Autonomous Blind Robotic Mapping

Matthew Appler, Stephen Hughes
Mathematics & Computer Science Department
Coe College
5008, 1220 1st Ave NE, Cedar Rapids, IA 52402
MatthewAppler@gmail.com

Abstract

This work set out to explore the issues associated with mapping an unknown environment using singular and pairs of robotic explorers. The results were then compared to a baseline in which the search order was randomized every turn. This was done to show if having rigid search orders were better or worse than random movement. Since it is unreasonable to use brute force to test all possible combinations of explorer pairs, generic strategies (combinations of search orders) were developed. These strategies included: 0-7,1-6, 2-5, and 3-4 Reflected Explorers, 45, 90, 135, 180, 225, 270, and 315 Degree Clockwise Rotated Explorers, Non-Shared and Shared Random Explorers, and Rigid Shared Pair.

MOTIVATION & BACKGROUND

Clearly, it is possible for a single explorer to systematically visit all accessible spaces in a given environment. However, variability in an unknown environment's size and complexity, make it difficult to understand the optimal approach. The problem becomes more interesting when we allow the possibility of multiple agents conducting the exploration. This work set out to explore the issues associated with mapping an unknown environment using singular and teams of robotic explorers.

The results of this work can be applied to almost any mapping or navigational problem. An obvious application would be the exploration of unknown environments such as the sea floor, other planets, and environments we are unable to explore due to size, such as ant colonies. Furthermore, other applications could include robotic vacuums, search and rescue operations, and sea trash collectors.

PROJECT DESCRIPTION

Map Limits, Assumptions, and Generation

The first step was to program a tool that created maps for the robots to explore. For this experiment, a two-dimensional array was used to represent the environment/map. The map generating algorithm would be given a specific number of columns, rows, percent obstacles, starting location for the robots, and number of maps to generate. See Figure 1 for a visual representation of a generated map.

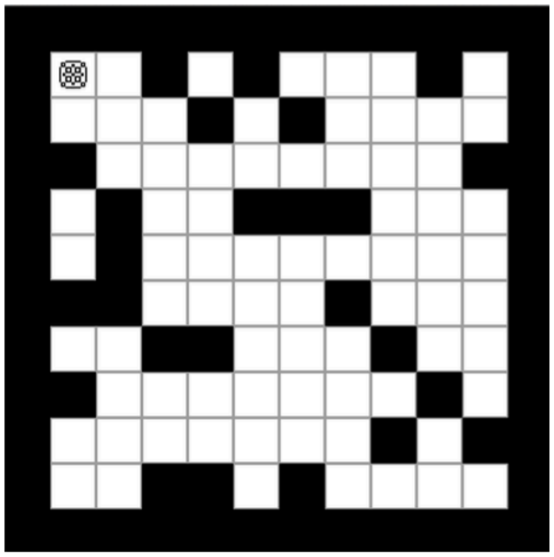

Figure 1: A generated map with 10 columns, 10 rows,
25% obstacles, and a starting position of column 0 and row 0.

To make a map, the algorithm creates an array free of obstacles of n+2 columns and rows. Then, a border of obstacles is placed to confine the robot. Next, (row*column*percent obstacles) obstacles are placed at random into the array with the exception of the starting location of the robots. This was done so the explorers do not start stuck inside an obstacle. The algorithm then checks with a recursive function if all open squares are connected either orthogonally or diagonally. This is done to make sure the percent difficulty is accurate. If an open square were to be impossible to reach, it could be effectively counted as an obstacle. Finally, steps were taken to eliminate duplication of maps. This process continues until the program made the requested number of mazes.

Robot Limits, Assumptions, and Generation

Much like the environment, the robot is given parameters before being deployed. These include a search order, starting row, starting column, and a map where all tiles except the starting tile are unknown. The search order is a string of eight unique digits ranging from zero to seven. Reading from left to right tells the robot which direction it should try to move first, second, … etc. See Figure 2 for an example of a search order. There are eight directions because the robots are allowed to move both orthogonally and diagonally. For the majority of the research the starting position was column 0 and row 0 due to some benefits which will be discussed later. It was noticed, however, that changing the starting location affects which search order performs better. For example, if the starting position was rotated 90 degrees, the best search order would also be rotated 90 degrees.

