Drone World Path Planning Using Evolutionary and A* Algorithms

Scott Axcell, Rejina Basnet, Aaron Blakeman, and Steve Kommrusch
Colorado State University
Department of Computer Science

***Abstract*—Building on our previous experience using various algorithms for planning block movements in a 3D world, we developed a two level planner that uses a Genetic algorithm (GA) to develop a high-level plan and an A* algorithm to find paths around obstacles. The primary goal of this paper was to develop an algorithm which could solve the widest possible range of Drone World problems. To accomplish this we developed a methodical test strategy that included testing for a multitude of scenarios as well as code coverage analysis. Ultimately, our algorithm performed well on all three novel cases provided after the code freeze and is capable of handling even more complex scenarios.**

I. Introduction

Building on our previous experience using various algorithms in Project 1, our primary motivation for this assignment was to develop the most versatile two-level planner algorithm possible. Our secondary motivation was to generate optimal paths when solving for a goal state. Our tertiary motivation was to accomplish the first two motivations in the shortest time possible. To accomplish our primary motivation, we pursued a methodical test development plan which included manually created and randomly generated tests.

A. Previous Work Motivates High Level Planning

In Project 1, we compared the performance of A*, Simulated Annealing, and Genetic algorithms against one another using three different Drone World starting states and goals. Project 1 allowed us to learn that the Drone World has a vast search space and because of this searching at the base level of the drone actions was inefficient. Thus we aimed to improve upon the best performing algorithms by merging them into a two level planner. We found the informed heuristic of the Genetic algorithm (GA) enabled it to be the most adaptable to various Drone World scenarios. It is because of these findings that we chose the GA to act as the high level planner in the two level planner for this paper.

In addition, the results of Project 1 showed that the A* algorithm performed very well with respect to wall time. The A* algorithm proved to be fastest of the three algorithms to find a suitable solution path when the start and end points for the drone are well defined. Thus, the A* algorithm appeared better at optimizing drone movement than planning high-level paths. This led us to use the A* algorithm as the low-level planner in our two-level planner algorithm.

B. Research

With the purpose of exploring alternatives to our Project 1 solution for our goal planner, we researched Goal Stack Planning and Partial Order Planning algorithms. Algorithms based on Goal Stack Planning are susceptible to the Sussman anomaly, where two interleaving goals can potentially force the algorithm into a deadlock state[3]. The draw back to Partial Order Planning is that it will increase the computational complexity even for a smaller search space[2]. The Drone World problem includes undetermined goal states, potentially multiple goal states, and a large search space. Therefore, we decided to stick with the GA from Project 1 and exploit its characteristics (randomness, mutation and fitness-function) further to map into our abstract level planner. Research shows that a GA with domain specific knowledge and a local search algorithm can find near optimal plans in a complex environment [4]. Inspired by this, our approach was to adjust the fitness and mutation functions to our problem space allowing us to solve the widest variety of problems possible.

C. Design decisions

After consideration, we decided that the unit for the top level planning should be block movement. The solution we chose is to create plans with action lists where a block action is of the form: (B_s, B_d, dx, dz) such that:

- B_s: ID of the block which should be moved
- B_d: reference destination block (or -1 for world location)
- dx,dz: offsets from B_d (or coordinate of world location)

Block IDs are in reference to the initial block list provided in the problem setup file. We chose not to track the height (Y) position in our plans as that adds detail not needed at the level of the planner. The Y coordinate is always presumed to be the top of whatever stack is at a given x,z coordinate in the world. We also support a final action of the form (-1,x,y,z) to specify the final drone movement. Here is an example action plan with description:

- (1,-1,7,5): Move block 1 to x=7, z=5 (y=top of stack)
- (2, 1,0,0): Move block 2 on top of block 1
- (3, 1,-1,0): Move block 3 to dx=-1 relative to block 1
- (-1, ANY,10,10): Move drone to (?,10,10)

Note that a plan implies the drone to move to and from locations in order to accomplish actions, but it does not specify the drone path required for the movement.

