CS 520: Assignment 2 - MineSweeper, Inference-Informed Action

Aditya Vyas
Vedant Choudhary
Nitin Reddy
Siddharth Sundararajan

March 27, 2019

This project is intended to explore how data collection and inference can inform future action and future data collection. This is a situation frequently confronted by artificial intelligence agents operating in the world - based on current information, they must decide how to act, balancing both achieving a goal and collecting new information. Additionally, this project stresses the importance of formulation and representation. There are a number of roughly equivalent ways to express and solve this problem, it is left to you to decide which is best for your purposes.

1 Questions and Write-up

1.1 Representation

- Code: While writing the code, the minesweeper game is represented as a numpy matrix. We have terminal based input variables through which we can change the size of the matrix (default size = 8x8) and the mine density. Being a constraint-satisfaction problem, we model the constraint equations as tuples - (*list of variables*, *equation value*).
- Visualisation: Figure 1 displays our visualisation of the minesweeper environment. The left matrix shows the actual map of minesweeper generated from our code. The right side is a copy of the left matrix, the only difference is that the agent of our model is run on the right matrix. So as the agent makes decisions, the right plot changes - safe cells are opened, the number displayed and mines are flagged. The left actual plot helps us test the efficiency and accuracy of our agent. The information is represented just like a normal minesweeper game, through numbers, dots(where mines are placed) and flags(where our agent thinks a mine is).

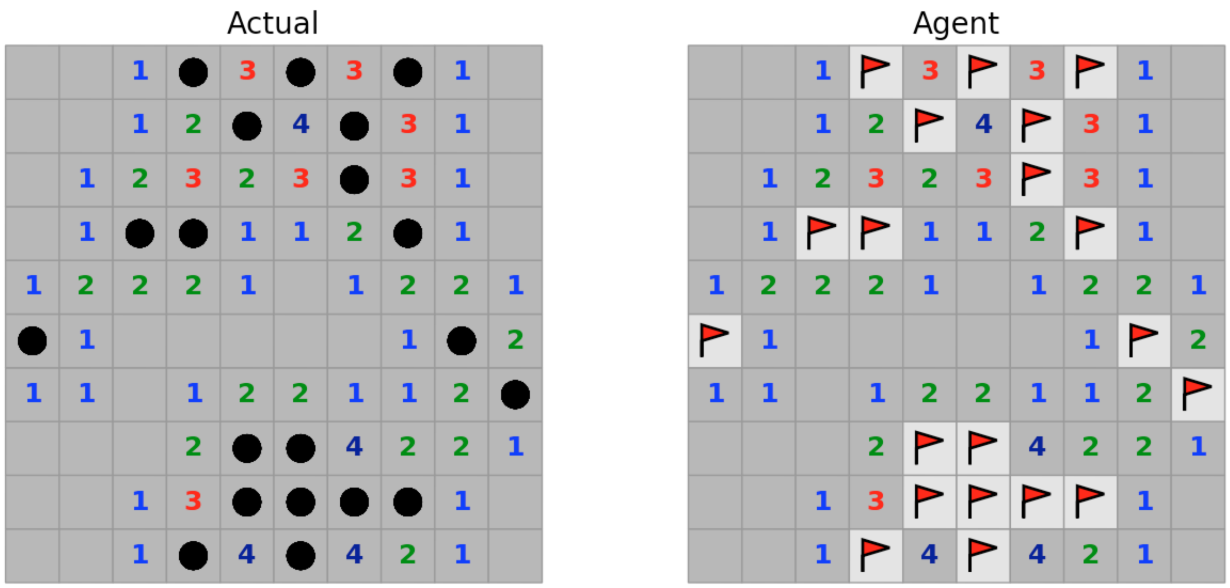


Figure 1: Minesweeper Game Visualisation

1.2 Inference

Our agent keeps 2 lists:

- mine-variables - a list of all cells/blocks which it thinks as having mines.
- non-mine-variables - a list of all safe cells/blocks

The agent plays the game in 3 phases recursively.

- **Basic solver:** The task of a basic-solver is obvious from its name - to solve the board just by looking at the cells without any equations solving or subset evaluation.
  - **Click all non-mine-cells:** By default, we have coded the environment in such a way that the first cell which the agent opens/clicks is not a mine. So the (0, 0) cell is always a non-mine variable. The agent iterates through all the variables in the non-mine-variables list and opens them. When a cell is opened by the agent, the following actions are taken by it
    - **Form the constraint-equation for the variable:** When a non-mine-variable is opened, a constraint-equation is formed which consists of addition of its neighbours and an equation value.
    - **Remove the variable from other equations:** SInce a cell is opened so it becomes harmless. Hence it has to be removed from all the constraint-equations of other variables in which it occurs
  - **Flag all mine cells:** If the number of variable in equation is equal to value then any variable in the list mine-variables is a potential mine and has to be flagged.

