Artificial & Computational Intelligence

AIMLCLZG557

M2 : Problem Solving Agent using Search

Indumathi V

Guest Faculty

BITS -WILP

BITS Pilani

Pilani Campus

Course Plan

M1	Introduction to AI
M2	Problem Solving Agent using Search
M3	Game Playing
M4	Knowledge Representation using Logics
M5	Probabilistic Representation and Reasoning
M6	Reasoning over time
M7	Ethics in AI

Module 2 : Problem Solving Agent using Search

A. Uninformed Search

B. Informed Search

C. Heuristic Functions

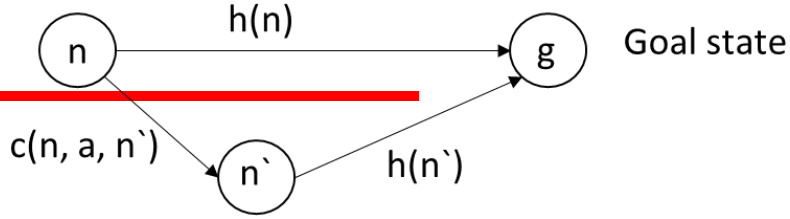
D. Local Search Algorithms & Optimization Problems

Learning Objective

At the end of this class , students Should be able to:

1. Differentiate between uninformed and informed search requirements
2. Apply UCS, GBFS & A* algorithms to the given problem
3. Prove if the given heuristics are admissible and consistent
4. Design and compare heuristics apt for given problem
5. Apply A* variations algorithms to the given problem

A* Search



Optimal on condition

h(n) must satisfies two conditions:

- Admissible Heuristic – one that never overestimates the cost to reach the goal
- Consistency – A heuristic is consistent if for every node n and every successor node n` of n generated by action a, h(n) <= c(n, a, n`) + h(n`)

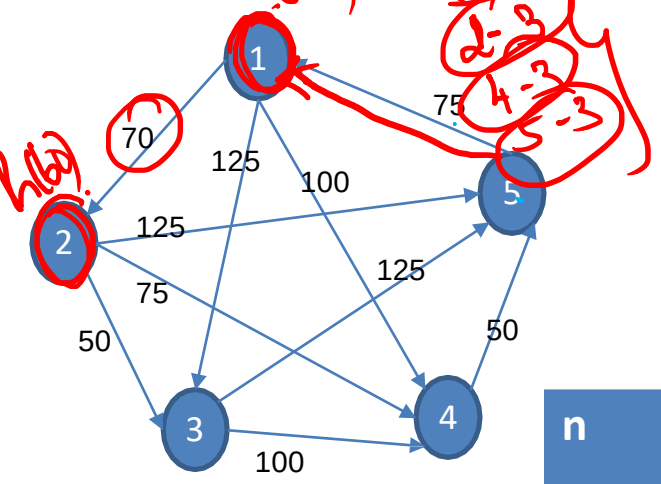
Complete

- If the number of nodes with cost <=C* is finite
- If the branching factor is finite
- A* expands no nodes with f(n) > C*, known as pruning

Time Complexity - $G(b^{\Delta})$ where the absolute error $\Delta = h^* - h$

A* Search

Is the heuristic designed leads to optimal solution?

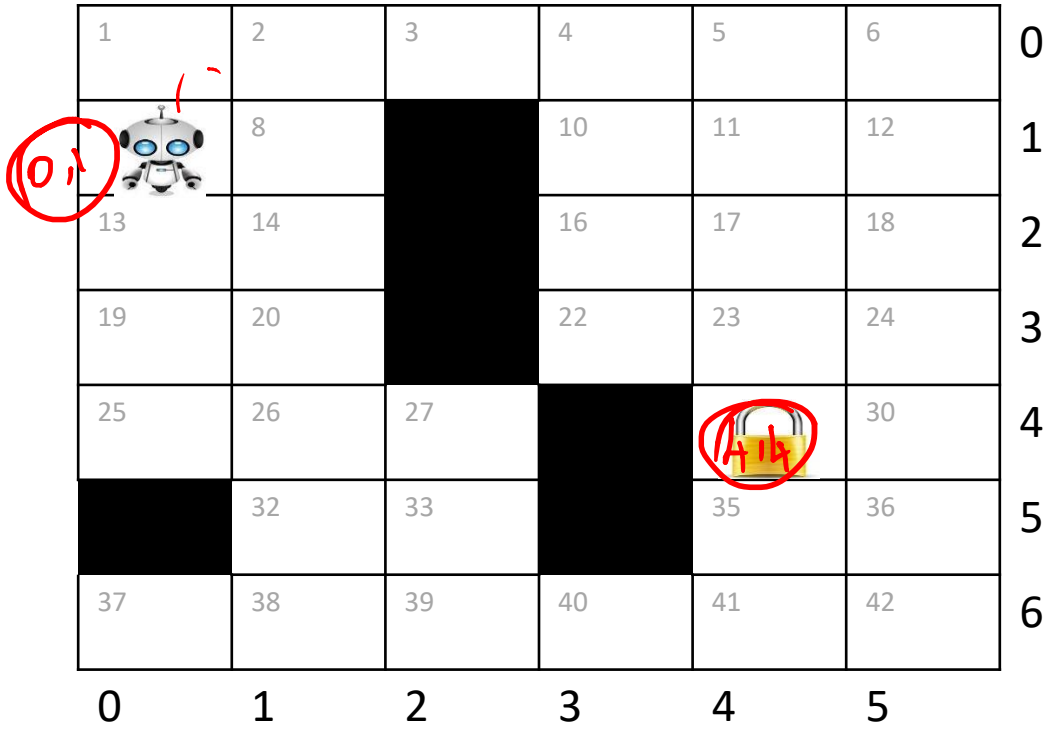


Assuming node 3 as goal, taking only sample edges per node below is checked for consistency

n	h(n)	Is Admissible? h(n) <= h*(n)	Is Consistent? For every arc (i,j): h(i) <= g(i,j) + h(j)
1	80	Y	N (5→1): 190<=155
2	60	N	Y (1→2) : 80<=130
3	0	Y	
4	200	Y	Y (1→4) : 80<=300 Y (2→4) : 60<=275
5	190	Y	Y (2→5) : 60<=315 Y (4→5) : 200<=240

