INTRODUCTION:

Missionaries and Cannibals problem is very famous in Artificial Intelligence because it was the subject of the first paper that approached problem formulation from an analytical viewpoint. The problem can be stated as follow. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Now we have to find a way to get everyone to the other side, without ever leaving a group of missionaries in one place outnumbered by the cannibals in other side. The above problem can be solved by a graph search method. Here I represent the problem as a set of states and operators. States are snapshots of the world and operators are those which transform one state into another state. States can be mapped to nodes of a graph and operators are the edges of the graph.

OBJECTIVES

The objectives of this project are:

- To implement AI search techniques to solve real world problem.
- To solve the Missionaries and Cannibals problem using Breadth first and Depth first search algorithm and find out the best algorithm which is best for this particular problem.

EXPLANATION

Missionaries and Cannibals uses Breadth first and Depth first search algorithm to find the solution. The node of the graph to be searched is represented by a state space. Each state space can be represent by

```
  State(no_of_missionaries, no_of_cannibals, side_of_the_boat)
```

Where no_of_missonaries are the number of missionaries at left side of river, no_of_cannibals are the number of cannibals at the left side of river and side_of_the_boat is the side of the boat at particular state. For our case
Initial State => State(3, 3, 0) and
Final State => State(0, 0, 1).
Where 0 represents left side and 1 represents right side of river.

We should make a graph search which traverse the graph from initial state and find out the final state in fewest moves. There are many AI searches that search the graphs like Breadth first search, Depth first search, or iterative deepening search. Each of these different search methods has different properties such as whether a result is guaranteed, and how much time and space is needed to carry out the search. This project uses Breadth first and Depth first search.

Production Rules for Missionaries and Cannibals problem.

Rule No	**Production Rule and Action.**
1	(i, j) : Two missionaries can go only when i-2>=j or i-2=0 in one bank and i+2>=j in the other bank.
2	(i, j) : Two cannibals can cross the river only when j-2<=i or i=0 in one bank and j+2 <= i or i+0 or i=0 in the other.
3	(i, j) : One missionary and one cannibal can go in a boat only when i-1>=j-1 or i=0 in one bank and i+1>=j+1 or i=0 in the other.
4	(i, j) : one missionary can cross the river only when i-1>=j or i=0 in one bank and i+1>=j in the other bank.
5	(i, j) : One cannibal can cross the river only when j-1 < i or i=0 in one bank and j+1<=I or j=0 in the other bank of the river.

Possible Moves
A move is characterized by the number of missionaries and the number of cannibals taken in the boat at one time. Since the boat can carry no more than two people at once, the only feasible combinations are:

Carry (2, 0).
Carry (1, 0).
Carry (1, 1).
Carry (0, 1).
Carry(0, 2).

Where Carry (M, C) means the boat will carry M missionaries and C cannibals on one trip.

Feasible Moves

Once we have found a possible move, we have to confirm that it is feasible. It is not a feasible to move more missionaries or more cannibals than that are present on one bank. When the state is state(M1, C1, left) and we try carry (M,C) then

M <= M1 and C <= C1

must be true.

When the state is state(M1, C1, right) and we try carry(M, C) then

M + M1 <= 3 and C + C1 <= 3

must be true.