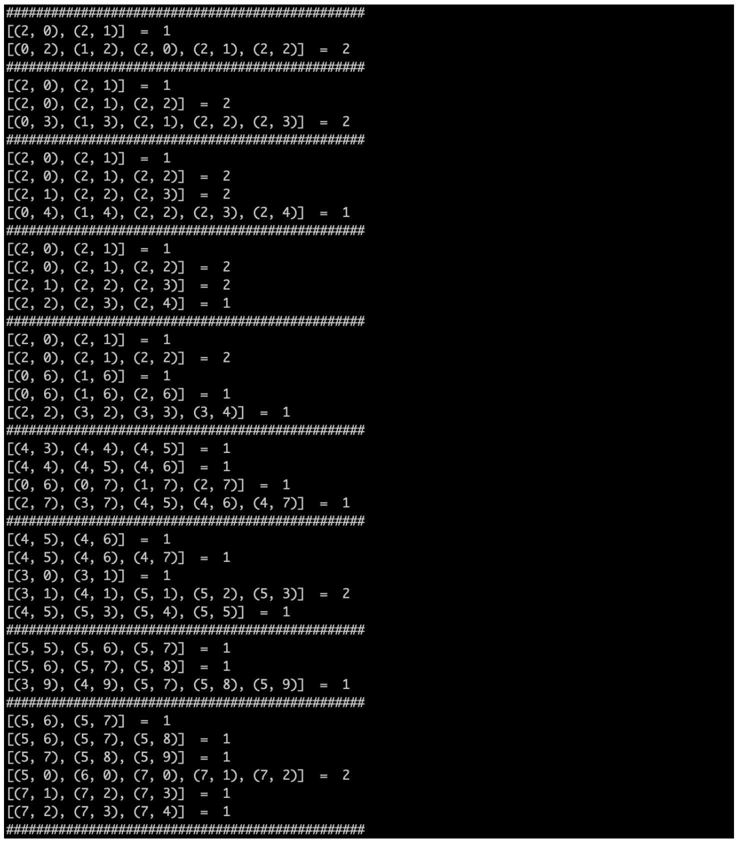
The agent flags all the cells in this list one by one. After flagging the cell it removes the cell from all the constraint-equations it is present in. However, there is a major difference here as compared to a non-mine cell. We do not only remove a mine-variable from the equation but also subtract the value of the equation, since now it contains one less mine-variable.

  - **Check the list of equations for more mine-variables and non-mine variables:** This is the step which yields more mine and non-mine variables for the agent to flag and open respectively. After removing non-mine variables and min variables from other equations, the agent checks it knowledge base for the following two types of equations
    - $A + B + C = 0$**:** Any equation with a value of 0 gives us non-mine-variables. Here A, B and C are all non-mine variables.
    - $A + B + C = 3$**:** Any equation where the number of variables in the equation is equal to the equation value gives us mine variables. Here A, B and C are all mine variables.

- **Subset solver**: There will be a time when the above basic solver can no longer find individual equations that yield more non-mine and mine variables. This is where we try breaking down subsets of equations. The agent creates subsets of constraint equations present in the knowledge base and starts solving them. For example: if there are two equations $A + B + C = 2$ and $B + C = 1$, then the agent deduces that A definitely has a bomb by substituting the second equation into the first one. This procedure is run iteratively and it solves the knowledge base quite efficiently.

- **Random opening solver:** Finally there will be a time when no further subsets can be deduced and all the equations are exhaustive of each other. For example: if there are two equations $A + B + C + D = 3$ and $B + C = 1$, solving the subset does not lead to any new inferences. In such a scenario, the agent relies upon random (with a heuristic) opening of cells to make and deduce new equations. If the agent comes to this stage, it means that the current knowledge base is not sufficient enough to make any further inferences. The next cell selection is not totally random. The agent uses a heuristic based on the probability of any cell having a bomb before making a random click. The lesser the value of this heuristic, the more likely it is to open that cell.

Based on the three phases the agent goes through, the agent deduces almost everything possible before continuing further. But, the agent can be improved by incorporating a backtracking algorithm. With this incorporation, we can ensure that the agent has been thorough by inferring all possible inferences from the knowledge base. This would ensure that the agent is extensive in solving the minesweeper. Our agent is able to deduce everything from a clue before proceeding because from the way we have coded our agent, it will try to break down equations to either of the two forms we described previously. If there are no such equations it will store them and proceed to getting more clues. We are sure about this because we also have a way of visualising the knowledge base of our minesweeper agent



```
################################################
[(2, 0), (2, 1)]  =  1
[(0, 2), (1, 2), (2, 0), (2, 1), (2, 2)]  =  2
################################################
[(2, 0), (2, 1)]  =  1
[(2, 0), (2, 1), (2, 2)]  =  2
[(0, 3), (1, 3), (2, 1), (2, 2), (2, 3)]  =  2
################################################
[(2, 0), (2, 1)]  =  1
[(2, 0), (2, 1), (2, 2)]  =  2
[(2, 1), (2, 2), (2, 3)]  =  2
[(0, 4), (1, 4), (2, 2), (2, 3), (2, 4)]  =  1
################################################
[(2, 0), (2, 1)]  =  1
[(2, 0), (2, 1), (2, 2)]  =  2
[(2, 1), (2, 2), (2, 3)]  =  2
[(2, 2), (2, 3), (2, 4)]  =  1
################################################
[(2, 0), (2, 1)]  =  1
[(2, 0), (2, 1), (2, 2)]  =  2
[(0, 6), (1, 6)]  =  1
[(0, 6), (1, 6), (2, 6)]  =  1
[(2, 2), (3, 2), (3, 3), (3, 4)]  =  1
################################################
[(4, 3), (4, 4), (4, 5)]  =  1
[(4, 4), (4, 5), (4, 6)]  =  1
[(0, 6), (0, 7), (1, 7), (2, 7)]  =  1
[(2, 7), (3, 7), (4, 5), (4, 6), (4, 7)]  =  1
################################################
[(4, 5), (4, 6)]  =  1
[(4, 5), (4, 6), (4, 7)]  =  1
[(3, 0), (3, 1)]  =  1
[(3, 1), (4, 1), (5, 1), (5, 2), (5, 3)]  =  2
[(4, 5), (5, 3), (5, 4), (5, 5)]  =  1
################################################
[(5, 5), (5, 6), (5, 7)]  =  1
[(5, 6), (5, 7), (5, 8)]  =  1
[(3, 9), (4, 9), (5, 7), (5, 8), (5, 9)]  =  1
################################################
[(5, 6), (5, 7)]  =  1
[(5, 6), (5, 7), (5, 8)]  =  1
[(5, 7), (5, 8), (5, 9)]  =  1
[(5, 0), (6, 0), (7, 0), (7, 1), (7, 2)]  =  2
[(7, 1), (7, 2), (7, 3)]  =  1
[(7, 2), (7, 3), (7, 4)]  =  1
################################################
```

