LU-6: Informed Search

Objectives

- To explain various informed search strategies

Outcomes

- Solve problem using Informed Search strategies

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

Review: Tree search

- A search strategy is defined by picking the order of node expansion

Best-first search

- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"

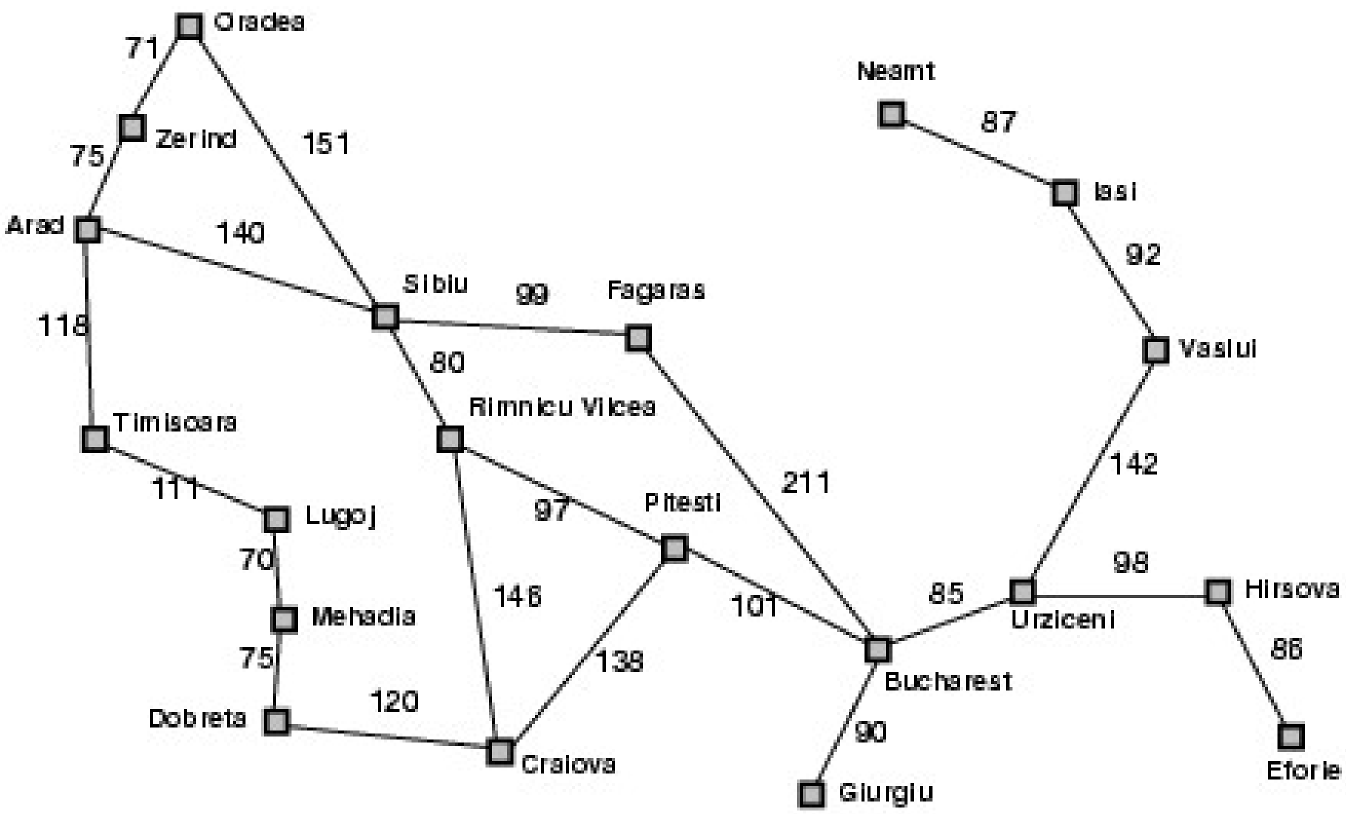
  → Expand most desirable unexpanded node

- Implementation:

  Order the nodes in fringe in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A^* search

Romania with step costs in km



Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

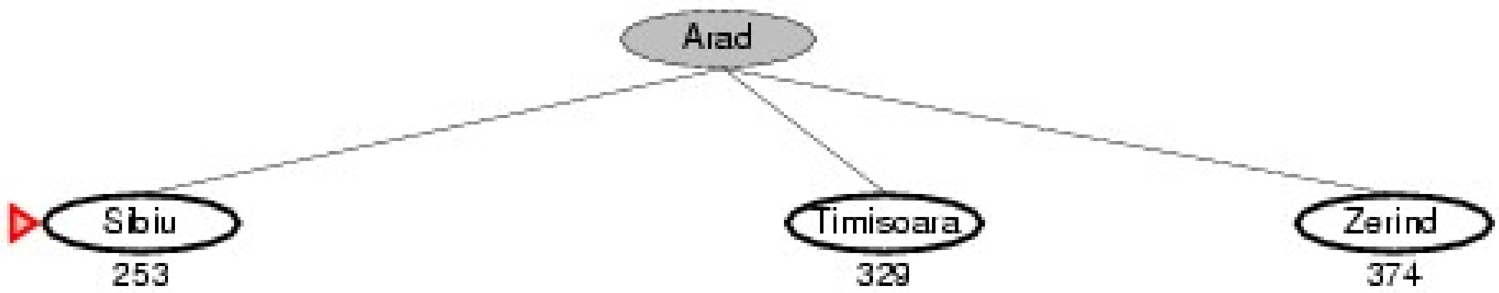
Greedy best-first search

- Evaluation function $f(n) = h(n)$ (heuristic)
- = estimate of cost from n to *goal*
-
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
-
- Greedy best-first search expands the node that appears to be closest to goal
-

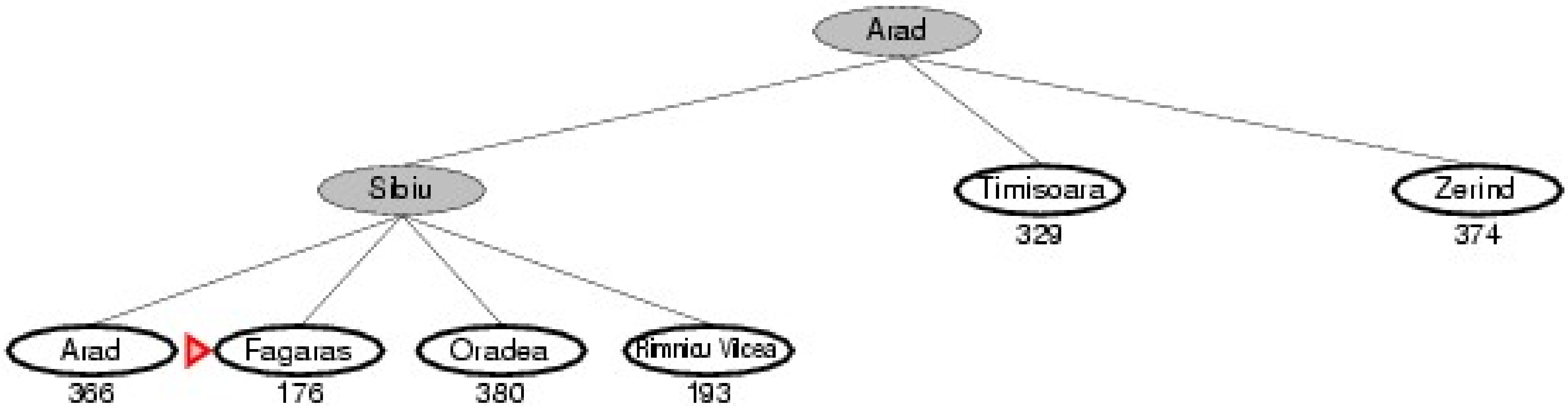
Greedy best-first search example

Heuristic Value – Calculated Guess value

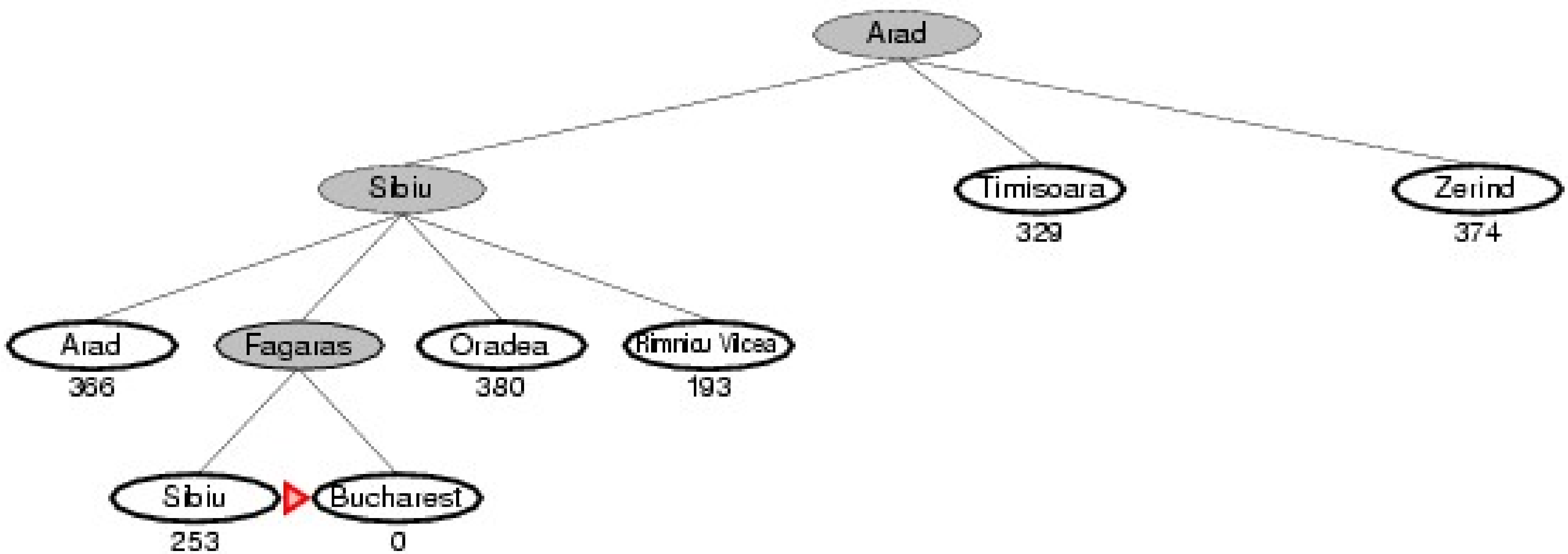
Greedy best-first search example



Greedy best-first search example



Greedy best-first search example



Properties of greedy best-first search

- Complete? No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ -- keeps all nodes in memory
- Optimal? No

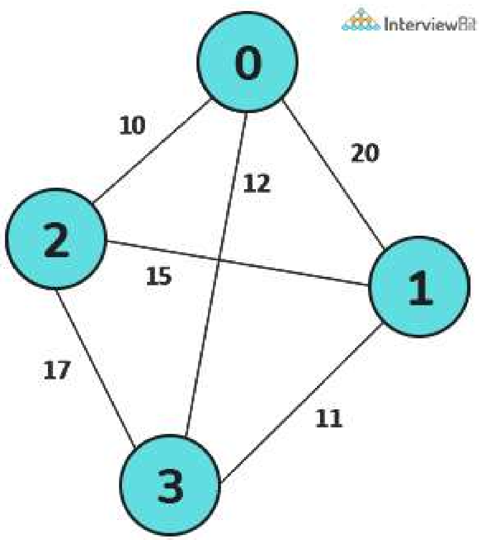
Heuristic Function

Heuristic Function

- Additional information about each node
- Technique to solve a problem quickly

In case of game developing or problem solving, always need to think about the path which leads to final solution.
A problem solving technique is **ARTIFICIAL INTELLIGENCE**
i.e. , Making a machine to think like a human to choose the path which might lead to the optimal solution.

An ADDITIONAL INFORMATION given to each path is the HEURISTIC FUNCTION. **Purpose**: Out of the possible solutions, it helps to ***converge quickly*** at the final solution.



Travelling Sales Person Problem

0 – Source
3 – Destination

From 0 – multiple possibilities are there to make a move. But preferable move is based on only the **low cost distance**. This additional information code may be taken as heuristic function.