A second key decision we made regarding how the high-level planner should work relates to what the length of the final drone path would be. When evaluating plans, the high-level planner needs to have some concept of drone path distance. We didn't choose to simply compute the straightest possible path while ignoring obstacles. There are block world problems where it is better to get a block which has a higher coordinate distance from the drone to avoid the need to go around a barrier. So our planner checks whether a direct movement path would have an obstacle. It does not do a path search to avoid the obstacle, that search is left to the low-level path planner. Although checking for obstacles in the direct movement paths adds some overhead to the planner, it allows the planner to make better action decisions for problems with obstacles that need to be avoided.

D. Algorithm Overview

The algorithm works in two phases. The first phase consists of an abstract high-level planner based on the GA [1] and actions described previously. By having actions be related to blocks, the state of the Drone World can be in various positions when an action is invoked allowing for the reordering of the attach and release actions as needed. Phase two is responsible for taking as input the high-level plan and creates an optimized legal path consisting of attach, release, and move drone actions. This low-level planner solution is accomplished using A* algorithm when the plan has encountered obstacles. If the GA fitness function did not encounter obstacles then the same simple movement sequence used in the fitness function is used to create the drone path. Figure 1 shows the sequence used to create the path for the drone.

1) Genetic Algorithm: The GA begins with its initialization step. In this step supporting goals are created for goals at a higher Y level (i.e., a goal of (?,1,0,red) results in (?,0,0,?) being created to support the red block). Also, goals are sorted from lowest Y positions to highest. The population starts with 40 initial plans with 5 random block actions. Random block actions used at initialization as well as the MutateEnd function which extends a plan chose B_s randomly and a random destination which can be a random B_d with offsets or a specific X,Z position provided in the goals (i.e., if (?,1,0,red) is a goal, then (1,-1,?,0) is an action that moves block 1 to any X at Z=0.

Once the population is defined and goals organized, the GA proceeds as depicted in Figure 1. From a population, there is a 60% chance that 2 parents are chosen for crossovers before mutations may apply, and a 40% chance that a parent is randomly chosen to pass to the mutations. Crossovers pick random positions from the parents to cross over actions between the two, allowing subplans that are beneficial to potentially merge with other subplans. The mutations add, delete, or adjust actions in the plan in random ways to explore the options similar to the current paths.

The fitness function in our GA is where progress to the goal is evaluated and optimal paths are explored. The fitness function starts with the initial world state each time and executes the plan by moving the drone between the attach and release targets given in the actions. The world is updated so that after the plan is done the goals can be evaluated. The goal evaluation proceeds from the lowest Y value goals to the highest. For each goal met (at any level), the fitness is incremented for the plan. If a goal is not met, then blocks are searched to find which are closest to a goal and the drone to give a partial fitness benefit to each goal. This partial score is only given for goals at the level that does not yet have all goals met. For example, if there are 3 goals at Y=0 but only 2 are met, then a partial score is added for the last goal with Y=0, but no partial scores are given for blocks with Y=1 or higher goals. Another key behavior in the fitness function is decreasing the fitness for a plan when a goal is blocked (because a block is stacked on top of a needed block or a needed goal location).

In deviation from traditional GAs, the fitness function returns not just a fitness value but also a suggested action to improve the plan. This possible suggestions are: adding or removing a block from a given location (to build a tower or reach a lower block); replacing a randomly chosen action such that actions with longer drone paths are more likely to be replaced; and move an action to another position in the plan. When the fitness function returns, the suggestion is applied and its fitness evaluated, this cycle continues as long as the fitness is increasing. The last child to be evaluated before the fitness decreased is added to the population for the next generation. This behavior adds a random hill climbing feature into the usual GA and greatly improved our plan quality and time to completion of goals.