Figure 2: Printing all the equations in the knowledge base

1.3 Decision

Given the current state of the board, and a state of knowledge about the board - the knowledge base - the agent uses the following 3 steps to move ahead

- Check the list of mine and non-mine variables using basic solver. If any mines and non-mines found, then flag and open them respectively. If nothing can be deduced, proceed to evaluating subsets.
- Solve subsets and find non-mines and mines by breaking down the equations.
- Click randomly if no subsets can be resolved.

There is one potential risk that is bound to occur in the third phase since the agent opens cells in a quasi-random manner. Although the heuristic reduces the risk, it does not guarantee the opening of a safe cell. As mentioned earlier, this can be tackled using a backtracking method. Another risk/disadvantage is that in environments with high-mine density the heuristic-method of randomly opening cells would fail.

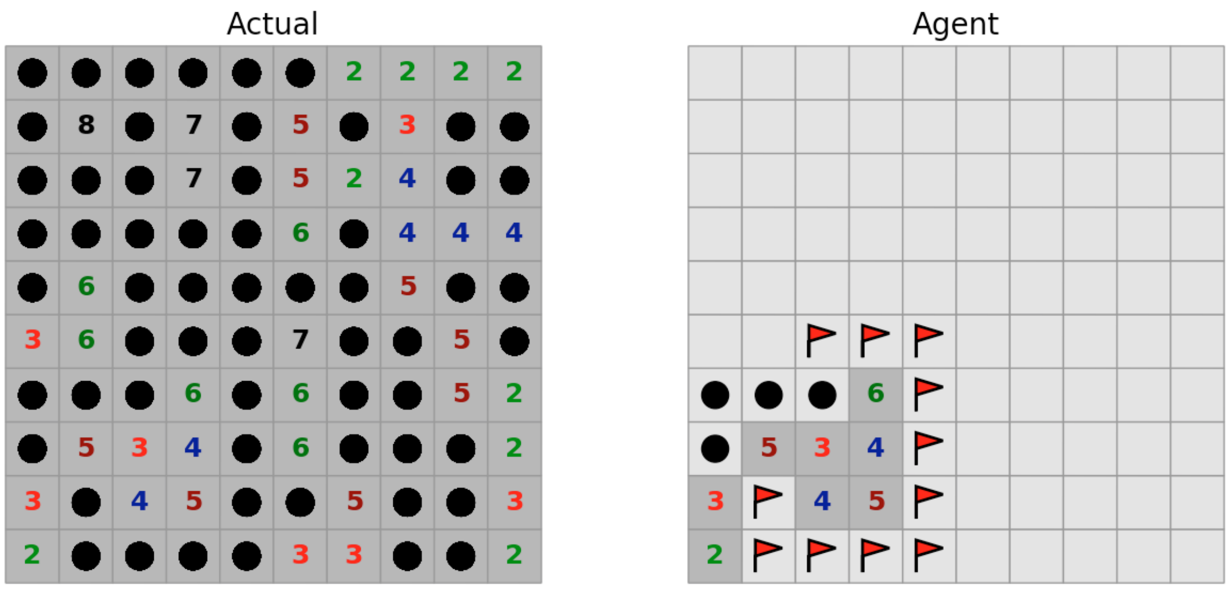


Figure 3: An example where the random opening strategy based on heuristics would get stuck

Our random opening algorithm looks at neighbours of all the open cells and then randomly clicks on the neighbours of an open-cell with the minimum risk value. However, in cases where all the neighbours of open-cells are mines, it will not be able to proceed further. For this we have coded a final phase for our agent - When everything else fails and there is no way of proceeding further just click randomly anywhere - something which we also do when we are stuck.

1.4 Performance

1.4.1 Performance 1

We take a 15x15 environment grid and use 0.13 mine density, which leads us to the following progression.

