Zekun Zhang, Ziqi Wang, Richa Rai

Assignment 2

Artificial Intelligence
October 15, 2019

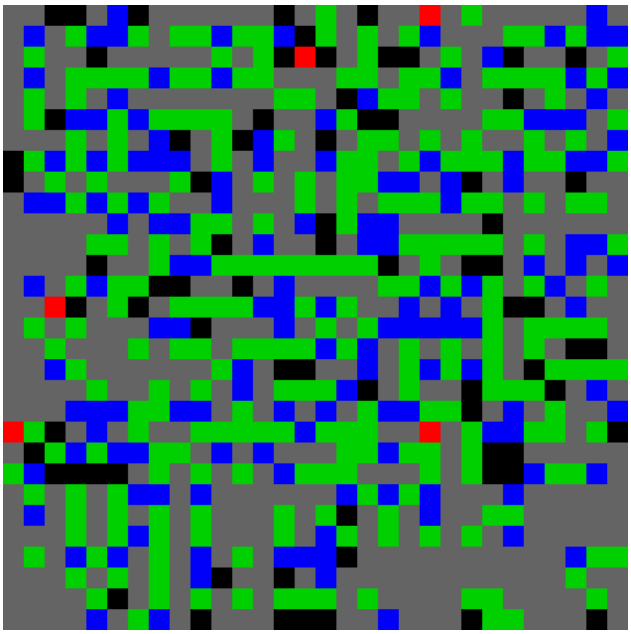

Figure 1: A sample traversed mine field! Dimenions: 20x20, mines = 240

1 Part 1

1.1 Representation

The mine field is represented as an dimxdim square board similar to the maze of the previous assignment. Actual mines are displayed in black color, flagged mine cells are displayed in green, flagged but unmined cells are displayed in blue and the traversal of the mine in gray if it its in empty space or red if we happen to step into a mine cell. A sample can be seen in figure 1.

1.2 Inference

Inference is done in the following way.

1. Initialize by assigning each cell to mine prob $P_m = |cells| * \frac{1}{|mines|}$
2. Pick the point with the lowest mine potential P_M .
3. Query the point. If Free get the mine clue. If mined, lose a life; continue.
4. Based on the clue distribute probability of mine in a a cell in the following manner:
   (a) if clue = 0, then all surrounding cells have $P_m = 0$

(b) if clue = 1, then all surrounding cells have $p_m = 1$

(c) when clue = 0 or 1,then redistribute or absorb the probability of that cell to all the other cells that where discovered on the same step and have not been visited.

(d) otherwise each cell's $P_m = P_M + \frac{|clue|}{d} - P_m\frac{|clue|}{d}$, where d is the amount of non discovered neighbors.

5. If a cell's $P_m > minedThreshold$ then it is considered mined, it is flagged and not queried until the the flag is removed.

6. Repeat steps 2-5 until all the cells have been visited or flagged.

The *minedThresholdparameter* can be arbitrarily chosen, it was fixed at 0.5 for the tries presented in this report.

1.3 Decision Making

AS presented above, decision is made by picking the cell where our belief that it is mined is the lowest. Since this is a probabilistic approach and the threshold for considering a cell has dangerous is picked as 0.5 (point of uncertainty), there is the risk exploding even in relatively safe cells. This can happen as if we don't query enough points around a candidate cell, we might have a skewed belief due to our initialization that the cell is safer that it actually is. A way to counter this would be to pick a much more conservative initialization scheme that assigns higher $P_{m_{init}}$ to cells, as in large field if can lead to overoptimism and abundant explosions!

1.4 Play Through and Performance

A playthrough is included in the gif folder for a field of dimensions 20x20 and 240 mines. The algorithm I implemented has some difficulty identifying large blops of bombs. It tends to start from edge point, which is a decision I support, as corner points tend to be the most informative. The score results can be seen in figure 2.

1.5 Efficiency

The complexity of the algorithm has a worst case of $O(n^3)$ as each cell could trigger a re-computation of almost the entire matrix. It has an amortized cost of $O(n^2)$ as , we limit the redistribution or absorption of potential mass to one hop neighbors. The recomputation of probability is the main culprit of the workload, and in larger fields it can add up. Although for mazes up to 30x30 it runs reasonable fast in under a second. The main improvement of the implementation would be vectorization of all the computations involved. Although care was to taken to minimize unnecessary computations, checks and for loops, there are some points that for loops could not be avoided, as the vectorized alternative is extremely complex.

