Mini-Project 2: Block World

Michael Maimoni
mmaimoni3@gatech.edu

***Abstract*—**In this document, we briefly describe the design, performance, and efficiency of an agent created to solve the Block World problem. In-addition, we expand on how the agent's approach to the problem compares to its human creator's approaches.

1 DESIGN OF THE AGENT AND A NOT-SO EASY WAY OUT

Early attempts to solve the Block World problem began with an "easy-out" agent that would simply unstack all non-bottom blocks, place them on the table, and then build all stacks in the target arrangement. While this approach is generally effective, it lacks a certain elegance (and it seems prone to sub-optimality, given no regard is ever given for whether a block could be used on another stack).

We recall from Professor Goel's characterization of knowledge-based artificial intelligence being composed of three fundamental processes: *reasoning*, *learning*, and *memory* - using this framework, the concept of a *semantic network* as a knowledge representation, the problem-solving technique of *problem-reduction*, and a rudimentary *production system* as our cognitive architecture, we begin our discussion of the alternative to the "easy-out" method mentioned above.

Specifically, the chosen alternative was an agent that begins by creating objects (*i.e.*, Python lists and sub-lists) in its working memory to allow for hypothetical state assessments as well as retrospective references - these include (but are not limited to) the original arrangement ("**original_arrangement**"), the target arrangement ("**target_arrangement**"), as well as a transitory / working state (call it "**current_arrangement**") with a list of empty lists (one for each stack in the target arrangement), into which & out from which blocks are moved in-alignment with the rules of Block World. The agent then uses a series of heuristics derived from identified sub-problems under the problem reduction approach:

(a) If the bottom block in a stack within the original arrangement is the first block in a stack in the final state, the agent substitutes the original stack for the corresponding, empty list object in **current_arrangement** by index number - original stacks with no matching bottom blocks relative to the target arrangement are placed in a separate list object ("**stacked_blocks**") as incorporating them into **current_arrangement** is net-neutral at-best;

i. *This step facilitates index matching against counterparts in **target_arrangement**;*

(b) For each stack in **current_arrangement**, the agent removes top blocks until the length of the stack is less-than or equal to its counterpart in **target_arrangement**, placing each in another list object ("**unstacked_blocks**") which represents the table, and recording each move;

i. *This step is intuitively appropriate move, as an optimal solution would not stack more blocks on top of a stack than exists in the target arrangement - however, the agent should be refined for cases in which other stacks require removed blocks (preferencing placement atop those stacks over the table);*

(c) For each stack in **current_arrangement**, the agent then attempts to match blocks by index number against their counterparts in **target_arrangement** from top to bottom. Matching blocks remain unchanged, whereas unmatched items are popped and placed on the table (and the moves recorded);

i. *Again, this step is intuitively appropriate move, as each stack should be reduced to the component blocks matching its counterpart in **target_arrangement**, similar enhancement opportunity to item (b) above noted;*

Following application of the above solutions to identified sub-problems, the agent then begins applying a fixed set of *production rules* (Stefik, 1995) to each stack and underlying block index in **current_arrangement** until reaching parity with **target_arrangement**. These rules can be summarized, generally, as follows:

1. If, within the given stack, the block at the given index matches its counterpart within **target_arrangement - Do nothing and pass.**
2. If within the given stack, no block is present at the given index...
   (a) If the needed block is on the table and the given index is 0 - **Reflect the block at index 0 of the given stack. Do not record a move.**
   (b) If the needed block is on the table and the given index is anything other than 0 - **Move block to the given stack at the given index; record move.**
3. If the needed block is not on the table, iterate through all unused stacks (**stacked_blocks**) from **original_arrangement**, considering their topmost blocks only...
   (a) If the needed block is the topmost block of the searched stack...
      i. If the given index is 0 (*i.e.*, the block is to be placed on the table) - **Move top block from searched stack to table; record move.**
      ii. If the given index is anything other than 0 - **Remove top block of searched stack; place it atop given stack; record move.**