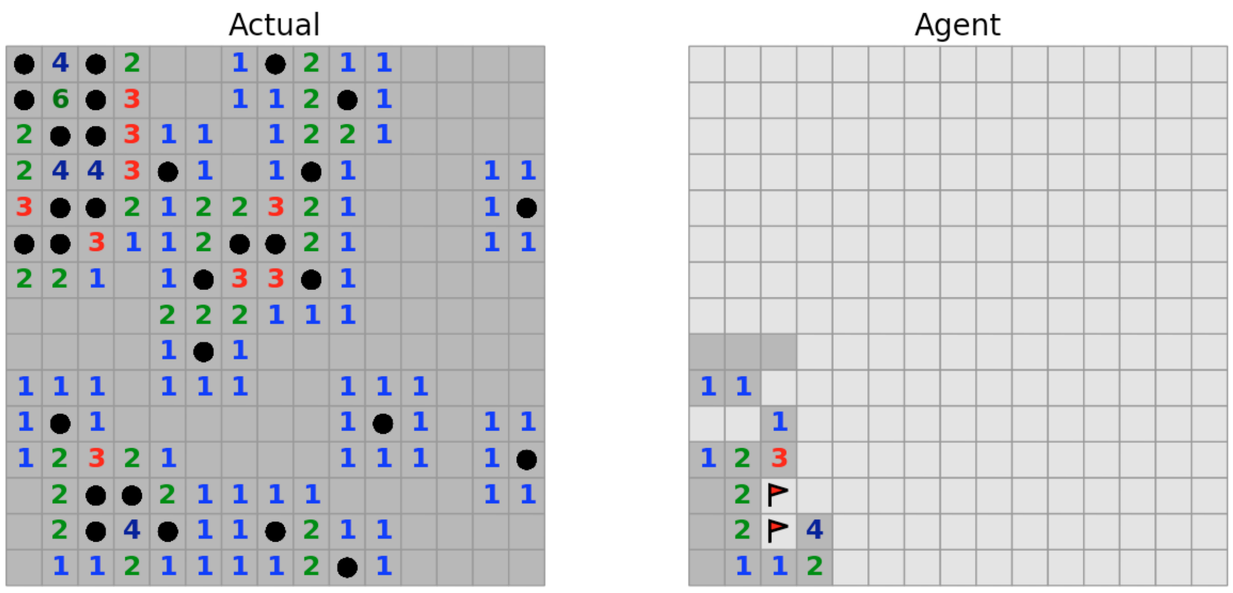


Figure 4: Initial stage of gameplay

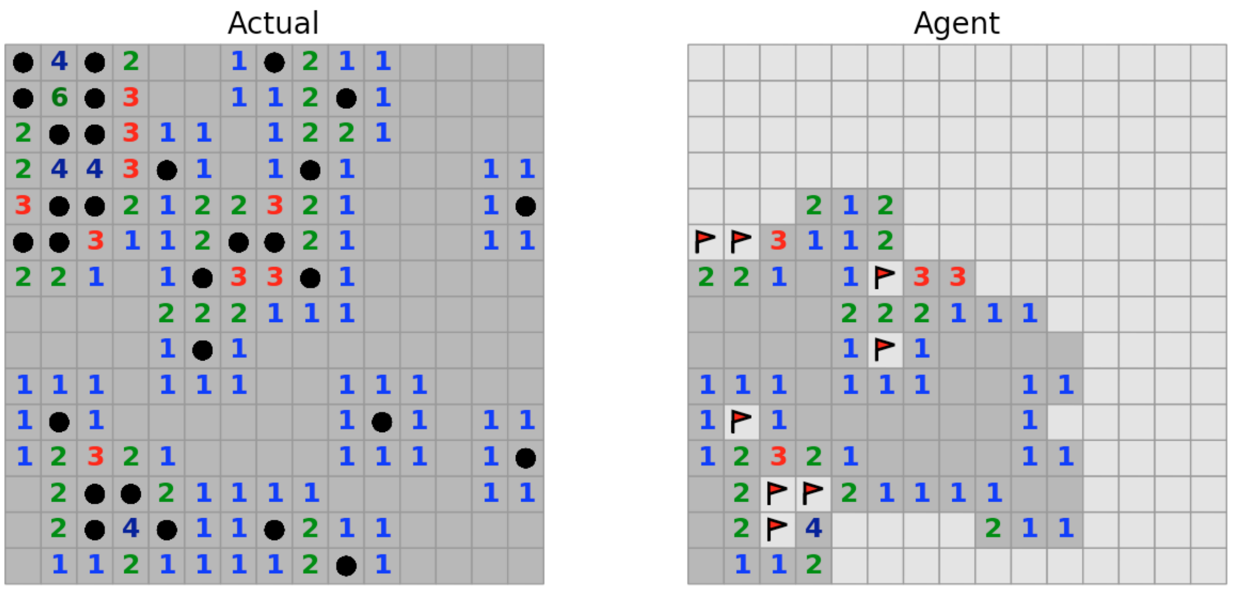


Figure 5: Midway of gameplay

