A * Algorithm

A * algorithm

A* is a type of search algorithm. Some problems can be solved by representing the world in the initial state, and then for each action we can perform on the world we generate states for what the world would be like if we did so.

If you do this until the world is in the state that we specified as a solution, then the route from the start to this goal state is the solution to your problem.

Use of state space search to find the shortest path between two points (path finding).

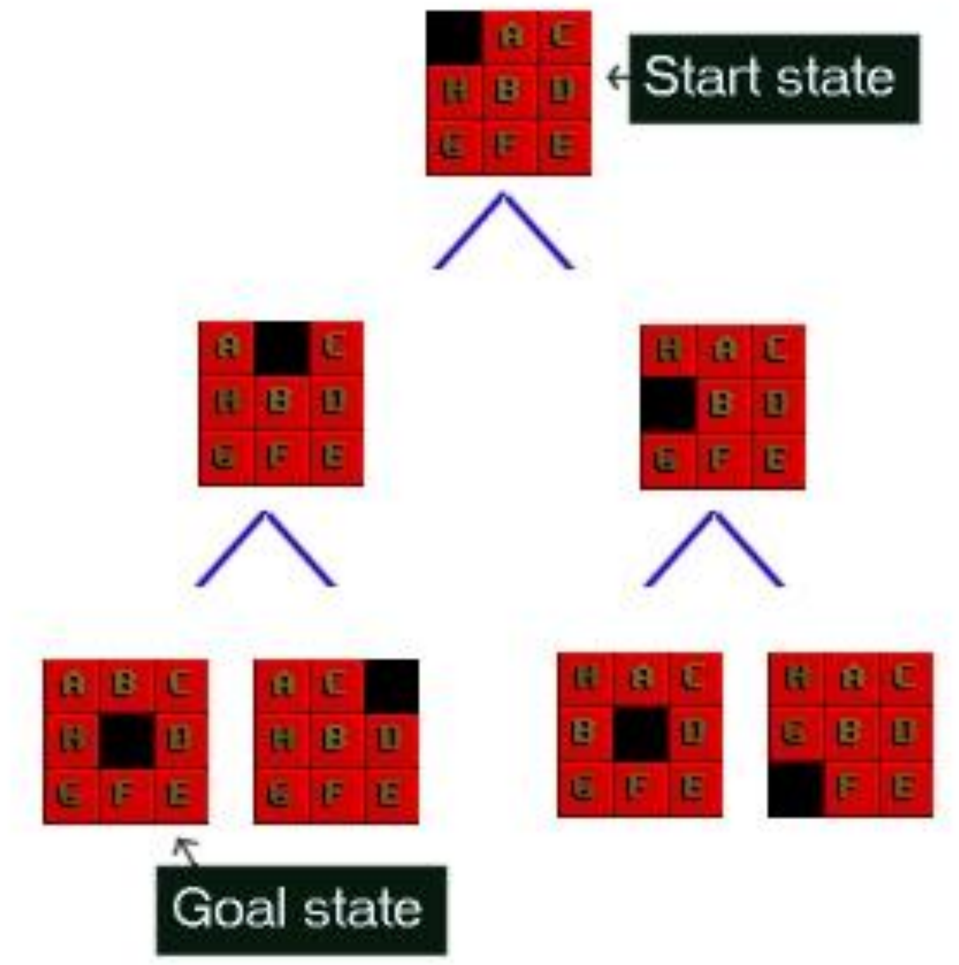
Heuristics and Algorithms

At this point we need a heuristic to help us cut down on this huge search problem. What we need is to use our heuristic at each node to make an estimate of how far we are from the goal.

In path finding we know exactly how far we are, because we know how far we can move each step, and we can calculate the exact distance to the goal.

