A* searching Algorithm

By

Dr. Ismael AbdulSattar

A* algorithm

D E F I N I T I O N

ALGORITHM A, ADMISSIBILITY, ALGORITHM A*

Consider the evaluation function ***f(n) = g(n) + h(n)***, where

n is any *state* encountered in the search.

g(n) is the cost of n from the start state.

h(n) is the heuristic estimate of the cost of going from n to a goal.

If this evaluation function is used with the **best_first_search** algorithm of

Section 6.1, the result is called algorithm A.

A search algorithm is ***admissible*** if, for any graph, it always terminates in the optimal solution path whenever a path from the start to a goal state exists.

If algorithm A is used with an evaluation function in which h(n) is less than or equal to the cost of the minimal path from n to the goal, the resulting search algorithm is called algorithm A* (pronounced "A STAR").

A* search Algorithm

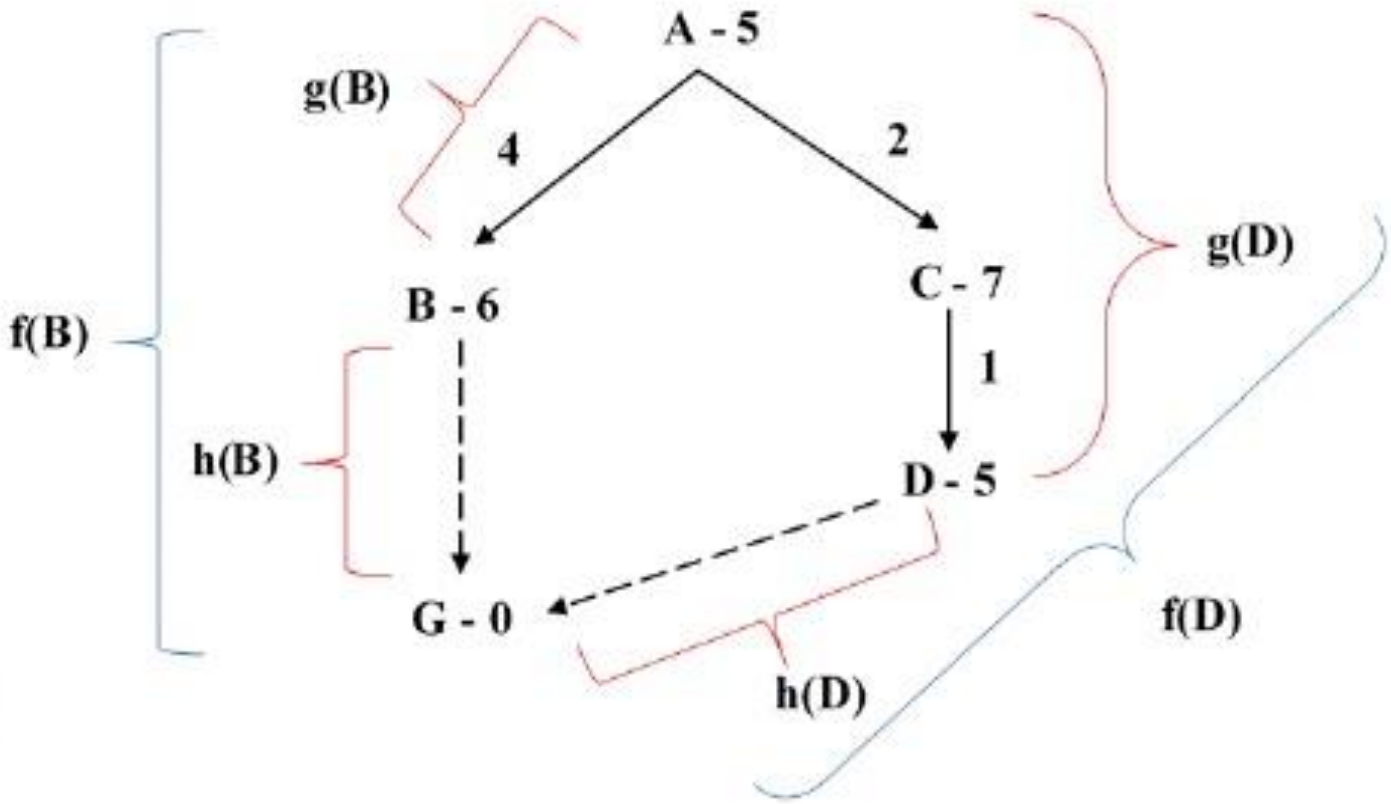
- Also called Best First Search with Cost Function
- It uses:

