SCHOOL OF COMPUTER SCIENCE

Mine Sweeper

Khanh Nguyen
March 22, 2021

1 ABSTRACT

In this report, we 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 real 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.

2 GENERAL ARCHITECTURAL

2.1 REPRESENTATION

Code: In the processing of writing the code, the minesweeper game (environment) is represented as a numpy matrix. Moreover, the users can change the size of the matrix (8 x 8 by default) and the mine density. However, in the given example and the figure, we will use a 10 x 10 matrix to give reader a more realistic view. Being a constraint-satisfaction problem, we model the constraint equations as tuples – (list of variables, equation value).

Visualization: Figure 1 displays our visualization of the mine sweeper environment. The left matrix shows the actual map of minesweeper generated from our code. The right sight is a copy of the left matrix. However, the main difference is that the agent of our model is implemented in the right side. So, as the agent makes decision, the right plot changes – safe cells are opened, the number is displayed, and mines are flagged. The information is represented as a normal mine sweeper game, where numbers, dots as mine and flags where agent thinks a mine is.

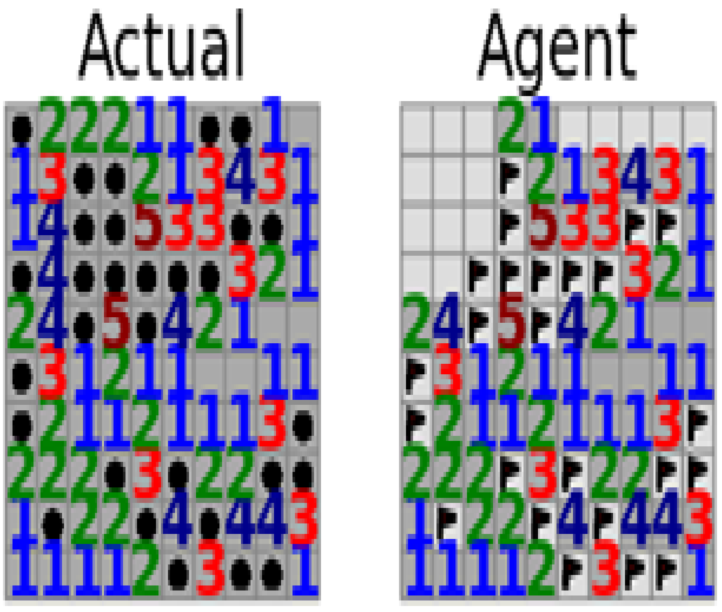


Figure 1: Actual gameplay

2.2 INFERENCE

Our agent keeps two lists:

- **Mine-variables:** a list of all cells/blocks which it thinks as having mines.
- **Non-mine-variables:** a list of all save cells/blocks.

The agent plays the game in 3 phrases recursively:

- **Basic solver:** The agent plays the game in 3 phrases recursively:
  - **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.
    - When a non-mine-variable is opened, a constraint-equation is formed which consists of addition of its neighbors and an equation value.
    - **Remove the variable from other equations:** After a cell is opened, it becomes harmless. Hence it must be removed from all the constraint-equations of other variables in which it occurs.

  - **Flag all mine cells**: If the number of variables 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-variable and non-mine-variable:** 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 a non-mine variables. Hence, all variables are all none-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. Hence, all variables are all mine-variables.

- **Sub-set 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. However, 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.

2.3 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.

However, 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.

Our random opening algorithm looks at neighbors of all the open cells and then randomly clicks on the neighbors of an open cell with the minimum risk value. However, in cases where all the neighbors 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.

3 Performance

3.1 Perfomance Gameplay

As usual, we take a 10 x 10 environment grid and use 0.3 mine density.