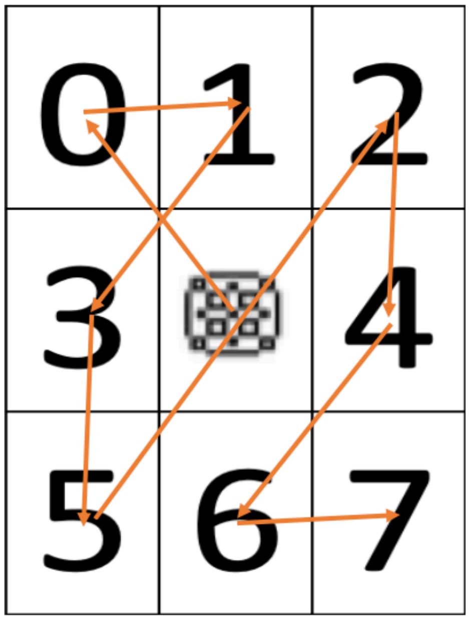

Figure 2: A visual representation of the direction assigned to each number between zero to eight. The arrows show order of direction checked for the order "01352467"

During an explorer's movement phase, it checks through its search order and tries to move in the indicted direction as long as the location has not already been visited. If the location it moves to is open, it marks down its previous spot as visited, adds it to the top of its backtracking stack, and updates its individual map. If the location is an obstacle it updates the map and continues to try using the search order until an open spot is found. If all surrounding spots are either visited or obstacles, an algorithm is run to try and optimize the backtracking stack by removing spots that do not need to be revisited. Then, the explorer pops the top spot off the backtracking stack and moves there. The main assumptions of this process are that all strategies/search orders revolve around the same backtracking technique and all strategies are context free (does not take into account previous moves when deciding which direction to move).

The efficiency of the explorer was evaluated based on the total open tiles divided by the total units moved to fully map an environment. This gives us the average fraction of unknown tiles discovered per unit moved. Moving orthogonally to an open square has a cost of one unit, moving diagonally to an open square has a cost of the square root of two units, but trying to move into an obstacle has a cost of zero. This was done because the robot could be assumed to be the same size as its current tile. Thus, the cost of moving into an obstacle would be approximately zero. The other assumptions made were that if the locations of the direction of "1" and "4" were obstacles but "2" was open then the explorer could move in the direction of "2" and that explorers could occupy the same spot.

Brute Force Backtracking – Single Bot

For a single explorer all 8! or 40,320 different possible search orders were tested. The results were then compared to a baseline in which the search order was randomized every turn. This was done to show if having rigid search orders were better or worse than random movement.

Since the 8! + 1 explorers could not be run over all possible maps, a random selection was made at the beginning of each run. Then, all explorers were run through the selection. This meant that there was some error in the performance of each search order. This error had an inverse relationship with the number of maps in the random selection compared to total possible number of maps at a given percent obstacles. This error range was able to be quantized for a single explorer when it was realized that if the explorers started at column 0 and row 0 for every search order there was a complementary search order that when all possible maps were tested, they had the same performance. See Figure 3 for how to find the complement.

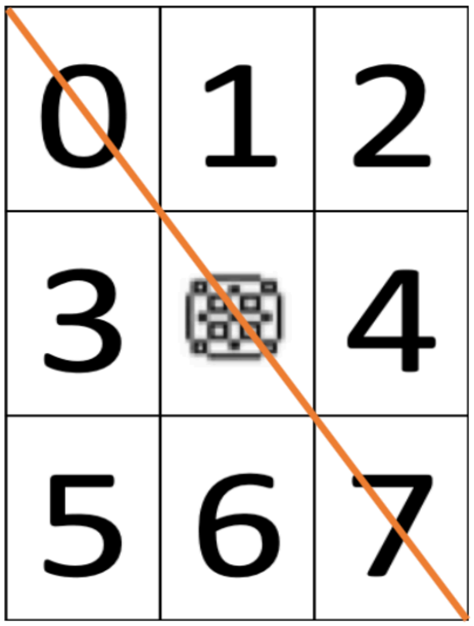

Figure 3: By reflecting the search order over a diagonal line from "0" to "7" the complimentary search order can be found for an explorer starting at column zero and row zero. For example, the complement of "01352467" is "03125647"

