Decomposed Search in a 3D Block World

Sean Russell, Jeffrey Van Etten, Wei Xi

Colorado State University Computer Science Department, Fort Collins, CO 80523

{slrussel, jvanette, xiwei26}@rams.colostate.edu

***Abstract—Path planning has always been a key study object in AI academia, especially robot path planning in a 2D world. There are many optimal methods like Best First Search and A\* to solve these types of problems. However, there isn’t much research on path planning in a discreet 3D world. To further the study of how solve these difficult problems, a second round of this type research is presented in this second project paper. The algorithm tested is a previously proven Decomposed Search algorithm working in a 3D planning environment. The Decomposed Search algorithm has been prepared for an unknown set of inputs that it will address and complete. Analysis of its performance is presented showing how the preparation of the algorithm, along with its heuristics for solving a 3D drone world path planning problem, demonstrate an ability to navigate and manipulate that environment.***

***Keywords—artificial intelligence; decomposed search; search; 3D simulation; three dimensional space path planning; drone;***

# 1. Introduction

Summarize what the reader will learn. What the paper will walk the reader through. The artificial intelligence community has long focused on several types of searches, usually on a flat plane, within a myriad of different or unique spaces in search of shortest distance, smoothest path, or unique answers to a problem. Recently, more and more drones have come into our lives requiring a path planning solution in an unknown 3D environment. Planning for the unknown is an interesting mental exercise and challenge, especially in software. As with any academic assignment, the unknown is a great mystery to be examined and studied. As a continuation of the first original cube world simulation project, the unknown in this project is the use of an outside source defining an initial state and end state made up of a set of cubes and drone in a simulated world. The original project was of our definition and, therefore, our programming and inputs were self-derived, customized, and implemented to make up our unique presentation. What makes this project difficult and a challenge, are the possible inputs of which we have only been given a hint.

The point of this project, therefore, was to use these externally supplied input files to create and manipulate cube objects, taking them from initial state to final state, by directing the drone using the Decomposed Search algorithm, thereby creating the final state as defined by the externally generated file. The time taken for the algorithm to process each pair of input files and to create the final solution were timed and paths measured to observe its performance. These results are presented in the results and results analysis sections.

The remainder of this paper is made up of the following sections. A description of the 3D simulation and its rules, preparations for the project, the project hypothesis, results, results analysis, and a conclusion of the algorithms performance.

# 2. The drone world Simulator

Like before, a 3D drone simulator was created with a height of 51 units and 101 units for both length and width. Using a text file as input, a list of blocks identified the blocks by color and initial position. This is considered the initial state. All blocks obey the rules of gravity, as they may be stacked on top of each other and none are allowed to float. A single drone is included in both input files and is used to manipulate the blocks within the space to create and match the final state as defined within a second text file; this file has the blocks, and drone, listed in their final positions. As said, this is a drone simulation. The allowed drone block manipulation actions are “attach”, “move” and “release.” A drone may not fly through a block nor carry a block through another block.

For testing, we used a world represented as a 3D grid eleven blocks wide by six blocks high by eleven blocks deep. Several input files were used to populate the world with the colored blocks and one drone. Complimentary input files were then used to provide the final definition of the world the algorithm was to make by manipulating the drone. The drone can be given instructions to move to any adjacent or diagonally adjacent cube in the grid as long as the cube is unoccupied. The drone can also pick up and move blocks around the world. Unless suspended from the drone, a cube cannot float in the air. It must either be on the ground or on top of another cube. All of the drone moves were tracked and timed.

##### 3. Algorithm description and Preparations

Why was this algorithm chosen? – show the results from the previous paper or just describe them? What do we think was good about the algorithm? A decision was made early to use the Decomposed Search algorithm due to its excellent performance in the first project. Although the required time to complete the tasks were a little longer, the performance of the Decomposed Search algorithm included the ability to successfully work through problems that a Greedy Best First Search algorithm was unable to complete. work th.

