Heuristic Search Techniques

N Geetha
A M & C S
P S G College of Technology

Informed (heuristic search)

- Large branching factors are a serious issue
- We must find a way to reduce the number of visited nodes
- Informed (heuristic) search proposes methods to help us choose smartly the nodes to expand
- It uses problem-specific knowledge beyond the definition of the problem itself
- This information helps to find solution more efficiently
- The information concerns the regularities of the state space

Heuristic

- root from Greek word eurisko = I discover / find
- Derived word is heuriskein

- A rule of thumb, simplification or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood
- Heuristics are based on intuition or experience
- Heuristics are formalized as rules to choose those branches in a state space that are most likely to lead to an acceptable solution
- Addnl. knowledge is imported into the search algorithm through heuristics
- heuristics do not guarantee feasible solutions
- usually work and find a good enough (if not optimal) solution most of the time

Blind vs. heuristic strategies

- **Uninformed (or blind) strategies**
  - They treat all the problems in the same way
  - They only exploit the positions of the nodes in the search tree
- **Informed (heuristic) strategies**
  - They use knowledge about the problem
  - The most “promising” nodes are placed at the beginning of the fringe
  - Employed in two situations
    - When a problem does not have an exact solution because of inherent ambiguities in the problem statement or available data
    - When a problem has an exact solution but the computational cost is combinatorially explosive. Uninformed search may fail to find a solution within any practical length of time

Heuristic Search

- It exploits state description to estimate how "good" each node is
- An evaluation function h maps each node n of the search tree to a real number $h(n) \geq 0$
  - $H : S \rightarrow R^+$ (S is the set of nodes)
  - Traditionally, h(n) is an estimated cost
  - The smaller h(n) is, the more promising 'n' is
  - refers to the estimated goodness of successor nodes
  - A good heuristic is optimistic, well informed and simple to compute
- Search sorts the fringe in increasing order of h
  - Random order is assumed among nodes with equal h

Better h means better search

- When h = cost to the goal
  - Only nodes on correct path are expanded
  - Optimal solution is found
- When h < cost to the goal
  - Additional nodes are expanded
  - Optimal solution is found
- When h > cost to the goal
  - Optimal solution can be overlooked

Admissible heuristics

- Let $h^*(n)$ be the cost of the optimal path from 'n' to a goal node
- The heuristic function $h(n)$ is *admissible* if:

$$0 \leq h(n) \leq h^*(n)$$

If G is a goal, then $h(G) = 0$

- An admissible heuristic function is always *optimistic* (never overestimates)

Creating an admissible h

- An admissible heuristic can usually be seen as the cost of an optimal solution to a *relaxed* problem (one obtained by removing constraints)
- In robot navigation:
  - The Manhattan distance corresponds to removing the obstacles
  - The Euclidean distance corresponds to removing both the obstacles and the constraint that the robot moves on a grid

Heuristics for 8-puzzle I

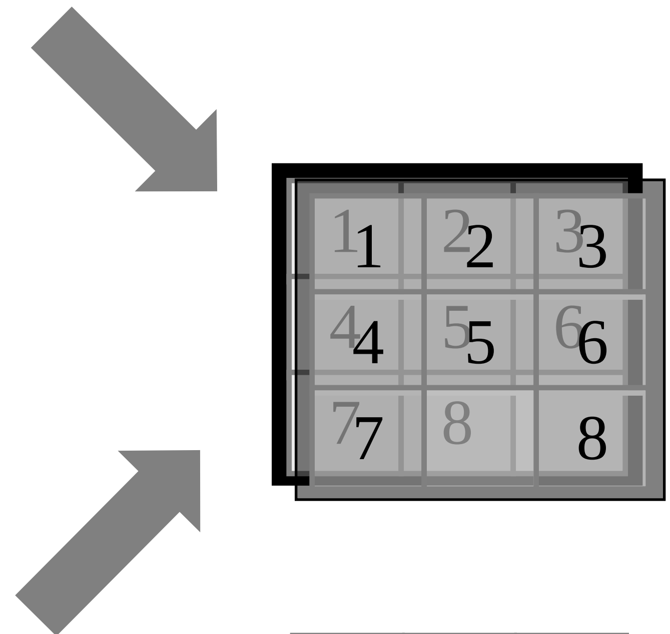
The number of misplaced tiles (not including the blank)

Current State

1	2	3
4	5	6
7		8

Goal State

1	2	3
4	5	6
7	8	



N	N	N
N	N	N
N	Y	

In this case, only “8” is misplaced, so the heuristic function evaluates to 1.

In other words, the heuristic is *telling* us, that it *thinks* a solution might be available in just 1 more move.

h(current state) = 1

Heuristics for 8-puzzle II

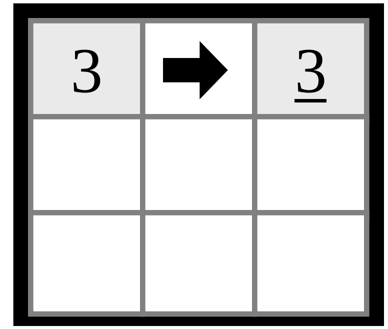
Current State

3	2	8
4	5	6
7	1	

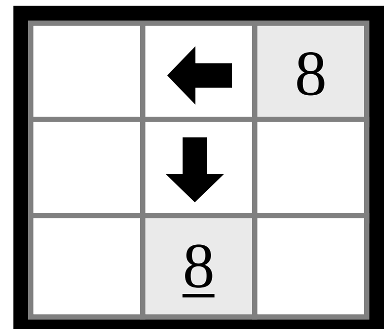
Goal State

1	2	3
4	5	6
7	8	

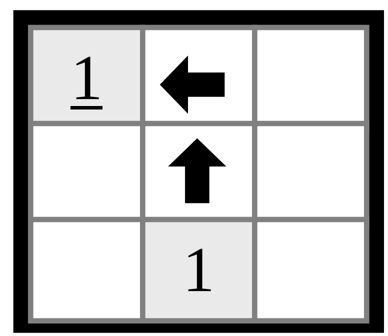
The Manhattan Distance (not including the blank)



2 spaces



3 spaces



3 spaces

Total 8

In this case, only the “3”, “8” and “1” tiles are misplaced, by 2, 3, and 3 squares respectively, so the heuristic function evaluates to 8.

In other words, the heuristic is *telling* us, that it *thinks* a solution is available in just 8 more moves.

$h(\text{current state}) = 8$

How to construct *h* ?

- Commonly used functions
  - **g(n)** is the cost of the path from the initial node to ‘n’
    - It is known
  - **f'(n)** is an estimate of the cost of a path from n to a goal node
    - It is a heuristic estimate
- Hill Climbing
  - **h(n) = f’(n)**
- Greedy Best First search
  - **h(n) = f’(n)**
- A search
  - **h(n) = g(n) + f’(n)**
- Using only *g* is equivalent to uninformed search
- Main problem: how to choose *the most helpful* *h* function?

Hill Climbing Search

- **Hill Climbing**

- Is a variant of generate-and test
- feedback from the test procedure is used to help the generator decide which direction to move in search space.
- The test function is augmented with a heuristic function that provides an estimate of how close a given state is to the goal state.
- Computation effort of heuristic function is negligible.
- is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available.

Hill Climbing: Algorithm

- **/* Let h'(n) be the inferred value of the cost from node n of the state space to the goal state */**

  S1. n = start state

  S2. Loop : If goal(n) then exit(success).

  S3. Expand n; Compute h'(n_i) for all child nodes n_i of n and take the child node which gives the minimum value for its cost. Call the child node as next_n.

  S4. If h'(n) < h'(next_n), exit (failure).

  S5. n = next_n.

  S6. Go to Loop.

Hill Climbing

This simple policy has three well-known drawbacks:

1. **Local Maxima**: a local maximum as opposed to global maximum.

2. **Plateaus**: An area of the search space where evaluation function is flat, thus requiring random walk.

3. **Ridge**: Where there are steep slopes and the search direction is not towards the top but towards the side.

