CS249 – ARTIFICIAL INTELLIGENCE- 1

5th Week Slide/Notes

- 2201AI22 Lokesh Singla
- 2201AI24 Umesh Chandra
- 2201AI25 Suresh
- 2201AI26 Nirjay
- 2201AI27 Pavan

HEURISTIC

- A heuristic is a function that determines how near a state is to the desired state. Heuristics functions vary depending on the problem and must be tailored to match that particular challenge. The majority of AI problems revolve around a large amount of information, data, and constraints, and the task is to find a way to reach the goal state. The heuristics function in this situation informs us of the proximity to the desired condition. The distance formula is an excellent option if one needed a heuristic function to assess how close a location in a two-dimensional space was to the objective point.

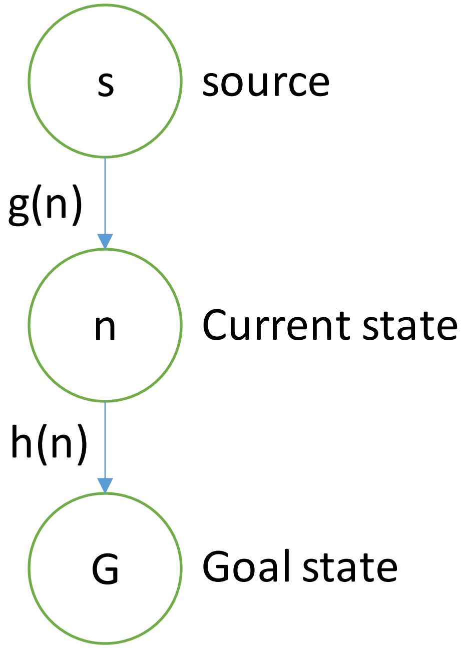
Heuristic search

- f(n) = g(n) + h(n)

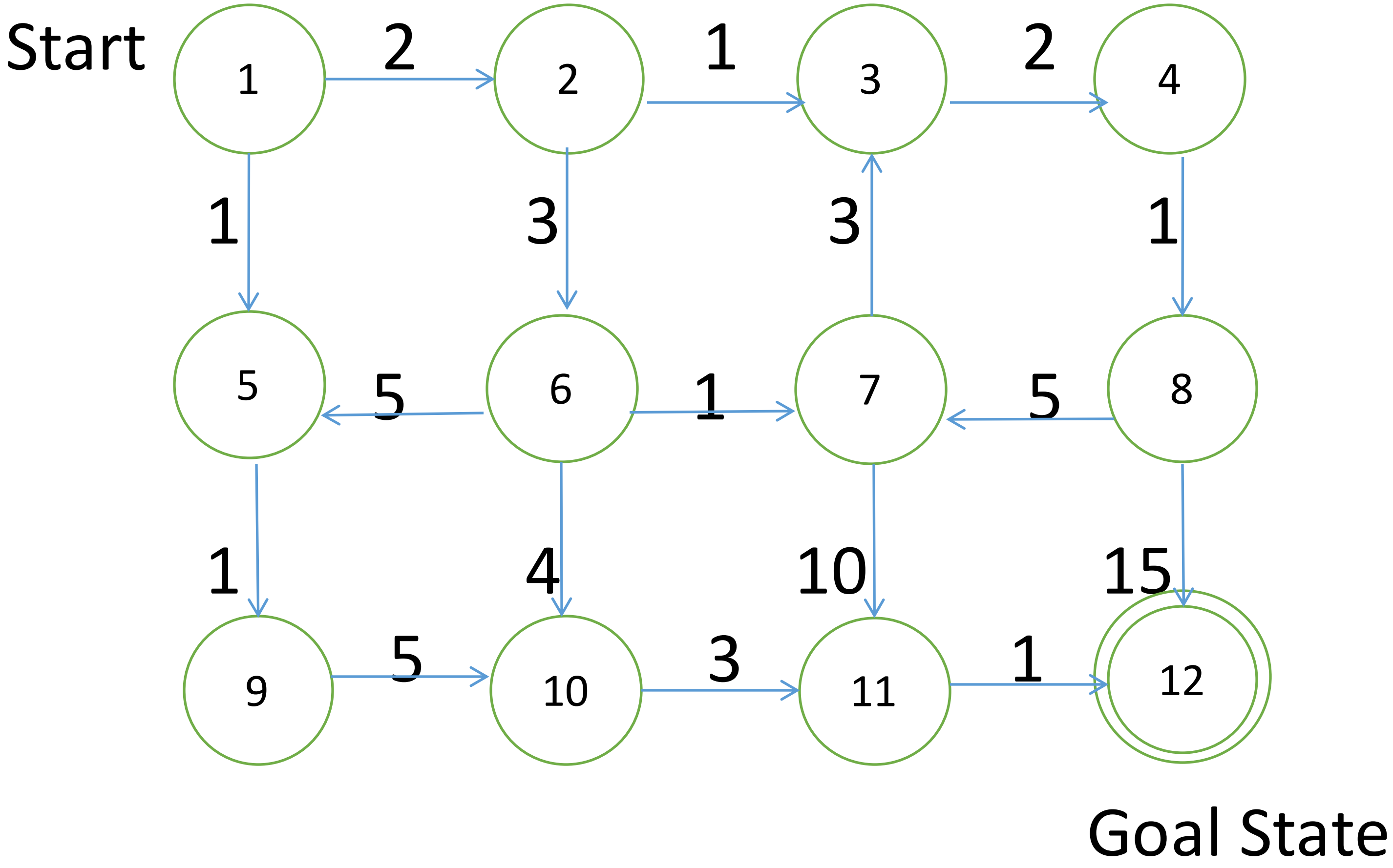
- g(n) = Actual Cost of n from s.
- h(n) = Estimated cost of n to reach G from n
- h*(n) = Optimal cost to reach G from n.

