Information Scarce Maze Solving

Kane Bonnette and Stephen Hilber

Abstract— **Since its inception, the field of planning has been concerned with pathfinding. Many methods for pathfinding require a great deal of information about the space to be explored. However, there are other methods that require nothing more than a start and end location. An effective class of planners that require little starting information are known as bug algorithms. Bug algorithms have issues with continuous space in finding appropriate paths; these issues can often be solved by discretizing the search space. We present an implemention of this algorithm in a space and results showing our implementation's correctness and completeness.**

I. INTRODUCTION

There is a strong overlap between the robotics and the planning communities. One of the areas that these two fields conspire heavily on is the field of pathfinding, maze solving, or just movement in general. These two fields have produced many useful constructs, and we have implemented an application of some of the most general algorithms to solve the general problem of pathfinding.

The issue of pathfinding is an important one to many applications. Shorter paths allow for less time spent traversing a maze; time is critical in many situations. Shorter paths also allow for reduced movement costs. The benefits to complete and efficient pathfinding are quite bountiful.

Many algorithms exist for solving maze-like areas optimally. However, these algorithms require complete information about the search space (or at least the part of the search space containing the optimal path). Mapping the area before solving for a path may be expensive. The data may not even be available. These algorithms, such as A-star(cite), while provably complete and optimal, may not be effective in areas where no information exists. In fact, they may fail if presented with a lack of data in which to find a solution.

Luckily, there exist algorithms for solving this class of problem. Some work by using the sight, or other long range sensors of the robot. However, these algorithms will also work if the only sensors the robot has access to have a unit distance from the robot. Even collision sensors will be sufficient. We plan to use this class of algorithms to implement our planner.

There are many situations where robots must find paths without any information. For instance, leading a blind man out of a burning building hardly leaves time for getting building blueprints from a government office, and the man himself cannot give his guide input other than when he bumps into a wall. Leading animals to a pen by their leash, pushing a block out of a maze, or retrieving a lost object from a human-inaccessable area are all good candidates for a planner of this nature.

For this reason we have taken the step of working to have a robotic assistant guide or push an object (whether inanimate or otherwise) out of an obstacle strewn area, such as a maze or a fallen building. It is our hope that our implementation will be quick enough to work within time constraints, effective given limited information, and applicable to real world robotics.

II. RELATED WORK

There have been many algorithms developed for the task of path finding. Some of the first co-opted the Classical Planning structures, such as State Space, Plan Space, or GraphPlan, which can be solved given a formulation of the problem domain. However, this class of planners requires knowing the postconditions of the actions it performs, making them heavily suboptimal at forming plans for a space it knows nothing about. These planners also have the issue of difficulty in formulating the problem space. While well suited to logical problems, geometric problems are generally tedious and/or difficult to formulate for Classical Planning frameworks.

There are solution frameworks that provide optimal solution to a path finding, but these algorithms requires knowledge of the problem space. These algorithms include Voronoi graphs and visibility graphs. Both provide optimal plans (for differing definitions of optimal; visibility graphs will provide the shortest distance to the goal, while Voronoi graphs keep the planner as far away from obstacles as possible).

For algorithms that don't require knowledge of the problem space, the A*[1] search algorithm is one of the better known algorithms. Once supplied a heuristic appropriate to the domain of the problem, A* is proven to search over a graph in a best-first order. By using an admissible heuristic, A* able to achieve optimal performance on any graph it is applied to - no previous knowledge of the problam space is necessary. However, A* needs to be applied with an appropriate heuristic in order to achieve this optimality. For certain domains that measure success in numbers (such as distance to goal), this heuristic is trivial to obtain; for other domains, it may not work as well.

The class of algorithms known as bug algorithms resolves this problem by using methods to circumnavigate obstacles(cite). The naive algorithm is to move along the edge

CS 8803 RIP

K. Bonnette is a Master's Student of Computer Science, Georgia Institute of Technology, 800 Atlantic Ave, Atlanta, GA, 30332, USA kane@gatech.edu

S. Hilber is an Undergraduate Student of Computer Science, Georgia Institute of Technology, 800 Atlantic Ave, Atlanta, GA, 30332, USA stephen.hilber@gatech.edu

of an obstacle until it is no longer obstructing a direct path to the goal, and then returning to that direct path. Extensions include mapping the entire object and going to the point of the object closest to the goal before resuming a direct path to the goal state. [2]

Many pathing algorithms suffer from an exploding search space given continuous space. This problem is often solved by discretizing the search space into a grid, and then exploring that grid at the time [3]. There exist extensions to this as well, allowing for dynamically resizing the grid squares, subdividing grid squares, and others [4].