Path finding Robot – Sample Planning Agent Design

Successor Function Design



L2

① $h_1(n)$ ⇒ Euclidean dist

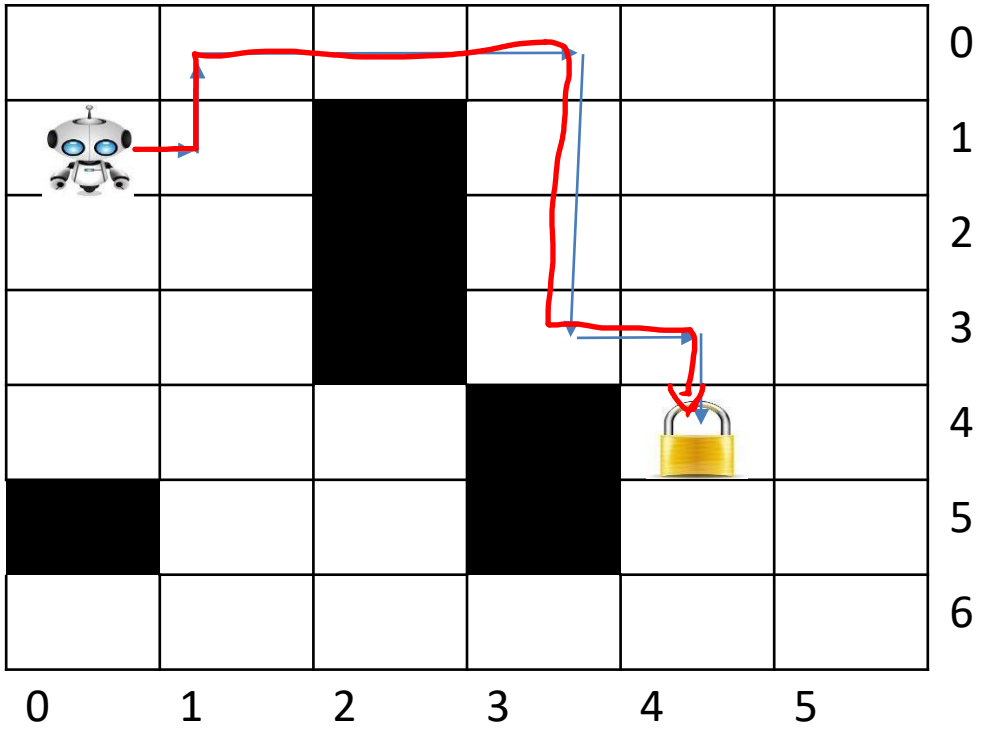
② $h_2(n)$ ⇒ Manhanten dist

L_1

N-W-E-S

A*

Demo



① A* h(n) → good
↓ optimal soln

② A* variation of BFS

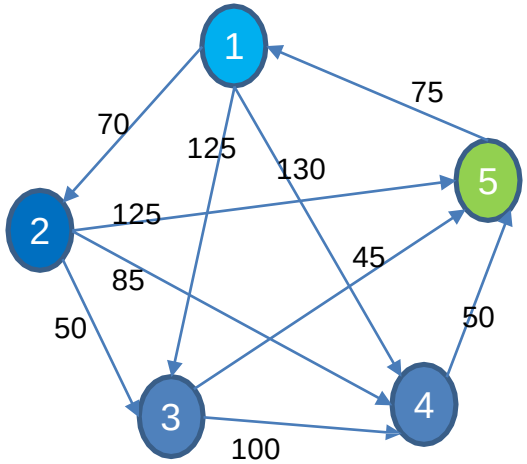
Variations of A*

Memory Bounded Heuristics

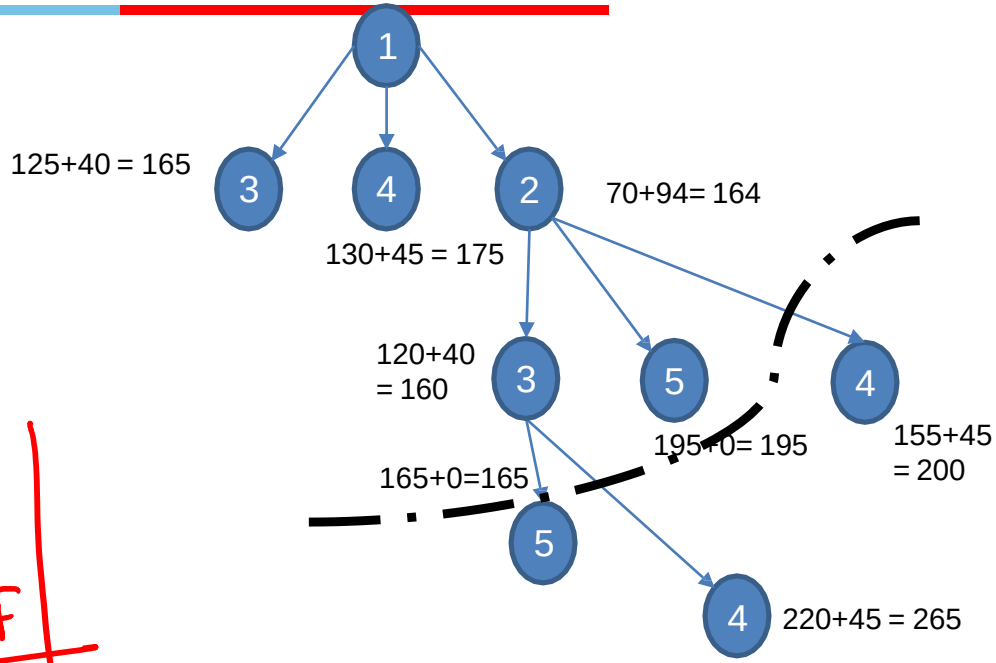
① IDA*
② RBFS

Iterative Deepening A*

Set limit for f(n)



n	h(n)
1	60
2	94
3	40
4	45
5	0



Cut off value is the smallest of f-cost of any node that exceeds the cutoff on previous iterations

Iterative Limit : Eg

f(n) = 180
f(n) = 195
f(n) = 200
.
.
.

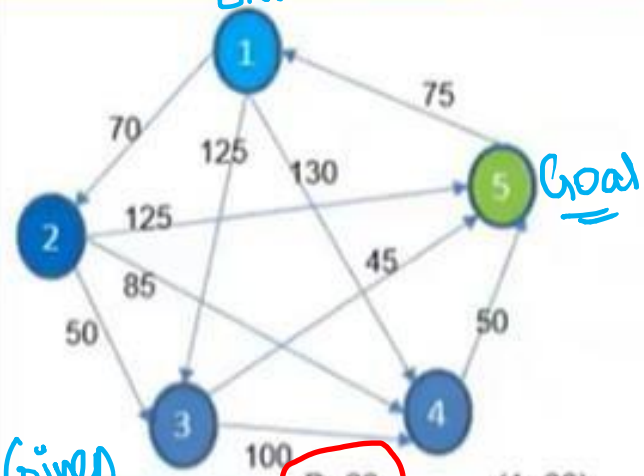
Iterative Deepening A*

DFS A*

limit ⇒ f(f(n) → g(n)+h(n))

No Priority Queue

bcoz DFS → last node located in top of the stack.



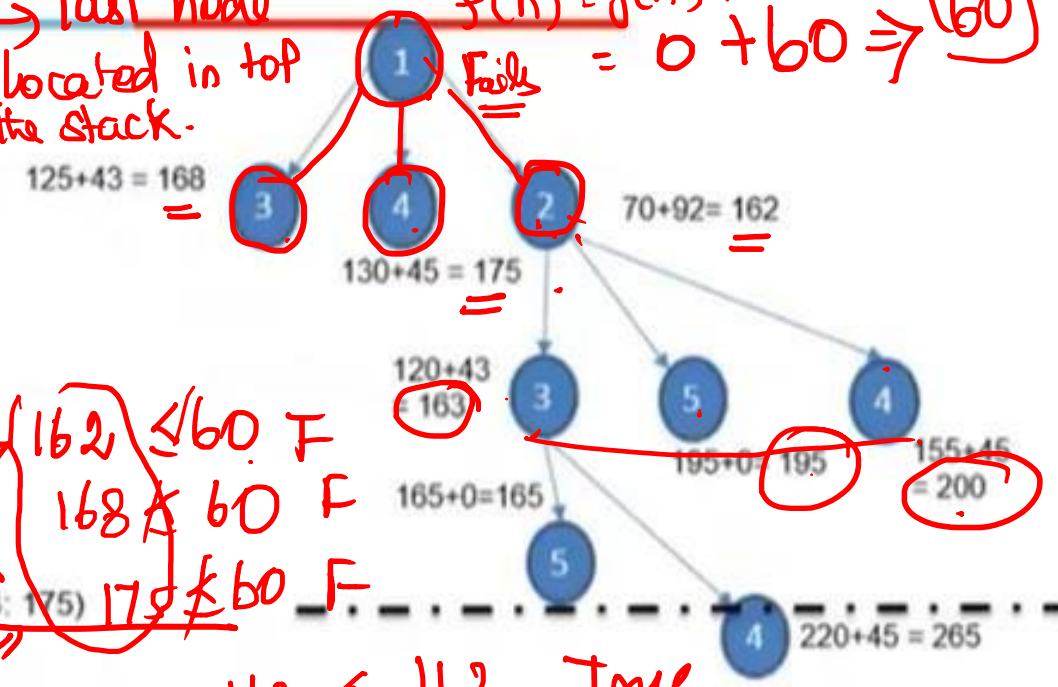
Given

n	h(n)
1	60
2	92
3	43
4	45
5	0

Set limit for f(n)

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

= 0 + 60 ⇒ 60



1-2 ⇒ 162 ≤ 60 F

168 ≤ 60 F

175 ≤ 60 F

162 ≤ 162 True

B=60

(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)

B=162

(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)

163 ≤ 162 Fail F

200 ≤ " F

195 ≤ 162 F

168 ≤ 162 F

175 ≤ 162 F

B=163

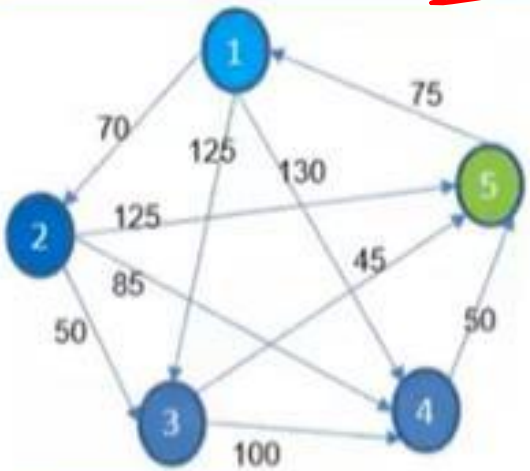
(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)
TEST-F

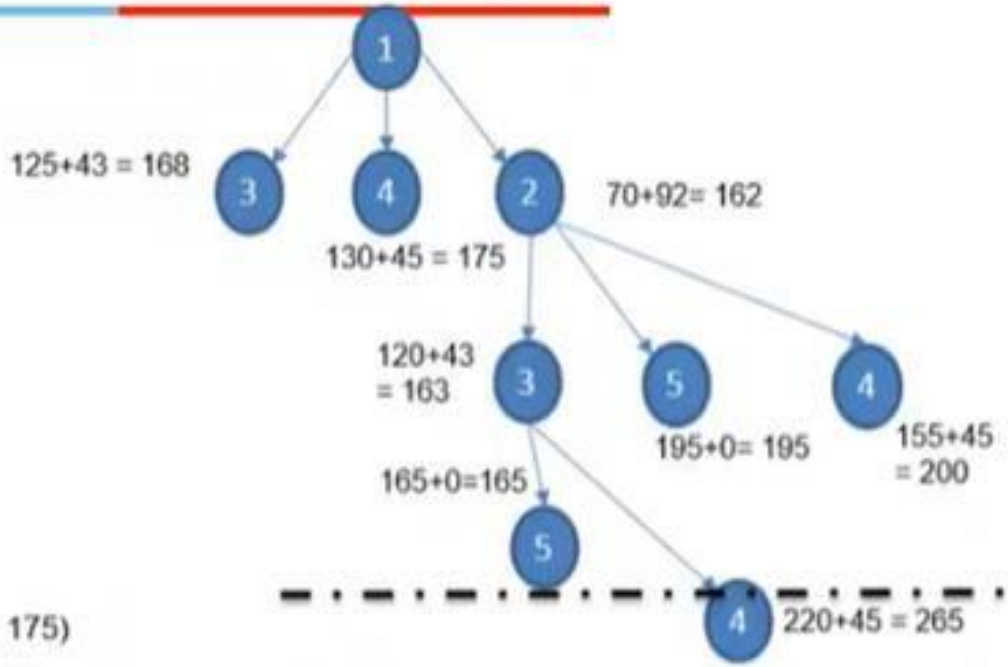
Iterative Deepening A*

Restart cost ↑ A* BFS

Larger

Set limit for f(n)





n	h(n)
1	60
2	92
3	43
4	45
5	0

B=163

(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3 5: 165) (1 2 3 4: 265) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)

