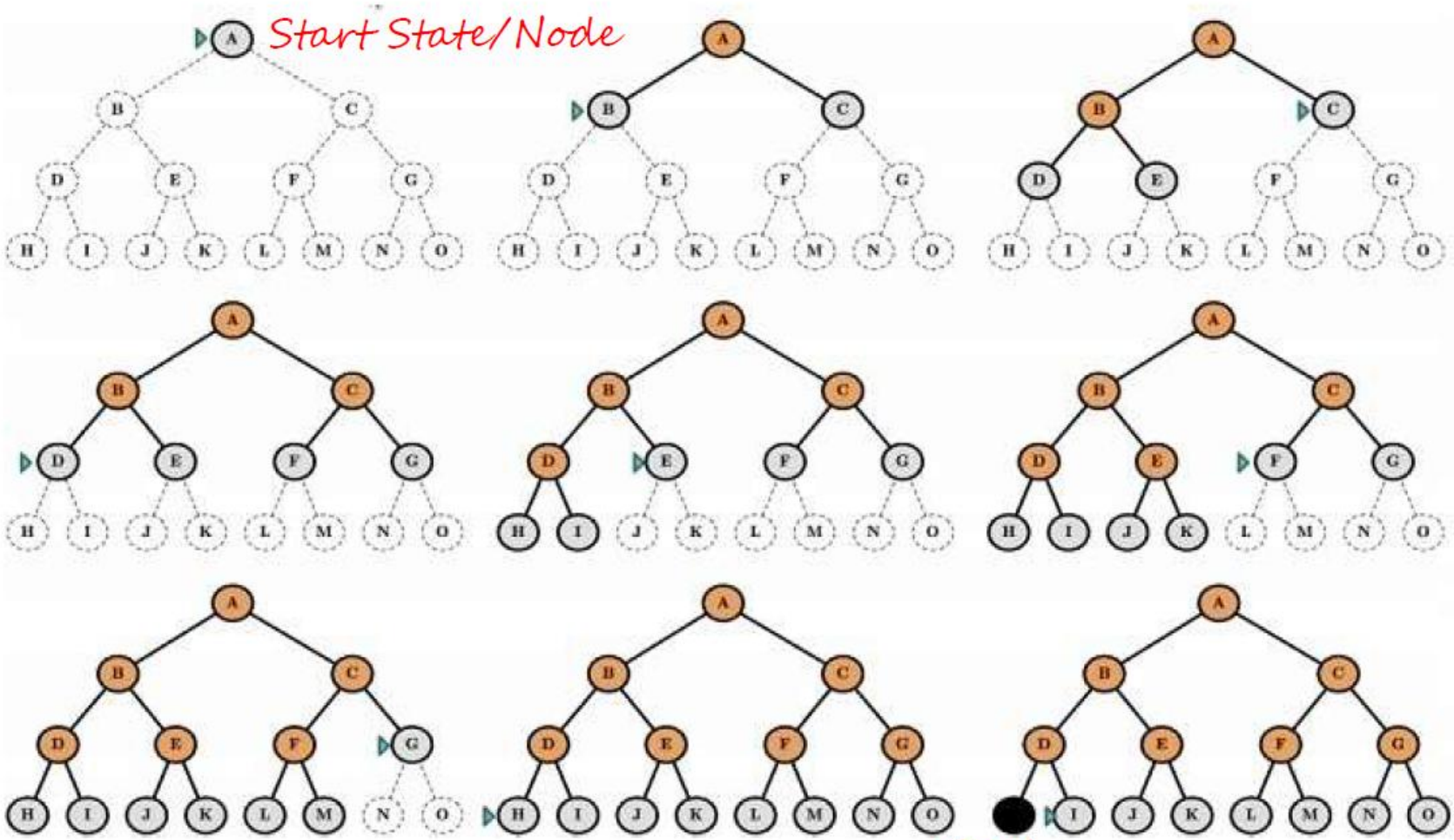
Search algorithms comparison

Algorithm	Strategy	Completeness	Optimality	Time Complexity	Space Complexity	Data Structure Used	Notes / Use Case
Depth-First Search (DFS)	Depth-first	No (infinite depth)	No	O(b^m)	O(m)	Stack (LIFO)	Fast but can get stuck in loops or infinite depth.
Breadth-First Search (BFS)	Breadth-first	Yes (if b finite)	Yes (if all costs = 1)	O(b^d)	O(b^d)	Queue (FIFO)	Guarantees shortest path if all edges have equal cost.
Uniform Cost Search (UCS)	Lowest path cost g(n)	Yes	Yes	O(b^(1 + C*/ε))	O(b^(1 + C*/ε))	Priority Queue (min-heap)	Like BFS but handles varying costs; optimal with positive costs.
Depth-Limited Search (DLS)	DFS with limit	No (if d > l)	No	O(b^l)	O(l)	Stack with depth counter	DFS with a cutoff to prevent infinite recursion.
Iterative Deepening Search (IDS)	Repeated DLS	Yes	Yes (if all costs = 1)	O(b^d)	O(d)	Stack (reused at each iteration)	Combines low space of DFS with completeness of BFS.
Greedy Best-First Search	Lowest heuristic h(n)	No	No	O(b^m)	O(b^m)	Priority Queue (min-heap on h)	Fast and goal-directed; not always optimal or complete.
A* Search	Lowest f(n) = g(n) + h(n)	Yes (if h admissible)	Yes (if h admissible)	O(b^d)	O(b^d)	Priority Queue (min-heap on f)	Optimal with good heuristic; can be memory-heavy.
Hill Climbing	Greedy local search	No	No	O(b^m)	O(1)	Single current state (variable)	May get stuck in local maxima; not complete or optimal.
Local Beam Search	Track k best states	No	No	O(k·b) per iteration	O(k)	List of top-k states	Improves on hill climbing; can explore multiple paths simultaneously.