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

- **g(n)** is a cost function that represents the actual length of path from n to the root
- **h(n)** is a heuristic Function that estimates the length from n to the goal

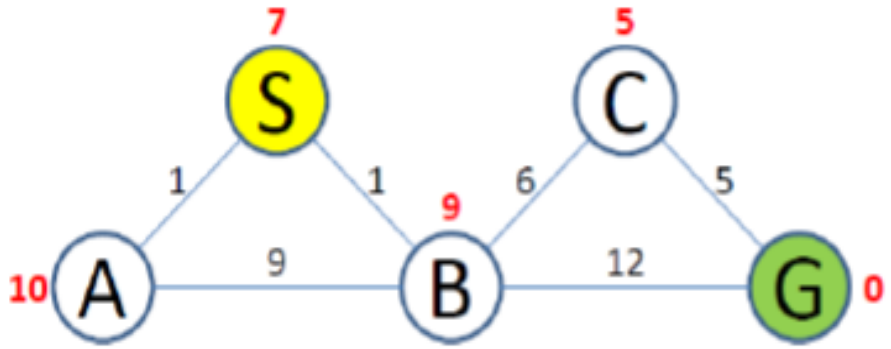
A* search Algorithm

- f(n) = g(n) + h(n)
- For state (A):
  f(A) = g(A) + h(A)
  f(A) = 0 + 5 = 5
- For state (B):
  f(B) = g(B) + h(B)
  f(B) = 4 + 6 = 10
- For state (C):
  f(C) = g(C) + h(C)
  f(C) = 2 + 7 = 9
- For state (D):
  f(D) = g(D) + h(D)
  f(D) = (2+1)+5 = 10



A* search Algorithm

- **Find the path from S to G using A* algorithm**



open = [S7]; closed = [ ]

open = [B10 , A11]; closed = [S7]

open = [A_S11, C12 , G13]; closed = [B10, S7]

open = [C12 , G13]; closed = [A_S11 , B10, S7]

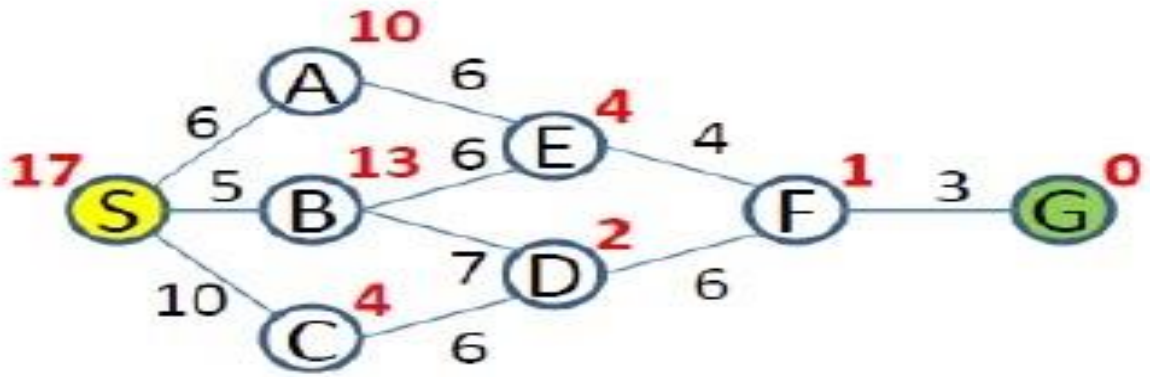
open = [G_C12]; closed = [C12, A_S11 , B10, S7]