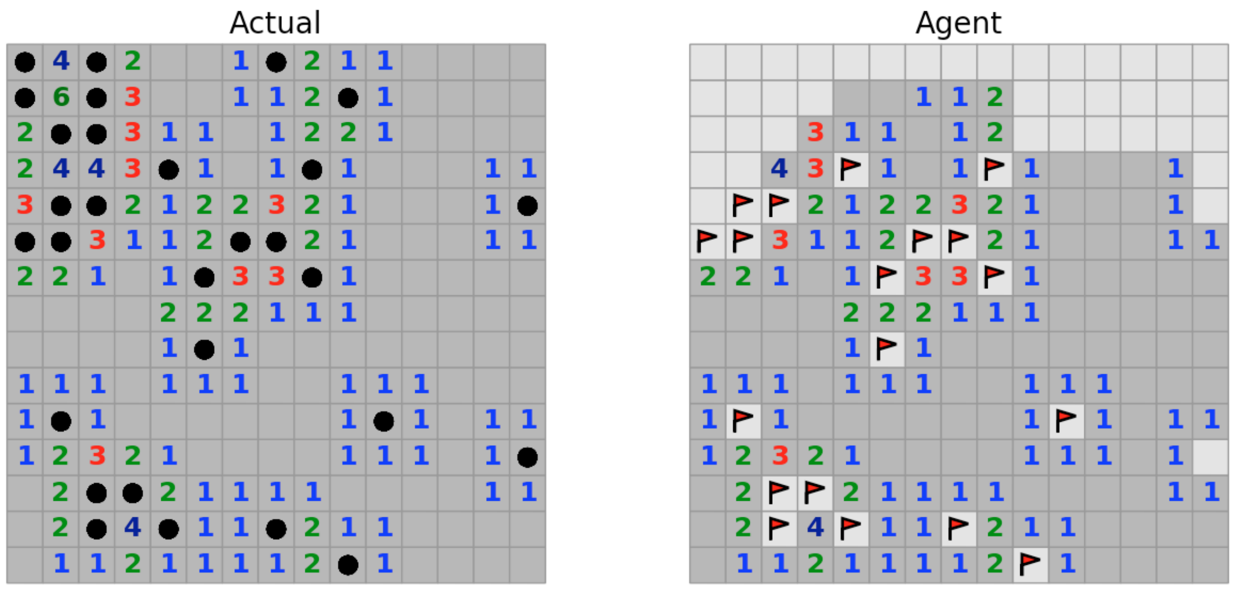


Figure 6: Almost finished playing the game

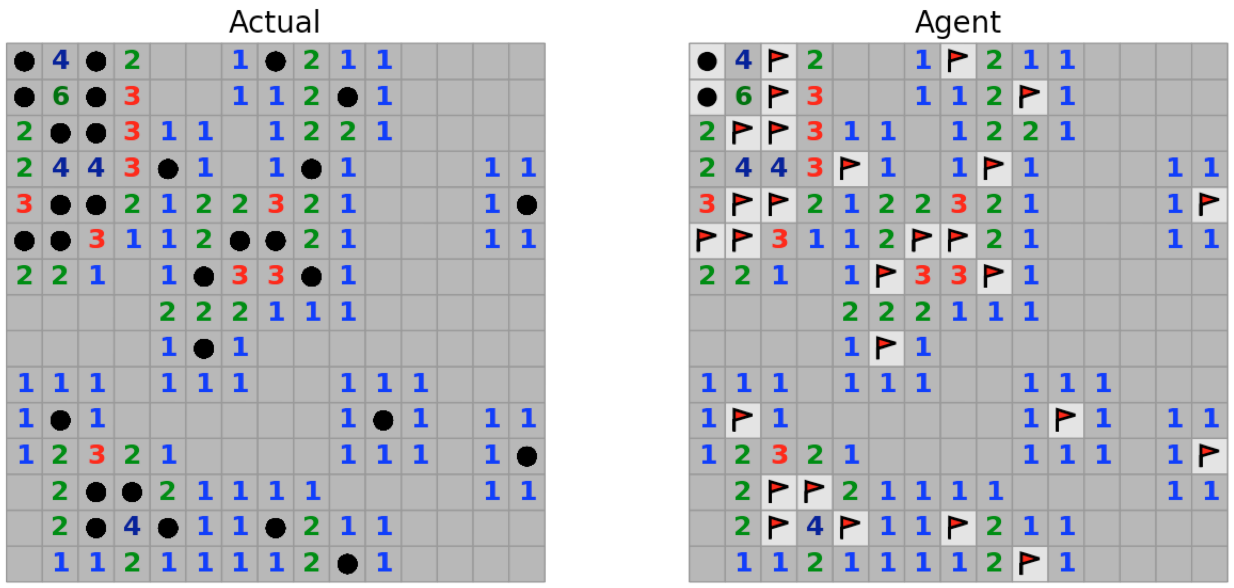
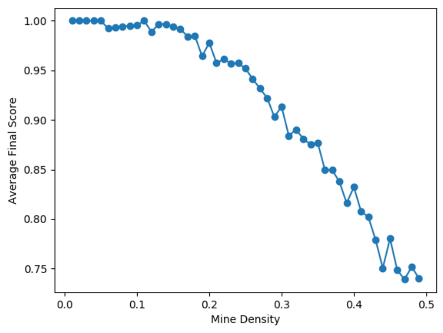