- h(n) <= h*(n) -> Underestimated
- h(n) > h*(n) -> Overestimated

- g(s) = 0 , where s is the start.
- h(G) = 0, where G is the Goal State.



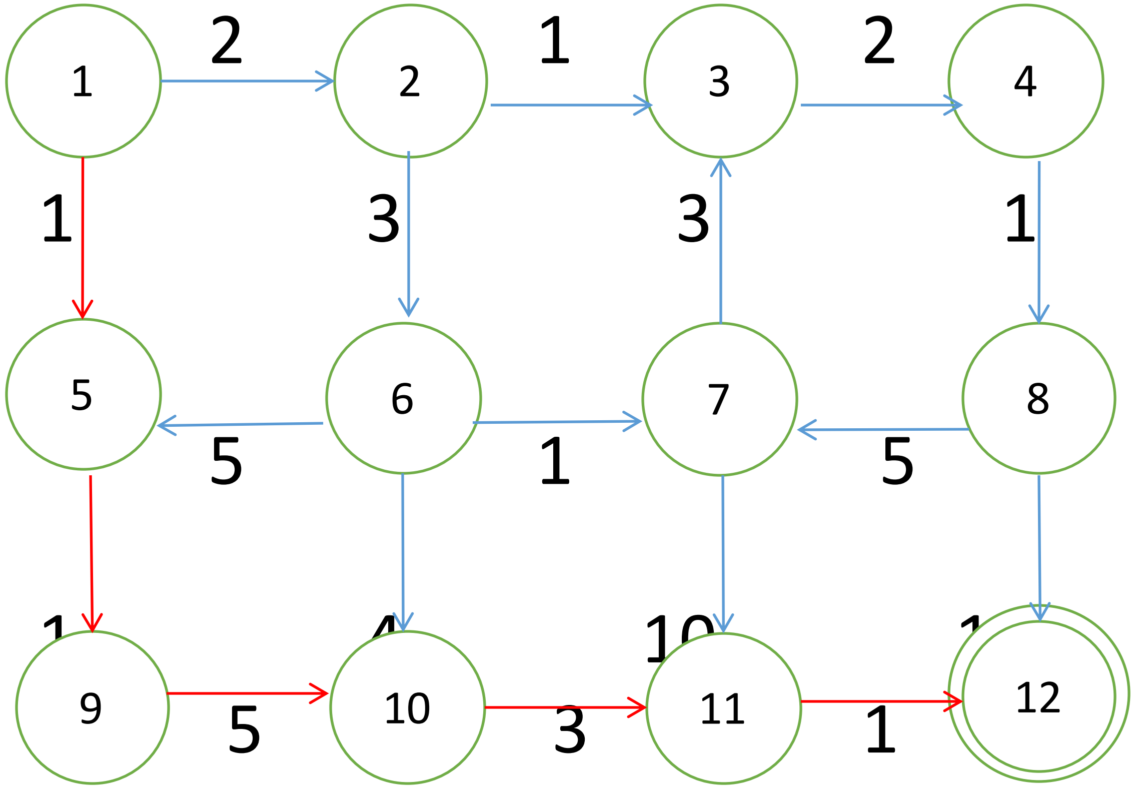
INFORMEDNESS:- h1 and h2 are two underestimated Heuristic . h1 is more informed than h2, if ∀ n belongs to S, h1(n) >= h2(n)



why h(1) =11 ?

