Implementing a Heuristic-Based Planning Algorithm to the “Blocks World” Problem

Layton Borst, Josh Charlton, Julian Cumps
St.Olaf College
borst2@stolaf.edu, charlt4@stolaf.edu, cumps1@stolaf.edu

Abstract

This paper presents an in-depth exploration of the "Blocks World" problem, aiming to develop an effective algorithm using a heuristic-based Greedy Best-First Search (GBFS) approach. Building upon fundamental principles of planning and search algorithms, our paper delves into the intricacies of automated reasoning and planning systems, demonstrating their relevance in the realm of artificial intelligence. Drawing inspiration from the Hash Distributed A* (HDA*) algorithm, we developed a heuristic strategy, resulting in a GBFS algorithm to manipulate blocks to mirror an inputted initial and goal state.

Introduction

Planning and search algorithms make up a large part of the computer science field and are also very important in the realm of Artificial Intelligence (AI). One way to incorporate these concepts into a simple problem is by randomly putting some blocks on a table. In this scenario, we are given five blocks, named A, B, C, D, and E. The blocks can either be sitting on the table itself, or stacked upon another block. Their location is input by a user, and this represents the initial state. The user can then re-arrange the block as desired, this can be called the goal state. To use our knowledge of planning and searches, we will come up with an algorithm that can run for any combination of initial and goal states, and print out each step (state) that it takes to reach the goal.

This "Blocks World" problem serves as a fundamental illustration of how automated reasoning and planning systems can work. Although it may seem like a simple and abstract problem, its applications and implications can be observed in various real-world scenarios and concepts, such as robotics, automated planning, operations management and software engineering. By studying and working with this problem, we hope to develop not only a working algorithm, but also to gain valuable insights into the foundational principles of artificial intelligence.

Thus, this paper will discuss our approach to solve the "Blocks World" problem. We will go into detail about our discussions, outlining the intricacies of how and why we arrived at our decisions, as well as the related work that we looked at and how it influenced our thinking process. Then we will explain how our approach to this certain problem, as well unpack our algorithm. Finally, we will discuss our results and how well our solution fits the scenario, by pointing out advantages and disadvantages.

Related Work

Before beginning work on our algorithm we explored other’s work in related fields. We found the paper Hash Distributed A* on an FPGA by Sakamoto, Ezaki, and Kondo, to be interesting and helpful. This work presents an FPGA (Field-Programmable Gate Array) based hardware accelerator designed to efficiently solve the shortest path problem using the A* algorithm. Given the recent growth in complexity and scale of graphs, the need to speed up solutions and reduce power consumption has become critical. The paper proposes a design that aims to address these challenges by leveraging HDA* (Hash Distributed A*) and FPGA technology. By comparing execution times and power consumption between CPU, GPU, and FPGA, the authors demonstrate that the FPGA implementation outperforms the others, while also achieving significantly reduced power consumption, and improved energy efficiency. This is particularly beneficial for battery-powered machines or devices such as self-driving cars and autonomous robots. Overall, this paper provides valuable insights into accelerating pathfinding algorithms in the context of large-

scale graph problems, with implications for various applications, including autonomous driving technology and motion planning for robots.

The research employs the A* search which is initially why it stood out to us. By looking at the paper's algorithmic details and pseudo-code, we could start deliberating how our own algorithm might look in the context of the "Blocks World" problem. Analyzing the performance evaluation section helped explain the algorithm's efficiency and even though we would not use an accelerated version of A* search, we considered the implications outlined in the discussion and conclusion to identify potential areas for improvement in your implementation.

Upon further discussion, we made the decision to eventually change our algorithm from A* because we noticed that we were only using the heuristic in our planning, and since we did not use a cost concept in the algorithm.

Approach

While planning a strategy to solve the "Blocks World" problem, we discussed a plethora of potential methods. Some of these included, making a the scenario into a binary tree and searching through, implementing a hill climb or simulated annealing algorithm, and using backtracking with our states. We explored each of these methods but found a downside to them. In the end, we decided that our approach would be to come up with a useful heuristic, allowing us to implement an A* search. This brought up more discussion because in order for A* to execute correctly or efficiently, we needed to come up with a way to enforce a sturdy heuristic. Our group discussed, number of block in correct location, number of block on table and other ideas, but in the end we would always find a scenario that it would not work for.

After some deliberation, we concluded that it may be helpful to think about setting some 'sub-goals' to lock certain stacks blocks in place if they were correct fro the table up. This bottom-up approach ended up working out for us, and we decided to base our heuristic off of this concept. This meant that we check the block's relationship all the way down a potential stack, and only give it value is the stack is correct all the way down, and it the block's location matches that of the goal state.

