Report - Assignment 2 : Minesweeper

Diksha Prakash Kathikeyan Sethuraman Arun Sinhmar

October 2019

Abstract

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 Background and Program Specification

The legend for the MineSweeper shown in Figure 1 is as under:

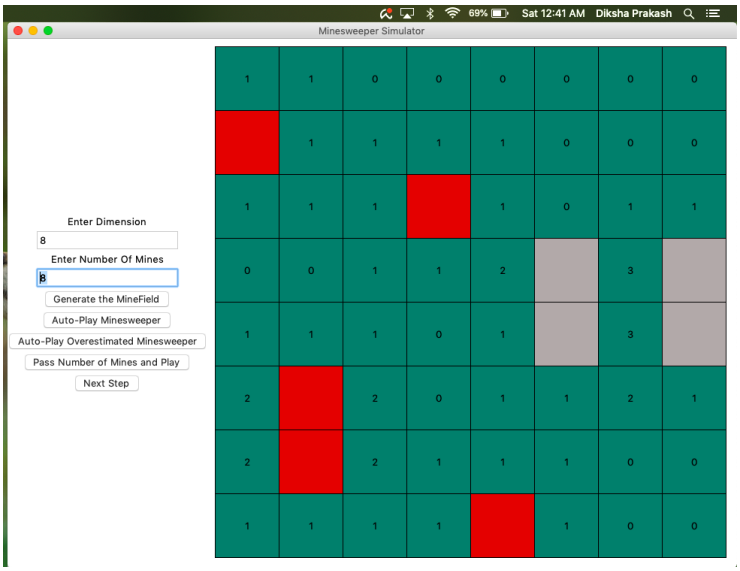

Figure 1: MineSweeper Simulator Window

In Figure 1, Red color indicates Mines. Green color indicates the safe and visited boxes and the number within the green boxes represents the clue i.e. number of mines in the neighborhood and the Gray color indicates the unvisited boxes.

2 Questions and Writeup

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

BOARD : The board is defined using the class 'Environment' and hence, represented using an object of the type 'Environment'. This contains the matrix to store and various functions to interact with the 'minefield'. The functions are:

1. mineFieldValues() : To create 'n' mines at random locations in a matrix of given 'dimension'.
2. getInformation() : To generate clues about the mines i.e. function to get print information about the neighbours of any given Cell of the mine field.
3. QueryMethodBox() : To return information about the queried box to other programs.

The 'mineField' is essentially an array of 'Cells'. Every 'Cell' has the following attributes:

1. 'row' and 'col' : To identify the position of the 'Cell' in the matrix.
2. 'mine' : Flag to indicate the Presence (IS_MINE) or absence (IS_NOT_MINE) of mine.
3. 'visited' : Flag to indicate if a given cell is visited (IS_VISITED) or not(IS_UNVISITED)
4. 'clue' : To indicate the number of mines around a given location. Its range is between 0 and 8 (inclusive). It is set to 'NUMBER_CLUE' by default.

Knowledge/Information that clue cells reveal : The MainProgram.py creates an object each of the Environment and the Agent, both of which contain independent mine fields. (The minefield has been represented by a matrix where each element is itself an object containing various information). The only difference between the mine fields generated in them are:

1. The Environment's minefield is generated by giving the row and column information (via Box[i,j]) to getInformation() method which stores the clues i.e. number of mines in the neighbouring cells.

2. The Agent's observed minefield i.e. 'agentObservedMineField' has no information about the mines. The only information passed to it are the row and column. The agent moves along the cells collecting the clues and keeps updating its data.

2.2 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?

The initial knowledge base is TRUE, whenever a new clue is revealed to the agent, the addInfoToKnowledgeBase() method generates all the possible combinations of the constraints using "combinations" method of "itertools" module. According to the nature of the clue revealed by a given neighbour i.e. 'True' in presence of a mine and 'False' in the absence, the value is substituted in the expression ('expression_subset') and finally OR-ed to the final expression ('expression_box_neighbours') which is then converted to CNF (Conjunctive Normal Form) using 'to_cnf' method from 'sympy' module.

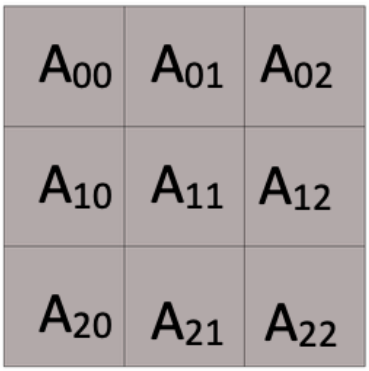

Figure 2: MineSweeper Denotation Map

In the Figure 2, let's say that the clue for cell A_{20} is 2. There exist three possible neighbours for this cell namely, A_{10}, A_{11} and A_{21}). Thus, the expression created for this cell is: ($A_{10} \wedge A_{11} \wedge \neg A_{21}$) $\vee$($A_{10} \wedge A_{21} \wedge \neg A_{11}$) $\vee$($A_{21} \wedge A_{11} \wedge \neg A_{10}$)

The Knowledge Base is updated depending on the status of the current cell, if identified as a mine, its denotation in replaced by 'True' in the knowledge base. After this, it's converted to CNF, if required, and then appended to the knowledge base. After this, equations in the knowledge base are combined with it to form consistent assignments and further update the knowledge base.

