Mini-Project 2:
CS 7637

Ryan Johnson
ryan.johnson@gatech.edu

1 PROBLEM OVERVIEW

The goal of the problem is to move a set of blocks stacked in any given order to some other goal state that may be the blocks stacked in any other given order.

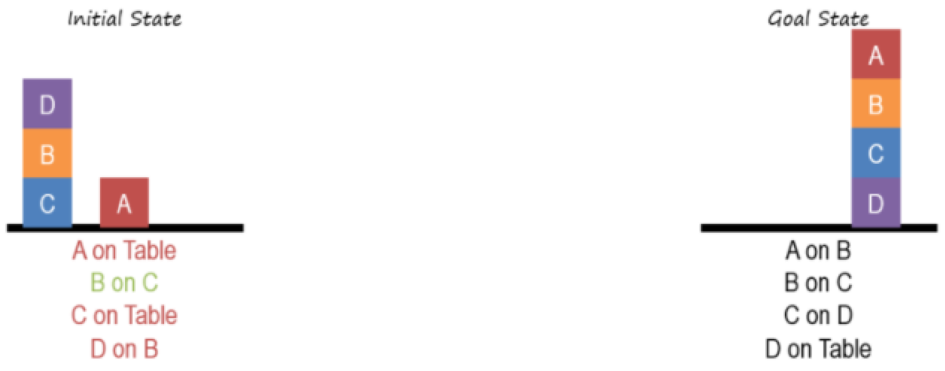


Figure 1 A sample initial and goal state. The goal of the problem is to move blocks one at a time to the top of another block or to the Table

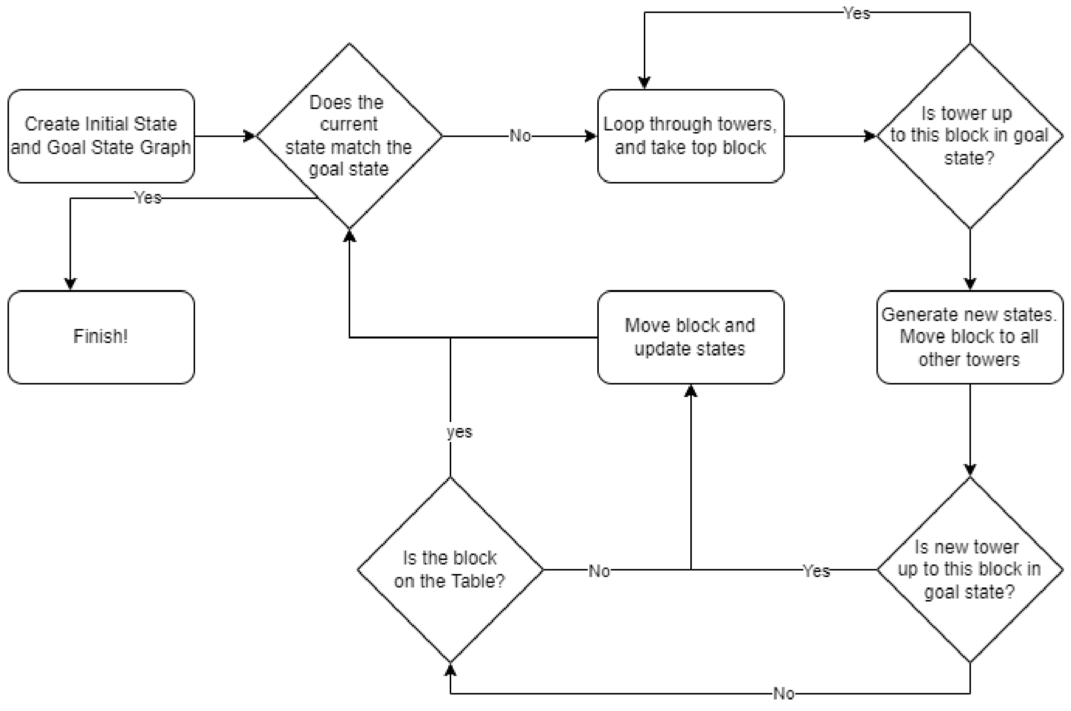
The challenge that lies within the problem is building a representation or semantic network of the state of the blocks, and then informing our agent with the proper logic to solve the problem without getting stuck in local optimas.

2 HOW THE AGENT WORKS

My agent follows a smart generate and test methodology that takes a simple heuristic framework I devised from solving block puzzles by hand, and then implemented into my solving agent. It starts with a basic semantic network represented as a dictionary with keys as unique blocks and values represented by what they sit on. The initial state above would be represented as follows:

{‘D’:’B’, ’B’:’C’, ’C’:’Table’, ‘A’:’Table’}

This structure represents the entire state and makes it easy to test for the correctness of both the entire state as well as subgoals. For example, I can start at any given block and loop through the keys in both the present state and the goal state. If there is any difference before we get to the "Table" we know from the starting block down that the tower is not in the correct state and can then take action to move blocks. With this in mind, let's take a look at the heuristic used by my agent to solve the problem.