After further implementation of our A* search we ran into more issues surrounding copying and getting costs for the algorithm. This meant that instead of using our heuristic combined with the cost, we would modify our algorithm to only use the heuristic. Thus, our A* search became simpler and was now a Greedy Best-First Search (GBFS). Using this new method, we were able to get around some of the difficulties with obtaining the cost of moving blocks around, and base the next state sole off of the heuristic.

Function 1: Heuristic

```
Input: current_state, goal_state
Parameters:
- current_state: Current state of the blocks
- goal_state: Goal state of the blocks
Output: Integer representing the heuristic value
Let new_overlap = []
Let sum = 0
current_table, current_not_table =
current_state.get_blocks_on_table()
goal_table, goal_not_table =
goal_state.get_blocks_on_table()
For each block in current_table do
    If block is in goal_table then
        sum += 1
        Add block to overlap
sum += heuristic_helper(overlap, current_not_table,
goal_not_table)
Return sum
```

Function 2: Heuristic Helper

```
Input: table_overlap, current_not_table, goal_not_table
Parameters:
- table_overlap: List of blocks that are on the table in both
the current and goal states
- current_not_table: List of blocks not on the table in the
current state
- goal_not_table: List of blocks not on the table in the goal
state
Output: Integer representing the heuristic value
Let new_overlap = []
Let sum = 0
    For each block_current in current_not_table do
        For each block_goal in goal_not_table do
            If block_current equals block_goal then
                If block_current.on in table_overlap
                and block_goal.on equals
                block_current.on then
                    sum += 1
                    Add block_current to new_overlap
                    Remove block_current from
                    current_not_table
                    Remove block_current from
                    goal_not_table
        If current_not_table is not empty and
        new_overlap is not empty then
            Return sum + heuristic_helper(new_overlap,
            current_not_table, goal_not_table)
        Else
            Return sum
```

Algorithms

For this project, we implemented a series of functions to complete our algorithm. We have provided pseudo-code for the function that calculated the heuristic (and its helper function) that allows our GBFS to work. Along with this we implemented a method that generates all the possible successor states from the current state. However, we decided not to display these methods for involving GBFS because they are a standard within the planning algorithm.

Results

After implementing our algorithm, we were able to take initial and goals states as an input and then print out the steps needed to arrive at the goal state. However, we noticed that when we went through the series of steps that led to the goal, there were some unnecessary block moves. We noticed that with our test, it would pick up some blocks that were already in the correct location. Eventually the algorithm would put the block down in the correct place again, but these moves were unneeded. Nevertheless, the algorithm would obtain the goal state, with each of the blocks in the correct position.

After noticing this complication, we deviled further into streamlining the GBFS to find a more logical pathway towards the goal state. In doing so we discovered that we had made a mistake in our code. Once fixing this, our algorithm found the best path to get to the goal state in the test example. However, when the goal state was changed to something different we got stuck again. This time the issue was that our algorithm would repeatedly attempt to stack a block onto itself after picking it up. This of course is not possible, thus we would get an infinite loop leading to no goal state being found.

After more research and discussion, we concluded that our heuristic and GBFS functions were valid. After working through our other methods by hand to find the expected results, we found out that the issue was in our ‘generate_-successors” function which helped GBFS calculate all the next states to the current state. After changing the function to compare the current state to the goal state we were able to solve some of the issues.

Conclusion

This paper aimed to develop an algorithm to solve the "Blocks World" problem, demonstrating the application of planning and search algorithms in artificial intelligence. Exploring various methodologies including tree search, hill climbing, and backtracking, the study settled on implementing a heuristic-based Greedy Best-First Search (GBFS) approach. Drawing inspiration from the HDA* algorithm presented in the *Hash Distributed A* on an FPGA* paper, our research critically analyzed the implications of hardware acceleration in the context of solving pathfinding problems. Leveraging insights from the related work, our team adjusted the heuristic’s strategy, resulting in a simplified GBFS algorithm.

However, during the implementation, the algorithm exhibited inefficiencies, prompting a deeper investigation into optimizing the block movement sequence. This analysis culminated in the identification of the redundant block movements, leading to a reevaluation of the algorithm's decision-making process. Consequently, the study delved into refining the GBFS to achieve a more streamlined and logical pathway toward the goal state, enhancing the algorithm's overall efficiency and performance.

References

Hash Distributed A* on an FPGA

Sakamoto, R.; Ezaki, Y.; and Kondo, M. 2022. The Persona Effect: Affective Impact of Animated Pedagogical Agents. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems.New York: Association for Computing Machinery. doi.org/10.1145/258549. 258797.