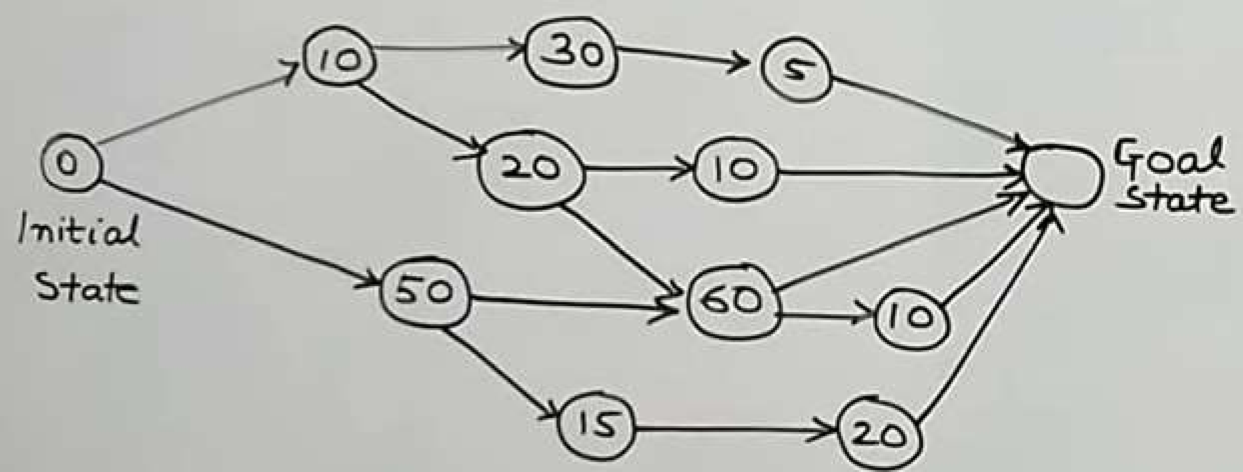
Heuristic Search Techniques

- https://www.youtube.com/watch?v=GL28VbiFkD0

Heuristic Search Techniques

Heuristic Search Techniques...

- problem solving method that uses shortcuts/calculated guess to provide good enough solutions.
- reduces time complexity to reach solution
- may not give best/optimal solution



Heuristic Search Techniques

- Significant one in Artificial Intelligence is DECISION MAKING
- Heuristic Search technique helps AI agent to make decisions.
- Hence it makes AI algorithms much more efficient

- AI Algorithms are different from TRADITIONAL algorithms
  - Traditional Algor: Merge Sort
    - repeatedly divides the array into two halves until we reach a stage where we try to perform MergeSort on a subarray of size 1
    - After that, the merge function comes into play and combines the sorted arrays into larger arrays until the whole array is merged.
    - Simple searching algor: Linear search can be applied in array
    - When apply this traditional algot., definitely gets the solution

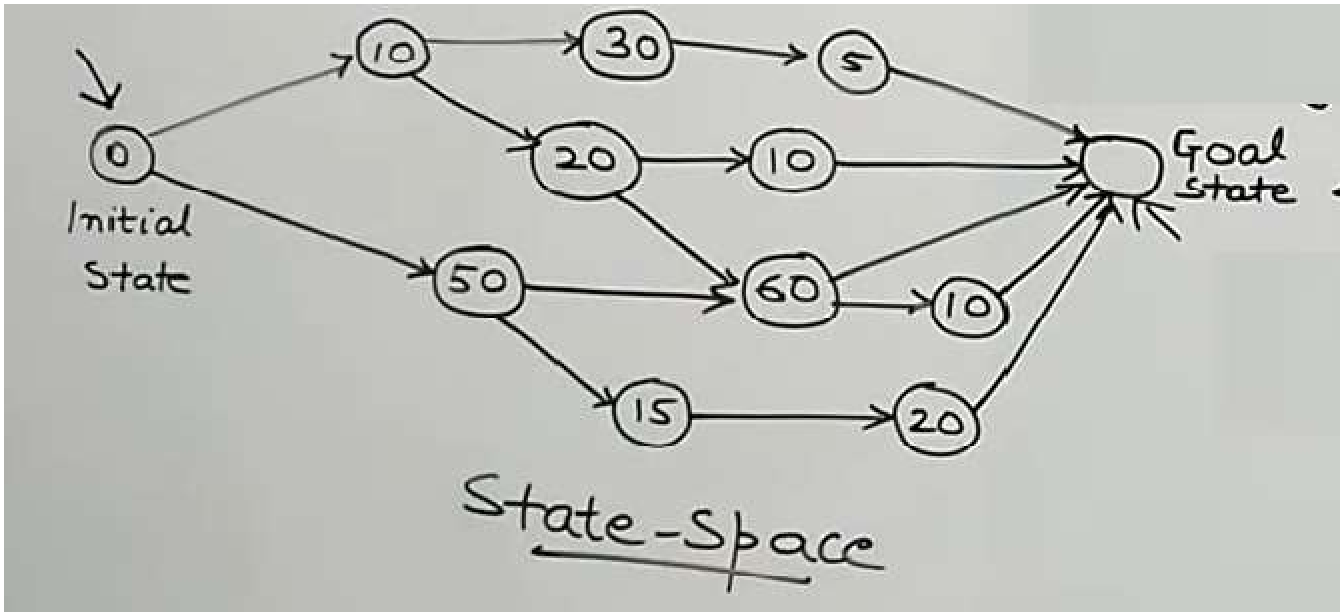
Heuristic Search Techniques

- AI Algorithms are different from TRADITIONAL algorithms
  - How AI algorithms are different from this Traditional algorithms?
    - Not follows the same rule in Traditional algorithms.
    - But APPLY DECISION MAKING

    - Let us learn how this DECISION MAKING is important to heuristic.
      - Also, learn, how these heuristics are used to make AI algorithms optimized.

    - **NOTE**: AI algorithms optimize the TIME COMPLEXITY by DECISION MAKING

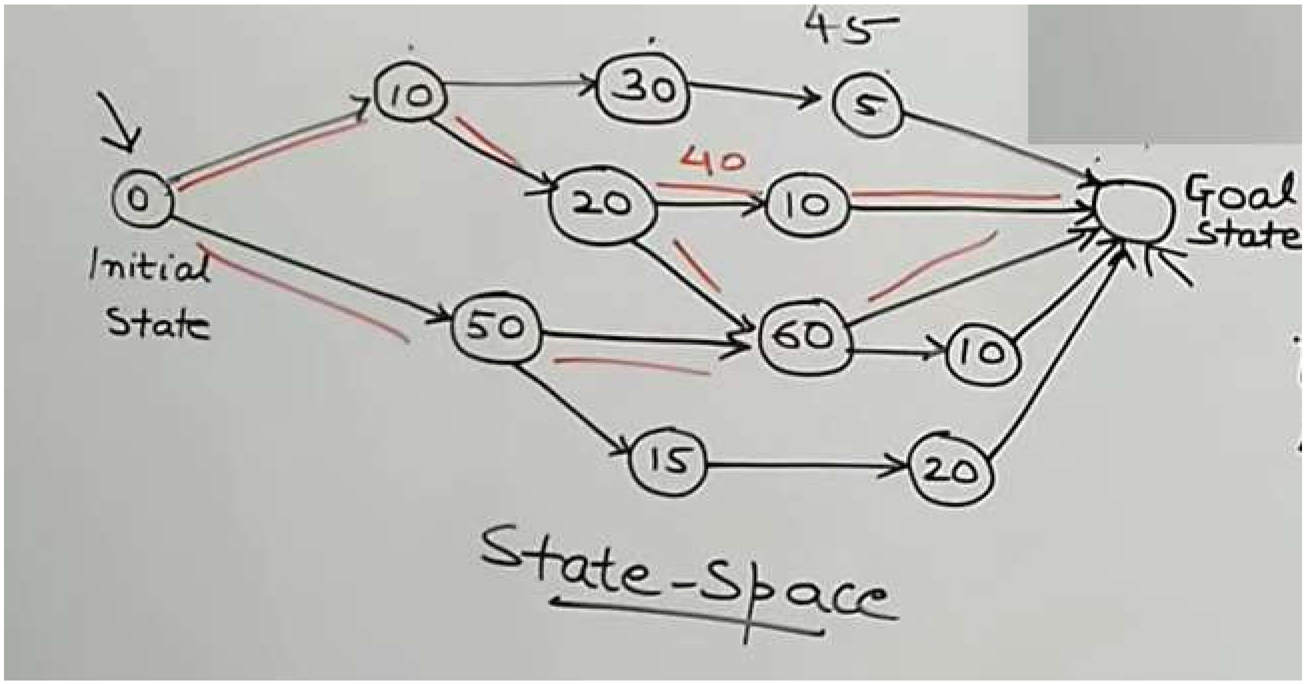
Heuristic Search Techniques



Number 0, 10, 30, … represents the time taken in seconds to reach a state to another state

Traditional Idea: Explore all paths and find the time taken by adding the time mentioned inside each node

Heuristic Search Techniques

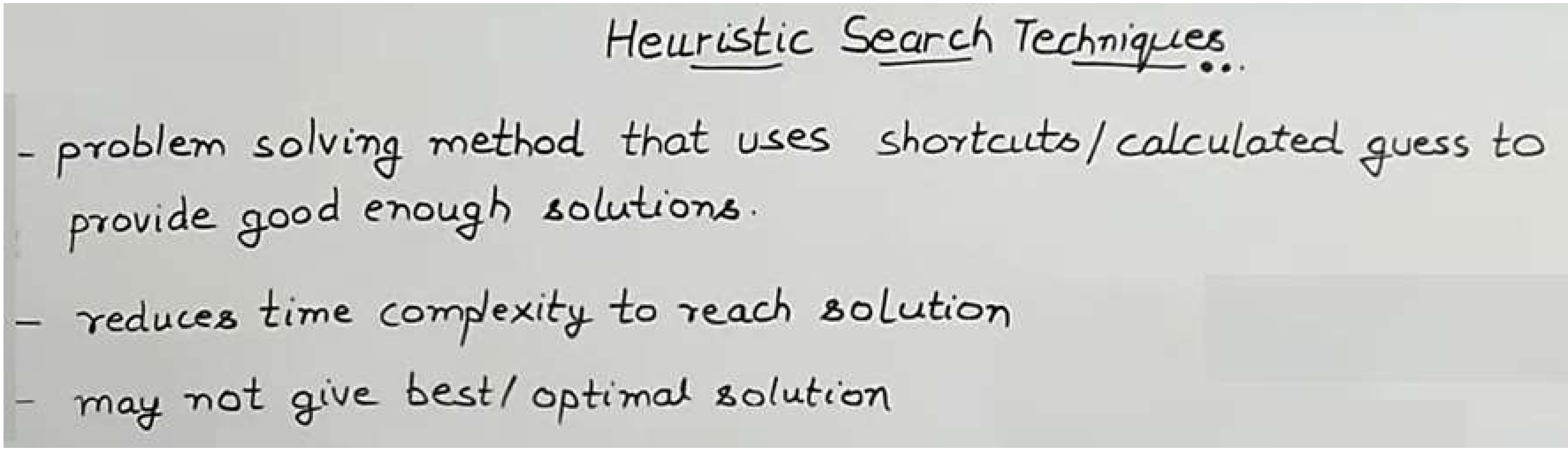


Number 0, 10, 30, … represents the time taken to reach a state to another state

Traditional Idea: Explore all paths and find the time taken by adding the time mentioned inside each node

After exploring the time taken in all paths, we will find out which one will be the **LEAST.** **Least** Time Taken will be the optimal solution of this problem.

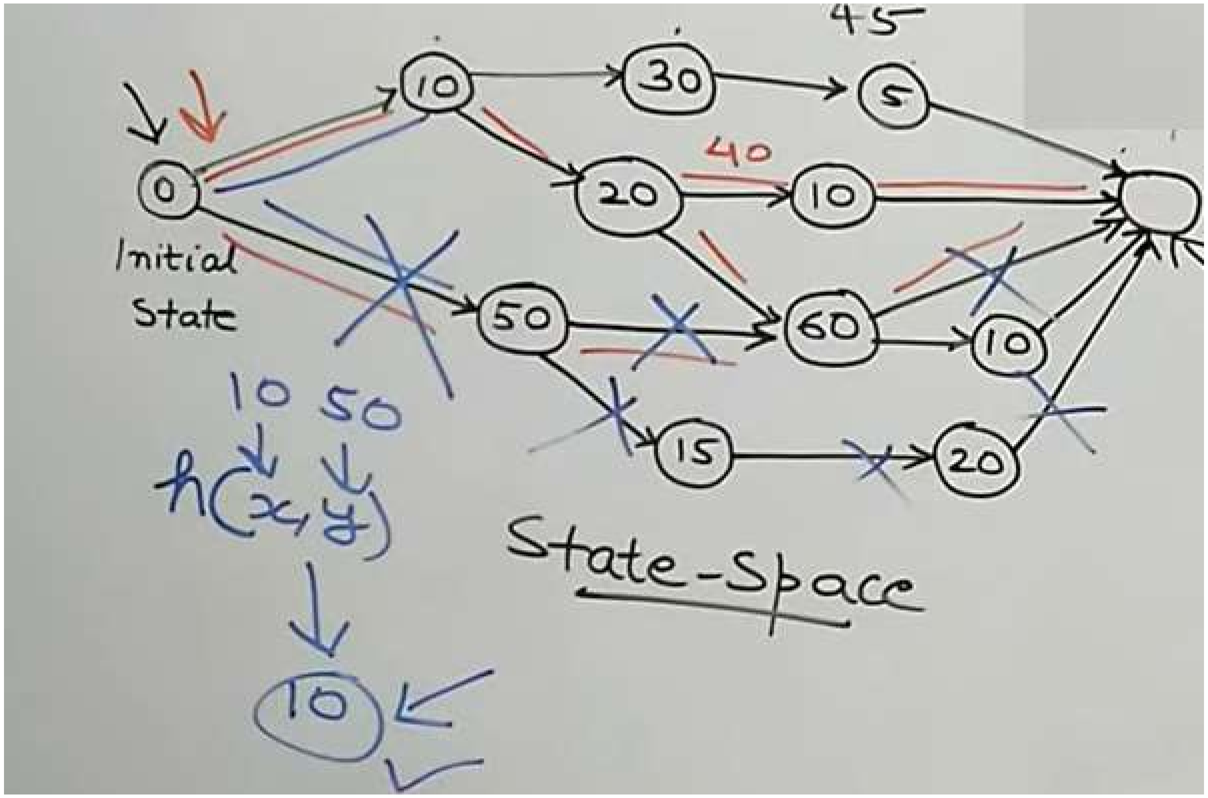
Why heuristics is introduced?


