Solving Minesweeper using Linear Programming

Olga Golovatskaia[1]

[1]Mount Holyoke College

May 18, 2025

1 Executive Summary

This paper explores the application of Linear Programming to develop a strategy for playing the Minesweeper game. By formulating the game's constraints as a set of linear equations, we can determine possible mine configurations and identify cells that are guaranteed to be safe to click. Our model, adapted from Bruo's [1] approach and implemented using the Online Linear Optimization Solver [2], uses binary decision variables to represent the presence or absence of mines in each cell. While the LP model can find a valid solution, multiple mine configurations might satisfy the constraints. To determine which cells are definitively safe, we introduce a new constraint to our model. We demonstrate the model's application on a 5x5 Minesweeper board, outlining the constraint formulation and solution process to identify a safe cell on the grid. Ultimately, this study highlights the potential of LP in solving a logic-based game while acknowledging the limitations of the Minesweeper game and the need for guesswork within the game.

2 Background

Minesweeper is a well-known, old computer game that has been loved by people all around the world [3]. Many have heard of this game, but not everyone knows exactly how to play this game, other than clicking random squares on the grid, hoping there is no mine hidden underneath. The game starts with a grid of covered squares. Players select squares to uncover, revealing numbers that indicate how many mines are present in the eight adjacent cells (horizontally, vertically, or diagonally). Players can set flags at the coordinates where they suspect a mine is located. The goal is to uncover all squares without triggering a mine. To play this game in a logical process rather than relying on chance, we implement an optimization approach that addresses this challenge.

3 Methods

In our model, the decision variable $x_{i,j}$ represents whether the square coordinate at (i, j) contains a mine. We set $x_{i,j} = 1$ if the cell (i, j) contains a mine, otherwise, we set it to 0. An objective function is not required in our model since we are not trying to minimize or maximize the number of mines on the grid; these are already predetermined by the game. Instead, we focus on finding feasible solutions that satisfy all constraints. The constraints in our model are:

1. For uncovered cells known to be safe: $x_{i,j} = 0$
2. For numbered cells: $\sum_{(k,l)\in N(i,j)} x_{k,l} = N$, where $N(i, j)$ represents the set of cells adjacent to cell (i, j), and N is the number displayed on cell (i, j). This means that if you add up all the mines in the cells adjacent to the cell (i, j), you would get the number N as a result.
3. Binary constraints: $x_{i,j} \in \{0, 1\}$ for all cells

This model would give a possible configuration of mines on the grid. However, it might not be the only possible configuration for the game - multiple valid layouts of mines may exist. To get a more solid solution, we add new temporary constraints to the model: $x_{i,j} = 1$. This new constraint forces that specific cell (i, j) to contain a mine. We then re-run the model to see if it is feasible or not. If the model becomes infeasible with the added constraint, meaning it cannot find a solution, we can conclude that cell (i, j) must be safe in all possible configurations (no mine at that location). We might need to run the model multiple times until we get infeasibility. Each iteration requires adding a new test constraint while removing the previous test constraint, ensuring that we are only testing one cell at a time against the original constraints. If the safe square cannot be found, a square is selected randomly.

4 Solving a 5x5 Minesweeper Board

4.1 Initial Board State

Consider a 5x5 Minesweeper board with the following revealed cells:

- $(1, 3)$: 1
- $(2, 3)$: 1
- $(3, 2)$: 1
- $(3, 3)$: 1
- $(4, 1)$: 1
- $(4, 2)$: 2
- $(5, 1)$: 1

These hints represent the number of adjacent mines surrounding each cell. The board is illustrated below, where known values are shown, unknown cells are marked with "?", and empty cells indicate tiles with no adjacent mines.

		1	?	?
		1	?	?
	1	1	?	?
1	2	?	?	?
1	?	?	?	?

4.2 Formulation of the Linear Program

Let $x_{i,j}$ be a binary decision variable indicating whether cell (i, j) contains a mine:

$$x_{i,j} = \begin{cases} 1 & \text{if cell } (i,j) \text{ contains a mine,} \\ 0 & \text{otherwise.} \end{cases}$$

Each revealed cell imposes a constraint equal to the sum of its adjacent variables. The full set of constraints based on the revealed cells is as follows:

$$x_{1,2} + x_{1,4} + x_{2,2} + x_{2,3} + x_{2,4} = 1 \tag{1}$$
$$x_{1,2} + x_{1,3} + x_{1,4} + x_{2,2} + x_{2,4} + x_{3,2} + x_{3,3} + x_{3,4} = 1 \tag{2}$$
$$x_{2,1} + x_{2,2} + x_{2,3} + x_{3,1} + x_{3,3} + x_{4,1} + x_{4,2} + x_{4,3} = 1 \tag{3}$$
$$x_{2,2} + x_{2,3} + x_{2,4} + x_{3,2} + x_{3,4} + x_{4,2} + x_{4,3} + x_{4,4} = 1 \tag{4}$$
$$x_{3,1} + x_{3,2} + x_{4,2} + x_{5,1} + x_{5,2} = 1 \tag{5}$$
$$x_{3,1} + x_{3,2} + x_{3,3} + x_{4,1} + x_{4,3} + x_{5,1} + x_{5,2} + x_{5,3} = 2 \tag{6}$$
$$x_{4,1} + x_{4,2} + x_{5,2} = 1 \tag{7}$$

