Minesweeper Solver

Oleksandr Prusakovskyy Simone Cargnin

December 18, 2022

1 Introduction

Minesweeper is a single-player puzzle game in which the player must locate and mark all the mines on a grid of squares. Each square may contain a mine, or it may be empty. The player is given some initial clues about the locations of the mines, in the form of numbers indicating the number of mines in the surrounding eight squares. The goal of the game is to mark all the mines while avoiding setting off any of them.

2 Representation of the Game

In order to solve Minesweeper with CSPs, we must first represent the game in a form that can be processed by a CSP solver. There are several ways to do this, but one common approach is to use a binary representation, in which each square in the grid is represented by a binary variable that indicates whether or not it contains a mine.

For example, consider the following Minesweeper grid:

```
? 1 1
1 2 *
0 1 1
```

In this grid, the squares containing mines are indicated by an asterisk (*). The other squares contain clues in the form of numbers indicating the number of mines in the surrounding eight squares.

To represent this grid in a CSP, we could create a binary variable for each square, with a value of 1 indicating that the square contains a mine and a value of 0 indicating that it does not. For example, the binary variables for the first row of the grid might be represented as follows:

x_1 x_2 x_3

We can then use constraints to ensure that the values of these variables are consistent with the clues provided in the game. For example, if a square contains a clue of 1, we can add a constraint that requires exactly one of the eight surrounding squares to be a mine.

3 Inference

Inference is an essential part of solving a minesweeper game, as it allows us to make deductions about the possible values of variables based on the constraints that have been imposed on them. There are several different approaches to inference in CSPs, but one common method is called arc consistency.

Arc consistency involves checking each variable in the CSP and ensuring that its domain of possible values is consistent with the constraints imposed by its neighbors. This is done by examining each constraint and eliminating any values that are not possible given the constraints.

For example, consider a Minesweeper grid with the following constraints:

x_1	x_2	x_3
1	0	0
1	1	*
0	1	1

In this grid, the constraints indicate that the first and second rows contain one mine each. To check for arc consistency, we can examine each variable and eliminate any values that are not consistent with these constraints. For example, the domain of possible values for x_1 might initially be $\{0, 1\}$, but after applying the constraint that the first row must contain one mine, the domain is reduced to $\{1\}$. Similarly, the domain of possible values for x_2 is reduced to $\{0\}$.

This process is repeated for each variable in the CSP until no more values can be eliminated. At this point, the CSP is said to be arc consistent, and we can be confident that any solution found will be a valid one.

In the context of Minesweeper, arc consistency can be used to eliminate values for the binary variables representing the squares in the grid that are not consistent with the clues and actions of the player.

4 Using the Solvers

In this section, we describe how the basic solver, subset solver, and random opening solver can be used to solve the Minesweeper game.

When a game of Minesweeper starts, the first pick is typically made at random. This pick is made by the player, or in the case of a computer program, by picking a random corner. The choice of the first pick is important, as it can influence the strategy that is used to solve the game.

Given the information revealed by the first pick, the program can then use one of the solvers described below to find a solution to the Minesweeper game. The basic solver is a brute-force approach that is relatively simple to implement, but it may be slow for large grids. The subset solver is a more efficient approach that takes advantage of the structure of the game, but it may be more complex to implement. The random opening solver is a probabilistic approach that can find a solution quickly, but it may not be as reliable as the other solvers.

In practice, it may be beneficial to use a combination of these solvers, depending on the size of the grid and the initial pick. For example, the program could first try the basic solver to see if it can find a solution quickly. If this fails, it could then try the subset solver. If that also fails, it could then try the random opening solver. By using a combination of these solvers, the program can increase its chances of finding a solution in a reasonable amount of time.

4.1 Basic Solver

The basic solver is a simple CSP solver that uses a brute-force search approach to find a solution to the Minesweeper game. This solver works by enumerating all possible configurations of mines in the grid and checking each one to see if it is a valid solution.

To implement the basic solver, we can start by creating a function that generates all possible configurations of mines in the grid. This function should take as input the size of the grid and the initial clues, and it should return a list of all possible configurations of mines that are consistent with these clues.

Once we have this function, we can use it to generate a list of all possible configurations of mines in the grid. We can then iterate through this list and check each configuration to see if it is a valid solution to the Minesweeper game. To do this, we can use the constraints that we defined in the representation of the game to eliminate any configurations that are not consistent with the clues and actions of the player.

If we find a valid solution, we can return it as the output of the solver. If we exhaust the list of configurations without finding a solution, we can return a message indicating that no solution was found.

4.2 Subset Solver

The subset solver is a more efficient CSP solver that uses a divide-and-conquer approach to find a solution to the Minesweeper game. This solver works by dividing the grid into smaller subgrids and solving each one independently.

To implement the subset solver, we can start by defining a function that takes as input a subgrid and returns a list of all possible configurations of mines that are consistent with the clues and actions of the player. This function can be implemented using the same approach as the basic solver, with the added constraint that the mines in the subgrid must be consistent with the clues and actions in the surrounding squares.

Once we have this function, we can use it to divide the grid into smaller subgrids and solve each one independently. We can then combine the solutions for the individual subgrids to form a solution for the entire grid.

One advantage of the subset solver is that it can often find a solution much more quickly than the basic solver, as it is able to take advantage of the structure of the Minesweeper game to eliminate many invalid configurations of mines.

4.3 Random Opening Solver

The random opening solver is a CSP solver that uses a probabilistic approach to find a solution to the Minesweeper game. This solver works by randomly selecting squares to open and using the clues revealed by these openings to make deductions about the locations of the mines.

To implement the random opening solver, we can start by defining a function that takes as input a grid and a set of openings and returns a list of all possible configurations of mines that are consistent with the clues and actions of the player. This function can be implemented using the same approach as the basic solver, with the added constraint that the mines in the grid must be consistent with the clues revealed by the openings.

Once we have this function, we can use it to randomly select squares to open and use the clues revealed by these openings to make deductions about the locations of the mines. We can then repeat this process until we have found a solution to the Minesweeper game, or until we have exhausted the search space.