The solution is found. closed = [G_C12 , C12, A_S11 , B10, S7]

Path is: S7 → B10 → C12 → G_C12

A* search Algorithm

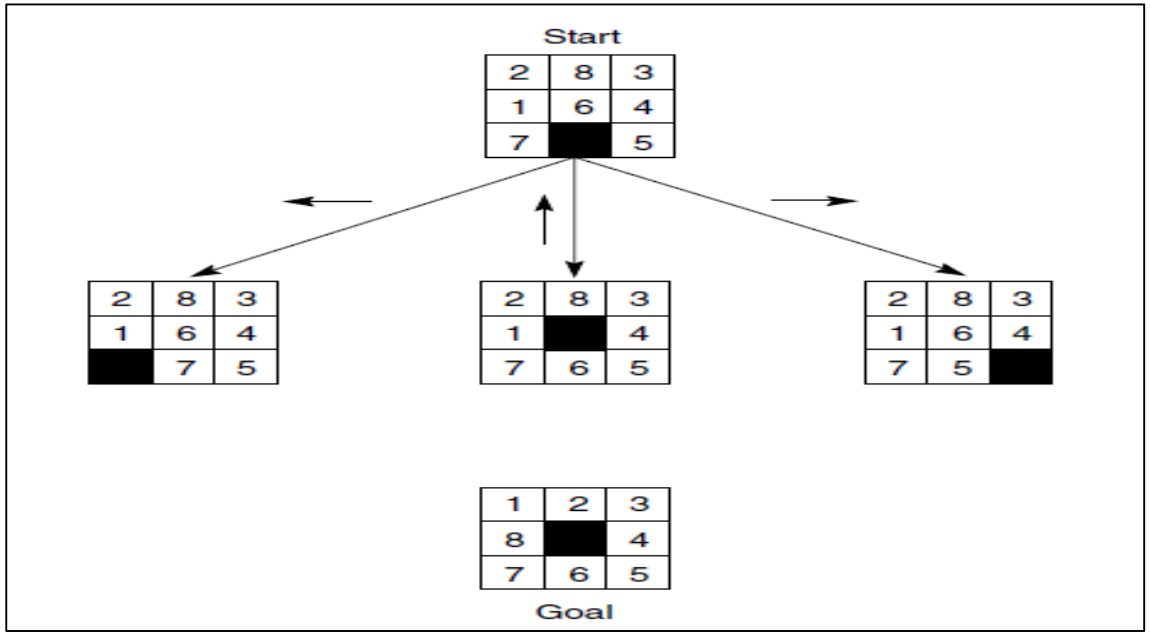
Find the path from S to G using A* algorithm



open = [S17]; closed = [ ]
open = [C14, A16, B18]; closed = [S17]
open = [A16, D18, B18]; closed = [C14, S17]
open = [E16, D18, B18]; closed = [A16 , C14, S17]
open = [F17, D18, B18]; closed = [E16, A16 , C14, S17]
open = [D18, B18, G19]; closed = [F17, E16, A16 , C14, S17]
open = [B_S18, G19]; closed = [D18, F_E17, E16, A16 , C14, S17]
open = [D_B14, E_B15, G19]; closed = [B_S18, F_E17, A16 , C14, S17]
open = [E_B15, G19]; closed = [D_B14, B_S18, F_E17, A16 , C_S14, S17]
open = [F_E16, G19]; closed = [E_B15, D_B14, B_S18, A16 , C_S14, S17]
open = [G18]; closed = [F_E16, E_B15, D_B14, B_S18, A16 , C_S14, S17]
The solution is found. closed = [G18, F_E16, E_B15, D_B14, B_S18, A16 , C_S14, S17]
Path is: S17 → B_S18 → E_B15 → F_E16 → G18