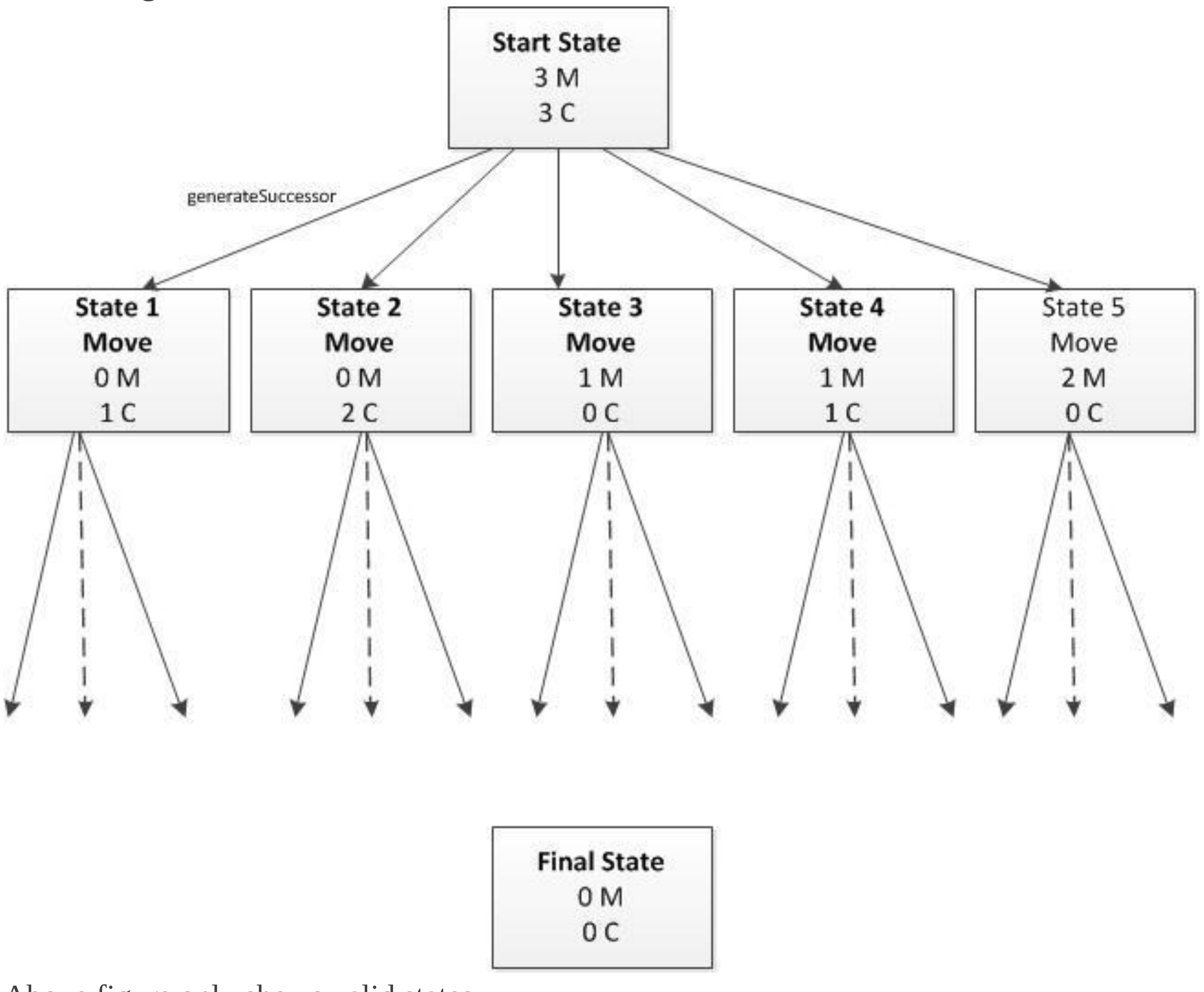
Legal Moves

Once we have found a feasible move, we must check that is legal i.e. no missionaries must be eaten.

Legal(X, X).
Legal(3, X).
Legal(0, X).

The only safe combinations are when there are equal numbers of missionaries and cannibals or all the missionaries are on one side.

Generating the next state



Above figure only shows valid states.

