KBAI - Mini Project 2: Block World (Spring 2022) Project Report

Badri Narayanan CG
cgbadri@gatech.edu

***Abstract*—An agent that solves the Block World problem for an arbitrary initial state has been implemented as part of this mini project. In this problem, an initial arrangement of blocks and a final arrangement both represented by a set of blocks are given. Each block is represented by an alphabet (A-Z) and is unique. The agent calculates the most optimal set of moves required to transition from the initial arrangement to the final arrangement. The constraints are that only one block can be moved at a time and that a block cannot be moved if it has other blocks on top of it.**

1 INTRODUCTION - IDENTIFYING THE RIGHT APPROACH

Defining the heuristics or scoring system for ranking the state transitions for this problem is very subjective and open to different interpretations. As a result, it was not easy to attempt methods like Best First Search or A* search which are popular algorithms for search problems. Breadth First Search could result in exponential expansion of the search space. The lectures themselves show why Means-Ends Analysis won't work well. One hint from the lectures was that future lessons will show better ways to solve such problems. Skimming through future lessons led to identifying the method implemented by this agent. **Lecture lessons 13 - Planning** along with **Problem Reduction**. The paper is organized as follows. Section 2 shows design and implementation details. Section 3 discusses the efficiency and performance. Section 4 presents the comparison with human approach and concludes the paper.

2 AGENT DESIGN

The agent uses the below vocabulary to formally representing a sub-goal.

- **ON(A,B)** => representing that block A is ON top of block B, here "B" can even be the table
- **Clear(A)** => representing that there are not block on top of a given block A
- **pre-conditions** => the constraints which need to be satisfied before the moves

needed for the goal can be legally made. For this problem, the only required pre-conditions are that both nodes should not have other nodes on top of each other before node A can be placed on top of node B. i.e. **Clear(A) and Clear(B)**

The agent uses the below classes.

Node - representing a single block in the initial arrangement. A collection of blocks with the right set of relationships between them represent a particular state/arrangement of this problem. Each node has

- **name** (the alphabet representing the block)
- what block is **above**, what block is **below** and whether the block is **clear** (no nodes on top)
- A list of **blocks on the same tower** (if there are multiple towers in the problem, this helps some of the optimizations) and
- A **seen-in-cur-goal** attribute which is used to track nodes explored in the current state.

Goal - A goal is the smallest possible chunk of the final arrangement. The final arrangement is broken into a set of sub-goals. Each goal represents a relationship between 2 nodes (smallest possible representation as a sub-goal) which needs to be true in the final-arrangement. If all the sub-goals are achieved, it indicates that the blocks have transitioned to the final arrangement. Each goal has

- **blocks** - A list of just 2 blocks where the first block needs to be on top of the second block in the final arrangement. for eg: if ON(A,B) is the sub-goal, this list has [A, B]
- **pre-conditions** - As explained in the vocabulary.
- **moves** - A list of moves that need to be made from current state to achieve this goal.
- **execute-after** - A list of goals which need to be achieved before this goal can be achieved and an is-completed flag

The current state of the problem is tracked using a global Nodes list which holds the current arrangement of nodes along with their relationships and a global Goals list which maintains the goals pending to be achieved to reach the final arrangement.

At each step, the agent comes up with the current Nodes list and the Goals list. For eg, Figure 1 in the appendix section shows how the agent represents the

nodes and goals in the first step of a problem. At each step, the agent incrementally generates the moves for achieving one of the remaining sub-goals. Picking the right sub-goal to satisfy in the current step is the most important activity at each step.