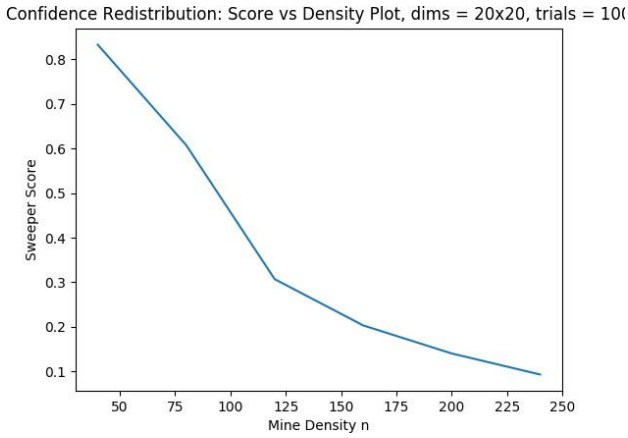

Figure 2: Performance of Implemented algorithm vs mine density.

The team members of this group are Zekun Zhang, Ziqi Wang, and Richa Rai. The three members discussed how to best implement the concepts and what methods may be more efficient. They then each worked on code and compared their different ideas. They then worked together to compile the project and discuss the analysis components.

2 Part 2

2.1 Representation:

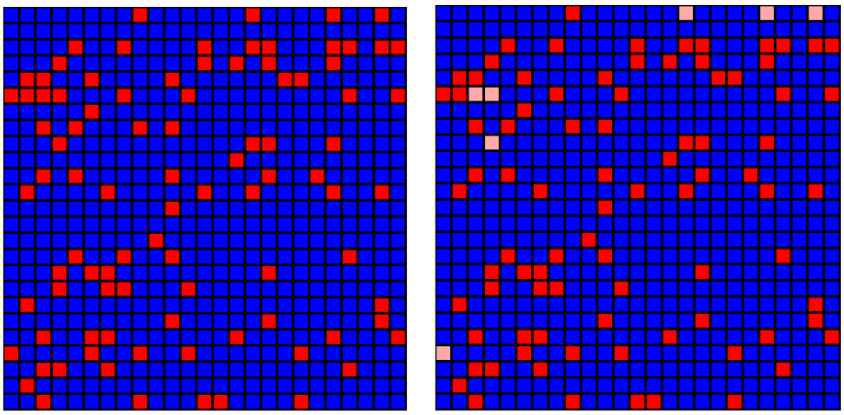

Figure. Left: mine map, Right: sweeper map (red: mine-marked, pink: mine-stepped, blue: others)

In our project, we are trying to use the data structure like the figure below. We use 3 2-dimensional arrays to represent a mine map, an agent map, and a probability map. In these three maps, the mine map is generated by the input which includes the dimension d and the number of mines n. Besides, the mines are separated randomly and are represented by 1s. The agent map which we actually manipulated is initialized with -1s, which means all of the cells are unrevealed. Then we are trying to reveal cells by the information from our mine map and in this process, we use 0 to 8 to represent the number of mines nearby the very cell we are manipulating now and -2s to represent the mines we marked by our knowledge and -3s to represent the mines we've stepped in. The last is the probability map whose probability is calculated by the information from our agent map. We traversal the whole map when we just changed one value of the corresponding cell and calculate the probability by adding weight 1 to the probability cell if the cells nearby have any tips that there would be a mine.

2.2 Workflow:

Our workflow can be divided into the following steps,

1. The same as the baseline, we first reveal the grid where is definitely safe to our agent. If we can't determine the next grid is safe or not, we simply go to another grid.

2. After that, when we can't reveal any grid of the mine map, we start to infer the mine. Firstly, we infer as the baseline, only when the unrevealed grid number + revealed mine number = the flag number then we mark the unrevealed grid as mine.

3. Before we can't reveal any mine, we continuously do the former steps.

4. if we can't reveal any mine and we can't extend the safe zone of our agent, we assign a "weight" to the unrevealed grid. the weight we assign to grids is related to the possibility of the grid of being a mine. And the weight of one grid can be calculated like this Wij = ∑ adjacent unrevealed mine number.

5. After the weight assignment part, we can get the grid which contains the lowest weight which is larger than 0. The reason why we don't choose the 0 weight is that if our knowledge base is small, there might be a lot of grid has a weight to be 0, and we can't determine which one is safe or not as the knowledge base can't give us enough information. So this is a trade-off between safety and information gathering.

6. After we get the information about the lowest weight, we consider the specific grid as a safe zone and make our agent walk on that grid. If the mine explodes, we mark it as a failure and continue the game, else we can update our knowledge base and do the 1. 2. step once again.

2.3 Inference:

We have a sweeper map as our current knowledgebase. And we have 3 ways of processing the given clue.

1. When we revealed a safe grid from the real mine map, we update it to our sweeper map, using the number on the same position in our mine map. If the number equals 0, we can go further and reveal all 8 adjacent grids.

2. Reveal mines. When the unrevealed mine = the unrevealed grid number of a specific grid, we can mark all unrevealed grids as mines.