(b) If the needed block is not the topmost block of the searched stack but is in the searched stack - **Remove the topmost block from the searched stack, place it on the table, and record the move.**

The agent continues to apply these production rules until its working state, **current_arrangement**, matches the target state, **target_arrangement**. Stepping back, we can conceive of each of the list objects (*i.e.*, **target_arrangement**; the post-heuristic **current_arrangement**, **stacked_blocks**, and **unstacked_blocks**; etc.) as *percepts* to which the agent has access. Given the fixed set of production rules, the agent's behavior is solely based on the *content* or *percepts* it is provided.

While this problem may be a basic example, it still serves to show the power of a properly designed set of production rules (itself, a *cognitive architecture*), this is conceptually quite appealing, as we may imagine that for more complex problems, assuming an adequate architecture can be identified and constructed, optimal agents can be created (if only within a fixed sphere of activity). As noted above, some improvements to the upfront heuristics would likely be beneficial, though see below for results of independently developed automated testing.

2 AGENT PERFORMANCE REVIEW

In attempting to form a supportable view on the performance of the agent, we subjected the agent to multiple instances of independently developed, automated tests (available on Gradescope). Instances used eight (8) previously known initial and target arrangement pairs, as well as unseen, pseudo-randomly-generated sets of twelve (12) initial and target arrangement pairs with up to 26 blocks each, to determine whether the agent generated a legal path to the target arrangement, as well as whether generated paths were optimal. This testing indicated perfect performance on all tests, without deviation on five (5) successive trials.

Given the above, supplemental testing was not deemed necessary to support the agent's ability to perform its intended function of solution-path generation, however, additional investigation into the agent's efficiency was conducted.

3 EFFICIENCY (AND THE EFFECTS OF INCREASED COMPLEXITY

In concluding on agent "efficiency", we can plausibly take the term to mean either some measure of computational complexity of the agent, or the ability of the agent to determine the optimal path, regardless of increases in the num-

ber of blocks / complexity of the target arrangement relative to the original arrangement. Given we have already explored the agent's ability to determine the optimal path above in our discussion of performance, we will not revisit that topic - instead, we focus on time complexity (refer to **Figure 1** below).

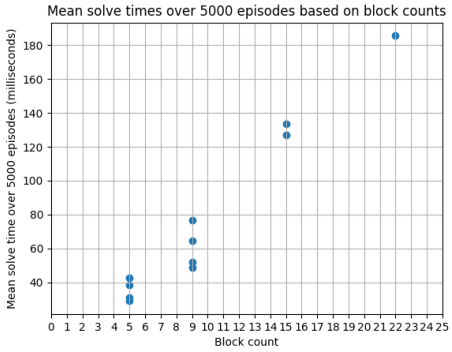

Figure 1—Scatter of mean solve times by block count over 5000 trials; Source: Author

A review of mean solve time results taken from 5000 trials (to avoid single- / low-count trial error) for various configurations reveals a general (arguably linear) increase in solve time as total block count rises.

4 DOES THE AGENT BEHAVE IN A CLEVER MANNER?

Beyond its heuristics (improvements pending), any 'cleverness' the agent shows is simply the result of the production rules noted above.

5 CREATOR VS. CREATED - COMPARISONS IN APPROACH

Reflecting on the approach taken by the agent and by me, its human creator, I noted a couple commonalities and some differences including:

- The agent's heuristics are similar to what I might think of, but remain sub-optimal (from my perspective). If I applied similar approaches, I would preference first moving blocks to other stacks instead of the table if such a move would be more ideal. Future iterations may incorporate such techniques (perhaps through *means-ends analysis*, tracking distance to goal);
- As a human, my short-term memory is definitely inferior to that of the agent's - I doubt that I would be able to consistently apply the developed production rules to complex arrangements without error.

6 REFERENCES

[1] Stefik, Mark (1995). *Introduction to Knowledge Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 155860166X.