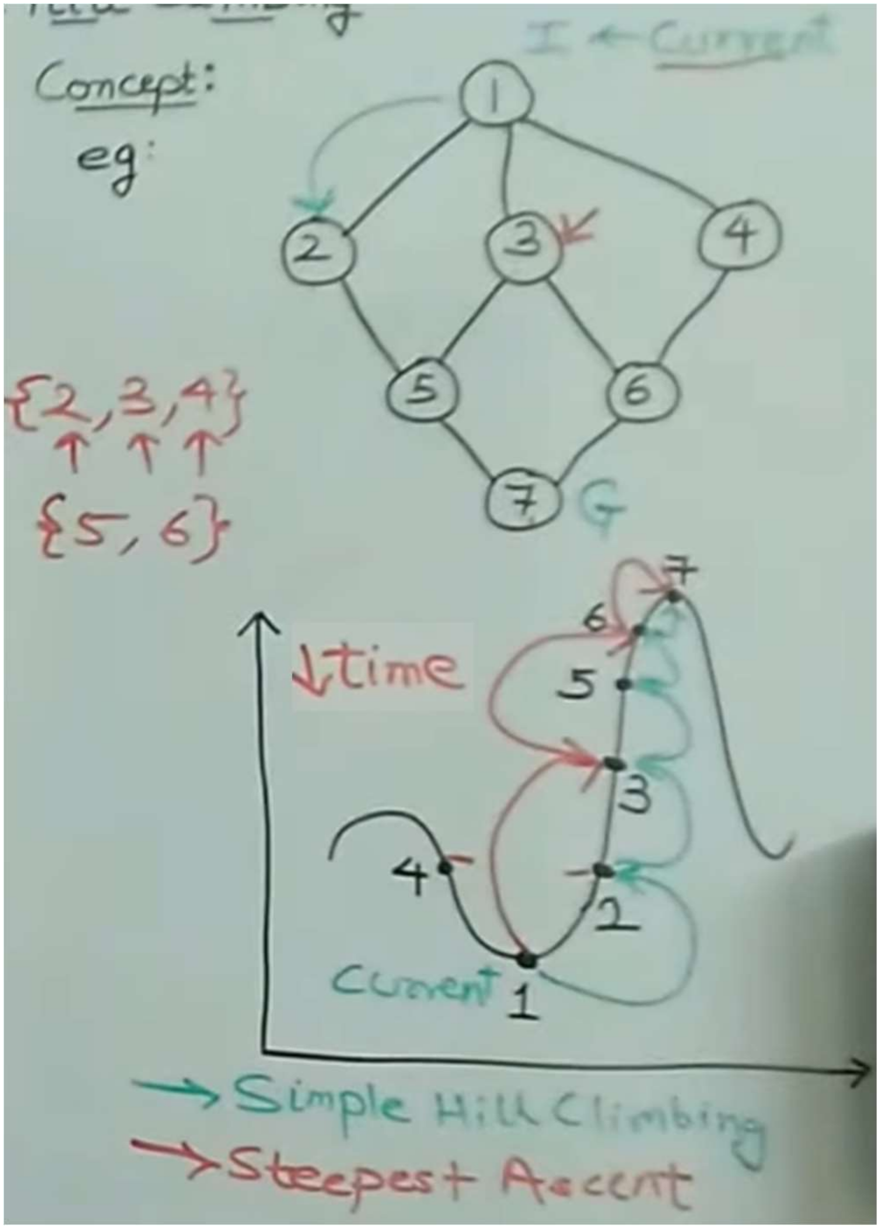
- Explore all branches (**5 and 6**) that are expanding from the CURRENT node 3
- Pick the best among all explored nodes
- From the graph, we see 6 is the BEST state

Steepest Ascent Hill Climbing



- Finally we reach the **goal** state **7**

Steepest Ascent Hill Climbing



Diif b/w Simple and steepest hill climbing

- **Simple :** Smaller steps are taken
- **Steepest Ascent :** Increasing and bigger steps are taken
  Steepest means Bigger steps
  Ascent means Increasing

Adv of Steepest Ascent : Consume LESS TIME

Steepest Ascent Hill Climbing

1. Evaluate initial state, if GOAL
   QUIT
2. Let SUCC -> state: any possible successors of current state is better than SUCC
3. LOOP until solution is found
   a) Apply operator to current state
      i. Generate NEW state
      ii. Evaluate NEW state, if not GOAL compare with SUCC
      iii. If NEW state is better, SUCC=NEW state
      else NO CHANGE in SUCC

   b) If SUCC is better than current state,
      CURRENT state = SUCC

Diif b/w Simple and steepest hill climbing

- **Simple :** Smaller steps are taken
- **Steepest Ascent :** Increasing and bigger steps are taken
  Steepest means Bigger steps
  Ascent means Increasing