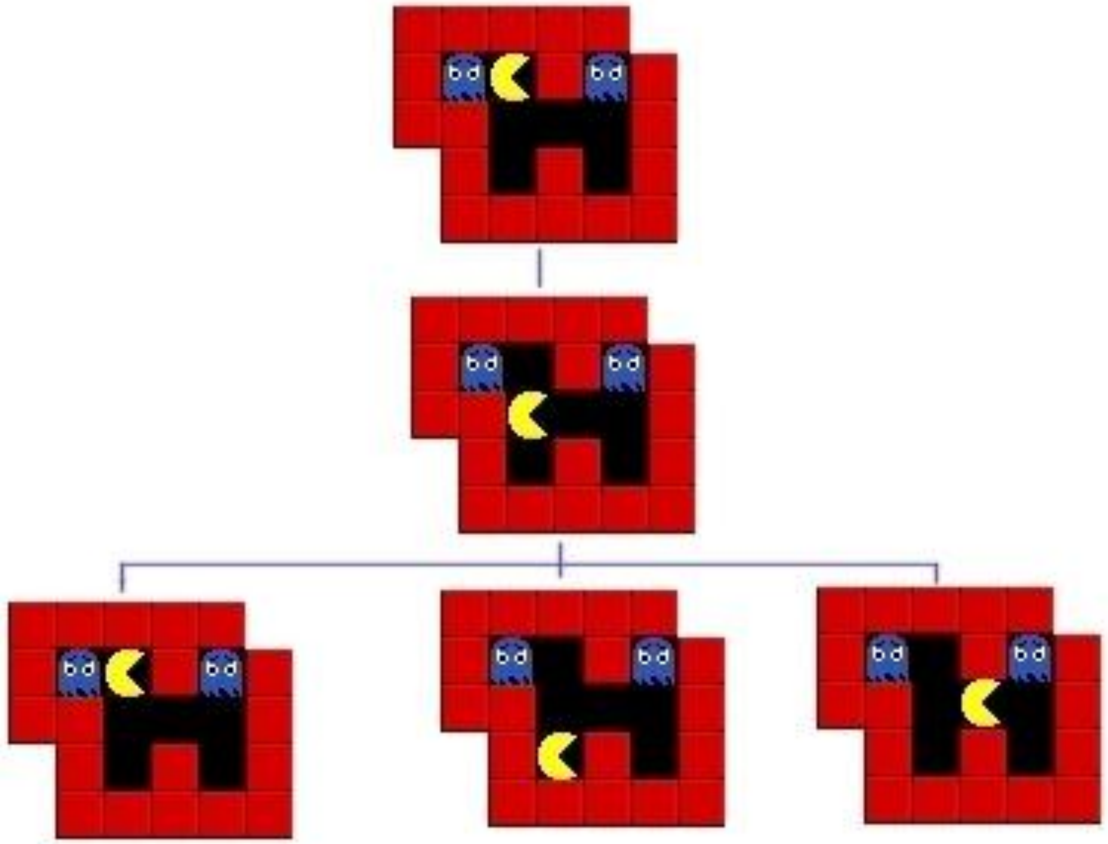
For example, in 8-puzzle there is no known algorithm for calculating from a given position how many moves it will take to get to the goal state. So various heuristics have been devised.

8 Puzzle



This is a simple sliding tile puzzle on a 3*3 grid where one tile is missing and you can move the other tiles into the gap until you get the puzzle into the goal position.