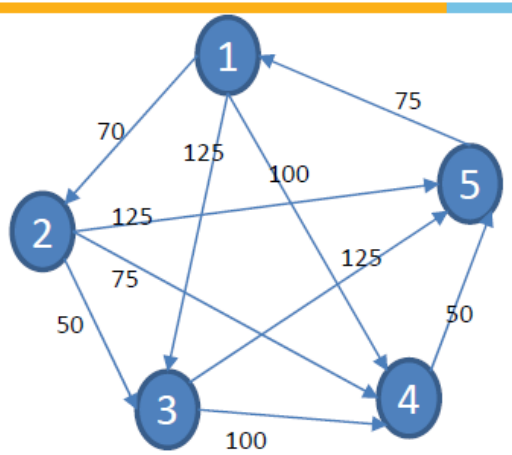
Breadth First Search (BFS)

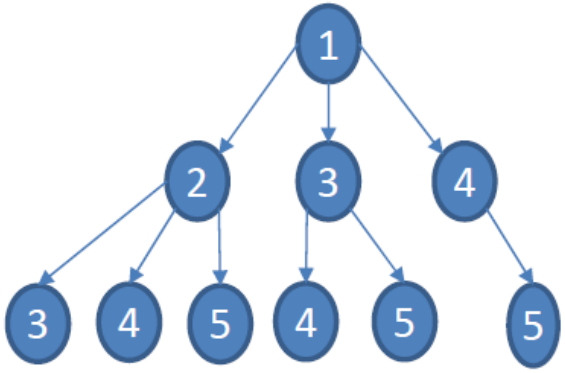


BFS – Uninformed

Start - 1

Goal - 5





(1)
(1 2) (1 3) (1 4)
TEST FAILED
:
:
(1 3) (1 4) (1 2 3) (1 2 4) (1 2 5)
.....
(1 2 3) (1 2 4) (1 2 5) (1 3 4) (1 3 5) (1 4 5)
.....
(1 2 5) (1 3 4) (1 3 5) (1 4 5)
TEST PASSED

Give the path...instead of node number

Queue - FIFO

C(1-2-5) = 70 + 125 =195
Expanded : 4
Generated : 10
Max Queue Length : 6

Breadth First Search – Evaluation

Complete – If the shallowest goal node is at a depth d, BFS will eventually find it by generating all shallower nodes

Optimal – Not necessarily. Optimal if path cost is non-decreasing function of depth of node. E.g., all actions have same cost

Time Complexity – $\mathcal{O}(b^d)$ b - branching factor, d – depth

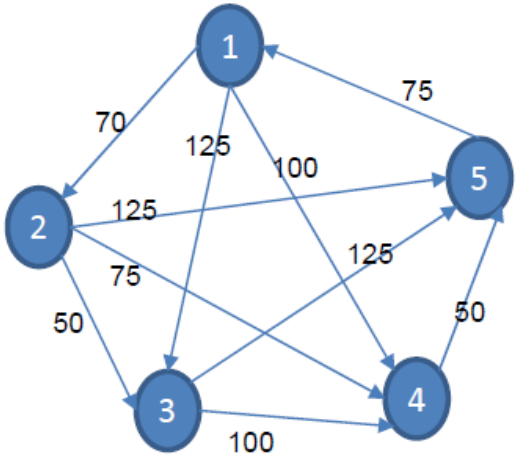
- Nodes expanded at depth 1 = b
- Nodes expanded at depth 2 = b^2
- Nodes expanded at depth d = b^d
- Goal test is applied during generation, time complexity would be $\mathcal{O}(b^{d+1})$

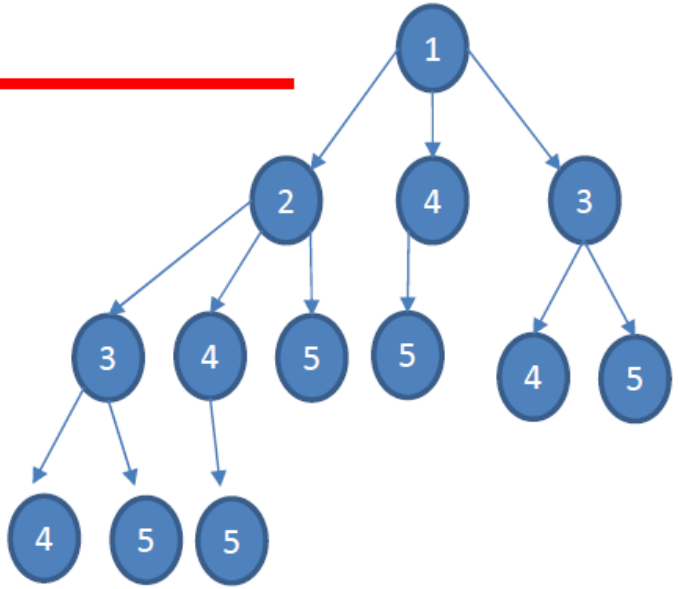
Space Complexity – $\mathcal{O}(b^d)$

- $\mathcal{O}(b^{d-1})$ in explored set
- $\mathcal{O}(b^d)$ in frontier set

innovate achieve lead

UCS





(1)
(1 2 : 70) (1 4 : 100) (1 3 : 125)
TEST-F
(1 4 : 100) (1 2 3 :120)(1 3 : 125) (1 2 4 : 145) (1 2 5 : 195)
TEST-F
(1 2 3 :120)(1 3 : 125) (1 2 4 : 145) (1 4 5: 150)(1 2 5 : 195)
TEST-F
(1 3 : 125) (1 2 4 : 145) (1 4 5: 150) (1 2 3 4:170) (1 2 5 : 195) (1 2 3 5 : 245)
TEST-F
(1 2 4 : 145) (1 4 5: 150) (1 2 3 4:170) (1 2 5 : 195) (1 3 4 : 225) (1 2 3 5 : 245) (1 3 5 : 250)
TEST-F
(1 4 5: 150) (1 2 3 4:170) (1 2 4 5 : 195) (1 2 5 : 195) (1 3 4 : 225) (1 2 3 5 : 245) (1 3 5 : 250)
TEST - P

Uniform Cost Search – Evaluation

Completeness – It is complete if the cost of every step > small +ve constant $\in$

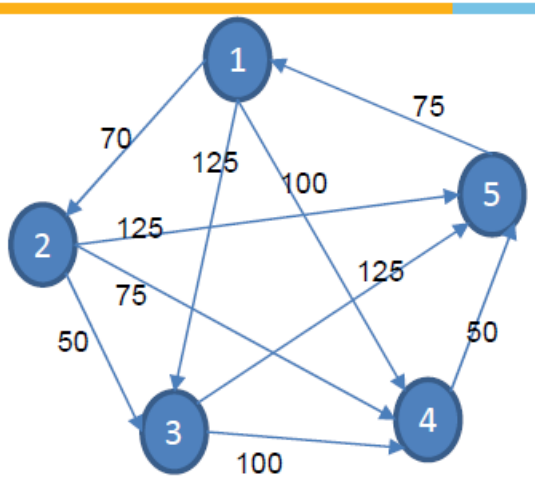
- It will stuck in infinite loop if there is a path with infinite sequence of zero cost actions

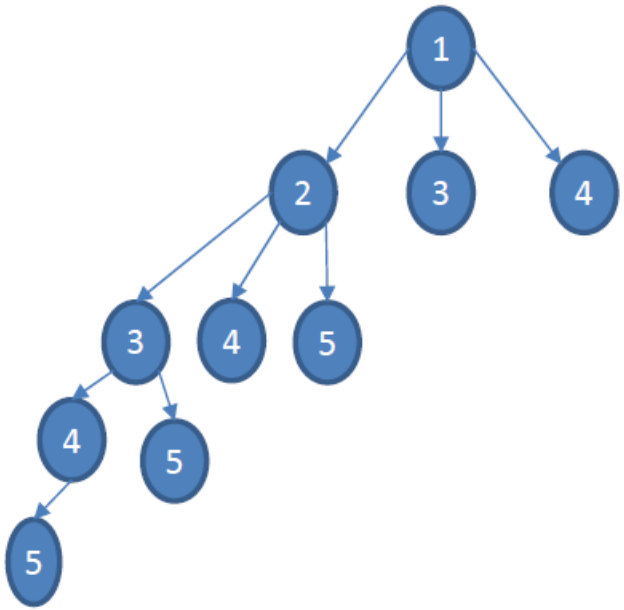
Optimal – It is Optimal. Whenever it selects a node, it is an optimal path to that node.

