CS 440: Assignment 2 - Inference-Informed Action 16:198:440

Dyujoy Majumdar (dm1467)   Syed Rayyan Ali (sra118)

2021

Representation

How did you represent the board in your program, and how did you represent the information / knowledge that clue cells reveal? How could you represent inferred relationships between cells?

The board is represented by a 2d array. Every cell should have 2 types of information, whether it is safe or mine and the number of surrounding cells that have a mine. So the 2d array is integer type with every cell containing the number of neighboring mines. Now to distinguish between safe and mine cells, I chose the number 9 to represent a mine cell because:

- A mine cell doesn't reveal any information about its neighbors
- A safe cell can have maximum 8 neighbors

So in this way:

- Every cell that contains a number < 9 means that it is a safe cell and the integer indicates the number of neighboring mines
- The cell containing the number 9 means that it is a mine cell

Inference

When you collect a new clue, how do you model / process / compute the information you gain from it? In other words, 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?

Key :

- Newly founded safe cell = 3
- Newly founded mine cell = -2
- Unexplored Safe cell = 2
- Mine cell = 0
- Explored safe cell = 1

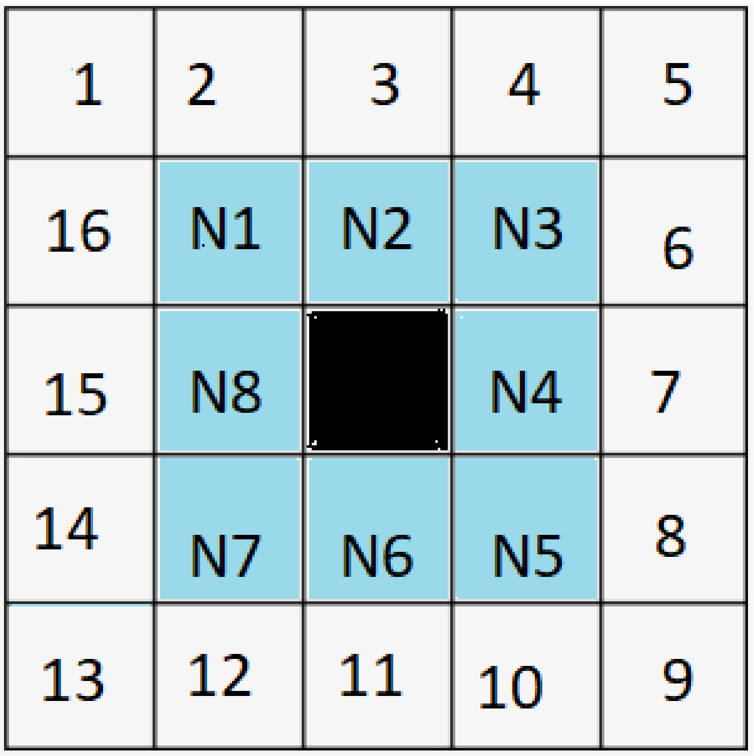
To model, process and compute information, I created a temporary board of the same size. The board was filled with -1s where a -1 indicated an unidentified cell. Whenever a cell was discovered, the -1 was replaced with:

- -2 if it is a mine
- 3 if it is a safe cell

This clue helps to make 3 types of deductions :

- If all hidden neighbors are found to be mines or safe cells.
  - In this case, these mines are converted to their appropriate values as described above

- Check all the clue cell's neighbors for any improvement. The idea is that this clue cell could be the (last safe cell) or (last mine cell) remaining in the neighborhood of the neighbors so that all other neighbors can be safely identified. Currently, the clue cell is newly founded so either -2 or 3. When all of its neighbors are checked, the value will be converted to 0 or 2 depending on if it is mine or safe respectively.
- When no hidden neighbor is conclusively identified to be safe or mine
  - This is where inference comes into play.



The intuition or the idea behind this inference is to minimize the region in which mines exist.

Consider this minesweeper board where black cell is explored and blue cells are the neighbors. If there is a case that (hidden neighbors - mines left) = 2, then it is a fact that every 3 neighbors will have at least 1 mine in them. Similarly, if (hidden neighbors - mines left) = 1 , then every 3 neighbors will have at least 2 mines in them and every 2 neighbors will have at least 1 mine in them.

