Composite Search in 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 Greedy Best First Search and A\* for these types of problems. However, there isn’t much research on path planning in a discreet 3D world. In this paper, therefore, we will present these algorithms and a brand new decomposed search algorithm working in a 3D planning environment. These strategies along with their heuristics for solving a 3D drone world path planning problem demonstrate an ability to navigate and manipulate that environment.***

***Keywords—artificial intelligence; best first search; greedy best first search; a\* search; search; three dimensional space; drone;***

# 1. Introduction

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. This paper will look into a unique search problem, that of a three-dimensional space with three dimensional searches looking for three-dimensional solution states. We built a 3D drone simulator with having a height of 51 units and 101 units for both length and width. Using a text file as input, the list of blocks identifies 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 to manipulate the blocks within the space to create and match the final state as defined within another text file. This file has the blocks listed in their final positions. The allowed block manipulation actions for the drone are “attach”, “move” and “release.” A drone may not fly through a block nor carry a block through another block.

The point of the project was to use these objects and manipulate them, taking them from initial state to final state, by directing the drone using different search algorithms, thereby creating the final state. The different searches were timed and paths measured to observe their performance.

# 2. Related work

Finding related works done in a three-dimensional simulation were unfound. A few works that were generally relevant were referenced while working on this project.

A path planning system for flying fixed wing drones [1] was a work where the, extra, kinematics of the fixed wing drone were essential and required as it flew through different environments. Quick decisions were required and the problem processing was optimized as the drone moved through the course. With a mapped-out course, much like the solving of our modifying the initial state to a final state, the drone made its way through the required points of the course in the required order. Of course, with our simulation the kinematics nor quick decisions were required. It was interesting, however, to read about their solving of problems and the paths and moves taken into consideration as the drone worked its way through the environment.

Path planning for multiple robots, in a two-dimensional world was referenced in [2]. There were several of the same type of obstacles, generally, but with additional robots roaming the field. An interesting aspect of their approach was prioritizing and re-prioritizing the multiple robots so that each could eventually complete its course. This may need inclusion in future work.

Reading about Greedy Best First Search and Best First Search Algorithms in [3], especially the identification of the importance of good heuristics as essential to the successful implementation of these types of searches was helpful. Their approach goes further in adding diversity into their search which wasn’t used in this project.

# 3. Construction in the drone world

For testing, we used a world represented as a 3d grid eleven blocks wide by six blocks high by eleven blocks deep. The world is populated by colored blocks and one 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.

Within this drone world we were particularly interested in the task of assembly. Figuring out the sequence of steps needed in order to complete a plan is very useful for robots of all purposes, as it enables them more autonomy to complete tasks on their own.

# 

Fig 3.1: Example of construction in block world. On top is a representation of an initial state, where blocks are placed in the simulator. The bottom is the goal (stack the blue block on the green block in the center), and the problem is to come up with a sequence of move for the drone to take in order to transition the simulator from the initial state to the goal state.

# 4. Approaches

## *4a. BFS*

Best first search usually only works on easy problems, like moving a cube 2 to 3 spaces but did well with stacking 3 cubes that were close together and parking the drone. Its branching factor can get up to 25, exponential complexity with length of path. However, when our drone world gets very complex, it had memory issues and took a long time.

## *4b. GBFS*

Greedy best first search is a search algorithm which explores and visits next state based on heuristic function value only. It actually did quite well for this project but can still suffer from memory issues when the simulation gets very large.

## *4c. A\**

A\* has been considered almost the best algorithm for path planning and it always gives you the optimal path. It did solves most of our cases but heuristics need to get really complicated to solve difficult problems. What surprised us is, it works really well for simply drone navigation.

# *4d. Composite search*

The composite search is composed of two seperate 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.