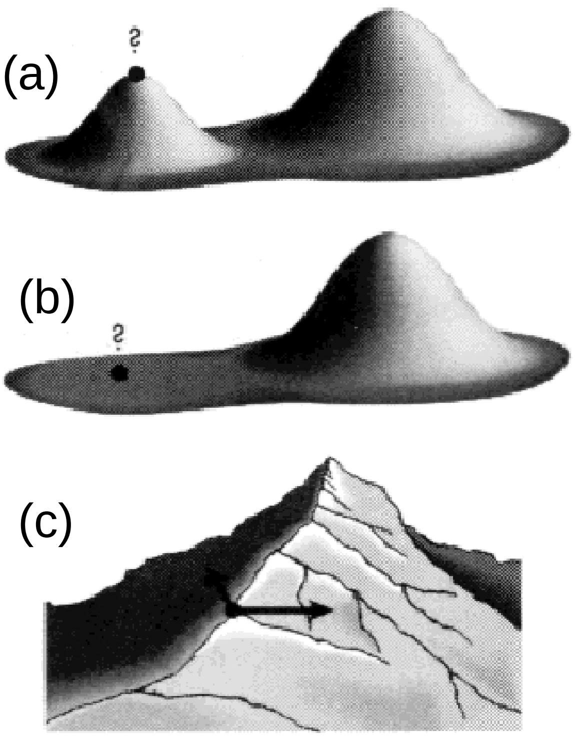


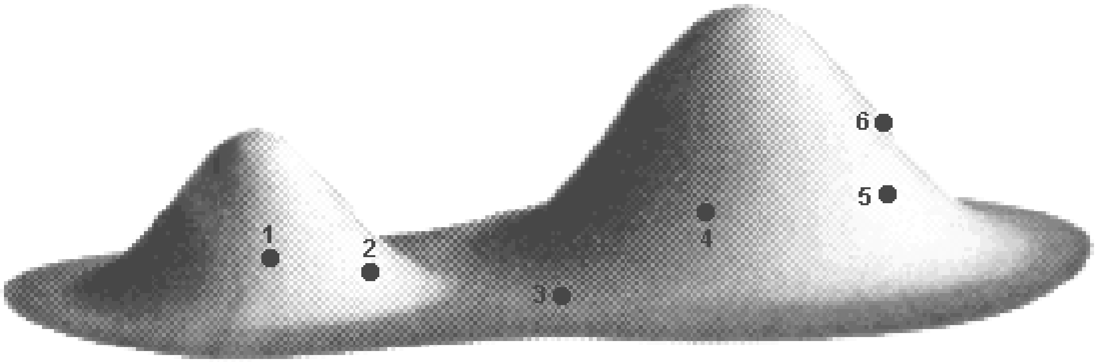
Figure: Local maxima, Plateaus and ridge situation for Hill Climbing

Overcoming drawbacks:

- Hill climbing a local search method which only looks at the ‘immediate’ consequences of its choices
- **To deal local maxima:** Backtrack to some earlier node and try going in a different direction (need to maintain visited nodes to use when the nodes visited leads to a dead end)
- **To deal with plateau :** Make a good jump in some direction to try to get a new section of the search space
- **To deal with ridge :** Apply 2 or more rules before applying the test. This corresponds to moving in several directions at once.

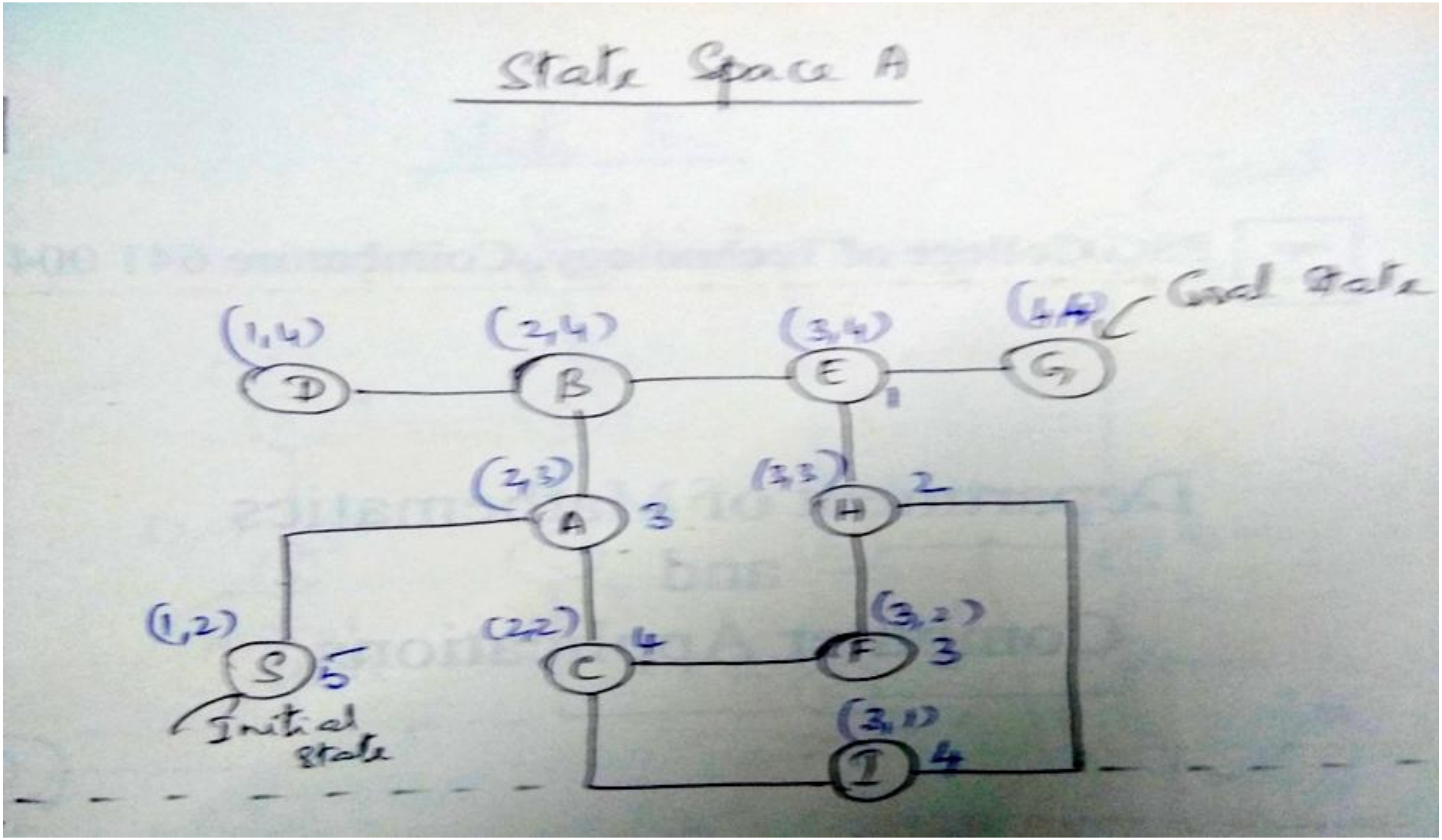
Hill-climbing

- In each of the previous cases (local maxima, plateaus the algorithm reaches a point at which no progress is being made.
- A solution is to do a **random-restart hill-climbing** - where random initial states are generated, running each until it halts or makes no discernible progress. The best result is then chosen.



Random-restart hill-climbing (6 initial values)

- A alternative to a random-restart hill-climbing when stuck on a local maximum is to do a '**reverse walk'** to **escape the local maximum**.
- This is the idea of simulated annealing.