Using the supplied example test inputs as a test oracle, the following changes were made to the Decomposed Search algorithm. Any? blah beblah blah blah.

What are the operating computer parameters where the software is running? GHz, GB RAM, processor(s)

# 4. Hypothesis

With the preparations completed, our thoughts were that the simulation and search software were ready. Hence, our hypothesis: the Decomposed Search algorithm and its supporting programs prepared for this project had been adequately updated to take any set of cube world definitions and render the final required cube positions within a reasonable amount of time.

##### 5. Results

Present the data based on the professor’s inputs: timing, completeness, failures, ordered by input only.

# 6. Results analysis

Generalize the results then focus on: were the results what were expected? Where did it fail/succeed? Why did it fail or succeed? What items tripped up the algorithm? Where was it prepared and best solved the problems? The composite search is composed of two separate parts: a planner and a mover. The mover uses A\* as described above to navigate the drone around the simulation. The planner uses a special set of rules to come up with a plan, and uses the mover to execute the in between parts.

While planning, the algorithm does not consider the individual actions of the drone. Instead, during the planning phase, the composite search only considers moving blocks from one position in the simulator to another. It does this with a local greedy hill climbing search paired with a heuristic designed for the planning phase.

The planner heuristic considers the Euclidean distance between each blocks current position and its goal. However, we found that this was not enough on its own. Certain circumstances can arise that lead to the algorithms failure. For instance, consider the case where a block on the bottom of a tower needs to be moved, but all the blocks above it are in the correct position. If the only heuristic measure is the distance of each block from its goal position, then in the aforementioned problem the planner will be discouraged from disassembling the tower, even if this is the only way to reach the goal state.

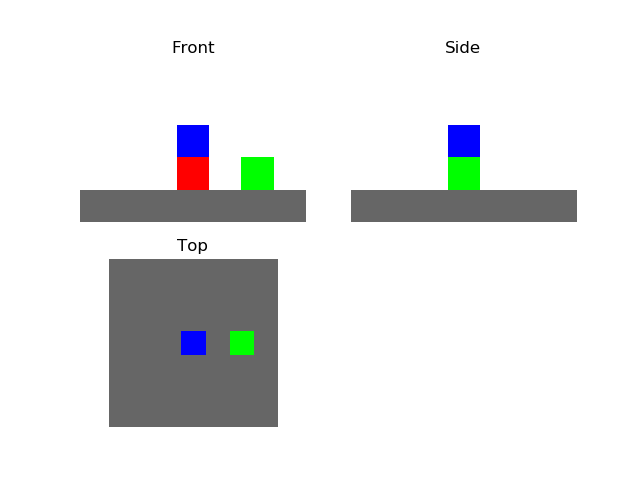
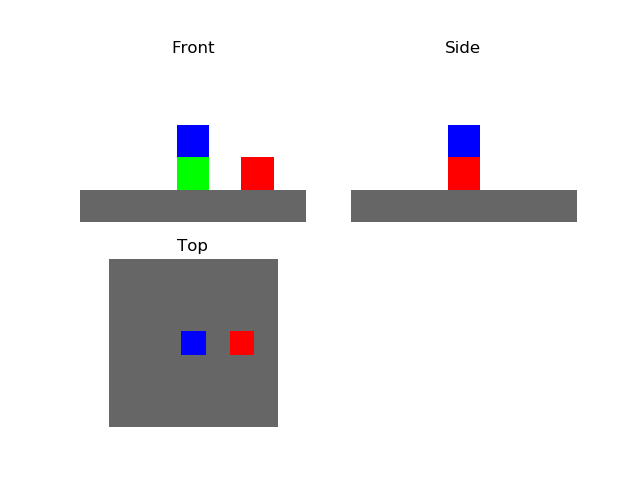


Fig 4.1: A potential issue for a planner. On the left is an example initial state and the right is a corresponding goal state. The blue block does not move from initial state to goal state, but the red block beneath it does. The planner somehow needs to figure out that the blue block must move off the red block in order to solve the problem.