But we don't care about every combination of 2 and 3, what we are looking for are the combinations which are common to another cell. If we look at the above three neighbors of the black cell, we can see that they are also neighbors of the Cell 3. Similarly, the left three neighbors are also neighbors of Cell 15 and below 3 are common to Cell 11 and right 3 to Cell 7. Therefore 4 combinations are made and this information is provided to the respective cells to deduce any information.

Similar to combinations of 3, 8 combinations of 2 can be made. N1 and N2 for cell 2, N2 and N3 for cell 4, etc.

The above information when provided to the neighboring cells is beneficial because now we have deduced that at least some mines are present in a specific region. So there can be many cases where only 1 or 2 mines for a cell are left and this information helps to clear out all the remaining neighbors.

Yes, my program deduces all the useful information from a clue because as I showed, there can be only 12 useful combinations made out of a clue and I am using all of them.

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?

Given the current state of the board, First I check for any newly founded safe cell or mine cell. If there is a newly founded safe cell (3), then its neighbors are explored for any improvement. The same goes for newly founded mine cells (-2). After checking for its neighbors, only the safe cell is then explored and converted to 1. If no newly founded cell exists, then check for any remaining unexplored safe cells and explore them first. If none of the above is true, then a random cell is picked to be explored.

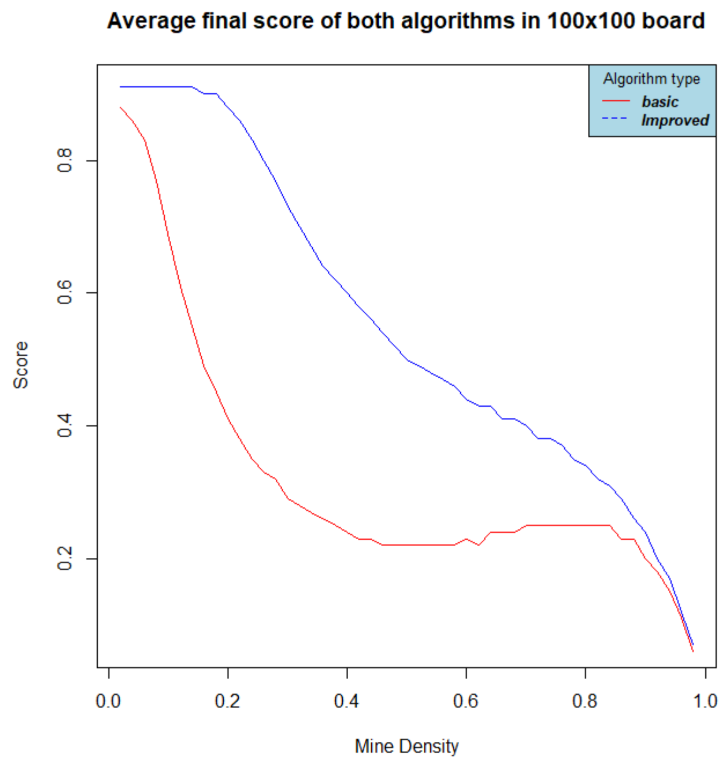
Our Algorithm : Minimize Mine Region (MMR)

Minimize the region of neighbors in which the mines exists

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? How frequently is your algorithm able to work out things that the basic agent cannot?



Does the graph make sense / agree with your intuition?

Yes, the graph does make sense because:

- The graph starts at the peak which shows that the average score is very high when there are less mines because both the algorithms have a much greater room to work with i.e. many safe cells.
- The graph gradually decreases because the mine density increases and therefore chances of clicking a mine cell randomly also increases.
- After mine density 0.50 till 0.8, the average score for the basic algorithm does not decrease. In Fact in addition to maintaining the score, the score also increases slightly.

    - This happens because when the mine density is high but not very high, there are a lot of mine cells and a safe cell has a greater chance of having 8 mine cells which will reveal all of them when we explore the cell.

- After mine density 0.8, the graph goes to a downward hill because now there are too many mine cells and a very high chance of clicking them.

When does minesweeper become 'hard' ?

Minesweeper becomes hard with increasing mine densities because of the increased chance of clicking a mine cell.