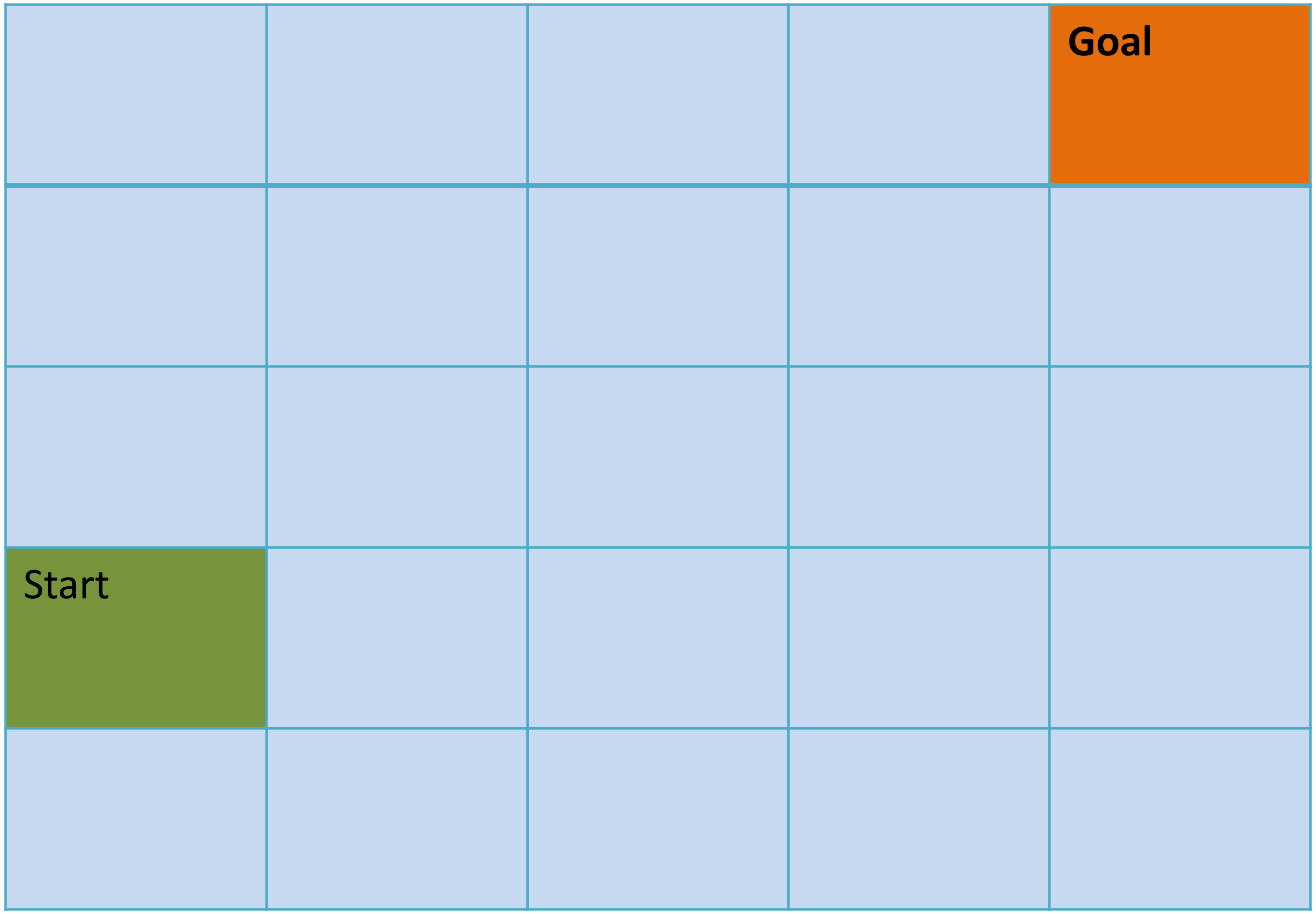
Example



- **Pathfinding in gaming**

  The A* algorithm will not only find a path, if there is one, but it will find the shortest path. A state in pathfinding is simply a position in the world.

  some other path finding scenario, you want to search a state space and find out how to get from somewhere you are to somewhere you want to be, without bumping into walls or going too far.


Goal
Start

A * Operation

At the operation of the A* algorithm. What we need to do is start with the goal state and then generate the graph downwards from there. We ask how many moves can we make from the start state? The answer is 2, there are two directions we can move the blank tile, and so our graph expands.

If we were just to continue blindly generating successors to each node, we could potentially fill the computer's memory before we found the goal node. Obviously we need to remember the best nodes and search those first. We also need to remember the nodes that we have expanded already, so that we don't expand the same state repeatedly.

				Goal
open	open			
Start	open			
open	open			

				Goal
open	closed			
Start	open			
open	open			

				Goal
open	closed			
Start	open			
open	open			

				Goal
open	open	open		
open	closed	open		
Start	open	open		
open	open			

				Goal
open	open	open		
open	closed	closed		
Start	open	open		
open	open			

Let's start with the OPEN list. This is where we will remember which nodes we haven't yet expanded.

When the algorithm begins the start state is placed on the open list, it is the only state we know about and we have not expanded it.

So we will expand the nodes from the start and put those on the OPEN list too. Now we are done with the start node and we will put that on the CLOSED list. The CLOSED list is a list of nodes that we have expanded.

Using the OPEN and CLOSED list lets us be more selective about what we look at next in the search.

We want to look at the best nodes first. We will give each node a score on how good we think it is. This score should be thought of as the cost of getting from the node to the goal plus the cost of getting to where we are.

f = g + h

Traditionally this has been represented by the letters f, g and h.

1. 'g' is the sum of all the costs it took to get here,
2. 'h' is our heuristic function, the estimate of what it will take to get to the goal.
3. 'f' is the sum of these two. We will store each of these in our nodes.