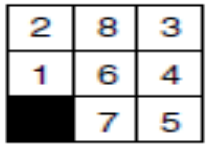
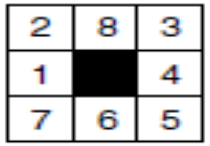
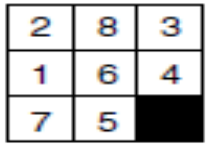
Implementing Heuristic Evaluation Functions

Know evaluate the performance of several different heuristics for solving problems will take 8-puzzle as example , Figure below shows a start and goal state for the 8-puzzle, along with the first three states generated in the search. the simplest heuristic *counts the tiles out of place* in each state when compared with the goal. This is intuitively appealing, because it would seem that, all else being equal; the state that had *fewest* tiles out of place is probably closer to the desired goal and would be the best to examine next.

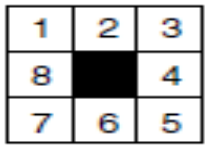


Implementing Heuristic Evaluation Functions

1	2	3	4	5	6	7	8
1	1	0	0	0	1	1	1

	5	6	0
	3	4	0
	5	6	0
	Tiles out of place	Sum of distances out of place	2 x the number of direct tile reversals

1	2	3	4	5	6	7	8
1	1	0	0	0	1	1	2



Goal

This is clear

Implementing Heuristic Evaluation Functions

The distance from the starting state to its descendants can be measured by maintaining a depth count for each state. This count is **0** for the beginning state and is incremented by **1** for each level of the search. This depth measure can be added to the heuristic evaluation of each state to bias search in favor of states found shallower in the graph.

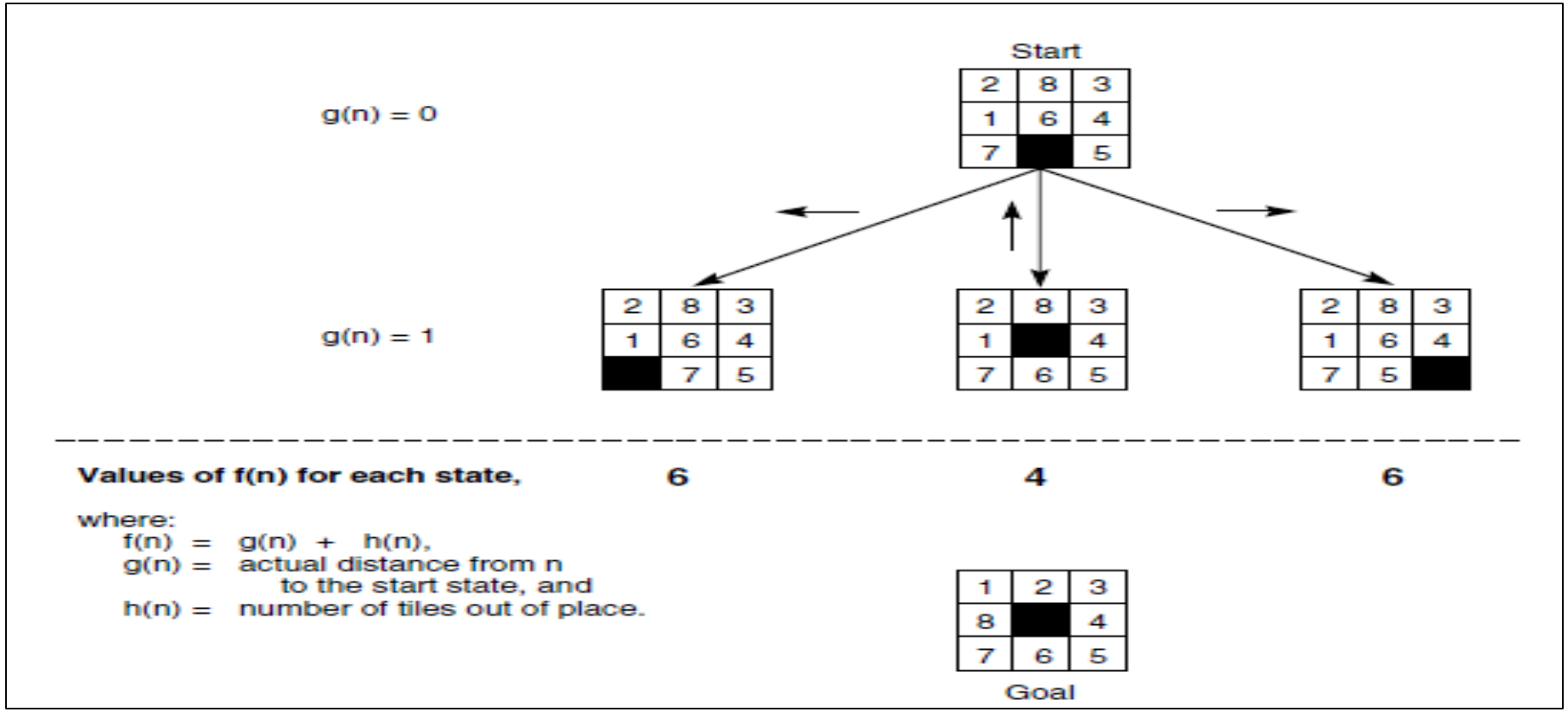
This makes our evaluation function, f, the sum of two components:

