CS 440: Assignment 2 - Inference-Informed Action

16:198:440

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: MineSweeper

In the game minesweeper, you are presented with a square grid landscape of cells. Hidden in some of the cells are 'mines'. At every turn, you may select a cell to uncover. At this point, one of two things will happen: if there is a mine at that location, it explodes and you lose the game; if there is not a mine at that location, it reveals a number, indicating the number of adjacent cells where there are mines. If the cell reveals 0, all the surrounding 8 cells are empty of mines. If the cell reveals 8, all 8 adjacent cells must have mines. For any value in between, for instance 4, you know that half of the adjacent cells have mines, but you cannot be sure by this clue alone which half are dangerous and which half are safe. *Note: Common implementations of MineSweeper have other features, for instance instantly opening cells that are obviously clear to save you some time. We are ignoring that for the purpose of this assignment, as well as making some other small changes. Read carefully.*

The goal of the game is to identify the locations of all the mines (if possible); by collecting clues and information, you can begin to infer which cells are dangerous and which are safe, and use the safe cells to collect more information. This process is iterated until, hopefully, all cells are either uncovered, marked as clear, or marked as mined.

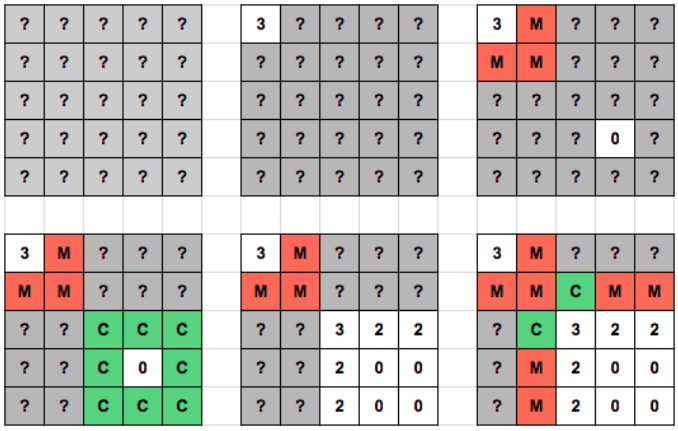

Figure 1: In the first board, all cells are unknown. The first move reveals a 3, in which case we may infer that all three surrounding cells have mines. We mark them as such, and know not to ever search those cells. The second move reveals a 0, in which case we know that all 8 surrounding cells must be clear. We mark them as clear, and know that we can search them without worry. Collecting data from the clear cells, we are able to infer the locations of more mines.

The goal of this project is to write a program to play MineSweeper - that is, a program capable of sequentially deciding what cells to check, and using the resulting information to direct future action.

2 Program Specification

There should effectively be two parts to your program, the **environent** representing the board and where the mines are located, and the **agent**. When the agent queries a location in the environment, the environment reports whether or not there was a mine there, and if not, how many of the surrounding cells are mines. The agent should maintain a knowledge base containing the information gained from querying the environment, and should not only be able to update its knowledge base based on new information, but also be able to perform inferences on that information and generate new information.

- The environment should take a dimension d and a number of mines n and generate a random $d \times d$ boards containing n mines. The agent will not have direct access to this location information, but will know the size of the board. *Note: It may be useful to have a version of the agent that allows for manual input, that can accept clues and feed you directions as you play an actual game of minesweeper in a separate window.*
- In every round, the agent should assess its knowledge base, and decide what cell in the environment to query.
- In responding to a query, the environment should specify whether or not there was a mine there, and if not, how many surrounding cells have mines.
- The agent should take this clue, add it to its knowledge base, and perform any relevant inference or deductions to learn more about the environment. If the agent is able to determine that a cell has a mine, it should flag or mark it, and never query that cell. If the agent can determine a cell is safe, it's reasonable to query that cell in the next round.
- Traditionally, the game ends whenever the agent queries a cell with a mine in it - a final score being assessed in terms of number of mines safely identified.
- However, extend your agent in the following way: if it queries a mine cell, the mine goes off, but the agent can continue, using the fact that a mine was discovered there to update its knowledge base (but not receiving a clue about surrounding cells). In this way the game can continue until the entire board is revealed - a final score being assessed in terms of number of mines safely identified out of the total number of mines.

