Mini-Project Two: Block World

Allen Worthley

mworthley@gatech.edu

1 INTRODUCTION

This document discusses the Mini-Project Two and the AI Agent code set that solves the Block problem. The problem entails an initial set of blocks and a goal set of blocks. The AI agent must determine the optimal moves needed to reconfigure the blocks from the initial set to the goal set. The proceeding sections demonstrate the design of the solution code, how it performs, and how it compares to human rationale.

2 DESIGN

The AI Agent uses a combination of the "Generate and Test" method, Means-Ends Analysis method, and a heuristic algorithm to determine the optimal solution. The generator attempts to minimize the unproductive states it produces to increase efficiency. A Means-Ends Analysis confirms whether or not a generated state is productive. The heuristic algorithm divides tasks based on the current state and the logic of the problem. The following sections dive into the more granular level of design including the design of the overall data model, generator, and validator.

2.1 Data Model

There are four important data structures that organize the program: a list of branches of states (Master_Tree), a list branches of moves (Moves_Tree), a list of non-duplicated previous states(Memory) and the Towers class. Figure 1 shows a high level representation of how the data is structured and related. Generally, the data structure centers around the Towers class which stores useful

information when processing through the different states. Also, the target is a dictionary with keys for each of the goals blocks and the corresponding value for the target block underneath.

The structure of the Master_Tree and Moves_Tree align; meaning, for a given branch in Master_Tree, the same branch index for Moves_Tree yields the corresponding moves made by the states in Master_Tree. Additionally, the branch index is tracked at the states level in branch_id.

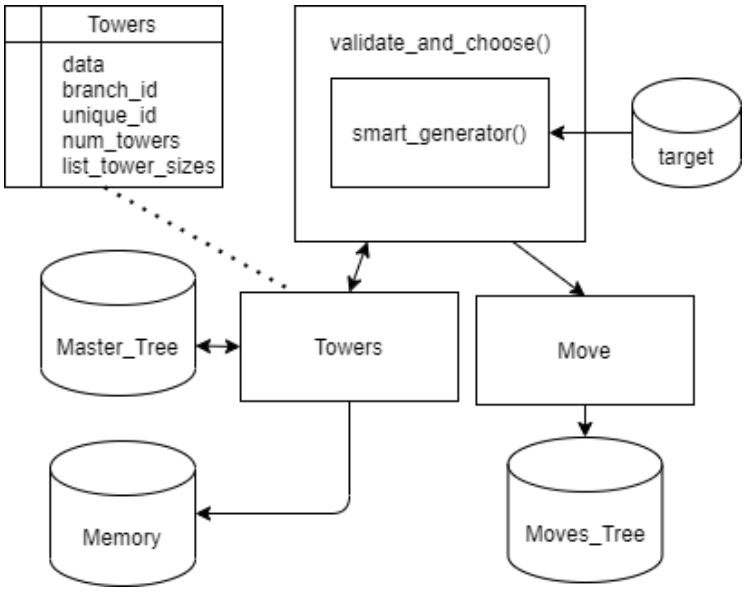

Figure 1— Data structure and relations.

2.2 Generating the Tree

The smart_generator() function follows the below heuristic to generate the states:

- In the current column, if all blocks below in the current column match their target, see if the other columns' top blocks have the target block for the top block. If so, add the target block to the current top block.
- If not all blocks below match their target, remove the current block and find a place to put it.

- If by adding the removed block doesn't increase the score of the next column, add to the table
- Else above, skip the column and iterate to next.

The design of the AI agent uses a mixture of a Generate and Test and the Means-Ends analysis because of the score functionality built into the agent. It evaluates whether or not each new state is closer to the goal set.

2.3 Scoring and Means-Ends Analysis

The agent uses two scoring functions: one for the column to be used in smart_generator, and two, for the entire validation. The scoring functions essentially work the same. A for loop iterates through each block of the state and compares whether the block below the current block is the target block; if not, value assigned for the block is the count of all above blocks in that tower or column.

2.4 Validating the states

The validate_and_choose function takes in a list of generated states and scores each state. It also checks for whether the state is a duplicate. The function returns the states with the lowest score or ranking and checks if a given state is the solution.

2.5 Wrapping it All Together

A While loop iterates on the condition that the number of new states generated is greater than zero and the branch containing the solution state (where all zeros are on the left coast) has not been found. While the condition is true, the AI Agent loops through all branches of the Master_Tree, calls the smart_generator function, and decrements the number of new states generated. Each iteration of the smart_generator() function adds new branches to the Master_tree for each

new state generated. Once the branch containing the solution is found, the program outputs the list of moves in the associated branch of the Moves_Tree.

3 PERFORMANCE

The cProfiler in python measures the performance in seconds of the AI Agent solving scenarios where the number of columns in the initial state and goal state are the same. The agent performs okay under small initial sets as shown in figure 2. However, the AI Agent cannot handle more than four columns in either the initial or goal set.

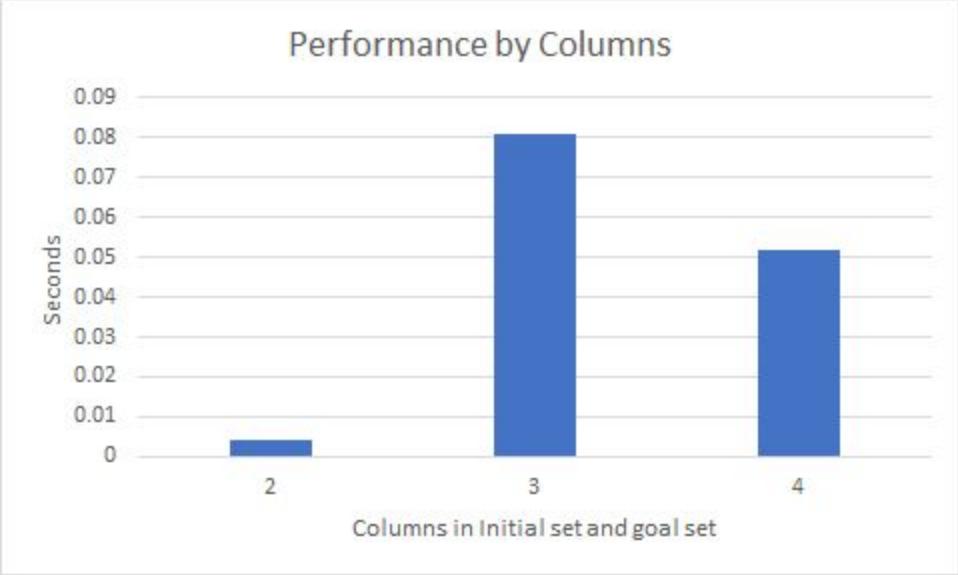

Figure 2— Mapping logic between moves_tree and master_tree.

The AI Agent can definitely be improved. The current build uses repeating data for convenience and heuristics requiring too high of complexity. Previous states are repeated for each branch of both the Master_Tree and Move_Tree.

4 HUMAN COGNITION RELATION

The AI Agent solves some of the problems but does not solve all. While almost trivial to humans, finding the best next move is quite challenging for the AI agent. Yet, if the agent was optimized a bit more, the agent could beat a human as a computer has more explicit memory than a human. And, humans are bound to make mistakes.