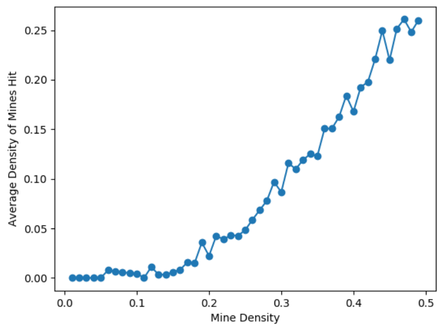
Figure 7: Game over

We can observe that in the above game, our agent played very efficiently and was able to flag all the mines properly except the last 2 in the top left corner. It was not a surprise for us because on observing the set of equations that were left at that stage, there were no subsets which could be evaluated and hence no way for the agent to deduce mine and non-mine variables. Technically, we can use backtracking in such cases but we have left it for future work. Hence, based on whatever we coded, the agent just clicked on the cells randomly.

1.4.2 Performance 2



(a) Average Final Score vs. Mine Density



(b) Average Density of Mines Opened vs. Mine Density

Figure 8: Performance metrics for a 15x15 minesweeper vs. Mine density

The agent plays minesweeper on a board of size 15x15 and for each mine-density, we play 10 games. The mine density ranges from 0.01 to 0.5 with a step-size of 0.01. We observe that the results obtained agree with our intuition.

- As the mine density increases, the average final score decreases. For the first few mine-densities the number of mines flagged is exactly equal to the total number of mines present giving a score of 1.0. However, with increasing mine-density the difficulty also increases and hence the average score goes down.
- Similarly, with increased mine-density the average density of mines hit by the agent also increases. This also agrees with our intuition and we see that although the agent opens more number of mines as the density increases, this increase is gradual and not steep.

Based on the above graphs, we see that the game can be called "hard" when the density of mines hit by the agent is 0.10 - the agent hits 10 percentage of the total mines - which occurs at mine-density 0.3.

1.5 Efficiency

While implementing the minesweeper agent, we did encounter a time constraint

- The knowledge base of the agent is represented by a python list with the individual equations as tuples. When the agent removes a variable from other constraint-equations, it will make a pass through its entire knowledge base. This is a time consuming operation. For very large grid sizes and large number of equations, this leads to increase in time complexity.

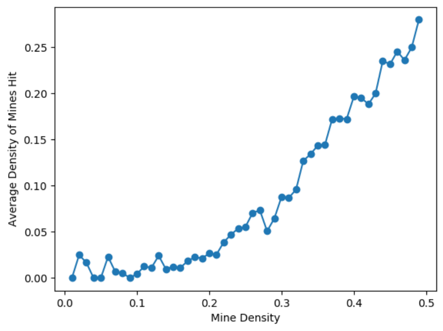
The above problem is an implementation-specific constraint which can be removed by changing one aspect of the way in which we store equations. We also store the equations which a variable is part of as a separate attribute of the variable. Hence, when we need to remove variable, we directly access this attribute containing all the equations which this variable is part of and make a pass through only those specific equations rather than the entire knowledge base.

1.6 Improvements

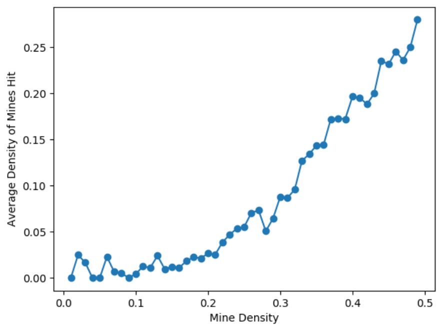
The information about the total number of mines in the environment is modelled in such a way that if the number of cells flagged becomes equal to the total number of mines in the game, then the agent will blindly open all the closed cells without any inferences. This check can be added at various stages in our code to make the solving of the game faster

- In the basic solver phase, the agent makes the check when all the obvious equations have been checked. This means that the agent makes the check at the end of basic solver. If the number of mines present in the game equal the number of mines uncovered, the agent uncovers all other cells and then exits the game.
- In the subset solver phase, the agent makes the check after solving the subsets at every iteration (i.e., before using the created subsets to go for the recursive round). Similarly, if the count found is equal, then the agent uncovers all the cells and exits the game.

This information helps the agent to be computationally more efficient. This is because the agent is able to reach a consensus based on this information, so it saves on checking the remaining cells and just uncovers them and exits. This is efficient compared to the normal CSP agent. Hence, the time to play the game decreases a lot which can be seen by the performance plots below.