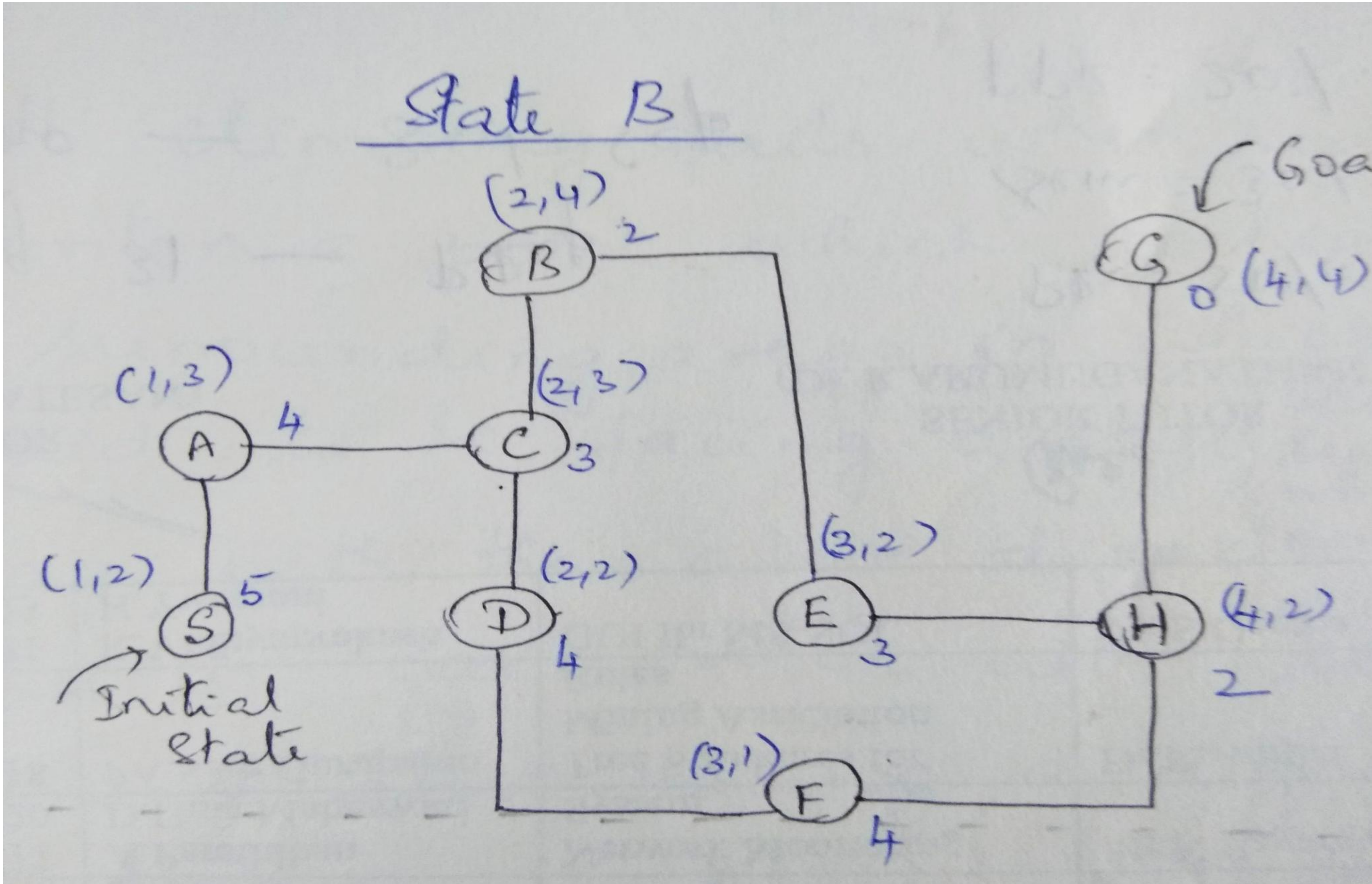
- h’(n) = | x-coord of ‘G’ – x-coord of ‘n’ | + | y-coord of ‘G’ – y-coord of ‘n’|

Path = SABEG

Cost = 2 + 1 + 1 + 1 = 5

n	n_i	next_n	$h'(n) > h'(n_i)$	Remarks
				n = S
S^5	A^3	A^3	$h'(S) > h'(A)$	n = A
A^3	B^2 C^4	B^2	$h'(A) > h'(B)$	n=B
B^2	A^3 D^2 E^1	E^1	$h'(B) > h'(E)$	n=E
E^1	B^2 H^2 G^0	G^0	$h'(E) > h'(G)$	n=G
G^0				Goal reached

Example 2



State B
(2,4)
B
2
Goa
G
0
(4,4)
(1,3)
A
4
(2,3)
C
3
(3,2)
(1,2)
S
5
(2,2)
D
4
E
3
H
(4,2)
2
Initial state
(3,1)
F
4

Measures

- ‘b’ is the branching factor
- Complete ?   No
- Optimal ? No
- Time Complexity : O(b)
- Space Complexity : O(1)

Best First Search

- **Procedure Best_First_Search**
- /* h'(n) : inferred cost from node n of the state space to the goal state */
- /* OPEN is a priority queue and CLOSED is a list */

S1. Put start state S into OPEN.

S2. Loop : If empty(OPEN) then exit(failure).

S3. n = first(OPEN)

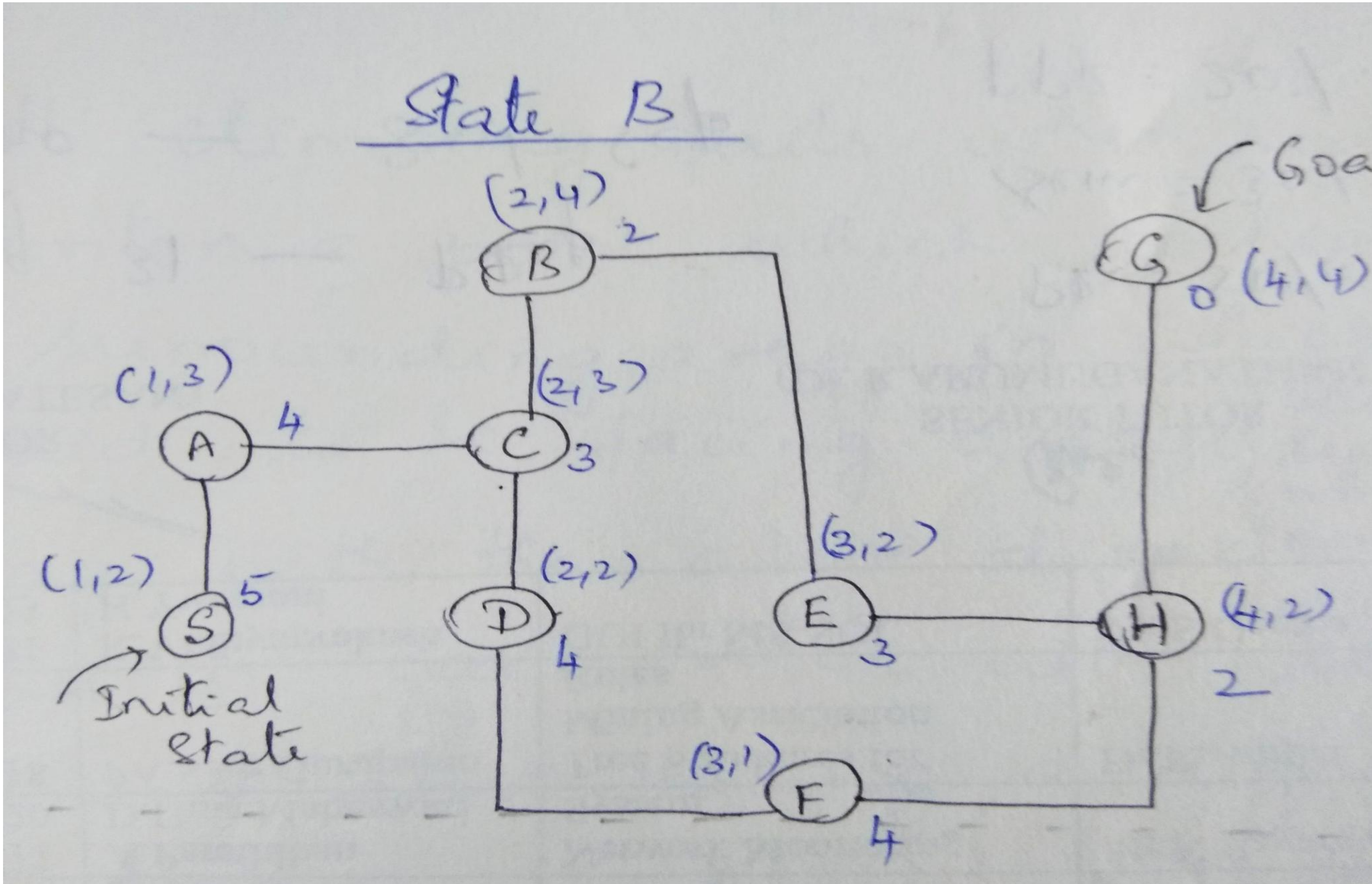
S4. If goal(n) then exit(success)

S5. remove(n, OPEN). Add(n, CLOSED)

S6. Expand n and generate all child nodes of n and only put those into OPEN that are neither in OPEN nor in CLOSED; Link each of these to node n. Compute h'(n) and list the nodes in OPEN in the order of the smallest h'(n).