Create Initial State and Goal State Graph
Does the current state match the goal state
No
Loop through towers, and take top block
Is tower up to this block in goal state?
Yes
Yes
Finish!
Move block and update states
Generate new states. Move block to all other towers
yes
Is the block on the Table?
No
Yes
Is new tower up to this block in goal state?
No

Figure 2 A representation of the logic the agent uses to solve the block puzzle

The agent starts by creating the representations of the initial and goal states. Then it enters a series of loops. If the states don't match, we know there is a block out of place and need to find which moves we need to make. To do this we take the top block from each tower and check the position of that block against its position in the goal state. If they match, then we know that at least that subtower matches the goal state, but if they don't we know we have to move the block to get to the destination. To do this we generate all possible moves for this one block to the top of each other tower, as well as to the table. We then check if any of these new towers are present in the goal state, if they are, we make the move there and update our state. If none of them are, we move the block to the table for use later if it is not already there. This deconstructs a tower down to the point where a block needs to move while also ensuring we don't make extra moves by moving too many blocks to the table.

3 PERFORMANCE AND EFFICIENCY

3.1 How the agent performs

I've found the agent performs very well! It accurately finds a solution to any given state in the optimal number of moves. We can be certain of this because of the built-in logic of the agent. We only ever move a block if there is something incorrectly stacked underneath the block. With this in mind, we know we will have to move that block in order to solve the problem.

3.2 Agent Efficiency

My agent is practically very quick and efficient for the current scope of the problem. Even with all 26 blocks in a scenario, we are still able to solve the problem in the worst cases are around a second, with the majority of generated states being solved far quicker than that.

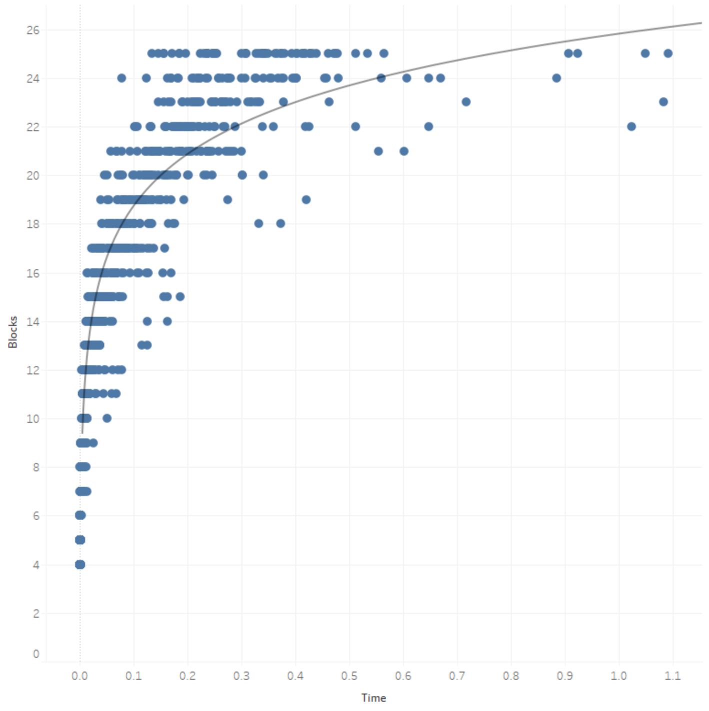


Figure 3 Agent performance by time with respect to the number of block

Here the growing tax of searching through a greater number of towers and states as we add more blocks. We see a strong cluster of performance times under the half second mark for scenarios with 20-26 blocks, however, there are some cases where we see outliers that double or triple the average performance time by number of blocks. I believe this is due to how many blocks may be moved to the table in these scenarios. If we have a large number of blocks on the table, we expand our search space for every test we do of new placement locations. While this is not a problem for the current agent, if we increased the number of blocks further to 100s or 1000s, I suspect this would be a significant performance hit. In order to solve this problem, the agent would need to be able to more cleverly place blocks into new stacked towers instead of the table. The table offers many advantages in solving this problem as it maximizes the number of blocks that are available to be moved when we find a correct location for the block to be placed. However, this creates more potential states to search through and isn't technically needed. For example, creating an inverse tower would be just as effective as placing many blocks on the table.

This isn't to say the agent doesn't do anything clever, however. We are already significantly pruning the search space by testing the correctness of a given stack or substack rather than an individual block.

4 Comparison to a Human

I think this is a very humanlike way to solve the problem. I came up with the process after solving several of these problems myself by hand and thinking about how I solve the problem. Humans tend to look lower in the tower to try to find a block that needs to be released in order to move it to the goal state. This is the exact same underlying principle that my agent uses, just in a more formalized and rule based fashion. However, one place where the agent differs, I think, is in generating and testing the moves once we know a block isn't in the right place. As a human, I can immediately trim my potential move locations down to typically 1 spot right away. This may be partly due to working the problem a few steps ahead, but also an intuition of the difference between a block on the table that's just a temporary placeholder and a tower that I am actively creating into the goal state. My agent doesn't differentiate between these two. Both are just as valid a location to place a block, and will be tested in turn.