B=165

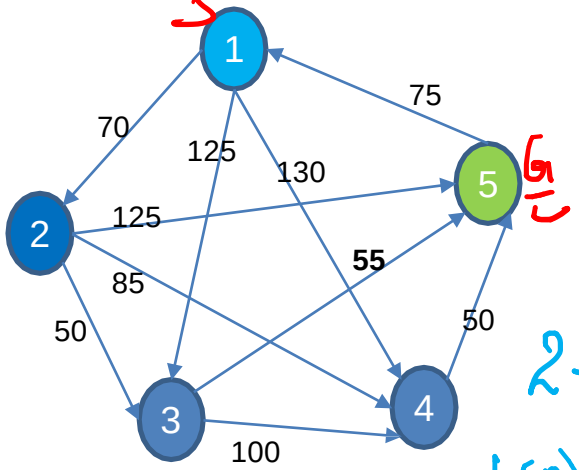
(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3 5: 165) (1 2 3 4: 265) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)

Path

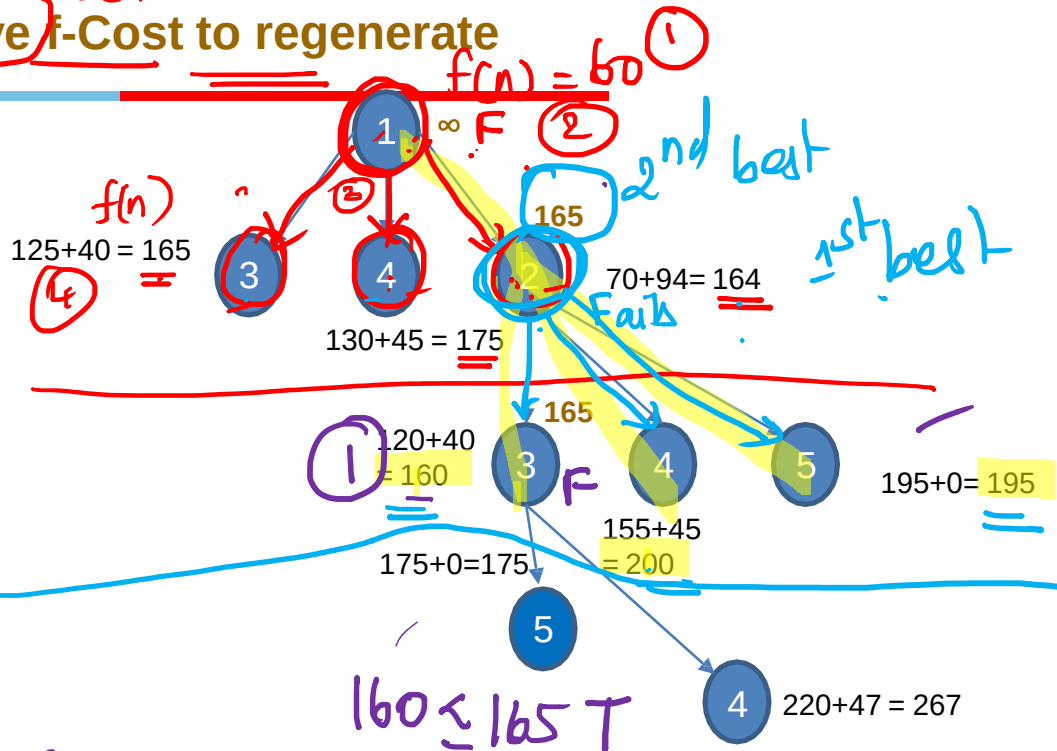
Recursive Best First Search A*

innovate achieve lead

Remember the next best alternative f-Cost to regenerate



n	h(n)
1	60
2	94
3	40
4	45
5	0



(1, 60)

(1 2 | 164) (1 3 | 165) (1 4 | 175)

(1 2 | 175) (1 4 | 175) (1 3 | 180)

(1 2 3 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200)

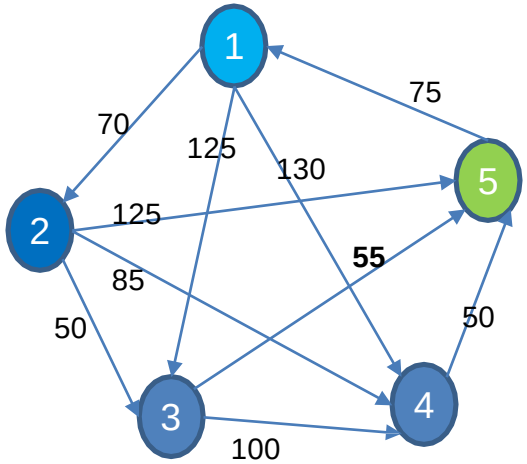
(1 2 3 5 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200) (1 2 3 4 | 267)

PASS

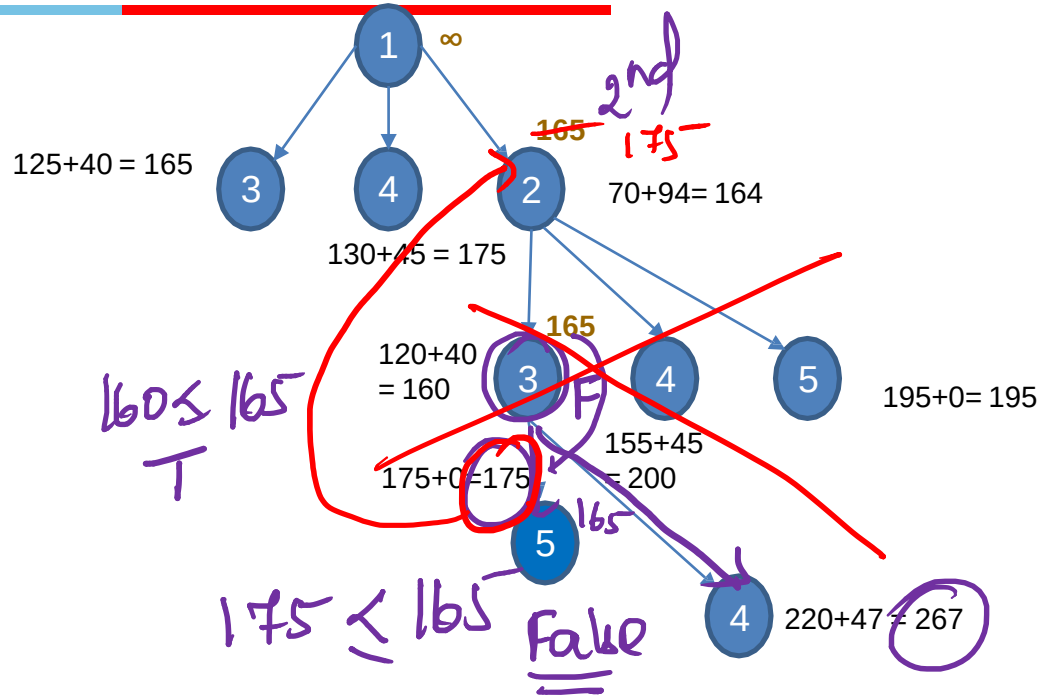
Recursive Best First Search A*

innovate achieve lead

Remember the next best alternative f-Cost to regenerate



n	h(n)
1	60
2	94
3	40
4	45
5	0



(1, 60)
(1 2 | 164) (1 3 | 165) (1 4 | 175)

(1 2 | 175) (1 4 | 175) (1 3 | 180)

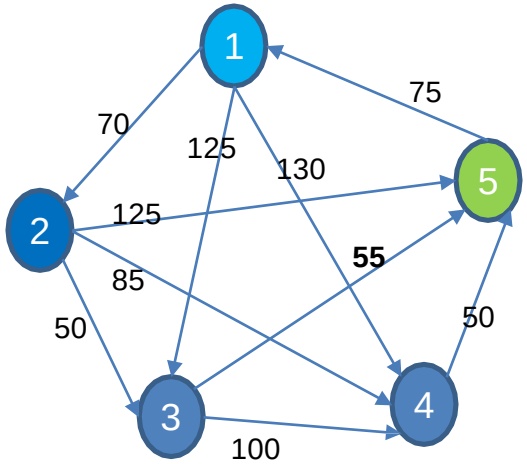
(1 2 3 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200)

