Solving Minesweeper Using Constraint Satisfaction

Team 13

1 Introduction

I will create an algorithm to attempt to solve minesweeper as a constraint based problem. Minesweeper is useful because the problem is NP-complete and demonstrates the Constraint Satisfaction Problem algorithms ability to solve this class of problems. Additionally, it reinforces the class concepts of CSP, and fog-of-war. Exploring fog-of-war problems is useful for robot navigation through extremely hazardous terrain with limited or less than perfect sensor input.

2 Related Works

Another researcher who has explored this problem is Chris Studholme. He elaborates that while all games of Minesweeper are consistent (winnable), an algorithm may have to make guesses to reach this state. Studholme makes several suggestions about guessing strategies and starting moves that I intend to incorporate into my strategy.

Bayer, Snider, and Choueiry, authors of [2], propose the same variable assignments and constraints as Studholme, but fail to elaborate on any specific methods they used. I would need to include appropriate backtracking and decision making when guesses must be made.

3 Implementation

3.1 Overview

My strategy was implemented within Programmer's Minesweeper (PGMS), a Java framework developed by John Ramsdell. It incorporates several other strategies that allowed for comparison against the constraint based strategy I was developing.

Below I describe the constraint satisfaction problem. Each cell within the gameboard is considered a variable. Each variable has a boolean domain determining whether that cell is a mine or not. All non-mine cells have a single constraint that dictates the value that the cell holds must be equal to the sum of the mines located in the surrounded cells.

$$\begin{aligned} &variables\ X = && \{x_{i,j} \mid i \ \epsilon\ 1...columns,\ j \ \epsilon\ 1...rows\} \\ &domain\ D = && \{0,1\}, Value(X) = \{0...8\} \\ &constraints\ C = && \{x_{i,j} = 1 \vee \\ & && Value(x_{i,j}) = \sum(NeighboringCells)\} \end{aligned}$$

Next I created two classes. One performed the bulk of the logic and contained the main game loop inherited from PGMS. The second class represented the constraints for each cell of the minesweeper game. It contains an enumeration(1) of what its type is, a value for how many mines this cell is touching, and an ArrayList that links to the 8 neighboring cells which acts as a constraint.

$$enum\{UNKNOWN, CLEAR, MINE\} \qquad (1)$$

3.2 Algorithm Steps

During each step in the game loop a series of steps were performed. These reduced the complexity of constraints and found the next best move.

1. Each cell checks its list of constraints for changes from the last move. If a mine is discovered, that variable can be removed from the constraint $Value(x_{i,j})$ modified. This reduces constraints that need to be checked in later steps.

$$\begin{array}{l} Value(x_{0,0}) = 2 = x_{0,1} + x_{1,0} + x_{1,1} \\ x_{0,1} = 1 \ //Discovered\ to\ be\ a\ mine \\ \hline Value(x_{0,0}) = 1 = x_{1,0} + x_{1,1} \end{array}$$

2. Reduce the complexity of constraints by comparing with the constraints of neighbors.

$$Value(x_{0,3}) = 2 = x_{0,2} + x_{1,2} + x_{1,3} + x_{1,4}$$
$$Value(x_{0,4}) = 1 = x_{1,3} + x_{1,4}$$

$$Value(x_{0,3}) = 1 = x_{0,2} + x_{1,2}$$

3. Look for cells where $Value(x_(i,j)) = \sum NeighboringCells_{UNKNOWN}$. All of these neighbors may be marked as $MINE$s.

4. Look for cells where $Value(x_(i,j))$ has been reduced to 0 through steps 1 and 2. All neighboring cells may be marked as $CLEAR$.

5. Repeat steps 1 through 5 while cells are still being marked $CLEAR$.

6. Perform a recursive backtracking algorithm to discover all solutions to the current state. The cell that is $CLEAR$ in the most scenarios is determined to be the best move.

7. If the game has not ended, return to step 1.

3.3 Backtrack

My implementation uses a recursive backtracking method to discover all possible boards from a given state. To reduce the complexity, certain optimizations are made. Only cells that are $UNKNOWN$ and are adjacent to $CLEAR$ cells are considered for branching, which eliminates the majority of most boards.

For each of the valid cells, the method sets the cell to $CLEAR$ and recurses into the next cell, sets the cell equal to $MINE$ and recurses into the next cell, and returns the cell to $UNKNOWN$.

The base case for the recursive method checks to see when all cells have their constraints satisfied, or any single constraint is invalid. At which point, all $CLEAR$ cells in valid solutions increment a counter for their position in a 2-D array and return.

4 Results

I consider the results adequate, but not reaching the level of performance I had hoped for when initially proposing the project. As seen in Table 1, the win rate of my strategy is between 5%-10% lower than the EqnStrategy provided in PGMS. Additionally, when the branching factor is high enough, normally performing the CPS with greater than 40 cells, the runtime of CSPStrategy appears to be signficantly worse.Given more time, I would like to study the ratio of cells observed to time taken and discover where it becomes less optimal than EqnStrategy.

Looking at Figures 1, 2, and 3, we can see that the majority of game losing moves occur in the first few moves, or when the game is almost won. This is the case for both my CSPStrategy, and the rival EqnStrategy.

