

# Missionaries and Cannibals Solution

Delos Chang

January 13, 2014

## 0.1 Introduction

## 0.2 Implementation of the model

The model is implemented in `CannibalProblem.java`. Here's my code for `getSuccessors`:

```
public ArrayList<UUSearchNode> getSuccessors() {
    // the final array to be returned
    ArrayList<UUSearchNode> retArr = new ArrayList<UUSearchNode>();

    int boatPlace = state[2];
    int candMissionaries = -1;
    int candCannibals = -1;
    int candBoat = -1;

    // Determine which missionaries and cannibals can travel
    if (boatPlace == 1){
        System.out.println("Boat_@_starting_side");
        // the candidate missionaries that can travel on the boat
        candMissionaries = state[0];
        candCannibals = state[1];
        candBoat = 0;
    } else if (boatPlace == 0){
        candMissionaries = totalMissionaries - state[0];
        candCannibals = totalCannibals - state[1];
        candBoat = 1;
    } else {
        // not valid input for boat
        System.out.println("Boat_place_not_valid:_ " + boatPlace);
        System.exit(1);
    }

    for(int missCount=candMissionaries; missCount>=0; missCount--){
        for(int cannCount=candCannibals; cannCount>=0; cannCount--){
            System.out.println("Checking_("+missCount+", "+cannCount+")");
            CannibalNode possNode;

            // Must fit in boat and have at least one missionary rowing
            if (missCount + cannCount <= BOAT_SIZE && (missCount > 0 || cannCount > 0)){
                // Must have something happen
                if (missCount == 0 && cannCount == 0){
                    continue;
                }

                if (boatPlace == 1){
                    // boat on starting side so starting side is subtracted
                    possNode = new CannibalNode(candMissionaries - missCount,
                                                candCannibals - cannCount, candBoat, depth+1);
                } else {
```

```

        // boat on other side so starting side is added
        possNode = new CannibalNode( state[0] + missCount,
            state[1] + cannCount, candBoat, depth+1);
    }
    System.out.println("(" + possNode.state[0] + " , " + possNode.state[1] + " , " + possNode.candBoat + " , " + possNode.depth + ")");
} else {
    continue;
}

boolean isSafe = isSafeState(possNode);
if (isSafe){
    // State is valid, add to the array
    System.out.println("Adding_" + possNode);
    retArr.add(possNode);
} else {
    // Node was not a valid state
    System.out.println("Not_a_safe_state!");
}
}
}
}
return retArr;
}

```

The basic idea of `getSuccessors` is that it returns an array of valid states based off of the node that was passed in. So, given the first start node: (331), it will run through each combination of missionaries and cannibals. So it checks 3 Missionaries and 3 Cannibals on the boat, then 3 Missionaries 2 Cannibals, then 3 Missionaries 1 Cannibal, 3 Missionaries 0 Cannibals, 2 Missionaries 1 Cannibal and so on.

Then, it will check if that combination is valid for the boat size. If it is, it will add or subtract (depending on where the boat is) to create the new state. This new state is passed into the `isSafeState` to verify whether the missionaries are safe.

I used a method `isSafeState` that returns `true` if the missionaries do not get eaten by the cannibals. We can check this by first validating that there are missionaries on either side of the river. Then we MUST make sure that the cannibals cannot outnumber missionaries on either side.

If there are no missionaries on one side, that implies that all the missionaries are on the other side. Thus, we just need to check whether the missionaries are outnumbered on the other side. This will cover the edge case where there are no missionaries but more than 0 cannibals on the same side. Without this edge case, the algorithm would not correctly process a node like (031).

```

// checks whether the humans get eaten :(
private boolean isSafeState(CannibalNode node){
    // miss + cannibals on starting side
    int startMissionaries = node.state[0];
    int startCannibals = node.state[1];
    // miss + cannibals on other side
    int otherMissionaries = totalMissionaries - startMissionaries;
    int otherCannibals = totalCannibals - startCannibals;
}

```

```

if (startMissionaries != 0 && otherMissionaries != 0){
    // must have more missionaries than cannibals or else eaten :(
    if (startMissionaries >= startCannibals &&
        otherMissionaries >= otherCannibals){
        return true;
    }
}

// If no missionaries are on one side, must mean missionaries are on other side
// Therefore, we check if the missionaries are outnumbered on the other side
// Also starting with 1 Cannibal and 0 Missionaries on starting side is NOT
// a valid state
if (otherMissionaries == 0){
    return startMissionaries >= startCannibals;
}

if (startMissionaries == 0){
    return otherMissionaries >= otherCannibals;
}

// not a safe state
return false;
}

```

### 0.3 Breadth-first search

### 0.4 Memoizing depth-first search

### 0.5 Path-checking depth-first search

### 0.6 Iterative deepening search

### 0.7 Lossy missionaries and cannibals