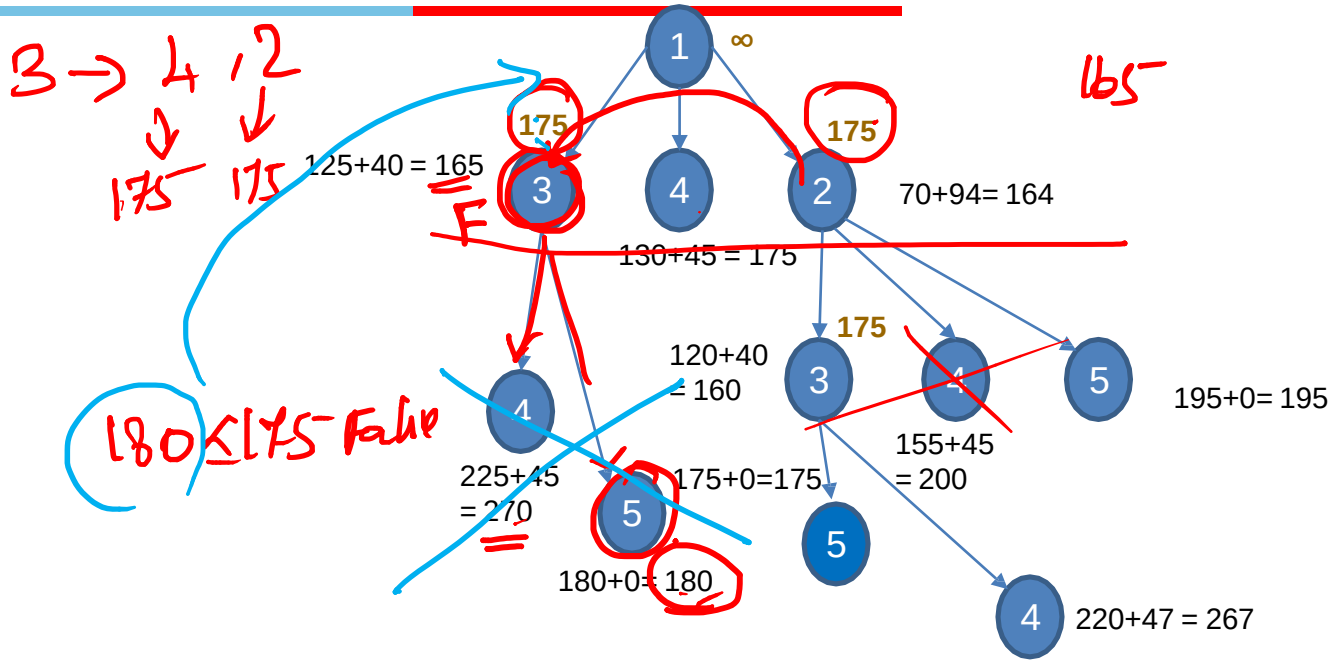
(1 2 3 5 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200) (1 2 3 4 | 267)

PASS

Recursive Best First Search A*

Remember the next best alternative f-Cost to regenerate





n	h(n)
1	60
2	94
3	40
4	45
5	0

(1, 60)
(1 2 | 164) (1 3 | 165) (1 4 | 175)

(1 2 | 175) (1 4 | 175) (1 3 | 180)

(1 2 3 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200)

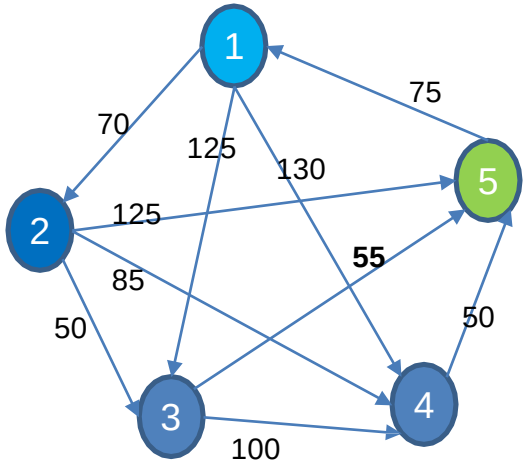
(1 2 3 5 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200) (1 2 3 4 | 267)

PASS

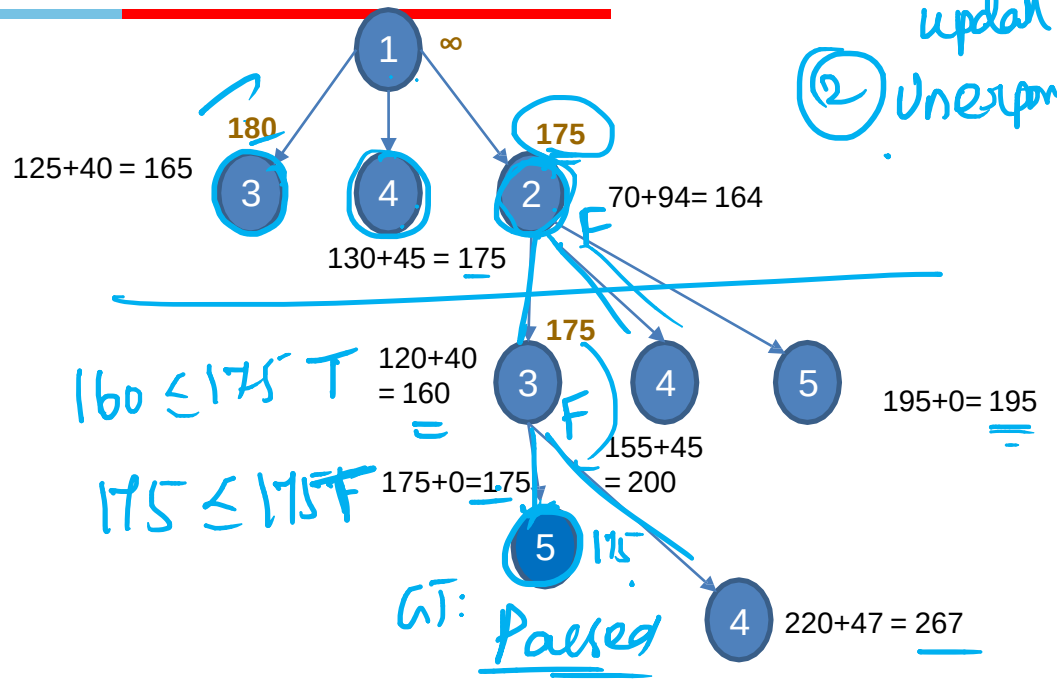
Recursive Best First Search A*

innovate achieve lead

Remember the next best alternative f-Cost to regenerate



n	h(n)
1	60
2	94
3	40
4	45
5	0



(1, 60)
(1 2 | 164) (1 3 | 165) (1 4 | 175)

(1 2 | 175) (1 4 | 175) (1 3 | 180)

(1 2 3 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200)

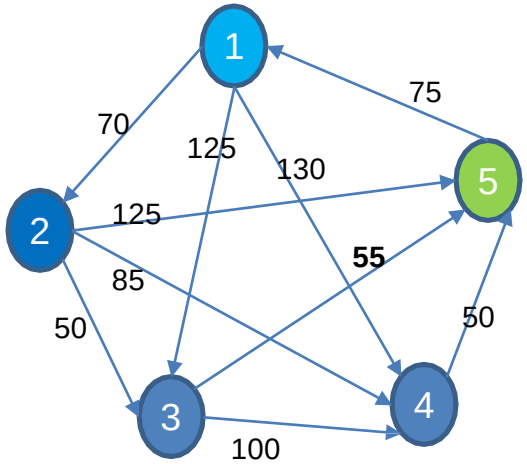
(1 2 3 5 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200) (1 2 3 4 | 267)