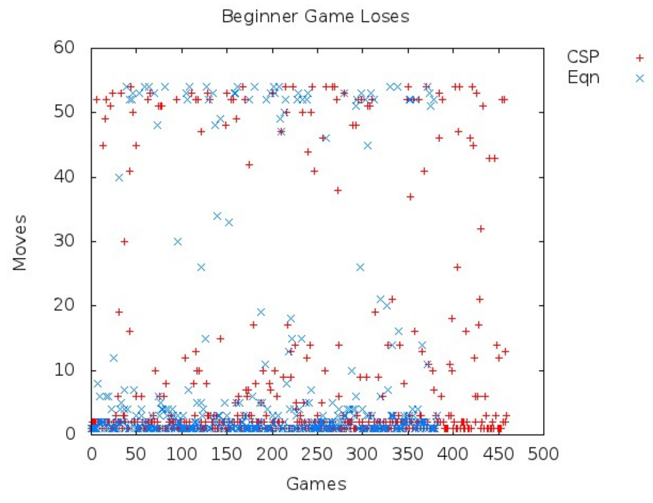

Figure 1: Shows the how many moves a lose took in a beginner game

5 Evaluation

I believe these results show that information is crucial to an solving a CSP efficiently. The beginning of a game can often hold very little information about the state of the board. Likewise, at the end of a game of minesweeper there are often groupings of cells that have relatively equal odds of being mines.

	Win Percentages		
Strategy	Beginner	Intermediate	Expert
EqnStrategy	73%	38%	24%
CSPStrategy	64%	33%	19%
SinglePointStrategy	38%	3%	1%

Table 1: Comparison of the win rates of various strategies at the three difficulty levels

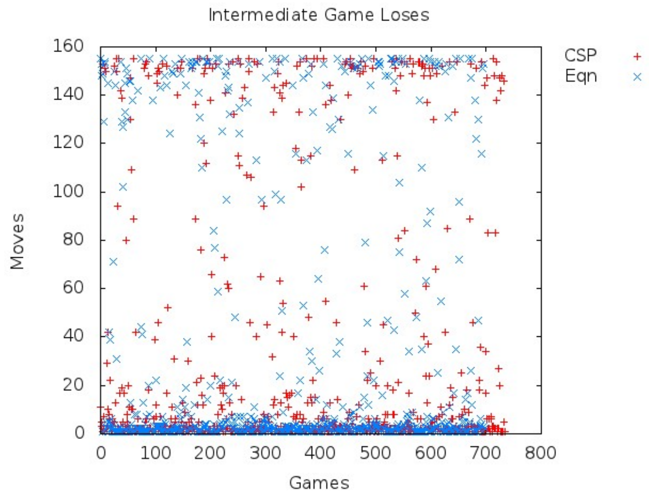

Figure 2: Shows the how many moves a lose took in an intermediate game

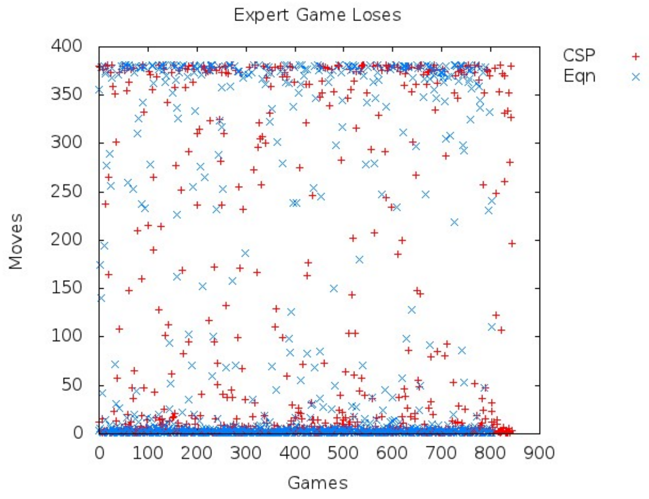

Figure 3: Shows the how many moves a lose took in an expert game

This is why there are relatively few loses in the middle of the games in Figure 1, 2, and 3. During those sections my CSPStrategy has a wealth of information to discover the positions of mines and determine clear spaces with a high probability. It's mainly at the beginning where it is acting blind, or at the end where several cells are equally likely to hold mines, that loses occur.

This is where my CSP strategy differs from Studholme[1]. When presented with these situations I continue to choose the cells that are most likely to be $CLEAR$. Studholme chooses to blindy choose a new cell elsewhere on the board in these situations that he refers to as "crap shoots". As it turns out, he choose a more optimal strategy. While my strategy has a 50% chance to guess the right mine in one area, his has:

$$\frac{RemainingCells - MinesLeft}{RemainingCells} \tag{2}$$

As long as $RemainingCells > MinesLeft$ his strategy will perform better than mine under those circumstances.

6 References

[1] Chris Studholme. 2001. Minesweeper as a Constraint Satisfaction Problem [Online] Available: http://www.cs.us.es/cursos/ia1-2007/trabajos/trabajo-2/minesweeper-toronto.pdf

[2] Ken Bayer, Josh Snyder, and Berthe Y. Choueiry. 2006 . An interactive constraint-based approach to minesweeper [Online]. Available: https://www.aaai.org/Papers/AAAI/2006/AAAI06-344.pdf