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

Where $g(n)$ measures the actual length of the path from any state **n** to the start state and $h(n)$ is a heuristic estimate of the distance from state **n** to a goal.

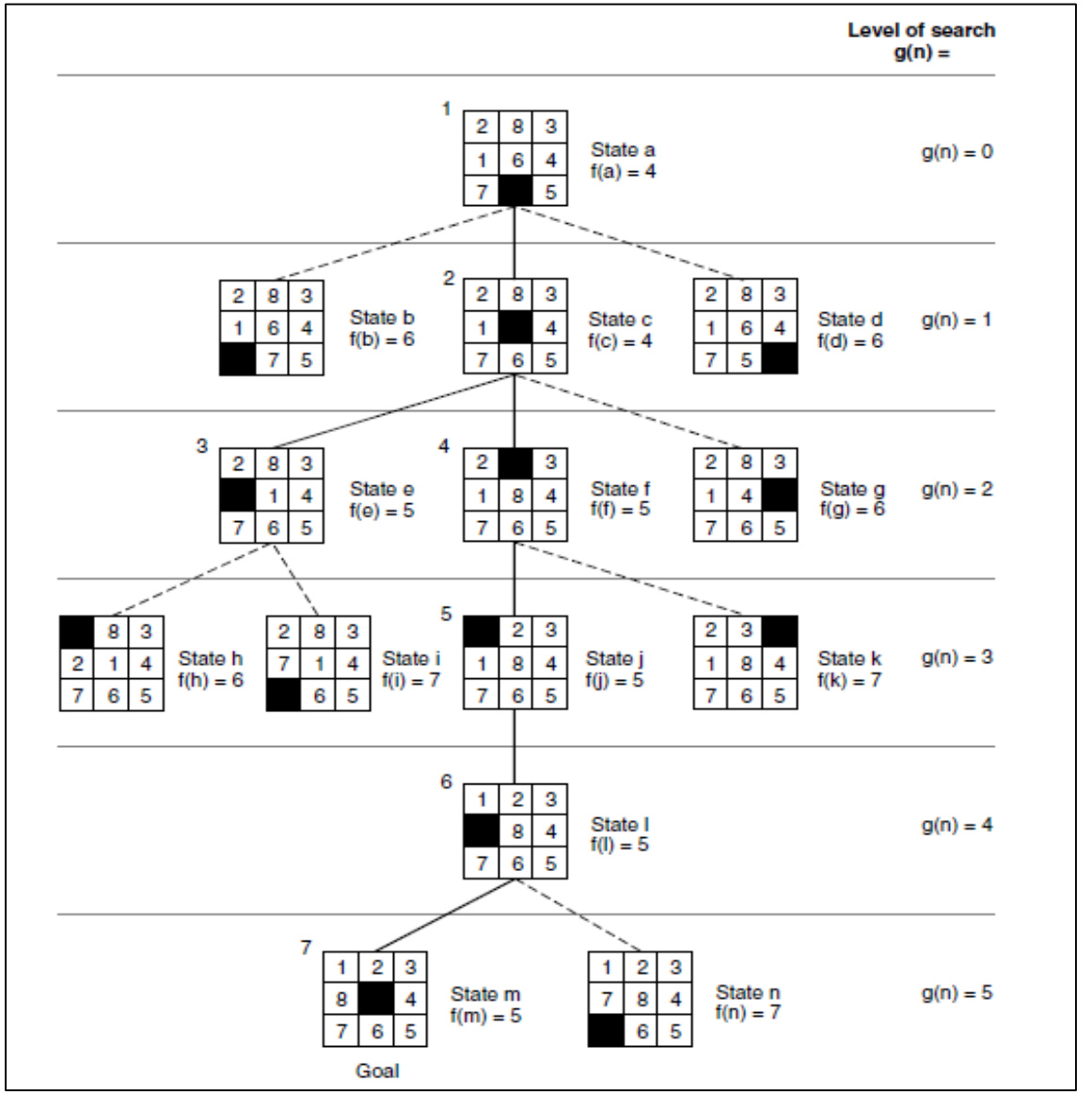
Implementing Heuristic Evaluation Functions