Source

State.



Goal State

We know ,

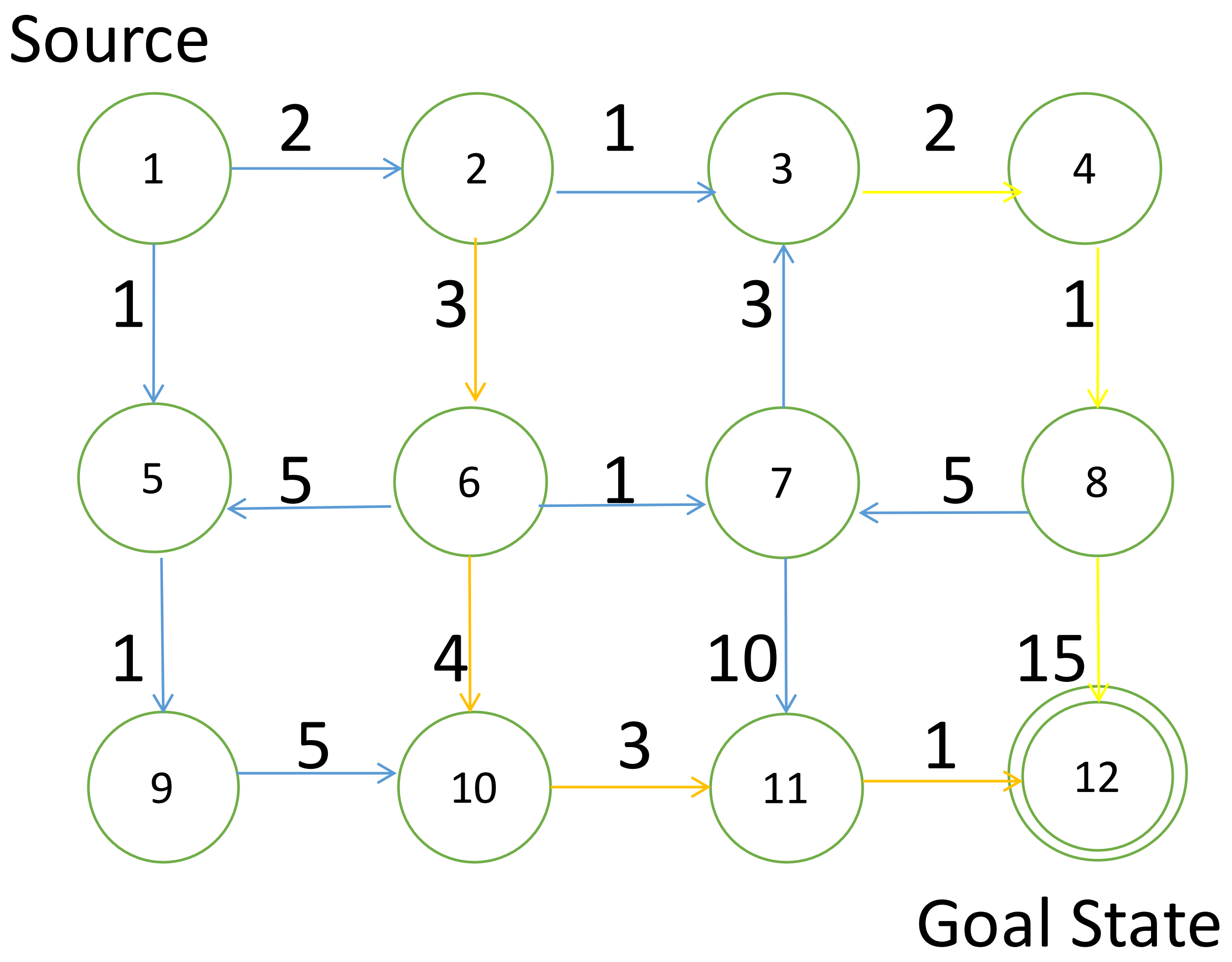
g(s) = 0 , where s is the Source.

h(G) = 0, where G is the Goal

g(1)= 0 , h(12) = 0

h(1) = 1+1+5+3+1=11

Similary find h(2),h(3),....h(12)



h(2) = 3+4+3+1 =11

h(3) = 2+1+15 = 18

h(4) = 1+15 =16

h(5) = 1 +5+3+1 =10

h(6) = 8

h(7) =11

h(8) =15

h(9) = 9

h(10) = 4

h(11) = 1 , h(12) = 0

- If ∀ n , <u>h(n) = h*(n)</u> then only the nodes in the <u>optimal path</u> will be expanded(Retried from open).