Time and Space complexity – Uniform cost search is guided by path costs not depth or branching factor.

- If C* is the cost of optimal solution and $\in$ is the min. action cost
- Worst case complexity = $\mathcal{O}(b^{1+\frac{C^*}{\in}})$,
- When all action costs are equal → $\mathcal{O}(b^{d+1})$, the BFS would perform better
  - As Goal test is applied during expansion, Uniform Cost search would do extra work

DFS – Uninformed





(1)
(1 2) (1 3) (1 4)
(1 2 3) (1 2 4) (1 2 5) (1 3) (1 4)
(1 2 3 4) (1 2 3 5) (1 2 4) (1 2 5) (1 3) (1 4)
(1 2 3 4 5) (1 2 3 5) (1 2 4) (1 2 5) (1 3) (1 4)

C(1-2-3-4-5) = 70 + 50 + 100 + 50 =270
Expanded : 4
Generated : 10
Max Queue Length : 6

Algorithm Tracing

Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

Iter	Open List / Frontiers / Fringes	Closed List	Goal Test
1.	(1)		Fail on (1)
2.	(1 2), (1 3), (1 4)	(1)	Fail on (1 2)

Depth First Search (DFS)

Completeness – Complete in finite state spaces because it will eventually expand every node

Optimal – Not Optimal as it would stop when the goal node is reached without evaluating if there is a better path

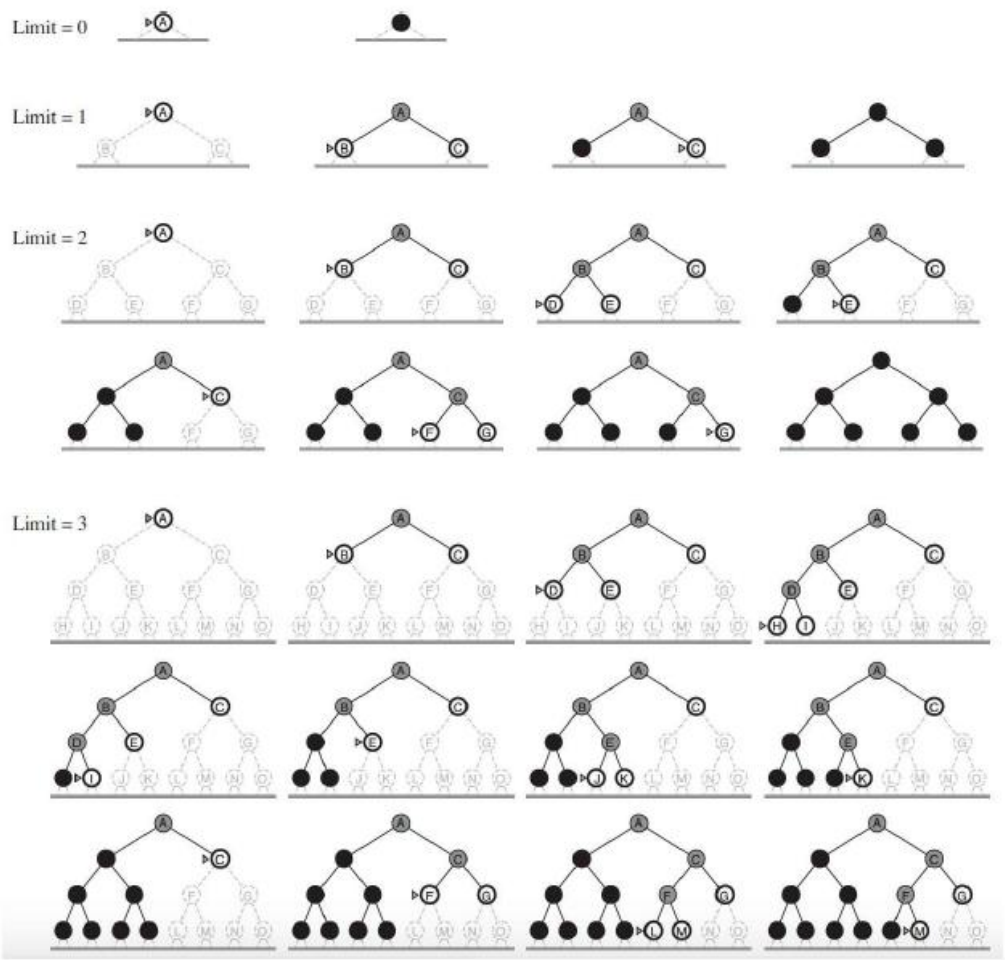
Time Complexity - $\mathcal{O}(b^m)$ where m = maximum depth of any node

- Can be much larger than the size of state space
- m can be much larger than d (shallowest goal)

Space Complexity – Needs to store only one path and unexpanded siblings.

- Any node expanded with all its children can be removed from memory
- Requires storage of only $\mathcal{O}(bm)$, b – branching factor, m - max depth

Iterative Deepening Depth First Search (IDS)



Iterative Deepening Depth First Search (IDS)

Run Depth Limited Search (DLS) by gradually increasing the limit l

- First with l=1, then l=2, l=3 and so on – until goal is found

Its is a combination of Depth First Search + Breadth First Search

Like DFS, memory requirement is a modest $\mathcal{O}(bd)$ where d is the depth of shallowest goal

Like BFS

- Complete when branching factor is finite
- Optimal when path cost is non decreasing function of depth

Iterative Deepening Depth First Search (IDS)

Time Complexity:

- Appears that IDS is generating a lot of nodes multiple times
- However, most of nodes are present in the lower levels which are not repeated often
- Generation of nodes
  - At level 1 - b nodes generated d times – (d)b
  - At level 2 – b^2 nodes generated d-1 times – (d-1)b^2
  - At level d – b^d nodes generated once – (1) b^d
- Time Complexity N(IDS) = $\mathcal{O}(b^d)$ same as BFS

IDS is the preferred uninformed search method when search space is large and depth is unknown