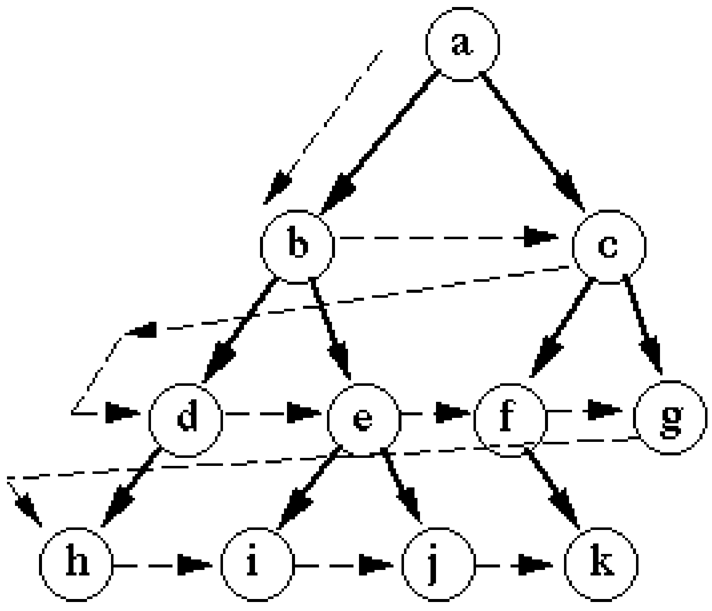
Breadth First Search Algorithm

Breadth first searches are performed by exploring all nodes at a given depth first proceeding to the next level. This means that all immediate children of nodes are explored before any of the children's children are considered. It has obvious advantage of always finding a minimal path length solution when one exists. However, a great many nodes may need to be explored before a solution is found, especially if the tree is very full.

Algorithm

BFS uses a queue structure to hold all generate but still unexplored nodes. The order in which nodes are placed on the queue for removal and exploration determines the type of search. The BFS algorithm proceeds as follows.

- Place the starting node s on the queue.
- If the queue is empty, return failure and stop.
- If the first element on the queue is a goal node g, return success and stop. Otherwise,
- Remove and expand the first element from the queue and place all the children at the end of the queue in any order.
- Return to step 2.



Breadth-first search

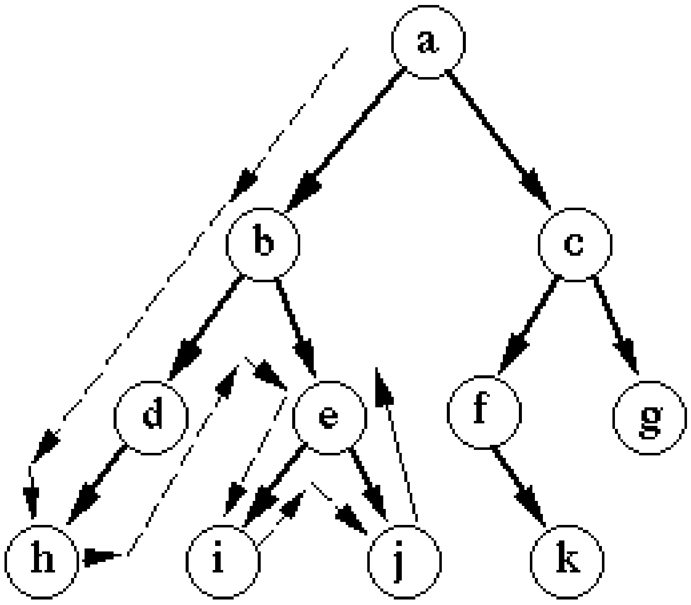
Depth First Search Algorithm

Depth first searches are performed by diving downward into a tree as quickly as possible. It does this by always generating a child node from the most recently expanded node, then generating that child's children, and so on until a goal is found or some cutoff depth point d is reached. If a goal is not found when a leaf node is reached or at the cutoff point, the program backtracks to the most recently expanded node and generates another of its children, This process continues until a goal is found or failure occurs.

Algorithm

An algorithm of the depth first search is the same as that for breadth first search except in the ordering of the nodes.