Using this information, a pruning algorithm was created to find the top performing search orders for a given percent difficulty and map size. To do this, the algorithm would first run all explorers through the selected maps. Then it would find the largest difference between all the search order complements and double the value. This represents the error range of that run. This value would then be added to the top performing explorer's performance creating the cutoff. Any pair of explorers in which neither one did better than the cut off would be removed from the running. This would decrease the number of explorers needed to be tested and thus allow the program to run a larger selection of maps further reducing the error. This process would continue until the top 25 explorer complements were found. This pruning algorithm was then repeated on 5% to 60% obstacles in intervals of 5% to see how the percent obstacles affected the top 25 explorer complements for 20x20 maps. See Table 1 for all future overall top search orders.

During the project, it was found that search orders starting with "310" and "130" preformed significantly better than strategies that started with "647" and "467" when the starting position was column 0 and row 0. This suggests that search orders that prioritize first hugging the starting corner and moving orthogonally before diagonally do better than strategies that initially search farther away. This makes sense because the cost of moving orthogonally is less than diagonally and if the explorer does not miss any tiles in its starting area it does not have to backtrack to reach them. This also meant that if the starting position was the bottom right of the map instead of the top left the roles would be reversed making search orders starting with "647" and "467" perform best. Finally, percent obstacles had a direct impact on the overall performance of the 8! explorers compared to the random baseline. At 5% obstacles, the majority of the robots performed better than the baseline. However, an increasing number of explorers performed worse than the baseline as the percent difficulty increased. It is believed that this is due to a reduction in large groups of open spaces and an increase of narrow passageways where there is only one path regardless of search order. Looking at Figure 4, it can be seen that at 60% difficulty even the best search order barely performed better than the random baseline.

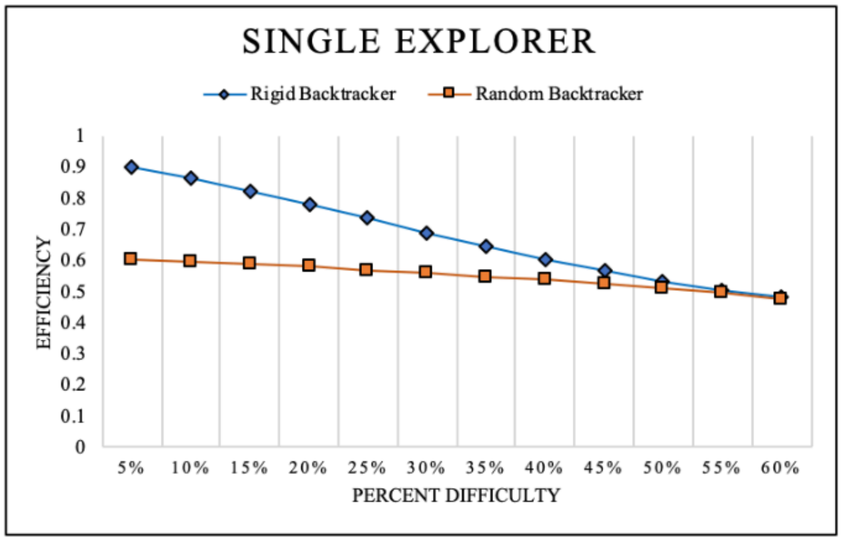

Figure 4: Comparison of best search order "31065427" vs. a random search pattern starting in the top left corner of a 20x20 maze.

Estimated Backtracking – Two Bots

Since it is unreasonable to try and use brute force to test all possible combinations of two explorers, generic strategies (combinations of search orders) were developed based on the results observed from single explorers. These strategies were then run on a random selection of 500 maps and compared over the same starting position and range of percent obstacles as the single bots were. Unless otherwise stated, there was no communication between robots when they were mapping. These strategies included:

0-7, 1-6, 2-5, and 3-4 Reflected Explorers

These four strategies involved taking every search order and reflecting it over a diagonal line between the two numbers listed to create a partner. This is the same process as in Figure 3. These strategies were chosen as a way to promote the robots exploring different parts of the environment. Looking at Figure 5, it can be seen that the 3-4 reflected performed best at 5 and 30 to 45% difficulties, 1-6 reflected preformed best at 50 to 60%, and the 0-7 reflected outperformed them from 10 to 25%. This data suggests that depending on the percent difficulty the best combination of search order changes.

