Mini-Project 2: Block World

Katherine Isabella
kisabella3@gatech.edu

1 AGENT DESIGN

The agent works using a generate and test method with a focus on best states to keep the number of potential states limited. At the start, the agent takes the initial and goal arrangement and puts them both into dictionaries. These dictionaries contain block objects for each block with each object containing the block on top and below the current block as well as whether or not the block is in place. The block is considered in place if every block underneath the block is in its correct location according to the goal arrangement. The agent then puts the initial state into a queue in order to find the optimal path of moves to solve the problem.

To find the optimal path, the agent loops through all the states in the queue until the current state and goal state match. On each loop, the next state in the queue is removed and becomes the current state. The agent then begins looking at the next potential states based off the current state. To do so, the agent looks at all the blocks which are on top, but are not already in place. To understand how the states are generated and chosen, Figure 1 has been provided.

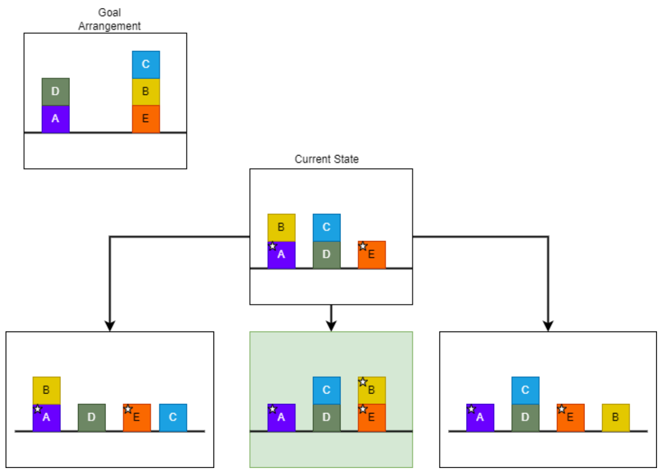

Figure 1 — Example of the agent's generate and test method

Figure 1 displays a current state, the goal arrangement, and the potential next states. The agent only generates states that are potentially helpful so it will only generate states where the block will either be on the table or in its correct place on a block column. In Figure 1, the blocks in their goal place already are denoted with a star. Therefore, out of the three states generated, there is one clear best state which is marked in green. This best state is the only state the agent will need at this time so the other states will be discarded while the best state is added to the queue. If there is more than one best state, then the first best state found is the only state added to the queue with the understanding the other best state(s) will be generated again and chosen in the future. However, if there is no best state, then *all* the potential next states are added to the queue as the agent does not know which state would be most optimal and must check them all. It is also important to note that no duplicate states are added to the queue and the states are checked against previous states before being added.

After this state selection, the next state in the queue will become the current state and the process will continue until all the block positions match up with the goal state. Once the goal state has been reached, the agent will then use the final state to retrieve and return the optimal path.

2 PERFORMANCE

The agent performs well in that it can find the optimal solution to all Gradescope test cases as well as a series of randomized test cases I created. With that being said, initially the agent did struggle with finding the optimal solution due to the generate and test method the agent uses. At the start, the agent struggled to complete the tests in time and the tests timed out because all potential next states were examined and the agent was not using best states. However, by adding in the best choice as described in the design section, the number of states generated and tested was drastically cut down and the performance increased.

3 EFFICIENCY

Looking at efficiency in terms of runtime, the agent has an efficient runtime as it is able to run through all test cases –even test cases with 50 blocks– in under a few seconds. In order to test the runtime efficiency, I tested a variety of scenarios with different initial and goal column amounts alongside the number of blocks increasing. Figure 2 shows how as the number of blocks increases so to does the runtime.

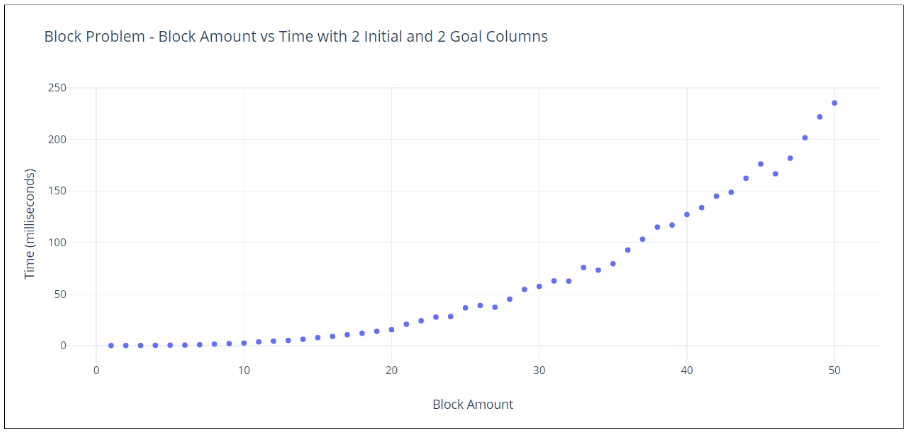

Figure 2— Agent runtime efficiency for 2 initial and goal columns as the number of blocks increases