Adv of Steepest Ascent : Consume LESS TIME

SUCC

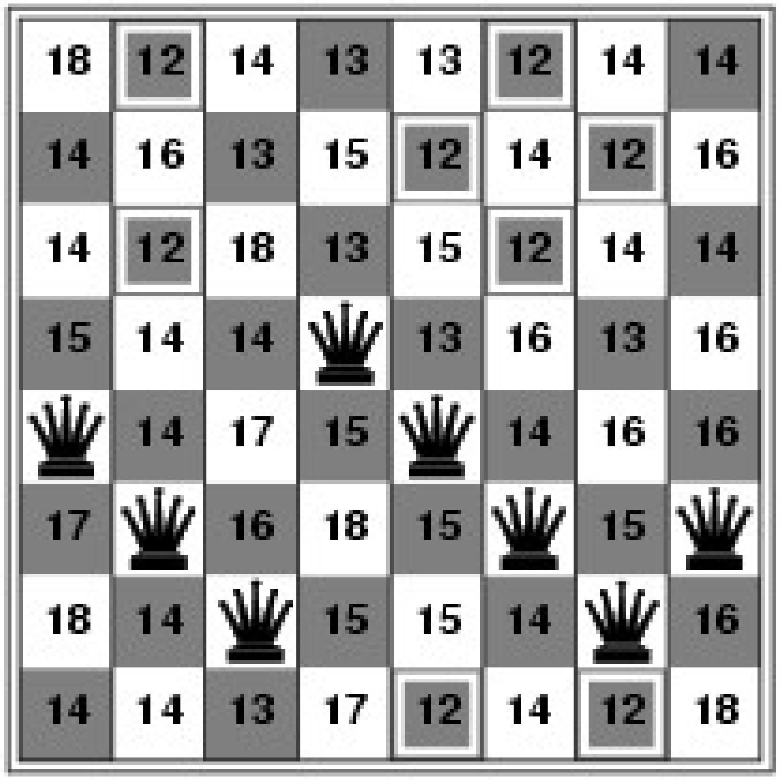
[4, 7, 5] max = -1

4 < 7 max = 7
5 < 7 max = 7 (No change)

Minimization problem:
Min = 99999

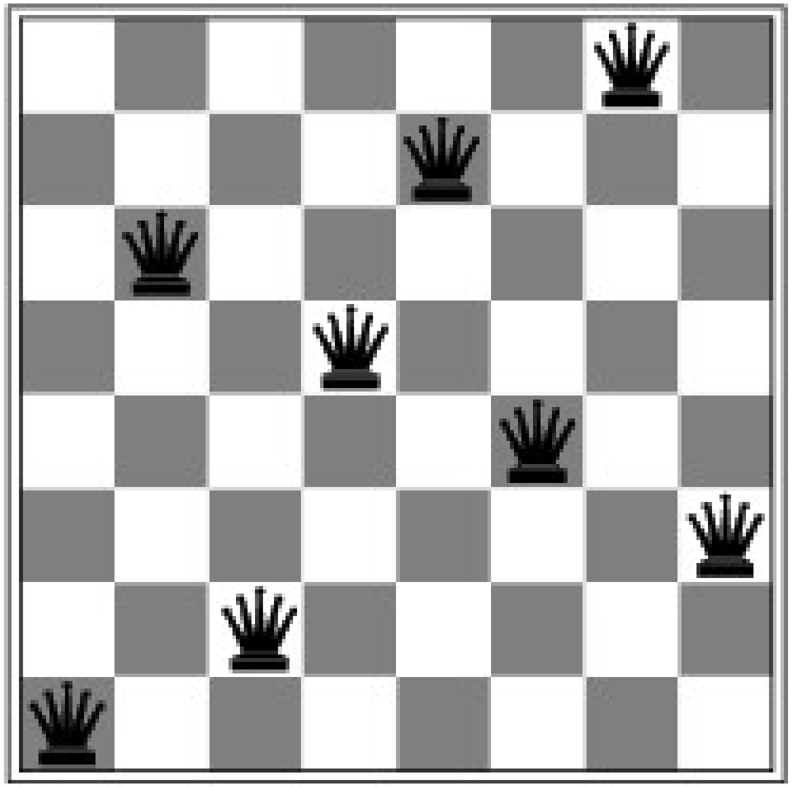
Applications

Hill-climbing search: 8-queens problem



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h = 17* for the above state

Hill-climbing search: 8-queens problem



- A local minimum with $h = 1$
-

Local beam search

- Keep track of *k* states rather than just one
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.