Greedy Best First Search

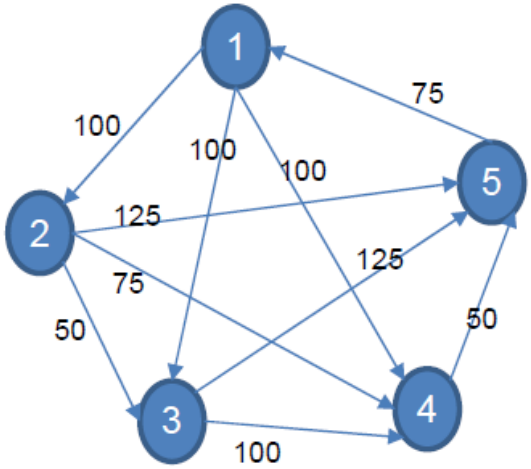
Expands the node that is closest to the goal

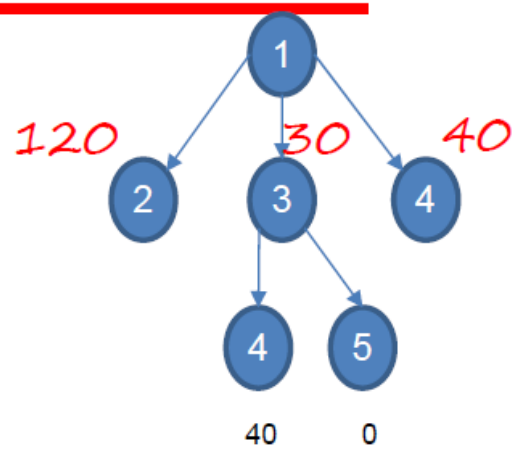
Thus, f(n) = h(n)

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Greedy Best First Search

Start state – 1
Goal state – 5





n	h(n)
1	60
2	120
3	30
4	40
5	0

(1)
(1 3) (1 4) (1 2)
(1 3 5) (1 3 4)

C(1-3-5) = 100 + 125 =225
Expanded : 2
Generated : 6
Max Queue Length : 3
Idea: Optimize DFS. Choose next nearest to goal in the same hill.

Greedy Best First Search

Not Optimal

- Because the algorithm is greedy
- It only optimizes for the current action

Not Complete

- Often ends up in state with a dead end as the heuristic doesn't guarantee a path but is only an approximation

Time and Space Complexity - $\mathcal{O}(b^m)$ where m – max depth of search tree

A* Search

Expands the node which lies in the closest path (estimated cheapest path) to the goal

Evaluation function **f(n) = g(n) + h(n)**

g(n) – the cost to reach the node

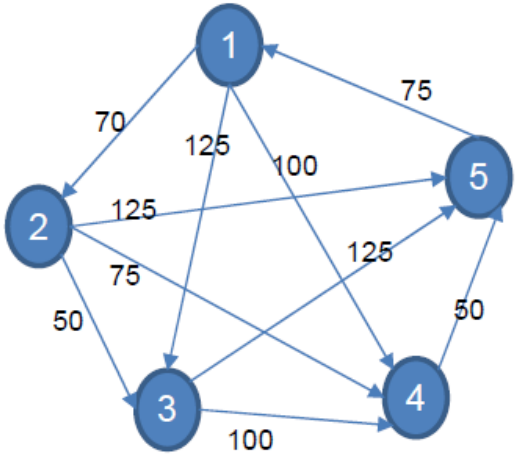
h(n) – the expected cost to go from node to goal

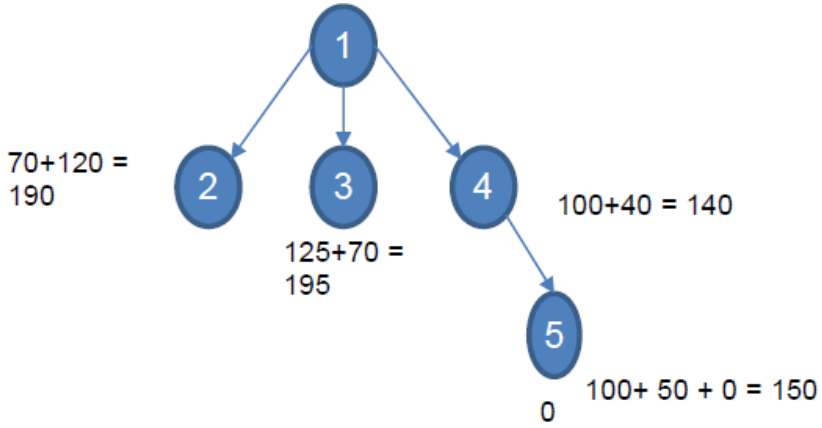
f(n) – estimated cost of cheapest path through node n

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

A* Search

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





n	h(n)
1	60
2	120
3	70
4	40
5	0

(1)
(1 4) (1 2) (1 3)
(1 4 5) (1 2) (1 3)

C(1-4-5) = 100 + 50 =150
Expanded : 2
Generated : 5
Max Queue Length : 3

A* Search

Test for Admissibility

Expands the node which lies in the closest path (estimated cheapest path) to the goal

Evaluation function **f(n) = g(n) + h(n)**

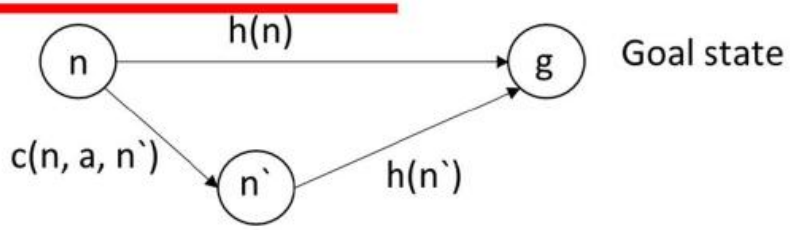
g(n) – the cost to reach the node

h(n) – the expected cost to go from node to goal

f(n) – estimated cost of cheapest path through node n

A heuristic is admissible or optimistic if , **$0 \leq h(n) \leq h^*(n)$**, where h*(n) is the actual cost to reach the goal

A* Search



Optimal on condition

h(n) must satisfies two conditions: $h(n) <= h^*(n)$ (actual cost)

- 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, $h(n) <= c(n, a, n`) + h(n`)$

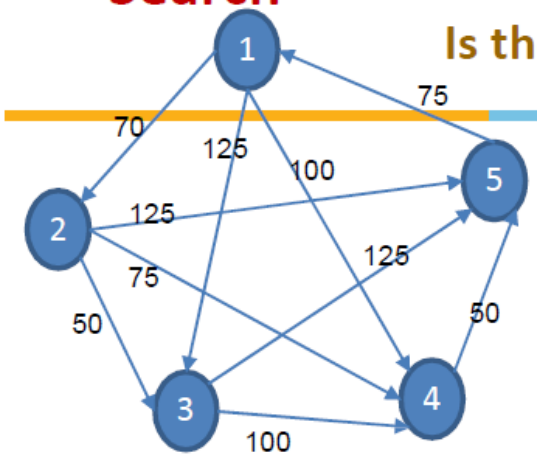
Complete

- 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 - $\mathcal{O}(b^{\Delta})$ 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