PASS

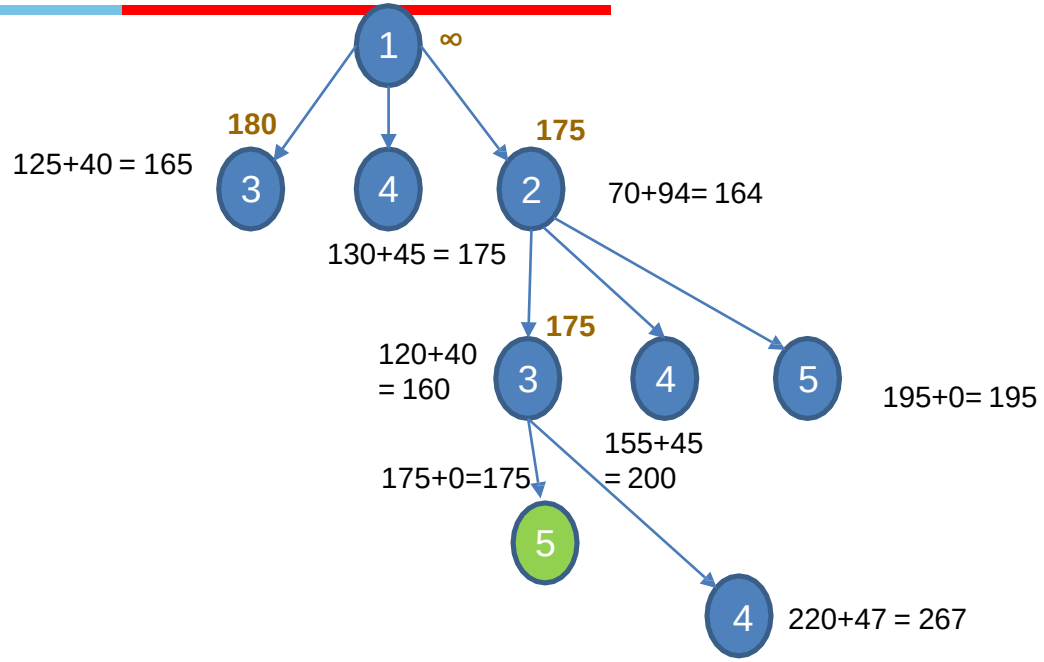
Recursive Best First Search A*

innovate achieve lead

Remember the next best alternative f-Cost to regenerate



n	h(n)
1	60
2	94
3	40
4	45
5	0



If the current best leaf value > best alternative path

Best leaf value of the forgotten subtree is backed up to the ancestors

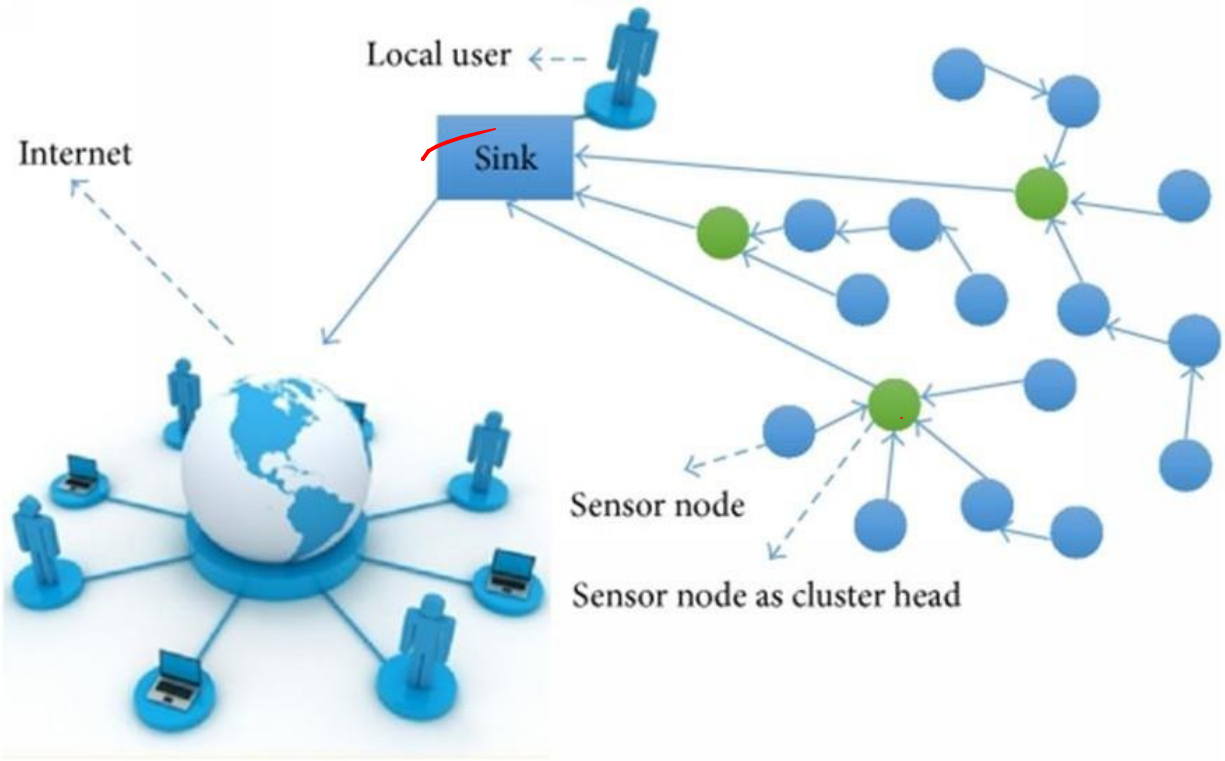
Recursion unwinds

Else

Continue expansion

Space Usage = O(bd) very less

Case Study – Search in Network Routing



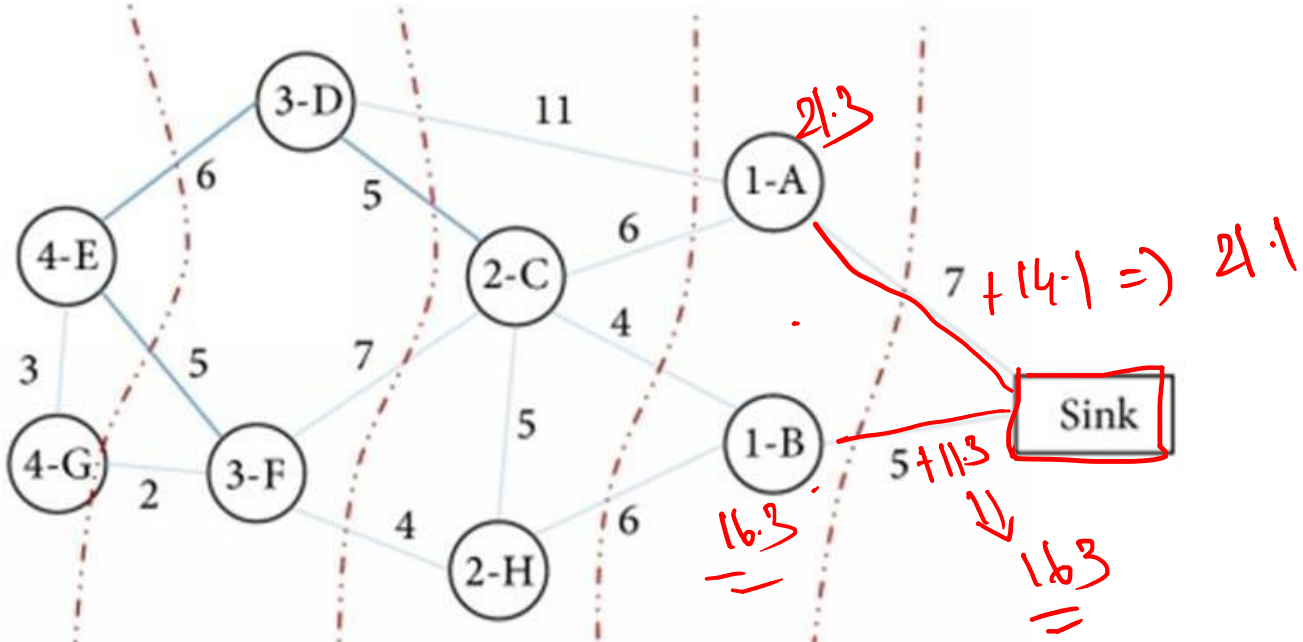
Source Credit :

AR-RBFS: Aware-Routing Protocol Based on Recursive Best-First Search Algorithm for Wireless Sensor Networks
https://doi.org/10.1155/2016/8743927

Case Study – Search in Network Routing

A	14.1
B	11.3
C	8.2
H	6.6
F	2
E	3
D	4.8

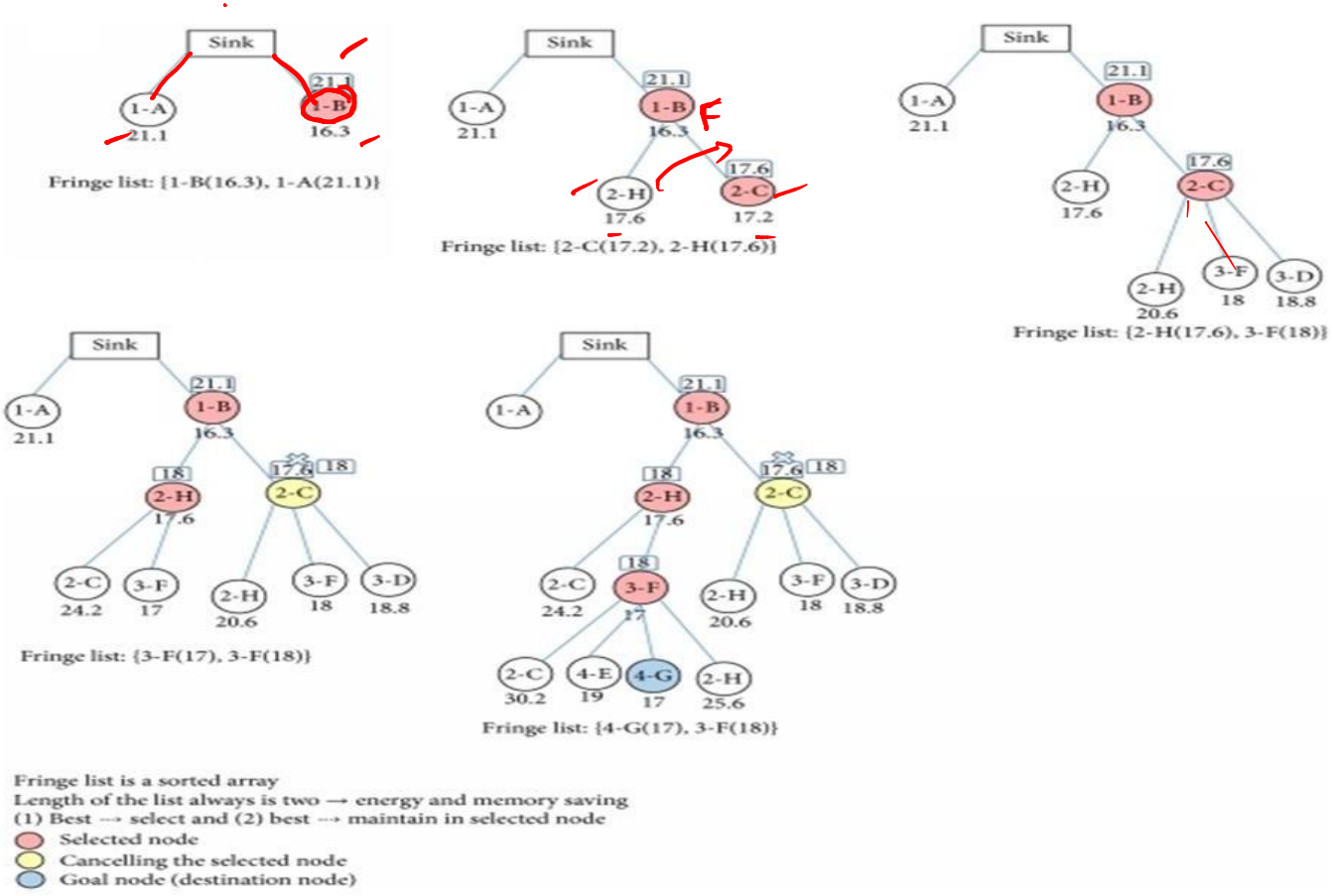
Heuristic values toward G



Source Credit :