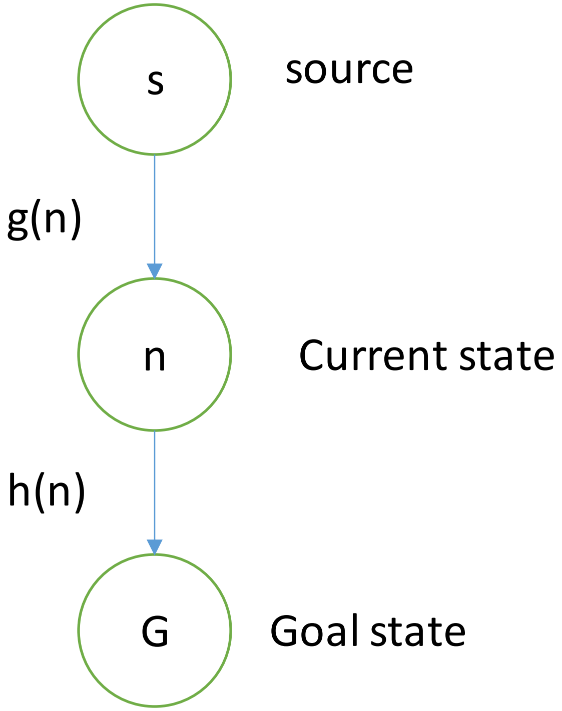
- f(n) =g(n) + h(n)
- f(1) =g(1) + h(1) = 0+11 =11
- f(2) =g(2) + h(2) = 2+11 =13
- f(3) =g(3) + h(3) = 3+18 =21
- f(4) =g(4) + h(4) = 5+16 =21
- f(5) =g(5) + h(5) = 1+10 =11
- f(6) =g(6) + h(6) = 5+8 =13
- f(7) =g(7) + h(7) = 7+11 =18
- f(8) =g(8) + h(8) = 6+15 =21
- f(9) =g(9) + h(9) = 2+9 =11
- f(10) =g(10) + h(10) = 7+4 =11
- f(11) =g(11) + h(11) = 10+1 =11
- f(12) =g(12) + h(12) = 11+0 =11

If we Underestimate

- When the heuristic is underestimate i.e , $h (n) <= h^*(n)$.
- Given k underestimated heuristic, drive the best heuristic
- h1,h2,h3.....hk => k
- Best ∀n max (h1,h2,h3........hk)
- suppose
- h1(n1) =5 | h1(n2) =6
- h2(n1) =7 | h2(n2) =4
- max {5,7} =7 max{6,4} = 6
- If h(n) =0
- f(n) =g(n) +0 =>g(n) (UCS algoarithm)

A*Algorithm

- One of the most important advances in AI
- Idea: avoid expanding paths that are already expensive
- g(n) = least cost path to n from S found so far
- h(n) <= h*(n) where h*(n) is the actual cost of optimal path to G(node to be found) from n.



A* ALGORITHM

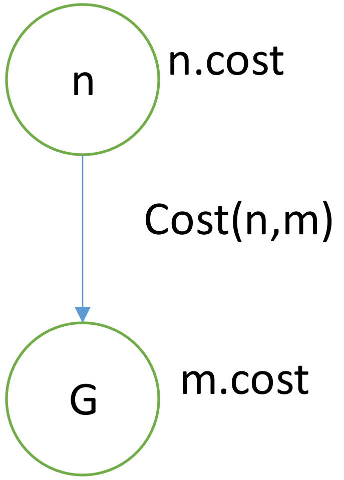
Initialization:

- open = [s]
- closed =[]
- s.cost = h(s) not Ø

Iteration :- Remove the lowest cost based on f(n) state n from open.

Selection :-

$$\left\{ \begin{array}{c} f(m) = g(m) + h(m) \\ g(m) = g(n) + Cost\ (n,m) \\ g(m) > g(n)+Cost(n,m) \end{array} \right\}$$



- $h(n) = 0$ => UCS (Many nodes will be expanded)
- $h(n) = h^*(n)$ => Minimum nodes will be expanded(all the nodes in the optimal will be expanded)

- A* uses heuristic search with underestimated heuristic.It always gives optimal solution.
- Q) Which nodes are being Expanded in UCS, but not in A* ?