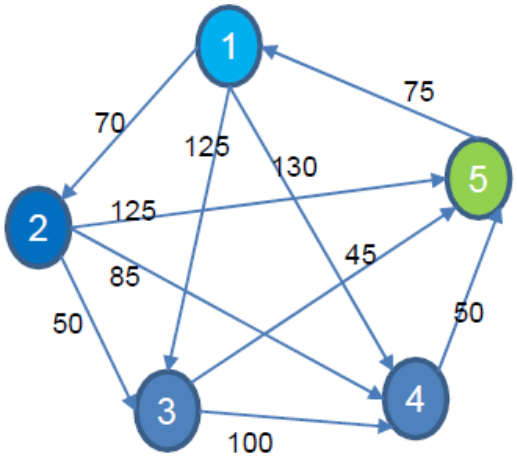
h*(n) – Actual cost

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	80 <= (70+50) : Y	N: (5->1) : 190 <= (75+80)
2	60	60 <= 50 : N	Y: (1->2) : 80 <= (70+60)
3	0	0 <= 0 : Y	
4	200	200 <= (50+75+70+50) : Y	Y: (1->4) : 80 <= (100+200) Y: (2->4) : 60 <= (75+200)
5	190	190 <= (75+70+50) : Y	Y: (2->5) : 60 <= (125+190) Y: (4->5) : 200 <= (50+190)

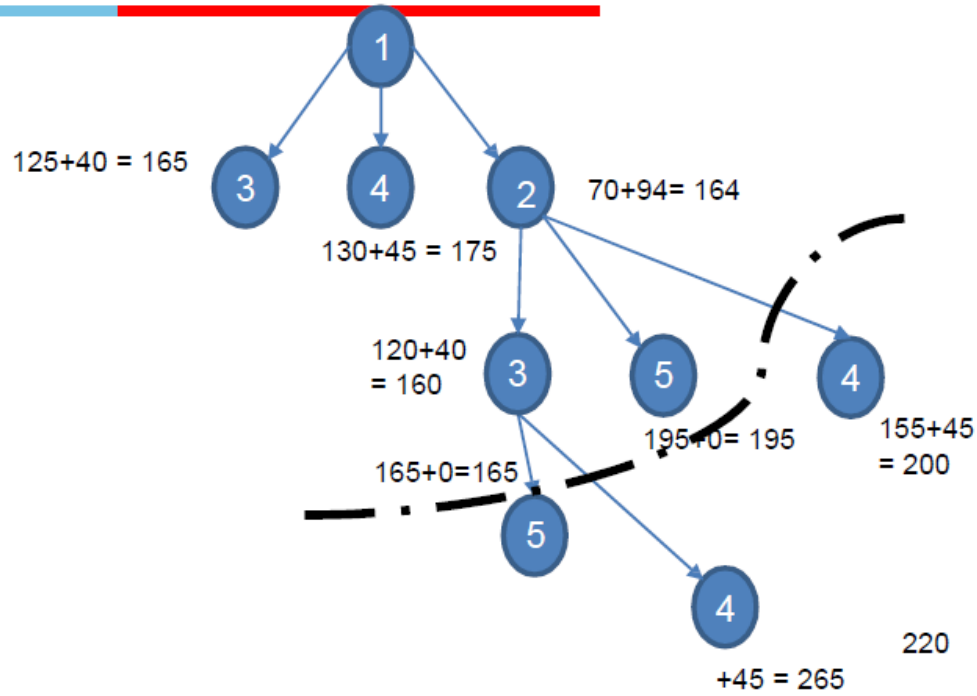
Iterative Deepening A*

innovate achieve lead

Set limit for f(n)



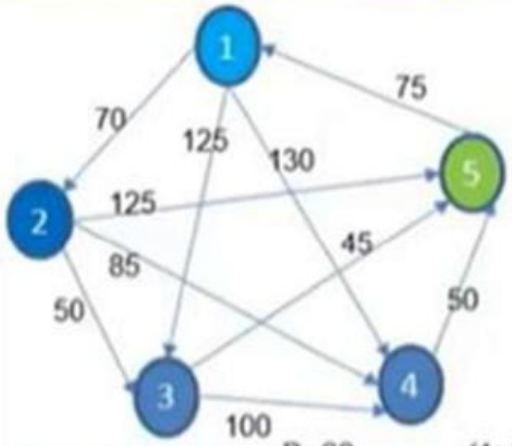
n	h(n)
1	60
2	94
3	40
4	45
5	0

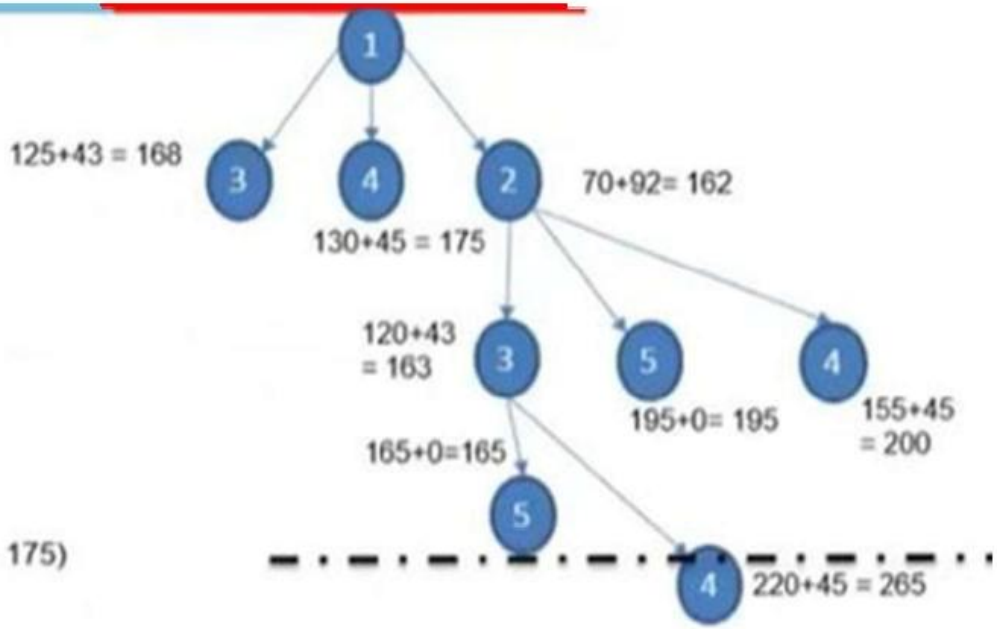


Cut off value is the smallest of f-cost of any node that exceeds the cutoff on previous iterations

Iterative Limit : e.g.,
f(n) = 180
f(n) = 195
f(n) = 200
.
.

