# COMP3270 – Assignment 1

Here I take notes about my approach and elucidate its strengths and weaknesses, and hacky workarounds.

Question 1 (a).

The code itself, though it might seem long, is quite straightforward. Most of the file is taken up by rather trivial code to generate successors given a node which I am sure can be made more terse (though less readable) by sneaky rearrangement of conditional blocks.

gets repeatedly called by with increasing values for . We know that a depth limited DFS is complete if is empty (that is, there is not any more nodes we can backtrack to). The last node to be emptied out is the starting node: . (technically then, by rearranging the first conditional block in the while loop, I could remove , but I kept it there just to spare me debugging headaches).

A node in my implementation is (rightly) not equivalent to state. Here a node is a len-3 array containing a state, the that we traversed to get to that state, and the depth of the node (which really is simply . The inequality of state and node is important, since a given state can be arrived at in multiple ways, taking multiple paths with different lengths. Skipping the expansion of a node with a state that is the same with a previously expanded node makes the search incomplete (and the result potentially suboptimal).

Question 1 (b).

Very similar business logic to question1\_a. The biggest difference is that , an unsorted array, is replaced by , a heap where returns the successor node with the best (smallest) heuristically estimated cost.

The misplaced tiles heuristic is admissible but not consistent, while the Manhattan distance heuristic is consistent. In theory, running the method with passed in as heuristic would find the solution with fewer expansions. And my program does seem to run faster with it as a heuristic.

Possible optimization strategies include memoizing Manhattan distance values, since computing them repeatedly means doing lots of divisions, which is a slow operation.

Question 2.

Problem formulation as a state-space search problem (following Russell and Norvig' s schema):

**States**: (no. of missionaries on left bank, no. of cannibals on left bank, boat location).

**Operators**: Move boat right if currently at left bank, move boat left if currently at left bank. 5 different possible passenger loads: MM, MC, CC, M, and C. Note that most states will have less than 5, since some operators will produce illegal states (death of missionaries).

**Goal test:** 0 missionaries on left bank, 0 cannibals on left bank, boat on right bank.

**Path cost:** Each time the boat traverses the river costs 1. Path cost is simply the number of times the boat crosses the river.

An optimal solution to this class of problems costs 11. My program uses Iterative Deepening DFS to search for an optimal solution. Here is its output:

Solved? True

Solution steps:

-> 2 cannibals

<- 1 cannibals

-> 2 cannibals

<- 1 cannibals

-> 2 missionaries

<- 1 missionaries and 1 cannibals

-> 2 missionaries

<- 1 cannibals

-> 2 cannibals

<- 1 cannibals

-> 2 cannibals