When moving the drone along a simple path for the plan, the number of actions that encountered obstacles are tracked. The GA tracks 3 'best' plans with increased fitness penalties for obstacles. bestLowPenalty only adds 0.5 drone path steps per obstacle; this is appropriate if the A* algorithm will be able to find a different drone move sequence that avoids the obstacle without needing to add any moves. bestMidPenalty adds 0.5 steps per obstacle and 5 steps if the obstacle it encountered is 10 Y units above where the drone tried to move; often an obstacle may be rather low and so the MidPenalty checks if it may be rather easy to go over. bestHighPenalty adds 10 drone steps per obstacle and 90 step if the obstacle is 10 Y units above the drone; this penalty is appropriate for large walls including walls reaching to the full Y=50 height of the world. These 3 fitness adjustments for obstacles can result in different plans, as discussed in section I-C. All 3 of these best plans are preserved in the population each generation as an elitist behavior.

2) A:* The A* algorithm being used is the traditional A*. A* searches for the possible routes and back traces using the shortest route. The heuristic and the cost for each move being used is the Euclidean distance between source and destination. This heuristics is fairly realistic and therefore is optimal for finding the shortest path. Each plan provided by the high-level planner contributes to two A* searches: the first search moves the drone to the attach location and the second search

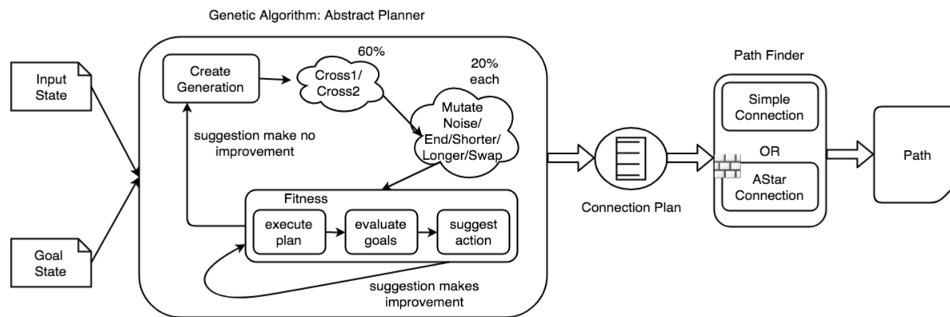

Fig. 1. Overview

move it to the release location. To reduce path length, the A* algorithm can drop a carried block from a height when the x and z coordinates match the target.

E. Testing Mechanism

As our goal is to have the most versatile application possible, we decided that the best way to achieve this was to submit our application to a wide variety of test cases and analyze the results. We ran our solution through a rigorous sets of tests divided into two different categories: random worlds and specific worlds. Random worlds included randomly generated single block columns, randomly generated multiple block columns, and randomly generated plateaus on the table surface. Specific worlds offered unique problems such as digging, separating colored blocks, building shapes, and finding ways through a single opening or down into enclosed areas.

The random testing proved valuable because it exposed the world to situations which we hadn't considered. For example, one of the randomly generated worlds created a tower of blocks in which the goal was to remove the top block t, $t-1$, $t-9$, and $t-11$. This is something we had a test for but in the randomly generated world $t-1$ and $t-9$ were the same color. Our algorithm needed to be adjusted to continue digging in the existing tower rather than moving the block it had already uncovered to a new location. The manually generated test cases allowed us to confirm that our algorithm behaved correctly for various world scenarios, but it also identified some cases in which our algorithm performed very poorly. For example, we developed a test case in which our algorithm had to gather blocks that were scattered throughout the table top and use them to build a flight of stairs. While our algorithm was able to complete this, it took over an hour for it to go through the optimization process. The algorithm was trying to re-use blocks close to the staircase rather than gathering other blocks on the board that could have been used to solve the goal. After the same optimizations used on the random world described above our algorithm was able to find an optimal solution in under 30 minutes.