Heuristic Search Techniques

- problem solving method that uses shortcuts/calculated guess to provide good enough solutions.
- reduces time complexity to reach solution
- may not give best/optimal solution

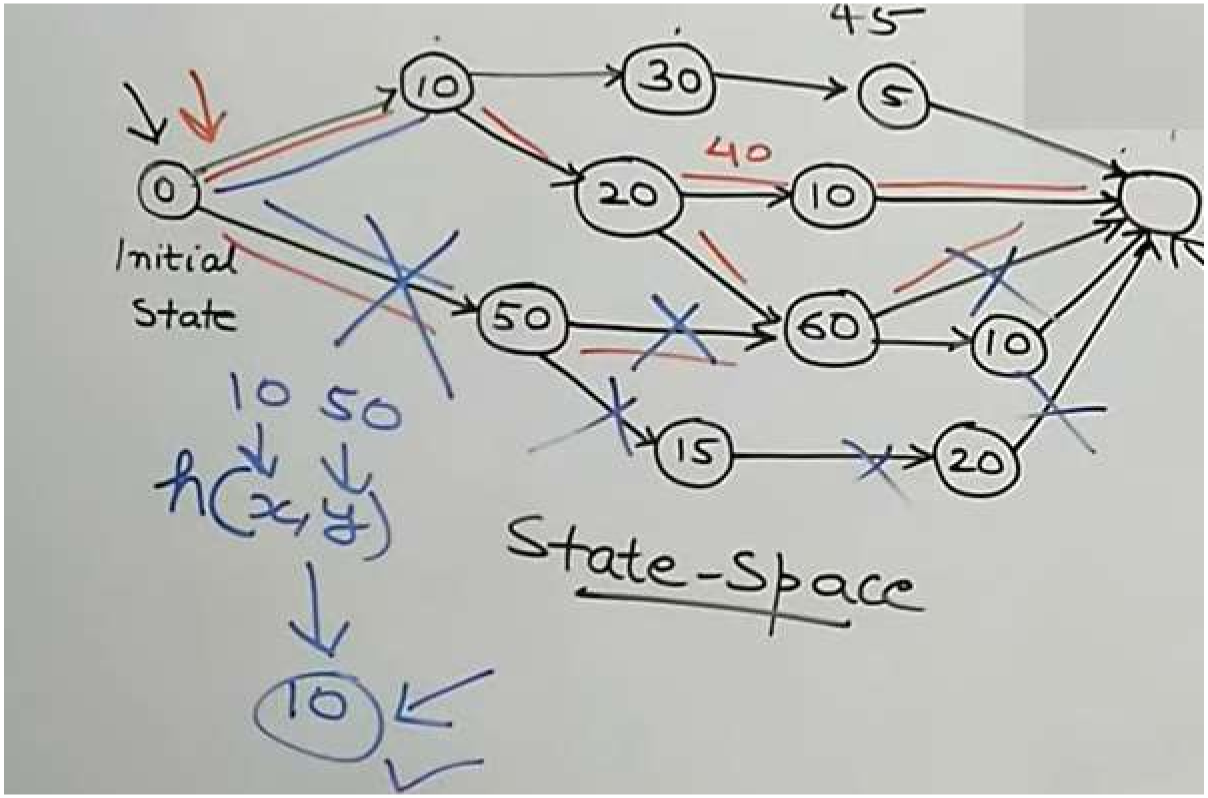
- Taking the guess from the Past experiences
- Making Decisions by not exploring those paths which are not optimal

Why heuristics is introduced?



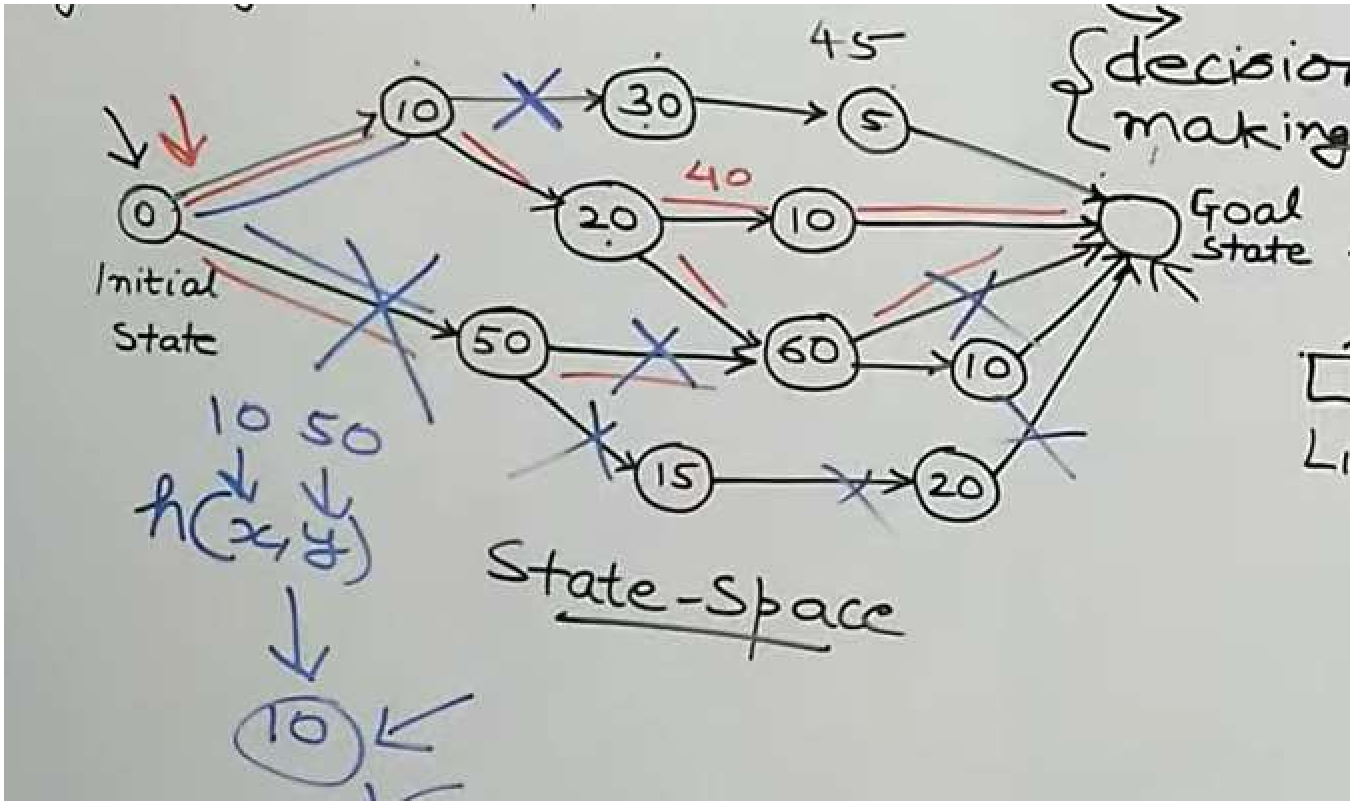
- Two Paths : 0 to 10 and 0 to 50
- Choose the path which has minimum value
- As 10 is minimum value, let us not explore the path 0 to 50
  - When 0 to 50 is not explored, the paths which are following it are also not explored

Why heuristics is introduced?



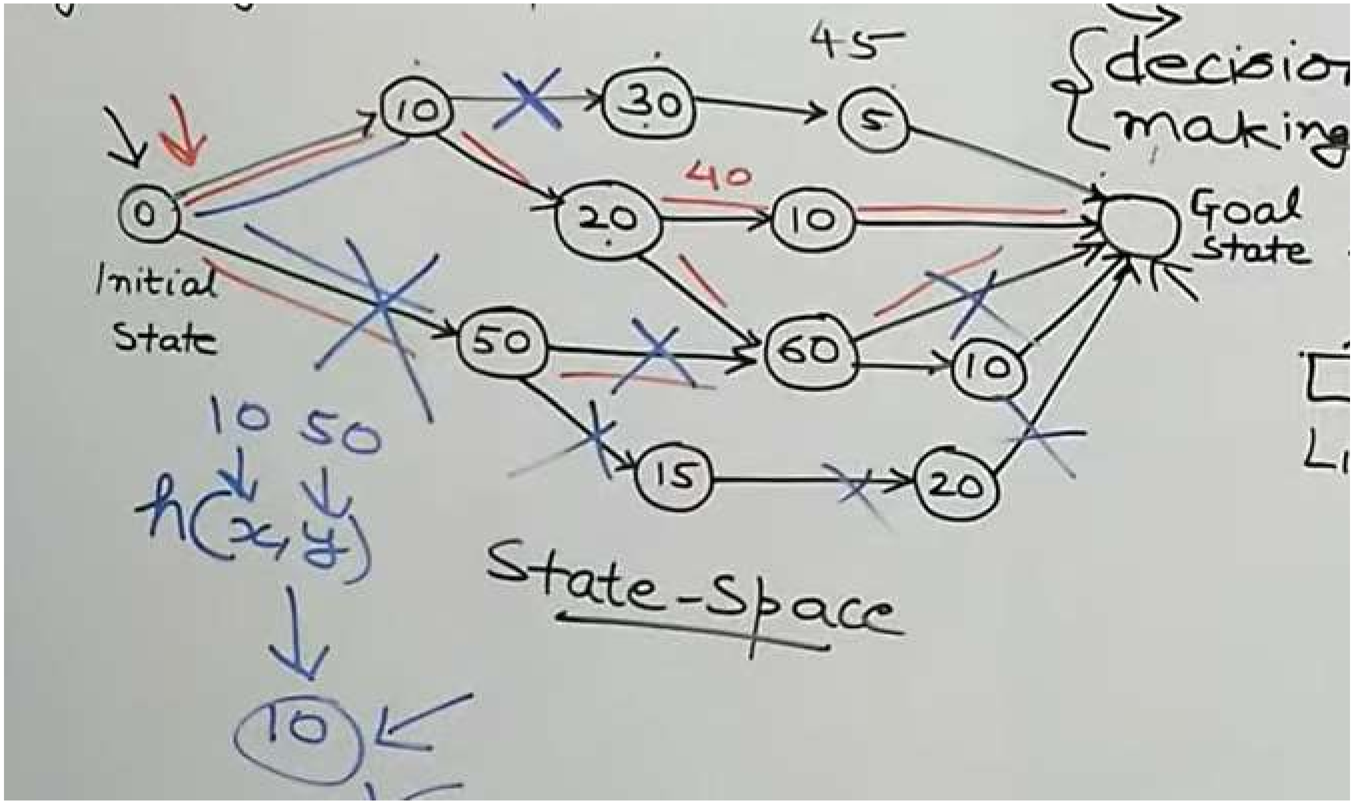
- Decision Making: h(10,50) = 10
  - Use this Heurstic function (strategy) to make this calculated guess, ignore the paths and explore only 10

Why heuristics is introduced?



- Similarly, From 10, h(30,20) = 20
- Repeat the process until and unless the goal is reached

Why heuristics is introduced?