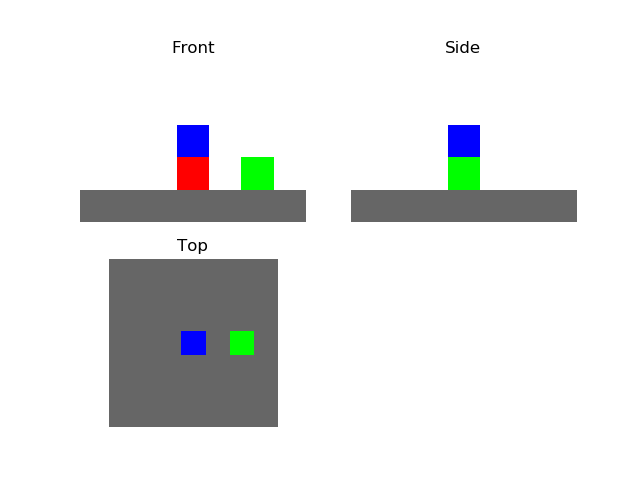
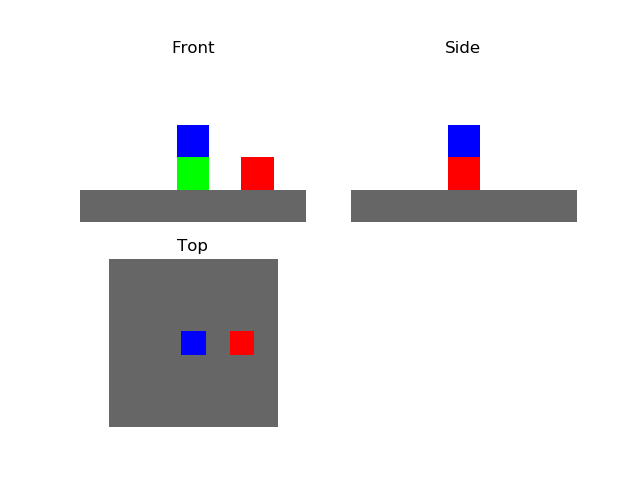


Fig 4.1: A potential issue for a planner. On the left is an example initial state and the right is a corresponding goal state. The blue block does not move from initial state to goal state, but the red block beneath it does. The planner somehow needs to figure out that the blue block must move off the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

##### 5. 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.

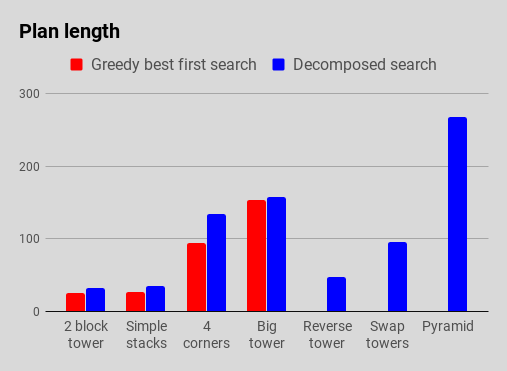
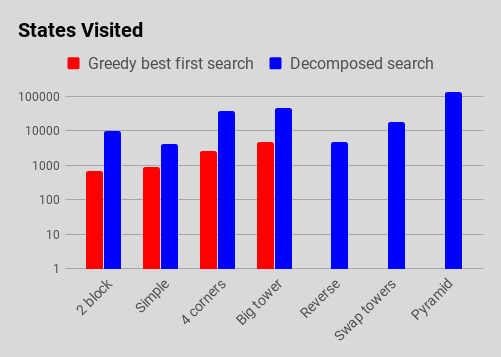
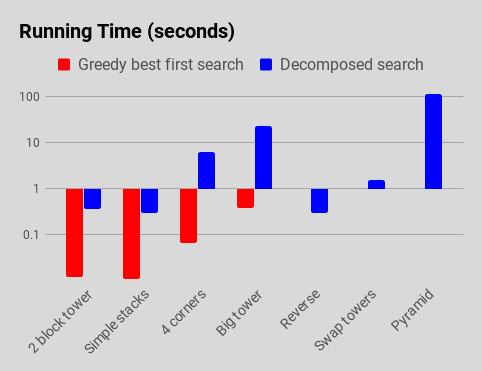
The data we collected is shown in fig 5.1. When 

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.

##### 6. Conclusion & Future work

The composite algorithm was extremely effective at solving problems that greedy 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.

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