Minesweeper Solver – CSP Based Agent

Abhineet Sharma, Ashwin Kallingal Joshy, Shubham Agrawal

Abstract

This report describes Minesweeper solver – A CSP based agent that solves the famous Minesweeper problem using the constraint satisfaction approach. It discusses two different approaches we followed in the attempt to solve the problem. Firstly, it describes an attempt at the problem using logic based approach. Then it describes the working approach based on a constraint satisfaction problem and the agent design that solves the problem.

INTRODUCTION

Minesweeper is a single-player puzzle video game. The objective of the game is to clear a rectangular board containing hidden "mines" without detonating any of them, with help from clues about the number of neighboring mines in each field. The game originates from the 1960s, and has been written for many computing platforms in use today. It has many variations and offshoots.

The player is initially presented with a grid of undifferentiated squares. Some randomly selected squares, unknown to the player, are designated to contain mines. Typically, the size of the grid and the number of mines are set in advance by the user, either by entering the numbers or selecting from defined skill levels, depending on the implementation.

The game is played by revealing squares of the grid by clicking or otherwise indicating each square. If a square containing a mine is opened, the player loses the game. If no mine is present in an opened square, a digit is displayed in the square indicating how many adjacent squares contain mines; if no mines are adjacent, the square becomes blank. The player uses this information to deduce the contents of other squares, and may either safely reveal each square or mark the square as containing a mine.

NP-Complete

Although minesweeper looks like a simple problem, Richard Kaye has proved that minesweeper consistency problem is in a complexity class of problems known as NP− complete problems. A common NP−complete problem has two characteristic features: they are computable by a non−deterministic Turing machine in polynomial time, and every other NP−complete problem can be reduced in deterministic polynomial time to the particular NP−complete problem in question. The former characteristic makes NP−complete problems difficult to solve computationally, at least in practice. Theoretically, NP problems may be as easy to solve as problems solvable in polynomial time by a deterministic Turing machine, but no one has managed to prove this, if it is even possible. The latter characteristic of NP−complete problems means that if a way is ever found to compute one NP−complete problem in polynomial time on a deterministic Turing machine, all NP−complete problems would then be solvable in polynomial time on a deterministic Turing machine. There are many well−known and important problems that are NP−complete. The problem of satisfiability of Boolean formula and the "traveling salesman" problem are two such examples.

SETUP

Minesweeper solver is a java based application and consists of 3 parts – Problem Manager, Solving agent and a user interface. Problem manager is responsible for generating a random problem based on an input of no of rows, no of columns and the difficulty level. This application has 3 difficulty levels – Naive, Medium and Expert. Difficulty levels decide the number of mines. If a user is playing at expert level, 20% of the squares contains mines. For Medium and naive level the percentage is 15% and 10% respectively. Problem Manager uses a random function to randomly allocate mines to various squares. Random function generates a random number following a uniform distribution i.e. it generates a number within a range with 50% probability. Problem Manager is also responsible for updating the model after every move requested by the solver and respond back with the updated model state. During the game, the entire game state is visible on the designed user interface, which will be discussed in the following sections of the report.

In this setup, it is assumed that the first selected square never contains a mine. Also, because of the calculated guess work involved, game is not stopped when a square containing a mine is selected, rather a count is maintained for performance evaluation purpose. The performance analysis of the Minesweeper solver agent would follow in the Runtime Data analysis section.

APPROACH

This report talks about two approaches, we followed during the project work on Minesweeper solver agent. Unfortunately, we could not follow through our work on logic based approach due to time constraints. This section discusses a basic idea of the logic based approach we had followed. Then a detailed description of the Constraint satisfaction based approach for solving the minesweeper problem is discussed.

Logic Based Approach

The idea was to make a knowledge base using Prolog and use logical inference models to deduce which squares may or may not contain mine. The approach includes updating the knowledge base after every move, following feedback from Problem Manager and make inferences for the rest of the mine field.