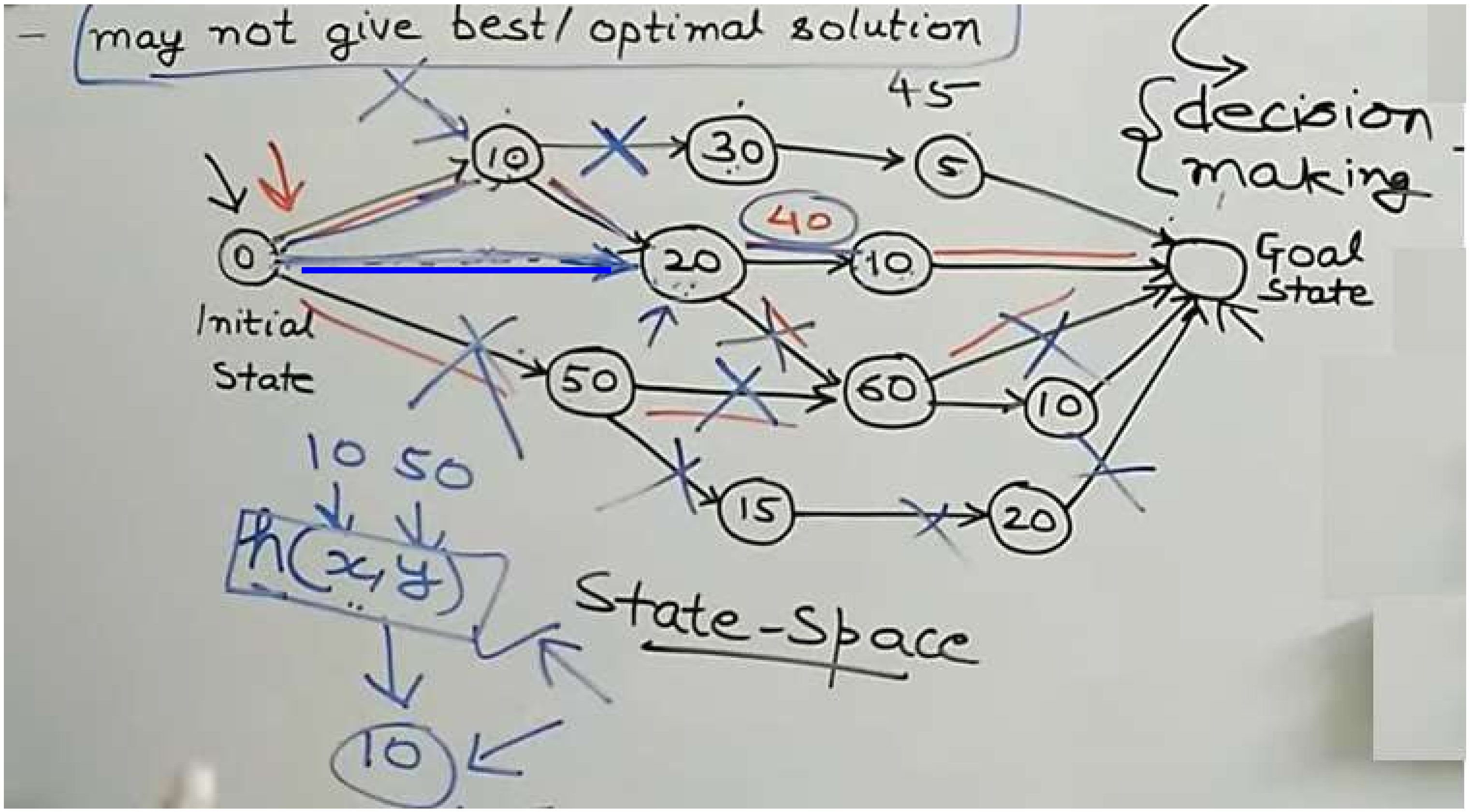
- Using the calculated Guess work to explore the path which can lead to an optimal solution
- On ignoring the paths, AI agents **do not need to traverse those paths** which will automatically reduce the TIME COMPEXITY to reach the solution

Why heuristics is introduced?

- Heuristic Search Techniques gives the ability to the AI agent to make the decision.
- Heuristic functions vary from problems to problems.
- NOTE: Actually giving decision making power to AI agent not to proceed onto a particular path since it has the greater value and traverse through the RIGHT path
- FINAL PATH sequence obtained using HEURISTIC SEARCH Technique will use the minimum value when the paths are exploring and ignoring the paths with greater value
- RESULT: 1) OPTIMAL solution

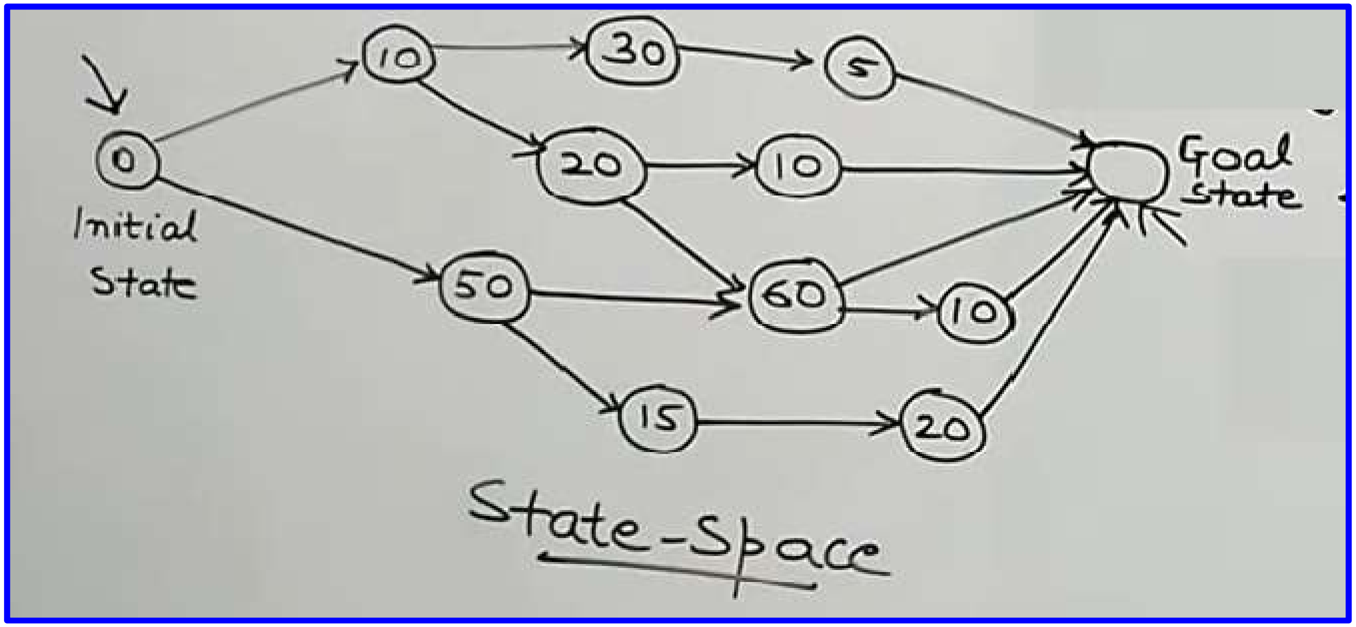
  2) Time Complexity is reduced

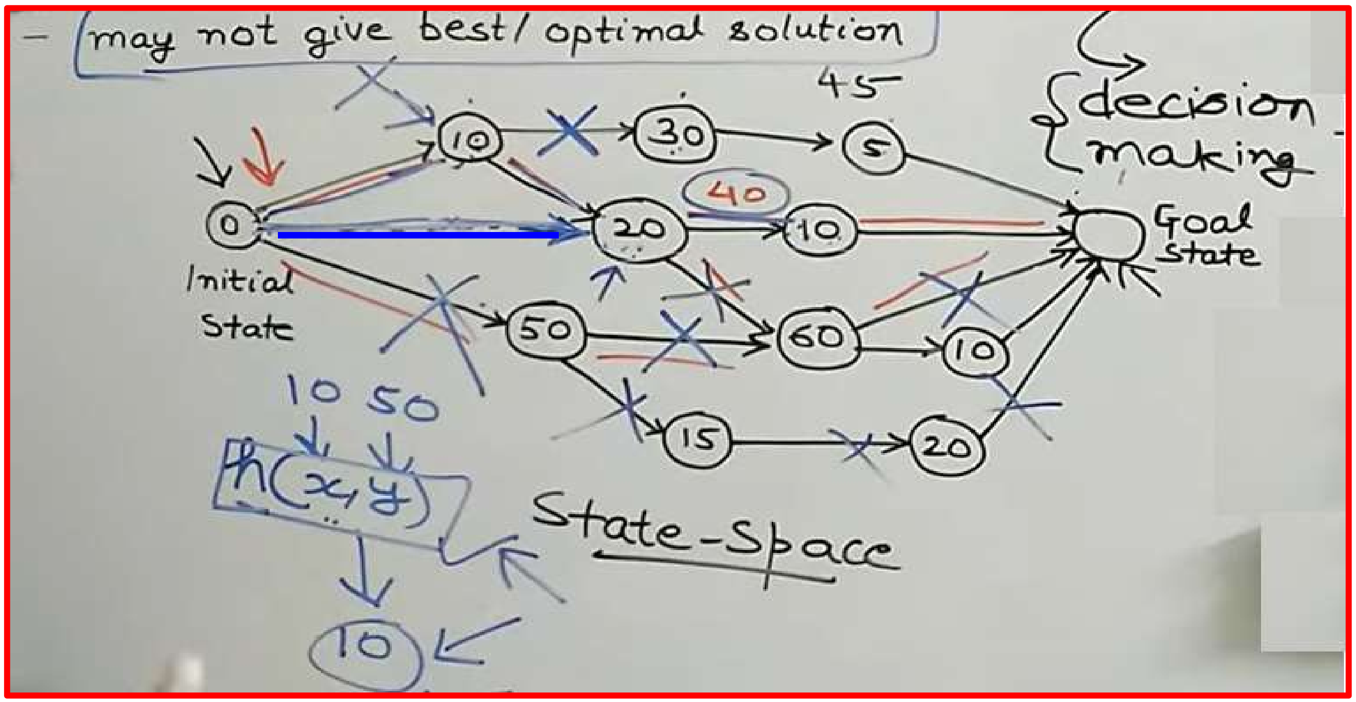
Making Changes in the graph



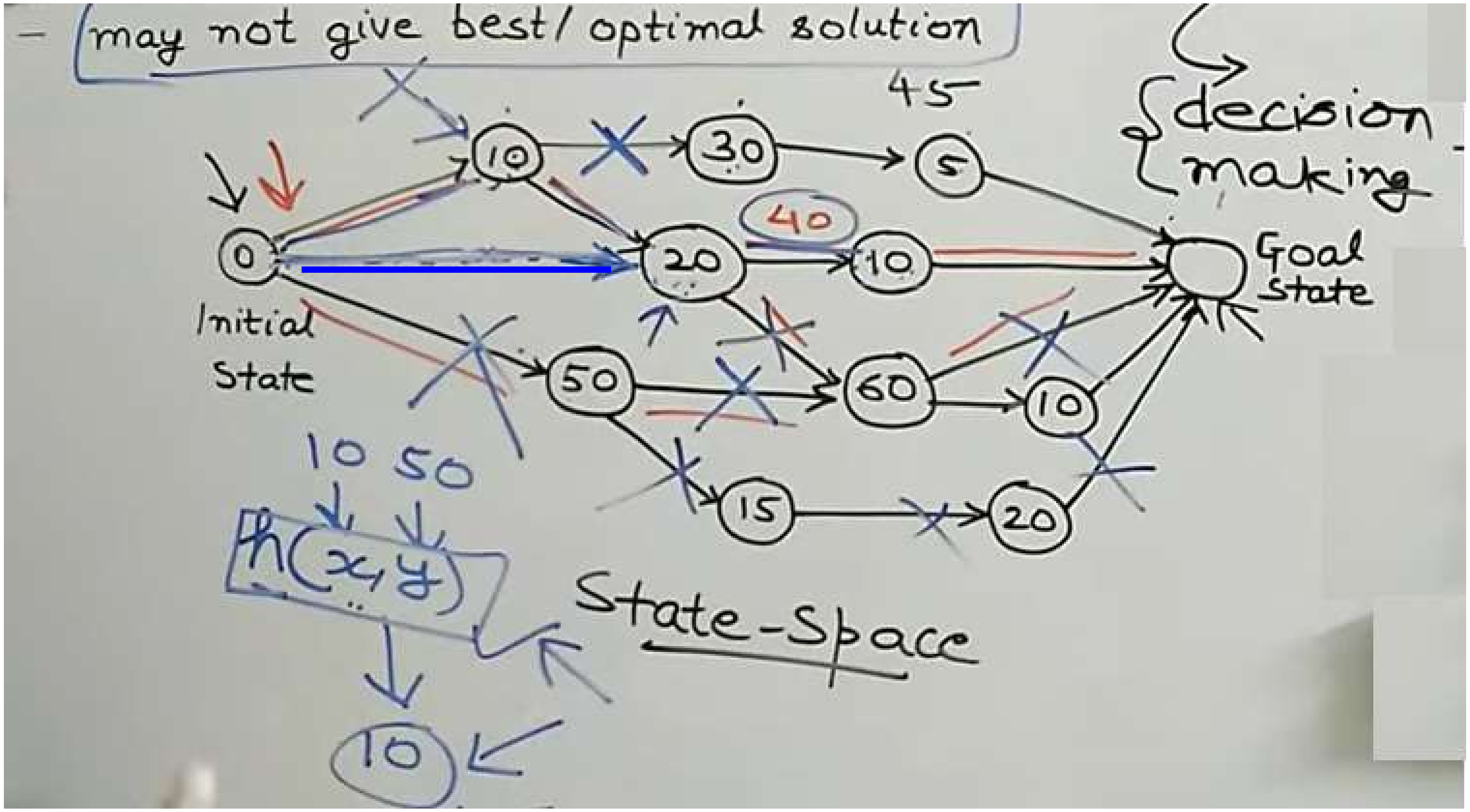
0 -> 10 -> 20 -> 10 -> Goal :   40 seconds

Establishing a new path between 0 to 20 :
0 -> 20 -> 10 -> Goal :   30 seconds


10
30
5
0
20
10
Goal state
Initial State
50
60
10
15
20
State-Space


– may not give best/optimal solution
45
decision making
10
30
5
0
40
20
10
Goal state
Initial State
50
60
10
15
20
10 50
h(x,y)
10
State-Space

Making Changes in the graph



0 -> 10 -> 20 -> 10 -> Goal : **40 seconds**

Establishing a new path between 0 to 20 :
0 -> 20 -> 10 -> Goal : **30 seconds**

Due to the present heuristic, ignoring path 0 to 20 and considering 0 to 10 will **not give the optimal solution.**

Tech 1: Eucledian distance

Heuristic Function

- Additional information about each node
- Technique to solve a problem quickly

- Techniques :
  - Eucledian Distance $\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$

Tech 2: Manhatten Function

Heuristic Function

Manhatten Function

1	4	5
6	8	3
2	7	

Current state

1	2	3
8		4
7	6	5

Goal state

$$H = 0 + 3 + 1 + 2 + 2 + 2 + 1 + 1$$

Single Agent / Player will play

Tech 2: Manhatten Function (cont..)