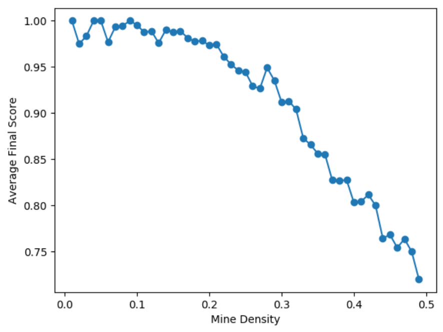


(a) The Improved CSP Agent

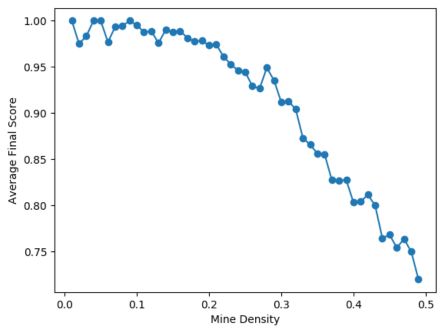


(b) The Normal CSP Agent

Figure 9: Average Density of Mines Hit vs Mine Density

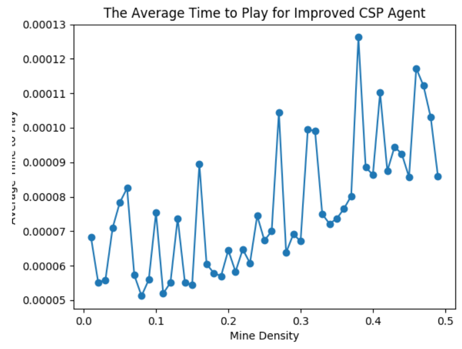


(a) The Improved CSP Agent

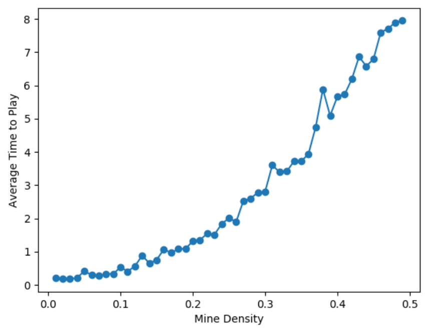


(b) The Normal CSP Agent

Figure 10: Average Final Score vs Mine Density



(a) The Improved CSP Agent



(b) The Normal CSP Agent

Figure 11: Average Game Playing Time vs Mine Density

From the above plots, we observe that the average final score and the average number of mines opened is the same for both the agents. However, there is a very significant difference in the playing times of these agents and the reason is what we mentioned before - the improved agent has an extra knowledge which makes it stop evaluating the equations once the total mines are opened or flagged.

2 Bonus: Chain of Influence

- **Based on your model and implementation, how can you characterize and build this chain of influence? Hint: What are some 'intermediate' facts along the chain of influence?**
  - In the program which we have developed, we start off with (0, 0) cell and we have made sure that this cell does not contain any mine. Opening the cell leads to the creation of our knowledge base. This KB gives information regarding the adjacent cells to the current cell (which is (0, 0) in the first step). Although not

implemented, but we can easily keep a record of the chain of influence by creating a dictionary/hash table. The key can be the current cell and the value can be the cells from which the state of the current cell can be deduced (parent cells) - for which the steps have been elaborated in previous sections. For example, the first step is opening (0, 0) - it will have no parent cells because there was no KB in our system. After that, to open/flag (0, 1) (since we open cells sequentially), we need the knowledge from opening cell (0, 0), hence in this case the hash table/dictionary will have key as (0, 1) and value as (0, 0). By following this simple logic, we can incorporate chain of influence in our current code base.

- **What influences or controls the length of the longest chain of influence when solving a certain board?**
  - The chain of influence can be either deep or wide. If the length of longest chain of influence has to be categorized by how deep it is, then that length can be controlled by how less random cells are opened. In our case, since we open up the cells sequentially, we are creating a deeper chain of influence with minimal branching (since neighborhood of cells are in constraint equations of each other, so are represented by a smaller KB). However, if the length of longest chain of influence has to be categorized by how wide it is, then that length can be controlled by how many random cells are opened. If more random cells are opened, we are looking at constraint equations having no dependency on each other, hence no new cell can be deduced in a deterministic way and we will end up exploring a lot of cells with no definite answer.
- **How does the length of the chain of influence influence the efficiency of your solver?**
  - The length of chain of influence has a direct impact on the efficiency of the solver. When there is a thin deep chain of influence, the efficiency of the solver is high because the solver is able to deduce the KB in a deterministic fashion, rather than deducing randomly. While, if the solver chooses random points to open up, its efficiency decreases as it is more unsure about the state of a particular cell. In our program, the thinner deep chain of influence is created by basic solver + subset problem, while more branched chain of influence is induced by the random heuristic step of the solver. Any new constraint equation expands the KB and hence, increases the computing cost. If the solution to the problem is a thinner deeper chain of influence, the constraint equations formed along the way are dependent on each other hence, the KB does not expand as much as it would if independent constraint equations are formed (due to random cell opening).
- **Experiment. Can you find a board that yields particularly long chains of influence? How does this vary with the total number of mines?**
  - A board which will have no mines will yield the longest chain of influence, because it will keep expanding based on the first query. This type of board can be achieved in our program when the only random cell (without any knowledge) opened is the

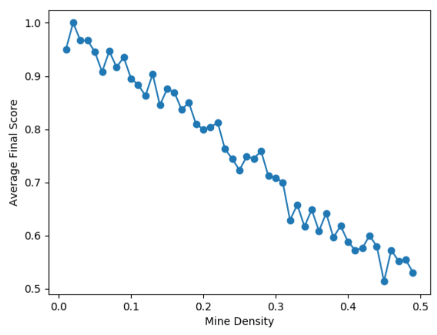
initial cell (0, 0). As the number of mines are increased, the chain of influence will decrease. The mines essentially block the expansion of the KB.