III. METHOD

For our approach, we created a solver that navigates around obstacles to a goal location for a given start location. The planner works in two dimensions. We discretize the workspace into a grid, and make the simplification that each grid square is only large enough to contain the object being moved through workspace. We assume squares for our grid to simplify the solution problem. If an object being moved through workspace is not square, we form a square of size length n around the object, where n is the smallest length that allows us to circumscribe a square around our object.

We make no assumptions about the size of the workspace; we assume no boundaries exist in the workspace unless we find a boundary while search for the goal. Therefore, we do not predetermine a size for out algorithm to explore. We presume the workspace to be initially empty, and therefore our algorithm attempts to reach the goal via the most direct path possible. As we disallow diagonal moves through the grid, we allow our object to only move one space through the grid at a time, thereby restricting our possible moves from any square to up, down, left, and right.

As the object is moved through the workspace, we remember the space that has been traveled through by creating a series of nodes. These nodes describe the locations above, below, and to both sides of the current location, as either null (unexplored area), open (capable of being entered), or closed (blocked by some obstacle, i.e. a wall). If we moved into an unexplored space and find that we are unable to do so, because out of a collision between the object and anything in the grid square being moved into, we mark the new square as closed and return the object to the original square. If in attempting to move, we encounter no obstacles, we mark the space we currently occupy as open.

Since the space has been discretized into a regular square grid, we use the manhattan distance between the object and the goal location as a heuristic to guide our search. In a effort to maintain consistency, we always move in the direction that has a greater distance (e.g., if the Y distance to the goal is greater than the X distance, we will move in the Y axis). If both the X and Y distance are the same, we bias our planner to move in the Y direction first. If an obstacle is encountered, we attempt to move around it. In moving around said obstacle, we first attempt to go in the cardinal direction of less magnitude. If that fails, or is unavailable (i.e., if there is an obstacle or if we are in line with the goal in one cardinal direction), we attempt to take the direction that increases the distance from the goal in the least cardinal direction. For example, if we are 5 units from the goal in the X direction and 7 units from the goal in the Y direction, we will first attempt to go towards the goal in the Y direction, then towards the goal in the X direction, then away from the goal in the X direction, and finally away from the goal in the Y direction. We always attempt to move into unexplored space, unless there are no unexplored spaces adjacent to our current location, in which case we move in the direction of the nearest unexplored space.

Since we do not backtrack unless there are no more paths to explore, we fully explore any unknown areas and thus will not miss paths that take us far away from the unobstructed path. If no free spaces are available, and the goal has not been found, we conclude that there is no path to the goal, and the maze is unsolvable. If the maze is solvable, we will eventually find a path to the goal. If the path to the goal requires infinite traversal, we will search for a path until there is no longer enough memory to store new nodes that have been explored. In this case, we will report the path to the goal as unsolvable.

Since we fully explore the space, any path to the goal must be found; our planner is therefore complete. As we bias our pathfinding to the closest grid space to the goal, we hopefully take the shortest path to the goal; our planner is therefore efficient. However, as the shortest path to the goal may require moving away from the location that minimizes our distance, (for instance, if we run into an alcove that cuts us off from the goal), the planner is not necessarily optimal.

IV. EXPERIMENTS

For our experimental runs, we utilized a series of five-by-five mazes. The mazes are presented below, with information and statistics related to each maze. For purposes of discussion, the grid starts at space 0,0 in the top left, and goes to 4,4 in the bottom right. This is the same numeration used in our experiments. Figure one shows the actions taken when the object moves from start location 4,2 to goal location 0,2. Figure two shows the actions taken when the object moves from start location 4,2 to goal location 0,2. Figure three shows the actions taken when the object moves from start location 2,4 to goal location 2,0. Figure four shows the actions taken when the object moves from start location 4,2 to goal location 0,2. Figure one shows the actions taken when the object moves from start location 4,0 to goal location 0,4. The last state (with object in goal location) is not show in any figure. In all maps, anything not shown is an obstacle.

V. ANALYSIS

In the empty maze shown in figure 1, the object explored 5 spaces, encountering 5 open spaces and 0 closed spaces. It took the only optimal path. This demonstrates the correctness of the algorithm in an empty workspace.