Heuristic Function

Manhatten Function

1	4	5
6	8	3
2	7	

Current state

1	2	3
8		4
7	6	5

Goal state

H = 8 (number of misplaced tiles)

Tic Tac Toe

O	X	O
		O
X		X

O's turn – 3 places to insert O

Difference

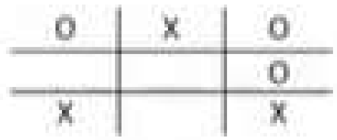
Calculate heuristic function : X's possibility to win – O's possibility to win

X's possibility to win -

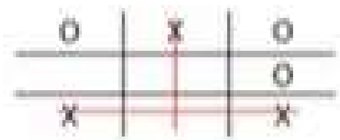
O's possibility to win -

Two Agents / Players will play

Tic Tac Toe



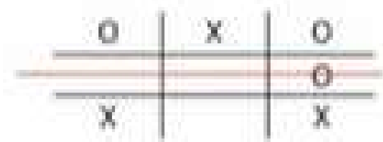
O's turn – 3 places to insert O



Calculate heuristic function

X's possibility to win - 2

O's possibility to win - 1



Heuristic Function h = 2-1=1

For the given input state, X has the more possibility to win. Such **extra** or **additional information** is said to be <u>Heuristic Function</u>

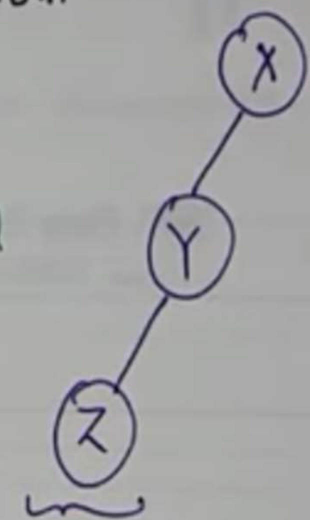
Heuristic Function

Heuristic Search:- Tries to Solve Problem in minimum Steps/Cost.

↳ Tries to optimize a problem using heuristic function. [Informed Search]

Heuristic Function: It is a function H(n), that gives an estimation on the cost of getting from node 'n' to the GOAL State. } estimated Value.

[Helps in Selecting optimal node for expansion.]



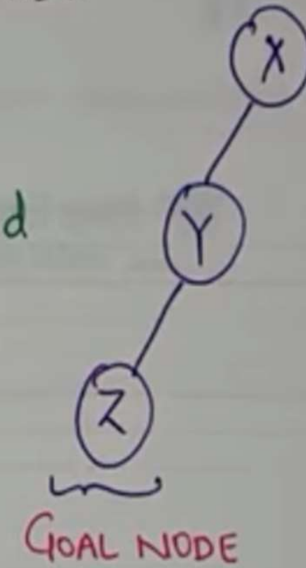
GOAL NODE

Heuristic Function

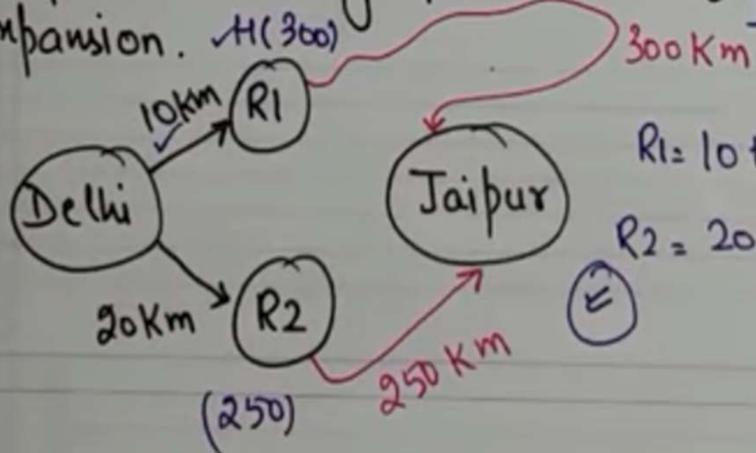
Heuristic Search:- Tries to Solve Problem in minimum Steps/Cost.

↳ Tries to optimize a problem using heuristic function. [Informed Search]

Heuristic Function: It is a function H(n), that gives an estimation on the Cost of getting from node 'n' to the GOAL State. } estimated Value.



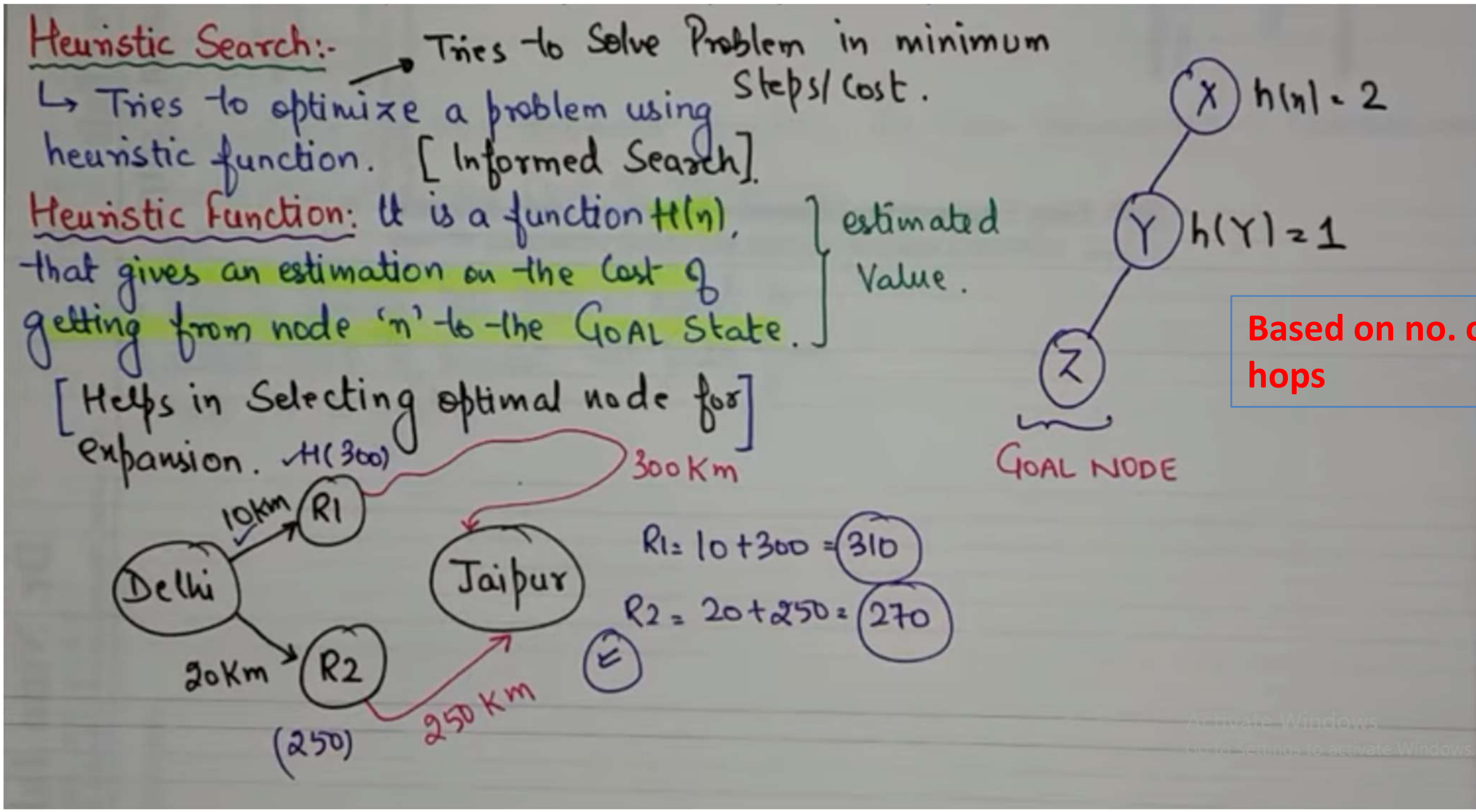
[Helps in Selecting optimal node for expansion.]



R1 = 10 + 300 = 310

R2 = 20 + 250 = 270

Heuristic Function



Based on no. of hops

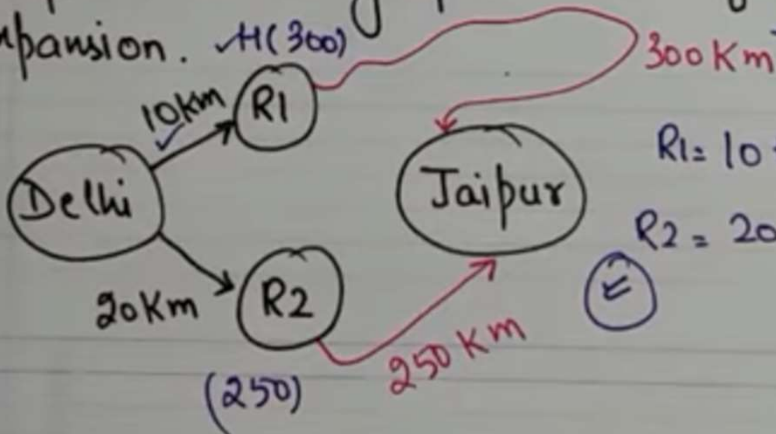
Heuristic Function

Heuristic Search:- Tries to Solve Problem in minimum Steps/Cost.

↳ Tries to optimize a problem using heuristic function. [Informed Search]

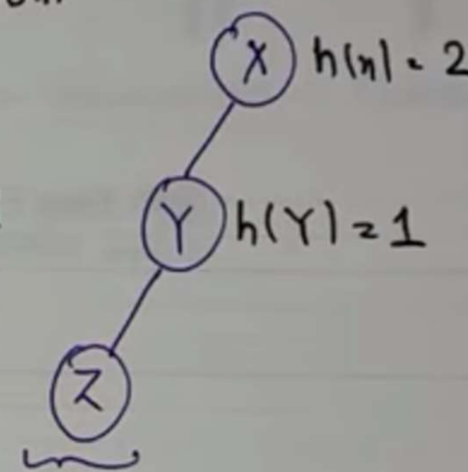
Heuristic Function: It is a function $H(n)$, that gives an estimation on the Cost of getting from node 'n' to the GOAL State. } estimated Value.

[Helps in Selecting optimal node for expansion.]



$R1 = 10 + 300 = 310$

$R2 = 20 + 250 = 270$



GOAL NODE

Admissible Heuristic

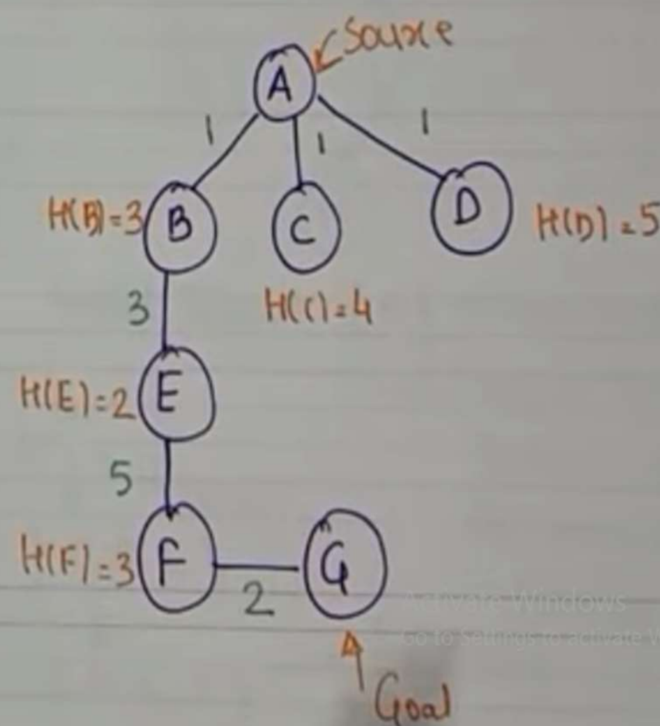
Types of Heuristic:- (underestimate)

↳ (i) Admissible: In this Heuristic function, never overestimates the cost of reaching the Goal. $H(n) \le H'(n)$ {Goal}

