The program can be executed by executing Playground.py

Results for iterations greater than equal to 25

The following results were obtained when we varied the parameters:-

Dimension	Accuracies						Epochs
	Mine Ratio (40%)			Mine Ratio (30%)			
	Basic Agent	ITAgent	ITAgent (that knows the mine count)	Basic Agent	ITAgent	ITAgent (that knows the mine count)	
100	26.92	86.61	86.8	26.17	89.67	90.07	50
50	25.09	80.81	80.87	24.94	90.17	90.2	50
25	24.1	79.2	79.3	24.65	88.62	88.7	50

Questions and Write-up

Q) Representation: How did you represent the board in your program, and how did you represent the information/knowledge that clue cells reveal?

- ➔ The board is represented as a 2 dimensional Python Numpy array. The values of this matrix carry various meanings: -100 means that it contains a mine. If the cell is devoid of a mine, it is denoted by 0. A separate 2 dimensional Python Numpy array is used to keep track of the visited cells, where -1 denotes that the cell has not yet been visited whereas, any whole number from the range [0, 8] represents the number of mines surrounding the cell and that the cell has been visited. Safely marked mines are represented by -9 and opened mines are represented by a 9. The value of clue is updated in the visited matrix, which is further used to infer how many and which cells need to be marked or revealed.

  Additionally, a probability number matrix is used to store the estimate of the probabilities of each cell, based on which the next cell is chosen. Initially, each cell is given a value of 0.99, which is updated as its neighbors are revealed. The value of probability number for a cell is updated as

  $P' = CV + VNC / N$

  CV = current value, VNC = value of neighboring revealed cell
  N = number of neighbors of the revealed cell

Only for the first update, the current probability number value is changed from 0.99 to 0, while the update is being made. The lower the value of the probability number, the more likely the algorithm reveals it next. After the cell is revealed, the probability number value is changed to a large number.

Q) Inference: When you collect a new clue, how do you model / process / compute the information you gain from it? i.e., how do you update your current state of knowledge based on that clue? Does your program deduce everything it can from a given clue before continuing? If so, how can you be sure of this, and if not, how could you consider improving it?

➔ Whenever a new clue is collected, it is stored in the visited array. Using this clue, we make predictions about the neighboring cells. Two of the most important predictions/updates that can be made are:
1) If the number of safe neighbors is equal to number of revealed safe neighbors, then we can mark off the remaining neighboring cells as mines.
2) If the total number of mines minus the number of revealed mines is equal to the number of hidden neighbors, then safely mark all those neighbors as mines.

Additionally, it assigns probability number values to the neighboring cells which determine the likelihood of those cells being revealed by the algorithm. On every revelation of a cell in the matrix this value gets updated for all its neighboring cells.

Yes, the program deduces everything that it can from a clue, before it moves on to the other cell. It completely examines the cell in a way that it does not need to be visited again to perform the same process. We are sure of this because the process of revealing cells is such that whenever a cell is revealed and a certain probability number is given to its neighbors, which is less than 0.99, it stays in the queue to be revealed next and one of the two conditions above being applied.

Although the algorithm deduces everything from a cell at the current state of the matrix, there are cases when a backtrack to the cell is made when clues from other parts affect the probability metric of the neighbors of this cell. For example: if a clue 1 is found at corner, and at a later stage two of its neighbors end up getting revealed, then a backtrack to cell containing 1 is made to mark the last location as a mine.

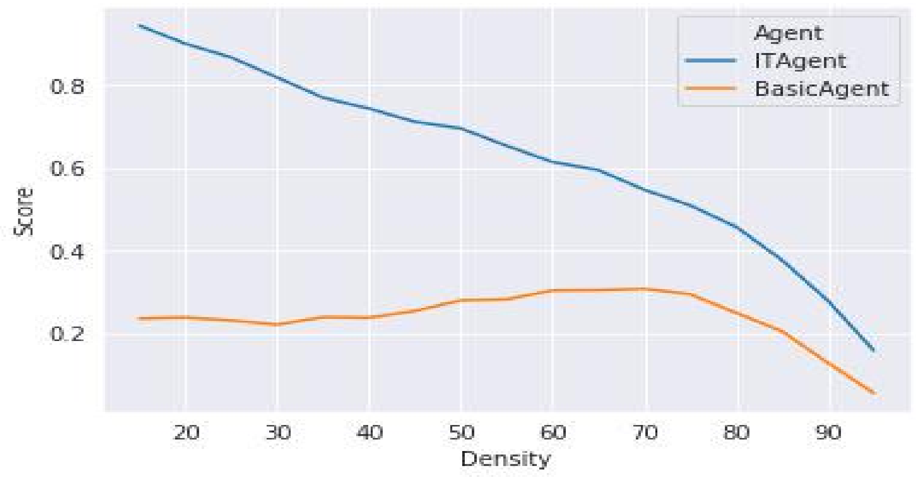
Q) Decisions: Given a current state of the board, and a state of knowledge about the board, how does your program decide which cell to search next? Are there any risks, and how do you face them?