In the *8-puzzle*, for example, we can let $h(n)$ be the number of tiles out of place. When this evaluation is applied to each of the child states in Figure below, their f values are **6**, **4**, and **6**, respectively



Implementing Heuristic Evaluation Functions

1. open = [a4];
   closed = [ ]
2. open = [c4, b6, d6];
   closed = [a4]
3. open = [e5, f5, b6, d6, g6];
   closed = [a4, c4]
4. open = [f5, h6, b6, d6, g6, i7];
   closed = [a4, c4, e5]
5. open = [j5, h6, b6, d6, g6, k7, i7];
   closed = [a4, c4, e5, f5]
6. open = [l5, h6, b6, d6, g6, k7, i7];
   closed = [a4, c4, e5, f5, j5]
7. open = [m5, h6, b6, d6, g6, n7, k7, i7];
   closed = [a4, c4, e5, f5, j5, l5]
8. success, m = goal!



Admissibility, Monotonicity, and Informedness

- We may evaluate the behavior of heuristics along a number of dimensions. For instance, we may desire a solution and also require the algorithm to find the shortest path to the goal. This could be important when an application might have an excessive cost for extra solution steps, such as planning a path for an autonomous robot through a dangerous environment. Heuristics that find the shortest path to a goal whenever it exists are said to be ***admissible***. In other applications a minimal solution path might not be as important as overall problem-solving efficiency.
- We may want to ask whether any better heuristics are available. In what sense is one heuristic "better" than another? This is the informedness of a heuristic.
- When a state is discovered by using heuristic search, is there any guarantee that the same state won't be found later in the search at a cheaper cost (with a shorter path from the start state)? This is the property of monotonicity.

Admissibility Measures

A search algorithm is ***admissible*** if it is guaranteed to find a minimal path to a solution whenever such a path exists. ***Breadth-first search*** is an admissible search strategy. Because it looks at every state at level **n** of the graph before considering any state at the level **n + 1**, any goal nodes are found along the shortest possible path. Unfortunately, breadth-first search is often too inefficient for practical use.

H.W

Illustration of **overestimation** and **underestimation** of h in A* algorithm?