The designed **algorithm for logic based approach** is as follows:

1. Start with a random square as the first square would never contain a mine, as per the setup of the game.
2. While **Game in Play**, do
    - **a. TELL –** Logic based module about the updated model of the environment following the feedback from Problem Manager.
    - **b. ASK –** Get the list of possible moves from the logic based module.
    - c. If **No Possible Move**
        - i. Make a random guess
    - d. Else, **Select a move**
    - **e. TELL –** Update the knowledge base with the selected move, for further inference.
    - f. Inform Problem Manager about the move and update the Model.

Due to time constraints, we were unable to build a working knowledge base, to be used with the Prolog module. Hence, we had to move onto another workable approach, using Constraint Satisfaction Problems.

The algorithm we have designed for Constraint Satisfaction Problem can be further improved to make a calculated guess instead of a random guess based on different approaches. During the project work, we came across a couple of approaches which have been discussed in the 'Making a calculated Guess' section of the report with the necessary and appropriate citations of the research papers.

Minesweeper as Constraint Satisfaction Problem

Minesweeper problem can also be viewed as a Constraint Satisfaction Problem. To fully understand the concept, let us consider a simple configuration:

0	**0**	**E**	
1	**1**	**D**	
A	**B**	**C**	

Figure 1

This configuration results in the following two equations:
For Green Square [0, 1]:

A + B = 1

Either square A or square B could have a mine.
Similarly, for Yellow Square [0, 2], we have:

A + B + C + D + E = 1

Either of Squares A, B, C, D or E could have a mine.

Solving the system of equations, we get two possible solutions:
Solution 1: A = 1, B = C = D = E = 0
Solution 2: B = 1, A = C = D = E = 0

Clearly, both the solutions, result in ambiguous solutions for A & B as we cannot decide for sure for those squares. But, there is a clear solution for squares C, D & E, which says that if the given configuration is true, there is no possibility of squares C, D & E containing a mine. This can be easily proved using Contradiction, which would result in the Yellow square [0, 2] having a number > 1, had any one of the other squares contained a mine.

Let us formalize this approach:

Variables: All the unexplored squares that are neighbors of explored squares, form the variables of the constraint satisfaction problem. The reason for such a selection is trivial, as there is no information for the rest of the squares and including them in the problem would simply raise the overhead without yielding any useful result.

Domain: Each variable has a domain {0, 1}. 0 implying the absence of mine and 1 implying the presence of one.

Constraints: Constraints for the problem is formed using the value of number of neighboring mines in the explored square.
Consider a square $[i, j]$, $S_{i,j}$ with number of mines N_{ij}.
$\mathbf{S_{i-1,j-1} + S_{i-1,j} + S_{i-1,j+1} + S_{i,j-1} + S_{i,j+1} + S_{i+1,j-1} + S_{i+1,j} + S_{i+1,j+1} = N_{ij}}$

The constraints include only the valid neighbors, for example corner squares may have less neighbors as compared to center squares. Similarly, squares with more opened squares will have lesser number of constraints.

For every move, we can formulate a CSP and use a tool to solve the same. This agent makes use of **JaCoP** [**Ja**va **Co**nstraint **P**rogrammer] tool to solve the CSPs. This is discussed in the Code section.

Algorithm for CSP Based Approach

1. Start with a random guess.
2. While, **Game in Play**, do
    a. Formulate a CSP, using the current state of the board.
    b. Solve the CSP using JaCoP.
    c. Filter out all the variables with ambiguous solutions.
    d. If, any variables remain,
        i. For the remaining variables, do
            1. If value = 0, mark it safe.
            2. Else, mark the square as mine.
    e. Else,
        i. Make a random guess.
    f. Inform Problem Manager to update the model.

MAKING THE GUESS

There are various approaches for making the guess. This section talks about all the approaches we were planning to incorporate for making guesses. Currently, the solving agent makes use of the Random Guess Approach. The idea was to implement various different approaches and analyze the performance of each for a fixed test set.

Approach I: Random Guess

This approach is a very naïve approach. Select a square randomly considering a 50% probability of each square containing a mine. This assumption is true in case of an unopened board but when the first square is opened and the model updated, the probability of each square containing the mine does not remain the same.