➔ Given a state of the board, we use the probability number matrix to go further into the game. We either reveal a cell, if it has a low probability out of all the hidden cells or mark a cell as a mine if its probability number goes beyond 1. A higher priority is given to marking a mine. This ensures completeness; as a cell is surely a mine if it has a lot of neighboring cells contributing to its high probability number value. A risky situation can be encountered when there exists a symmetrically distributed sub-region of the board, where every cell has an equal probability of having a mine but not all of them can have a mine. This is the situation where the algorithm has a chance of making a mistake. In such a scenario, the probabilities of all the cells of the board is taken into account and a exploration away from the current cell is made so as to reduce the possibility of encountering a mine. Such an exploration might end up contributing to the probability metric of the neighbors of the current cell and provide additional insight. If no cell is available, then a cell is randomly chosen in the vicinity of the current cell.

Q) Performance: For a reasonably-sized board and a reasonable number of mines, include a play-by-play progression to completion or loss. Are there any points where your program makes a decision that you don't agree with? Are there any points where your program made a decision that surprised you? Why was your program able to make that decision?

➔ For a 100*100 dimensional board and 40% of the board having mines, an accuracy of 86.4% was achieved on an average, over 50 epochs. There are certain times that the program made decisions contrary to what a human mind may think. This is when the probability of all the surrounding neighbors is the same, the algorithm tends to pick a cell with lesser probability number even if the cell is not in the vicinity of the current cell. A human would tend to solve the minesweeper problem sequentially, but the computer recognizes only the probabilities and often ends up making a random choice if the probabilities of different cells is the same. This results in the errors that are observed in our program execution.

Q) Performance: For a fixed, reasonable size of board, plot as a function of mine density the average final score (safely identified mines / total mines) for the simple baseline algorithm and your algorithm for comparison. This will require solving multiple random boards at a given density of mines to get good average score results. Does the graph make sense / agree with your intuition? When does minesweeper become `hard'? When does your algorithm beat the simple algorithm, and when is the simple algorithm better? Why?



➔ This experiment was tested under a 10*10 mines map. All values(score) in this plot were the mean of 100 times similar experiment with the same setting, like the same density, same agent, same-size board but with mines in different locations.

From the plot, we can observe the average score of the two distinct agents, ITAgent and the Basic Agent. These results make sense. Basic agent had fewer inference rules than the other one. In addition, we suppose that most inference rules of ITAgent were beneficial to the result. Furthermore, the ITAgent, who knew the number of mines, had other beneficial inference rules based on this new information. As a result, the plot clearly shows that the performance of the Basic agent is much worse than the ITAgent.

The minesweeper becomes harder when the density of mines on it becomes higher. Before discussing the exact situation when the mines map become hard, let's define "hard" mines map as the one, agent cannot get the score higher than 0.5. So, the "hard" map can be different for different agents. Basic agent felt that the map is hard almost all the time and only felt comfortable when there is no any mines in the map. However, ITAgent felt great in most of the situations and only felt that board is hard when there are more than 80% mines in all cells.

Obviously, our algorithm beats the basic agent almost all the time. Their score only equaled when 100% of cells were mines. ITAgent is better than the basic one because of many reasons. For example, when selecting mines, it not only does "reveal" or "mark" actions but also update the mine-probability of neighbor cells. In addition, when doing "reveal" actions, it not only revealed those neighboring safe cells but also checked whether the surrounding hidden cells of these neighbors need to be revealed or marked or updated on mine-probability. Furthermore, when a selected cell contained a mine, it not only recorded the mine information but also check whether the surrounding cells should update their neighbor information.

Q) Efficiency: What are some of the space or time constraints you run into in implementing this program? Are these problem specific constraints, or implementation specific constraints? In the case of implementation constraints, what could you improve on?

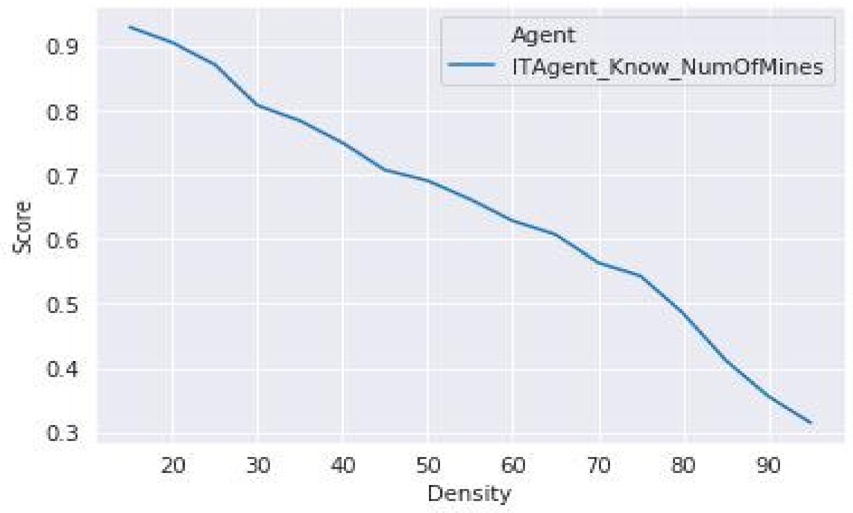
- ➔ Space Constraints: An additional map of the visited nodes is required to be maintained on the agent apart from the actual board. The combination of these two matrices provide us with the cell clue and the probability associated with the cell and helps the agent make a more informed decision. The size of either of the matrices increases with increase in the dimension d. This is required to keep track of the mines in revealed in addition to the clues that are revealed. Also, various variables like n_hid and clues have been used to evaluate and account for various corner cases like:
    - o n_hid == clue - number of revealed mines, implies all the immediate neighbors are mines
    - o n_hid == 8 - clue – number of revealed safe cells, implies all the neighboring cells are safe