UCS	A*	
$\forall n \quad g(n) <= g^*(y)$	$\forall n \quad f(n) <= g^*(y)$	$(g^*(y) = f^*(n))$
$c(n) <= c^*(n)$	$f(n) = g(n) + h(n)$	

-
- $\forall n \ g(n) < \begin{pmatrix} g^*(y) \\ f^*(y) \\ \ \end{pmatrix} < f(n)$

- These nodes are being expanded in ucs but not in A*

In A*, take priority Queue(Q) based on f(n)

In UCS, take priority Queue(Q) based on g(n)

Hill-climbing search

- Hill climbing algorithm is a **local search algorithm** which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value

- It is also called **greedy local search** as it only looks to its good immediate neighbor state and not beyond that.

- A node of hill climbing algorithm has two components which are **state** and **value**.

- Hill Climbing is mostly used when a good heuristic is available.

Features of Hill Climbing:

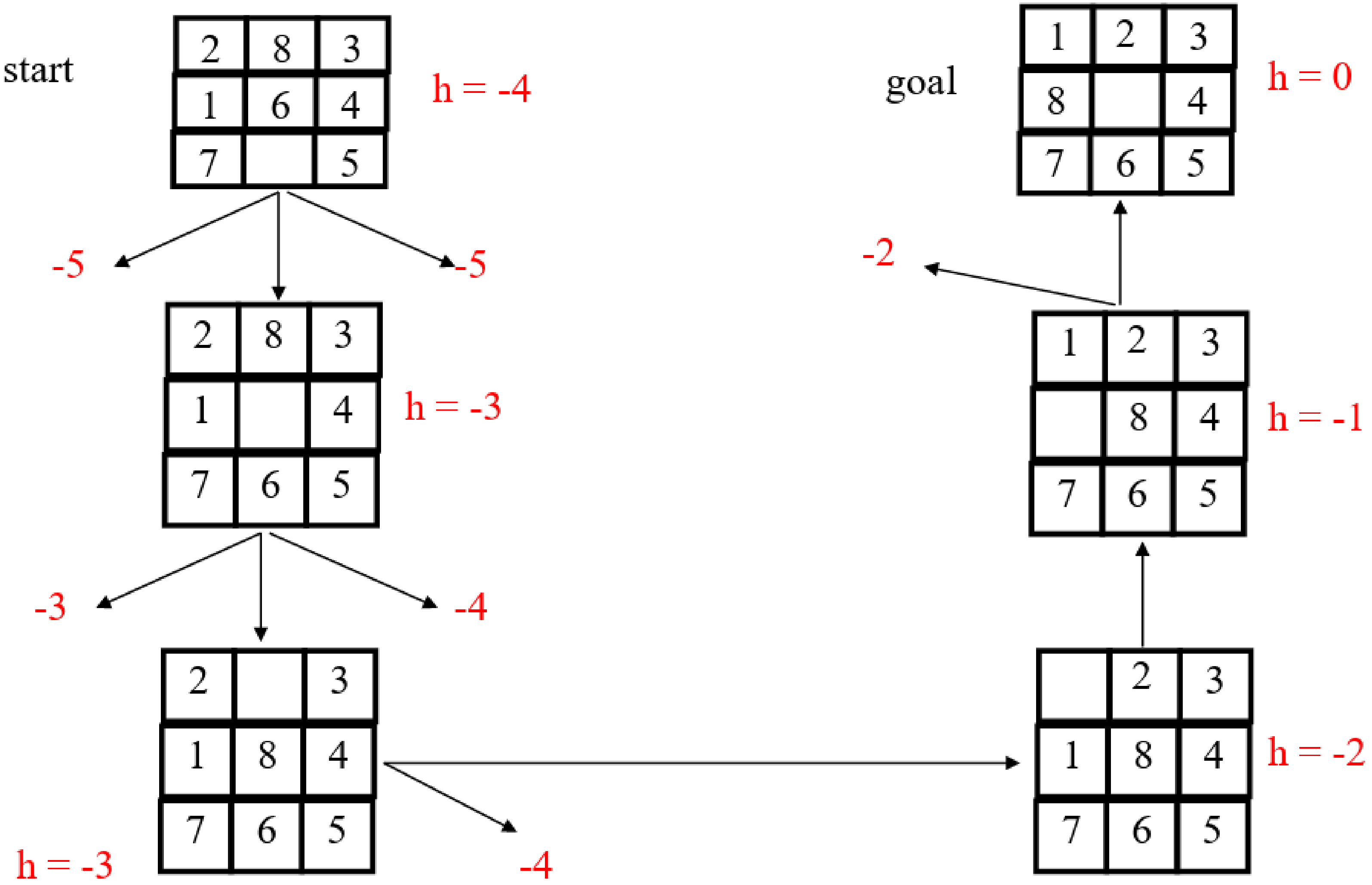
Following are some main features of Hill Climbing Algorithm:

- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.

- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.

- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

Hill Climbing Example



2	8	3
1	6	4
7		5

2	8	3
1		4
7	6	5

2		3
1	8	4
7	6	5

	2	3
1	8	4
7	6	5

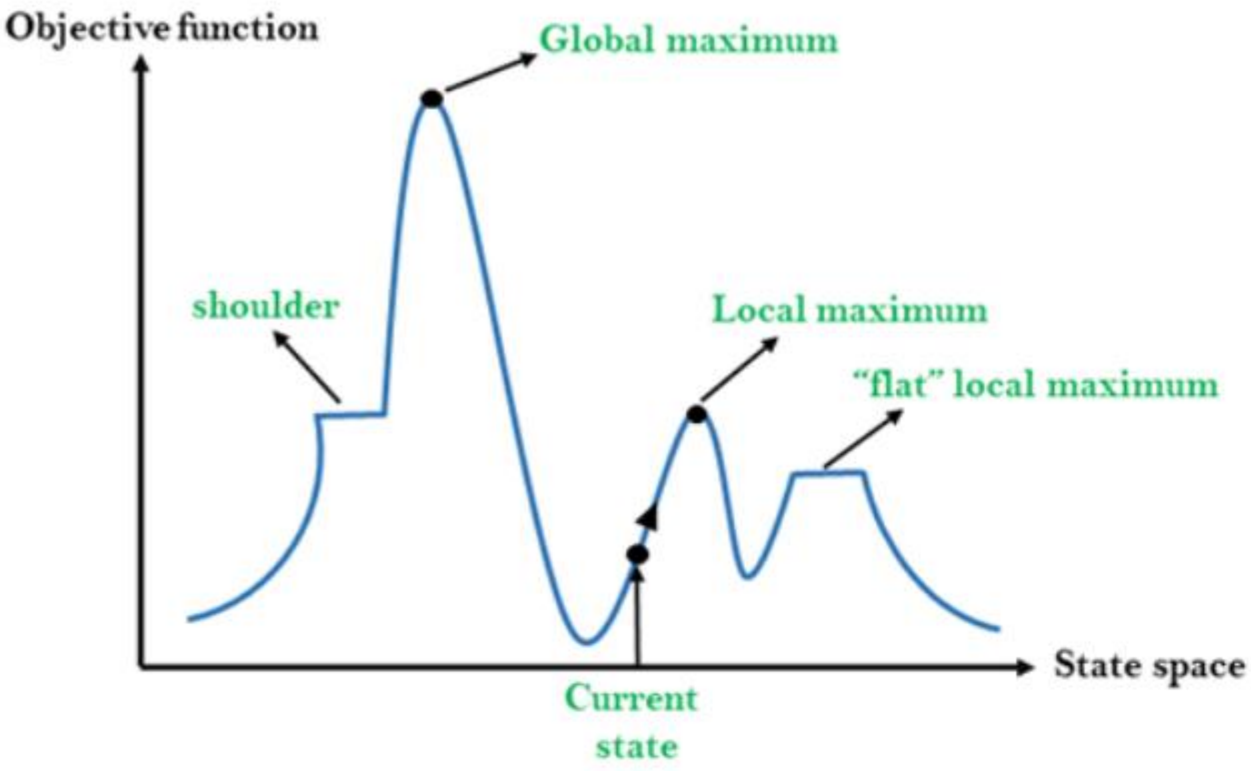
1	2	3
	8	4
7	6	5

1	2	3
8		4
7	6	5

f(n) = -(number of tiles out of place)

State-space Diagram for Hill Climbing:

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.



Y-axis - Cost Function
X-axis – State space

Types of Hill Climbing Algorithm :

- Steepest-Ascent hill-climbing
- Stochastic hill Climbing
- Random-restart hill-climbing
- Simple hill Climbing

Hill-climbing variations

Stochastic hill-climbing

- Random selection among the uphill moves
- Selection probability can vary with the steepness of the uphill move (exploration continues if better than the current)
- Converges slowly than steepest ascent (best first search) but finds better solution in some cases .