Our program focuses on converting the generated MineSweeper puzzle to a 'Constraint Satisfaction Problem' and then solving it. The program focuses on drawing inferences out of a given clue and continuously updating the knowledge base by taking into account all the possible assignments and checking for consistency. However, there exist cases where the clue cannot exactly help in inferring what value to assign. In such uncertain cases, the next cell in the matrix is chosen randomly.

2.3 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?

The attribute 'prob' corresponding to every 'Box' and indicates the probability of every site (cell) having a mine. Once constraint satisfaction problem is solved within the 'updateKnowledgeBaseInfo' method, multiple models for the given CNF are generated via the 'satisfiable' function.
If in each of the models generated above, any given variable say, Aij i.e. any given cell location is always 'True', that particular box denoted by Box(i,j) is marked as a mine.
During exploration, for each box loaded onto the fringe, the ratio of the number of true values for a given symbol, over all the models generated, to the total number of models is calculated. This ratio helps us in making inference if mine is present at a given box. If the ratio is 1, then with 100% confidence, we say that the box is a mine. If not 1, we store that value in 'prob' attribute for that given Box.
When choosing the next box to be explored, the boxes present on the fringe are first sorted according to their probability values ('prob' values here). The box with the lowest probability is chosen as the next box to be explored.
An upper threshold of '0.2' is defined for the probability for exploration to happen. It is the maximum probability that any chosen box can have. This value is a result of various experiments which showed that the choice of 0.2 led to an improvement in the final score. If a case arises when no box on the fringe satisfy this condition, a random unsolved box is chosen and queried.

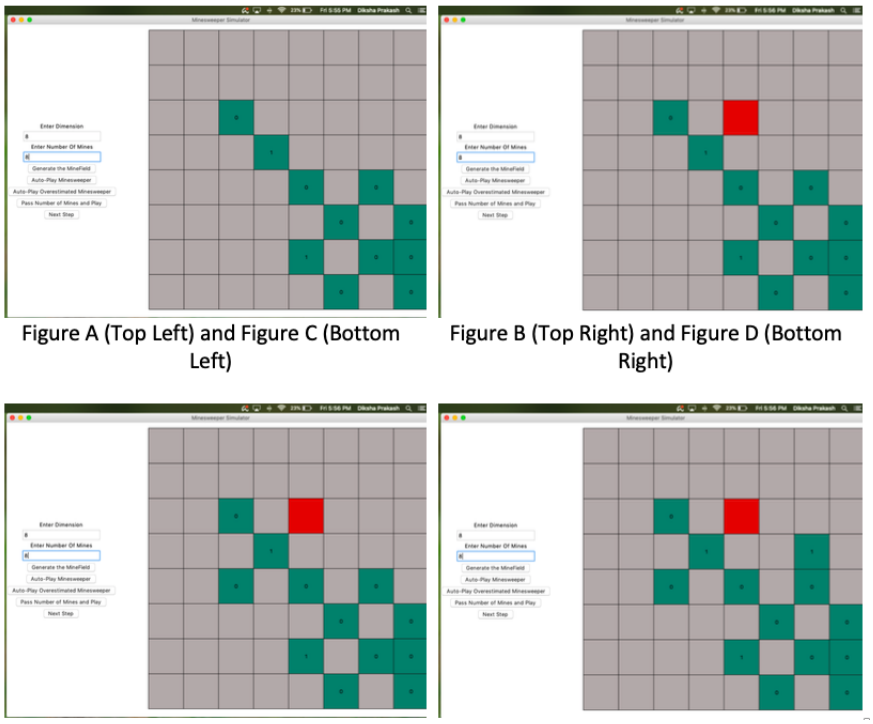

Figure 3: MineSweeper Performance Sequence Map

2.4 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?

At the Figure 'A' shown in Figure 3, I would have chosen any of the neighbors of A_{22} as the clue for it is zero i.e. all its neighbors are safe. However, the agent randomly chose A_{24}. This is due to the way the fringe has been designed.
For every box being visited by the agent, the fringe contains the new neighbors as well as the old neighbors from the previous fringe. The idea behind our fringe implementation is continuous addition of new neighbors to the fringe while retaining the old neighbors from past visits. The next move of the agent has been explained in depth in the 'Decision' sub-part of this question.
The reason for this approach is that we didn't want to miss out the case when the clue is 0 i.e. all the neighboring boxes are safe. If we happen to choose one of the neighboring boxes, the fringe will change, and the agent will lose the chance to select the other possible safe cells from the previous fringe.