This last modification allows the game to 'keep going' and avoids the situation where you accidentally find a mine early in the game and terminate before the game gets interesting.

You may either do a GUI or text based interface - the important thing for the purpose of the project is the representation and manipulation of knowledge about the mine field. This will draw on the material discussed in a) Search, b) Constraint-Satisfaction Problems, and c) Logic and Satisfiability. You must implement a basic MineSweeper agent, described in the next section, and your own agent as an improvement on the basic one.

As usual, you must create your own code for solving the main issues and algorithmic problems in this assignment, but you are welcome to use external libraries for things like visualizations, etc.

2.1 A Basic Agent Algorithm for Comparison

Implement the following simple agent as a baseline strategy to compare against your own:

- For each cell, keep track of
  - whether or not it is a mine or safe (or currently covered)

  - if safe, the number of mines surrounding it indicated by the clue
  - the number of safe squares identified around it
  - the number of mines identified around it.
  - the number of hidden squares around it.

- If, for a given cell, the total number of mines (the clue) minus the number of revealed mines is the number of hidden neighbors, every hidden neighbor is a mine.
- If, for a given cell, the total number of safe neighbors (8 - clue) minus the number of revealed safe neighbors is the number of hidden neighbors, every hidden neighbor is safe.
- If a cell is identified as safe, reveal it and update your information.
- If a cell is identified as a mine, mark it and update your information.
- If no hidden cell can be conclusively identified as a mine or safe, pick a cell to reveal at random.

2.2 An Improved Agent

The algorithm described for the basic agent is a weak inference algorithm based entirely on local data and comparisons - it is effectively looking at a single clue at a time and determining what can be conclusively said about the state of the board. This is useful, and should be quite effective in a lot of situations. But not every situation - frequently multiple clues will interact in such a way to reveal more information when taken together.

Your improved agent should model the knowledge available, and use methods of inference to combine multiple clues to draw conclusions when possible or necessary. Note that 'knowledge' to model includes potentially: a) whether or not the square has been revealed, b) whether or not a revealed cell is a mine or safe, c) the clue number for a revealed safe cell, and d) inferred relationships between cells.

3 Questions and Writeup

Answer the following questions about the design choices you made in implementing this program, both from a representational and algorithmic perspective.

- 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?
- 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?
- 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? Aside from always opening cells that are known to be safe, you could either a) open cells with the lowest probability of being a mine (*be careful - how would you compute this probability?* or b) open cells that provide the most information about the remaining board (*what could this mean, mathematically?*). Be clear and precise about your decision mechanism and how you implemented it. Are there any risks you face here, and how do you account for them?

- 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?
- 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?
- 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?
- 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.

4 Bonus: Dealing with Uncertainty

Suppose that your mine detector has a certain false negative rate ($p_{\mathbf{neg}}$) at which it mistakenly reports a mined cell to be empty. It additionally has a false positive rate ($p_{\mathbf{pos}}$) at which it mistakenly reports an empty cell to be mined. This means that you cannot always trust the clue number / mine counts to be accurate.

How could you adapt your solve to each of the following cases (separately)? Try to implement at least one, and experiment with what kind of performance hit results, in terms of your ability to safely clear the board.

i) You know in advance $p_{\mathbf{neg}} = 0, p_{\mathbf{pos}} = 0.2$.

ii) You know in advance $p_{\mathbf{neg}} = 0.2, p_{\mathbf{pos}} = 0$.

ii) You know in advance $p_{\mathbf{neg}} = 0.2, p_{\mathbf{pos}} = 0.2$.