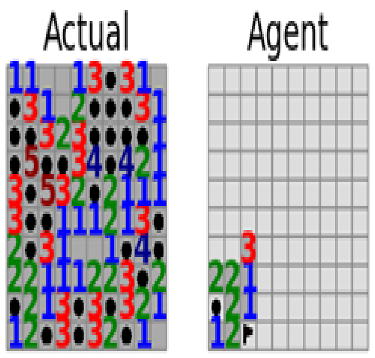


Figure 2: Initial state of gameplay

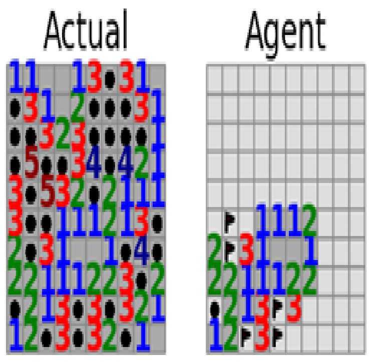


Figure 3: Mid way of gameplay

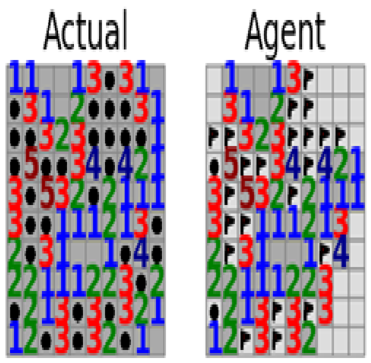


Figure 4: Almost finish playing the game

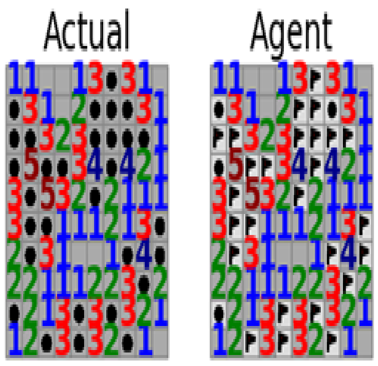


Figure 5: Game over

We can observe that in the above game, our agent played very efficiently and was able to flag almost all the mines properly. However, during the beginning of them game, even though we code the base as a non-mine variables, we cannot exclude the fact that the unopened neighborhoods can be mines. It was not a surprise for us because on observing the set of equations that were at the beginning of the stage, there were no subsets which could be evaluated and hence no way for the agent to deduce mine and non-mine variables. Hence, based on what we coded, the agent just clicked on the cells randomly.

3.2 Performance Visualization

The agent plays minesweeper on a board of size 10 x 10 and for each mine-density, we play 10 games. The mine density ranges from 0.1 to 0.5 with a step-size of 0.1. 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 a greater 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.

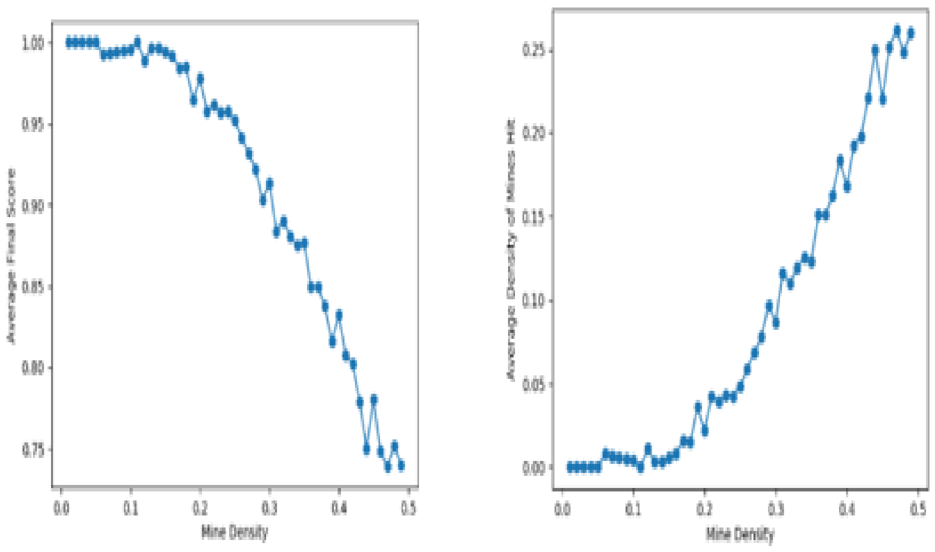


Figure 6: Average final score with average density of mines hit vs mine density

4 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 a 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

5 Improvement

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 plot below.