In the single obstacle maze shown in figure 2, the object explored 8 spaces, encountering 7 open spaces and 1 closed

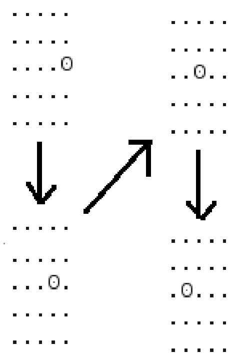

Fig. 1. An empty workspace

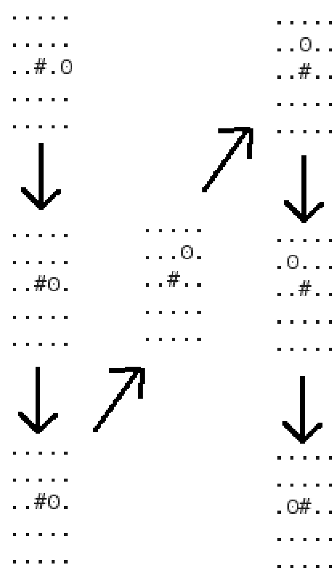

Fig. 2. A workspace with a single obstacle between the start and the goal

space. It took one of two optimal paths. This demonstrates the correctness of the algorithm in a sparse environment.

In the line obstacle maze shown in figure 3, the object explored 11 spaces, encountering 9 open spaces and 2 closed spaces. It took one of several optimal paths. This demonstrates the correctness of the algorithm in a denser environment.

In the alcove maze shown in figure 4, the object explored 21 spaces, encountering 14 open spaces and 7 closed spaces. It did not take an optimal path. This demonstrates the ability of the algorithm to backtrack when necessary.

In the squiggle maze shown in figure 5, the object explored 26 spaces, encountering 17 open spaces and 9 closed spaces. It took the only optimal path. This demonstrates the ability of the algorithm to handle feature rich environments.

VI. DISCUSSION

Bug algorithms are fully capable of solving this class of problems, and can do so quickly. All of our experimental