In addition, all revealed cells are known to be safe and thus must not contain mines:

$$x_{1,1} = x_{1,2} = x_{1,3} = x_{2,1} = x_{2,2} = x_{2,3} = 0 \tag{8}$$
$$x_{3,1} = x_{3,2} = x_{3,3} = x_{4,1} = x_{4,2} = x_{5,1} = 0 \tag{9}$$

4.3 Finding a Feasible Solution

Solving this linear program yields a configuration where:

$$x_{1,4} = x_{4,3} = x_{5,2} = 1,$$

with all other $x_{i,j} = 0$. This configuration satisfies all constraints and represents one potential placement of mines:

		1	*	?
		1	?	?
	1	1	?	?
1	2	*	?	?
1	*	?	?	?

4.4 Finding Safe Cells

We now test whether a given cell is guaranteed to be safe by adding a temporary constraint assuming it contains a mine and checking for feasibility. For example, choose cell (1,5):

$$x_{1,5} = 1 \qquad \text{(test constraint)} \tag{10}$$

We find that this system is still feasible, so the cell (1,5) **could** contain a mine. Let's try cell (2,4):

$$x_{2,4} = 1 \qquad \text{(test constraint)} \tag{11}$$

This system is infeasible, so cell (2,4) **cannot** contain a mine.

4.5 Conclusion

After one iteration of our LP analysis, we determine that cell (2,4) is definitely safe to click. After clicking it, we would then update our constraints with the newly revealed information and repeat the process.

5 Discussion

5.1 Interpretation of Results

Feasible Solutions: Our model gives us at least one way mines could be arranged on the board that matches all the revealed numbers. In our example, it showed one possible layout with mines at positions (1,4), (4,3), and (5,2).

Guaranteed Safe Moves: The most valuable part is that we can prove some squares must be safe. When we tested position (2,4) by forcing it to contain a mine, the model couldn't find any solution, meaning this square does not contain a mine.

5.2 Dual Variables and Reduced Costs

Dual Variables: In this problem, we are dealing with a Binary Integer Program and are primarily interested in **feasibility** rather than optimization. So, dual variables are not meaningful in this context, as they're tied to optimization and sensitivity analysis, not feasibility. However, when solving our model with the command `display c13.dual, c23.dual, c32.dual, c33.dual, c41.dual, c42.dual, c51.dual;`, we obtained zero values for all dual variables:

```
c13.dual = 0
c23.dual = 0
c32.dual = 0
c33.dual = 0
c41.dual = 0
c42.dual = 0
c51.dual = 0
```

1. These zero values indicate that none of our numbered cell constraints are uniquely binding at the optimal solution.
2. The zero dual values confirm that multiple valid mine configurations can satisfy the same revealed numbers on the board. This reflects the inherent uncertainty of the game.

Reduced Costs: Similarly, reduced costs, which normally tell us how much the objective would change if we changed the value of a variable, are not relevant here, since we are not trying to improve an objective function but simply looking for any solution that satisfies all the constraints.

5.3 Recommendations for Future Work

Our LP-based approach to Minesweeper highlights the limitations inherent to the game itself. There is no perfect model that can eliminate all uncertainty in Minesweeper because the game explicitly contains situations where a player must make a guess, and no amount of logical deduction or optimization can resolve this. Rather than attempting to enhance our model, a more promising direction would be to modify the game itself. For example, one could implement an algorithm that generates Minesweeper boards where every move can be determined through logical deduction without guessing.

5.4 Limitations of the Model

Solution Time: Testing each cell individually for safety requires multiple LP solves, which may take considerable time for large board states.

Computational Complexity: As board size increases, the number of variables and constraints grows quadratically, potentially making large boards computationally intensive to solve.

Initial Move Challenge: The very first move in Minesweeper cannot be determined by our model since no constraints exist yet. Some Minesweeper implementations ensure the first click is safe.

6 References

References

[1] Brou, A. (2023). *The perfect Minesweeper AI: an approach using Linear/Constraint Programming*. Medium.
Medium article: "The perfect Minesweeper AI"
This article provided the foundational approach for building our Linear Programming model for Minesweeper.

[2] Sierksma, G., & Zwols, Y. *Online Linear Optimization Solver*.
Linear Optimization Solver
An online tool used to formulate and solve our LP model.

[3] *Minesweeper (Video Game), Wikipedia*
"Minesweeper (Video Game)."
This explains the fundamental rules of Minesweeper.