F. Limitations and Risks

As noted, the primary goal of our approach was to create a system which could solve any legal goal problem set provided. Some limitations and risks were seen as we completed our test suite and considered our design choices.

In the planner, when a random block move is added during a mutation step, the destinations are initially limited to: on top another block, within 1 step of another block, or to a location seen in the goal list. This prevents a block from being randomly moved to arbitrary points in the world. The MutateNoise function can adjust the target relative to a block, but we did not include the ability to mutate the X,Z position targets in the plan (this is discussed more later relative to testcase 2).

Some of our last bugs/improvements done on Monday were in relation to poor initial block actions which tied blocks into interrelated towers (such that it was hard for random mutations to make any progress to undo a bad early decision). We fixed this with improved suggestion logic which worked well for all of our test cases, but there is a risk that some problem might be able to 'lock up' the algorithm into plans which cannot progress.

The planner runs in $\mathcal{O}(gnp)$ time per generation, where g is the number of goals, n is the number of blocks, and p is the population size. Large values of g,n, or p would slow down the generation rates. We explored the benefits of big and small populations in assignment 1, but for this assignment we simply used p=40.

We had 2 test cases that were huge - one had 12,000 blocks and another had 250,000. While both test cases were usually solved by the planner, the optimization was taking an unacceptable amount of time. We decided not to pursue debugging this once it was clear that there would not be a large block world test case.

With respect to the A* algorithm, given a feasible goal state, it will always find a path regardless of obstructions. In presence of obstructions the search space becomes larger. Additionally, when the drone is attached, the available space becomes further restricted resulting in needing more exploration. One other limitation is in the optimization we did. Since, the heuristic is the distance between start and goal which is one step up of the real goal, it might not always choose a path where the blocks can be dropped from some height. However, we think having this feature will aid whenever possible.

II. Performance

A. Results

The following results were obtained from the systems in CSB325. These machines have 6 core Intel Xeon CPUs (E5-1650 v4) running at 3.60GHz.

TABLE I
RESULTS FOR TEST CASES

		Test 1	Test 2	Test 3	Cube
Goal Runtime (s)	**Mean**	0.18	0.52	30.21	13.83
	Stdev	0.06	0.03	18.89	9.86
Plan Optimization Runtime (s)	**Mean**	6.61	10.69	236.62	84.70
	Stdev	2.22	3.22	103.7	29.09
Total Runtime (s)	**Mean**	6.61	10.69	365.8	84.93
	Stdev	2.22	3.22	701.16	29.10
Path Length (Steps)	**Min**	179	162	967	127
	Mean	179.86	166.80	1055.58	141.73
	Stdev	3.79	2.28	133.60	8.82

B. Problem analysis

Because none of the direct move paths from evolvePlan hit a barrier, A* is not called on the 3 problems we were given. All the drone movements use simpleConnect, which is optimal given no barriers. We chose to add a 3x3x3 cube problem as a case for study. Below a brief discussion of each problem is given including a calculation by us of the shortest possible path length for the test case.

test1: The shortest possible path for this problem is 179 drone steps (including 1 step for the initial position). Our drone movement in the planner is smart enough to drop blocks from a height during the early stage of this algorithm, and the suggestions from the fitness function do well building towers and digging for blocks so our code often finds the optimal solution quickly.

test2: The shortest possible path for this problem is 160 drone steps. Our frozen code is unlikely to find this due to move options for blocks. For our algorithm, we typically find a plan that results in 167 drone steps; the plan starts by moving black to location 0,0, brings red near the black, then moves black away and moves red to 0,0,0, then stacks black. An unlikely sequence of crossovers and mutations could in theory find the optimal path by moving the black block with action (0,1,49,49) and one of our runs got close to this with a path length of 162. Given this behavior, we changed MutateNoise to allow it to mutate the target x,z position (not just block offsets) and then the algorithm returned the optimal 160 for most runs.

test3: The shortest possible path for this problem is not trivial to compute. If the tower is set at x=z=0, then the shortest path is 976 drone steps. Our code is good at this kind of search, and the shortest path it found in 50 runs was 967. The first action for this result sets a block with offset to another one. Then all other blocks stack on top of it which makes this action list easy for MutateNoise to optimize.