Although the runtime remains under 250 milliseconds, one can see the runtime increasing as the number of blocks increases, likely due to an increase in the amount of states which are generated. With more blocks, the amount of states which branch off increases which in turn increases the runtime. Figure 2 shows a scenario where the column amounts were the same. If the number of columns in the goal and initial arrangements are different this runtime increase still occurs, which can be seen in Figure 3.

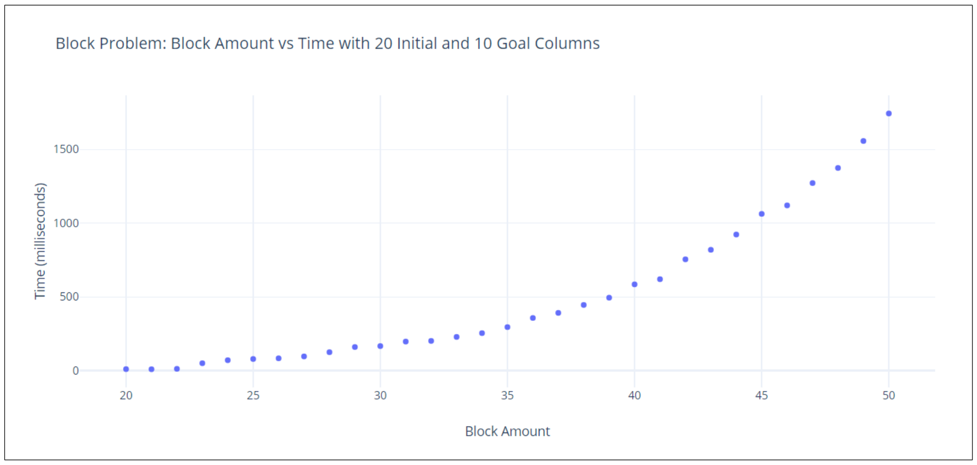

Figure 3— Agent runtime efficiency as number of blocks increases for differing initial and goal column amounts

4 FINDING AN ANSWER EFFICIENTLY

In order to arrive at an answer more efficiently, the agent prioritizes certain next states and discards the rest. When I first began looking at the block problem, I realized the only states which should be generated using the top blocks is to either move the block onto the table or to move the block onto a block-column only if the block and all the blocks below will be in their correct goal state. The agent is designed to only generate these states, and once all those states are generated the agent is also designed to pick the best potential next state and discard the rest. If one of the next potential states will put the block in its goal location, then that is considered a best state. Therefore, the agent is smart enough that when generating the states, if a best state exists then that state should be the only state added to the queue of next states to continue the generate and test method. By using this method, the generate and test process does not balloon into having far too many states to efficiently run through. Instead, by pruning the states, the agent can efficiently solve the problem and find the optimal path.

5 AGENT VS. HUMAN

5.1 Similarities

For cases where the number of blocks is small the agent solves the problems in a similar manner a human would. I designed the agent after first going through some problems by hand and basing the agent off how I solved the problem. When there are only a handful of blocks, a human is likely to use the generate and test method to find the optimal path similar to how the agent does. However, when the number of blocks and columns increase, the similarities between human and agent begin to disappear.

5.2 Differences

When the complexity of the problem goes up such as when the block count increases, the agent begins to solve the problem in a more optimal and efficient manner than a human. During scenarios with a high block count, as a human I am not likely to find the optimal path. For example, if there were over 10 blocks, I would likely just put all the blocks on the table and then just restack them into the goal arrangement. While the path would not be optimal, the time it would take to find the optimal path would be longer than I would want to spend on the problem. However, unlike a human the agent does not get bored of the problem and can go through the numerous states very quickly. Even when the number of blocks increases to large amounts, the agent will go through the many potential states in order to find the optimal path.