- Place the starting node s on the top of the stack
- If the stack is empty, return failure and stop.
- If the element on the stack is goal node g, return success and stop. Otherwise,
- Remove and expand the first element, and place the children at the top of the stack.
- Return to step 2.



Depth-first search

BREADTH FIRST VS DEPTH SEARCHES IN MISSIONARIES AND CANNIBALS PROBLEM.

Breadth first search is guaranteed to find the shortest path from starting state to ending state. In my program it finds the solution at level 12 from the starting node. The state utilization of breadth first, measured in terms of size of the open list, is B^n where B is the branching factor – the average number of descendants per state – and n is the level.
Depth first search is NOT guaranteed to find the shortest path but its gets deeply into the search space. In missionaries and Cannibals problem, depth first search enters into a graph cycle and suffers a infinite loop. That is, it cannot find the goal state. This problem is removed by using a arraylist which keeps record of the visited nodes. HOWEVER this makes the space utilization of depth first search as poor as Breadth first search. In normal cases, depth first is more efficient for spaces with large branching factor, because it does not have to keep all nodes at a given level on the open. Therefore space utilization of depth first is linear : B * n, since at each level the open list contains the children of a single node.

METHODOLOGY

I built the project “Missionaries and Cannibals” using programming language C# in Visual Studio 2010 ultimate. C# is an Object Oriented language just like C++. I used Console window to show the output instead of attractive graphical UI. First I attempted to do this project in C++ as I committed in project proposal but due to some technical problem [especially memory access problem associated with pointers], I switched the project to C#. I started building the application using Paper and Pencil and tried to draw the graph of states. The project consists of three main C# files. One for representing object State, another for implementing the search algorithm and generating the successors and final one to finalize the program and print the result in console window. The function which find the optimal solution state is

```
State temp = getSolutionState(State initial, State goal);
```

Input parameter 1: initial (3, 3, 0).

Input parameter 2: goal (0, 0, 1).

Output = goal state with memory reference of parent.

OUTPUT

```
Solution using Breadth First Search
This solution was found at level [12]

3M/3C <-BOAT LEFT    0M/0C
3M/1C   BOAT RIGHT-> 0M/2C
3M/2C <-BOAT LEFT    0M/1C
3M/0C   BOAT RIGHT-> 0M/3C
3M/1C <-BOAT LEFT    0M/2C
1M/1C   BOAT RIGHT-> 2M/2C
2M/2C <-BOAT LEFT    1M/1C
0M/2C   BOAT RIGHT-> 3M/1C
0M/3C <-BOAT LEFT    3M/0C
0M/1C   BOAT RIGHT-> 3M/2C
0M/2C <-BOAT LEFT    3M/1C
0M/0C   BOAT RIGHT-> 3M/3C



Solution using Depth First Search
This solution was found at level [12]

3M/3C <-BOAT LEFT    0M/0C
2M/2C   BOAT RIGHT-> 1M/1C
3M/2C <-BOAT LEFT    0M/1C
3M/0C   BOAT RIGHT-> 0M/3C
3M/1C <-BOAT LEFT    0M/2C
1M/1C   BOAT RIGHT-> 2M/2C
2M/2C <-BOAT LEFT    1M/1C
0M/2C   BOAT RIGHT-> 3M/1C
0M/3C <-BOAT LEFT    3M/0C
0M/1C   BOAT RIGHT-> 3M/2C
1M/1C <-BOAT LEFT    2M/2C
0M/0C   BOAT RIGHT-> 3M/3C
```

REFERENCES

- S. Russel and P. Norvig, *Artificial Intelligence – A Modern Approach*, Second Edition
- http://www.cse.unsw.edu.au/~billw/cs9414/notes/mandc/mandc.html
- http://en.wikipedia.org/wiki/Missionaries_and_cannibals_problem
- http://www.codeproject.com/Articles/16234/AI-Search-to-Solve-the-Missionaries-and-Cannibals