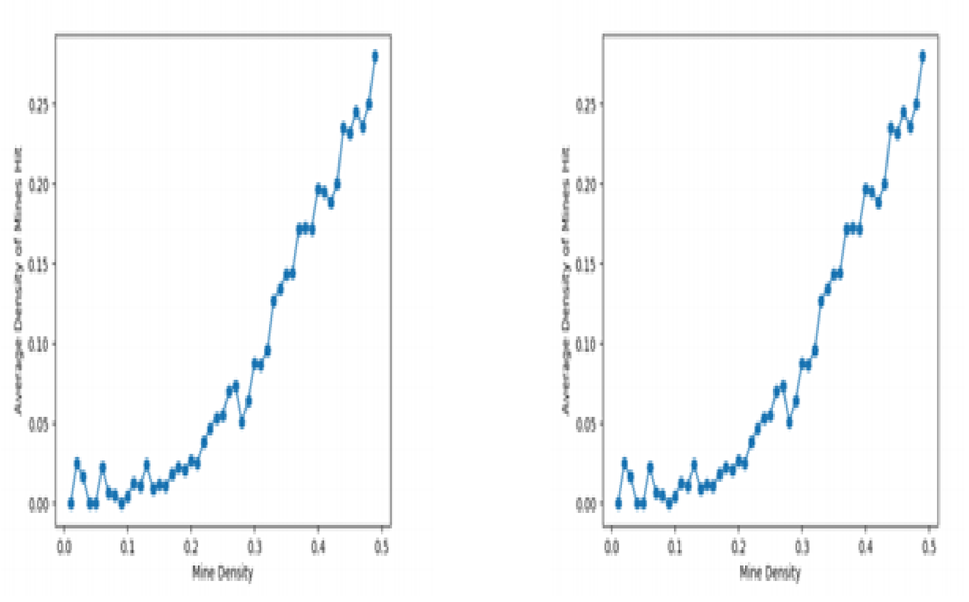


Figure 7: Average density of mines hit vs mine density with improved CSP Agent(left) and Normal CSP Agent(right).

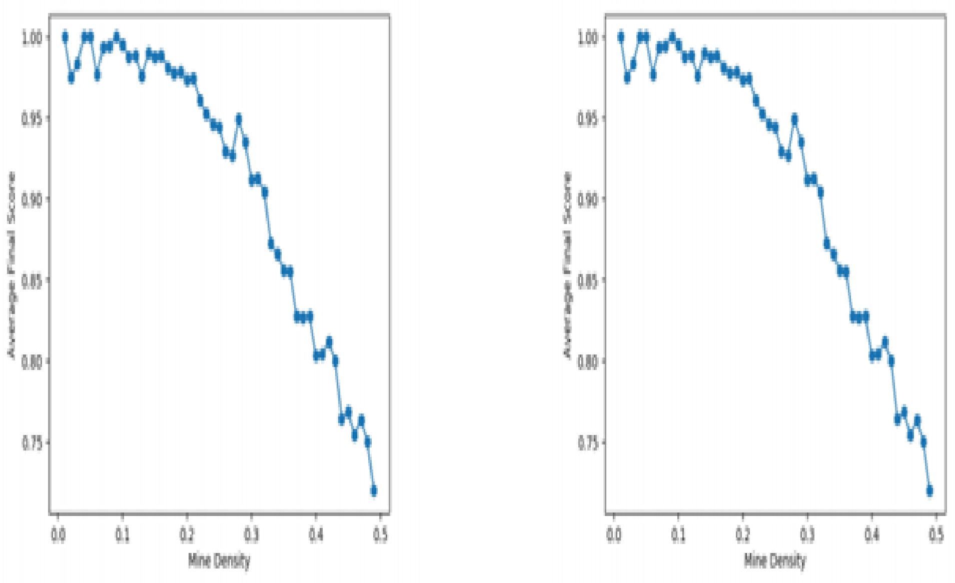
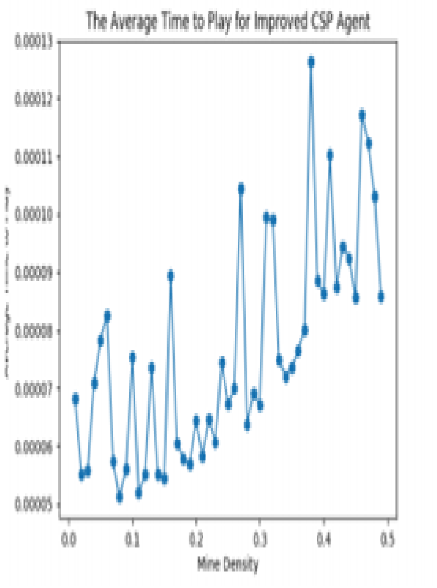


Figure 8: Average final score vs mine density with improved CSP Agent(left) and Normal CSP Agent(right).



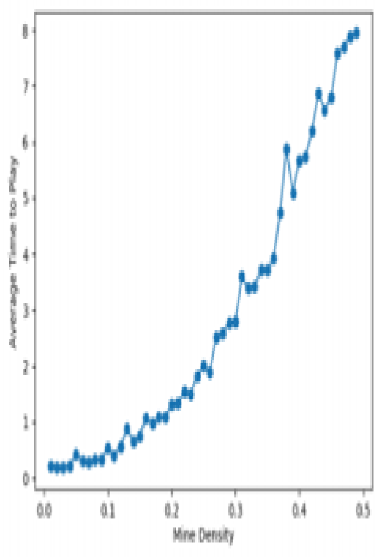


Figure 9: Average game playing time vs mine density with improved CSP Agent(left) and Normal CSP Agent(right).

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.