Below is the rough algorithm:

1. Upon entry, generate the Nodes list from the initial arrangement and Goals list from final arrangement.
2. Partial planning - for each sub-goal, generates the list of moves needed from current state to achieve that goal. Figure 2 in the appendix shows an example of the generated moves for the goals in Figure 1.
3. Detecting conflicts - for each sub-goal, compare its preconditions with the moves generated for each of the other goals to identify conflicts. If there is a conflict, then the goal with the conflicting moves should be executed only after satisfying the current goal. Append the current goal to the offending goal's **execute-after** list
4. Choose the next goal to execute - Once conflicts are detected, topologically sort the goals based on a graph constructed using the "execute-after" lists. Execute the moves for the first goal from the sorted result as it is the most ideal to execute without affecting the progress of the other goals.
5. Make changes to the Nodes list to represent the state of the nodes after the chosen goal has been achieved. Eliminate chosen goal from list of sub-goals as it is completed.
6. Repeat steps 2 to 5 until no more goals are pending.
7. Return the whole list of moves generated at each step in sequential order.

3 AGENT EFFICIENCY AND PERFORMANCE:

The agent performed exceedingly well on the example set provided in the starter code as well as all test-cases provided by the auto-grader and generated the most optimal solution for all test-cases. With regards to efficiency, there is definitely room for some minor improvements as getting the code to work was the first priority rather than coding up an efficient solution. The agent walks through the list of goals and nodes in nested loops in the conflict detection step and the topological sorting step. This can be optimized and there are no particularly clever moves to arrive at solutions quicker. That said, the agent still achieves short turn-around times even for problems involving many blocks and with multiple

towers. Table 1 shows the time taken in seconds and the number of moves made by the agent to each goal state for a combination of total number of blocks and total number of towers/stacks.

Table 1—Table depicting the time taken and number of moves to solve different combinations of blocks and tower sizes

Number of nodes	Number of towers	Time taken to solve in s	Number of moves
6	2	9e-05	3
6	2	0.00013	3
10	3	0.00036	12
15	5	0.00083	17
19	3	0.00162	23
20	4	0.00198	28
21	5	0.00233	29
24	3	0.0032	37

As one can see there is a slight linear increase in the time taken when number of nodes or stacks increases.

4 COMPARSION WITH HUMAN METHODS:

The agent's approach does not match the human method in any way as the agent uses **planning** which solves the sub-goals one by one in a top-down manner. On the other hand, humans may approach this problem intuitively by making moves which take them closer to the solution, possibly in a bottom-up manner. Humans generally do not think about sub-goals or about deducing the moves from the end goal. Humans can visualize how the state space might expand or shrink if a move is made without necessarily making the move. This is the differentiator. Machines cannot deduce whether a move might expand the state space exponentially without actually making the move. Also, it is difficult to determine when to stop exploring a particular branch of state transitions if it is not producing favourable moves. Back-tracking from an unfavourable state and reaching the last visited optimal state is costly in terms of time and memory. The approach taken by the agent presented here attempts to eliminate these consequences. It tracks the correct order of resolving the sub-goals to reach the goal-state. This might still not make the agent more efficient than humans.

5 APPENDIX

Figure 1 shows how node relationships and sub-goals are represented using the representation mentioned in section 2.

```
Initial arrangement
[['A', 'B', 'C'], ['D', 'E']]
Goal arrangement
[['A', 'C'], ['D', 'E', 'B']]
Goals are
A On Table:
  Pre conditions: Clear (A)
C On A:
  Pre conditions: Clear (C) and Clear(A)
D On Table:
  Pre conditions: Clear (D)
E On D:
  Pre conditions: Clear (E) and Clear(D)
B On E:
  Pre conditions: Clear (B) and Clear(E)

================
Nodes cur state
Nodes are

Table is clear 0 is on top of NA is below NA
A is clear 0 is on top of Table is below B
B is clear 0 is on top of A is below C
C is clear 1 is on top of B is below NA
D is clear 0 is on top of Table is below E
E is clear 1 is on top of D is below NA
```

Figure 1—Goals and node relationships for a given problem

Figure 2 shows the moves generated for each sub-goal generated for the final arrangement mentioned in figure 1

```
moves generated for sub-goal A on Table
No moves required
moves generated for sub-goal C on A
Move C -> Table
Move B -> Table
Move C -> A
moves generated for sub-goal D on Table
No moves required
moves generated for sub-goal E on D
No moves required
moves generated for sub-goal B on E
Move C -> Table
Move B -> E
```

Figure 2—Goals and node relationships for a given problem