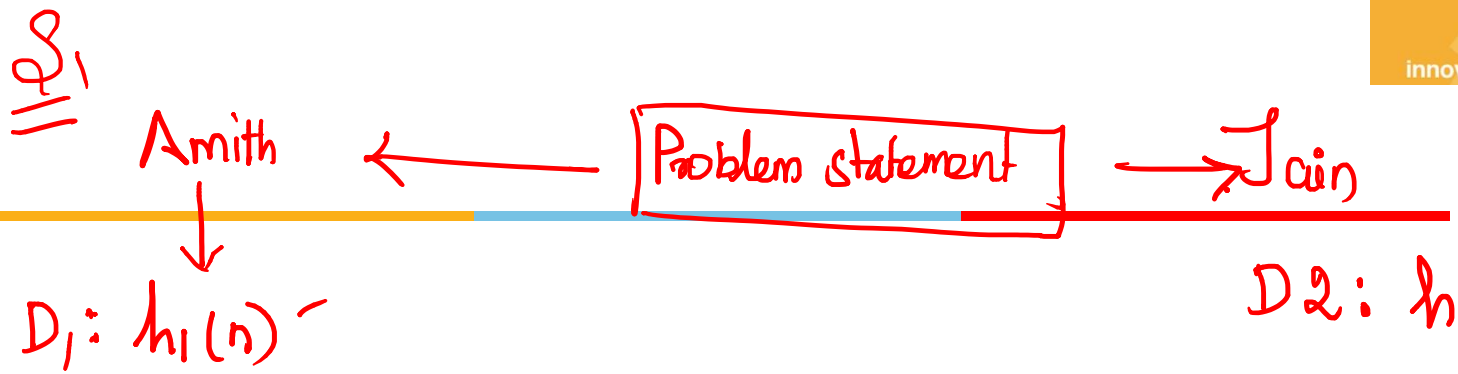
AR-RBFS: Aware-Routing Protocol Based on Recursive Best-First Search Algorithm for Wireless Sensor Networks
https://doi.org/10.1155/2016/8743927

Case Study – Search in Network Routing



Source Credit :

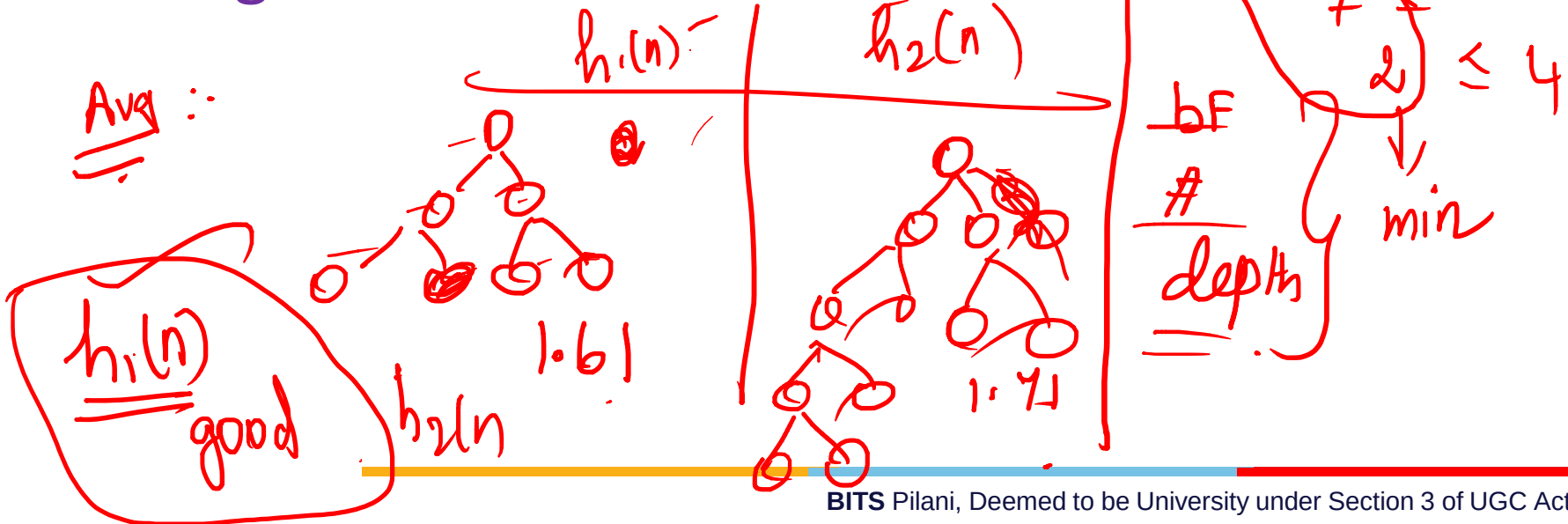
AR-RBFS: Aware-Routing Protocol Based on Recursive Best-First Search Algorithm for Wireless Sensor Networks
https://doi.org/10.1155/2016/8743927



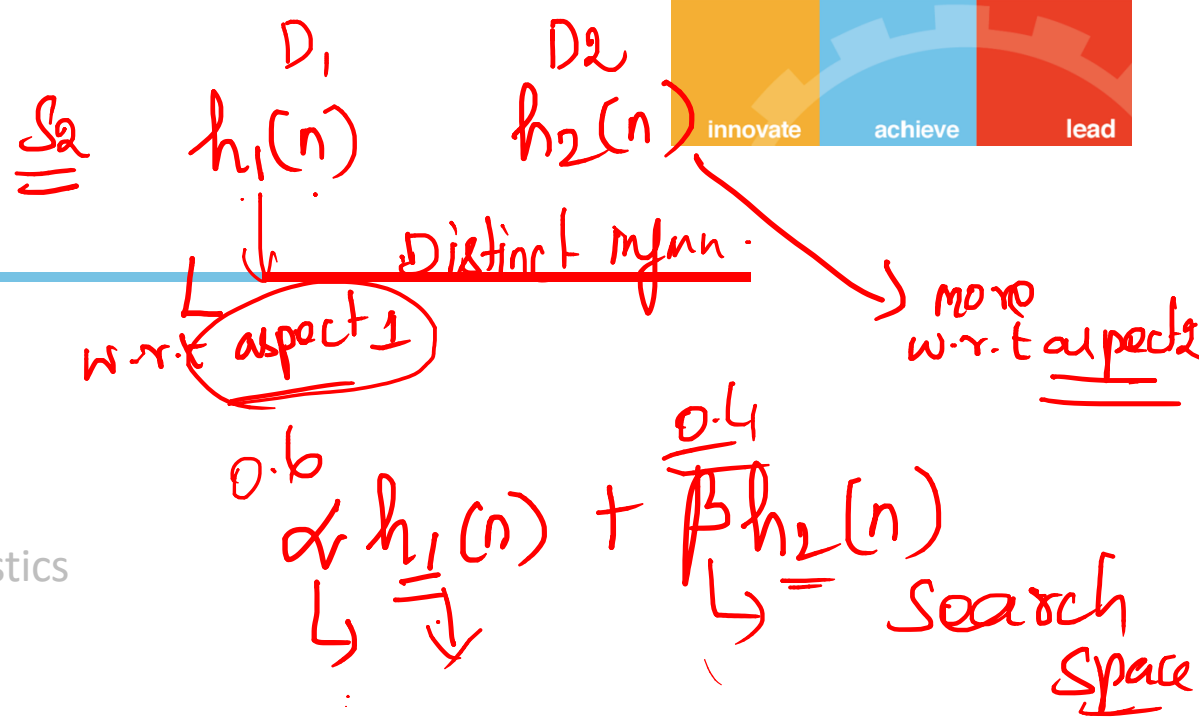
Information

* branching
* # nodes generated
* $h_1(n)$

Design of Heuristics



Heuristic Design



- **Effective Branching Factor**
- Good Heuristics
- Notion of Relaxed Problems
- Generating Admissible Heuristics

Effective branching factor (b^*):

If the algorithm generates N number of nodes and the solution is found at depth d, then

$$N + 1 = 1 + (b^*) + (b^*)^2 + (b^*)^3 + \ldots + (b^*)^d$$

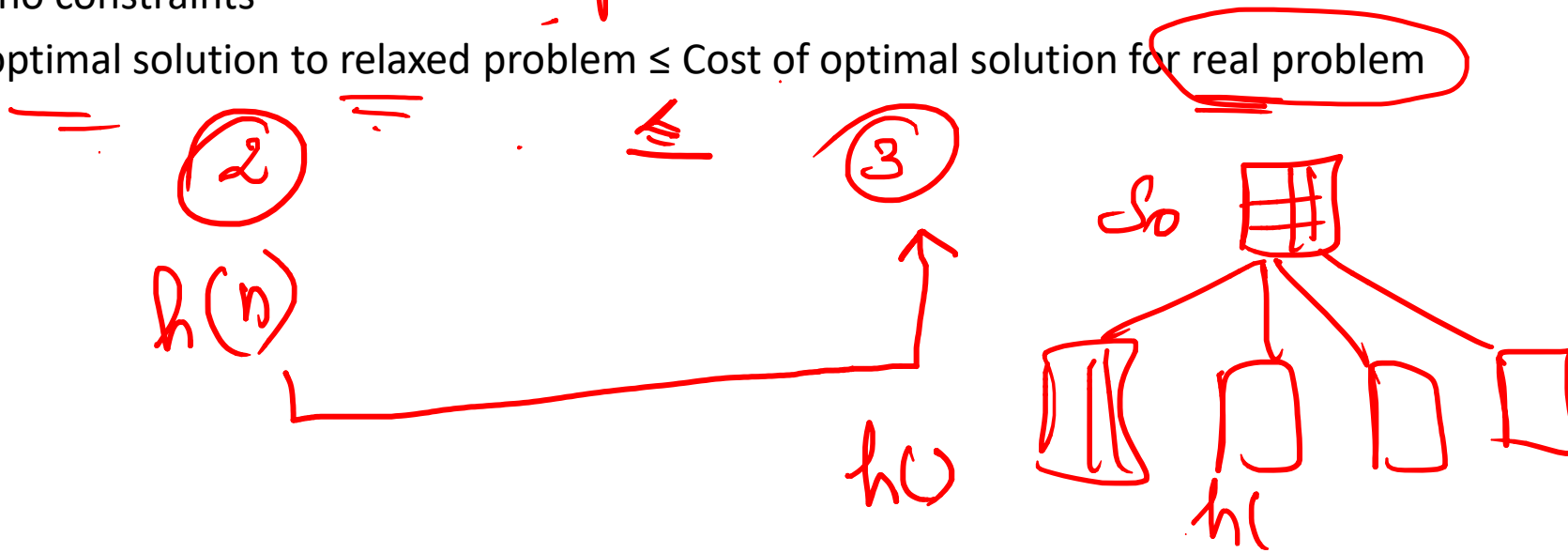
Heuristic Design

- Effective Branching Factor
- Good Heuristics
- **Notion of Relaxed Problems**
- Generating Admissible Heuristics