S7. Go to Loop.

Example 2


State B
(2,4)
B
2
G
Goa
0
(4,4)
(1,3)
A
4
(2,3)
C
3
(3,2)
(1,2)
S
5
(2,2)
D
4
E
3
H
(4,2)
2
Initial
state
(3,1)
F
4

Path = SAC..

Cost =

n	OPEN	CLOSED	Remarks
	S^5		Initialise OPEN
S^5		S^5	Goal(S) is false
	A^4	S^5	Expand S
A^4		S^5A^4	Goal(A) is false
	C^3	S^5A^4	Expand A
C^3		$S^5A^4C^3$	Goal(C) is false
	B^2 D^4		Expand C
..			

Measures

- ‘b’ is the branching factor
- Complete ?   No; greedy best-first search can start down an infinite path and never return to try other possibilities,
- Optimal ? No; best-first search resembles depth-first search in the way it prefers to follow a single path all the way to the goal, but will back up when it hits a dead end
- Time Complexity : $O(b^m)$
- Space Complexity : $O(b^m)$

A Algorithm

- Algorithm ***A***.

  If the evaluation function h’(n) = g(n) + f’(n) is used with the best_first_search algorithm, the result is called ***Algorithm A***, where,

  - **n** is any state encountered in the search,
  - **g(n)** is the cost of n from the start state (e.g., measures the depth at which the state has been found), and
  - **f’(n)** is the heuristic estimate of the cost of going from n to a goal.

- Thus **h’(n)** estimates the *total* cost of the path from the start state through n to the goal state.

A Algorithm

- **Procedure A_Algorithm**
- /* g(n) : computed minimum cost of path from start state to node n;
- f'(n) : inferred cost from node n to the goal state */
- /* OPEN is a priority queue and CLOSED is a list */

S1. Put start state S into OPEN.

$$h'(S) = g(S) + f'(S) = 0 + f'(S) = f'(S)$$

S2. Loop : If empty(OPEN) then exit(failure).

S3. n = first(OPEN)

S4. If goal(n) then exit(success)

S5. remove(n, OPEN). Add(n, CLOSED)

S6. Expand n; For all child nodes n_i, compute $h'(n_i) = g(n_i) + h'(n_i)$ using the exact cost of the node n_i from S and the estimated cost of n_i with the goal node.

A Algorithm contd.

S6. – Put nodes neither in OPEN nor in CLOSED into OPEN and set pointer to n.

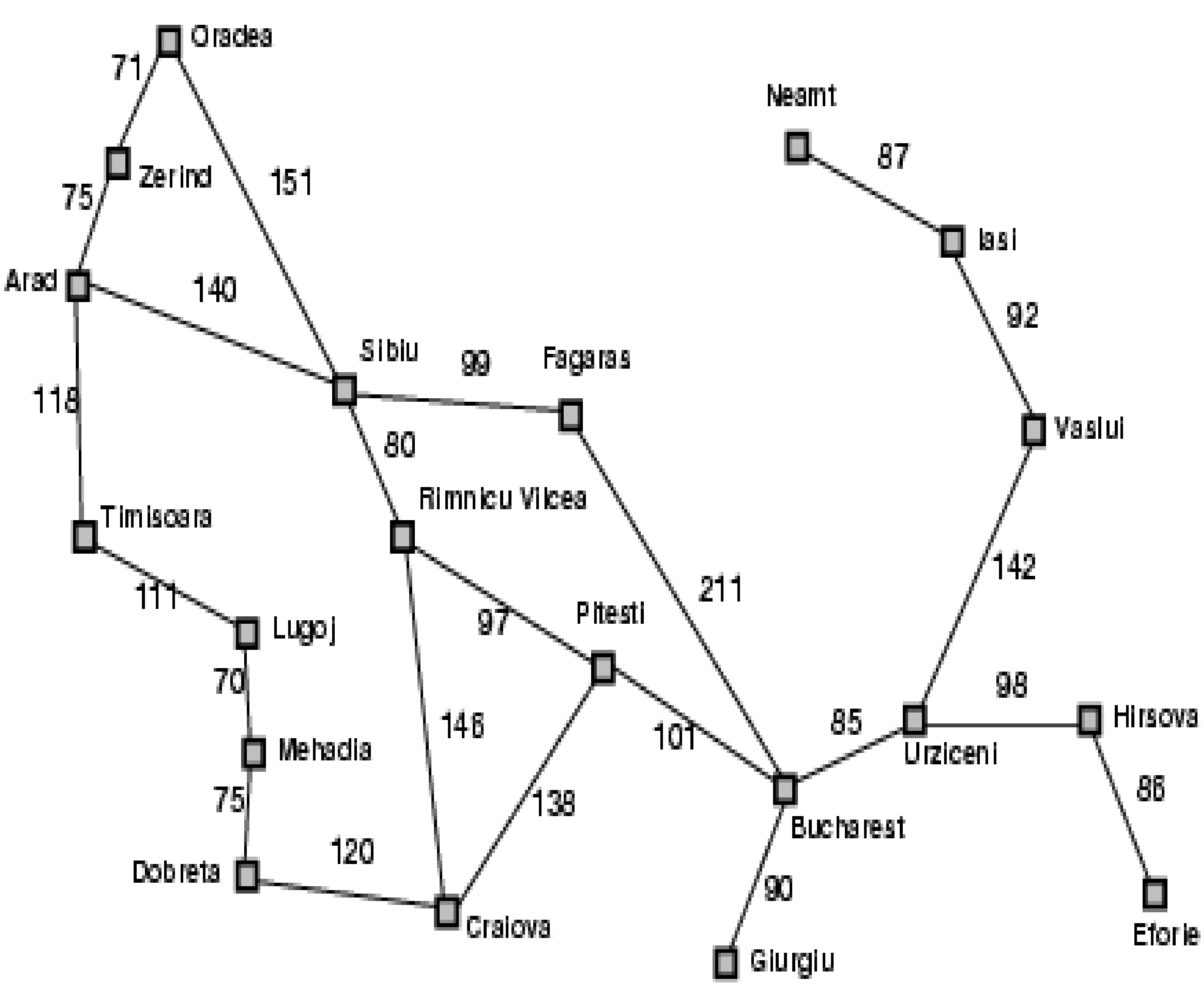
- For nodes contained in OPEN, compare $h'(n_i)$ with its existing ones before expanding n. If new $h'(n_i)$ is smaller update the same and reset the pointer from n_i to n.

- If the child node n_i is contained in CLOSED and the new $h'(n_i)$ is smaller than the existing one, then reset the value, update the pointer from n_i to n and put n_i in OPEN.

- List the nodes in OPEN in the order of the least $h'(n)$.

S7. Go to Loop.