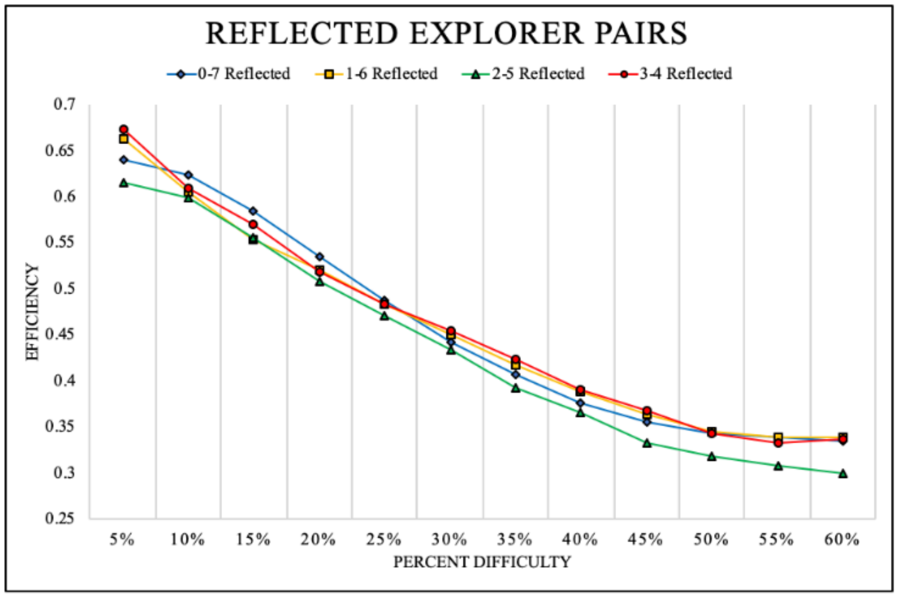

Figure 5: Comparison of best search orders for various reflected explorer pairs starting in the top left corner of a 20x20 maze.

45, 90, 135, 180, 225, 270, and 315 Degree Clockwise Rotated Explorers

These seven strategies involved taking every search order and rotating it clockwise to create a partner. For example, a 90-degree rotation for “01352467” is “24107635.” These strategies were chosen because as noted before rotating the starting position rotates the best search order. With that in mind, having rotations of a search order should allow it to effectively deal with a larger range of situations. From Figure 6, two conclusions can be seen. First, it was found that the 180-degree rotated explorers performed significantly better than the other rotated strategies. This is probably due to the 180-degree strategy making the explorers move as far away from each other as possible, thus reducing the amount of turns searching already visited tiles. Secondly, That the 45 and 315, 90 and 270, and 135 and 225-degree rotated strategies created pairs with very similar results. This is due to the similarity of their rotations with the ability of the same pairs being able to be generated. This is supported by Table 1, where both 45 and 315-degree rotated best search order pairs are the same.