When does your algorithm beat the simple algorithm, and when is the simple algorithm better?

My algorithm is always better than the simple algorithm as seen by the graph that the average final score of the improved algorithm is constantly better or equal to the basic algorithm. The simple algorithm is only better when either mine density is really low or very high because of randomness. For example:

- Suppose there are only 2 mines out of 1000. It is possible that the basic algorithm safely catches these 2 mines by hitting a "good" safe cell randomly whereas the improved algorithm clicks these 2 cells randomly at beginning when there is nothing to infer.
- Suppose there are 990 mines out of 1000 and all 10 cells have 8 mines around them. Now, it is entirely possible that the basic algorithm hits the safe cell before any of its neighboring mine is clicked. In this way, all 8 mines are safely discovered and in case of improved algorithms, maybe we click 2 or 3 mines at random(when nothing to infer and inference basically fails at such high mine density) before exploring the safe cell and in this case only 5 or 6 neighboring mines are safely discovered.
- To summarize, the basic algorithm only beats my algorithm in the aspect of randomness.

How frequently is your algorithm able to work out things that the basic agent cannot?

The inference and revisiting safe cells mechanisms in my algorithm are able to work out things very often as compared to the basic algorithm

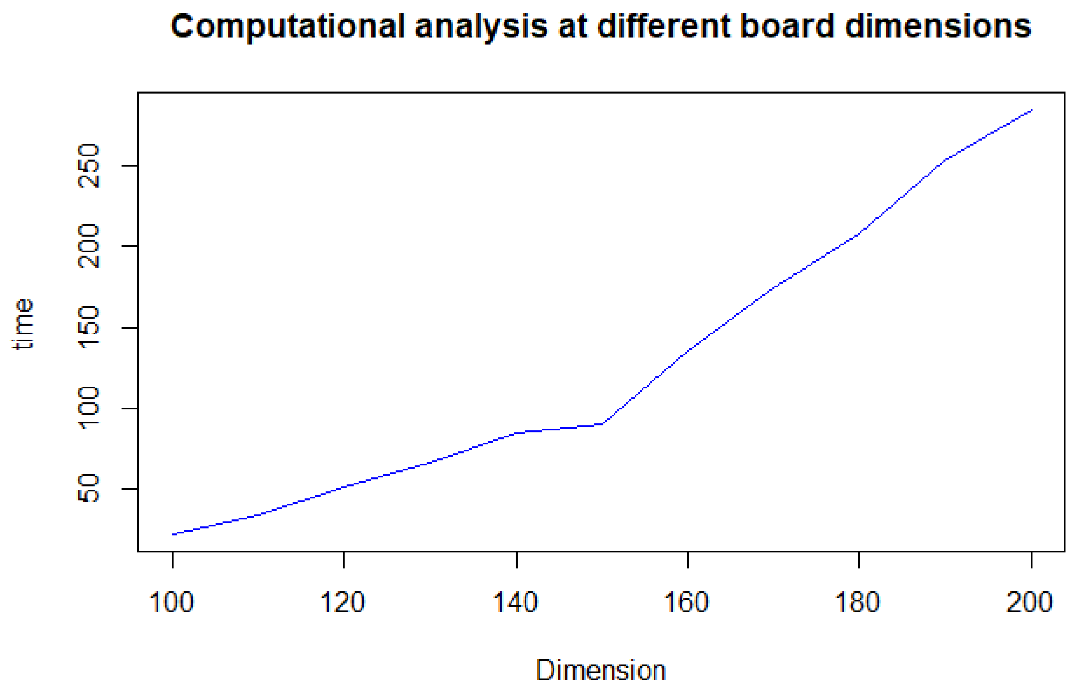
Efficiency

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?

Some of the time constraints involved in this program include:

- Picking a cell randomly : The way I implemented the program with a temporary board, randomly picking an unidentified cell can be time consuming. This is because the program randomly generates cell coordinates and checks whether it is unidentified. In large mazes when there are only a few unidentified cells remaining, it will take a long time for the program to pick the unidentified cell randomly. It can be improved by storing the remaining cells in an array and then randomly selecting a cell. But this method will increase space consumption because we will have to create an array equal to the number of cells i.e, Dimension*Dimension

- The following graph shows that the time consumption is increasing at a fast pace with increasing dimension of the board



There are no space constraints in my program because there is no space consumption except for the temporary board that I am creating and the original minesweeper board. The space consumption is random and not affected by the board dimension but only the structure of the board.

