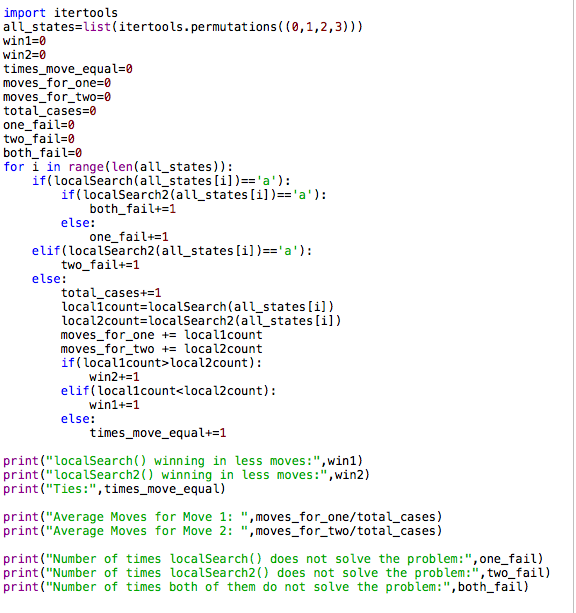
**ENGG1202- Introduction to**

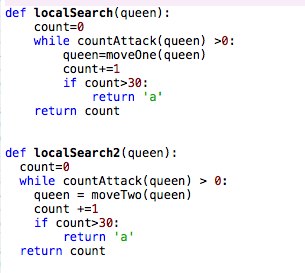
**Computer Science**

**Questions related to Local Search**

Q1. We can classify initial boards on the basis of the positions of the Queens. For boards, where all Queens are in different columns, I wrote the following code to find the average number of times, moveOne() and moveTwo() are executed for different board sizes. It also shows the number of times moveTwo () is better than moveOne().

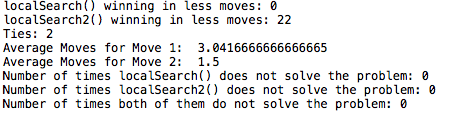


In the localSearch() and localSearch2() functions, a minimum count number has been introduced. This is to prevent the two functions from taking too much to find the solution for a particular state. If the count is exceeded, the functions automatically return a character, which tells the program below not to consider that particular case in the statistical analysis of that Local Search method.

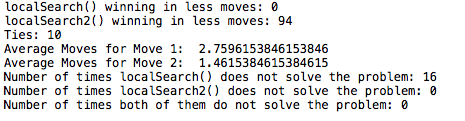
****

**Number of times moveOne() and moveTwo is executed**

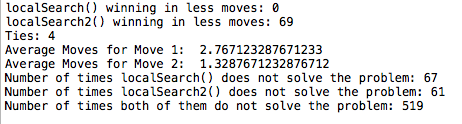
For a board of size-4 the count was set at 30. For a board of 4-size, the following results were obtained:

****

As we can see, moveTwo() is better than moveOne() in most cases as it solves the board in lesser moves. We can see that to solve an n-Queen board, moveTwo is executed almost half the times than moveOne() on average. A similar result was obtained for a 5-size board (the count was set at 100).



A similar result was obtained for a 6-size board (the count was set at 1000).



**Ability to reach the solution within execution time limit**

The minimum count sets the execution time limit for both these functions. As we can see from the first picture, both moveOne() and moveTwo() are able to reach the solution state within the execution time limit. If we look at the second figure, we see that localSearch() is not able to reach the solution in 16 cases whereas localSearch2() can reach the solution for all the cases. In the third picture, for 6-queens, the count was set we can see that the n-Queen problem is only solved 73 times out of a total of 720 times. Out of those 73, moveTwo() solves it in less moves 69 times and fails to find the solution in 580 cases while moveOne() never wins and fails in 586 cases. This clearly shows moveTwo() can reach the solution within the execution time-limit more number of times as compared to moveTwo().

If two or more queens lie in the same column then moveTwo() will never be able to reach the solution as it relies on swapping the positions of to queens. Since two queens will always lie in the same column, it will never be able to reach a solution for the particular n-queen problem. On the other hand, moveOne() maybe able to solve the n-Queen board for smaller values of n within the execution time limit.

For different board sizes of where the value of n is higher, both moveOne() and moveTwo() wont be able to reach the solution within the execution time limit.

Q2. No, search will not be successful if a longer execution time limit is allowed when the moveOne() function is used. This is because the moveOne() function is designed in such a way that if there are no better alternatives than the current state, which means that there does not exist a state with a countAttack() lower than or equal to the countAttack() of the current state, then the moveOne() function would make the program go to another state which has a higher countAttack(), but is the lowest amongst the worst. When the search moves forward, moveOne() function will be executed again. This time, it would be able to find a better state, but this better state would be the previous state that it has originated from. Now moveOne() will be executed again, and it will go to higher state again (the state it has changed from). This cycle will keep on happening, alternating between the lower and the higher state. And thus, the search function would never be able to find the solution even if the execution time-limit is increased. This can also happen when the countAttack() of a particular is state is ‘n’ and there are no better options available for moveOne() other than another state whose countAttack() is also n. This continuous cycle will also take place over here, and the search function will keep alternating between the two states. It is basically stuck on a plateau state. Hence, it will never be able to find the solution even if execution-time limit is increased.