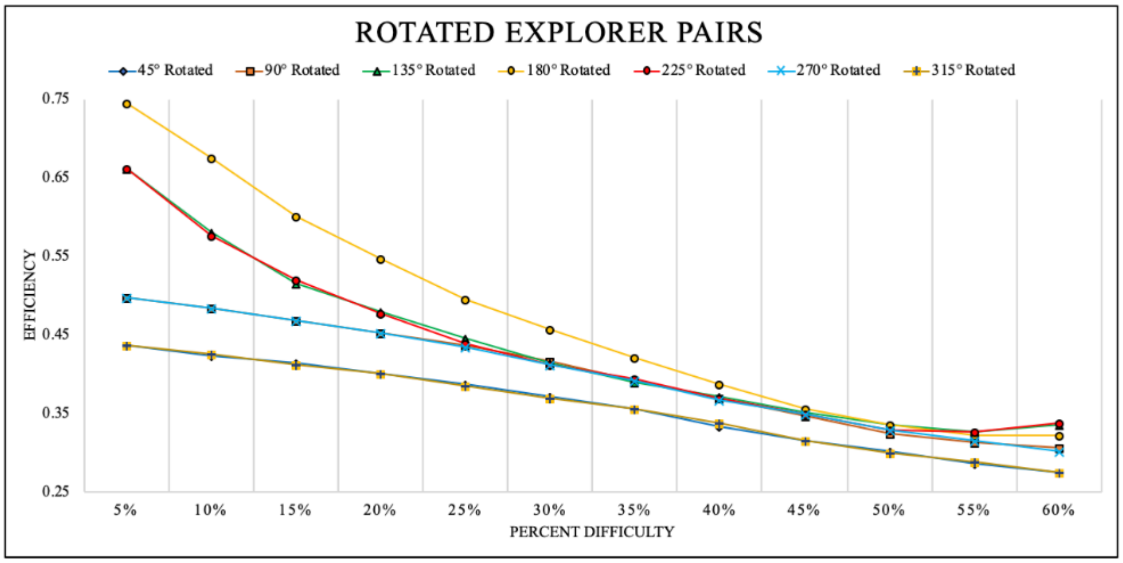

Figure 6: Comparison of best search orders for various rotated explorer pairs starting in the top left corner of a 20x20 maze.

Non-Shared and Shared Random Explorer Baselines

The non-shared random strategy involved having two explorers each with search orders that were randomly generated every movement phase. This strategy was used as a baseline for the other non-shared strategies. On the other hand, the shared random strategy differs by having explorers share where they have found open tiles and obstacles. This strategy was used as a baseline for shared strategies. Figure 7 shows that the random shared strategy performed better than the non-shared supporting that there is a benefit, at least in terms of units moved, of sharing information between explorers.

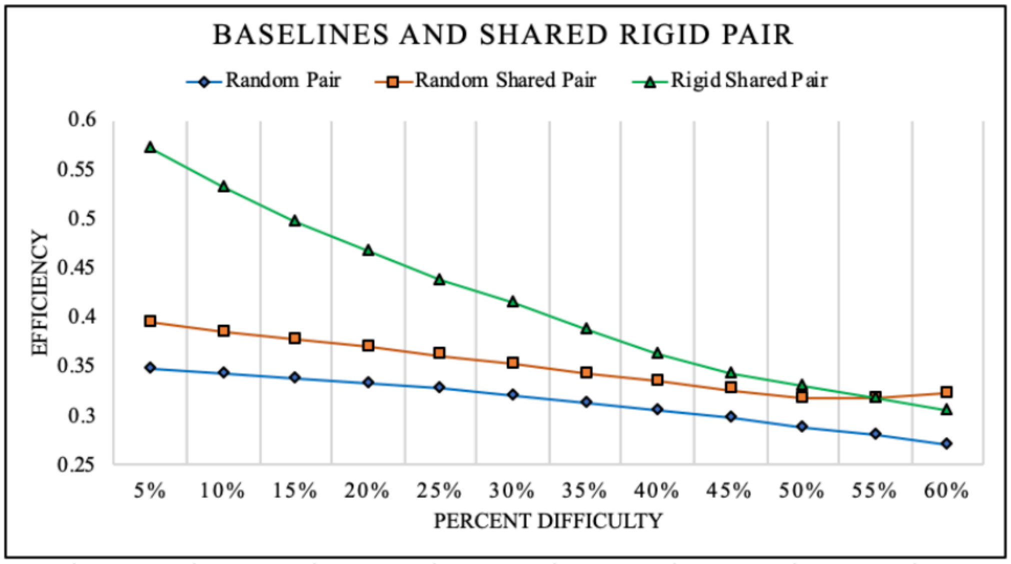

Figure 7: Comparison of best search orders for random and rigid shared explorer pairs and random pair starting in the top left corner of a 20x20 maze.

Rigid Shared Pair

This strategy makes a second explorer with the exact same search order as the original and has them share where they have found open tiles and obstacles. They were allowed to share because otherwise they would take the exact same paths and thus require exactly double the units moved compared to using only one. Looking at Figure 7, this strategy performed better than non-shared random but lost in performance when compared to shared random at higher percent obstacles.