Iterative Deepening A*





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

B=60

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

B=162

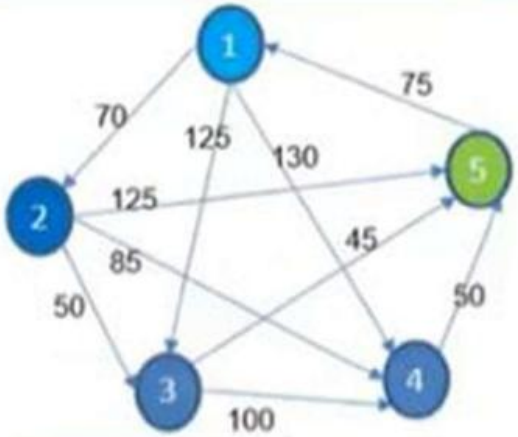
(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)

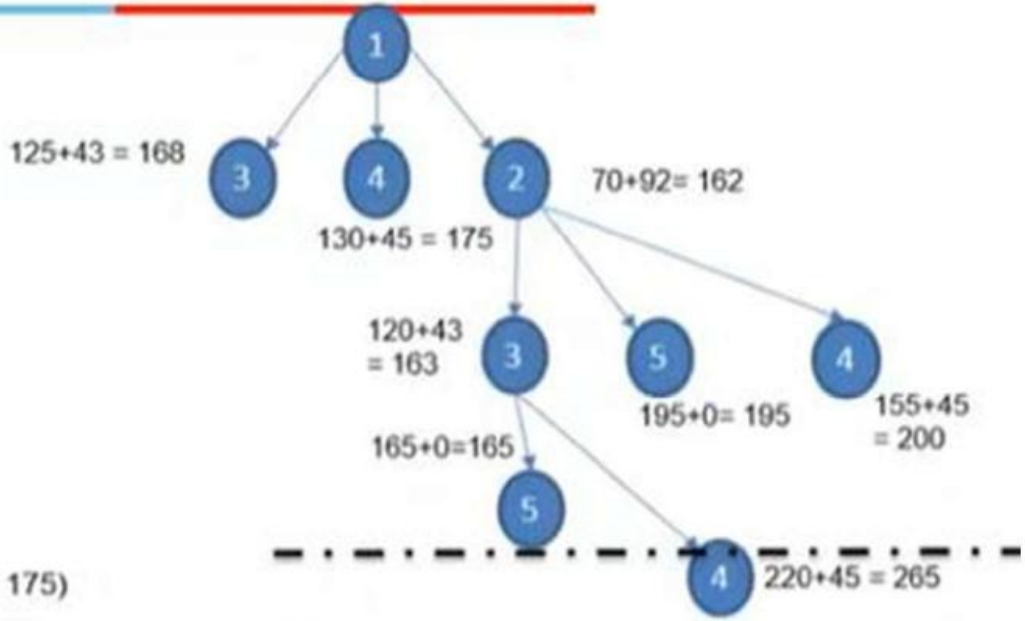
B=163

(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*

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)

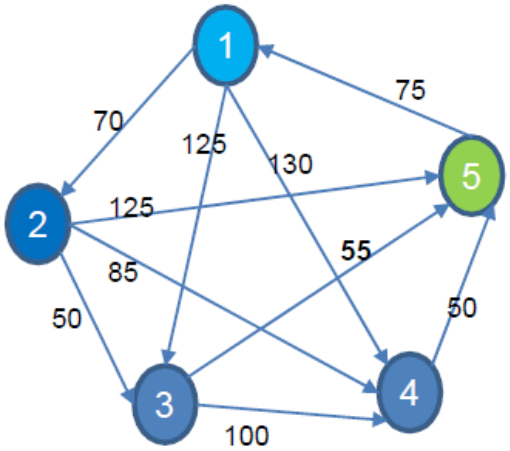
B=165

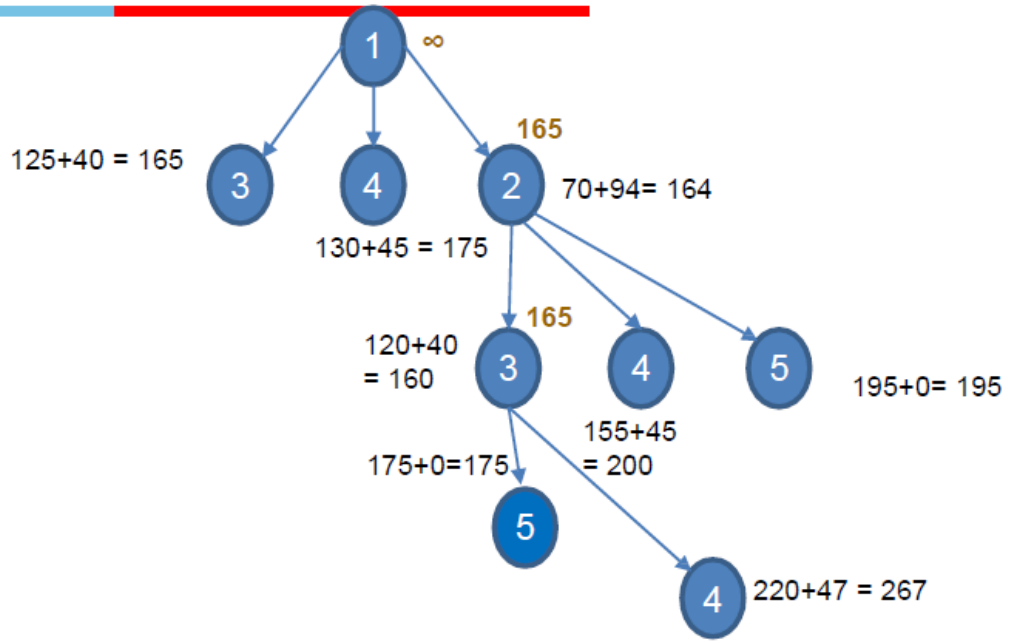
(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3 5: 165) (1 2 3 4: 265) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)

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)

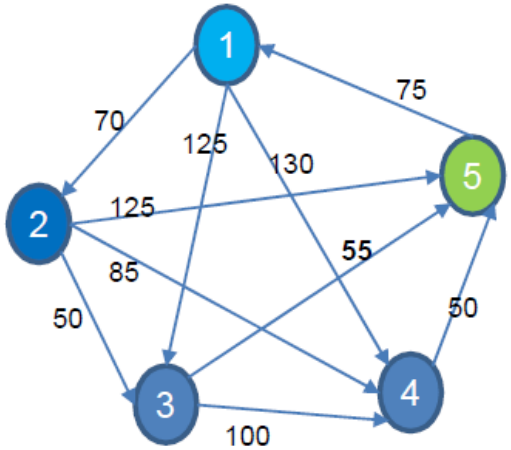
(1 2 3 5 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200) (1 2 3 4 | 267)