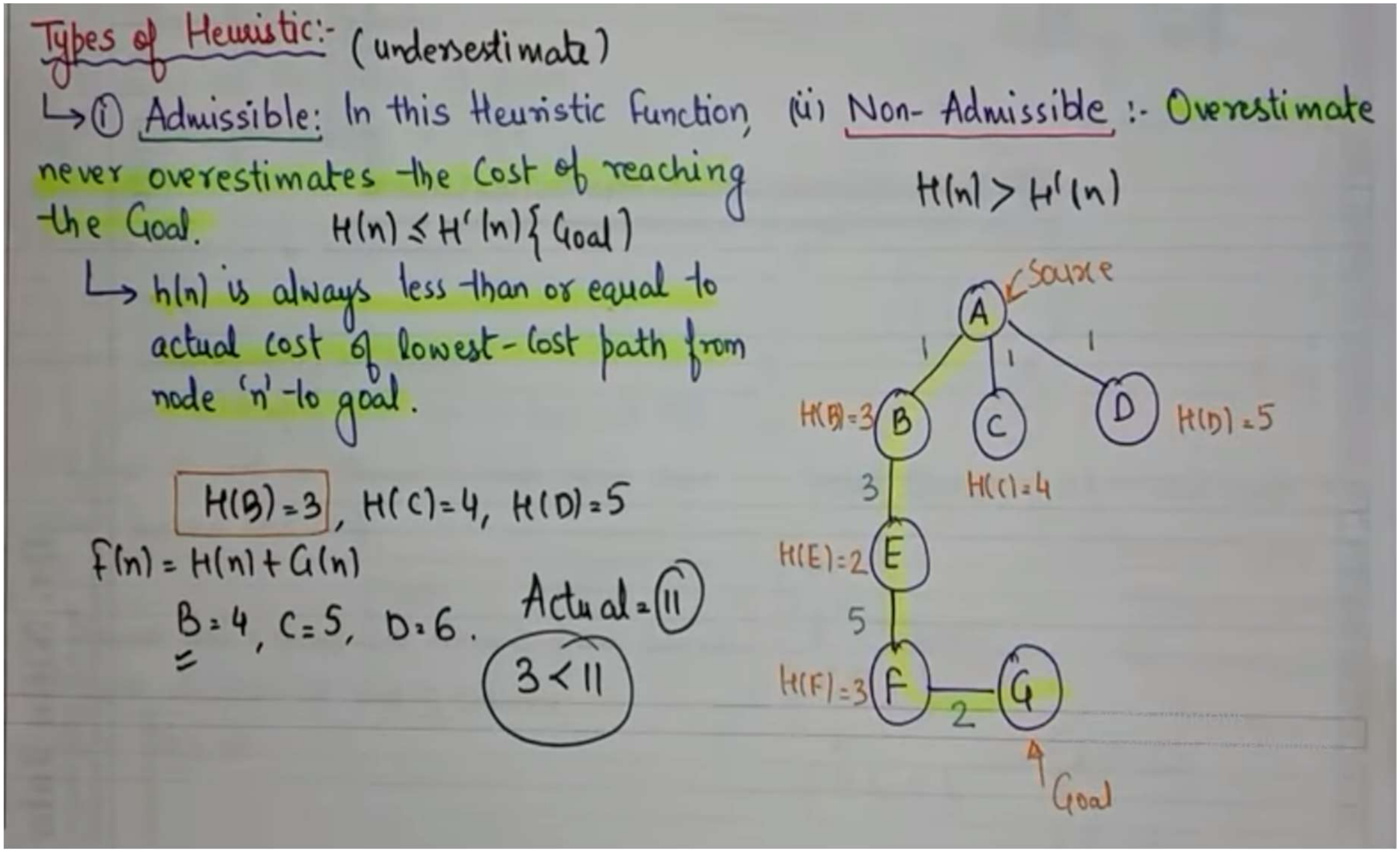
↳ h(n) is always less than or equal to actual cost of lowest-cost path from node 'n' to goal.

(ii) Non-Admissible :- Overestimate

$H(n) > H'(n)$



Admissible Heuristic (cont..)



With the help of F(n), agent decides the node to be further explored.
Among F(B)=4, F(C)=5, F(D)=6, Agent can explore only at B since F(B) is **minimum**

Non-Admissible Heuristic

Types of Heuristic:- (underestimate)

↳ (i) Admissible: In this Heuristic function never overestimates the cost of reaching the Goal. $H(n) \leq H'(n)$ {Goal}

↳ h(n) is always less than or equal to actual cost of lowest-cost path from node 'n' to goal.

$\boxed{H(B)=3}$, $H(C)=4$, $H(D)=5$

$f(n) = H(n) + G(n)$

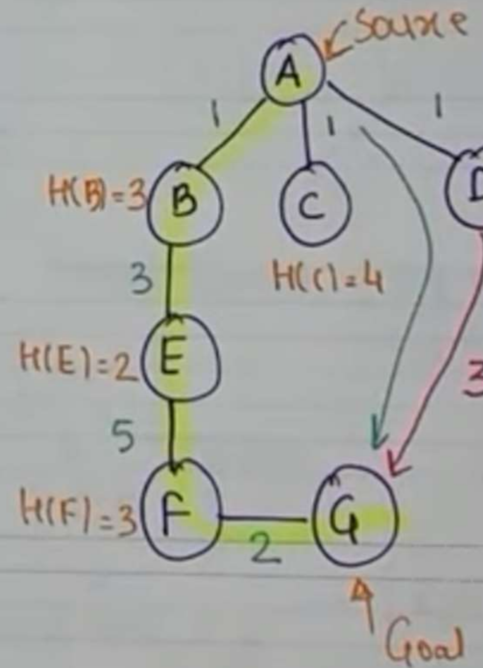
$B=4$, $C=5$, $D=6$.

Actual = 11

$3 < 11$

(ii) Non-Admissible :- Overestimate

$H(n) > H'(n)$



$\boxed{H(D)=5}$

$A \xrightarrow{1} D \xrightarrow{3} G$

4

$5 > 4$

Admissible heuristics

- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)
- Theorem: If $h(n)$ is admissible, A^* using `TREE-SEARCH` is optimal

A* search

- **Idea**: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal
-

2) A* Search Algorithm:

→ A* search Alg finds the shortest path through the search space using the heuristic function.

→ It uses h(n), & cost to reach the node n from the start st[ate]
g(n).

→ This alg expands less search tree and provides optimal resu[lt]
faster.

→ It is similar to UCS except that it uses g(n) + h(n) instead
of g(n).

- A^* use search heuristic as well as the cost to reach the node. Hence we combine both costs as,

$$\boxed{f(n) = g(n) + h(n)} \quad \{\text{fitness number}\}$$

$f(n)$ = Estimated cost of the cheapest soln

$g(n)$ = Cost to reach node n from start state

$h(n)$ = Cost to reach from node n to goal node.

Algorithm of A* Search

Step1 : Place the starting node in OPEN list

Step2 :- Check if the OPEN list is Empty or not, if the list is Empty then return failure & stops.

Step3 :- Select the node from the OPEN list which has the smal value of evaluation function (g+h) if node n is goal no then return success & stop, otherwise

Step4 :- Expand node n & generate all of its successors, & put n in the closed list.

- For each successor 'n', check whether 'n' is already in the OPEN or CLOSED list,
- If not then compute evaluation function for 'n' and place into OPEN list.

Step5: Else if node 'n' is already in OPEN & CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step6: Return to step 2

Advantages:

- It is best alg than other search alg
- It is optimal & complete
- It can solve very complex problems.

Disadvantages:

- It does not always produce shortest path
- It is not practical for various large-scale problems

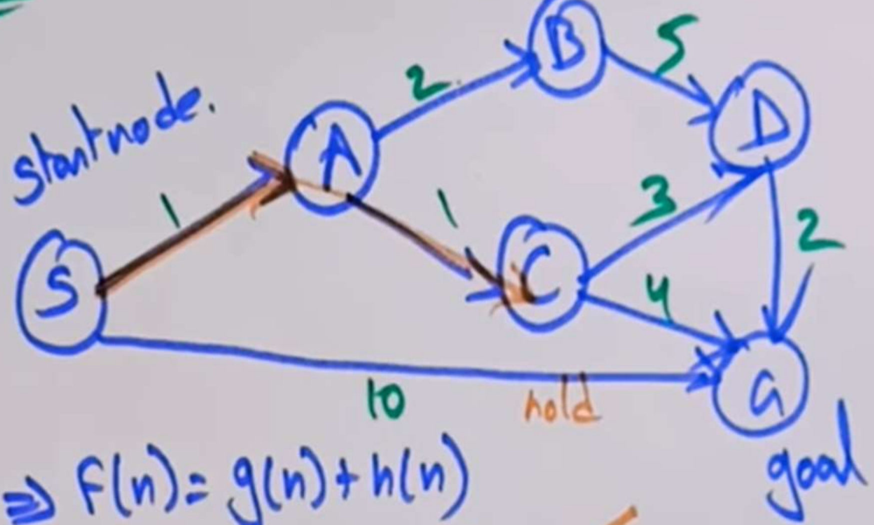
A* search

- **Idea**: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = cost so far to reach n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal
-

A* Search Algorithm:

Example:

state	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



① $S \rightarrow A \Rightarrow f(n) = g(n) + h(n)$
$= 1 + 3 = 4$ ✓

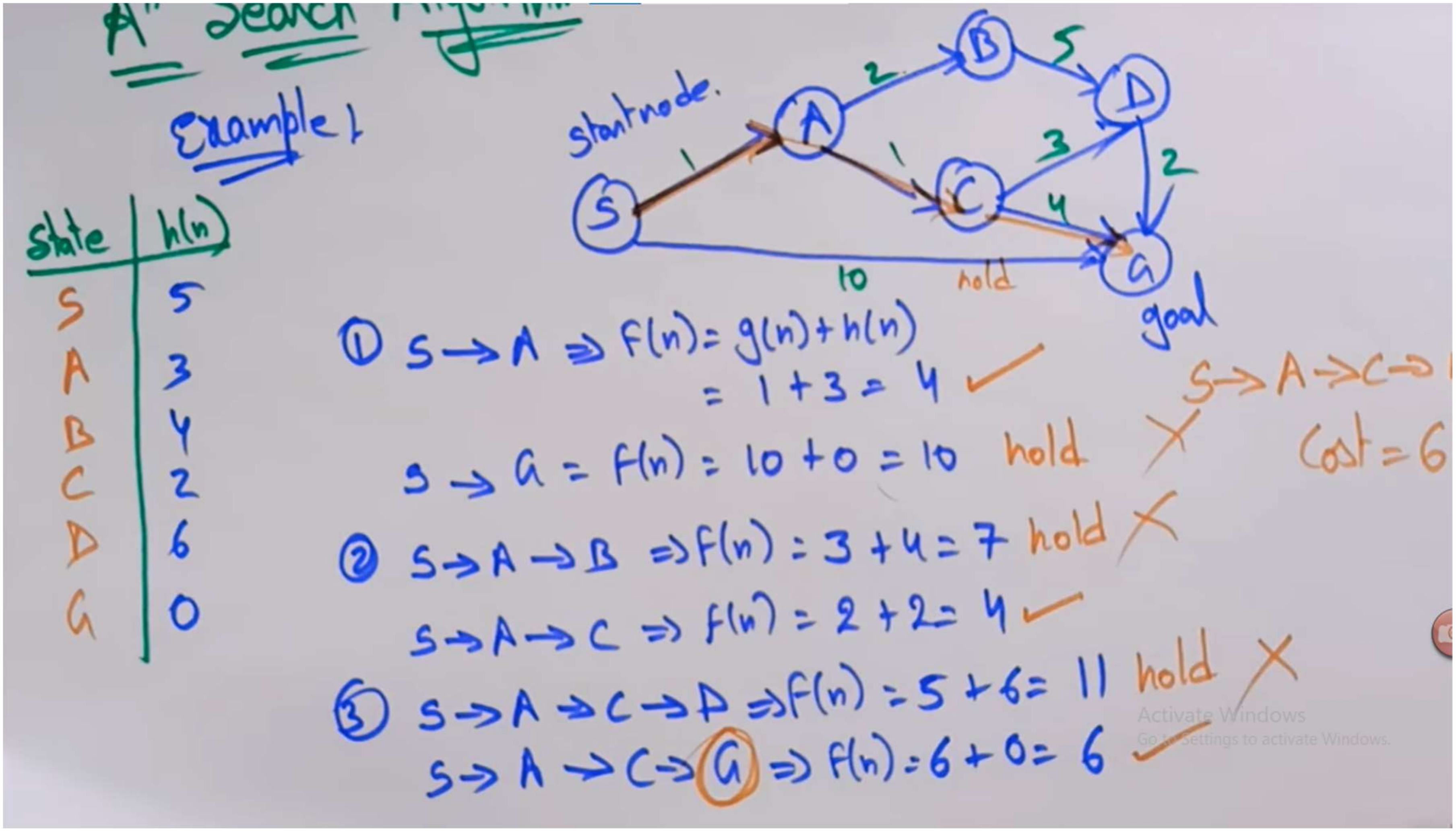
$S \rightarrow G = f(n) = 10 + 0 = 10$ hold

② $S \rightarrow A \rightarrow B \Rightarrow f(n) = 3 + 4 = 7$ hold

$S \rightarrow A \rightarrow C \Rightarrow f(n) = 2 + 2 = 4$ ✓

③ $S \rightarrow A \rightarrow C \rightarrow D \Rightarrow f(n) = 5 + 6 = 11$ hold