- **Experiment. Spatially, how far can the influence of a given cell travel?**
  - The farthest a chain of influence can go is the whole spatial map of the board. This can be strengthened by an example. Suppose, we start off at (0, 0) (as in our program), if there is only one mine and it is at the position (Size-1, Size-1), then this cell would get affected by the previous cell opened, and that previous cell would depend on its parent cell. This will basically form a chain of influence from the starting point (0, 0) to (Size-1, Size-1). Since, no other mine is present in this scenario, the chain of influence never breaks till last cell is opened.
- **Can you use this notion of minimizing the length of chains of influence to inform the decisions you make, to try to solve the board more efficiently?**
  - If we pick the cells which are more connected to the constraint equations (coming most of the time in a set of equations), we can minimize the length of chains of influence.
- **Is solving minesweeper hard?**
  - Solving the maze from our solver is mostly not that hard, given the computing capabilities allows us to. As per our agent, it correctly deduces most of the cells based on basic solver + subset solution. However, in the cases when no deduction can be made, the agent uses a heuristic approach based on probabilities to open up a new cell. This can lead to an error that the opened cell is actually a mine. When this kind of scenario arises quite often, we can say that at that point, the size of the board and mine density makes solving the game harder.

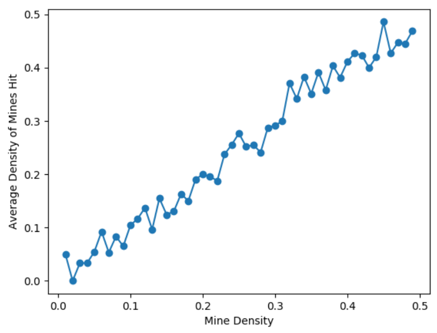
3 Bonus: Dealing with Uncertainty

- **When a cell is selected to be uncovered, if the cell is 'clear' you only reveal a clue about the surrounding cells with some probability. In this case, the information you receive is accurate, but it is uncertain when you will receive the information.**
  - We have implemented this part of the bonus question, and the agent will apply the above condition when we pass a terminal argument "-bp" along with a probability of the clue being revealed about the surrounding cells.
  - Everything remains the same except that every time the agent clicks open a cell, it's constraint-equation is formed with the above probability and so we either add the constraint equation to our knowledge-base or we dont. In this way, the agent always has less information to make any inference or deductions than the previous version. This also has an effect on randomly clicking any square and the clue in the cell has an impact on the risk factor being calculated. If the cell is clicked and not revealed, we do not consider it towards risk factor calculation.

- We see that, the performance of the Minesweeper agent with this uncertainty condition has a significant impact on its performance. As the mine density increases(difficulty increases) the performance decreases. This is due to the fact that, when there are fewer mines, most of the clue tends to reveal almost same inferences or deductions, but however, as density increases, each clue tends to be more important and discarding such clues based on the probability will lead to loosing important information about the environment.
- We can see the performance of this agent here:



(a) Average Final Score vs. Mine Density



(b) Average Density of Mines Opened vs. Mine Density

Figure 12: Performance metrics for a 15x15 minesweeper vs. Mine density

We observe that this agent performs worse than the normal CSP agent. For the normal CSP agent, the average final score tends to decreases gradually and remains constant for the first few mine-density values. However, the probability agent's score decreases steeply and reaches almost 0.5 at mine-density of 0.5. Similarly, the average density of mines hit also increases steeply - this can be seen by the fact that for the normal CSP agent the density of mines hit crosses 0.1 at mine-density of 0.3 at which point it becomes a hard game for it. However for the probability agent, it becomes hard at mine-density of 0.1 at after which 10 percentage of mines are hit.

- **When a cell is selected to be uncovered, the revealed clue is less than or equal to the true number of surrounding mines (chosen uniformly at random). In this case, the clue has some probability of underestimating the number of surrounding mines. Clues are always optimistic.**
  - Similarly as in the case above, on clicking the square, we would receive the clue to be less than or equal to true surrounding mines, which will be chosen uniformly at random. So once, we have the new equation added to the whole set of constraints, we will only proceed with flagging the mines since that is what the agent will be fully confident about.

  - For example, $A+B+C=3$, would only mean that all the cells A, B, C are mines, but $A+B+C=0$ does not mean that all cells are safe to click since the clues are optimistic. Based on this logic, the agent only flags mines.

- **When a cell is selected to be uncovered, the revealed clue is greater than or equal to the true number of surrounding mines (chosen uniformly at random). In this case, the clue has some probability of overestimating the number of surrounding mines. Clues are always cautious.**

  - Similarly as in the case above, on clicking the square, we would receive the clue to be greater than or equal to true surrounding mines, which will be chosen uniformly at random. So once, we have the new equation added to the whole set of constraints, we will only proceed with opening the safe square since that is what the agent will be fully confident about.
  - For example, $A+B+C=0$, would only mean that all the cells A, B, C are safe to click, but $A+B+C=3$ does not mean that all cells are mines since the clues are cautious. Based on this logic, the agent only clicks squares. Also, follows the same logic for choosing square to click obtained from the subset constraints solver.