Simplify the problem

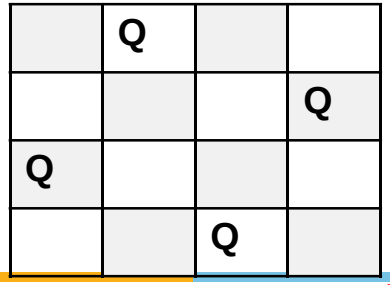
Assume no constraints

Cost of optimal solution to relaxed problem ≤ Cost of optimal solution for real problem



Design of Heuristics

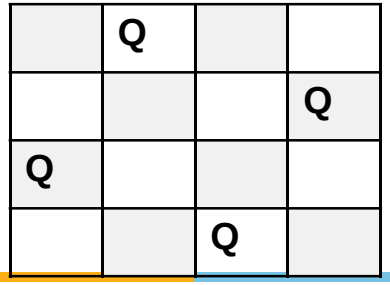
N-Queen

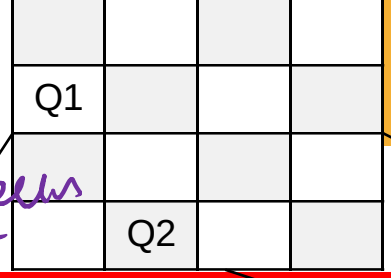


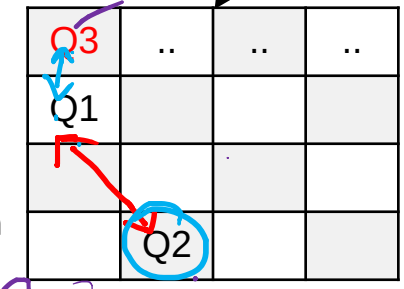
- Construct the search tree by considering one row of the board at a time
- State space graph of relaxed problem is a super graph of original state space because of removal of restrictions

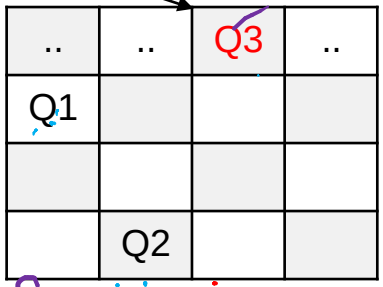
Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
< Xi , Yi >	Place in any non-occupied row in board		isValid Non-Attacking	Transition + Valid Queens	n!

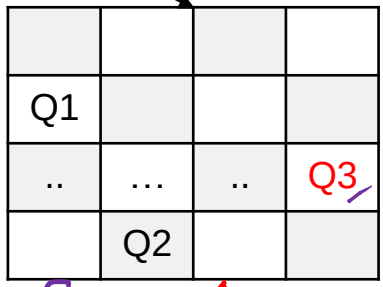
N-Queen







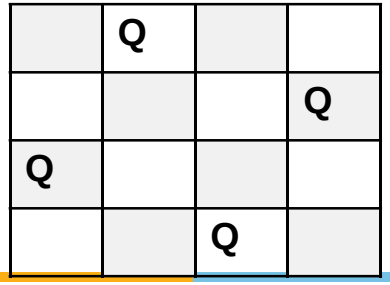


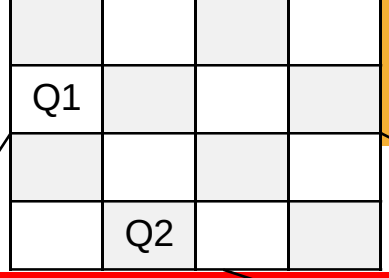


- Construct the search tree by considering one row of the board at a time
- State space graph of relaxed problem is a super graph of original state space because of removal of restrictions

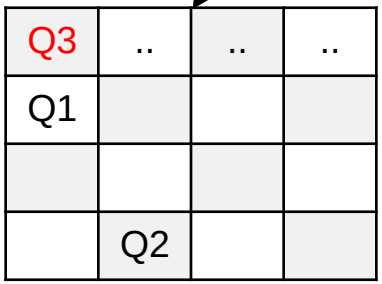
Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
< Xi , Yi >	Place in any non-occupied row in board		isValid Non-Attacking	Transition + Valid Queens	n!

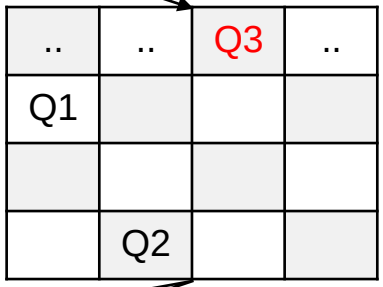
N-Queen

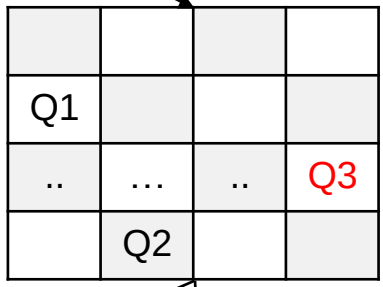


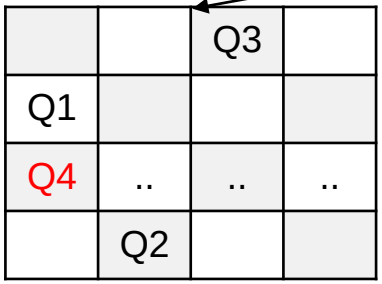


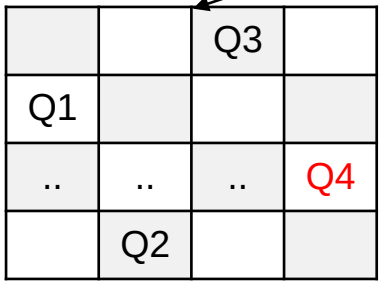
- Construct the search tree by considering one row of the board at a time
- State space graph of relaxed problem is a super graph of original state space because of removal of restrictions