For a reasonably-sized board and a reasonable number of mines, include a play-by-play progres- sion 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?

The following gif shows the play by play progress of the improved algorithm in a 5x5 board with 10 mines.
Key :

Cell type	Color
Unidentified	Black
Newly founded mine cell	Maroon
Newly founded safe cell	Brown
Mine cell	Red
Unexplored safe cell	Green
Explored safe cell	Blue

Note : If colors changed simultaneously, it means that the cell was deduced from the current cell. Otherwise if a black color changes without any other color at the same time, then it was discovered on random picking(when there was no more information)

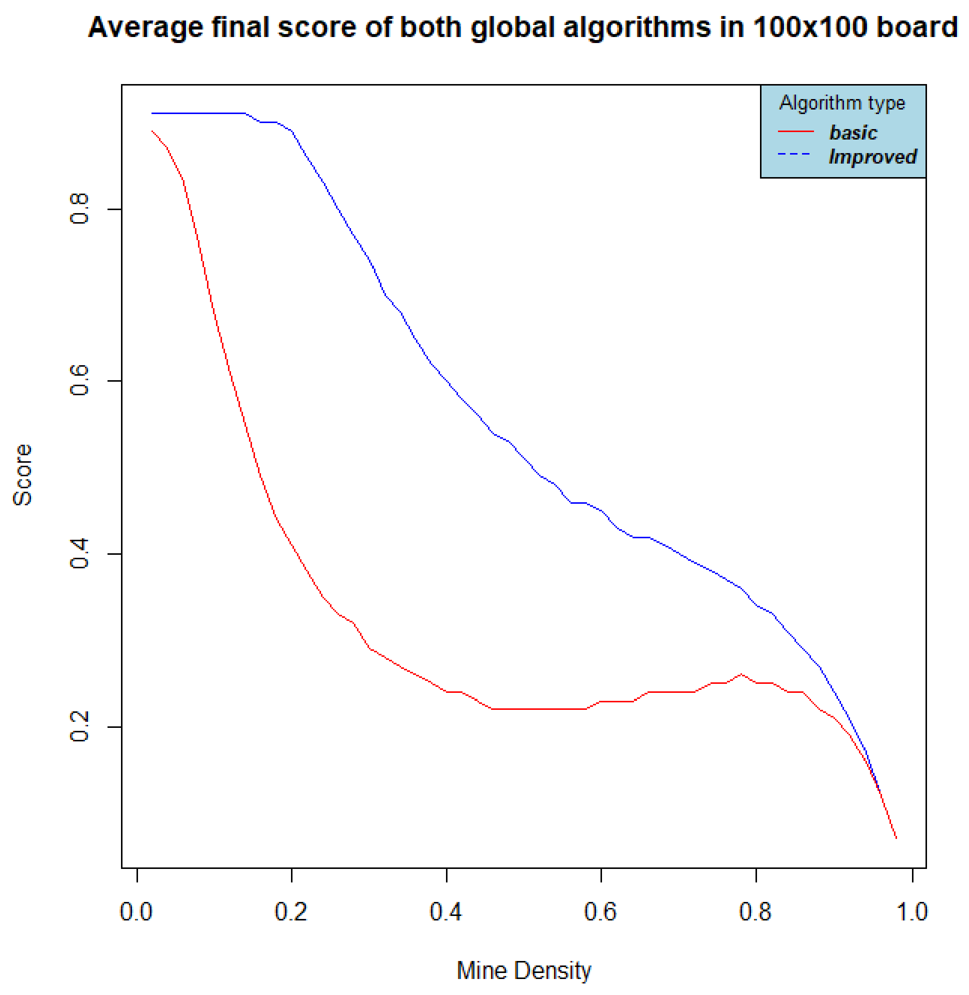
[The animation is present in another file.]