Cube: This problem starts with a 3x3x3 cube of blocks with various colors and has a complex goal state. Like test3, the shortest possible path for this problem is hard to compute, but given the number and distance for block moves, the shortest path should be around 120. This case encounters obstacles during movement so A* gets called. The goals are: (2,0,0,blue), (1,0,0,blue), (?,?,3,magenta), (?,3,?,magenta), (3,?,?,magenta), (?,?,4,magenta), (?,4,?,magenta), (4,?,?,magenta), (?,6,?,?), (0,0,0,drone). The blue and drone goals require unstacking 3 blocks that are in the way; the 6 magenta goals can be solved by the 2 magenta blocks; and the goal at Y=6 tests that the ? color can be matched by any block.

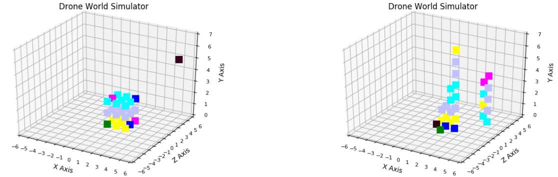

Fig. 2. The left figure shows a 3x3x3 cube which needs to be pulled apart so that a complex set of goals are met by the image on the right.

Wall: The world is complex with 29,701 blocks randomly placed at random locations building up towers of random height. Midst the tower are two walls blocking the straight path from drone to goals as shown in 3. The drone was initially placed at (0,50,0). It had to pick a blue block from (-50,0,-50) and place it at (50,0,50). The GA generated a single connection plan and A* found the path to complete the goal. Interestingly, it revealed that the shortest path was to find the path through the tower instead of taking the high ground.

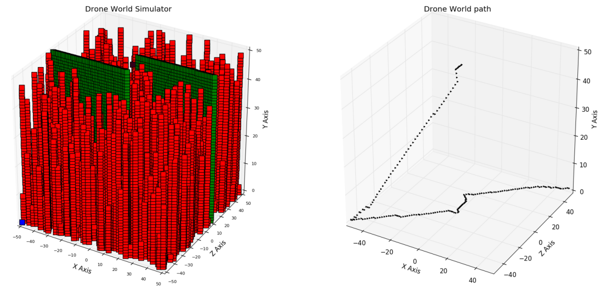

Fig. 3. The left figure shows the initial world and the right shows the path avoiding the green walls and red towers to reach the goal.

III. Conclusion

By using a methodical test strategy, our two level planner algorithm was able to solve all three test cases. More often than not a solution path only a few steps from the optimal, if not the shortest, path was generated. Our algorithm was able to generate these solutions in an acceptable time complexity. In particular, using GA as the high-level planner while choosing to abstract some low level drone actions and details enabled our two level planner to work efficiently and successfully meet our three motivations for this paper.

REFERENCES

[1] Maram Alajlan, Anis Koubaa, Imen Chaari, Hachemi Bennaceur, and Adel Ammar. Global path planning for mobile robots in large-scale grid environments using genetic algorithms. In *Individual and Collective Behaviors in Robotics (ICBR), 2013 International Conference on*, pages 1–8. IEEE, 2013.

[2] Mark Drummond and Ken Currie. Goal ordering in partially ordered plans. In *IJCAI*, pages 960–965. Citeseer, 1989.

[3] Naresh Gupta and Dana S Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3):223–254, 1992.

[4] Yanrong Hu and Simon X Yang. A knowledge based genetic algorithm for path planning of a mobile robot. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 5, pages 4350–4355. IEEE, 2004.