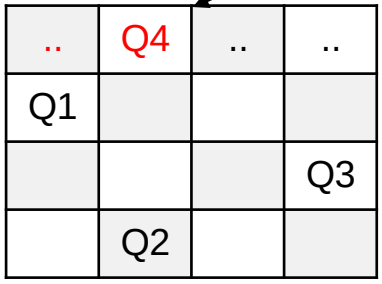


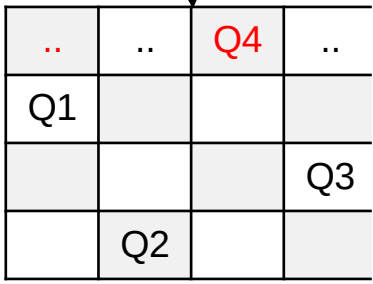




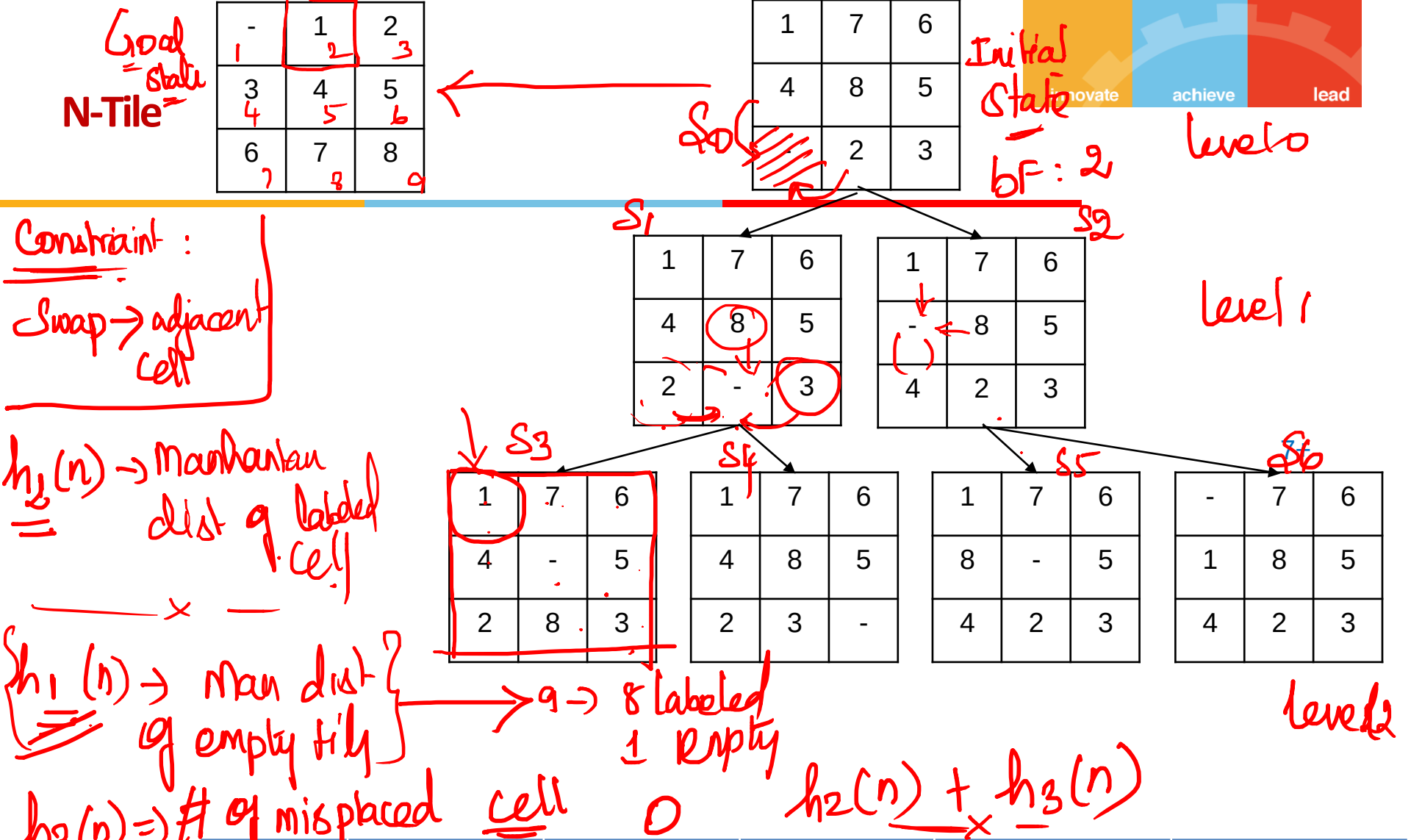






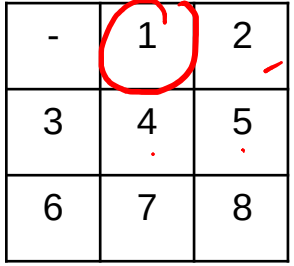


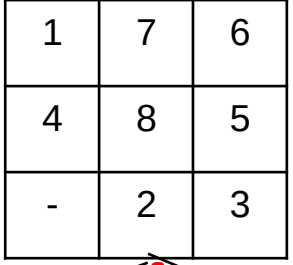
Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
< Xi , Yi >	Place in any non-occupied row in board		isValid Non-Attacking	Transition + Valid Queens	n!

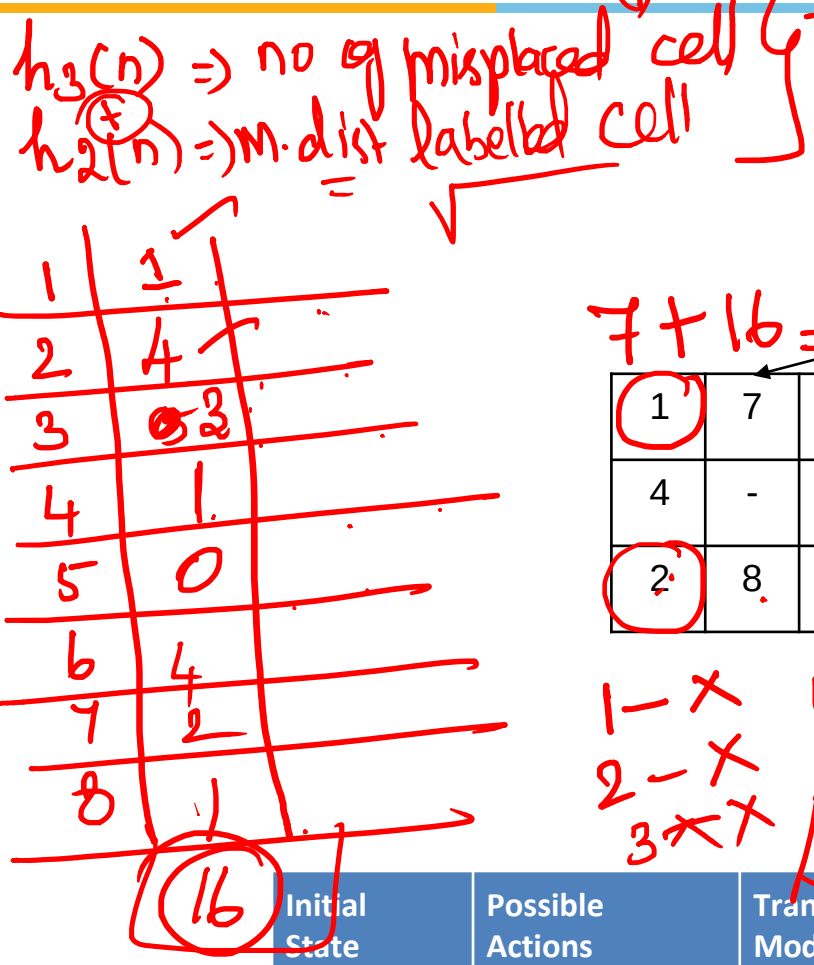


Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
<LOC, ID>	Move Empty to near by Tile		ID=LOC+1	Transition + Positional + Distance+ Other approaches	9!

N-Tile



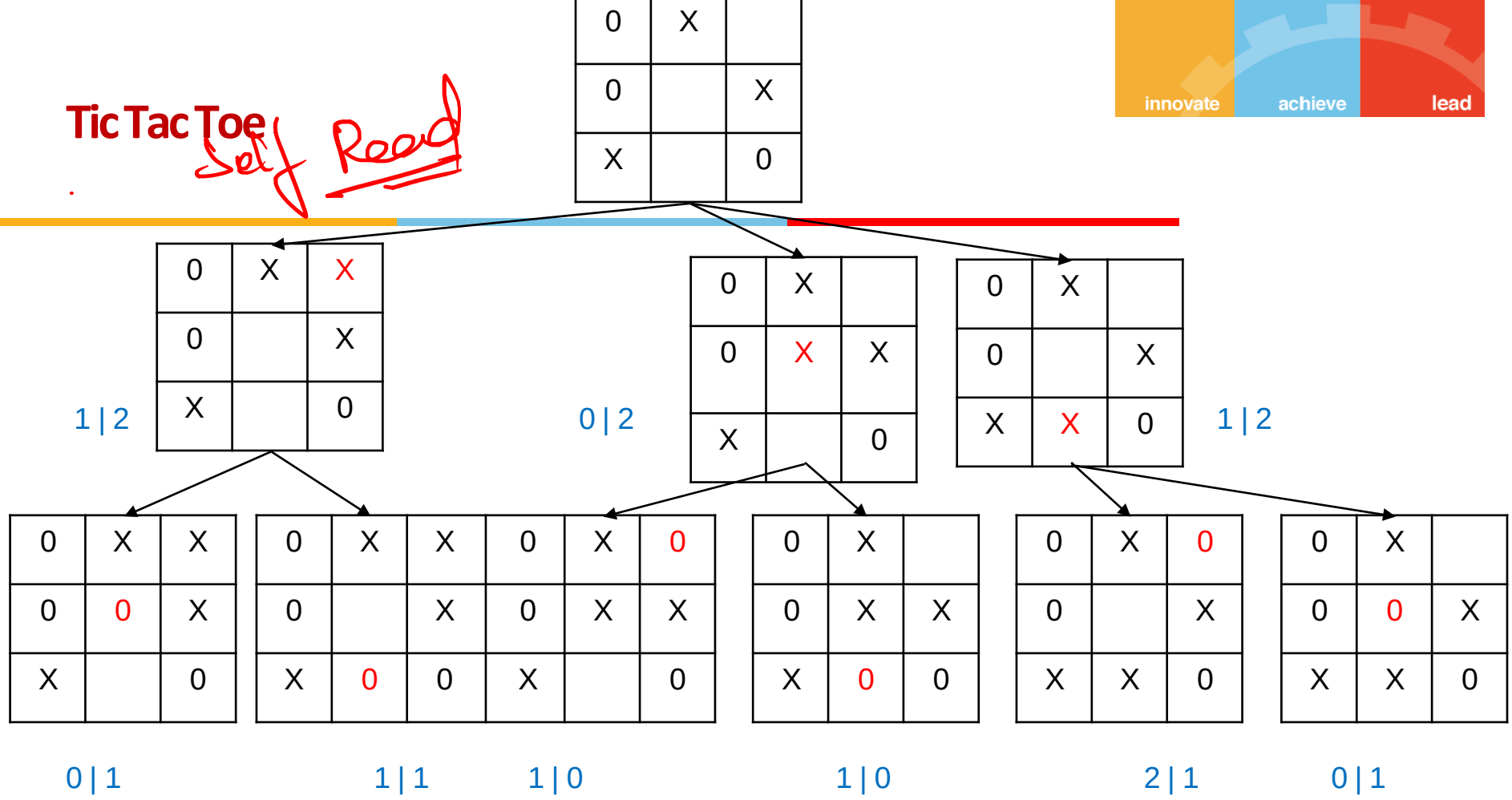




Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
<LOC, ID>	Move Empty to near by Tile		ID=LOC+1	Transition + Positional + Distance+ Other approaches	9!

Tic Tac Toe

Self Read



Opposite Win | Player Win

Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
([Xij], [Yij])	Place a coin in unoccupied (i,j)		N : i's N : j's N : i=j	No.of.Steps + Opp.Win + (N-1-Curr.Win)	$19,683=3^9$

Learn from experience

Trail / Puzzle	X1(n) : No.of.Misplaced Tiles	X2(n): Pair of adjacent tiles that are not in goal	X3(n): Position of the empty tile	……h`(n)
Example 1	7	10	7	……
Example 2	5	6	6	……
……	..	..	..	……

-	1	2
3	4	5
6	7	8

1	7	6
4	8	5
-	2	3

Create a suitable model:
h(n) = c1*X1(n) + c2*X2(n) + ……..