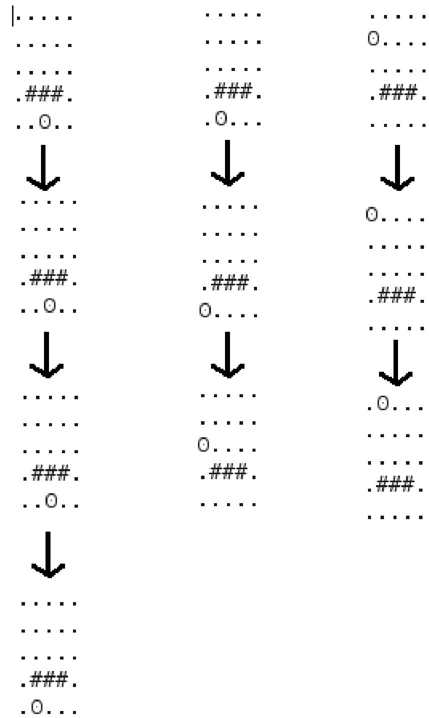

Fig. 3. A workspace with a line between the start and the goal

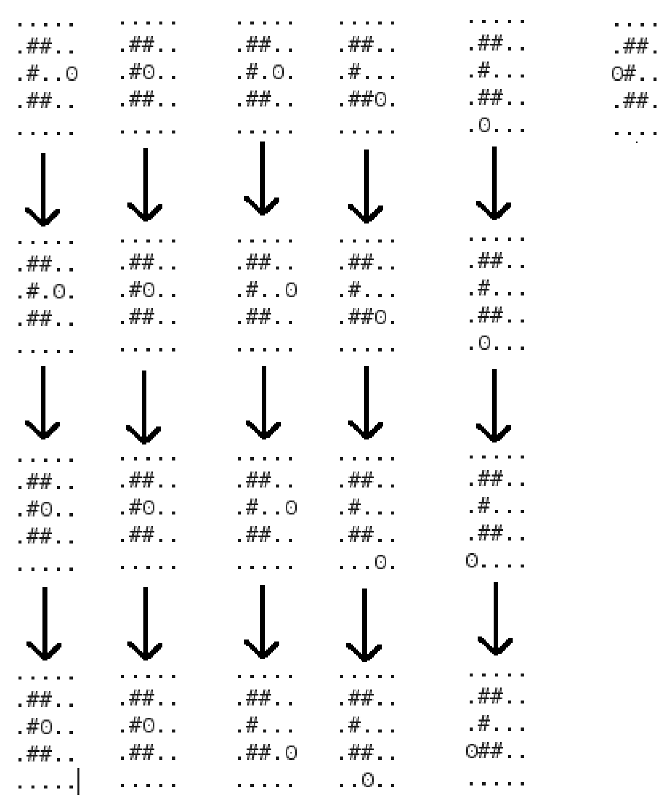

Fig. 4. A workspace with a local minima

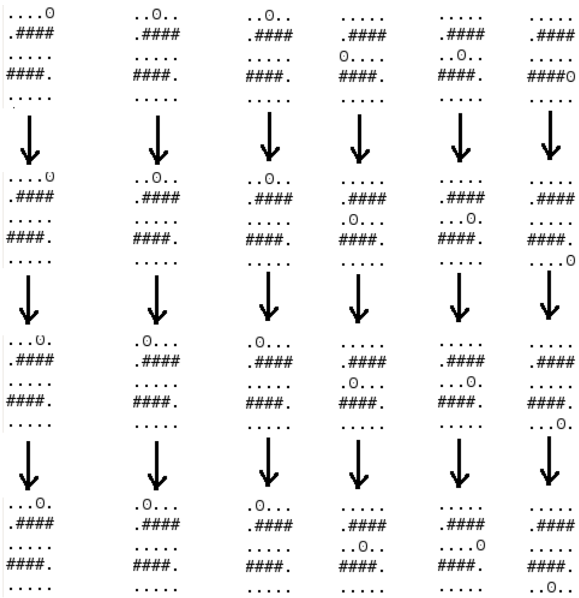

Fig. 5. A workspace with multiple lines requiring a zig zag to navigate

runs finished in under a second, and none required a human-noticeable amount of time to solve. We would like to conduct more experiments with larger test mazes to develop a greater understanding of the strengths of the algorithm, and provide a better comparison with other planning methods already in existence.

Results show that we explored a minimal amount of the space where possible, and took direct routes as much as possible. However, due to some errors in backtracking code, not all paths were optimal. We hope to correct these errors in future work.

On maps without obstacles, our planner takes the most direct route to the goal, thus showing its correct bias. If an obstacle is encountered, it takes the closest path around the obstacle, coming to the goal in an efficient manner. However, since it has no idea which side of an obstacle to traverse, it sometimes takes longer paths than necessary. Cases exist where further exploration of the workspace would have lead to shorter paths, but without prior knowledge of the workspace there is no way to conclusively establish the shortest path to the goal. Since our algorithm operates without any prior knowledge of the environment, we are unable to make decisions off of any information other than the feedback we receive from our sensors.

VII. Future Work

Our first goal of future work is to integrate the planning engine into a simulator that will allow us to run the planner in a simulated environment. While test harnesses are excellent for testing the logic of a planning function, the real world issues to make such a planner useful are difficult to simulate without a robot to test with. Simulators provide enough of a balance between real-world complexity and abstraction of side issues to accurately demonstrate the performance of the algorithm.

Once the planner can be shown to work in a simulated environment, the next natural step will be to integrate the planner with an actual robot. Robotic arms seem like the optimal platform for this integration. When dealing with robotic arms, forward and inverse kinematics must be computed, and it becomes much more important to tune features such as collision detection and step size. However, success integrating our algorithms with a robotic arm would demonstrate the effectiveness of our planning algorithms in physical situations and would be a springboard for further applications.

Another area for future work is to extend the planner to work in three dimensions instead of the more limiting two dimensions it currently operates in. Much more interesting results can be demonstrated with a three-dimensional feature space. Most problems in the real world also require more than two dimensions. As the implementation of our work in a robotic arm would allow us to operate in three-dimensional space, we can easily pursue both of these goals simultaneously.

However, extending the planning to handle three dimensional problem will also cause issues with the robot itself, not just the object it is moving. In trying to use a robotic arm to solve three dimensional problems, the links of the arm itself may have collisions with the workspace obstacles. Integrating these concerns into the planner will be difficult but is necessary if we want to produce a viable three-dimensional solution.

No prior knowledge was assumed in the methodology of the planner, but knowing something about the workspace can allow planners to form more efficient plans. Storing maps as the planner moves through a workspace for use on other problems in the workspace can speed solutions by precluding the chance of taking false paths if a route to the goal is already known. This would also allow the planner to handle finding multiple goal spaces without having to re-explore the workspace.

VIII. Acknowledgments

The authors gratefully acknowledge the instruction and direction of Dr. Mike Stilman, who made robotics not only interesting but fun. We would also like to extend our thanks to Venkat Mahalingam and Jon Scholz for their invaluable help in debugging in the early hours of the morning.

References

[1] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of a*," in *Journal of the ACM 32*, 1985, pp. 505–536.

[2] S. Rajko and S. M. LaValle, "A pursuit-evasion bug algorithm," in *Proceedings IEEE International Conference on Robotics and Automation*, 2001, pp. 1954–1960.

[3] J. Reif and H. Wang, "Non-uniform discretization approximations for kinodynamic motion planning," in *Algorithms for Robotic Motion and Manipulation*, J.-P. Laumond and M. H. Overmars, Eds. Wellesley, MA: A.K. Peters, 1997, pp. 97–112.

[4] U. Fayyad and K. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *JPS TRS 1992+*, 1993.