PASS

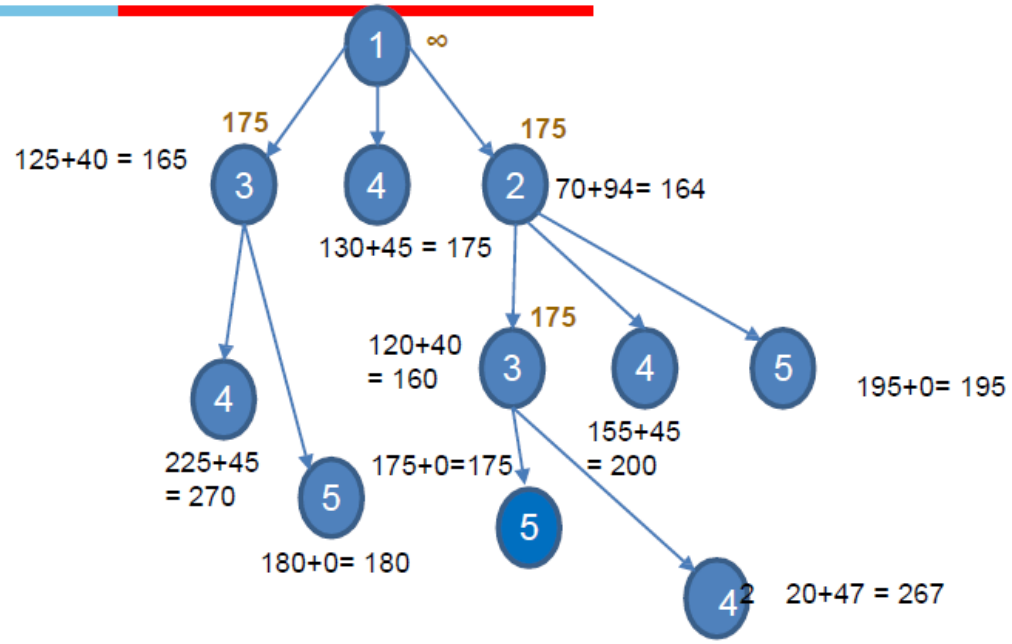
Recursive Best First Search A*

innovate achieve lead

Remember the next best alternative f-Cost to regenerate



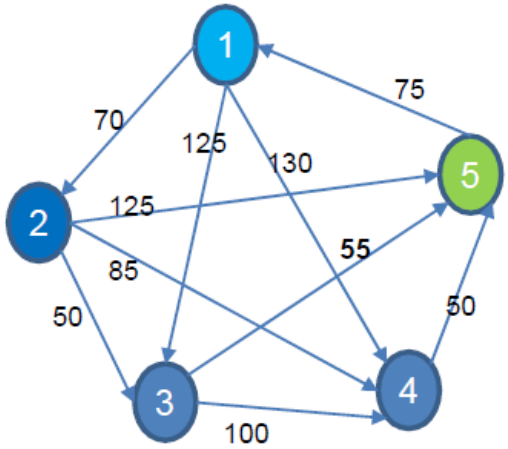
n	h(n)
1	60
2	94
3	40
4	45
5	0



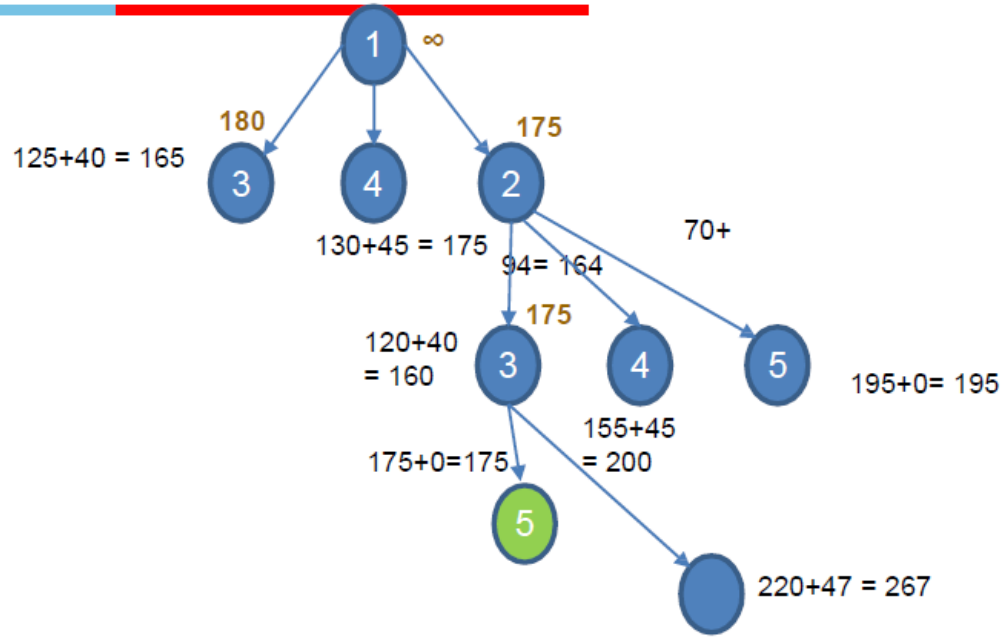
Recursive Best First Search A*

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

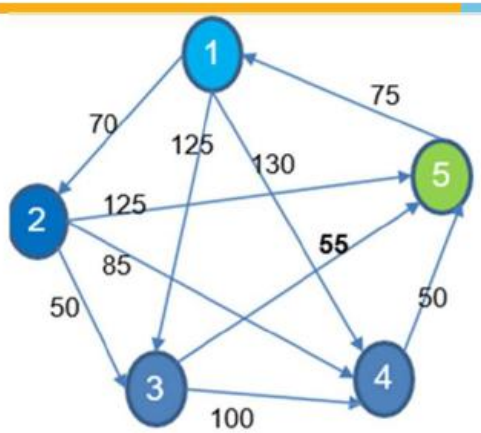
Continue expansion

Space Usage = O(bd) very less

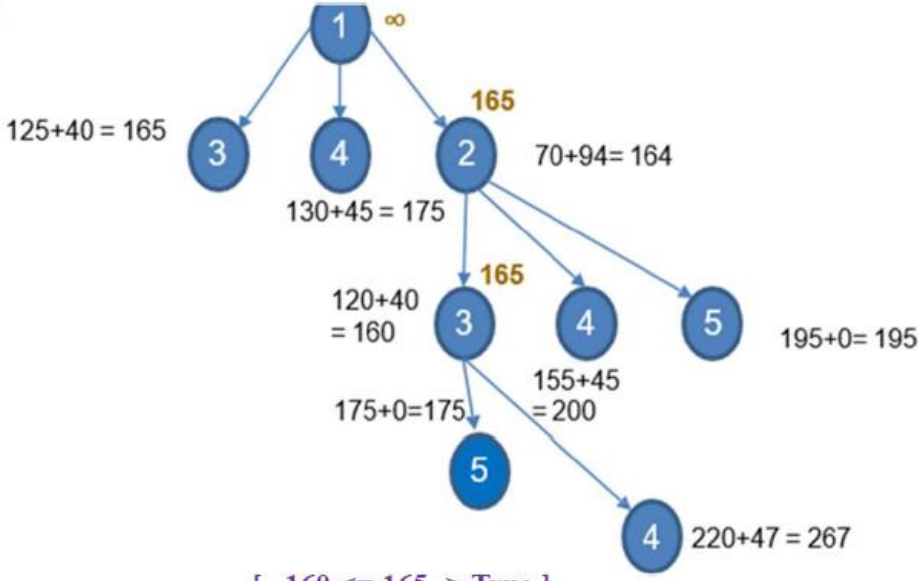
Recursive Best First Search A*

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 3 | 160) (1 3 | 165) (1 4 | 175) (1 2 5 | 195) (1 2 4 | 200) [160 <= 165 -> True]

(1 2 3 5 | 175) (1 3 | 165) (1 4 | 175) (1 2 5 | 195) (1 2 4 | 200) (1 2 3 4 | 265) [175 <= 165 -> False]

(1 3 | 165) (1 2 | 175) (1 4 | 175) [165 <= 175 -> True]

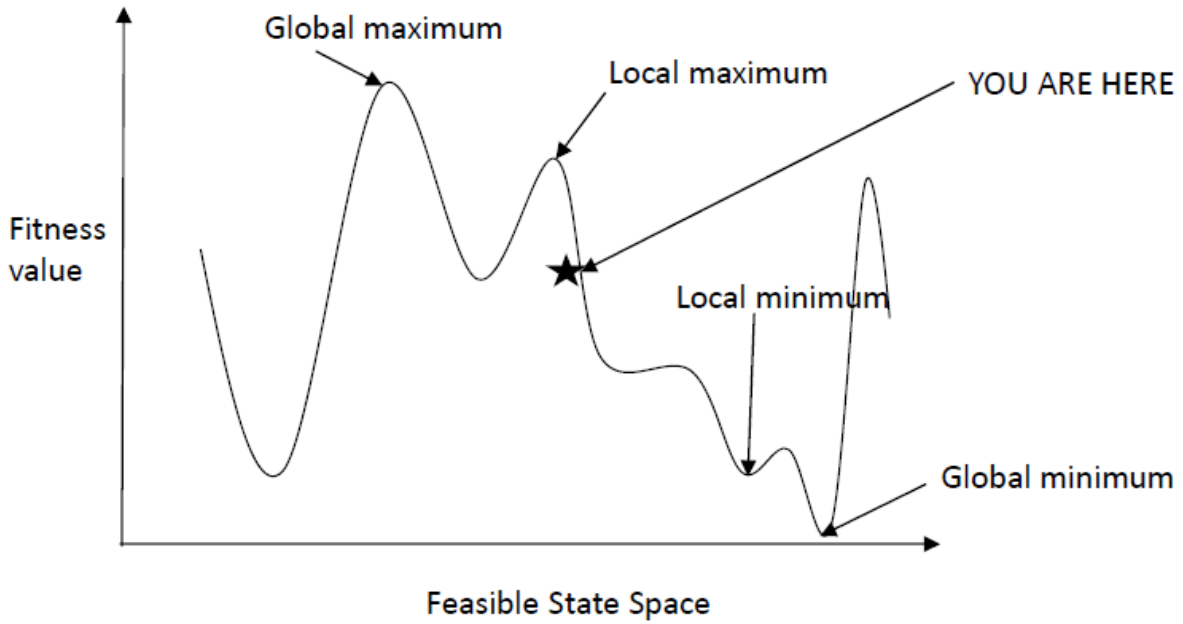
(1 3 5 | 180) (1 2 | 175) (1 4 | 175) (1 3 4 | 270) [185 <= 175 -> False]

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

(1 2 3 | 160) (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)

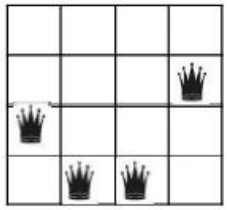
Hill Climbing



Hill Climbing

Random Restart

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Select the next state based on the highest fitness
4. Repeat from Step 2



3	4	4	2	3

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

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

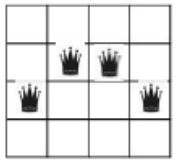
Simulated Annealing Algorithm

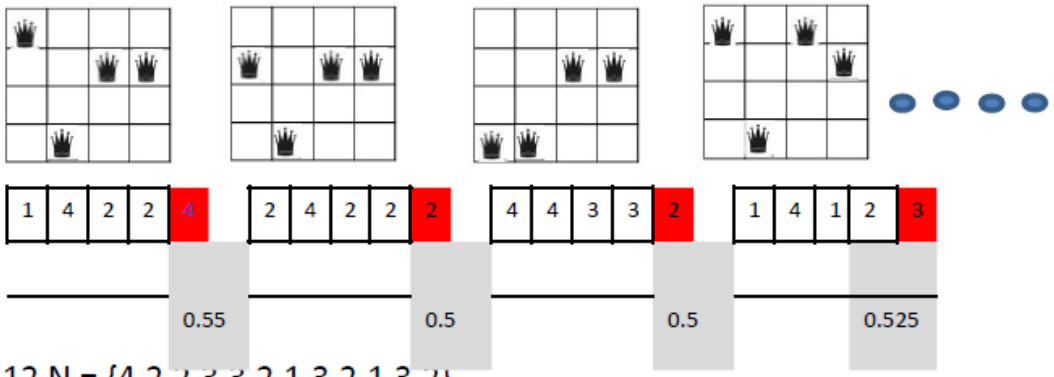
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← next.VALUE – current.VALUE
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

Simulated Annealing

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2





12 N = {4,2,2,3,3,2,1,3,2,1,3,2}

Init = 2

Simulated Annealing

Current Value = 4 (Local Maxima)

Global Maxima = 6

		t = 20			t = 5		
Next Value	ΔE	ΔE/t	$e^{\Delta E/t}$	$\frac{1}{1+e^{\Delta E/t}}$	ΔE/t	$e^{\Delta E/t}$	$\frac{1}{1+e^{\Delta E/t}}$
2	2	0.1	1.12	0.47	0.4	1.49	0.40
3	1	0.05	1.05	0.49	0.2	1.22	0.45
5	-1	-0.05	0.95	0.51	-0.2	0.82	0.55

Beam Search

1. Initialize k random state
2. Evaluate the fitness scores for all the successors of the k states
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. If the goal is not found, Select the next ‘k’ states randomly based on the probabilit
6. Repeat from Step 2