A Algorithm : Example

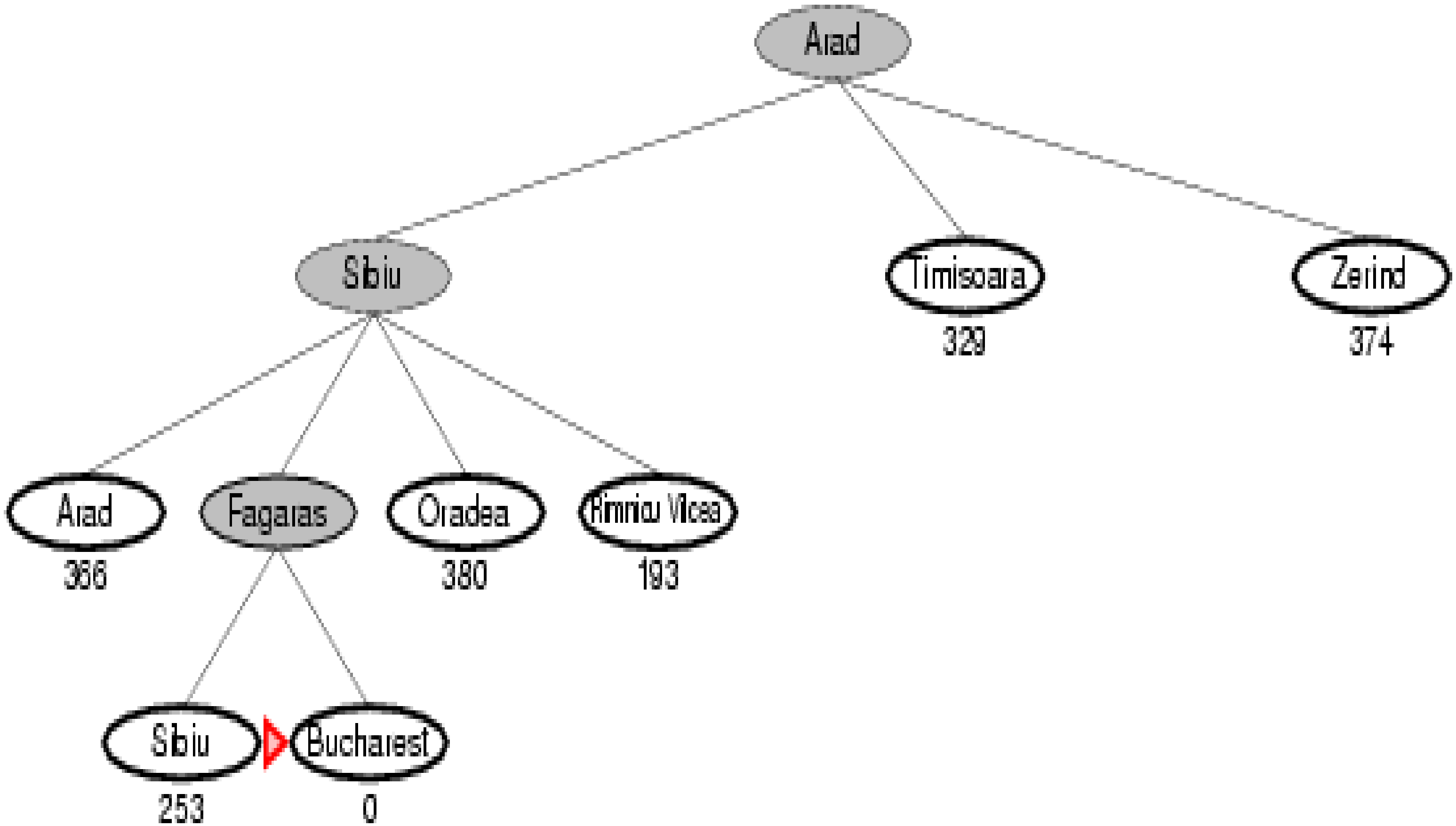


Straight–line distance
to Bucharest

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

Best First Expansion

- Total Cost = 253 + 176 = 429 kms

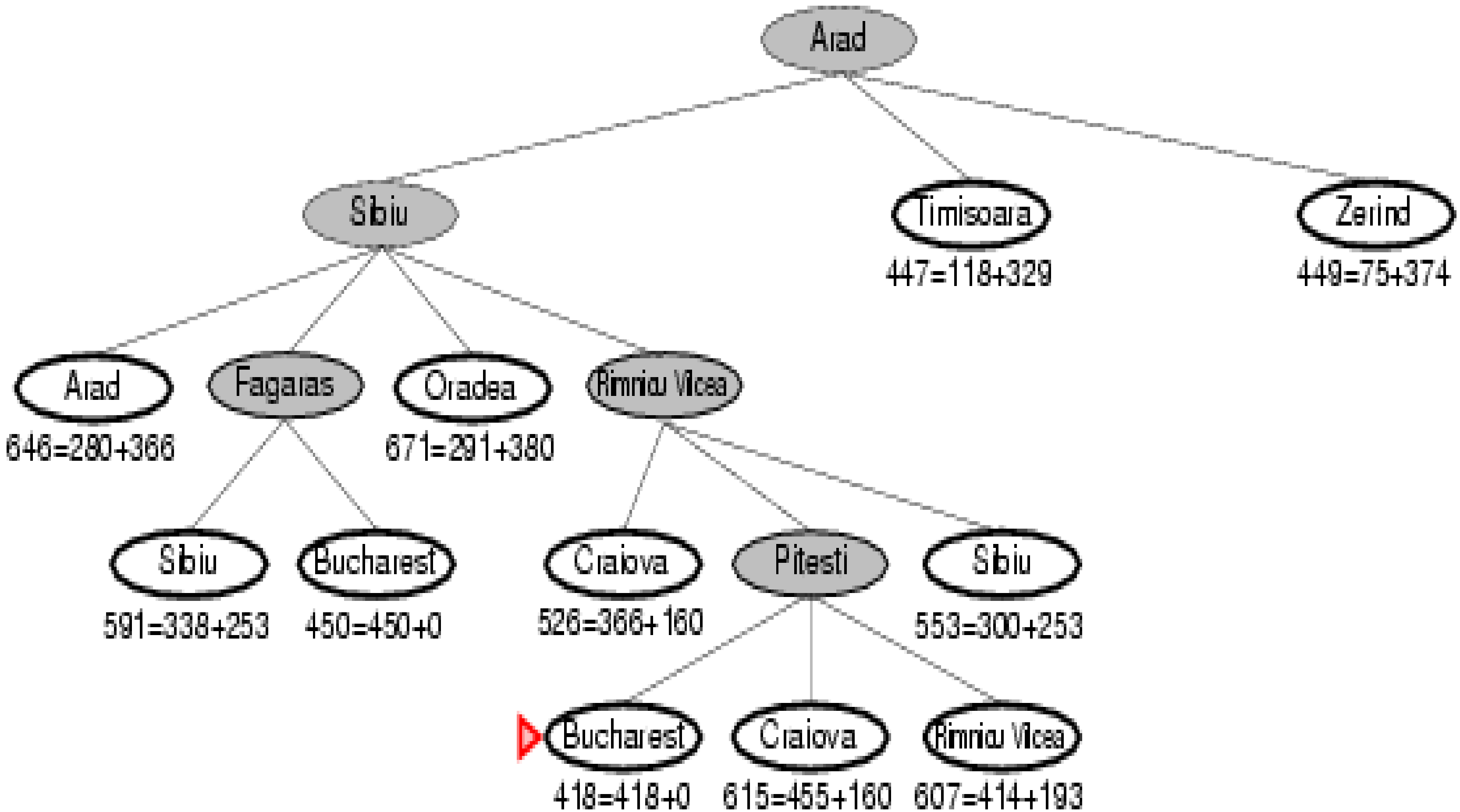


A Algorithm computation

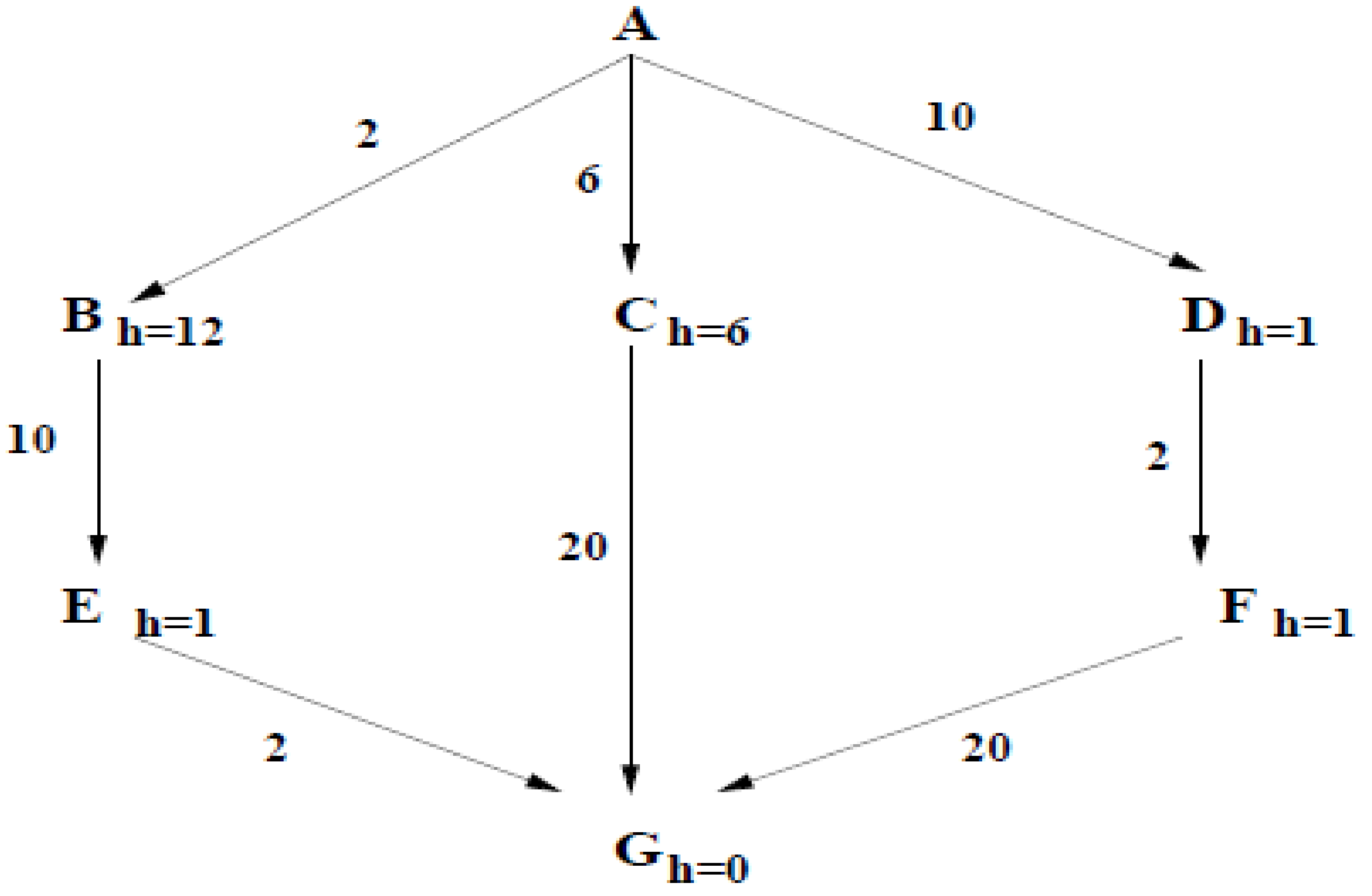
n	OPEN	CLOSED	h'(n) computation	Remarks
	A^{366}		h'(A) = 0+366=366	
A^{366}		A^{366}		Goal(S) is false
	S^{393} $T^{447}Z^{449}$		h'(S) = 140+253=393 h'(T) = 118+329=447 h'(Z) = 75+374=449	Expand A
S^{393}	$T^{447}Z^{449}$	A^{366}		Goal(A) is false
				Expand S
	..			
	..			
	..			

A Algorithm Expansion

- Total cost = 418



Example 2



A* Algorithm

- Algorithm ***A****.

If we use the evaluation function $h^*(n) = g(n) + f'(n)$ in which $f'(n)$ is less or equal to the cost of the minimal path from n, the result is called ***Algorithm A**** , where,