Approach II: Guess based on Density

Although, the agent does not implement this approach, we would like to discuss the idea here. The idea is to make a guess based on density of mines around a square. For example, a square around which a lot of neighbors have been explored is more likely to contain the mine as compared to a square where less neighbors have been explored. This idea can also be extended to include the number of mines hinted by the explored neighbor. Intuitively, if a neighbor indicates a higher number of mines around, then the square having such a neighbor is more likely to be a potential for a mine. The idea was to extend this idea to a mathematical formula which would work as a heuristic to make a calculated guess.

PROGRAM DESIGN

This section briefly describes the design used for the software.

Overall Design

The entire program consists of 3 parts:

- Problem Manager
- Minesweeping Agent
- User Interface

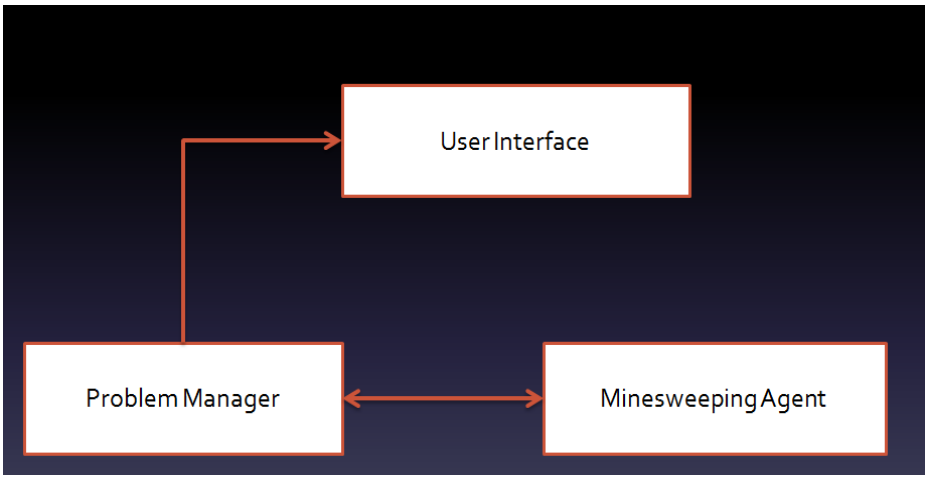

Figure 2

Timeline View

The following diagram presents the timeline view of the agent, as per the algorithms presented in the previous sections.

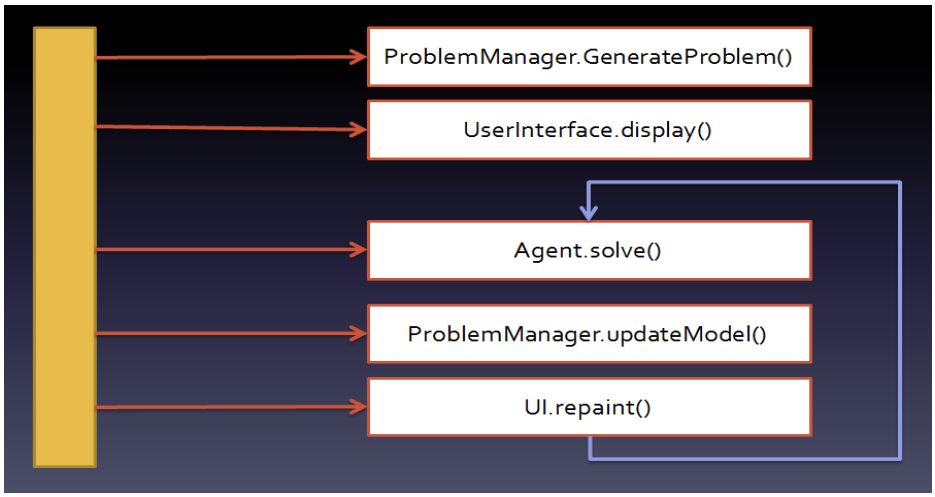

Figure 3

MINESWEEPER IN ACTION

Following is a screenshot of Minesweeper in action.

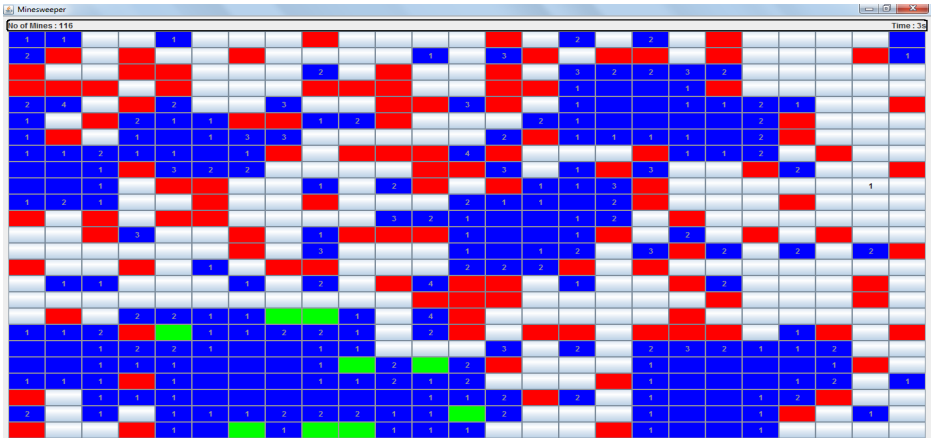

Figure 4

Here, Blue squares indicate the squares identified by solving agents to be safe. Green squares the ones identified as mines. Red Squares are the squares that contain mine but not yet identified by the agent.

RUNTIME DATA

Following graph shows the Guess count data collected from the software run for various configurations at different levels.

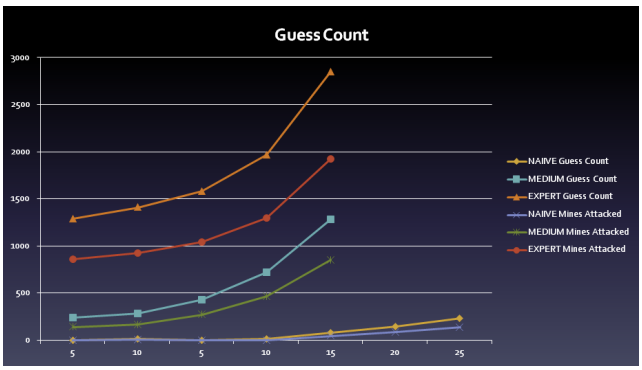

Figure 5

Following graph, shows the time taken to solve the problems of different sizes at different levels.

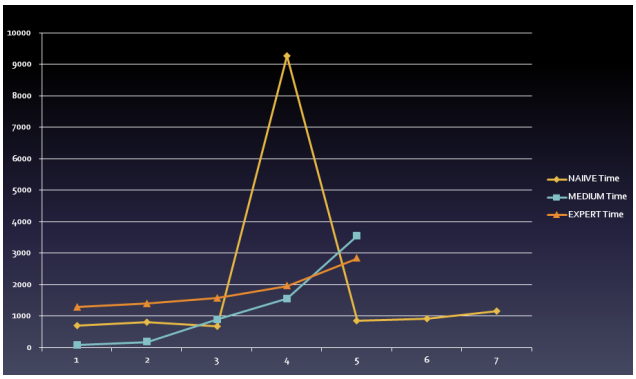

Figure 6

The spike in the middle corresponds to spike in time caused by some other process running on the Operating system during the test run.

CONCLUSION

We were able to implement a Minesweeper solver agent successfully using the Constraint Satisfaction Problem approach. The Minesweeper solver we have implemented have a few restriction that prevents it from being better than humans. At best, the agent, in its current form, is only as good as a human. This drawback arises from the lack of implementation of probabilistic guessing. If such a guessing algorithm is incorporated into the agent, it would be able to perform exponentially better. In the current form, the agent never detonates a mine, if it is not making a guess. This means that we were able to reduce the minesweeper game into a CSP problem and make correct inferences from it. Due to the randomness of each problem generated we are not able to put a number to the success-rate for our agent, but the graphs in the Run time Data section would give a good idea of how close our implementation is to a perfect solution.

References

- Chris Studholme. Minesweeper as a Constraint Satisfaction Problem.
- Richard Kaye. Minesweeper is NP−Complete. Mathematical Intelligencer , volume 22 number 2, pages 9−15, 2000.
- Wikipedia, for Game description
- Kasper Pedersen. The complexity of Minesweeper and strategies for game playing; University of Warwick, 2003-2004.