First-choice hill-climbing

- Stochastic hill climbing by generating successors randomly until a better one is found
- Good strategy when a state has many successors

Random-restart hill-climbing

- Tries to avoid getting stuck in local maxima.
- If at first you don’t succeed, try, try again…

1.) Steepest Ascent Hill Climbing :

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.

- **Step 2:** Loop until a solution is found or the current state does not change.

    a.) Let SUCC be a state such that any successor of the current state will be better than it.

    b.) For each operator that applies to the current state:

        a.) Apply the new operator and generate a new state.
        b.) Evaluate the new state.
        c.) If it is goal state, then return it and quit, else compare it to the SUCC.
        d.) If it is better than SUCC, then set new state as SUCC.
        e.) If the SUCC is better than the current state, then set current state to SUCC.

- **Step 5:** Exit.

2.) Stochastic hill Climbing

Stochastic hill climbing does not examine
for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

3.) Random restart hill climbing

Conducts a series of hill-climbing searches from randomly generated initial states and stop when goal is found.

- Tries to avoid getting stuck in local maxima.
- If at first you don’t succeed, try, try again…

If there are few local maxima and plateaus, it will find solutions very quickly! (for three million queens solutions can be obtained within a minute)

Drawbacks of hill climbing

- **Local Maxima**
  - Peaks that aren’t the highest point in the space (below global maxima)
  - Higher than each of its neighboring states

- **Plateau: region where evaluation function is flat**
  - Flat local maximum: no uphill exit exists
  - Shoulder: possible to make progress
  - Could give the search algorithm no direction (random walk) for local maximum

Drawbacks of hill climbing

- **Ridges:**
  - Flat like a plateau, but with drop-offs to the sides; steps to the North, East, South and West may go down, but a combination of two steps (e.g. N, W) may go up
  - Results in a sequence of local maximum not connected to each other
- **For 8-queens problem**
  - Hill climbing gets stuck for the 86% problem instances
  - Solves only 14% instances
  - Average # steps for successful moves: 4
  - Average # steps for unsuccessful moves: 3
  - Acceptable solution for a 17-million states

Simulated annealing

- Pure random walk: choosing successor from a list is complete but inefficient
- Hill climbing that does not make downhill move is incomplete

Solution: hill climbing + random walk = simulated annealing to ensure both efficiency and completeness

Analogy :

- Slowly cool down a heated solid, so that all particles arrange in the ground energy state.
- At each temperature wait until the solid reaches its thermal equilibrium.
- Probability of being in a state with energy E :

$$Pr \{ E = E \} = 1/Z(T) \,.\, \exp(-E / k_B.T)$$

E	Energy
T	Temperature
k_B	Boltzmann constant
$Z(T)$	Normalization factor (temperature dependent)

Simulation of cooling (Metropolis 1953)

At a fixed temperature T :

- Perturb (randomly) the current state to a new state
- Δ E is the difference in energy between new and current state
- If Δ E < 0 (new state is lower), accept new state as current state
- If ΔE ⍰ 0 , accept new state with probability

  $$\text{Pr (accepted)} = \exp\ (- \Delta E\ /\ k_B.T)$$

- Eventually the systems evolves into thermal equilibrium at temperature T

- When equilibrium is reached, temperature T can be lowered and the process can be repeated