Global Information

If the total mines are already known, then it can be useful by the following model

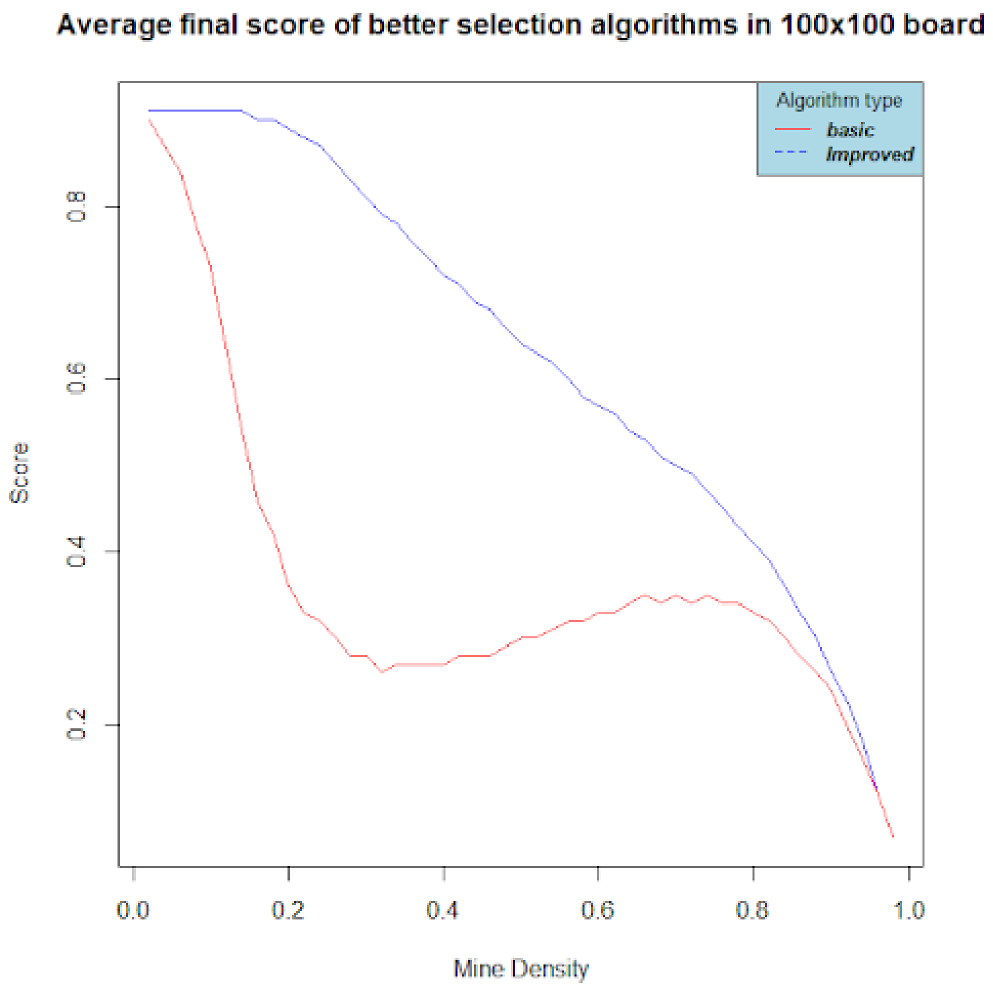
- Find the total number of safe cells i.e., total cells - mines
- Keep track of the total mines and safe cells
- Whenever you see that all mines are identified, then the rest are safe cells
- Whenever you see that all safe cells are identified, then the rest are mines
- It greatly depends on randomness and how quickly all mines or safe cells get identified.

The following graph shows the score of both algorithms with the extra knowledge of knowing the total number of mines. As you can see, there is only slight betterment when global information is known and it all depends on randomness because the agent might randomly pick "good" cells first. However, when the mine density is 1.0, global information has the highest score of 1 whereas without global information, this score would be 0. Another advantage of global information is that once you find out that all mines or all safes are identified, you don't need to go through the whole board which saves a lot of computational time and space.



Better Selection

The selection method that I used is based on probability. Whenever a new cell has to be clicked, I go through the whole board and check which cell has the least probability of having a mine. This cell is then selected to be clicked. The justification is that this cell has the least chance of having a mine out of all the remaining cells.
The following graph shows how this method helped to increase the average score of both the algorithms:


Average final score of better selection algorithms in 100x100 board
Algorithm type
basic
Improved
Score
Mine Density