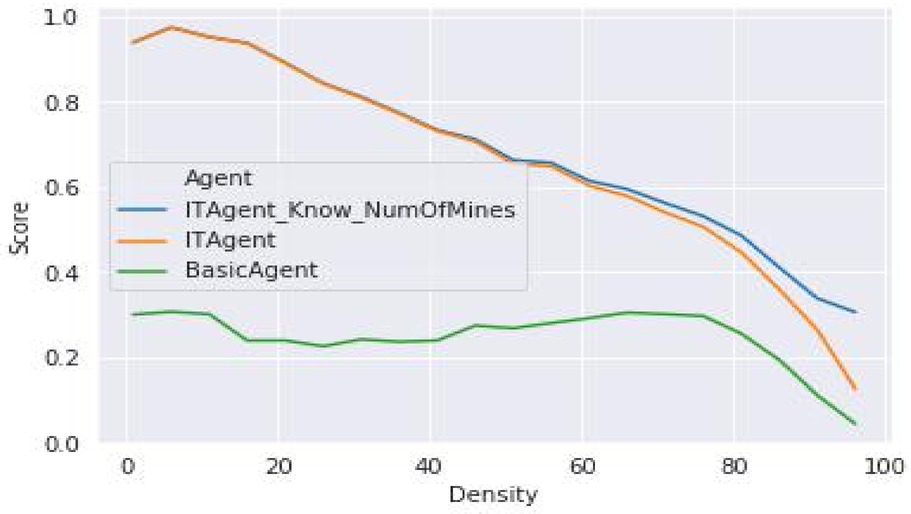
  To handle these cases, a larger subset of variables have been entertained than were actually desired.

- ➔ Time Constraints: This was primarily a concern for a more complicated agent that provided information and took decisions based on global information rather than just the local one. As soon as a cell was marked safe or dangerous, the probabilities of all the other cells have to be changed dynamically so as to enable the algorithm to make a more informed decision. This update was called for every cell explored and thus increased the running time of the algorithm for dimension d greater than 40.

Q) Improvements: Consider augmenting your program's knowledge in the following way - tell the agent in advance how many mines there are in the environment. How can this information be modeled and included in your program, and used to inform action? How can you use this information to effectively improve the performance of your program, particularly in terms of the number of mines it can effectively solve? Re-generate the plot of mine density vs expected final score for your algorithm, when utilizing this extra information.

- ➔ If the agent is informed in advance about the total number of mines, a counter can track the number of mines left to be identified or tracked in the matrix. With this information, we can prioritize a part of the matrix to be revealed or marked for mines. When the counter turns to 0, we can safely reveal the remaining mines as all of them will be safe. Alternatively, if the matrix is yet to be explored completely and the number of remaining mines is equal to the counter, we can mark all the cells as mines. This additional information may also be used to calculate the probability of a cell to have a mine, based on the value of counter, the number of unrevealed mines and the state of the cells around the unrevealed cell.





The ITAgent, who knew the number of mines, had other beneficial inference rules based on this new information. As a result, the plot clearly shows that the performance of the Basic agent is much worse than the other two. ITAgent, who knows more information, also perform slight well than the one, who did not know that.

ITAgent, who knew more, only felt that the mine map is hard when there are more than around 82%. Furthermore, ITAgent, who knows the number of mines in the map, perform better than the original one is only because it had one more additional inference rule based on this information. It could stop the game in earlier iterations when the number of all hidden cells is the number of hidden mines. So, the benefits were only brought in the end-stage of the game. The benefits are also not obvious when there is no extremely high number of mines, like the plot explaining.

Project Work Distribution:

Team:
Atharva Prabhat Paranjpe: ap1746
Yash Bhushan Barapatre: ybb4
Chi-Hui, LIN: cl1288

Basic Agent + ITAgent with known number of mines: Atharva + Yash
ITAgent: Lin
Graphs and Visualizations: Lin
Documentation + Bonus questions: Lin + Atharva + Yash