$S \rightarrow A \rightarrow C \rightarrow G \Rightarrow f(n) = 6 + 0 = 6$ ✓



Final Optimal path : S-> A-> C-> G

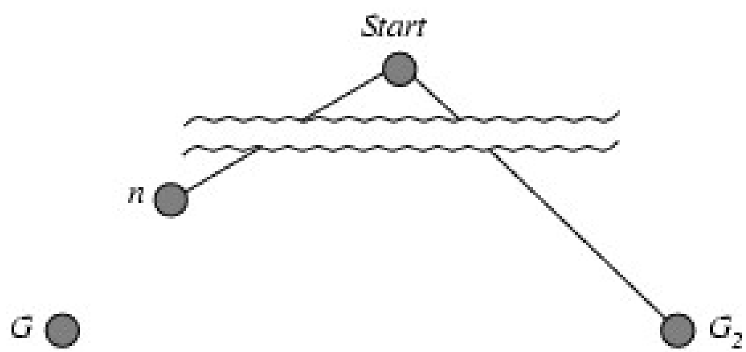
A* search

- Keypoints
  - **Complete** – Guarantee that reach the goal node from start node
  - **Optimal** – Yes.
  - Actually using the whole actual cost from the starting node and also using Heuristic Function which makes this algorithm a lot more optimal
  - How to calculate the value of Heuristic?
    - It is a guessing / calculated value
    - Use Manhatten distance, Euclidiean distance, diagonal distance
    - Usually the exact value of heuristic is not calculated

A^* search

- How to calculate the value of Heuristic?
  - It is a guessing / calculated value
  - Use Manhatten distance, Euclidiean distance, diagonal distance
  - Usually the exact value of heuristic is not calculated
  - Heuristic value is the value of the cost from one node to the GOAL node
  - Initially, in the problem space, we do not know exact cost from a node to the goal node
  - Hence, the guesses are made.
    - If we can calculate heuristic value, the whole algorithm becomes more optimal.
    - So finding exact time between a node and goal node increases TIME COMPLEXITY

Optimality of A* (proof)

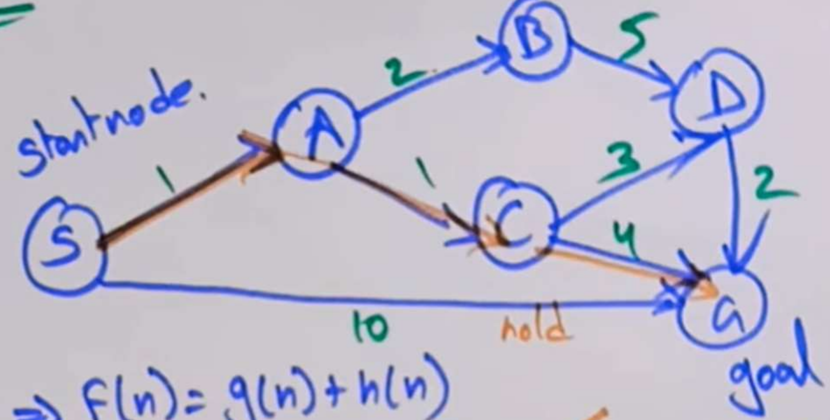
- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.
- 

- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

A* Search Algorithm

Example 1

State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



$S \to A \to C \to$

Cost = 6

① $S \to A \Rightarrow f(n) = g(n) + h(n)$
$= 1 + 3 = 4$ ✓

$S \to G = f(n) = 10 + 0 = 10$ hold ✗

② $S \to A \to B \Rightarrow f(n) = 3 + 4 = 7$ hold ✗

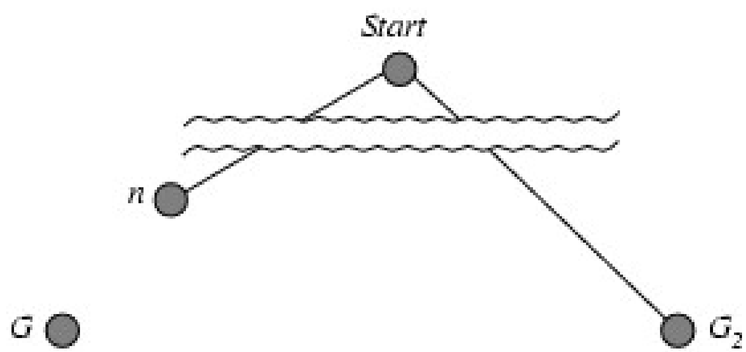
$S \to A \to C \Rightarrow f(n) = 2 + 2 = 4$ ✓

③ $S \to A \to C \to D \Rightarrow f(n) = 5 + 6 = 11$ hold ✗

$S \to A \to C \to G \Rightarrow f(n) = 6 + 0 = 6$ ✓

Optimality of A* (proof)

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.
-



- $f(G_2)$ > $f(G)$
- $h(n)$ ≤ h^*(n) since h is admissible
- $g(n) + h(n)$ ≤ $g(n) + h^*(n)$ [**f(S-> A)** = 4 ≤ **further f(..) values for various heuristic values**
- **f(n)** ≤ **f(G)** [**f(S-> A)** = 4 < **f(S-> A->C->G)** = 6]
-

Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion

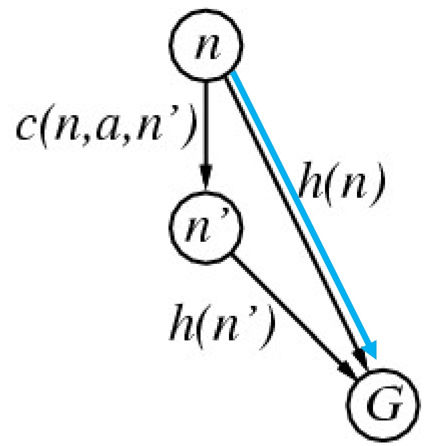
A* Search

- Expand node in frontier with best evaluation function score **f(n)**
  - **f(n) = g(n) + h(n)**
  - **g(n) :=** cost to get from initial state to **n**
  - **h(n) :=** heuristic estimate of cost to get from **n** to goal
- Optimal in trees if **admissible** **h(n)** <= true cost to goal
- Optimal in graphs if **consistent** **h(n)** <= **c(n, n')** + **h(n')**

Consistent heuristics

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*,

  $h(n) \leq c(n,a,n') + h(n')$



If *h* is consistent, we have

$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

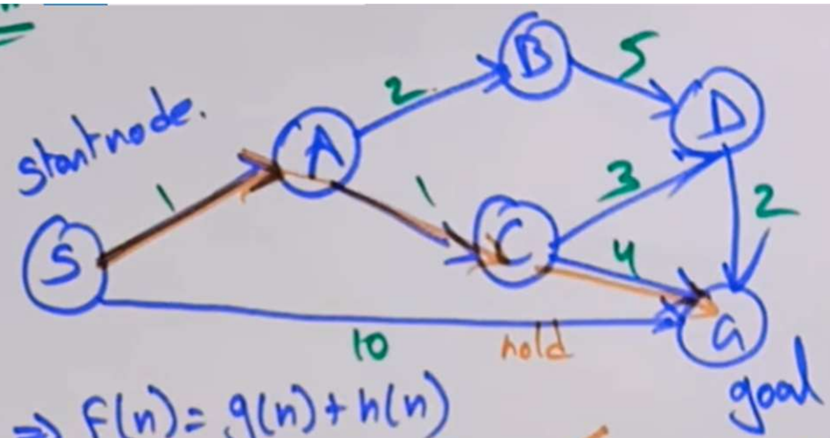
i.e., *f(n)* is non-decreasing along any path.

- Theorem: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal

A* Search Algorithm

Example:

state	h(n)
S	5
A	3
B	4
C	2
D	6
G	0



① $S \rightarrow A \Rightarrow f(n) = g(n) + h(n)$

$= 1 + 3 = 4$ ✓

$\boxed{S \rightarrow G = f(n) = 10 + 0 = 10}$ hold ✗

② $S \rightarrow A \rightarrow B \Rightarrow f(n) = 3 + 4 = 7$ hold ✗

$S \rightarrow A \rightarrow C \Rightarrow f(n) = 2 + 2 = 4$ ✓

③ $S \rightarrow A \rightarrow C \rightarrow D \Rightarrow f(n) = 5 + 6 = 11$ hold ✗

$S \rightarrow A \rightarrow C \rightarrow G \Rightarrow f(n) = 6 + 0 = 6$ ✓

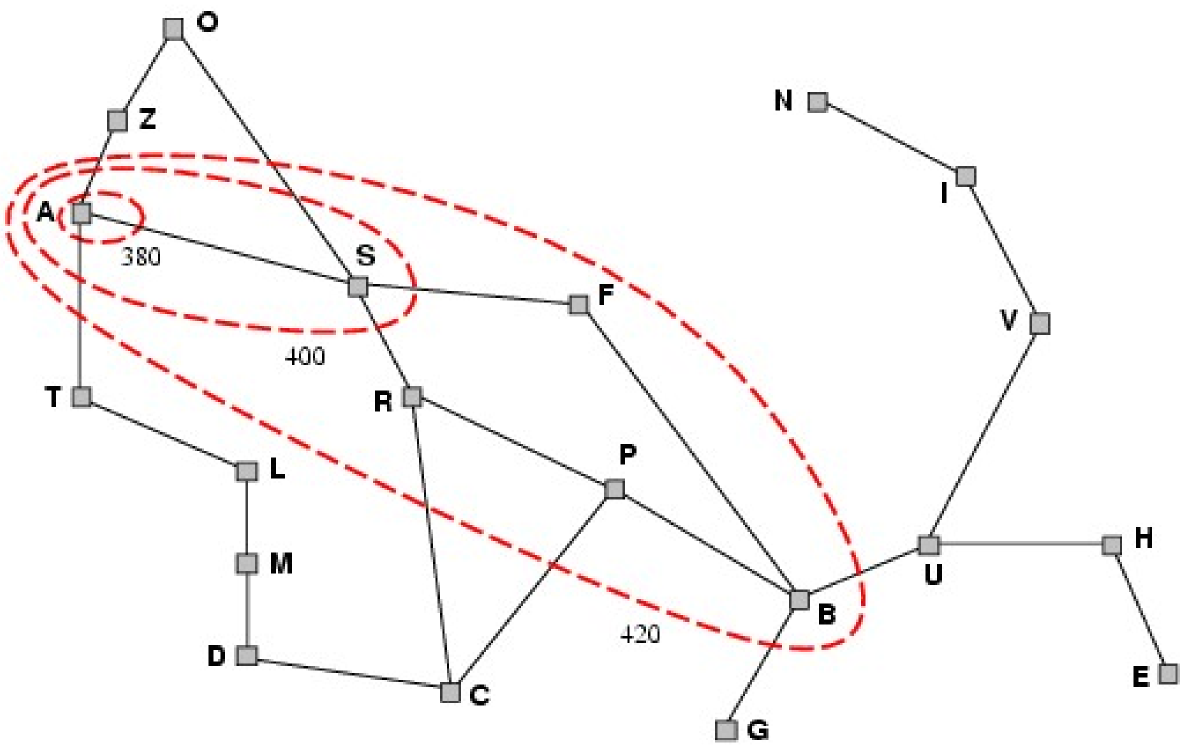
$S \rightarrow A \rightarrow C \rightarrow$

Cost = 6

Optimality of A*

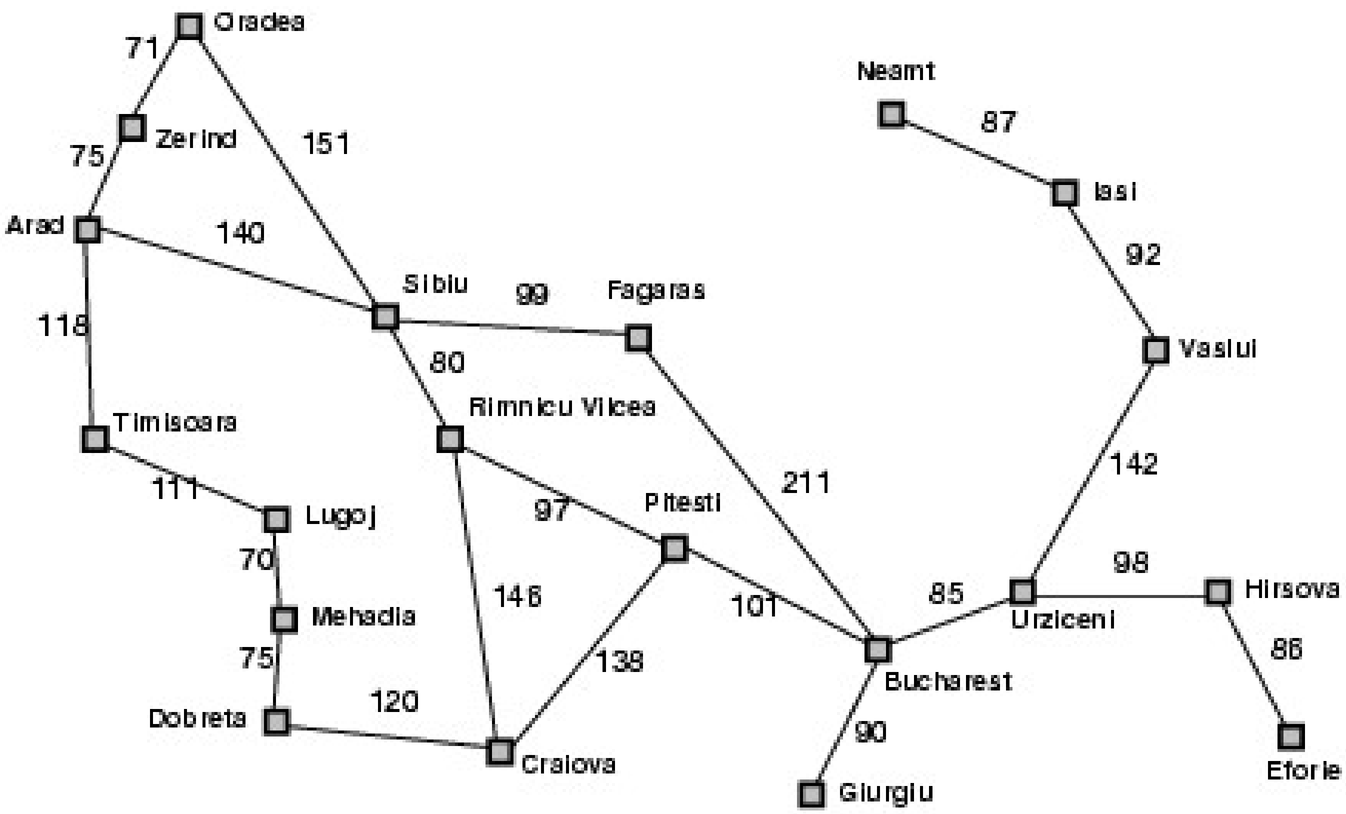
- A^* expands nodes in order of increasing f value