Overall Comparison

Figure 8 shows the best strategy from each category compared to the random baselines. Much like in Figure 4, the strategies tended to converge as the percent difficulty increased, especially for rigid and random shared. However, this time both 3-4 reflected and 180-degree rotated maintained a visibly higher efficiency then their random pair baseline at 60% percent difficulty. Suggesting that as the number of explorers increases so does the importance of a strategy.

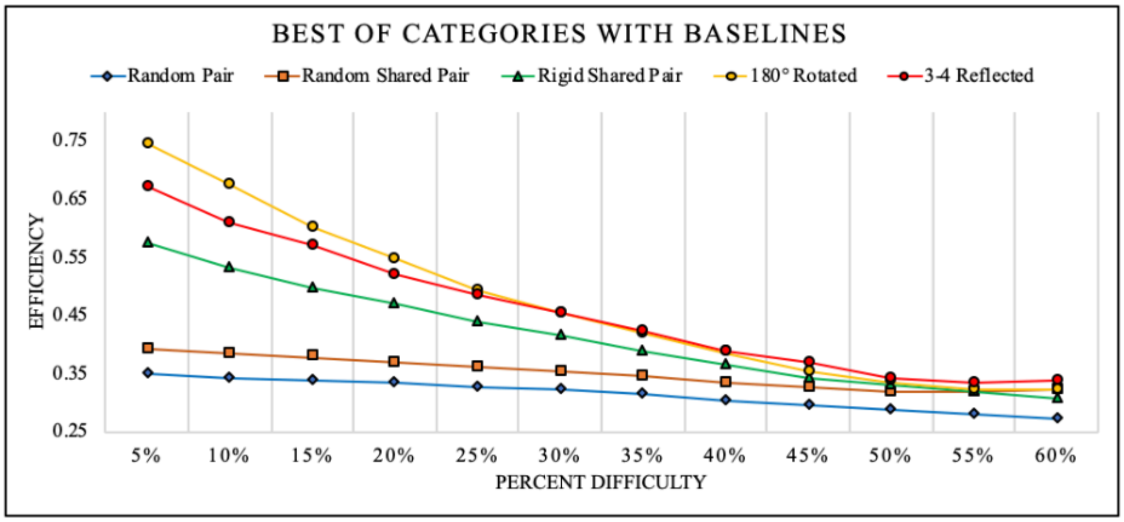

Figure 8: Comparison of best search strategies vs baselines starting in the top left corner of a 20x20 maze.

Strategy	Group Size	Best Search Orders	
Rigid Bactracker	1	31065427	
0-7 Reflected Explorer	2	21047365	53067142
1-6 Reflected Explorer	2	01365472	21467350
2-5 Reflected Explorer	2	76452310	03152647
3-4 Reflected Explorer	2	03142675	53647120
45° Clockwise Rotated	2	35016472	03125764
90° Clockwise Rotated	2	36510472	13042657
135° Clockwise Rotated	2	35061472	21407536
180° Clockwise Rotated	2	56473120	21304657
225° Clockwise Rotated	2	12043675	53607214
270° Clockwise Rotated	2	14230675	31065427
315° Clockwise Rotated	2	03125764	35016472
Rigid Shared Pair	2	13042675	

Table 1: Best search orders for various strategies starting in the top left corner of a 20x20 maze.

ONGOING AND FUTURE WORK

The research project is currently in the process of expanding the program to be able to run complex multi explorer strategies, create contextualized individual and double explorer strategies, further investigate the pros and cons of sharing between explorers, and run explorers on larger selections of maps to reduce error. The multi robot strategies would be based on the results and observations made from the single and double explorer tests. However, they would also have extended features which include staged deployment of explorers, a network hub which receives all information gathered to make decisions as to where/when to send and the search orders given to new/inactive explorers. As for the contextualized strategies, some examples being “always turning left when reaching an obstacle” and “keep going straight until hitting an obstacle”, a similar analysis would be conducted and then compared to past results. Finally, the map making algorithm is being optimized to be able to effectively make maps with percent difficulty greater than 60%.