Q3. The similarities of Local Search with other Search methods are as follows:

1. Bread-First Search (BFS): Both Local Search and BFS do not care about the path-cost and the number of steps required to reach the solution state. The only thing that matters to them is to reach the solution state. Thus both Local Search and BFS are not optimal.
2. Depth-First Search (DFS): Just like BFS, both Local Search and DFS do not care about the path-cost or the number of steps required to reach the solution state. The only thing that matters to them is to reach the solution state. Hence, both Local Search and DFS are not optimal.
3. Uniform-Cost Search (UCS): Both local Search and UCS are dependent on a cost-function/evaluation metric to evaluate their next move to find the solution.

The differences between local Search and other Search methods are as follows:

1. Bread-First Search (BFS): BFS requires an extra data structure to store its frontier (and also its explored states, in-case it is a GSA), but Local Search does not require any kind of data-structures and requires very little space.
2. Depth-First Search (DFS): DFS requires an extra data structure to store its frontier (and also its explored states, in-case it is a GSA), but local Search does not require any kind of data-structures and requires very little space.
3. Uniform-Cost Search (UCS): UCS tries to find a solution such that it reduces the path cost required to reach it, whereas local Search is only concerned about finding the solution state and not the path cost to reach it. Therefore, UCS is optimal but local Search is not.

**Question related to A\*Search**

Q4.

Macintosh HD:Users:PranavTalwar:Desktop:Screen Shot 2018-04-03 at 7.03.31 pm.png

Using the function provided in the assignment, A\*-TSA was executed and the solution to the above problem was found to be:

Macintosh HD:Users:PranavTalwar:Desktop:Screen Shot 2018-04-03 at 7.11.20 pm.png

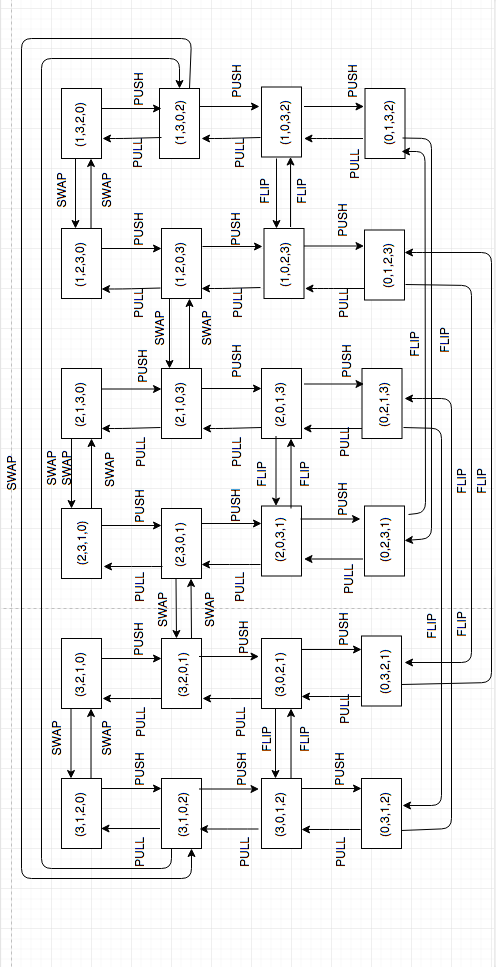
['PUSH', 'SWAP', 'PUSH', 'PUSH', 'SWAP', 'PUSH', 'FLIP', 'PUSH']

The cost of the moves as given in the assignment is:

* **PUSH**: 10 unit of time
* **PULL**: 5 unit of time
* **SWAP**: 17 unit of time
* **FLIP**: 8 unit of time

Hence, the cost of the solution is=(5\*Cost of PUSH operation) + (2\*Cost of SWAP operation) + (1\* Cost of FLIP operation)

=(5\*10) + (2\*17) + (1\*8) = 92



Q5. The starting state here is (2,3,1,0) while the goal state is (0,1,2,3). The state space graph for 3 balls is as follows

Q6. If the number of balls is N, this means there are (N+1) objects inside the tuple (including the zero as well). Since the first object can be assigned (N+1) positions, the second object N positions, the third (N-1) positions and so on and so forth, by Permutations and Combinations formula, the total number of states are:

Number of state-spaces= Number of options for 1st object x Number of options for 2nd object x Number of options for 3rd object x…………x Number of options for (N+1)th object

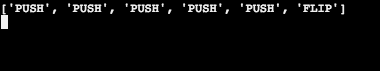
=(N+1) x N x (N-1) x ………… x 2 x 1

=(N+1)!

Hence the total number of state-spaces are (N+1)! .

Q7. The new heuristic function is not admissible. This can be proved

using the following example of (5,4,3,2,1,0). If we use A\*-TSA on this, we obtain the following solution:



If we calculate the true cost of the solution, it comes out to be:

Cost=(Number of PUSH operations\*Cost of PUSH operation) + (Number of SWAP operations\*Cost of SWAP operation) + (Number of FLIP operations\* Cost of FLIP operation) + (Number of PULL operations\*Cost of PULL operation)

Cost=(5\*10) + (0\*17) + (1\*8) + (0\*5)

=(50) + (8) = 58

The true cost of the solution is h\*(n)=58

Using the new heuristic calculation employed by James, the heuristic value is as follows:

P = Number of balls in the Holder x Cost of PUSH

=5\*10 = 50

Q = Number of balls in the Holder that is on the right of at least one ball with a greater value x Cost of PULL

=4\*5 = 20

R = Number of balls in the Package that is on the right of at least one ball with a greater value x (Cost of PULL + Cost of PUSH)

= 0 \*(10 + 5)=0

The heuristic value is therefore, P + Q + R = 50 + 20 + 0 = h(n) = 70

Since 0<=h(n)<=h\*(n), is not true, as 70>58, we can say that this new heuristic employed by James is not admissible.