Using the f, g and h values the A* algorithm will be directed, subject to conditions we will look at further on, towards the goal and will find it in the shortest route possible.

So far we have looked at the components of the A*, let's see how they all fit together to make the algorithm :

A* pseudocode

Steps

1.Create a *search graph* G,consisting solely of the start node s.Put s on a list called OPEN.

2 .Create a list called CLOSED that is initially empty.

3.LOOP:if OPEN is empty,exit with failure.

4.Select the first node on OPEN,remove it from OPEN and put it on CLOSED.Call this node n.

5.If n is a goal node,exit successfully with the solution obtained by tracing a path along the pointers from n to s in G.

6.Expand node n,generating the set,M,of its successors
and install them as successors of n in G.

7. Establish a pointer to n from those members of M that were not already in G(I .e, not already on either OPEN or CLOSED). Add these members of M to OPEN.For each member of M that was already on OPEN or CLOSED,decide whether or not to redirect its pointer to n.For each member of M already on CLOSED,decide for each of its descendents in G whether or not to redirect its pointer.
8. Reorder the list OPEN,either according to some scheme or some heuristic merit.

9. Goto LOOP

A^* search

To 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

Functions used in the algorithm:

Evaluation Function f(n):At any node n,it estimates the sum of
the cost of the minimal cost path from
the start node s to node n plus the cost
of a minimal cost path from node n to a
goal node.

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

Where g(n)=cost of the path in the search tree from s to n;
h(n)=cost of the path in the search tree from n to a goal node;

Function f ‘ (n):At any node n,it is the actual cost of an optimal path
from node s to node n plus the cost of an optimal
path from node n to a goal node.

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

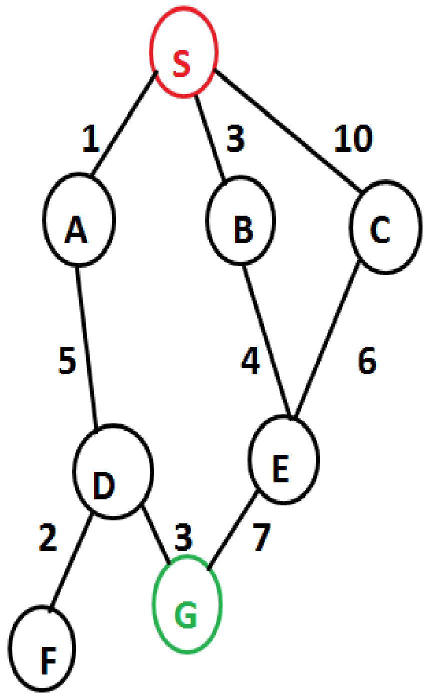
Where g’ (n)=cost of the optimal path in the search tree from s to n;
h’ (n)=cost of the optimal path in the search tree from n to a
goal node;

h’ (n):It is the cost of the minimal cost path from n to a goal node and any path from node n to a goal node that acheives h*(n) is an optimal path from n to a goal.

h is an estimate of h ’.

h(n) is calculated on the heuristic information from the problem domain.

A Graph search tree:

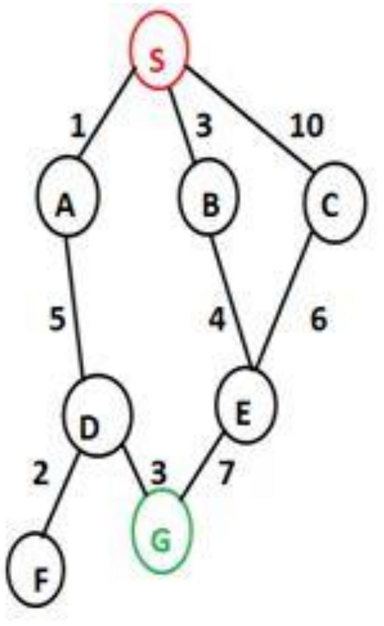


Here the Starting node S is represented in RED and the goal node is represented in GREEN.

Our aim is to reach the goal node G by tracing out the algorithm described.

F(n)=g(n)+h(n)

Calculating the f value for each of the following nodes,
we get f as:



A: 1+5+3=9

B:3+4+7=14

C:10+6+7=23

D:1+5+3=9

E:3+4+7 (10+6 is not minimal so not taken)

F:the goal node G cannot be reached from F