Mohit Khatwani
AJ75499

Part I: Beads on Wire Problem

From what I understand I'm restating the problem as follows:

Given: We are given three beads with four faces on wire which have 3 colors on faces in a specific order.

- There are two green colors, one Red and one blue color
- Green color will always be opposite to each other
- Red and blue color will also be on opposite sides

Allowed actions

- We can rotate the bead by 90 degrees in any direction (anticlockwise or clockwise)
- We can rotate the bead-wire arrangement in space (though this will not help in reaching the goal state)

Goal State

- Goal state is achieved when all the faces are aligned according to color.

More on Goal state

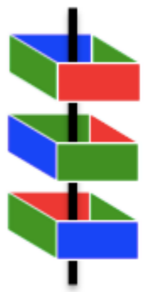
- The search space can be simplified by using some implications from problem
- As every bead contains 3 colors and 4 faces, we can consider a color which is not repeated (RED or BLUE) and check for goal state by checking whether faces of this color are aligned or not.

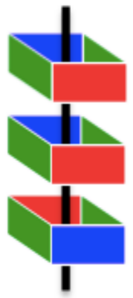
Here I'm assuming RED color as my condition for reaching goal state. BLUE can also be used for this checking but not GREEN, because there can be a case where GREEN color is aligned but the problem is not solved, as we cannot differentiate between two GREEN faces.

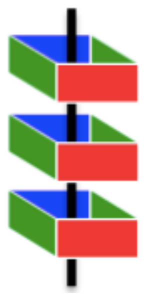
In the solutions proposed below I have assumed to move top and bottom bead. The middle bead is not moved in any direction.

Note that solution can also be given by selecting any two out of three beads, the state space generated will be same. The condition while solving this problem is that we should keep the non-moving bead constant till we get to solution

1. How many unique, legal, reachable states are there is this search space?
   - We have to select any two beads and align these two beads along the third bead
   - A State is defined as a [2*1] vector.
   - State matrix for following conditions will consider top and bottom bead with respect to middle bead
   - Vector is represented as [a,b] where a shows how far is color red in topmost bead from the middle bead. b shows how far is red color face of bottom bead from red face of middle bead


(S): [-1,1]


(M): [0,2]


(G): [0,0]

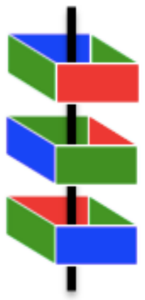
There are 4 possible ways in which first bead can be aligned 0, -1, 1, 2. Same is the case with second bead. So, we have to select a number for first bead as well as second bead. This can be done in ${}^{4}_{1}C.{}^{4}_{1}C$ possible ways i.e. 16 ways.

There are 16 unique, legal and reachable states in this search space

2. What are legal operations that fully encode this search problem?
    a) **Operation 1**: Select any two beads out of the three given beads
    b) **Operation 2**: Rotate the selected bead in left direction (clockwise direction) by 90 degrees
    c) **Operation 3**: Rotate the selected bead in Right direction (anti-clockwise direction) by 90 degrees

3. Are there any loops in search space?
    - **YES, there are loops in this search space**
    - Suppose consider a case where we rotate bead in clockwise direction and after that rotate the same bead in anti-clockwise direction then the state will be repeated.
    - I'm representing search space in form of graph

4. Given the order of operation above (ex. the search space is expanded in a-b-c order), how many states are visited in depth-first search?
    - Given the order of operation above **12 states** are visited in DFS until goal state is reached using following start state.
    - Output of DFS from following start state will be,
      [-1,1] -> [2,1] -> [1,1] -> [0,1] -> [1,0] -> [1,2] -> [2,0] -> [-1,0] -> [2, -1] -> [2,2] -> [-1,2] -> **[0,0]**



[-1,1]

5. If you are using heuristic search, what heuristic would you use for evaluating states?
    - My heuristic value would be calculated as the absolute values of contents of vector which I'm using to represent states.
    - This heuristic value would represent how far is the selected face from other in reference
    - Heuristic value for vector [a, b] can be written as **h = |a| + |b|**
    - For example, consider above states given, my heuristic value would be 2 = |-1| + |1| for State S, |0|+|2| = 2 for State M, |0|+|0| = 0 for state G.

6. Of the informed, uninformed and local search methods(only) that we have looked at:
    1. What algorithm would you choose for beads on a wire problem in general?
        - I would choose **IDA*** algorithm
    2. Why is this algorithm a good choice?
        - Using an informed search with proper heuristic would be the best choice. There are various options in informed search algorithms, out of which I have chosen IDA* because of its space requirements. A* also would have been good choice, but it requires space in the order of b^d where *b* is branching factor and *d* is depth of solution. IDA* has lower memory requirement of *O(d)*

Part III: Programming Choices

1. What search algorithm did you implement in Part II?
    - I have implemented **A* search algorithm**

2. Why did you choose that algorithm? Why was it a good choice compared to the others?
    a) Out of the 5 algorithms given: A*, beam search, breadth-first search, Depth-first search, Iterative deepening. According to me an informed search algorithm would be the best choice, because it will explore only those states which are useful or which will help in reaching goal state optimally.
    b) There are two informed search algorithm available A* search and beam search. Beam search will not be useful here as there are only two possible operations for this state (going left or right). Beam search is useful where there are more number of operations i.e. branching factor is more.

3. What advantages does it have for this algorithm? What disadvantages?
    a) **Advantages**:
        i. A* will require less memory than other uninformed search algorithm as it will not generate child nodes for those states which will not reach the solution.
    b) **Disadvantages**:
        i. Time Complexity of A* algorithm is exponential O(b^d)
        ii. Space Complexity of A* search algorithm is O(b^d), which can be improved by using IDA* search algorithm

4. If you had to use a heuristic search algorithm, what would you use? Why?
    - I would use **IDA*** to solve this problem because it has lower memory requirements than A*.
    - If the solution is present at depth *d* then **space complexity** of IDA* is O(*d*) which is much less than exponential complexity of A*

REFERENCES:

1. Artificial Intelligence: A Modern Approach by Stuart Russell, Peter Norvig
2. https://en.wikipedia.org/wiki/Iterative_deepening_A*
3. https://en.wikipedia.org/wiki/A*_search_algorithm