n is any state encountered in the search,

g is the cost of the current path from the start state to n,

f is the actual cost of the shortest path from n to the goal that passes through n.

Thus,

h^* is the cost of the "optimal" path from a start node to a goal node that passes through n.

In algorithm A, $g(n)$, is a reasonable estimate of g^*, but they may not be equal:

$$g(n) \geq g^*.$$

If algorithm A uses an evaluation function in which $f^*(n) \geq f'(n)$, then it is called ***algorithm A**** (A STAR).

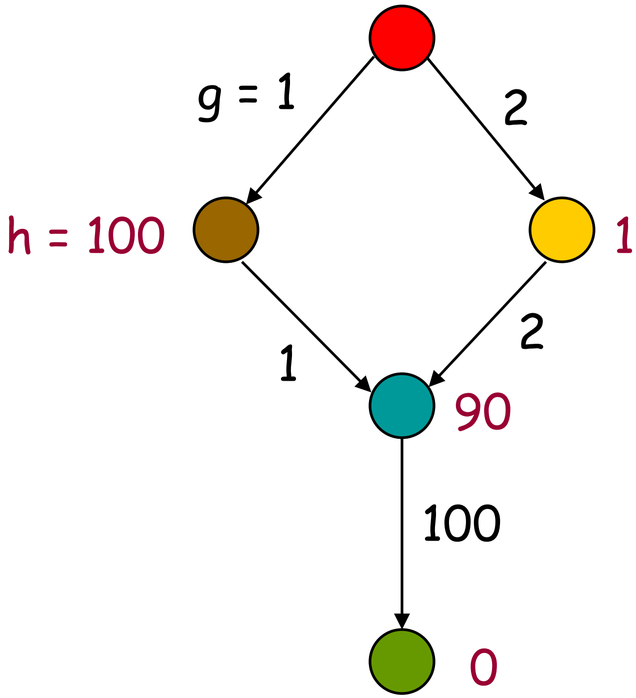
A* Search

- One of the most popular algorithms in AI
- $h(n) = g(n) + f(n)$, where:
  - $g(n)$ = cost of best path found so far to n
  - $f(n)$ = *admissible* heuristic function
- $\forall n,n' \; \exists \varepsilon : c(n,n') \geq \varepsilon > 0$
  - There is a positive cost between every two different nodes
  - Infinite paths will have infinite costs