3. Guess the safe zone. When we can't get enough information to reveal the mines, we go through the map and assign weights to the unknown grids. Then we assume the lowest weight (>0) grid is safe, and we can reveal the grid.

In the last process, we assume that the lowest weight (>0) grid is safe, but sometimes it contains mine and explodes, which means we can still improve the deduction of the current knowledge base when we can't infer the next mine. We think we can use conditional probability to maximize the deduction and reduce the possibility of making mistakes.

2.4 Decision:

There are several different situations when are faced with choices:

1. The current value is 0. It means we can go through all 8 grids safely.

2. The current value is x, the unrevealed grid number is y, and the revealed mine number is z. If x = y + z. we can mark all unrevealed grids as mine.

3. The current number is x, and the information is not enough for the agent to determine which grid is safe. Firstly, we go to positions where are definitely safe. If there is no safe grid we can reveal, we do a infer process to reveal a grid that is less likely to be a mine.

We have to take a risk to step on a new grid when we don't have enough information to reveal a new safe grid. So we use weight to reduce the risk as much as we can to choose a grid that is more likely to be safe.

2.5 Performance:

In some situations, the agent will step on a mine and kill himself. That is because when the knowledge base is not sufficient enough, we will randomly choose a less dangerous grid according to the weight and step on that to get extra information. However, this process doesn't guarantee the step on grid is safe, so sometimes the agent will kill itself.

2.6 Performance against baseline:

p = 0.15, p = 0.2, p = 0.25

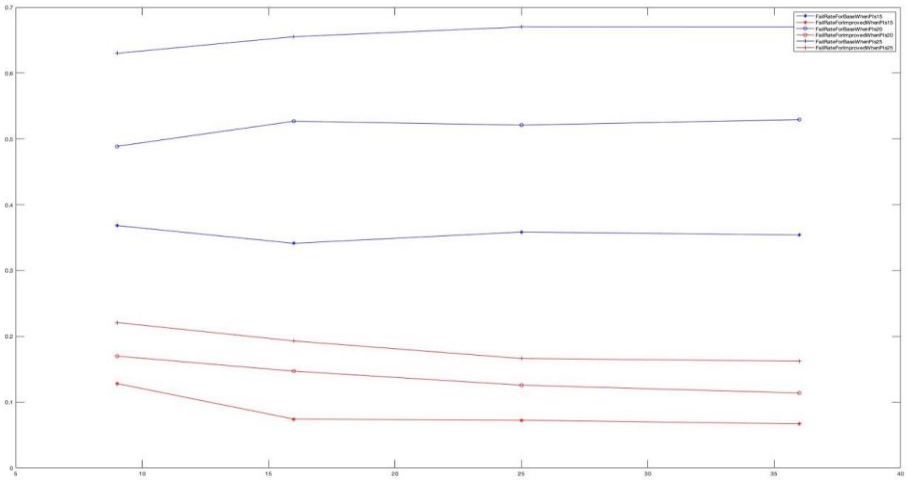

figure The evaluation graph

We test both algorithms 100 times at each point and take the average success rate and the overall consuming time as the overall result as the time-consuming of every unit run is very tiny.

In the success rate part, we test mine map in size 9x9, 16x16, 25x25, 36x36. When the possibility of mine becoming higher and higher, the failure rate is becoming higher and higher, when the value of p reaches to 0.25, we can find that the baseline algorithm is more likely to fail (0.65). Thus, we can say that when the possibility reaches to 0.25, the mine map is hard. Moreover, the failure rate of the improved algorithm is reduced by approximately 30% in each situation.

In the time-consuming part, we can get it from graphs that when the map becoming larger and larger, the time consuming of both algorithms is increasing at a similar speed.

Thus, we can draw the conclusion that our improved algorithm is better than the original baseline algorithm most of the time, from small size map to big size map, from easy map to hard map.
URL link:https://github.com/zzkzzk1996/CS520/blob/master/Project2/FailRate.jpg

2.7 Efficiency:

In our project, we are trying to use an algorithm whose time complexity is near O(N^3) and space complexity is O(N^2). We are just trying to traversal the whole map when we reveal the agent map to keep every step in our program to be the correct decision. Because we don't use any fancy search algorithm in our project, the time cost would be a little bit higher than other search algorithms.

2.8 Improvements and Bonus

Using this method, we once we found all the mines, we can quit the mine map immediately, which means we

don’t need to reveal the safe zone. However, as we already use this assumption in our baseline and improved algorithm before, we think we can improve our algorithm in another way.

Until now, we are using two different maps: sweeper map, mine map to represent our knowledge base and the original map. But this is not space-efficient, we think we can use one data structure to solve this problem. We can use nodes in two-dimensional map contains different kinds of status to squeeze two maps into one map. The node in map can both represent the current original status and how the agent recognizes it using two different attributes.