Simulated Annealing

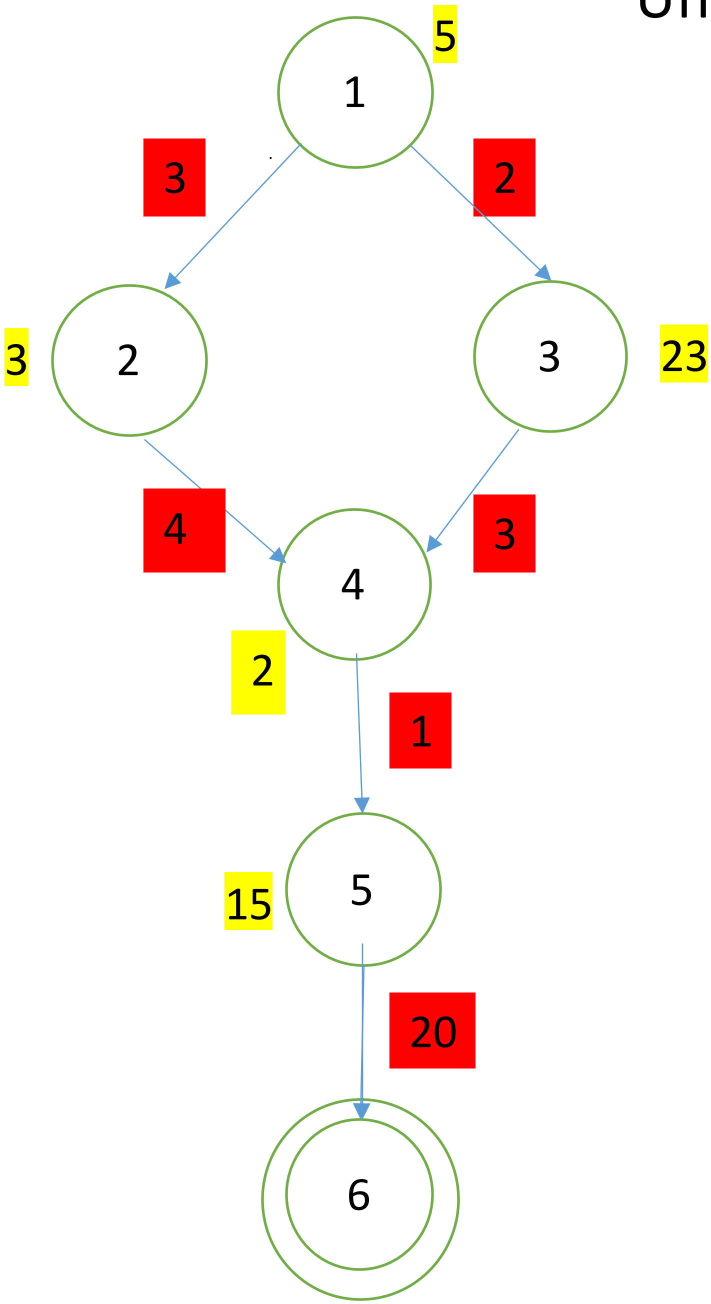
Same algorithm can be used for combinatorial optimization problems:

Energy E corresponds to the Cost function C
Temperature T corresponds to control parameter c

$$\Pr \{ \text{configuration} = i \} = 1/Q(c) \,.\, \exp(-C(i) / c)$$

C	:	Cost
C	:	Control parameter
Q(c)	:	Normalization factor (not important)

Underestimates Heuristic performs A* steps



OPEN

1(5)

2(7),3(25)

3(25),4(5)

3(25),5(11)

3(25),6(28)

4(7),6(28)

5(9),6(28)

6(26)

CLOSED

1(5),2(7)4(9),5(11),3(26),4(7),5(4),6(28)

Few points on A*

1. A heuristic is always admissible if it always underestimate, i.e . We always have h(n) <=f*(n), where f*(n) denotes the minimum distance to a goal state from state n.

for finite state spaces, A* always terminates.

2.At any time before A*terminates there exists in open, a state n that is an optimal path from s to a goal state with f(n)<f*(s).

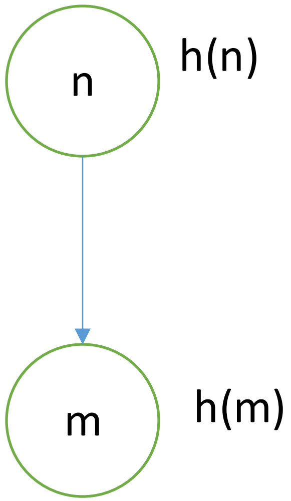
If there is a path from s to a goal state, A* terminate(even the state space is infinite).

3. Algo A* is admissible, i.e if there is a path from s to a goal state, A* terminate by finding an optimal path.

If we are given two or more admissible heuristic, we can take their max to a stronger admissible then.

.

4. A* heuristic, if monotoneand admissible, then we don’t need to transfer any node from closed to open.
but if not monotone, then we may need to transfer.



CONSISTENT/MONOTONE

For every node n ,h(n) <=c(n,m)+h(m)

1.If the monotone restriction is satisfied,then A* has already found an optimal path to the state at selects for expansion.
2.If the monotone restriction is satisfied, f values of the state expanded by A* is non decreasing.

.

PATH MAX:-

Convert a non monotonic heuristic to monotone heuristic

1. During generation g successor m of n,

   Set h’(m) = max[h(m), h(n)-c(n,m)] and use h(m) at m.

admissible underestimate --> optimal
we may reach to optimal from an over estimate but it is not admissible.

CONTRIBUTIONS

• 2201AI22 Lokesh Singla	20%
• 2201AI24 Umesh Chandra	20%
• 2201AI25 Suresh	20%
• 2201AI26 Nirjay	20%
• 2201AI27 Pavan	20%