Mini-Project 2 Journal

Juejing Han
jhan446@gatech.edu

1 AGENT IMPLEMENTATION

My agent solves the Block World problem by searching the state space with the **A^* (A-star) algorithm**. It finds the minimum-cost solution (optimal number of valid moves) by expanding the frontier state with the smallest

$$f(s) = g(s) + h(s)$$

Where $g(s)$ is the number of valid moves from the initial state to the current state s, and $h(s)$ is a heuristic function that estimates the cost from s to the goal. With an admissible heuristic, the algorithm returns a least-cost solution[1].

My agent implements the A^* algorithm with a priority queue to select the minimum-cost (smallest f) state to expand during each iteration. The heuristic function counts support mismatches: for each block, compare its immediate supporter (block below or Table) in the current state with its supporter in the goal state, and a difference is a mismatch. Each mismatched block must move at least once to reach the goal, leading to an admissible lower bound. This mirrors **Means-Ends analysis**: the difference between the current state and the goal state is the mismatch count, the end is eliminating the difference to reach the goal, and the means are applying legal moves to reduce it.

My agent uses **Generate & Test** to generate and test new states. The generator applies valid moves (rules 1-2) and integrates a **pruning strategy** (rule 3) as follows:

1. Only move the top block, either to the table or onto another top block.
2. Move a top block to the table only if it changes the state (i.e., the source stack's height >=2).
3. Only move the top block onto the top of a goal-prefix stack whose block sequence matches the target stack in the goal.

1 https://en.wikipedia.org/wiki/Admissible_heuristic.

The tester then tests the validity of the new states by removing duplicates generated from the current state (popped from the priority queue), and valid new states are added to the state space by pushing them to the priority queue with priority $f = g + h$. During each iteration, the tester also checks whether the current state matches the goal state. If the goal is achieved, my agent returns the optimal solution – a list of the minimum number of valid moves from the initial state to the goal. If my agent cannot reach its goal after exhausting the state space (which should not occur for well-formed problems), it returns an empty list as a fallback, indicating the problem is unsolvable.

Each priority queue entry stores f, g, current state, and path (the sequence of moves from the initial state to the current state). When the goal state is reached (popped from the priority queue), my agent returns the stored path (the optimal solution) immediately – no backtracking needed.

2 AGENT PERFORMANCE

My agent passes 20 out of 20 test cases during the performance test, yielding optimal solutions for all 20 solvable cases. It receives full credit and does not struggle with any particular cases.

As explained in Section 1, the A^* algorithm uses a priority queue to expand the state with the smallest $f = g + h$. The mismatch-count heuristic function acts as a lower bound on the remaining moves (admissible – never overestimating the true cost). It is guaranteed to return a least-cost solution from the initial to the goal state.

3 AGENT EFFICIENCY

My agent performs efficiently by passing 20 out of 20 test cases within the allowed runtime/memory and receives full credit.

The time complexity of A^* algorithm is $O(b^d)$, where b is the branching factor[2] and d is the depth of the optimal solution[3]. In the Block World problem, $b = O(s^2)$, where $s\ (\text{number of stacks}) \leq n\ (number\ of\ blocks)$. Therefore, the

2 https://en.wikipedia.org/wiki/Branching_factor.

3 https://en.wikipedia.org/wiki/A*_search_algorithm.

theoretical time complexity of my agent is roughly $O(n^{2d})$, which is exponential in the solution depth d (i.e., the number of moves).

Table 1 — Agent performance by problem size.

Number of Blocks	Optimal Solution Found	Number of Moves	Runtime (s)
4	*True*	3	0.0001
8	*True*	9	0.0013
16	*True*	21	0.1266
26	*True*	36	14.3619

Table 1 demonstrates that as the problem size (i.e., number of blocks) increases from 4 to 26, my agent always finds the optimal solution, while the runtime rises from 0.0001 to 14.3619 s. The runtime increases dramatically as the problem size increases, reflecting the exponential growth (in the number of moves) predicted by the theoretical bound. Within 26 blocks in total, my agent remains tractable as the problem size grows. This is because the pruning strategy reduces the state space and keeps the space searching manageable.

4 AGENT OPTIMIZATION

My agent does the following things to yield answers efficiently.

First, it utilizes **the A^* algorithm** with an admissible heuristic function, which guarantees an optimal solution. The A^* algorithm combines the strengths of Dijkstra's Algorithm and Greedy Best-First Search[4], enabling informed choices about which path to explore next.

Second, it defines a **mismatch-count heuristic**. Because each mismatched block must move at least once, the heuristic function never overestimates the remaining cost to the goal (admissible), guiding the A^* algorithm effectively.

Third, it implements **pruning strategies** to reduce the search space. It excludes generating new states by placing legal moves onto a mismatched (or non-goal-

4 https://www.datacamp.com/tutorial/a-star-algorithm.

prefix) stack. Moreover, each state is stored in a canonical form[5] by sorting stacks, so logically identical configurations share a unique representation to prevent duplicates.

In addition, my agent tracks **the smallest *g*** for each state and skips further exploration of the same state with a larger g when it pops from the priority queue. This setting improves efficiency without affecting optimality.

5 AGENT VS. HUMAN

My agent does not solve the problem the same way I would.

The A^* algorithm with an admissible heuristic function that my agent uses is not something I would execute mentally. I would instead visualize the stacks and move blocks by intuition.

My agent outperforms a human.

For simple problems, e.g., from the initial state [('A'), ('B')] to the goal state [('A', 'B')], there may be little noticeable difference between a human and my agent.

However, as the problem size grows, my agent shows consistent efficiency and accuracy (as demonstrated in Sections 2 and 3). In contrast, I would need considerably more time to find the optimal solution, and it is hard for me to debug my own process because my reasoning is based on instinct rather than a step-by-step snippet of code.

Therefore, my agent is more efficient and accurate at solving the Block World problem than a human, especially for larger problem sizes.

5 https://en.wikipedia.org/wiki/Canonical_form.