- Gradually adds "f-contours" of nodes
- Contour i has all nodes with $f=f_i$, where $f_i < f_{i+1}$



Example 3 – A* Search

Romania with step costs in km

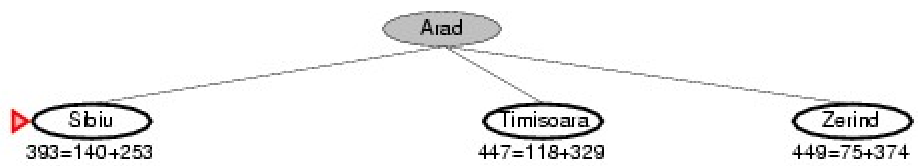


Straight–line distance
to Bucharest

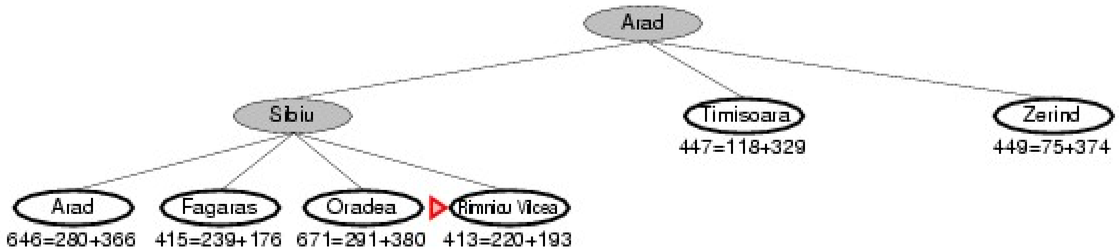
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

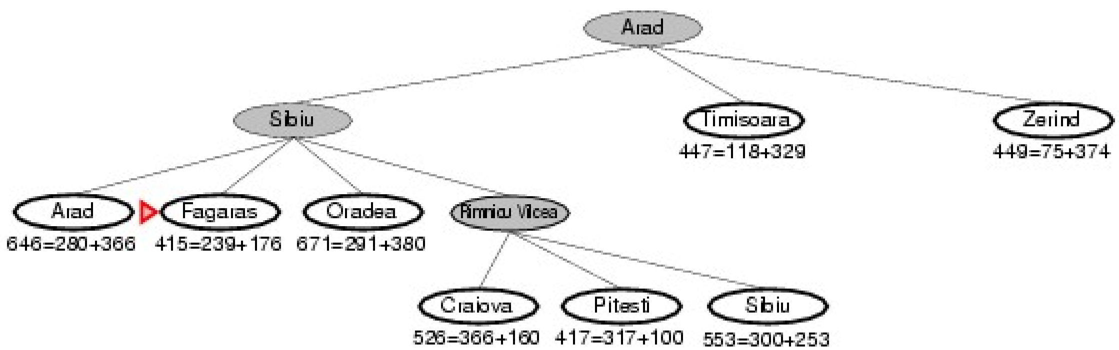
A* search example



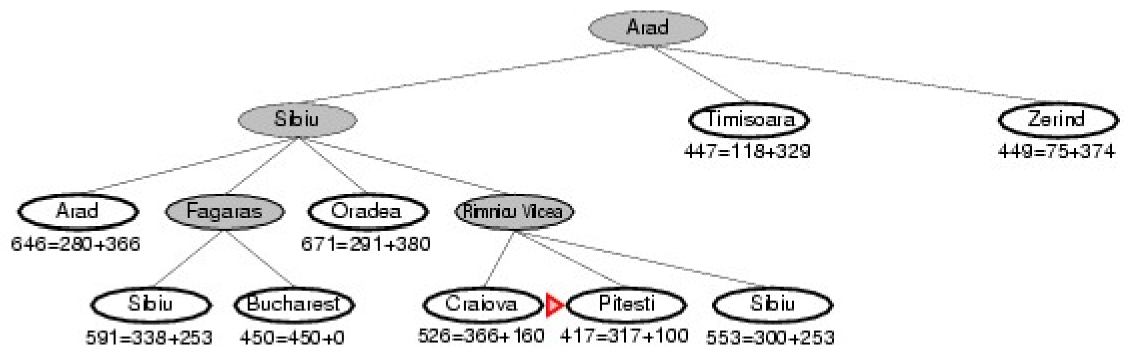
A* search example

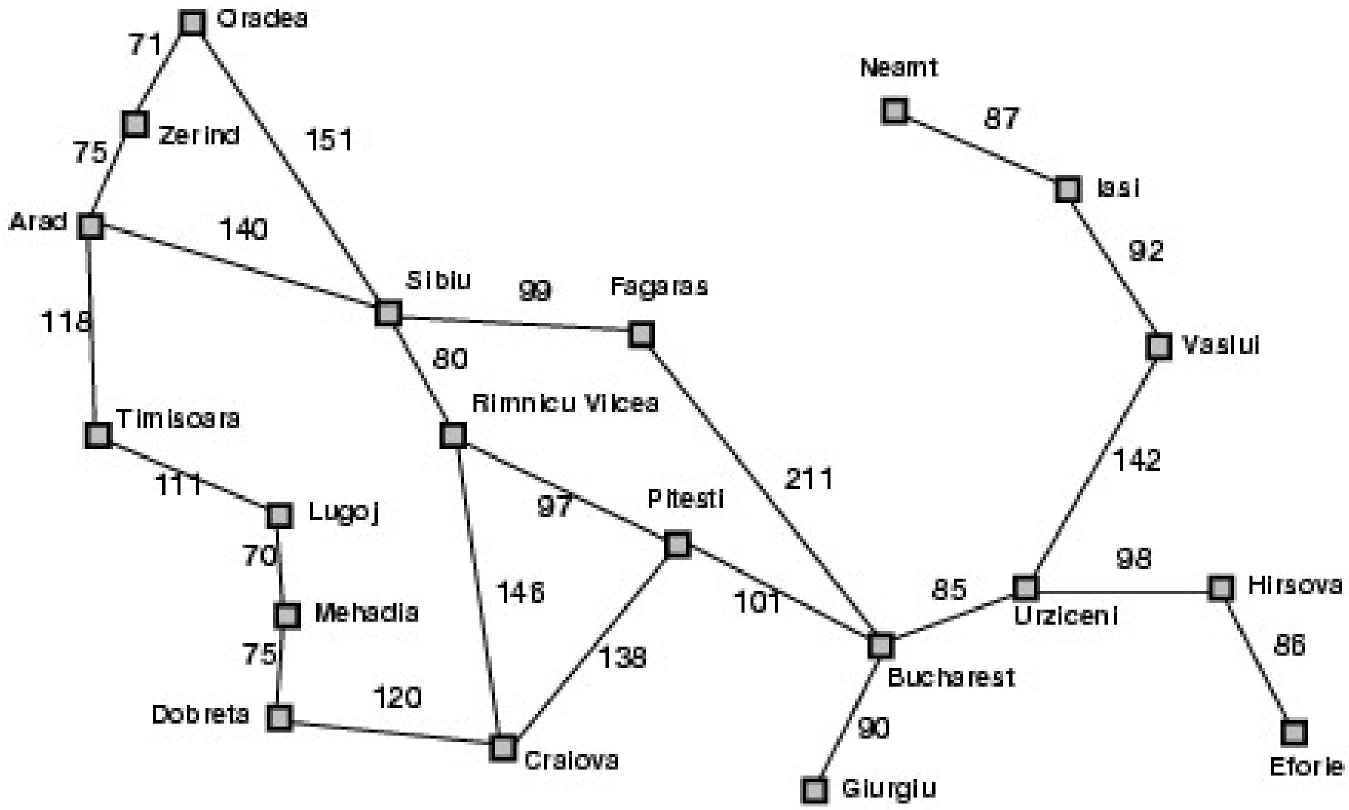


A* search example



A* search example

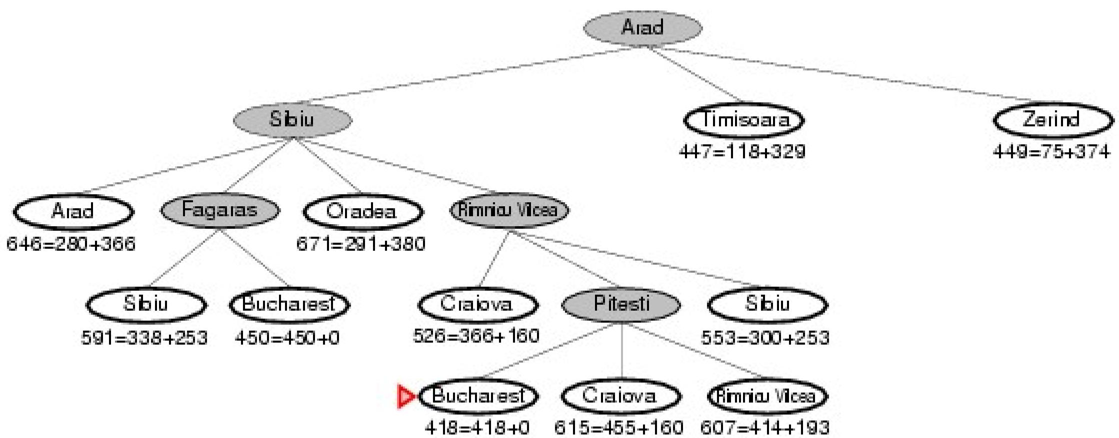




Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example



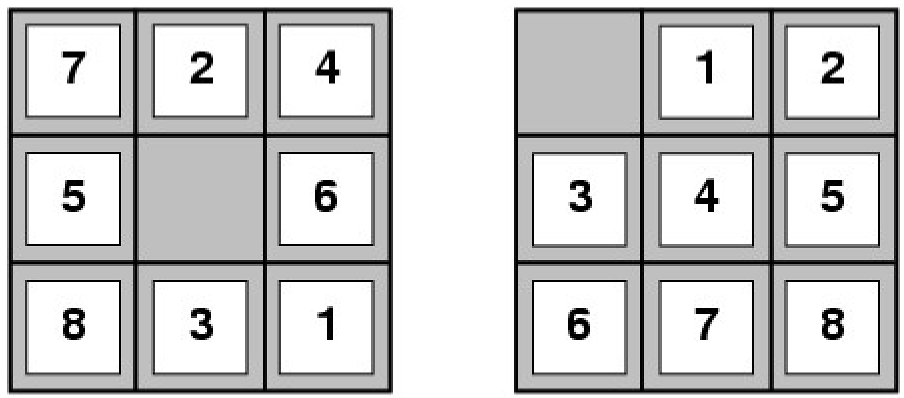
8-puzzle - **Admissible heuristics**

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State

Goal State

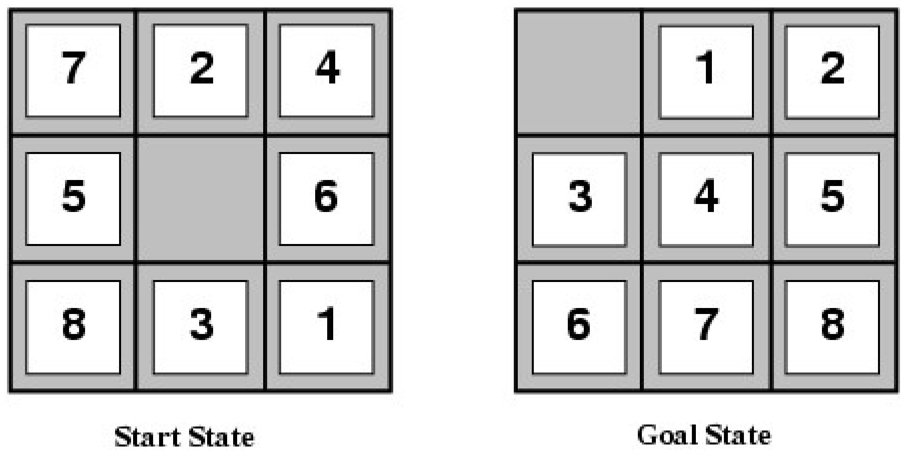
- $\underline{h_1(S) = ?}$
- $\underline{h_2(S) = ?}$

Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
- then h_2 dominates h_1
- h_2 is better for search
- Typical search costs (average number of nodes expanded):

- $d=12$ IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- $d=24$ IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution