Mini-Project 2

Jameson Cook
jcook323@gatech.edu

1 Agent Design

My agent is designed to use a combination of the problem solving methods Generate and Test, Means End Analysis, and Problem Reduction. My agent uses a similar method used in Mini Project 1 of generating all possible next states for each state it is visiting based on the list of total actions available created from the problem's given blocks. The generator is relatively underpowered and only checks if the state is a repeat, while relying on the tester to ensure a move is valid before adding it to a Priority Queue of next states to visit. The tester determines both whether a given move is valid based on the current state and whether the move will effectively lead towards the goal: examples of ensuring valid moves include checking if the move would create the same state, if the block being moved is currently on the top of a stack, and if the block that that the moved block will be placed upon is currently at the top of a stack or alone on the table. The tester also checks for productive moves by only allowing moves that would place the moved block on another block and creating a "pair" that are present in the goal as well as only allowing moves that create a "pair" which are found at the top of a stack in the goal if the rest of the stack beneath is fully in place.

My agent expands upon the idea of a Means Ends Analysis implementation for the Block World problem in the lectures and implements it in two ways. When adding valid and productive next states to explore, the agent calculates a "distance" value between the next state and goal state based on prescribed heuristics. As its method of choosing the operator that will reduce the differences between a current and goal state, the agent utilizes a priority queue to choose the next state and associated action by picking the next state with the lowest heuristic distance. Table 1 outlines the heuristics that the agent calculates for the goal state and every state generated to determine a quantified distance between a given state and the goal. As can be seen in the heuristic value added when a state matches a goal states condition, the agent places an emphasis on states that have the bottom of every block stack in the goal in place by more heavily weighting matching single blocks, bottom block pairs, and bottom three blocks.

Table 1—Heuristics and Associated Conditional Values To Calculate Goal Distance

Condition	Bottom Blocks	Top Blocks	All Block Pairs	Bottom Block Pairs	Top Block Pairs	Bottom Three Block Pairs
Match	+7	+5	+7	+5	+10	+20
Missing	-3	-1	0	-7	-5	0

When designing the agent, I noticed that it had difficulty prioritizing an effective operator at the beginning of a given problem using the above implementation of Means Ends Analysis and prescribed heuristics. Therefore, I decided to apply a mixture of problem reduction and production rules to assist the agent in finding optimal starts to block problems. I found and prioritized the three most critical sub-problems that each larger block problem consisted of: get all bottom blocks in place, get all single block stacks in place, and get all bottom block pairs in place. I implemented the ability for the agent to prioritize these sub-problems by using a set of rules that the generator would check against and assign a conditional move to take next if the current state, or percepts, matched the goal values it was compared against from a problem's long term memory.

2 Agent Performance and Efficiency

When measuring my agent's performance and efficiency, I found that it's computational time to solve problems increased drastically as the number of towers between the goal state and initial state were more equal as seen in Figure 1. Therefore, I analyzed my agent performance in terms of computation time in seconds and efficiency in terms of states explored vs states added to its queue against problem difficulty as measured by the difference in number of towers from goal to initial state and total number of blocks in each problem.

My agent was able to solve all problems given to it, but, as seen in Figure 2, the time it took to solve problems increased greatly when the number of towers between the initial and goal states were close together, mostly within a difference of 2, and as the number of blocks surpassed 15. I attempted to measure my agent's efficiency by comparing the number of states it added to its priority queue and number of states it visited against the problem difficulty as seen in Figure 3. This reinforced the prior conclusion that the agent had difficulty with problems in correlation with the increase of blocks while holding the number of towers equal. This can be seen by how many more states the agent had to add its queue to solve these types of problems.

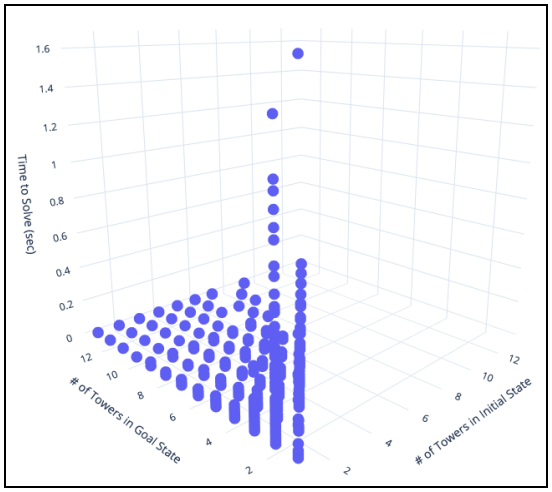

Figure 1—Time to Solve Compared To Number of Towers in Initial and Goal State

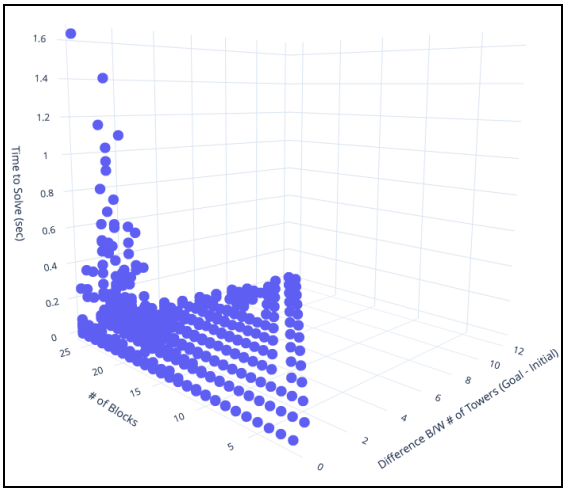

Figure 2—Time to Solve Compared to Diff in Towers B/W Goal & Initial State and Number of Blocks

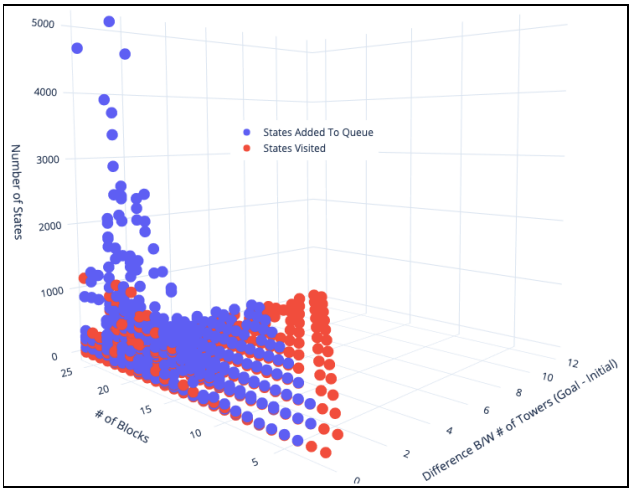

Figure 3—States Visited & Added to Queue Compared to Diff in Towers B/W Goal & Initial State and Number of Blocks

3 Cognitive Connection

My agent uses problem reduction in a way that is quite clever and similar to how humans would solve Block World problems. Like a human, my agent knows that in order to start moving productively towards a solution all of the bottom blocks of a stack must be in place. Therefore, it prioritizes this task first and does not explore any states that will continue to bury critical bottom blocks underneath newly placed blocks. On top of being intelligent enough to determine what an optimal first subgoal is, the agent employs the technique of moving all blocks on top of a target subgoal block, i.e. block that is on the bottom of a stack in the goal state, until it is able to directly interact with that target block. This method of analyzing the goal state and immediately prioritizing moves that uncovered and moved critical bottom building blocks into their goal position is exactly how I solved the Block World problems I encountered.

My agent also uses heuristics to assess the difference between each candidate state and the goal state in order to select a next state that moves closest to the goal state; however, the heuristics that my agent employs are quite basic and the agent has no way of breaking a tie in heuristic distance intelligently. In the development of my agent, I could not understand why it would make some initial moves based on the heuristic calculations that I created. Humans have a more intuitive understanding of assessing candidate states and determining optimal vs wasteful moves, where my agent had to rely on a simplified quantitative calculation. Because the agent did not have a way to intelligently assess and choose between next states that had the same heuristic distance, it would choose the first of the tied next states and often lead towards an inefficient solution. Due to this lack of ability to discern successful moves early in the problem or efficiently calculate them in a heuristic, I instead chose to rely on problem reduction to solve the early moves of each problem.