To avoid such pitfalls, the planner’s heuristic penalizes blocks that are obstructing the movement of any block that is out of position. This idea was loosely inspired by the idea of agent empowerment as presented in [4], the theory being that it is preferable for the simulator to be in a state where all blocks that need to be moved are accessible.

plannerH = 0

for each block in state not at goal:

planner += euclidean distance from goal

planner += penalty \* number of blocks above

if block is in another block’s destination:

planner Heuristic += penalty

Fig 4.2: Pseudocode for calculating planner heuristic

In our implementation, the penalty term that appears in the planner heuristic was equal to the maximum possible distance between two blocks in the simulator. This was to be certain that the planner never got stuck with a block interrupting the movement of another, even if the plan requires moving the offending block across the entire simulator.

The planner utilizes this heuristic in order to come up with the next block that should be moved. It searches through the entire space of potential blocks that can be moved (any block that has nothing on top of it) and the places that each of those blocks can possibly be moved and finds the move that reduces the heuristic the most.

plan = [start]

while plan[last] != goal:

actions = GetPlannerActions(plan[last])

find action that leads to state with lowest planner heuristic score

plan.append(MoveDrone(bestAction.from))

plan.append(AttachDrone)

plan.append(MoveDrone(bestAction.to))

plan.append(DetachDrone)

Fig 4.3: Pseudocode for composite search

Once the planner finds the best block to move, it dispatches the mover to execute the details of its plan. First it moves the drone to the block’s current location, then attaches the drone to the block. Then the drone is moved to the destination specified by the planner and drops the block off.

Since this second part is really just a two part navigation problem and A\* has been shown to work very well on this sort of problem, we let A\* handle moving the drone around the search space. We then append the route found by A\* to our plan.

We repeat this entire planner-mover sequence until the goal is found, and the algorithm returns the sequence of steps taken in order to find the goal

##### 1665. Results

We ran greedy best first search and the composite search on seven different problems and recorded the time each search took to run, how many states each search visited, and the length of the plan that the search came up with.

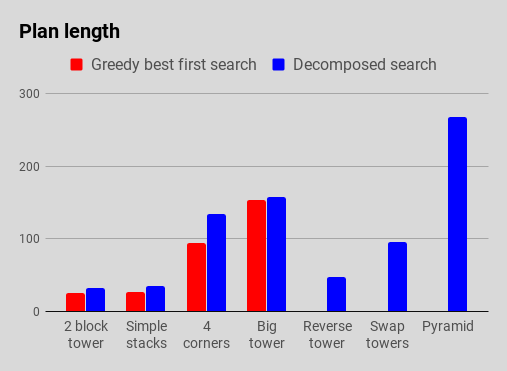
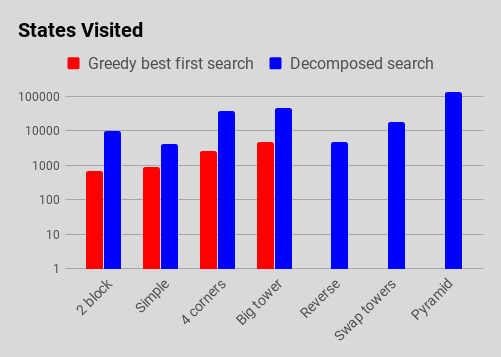
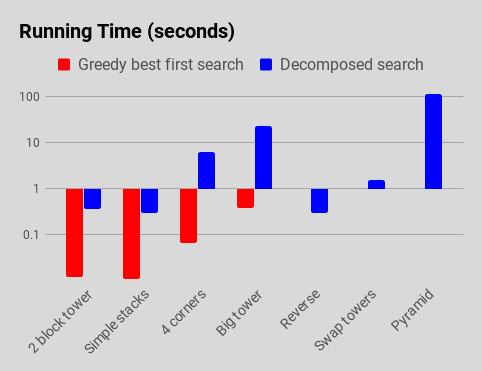
The data we collected is shown in fig 5.1. When 

Fig 5.1: Collected data. Both running time and states visited are presented on a logarithmic scale

reading these charts it is useful to remember that good search algorithm needs to both find a path and also minimize each of these measures.