Result #1

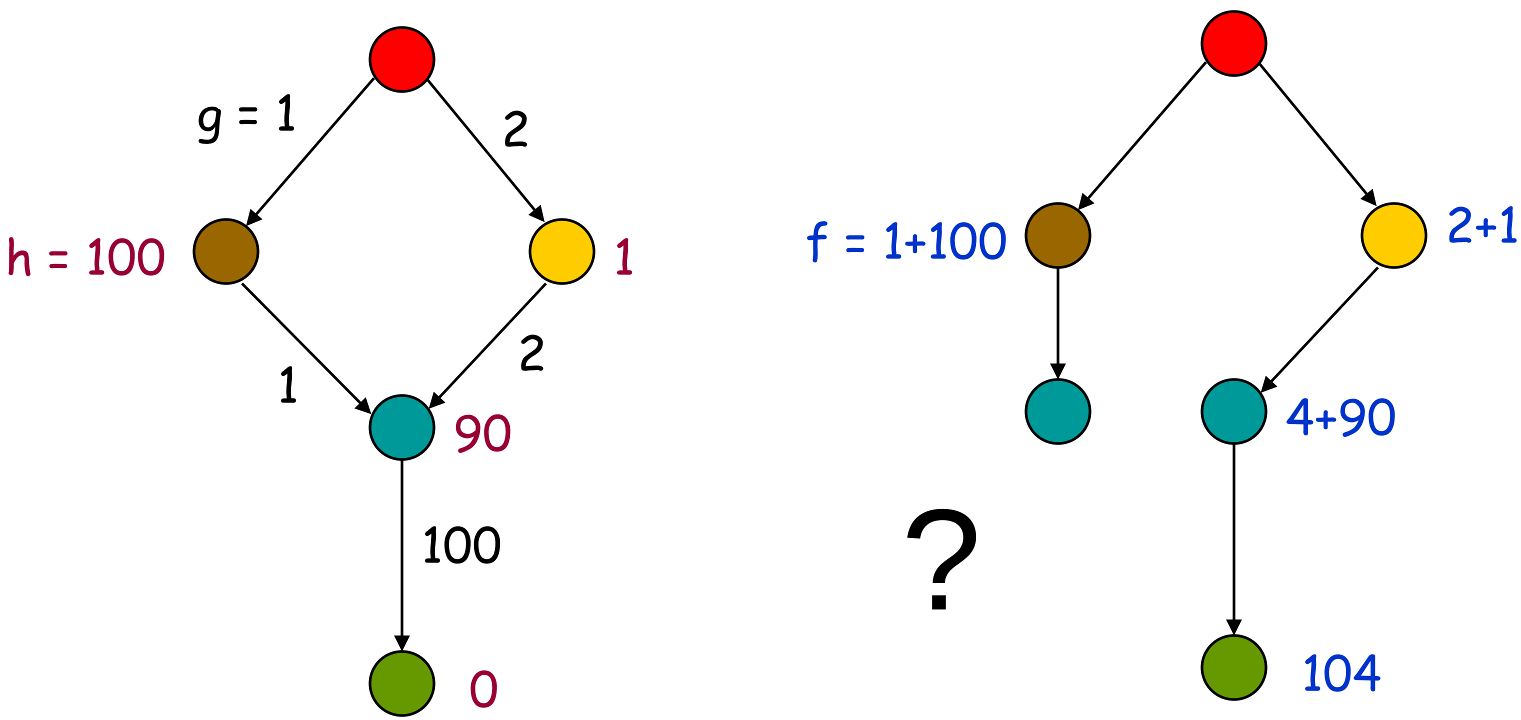
- A* is complete and optimal
  - If nodes revisiting states are not discarded

What to do with revisited states?



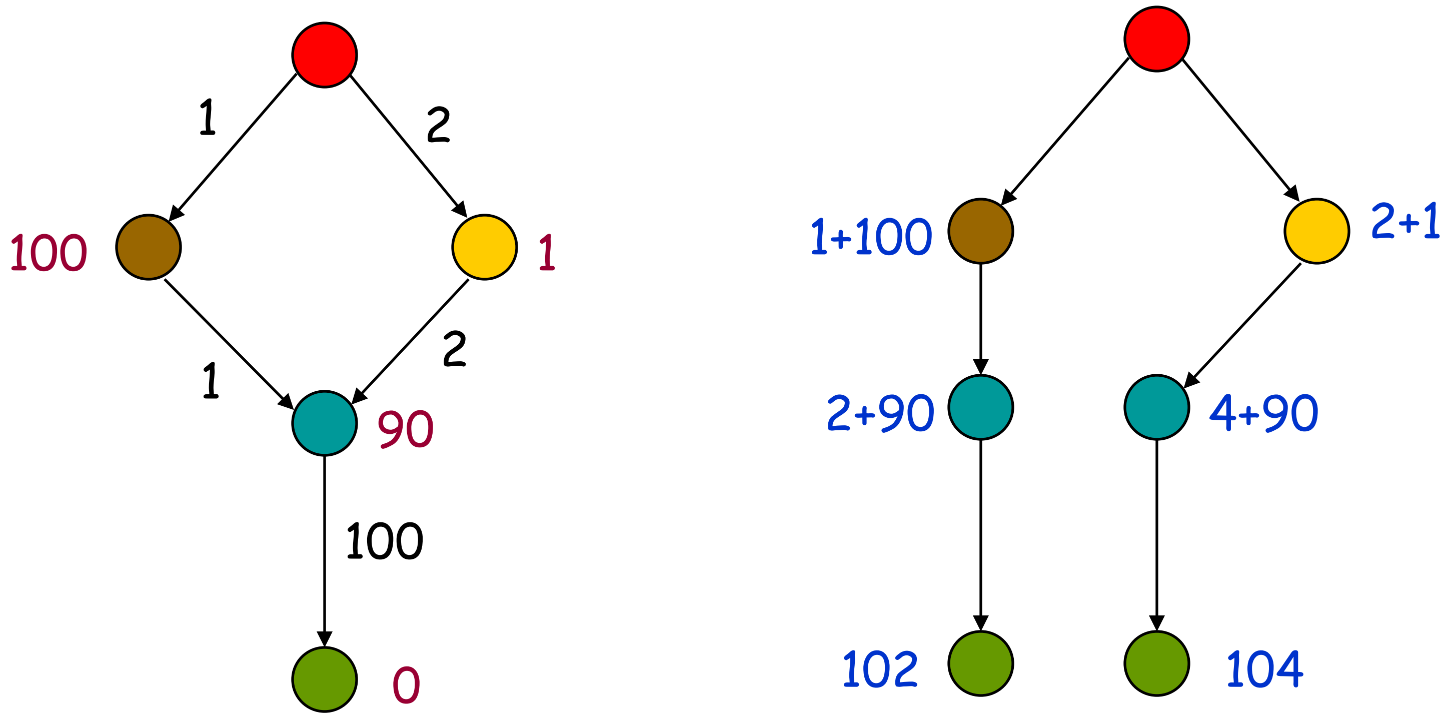
- The heuristic h is clearly admissible

What to do with revisited states?



If we discard this new node, then the search algorithm expands the goal node next and returns a non-optimal solution

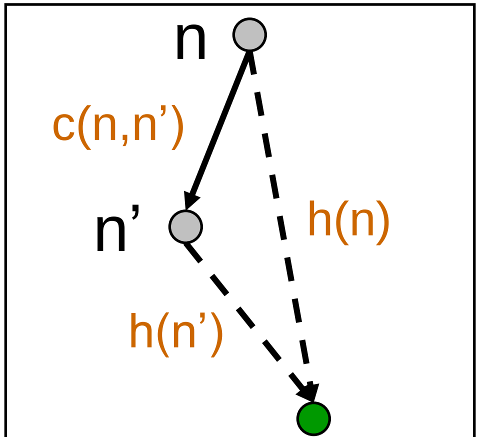
What to do with revisited states?



Instead, if we do not discard nodes revisiting states, the search terminates with an optimal solution

Consistent heuristics

- A heuristic h is *consistent* (or *monotone*) if:
  - for each node n and each child n' of n:
    - $h(n) \leq c(n,n') + h(n')$
  - for each goal node G
    - $h(G) = 0$



(triangle inequality)

- A consistent heuristic is also admissible
- A consistent heuristic becomes more precise as we go deeper in the search tree

Result #2

- If h is consistent, then whenever A* expands a node, it has already found an optimal path to this node's state