2.5 Performance: For a xed, reasonable size of board, plot as a function of mine density the average nal score (safely identied mines / total mines). 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'?

For this we chose the board dimension to be 8x8 solved the board for different mine densities. The Final average score is obtained by taking mean of 10 random 8x8 boards

The board dimension has been taken to be 8 X 8 and solve the board for different mine densities. Score is defined as the ratio of the total number of identified mines to the total number of mines. The 'Final Score' in the graphs below have been defined by taking the mean of 10 randomly generated 8 X 8 boards.

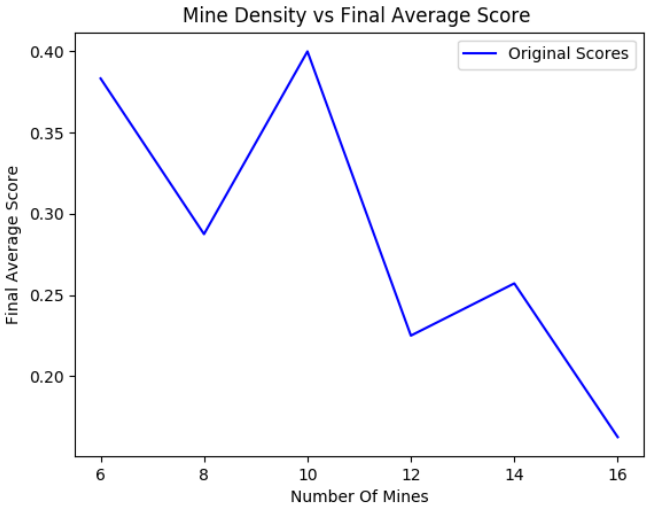

Figure 4: Mine Density vs. Final Average Score

It can be observed from the graph that the MineSweeper becomes very hard when the number of mine exceeds 12.

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

The idea implemented in our code is to convert the Minesweeper to a 'Constraint Satisfaction Problem'. For this every clue is converted to its equivalent CNF notation.
The complexity for solving this is as under:

1. Time Complexity: $O(2^n)$
2. Space Complexity: $O(n)$

Here, n refers to the number of variables in the knowledge base. This is a problem specific constraint.

2.7 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 modelled 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? Regenerate the plot of mine density vs expected final score, when utilizing this extra information.

In the case when the agent knows the total number of mines in the minefield, the 'prob' for each box is initially set to (number of mines)/(total number of boxes) i.e. the probability that any box is bound to have a mine.
This value changes as the agent progresses and uncovers the cells on the basis of the number of mines identified and the number of mines yet to be explored.
It can also be inferred from Figure 5 that hard mazes can now be solved when the number of mines is provided.

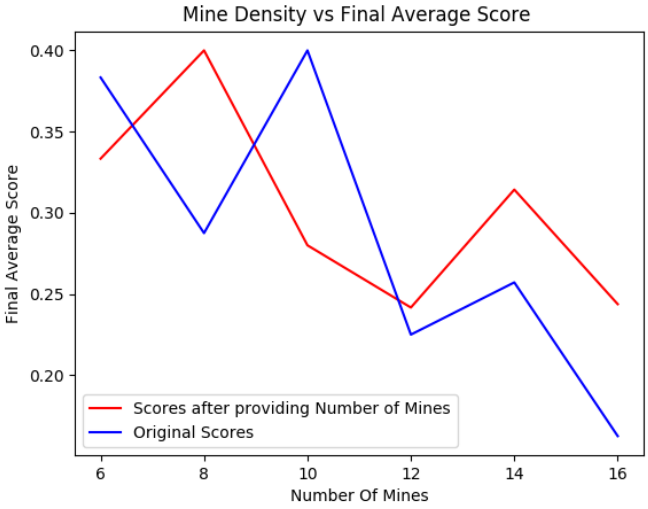

Figure 5: Mine Density vs. Final Average Score

3 BONUS : Minesweeper Agent with Overestimated Clues

We have implemented the solution for an overestimated clue. The environment will return a clue that is overestimated i.e. the clue maybe larger than the actual number of mines in the mine field generated.
For the implementation, we have created a class 'MineSweeperOverestimatedAgent' which inherits from the actual 'Agent' class. The primary difference lies in the 'addInfoToKnowledgeBase' function where the clues are converted to logical expressions.
The expression can be defined as under:
$N_L(clue) = \vee_{c=0}^{clue} P(c)$

where, $N_L(clue)$ states the presence of at least 'clue' number of mines in the neighborhood and $P(c)$ indicates the presence of exactly 'c' mines in the neighborhood.
Alternatively, the above equation can also be written as:
$N_L(clue) \equiv \neg(\neg N_L(clue)) \equiv \neg \vee_{c=clue+1}^{8} (\neg P(c))$