There were a number of problems that GBFS was unable to solve. Any problem where a blocks movement was prevented by another block caused the GBFS search to fail after running out of memory.

However any problem that GBFS was able to solve, it solved in less time and found a shorter path than the composite algorithm. Our belief is that because the GBFS uses a simpler heuristic to evaluate each state, it can run to completion much faster; however this simpler heuristic also leaves it unable to cope with more difficult problems.

Another artifact of note from the collected data is the pyramid problem. The composite algorithm was able to solve all the other problems we gave it within thirty seconds; however the solution to the pyramid problem took nearly two minutes. We suspect this dramatic jump in run time is due to the composite algorithm’s planner component. The planner evaluates the result of moving each available block to every spot in the search space, so the number of actions evaluated by the planner at each step is equal to the number of movable blocks \* the width of the simulator \* the depth of the simulator.

##### 7. Conclusion

Does the data support the conclusion – how? The composite algorithm was extremely effective at solving problems that best first search was unable to solve. However, it did run slightly slower than greedy best first search and the plans the composite algorithm discovered involved more steps than those produced by GBFS.

We believe that there are a number of possible refinements that could be made to make the composite search run faster. Swapping out the basic hill climbing algorithm in the planner for a more sophisticated search may lead to a more robust algorithm.

We also believe that there is room to examine the theory of the planner heuristic to see if a more accurate estimation could be made, particularly with regards to the penalty term that is used when an block impedes another blocks movement.

##### 8. (No mention of this) Other work???

Considering the nature of this 3D project, future work has many interesting possibilities. Since one of our searches was a Greedy Best First Search, an interesting extension to this first project would be a test that is much like a simple game simulation. That is, to weight areas of the simulation and require the algorithm to determine best paths, either around or through, these zones to get to a goal position or build a cube tower. And/or if the drone approaches too closely, these zones could be populated with cubes that are a danger to the drone. The drone could then be penalized for coming too close to the cube in some way which could affect its ability to make it to the goal or create the final goal state.

Much like what was done in [1] another future project would be the inclusion of one more or multiple drones each having an assigned goal. Although the system in [1] was primarily two dimensional, the challenge could be considerable. That is, the system would then need to plan and coordinate multiple movements and, possibly, drone cube movements with prioritizations based on the respective assigned goals of each drone or the ultimate final state.

Although it would be quite an undertaking, another future project would be taking into account real world physical drones like done in [2]. Although the drone in [2] was a fixed wing craft and constantly moving, the our helicopter-like drone would be assigned physical parameters and the system would have to consider those aspects as the drone flies around the simulation – especially where a turn into an opening would require the consideration of the kinematics of the drone.

##### References

[1] Wenjie Wang, Wooi Boon Goh, “Spatio-Temporal A\* Algorithms for Offline Multiple Mobile Robot Path Planning (Abstract),” Proceedings of 10th International Conference on Automated Agents and Multiagent Systems, pp 1091-1092, 2011.

[2] Myung Hwangbo, James Kuffner, Takeo Kanade, “Efficient Two-phase 3D Motion Planning for Small Fixed-wing UAV’s,” IEEE International Conference on Robotics and Automation, 2007.

[3] Tatsuya Imai, Akihiro Kishimoto, “A Novel Technique for Avoiding Plateaus of Greedy Best-First Search in Satisficing Planning,” Proceedings, The Fourth International Symposium on Combinatorial Search, AAAI, 2011.

[4] Alexander S. Klyubin, Daniel Polani, Chrystopher L. Nehaniv, “Empowerment: A Universal Agent-Centric Measure of Control,” Adaptive Systems and Algorithms Research Groups, 2005