Complexity of A*

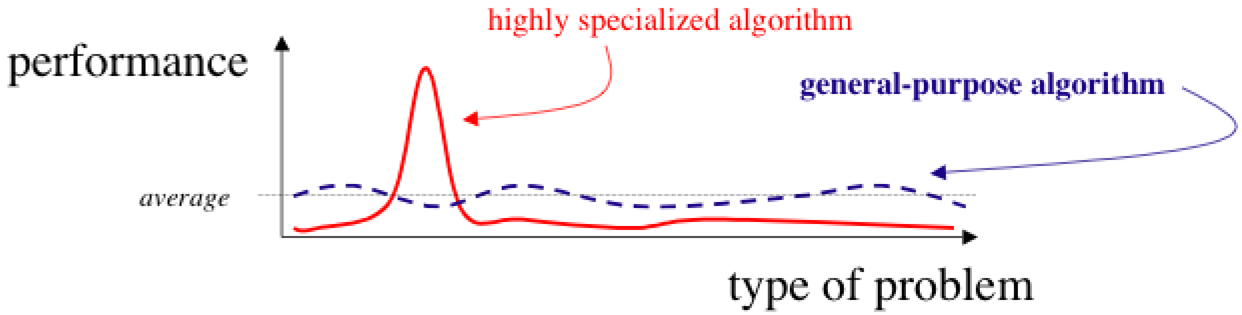
- Time
  - The better the heuristic, the less time
    - Best case: h is perfect, O(d)
    - Worst admissible case: h is 0, $O(b^d) \rightarrow$ BFS
- Space
  - All nodes (open and closed list) are saved in case of repetition
  - Worst case: $O(n_S)$
    - n_S is the number of states
- A* generally runs out of space before it runs out of time

IDS vs. A*

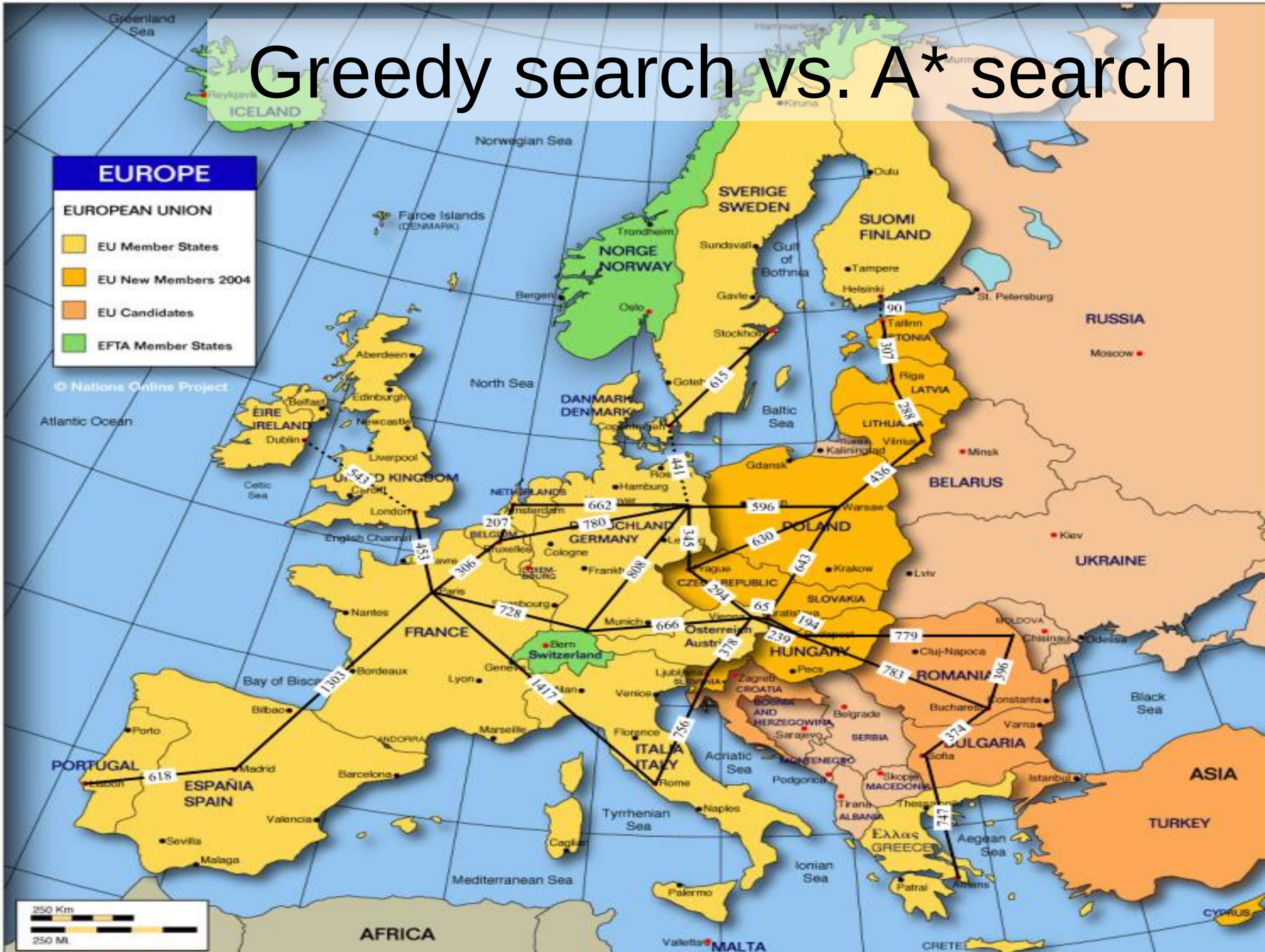
	For 8-puzzle, average number of states expanded over 100 randomly chosen problems in which optimal path is length...		
	... 4 steps	... 8 steps	... 12 steps
Iterative Deepening (see previous slides)	112	6,300	3.6×10^6
A* search using "number of misplaced tiles" as the heuristic	13	39	227
A* using "Sum of Manhattan distances" as the heuristic	12	25	73

“No Free Lunch” theorem

- Any two algorithms are equivalent when their performance is averaged across all possible problems (Wolpert and Macready)
- The performance of an algorithm may be excellent for a problem but catastrophic for all other cases
  - Great performance, no robustness
- Contrary, another algorithm may have mediocre performance for all cases, but it doesn’t fail
  - Poor performance, high robustness
- Common sense
  - Robustness * Efficiency = Constant
  - Generality * Depth = Constant



Greedy search vs. A* search



Problem #1

- Greedy Best First
  - Prague – Warsaw – Bratislava – Budapest – Iasi
  - Total cost: 2246 km
- A*
  - Prague – Vienna – Budapest – Iasi
  - Total cost: 1312 km

Problem #1

- Prague – Warsaw
  - g = 630 (road distance Prague – Warsaw)
  - f = 723 (direct distance Warsaw – Iasi)
- Prague – Vienna
  - g = 294 (road distance Prague – Vienna)
  - f = 842 (direct distance Vienna – Iasi)
- Greedy (h): 723 (W) < 842 (V) $\rightarrow$ chooses Warsaw
- A* (g+f): 1353 (W) > 1136 (V) $\rightarrow$ chooses Vienna

Problem #2

- Greedy
  - Bruxelles – Berlin – Prague – Vienna – Budapest – Bucharest – Sofia – Athens Total cost: 3562 km
- A*
  - Bruxelles – Paris – Konstanz – Vienna – Budapest – Bucharest – Sofia – Athens Total cost: 3843 km

Problem #2

- Bruxelles – Berlin
  - g = 780 (road distance Bruxelles – Berlin)
  - f = 1824 (direct distance Berlin – Athens)
- Bruxelles – Paris
  - g = 306 (road distance Bruxelles – Paris)
  - f = 2103 (direct distance Paris – Athens)
- Greedy (h): 1824 (B) < 2103 (P) → chooses Berlin
- A* (g+f): 2604 (B) > 2409 (P) → chooses Paris

Statistics for city route

- They apply to the previous problem space only (every problem is different)
- Considering all possible problems (routes between any two cities)
  - A* is better for 17.65% of the problems
  - Greedy is better for 0.15 % of the problems
  - Identical results for 82.2% of the problems

Conclusion

- Heuristic Search methods are useful when:
  - The search space is large, and
    - Some additional info is given in the problem statement
    - No other technique is available, and
    - There exist “good” heuristics

References

- Elaine Rich, Kevin Knight, “Artificial Intelligence”, Tata Mcgraw Hill, 2002.
- Amit Konar, ”Artificial Intelligence and Soft Computing”, CRC Press, 2000.
- Peter Norvig, Stuart Russel, “Artificial Intelligence: A modern Approach”, Prentice Hall of India, 2006.
- Florin Leon, http://eureka.cs.tuiasi.ro/~fleon