HW 8: Portfolio Project

Jeremy Tsang

December 7, 2020

The Problem

In Minesweeper a player is given board of m rows and n columns with $k < m \cdot n$ mines. The goal is for the player to open all cells that do not contain a mine.

Decision Question

From Kaye (Springer-Verlag New York, Volume 22, Number 2, 2000) the decision problem is: "Given an arbitrary set of mines and numbers on a rectangular grid, can mines be placed consistently, following the usual minesweeper rules?"

This problem is NP-complete. A proof by Kaye is given in the aforementioned paper where he goes on to show that SAT can reduce to the Minesweeper problem. He goes to show that one can construct the basic logic gates AND, OR, and NOT (and therefore an entire computer) using regions of a Minesweeper board to model a wire carrying voltage. This can be done by assigning a "direction" to the region where the unopened cells are the digital logical inputs. Hence, by strategically placing the unopened cells it is possible recover the inputs of SAT.

The Verification Algorithm of User's Solution

Store the mines in a 2D Boolean array with m rows and n columns, where if True than the row-col coordinate holds a mine. Consider the player's moves as nested array each row being a 2-element positions. This array's length is at most $m \cdot n$ since the user can't open more cells than there are on the grid. Note that after we discover how many mines are on the board we know that must be equal to the number of opened cells (total cells minus opened cells) for the player to win.

```
# opened_cells: 2D array of bool. Each row is length 2.
# move_count: number of rows in opened_cells
# mines: 2D array of bool with height m and width n. True if mine else False
# m: int height of the minesweeper board
# n: int width of the minesweeper board
verification(opened_cells, move_count, mines, m, n):
    mine_count = 0
    for i in range(m):
        for j in range(n):
            if mines[i][j]:
                mine_count += 1

    if (m * n) - move_count != mine_count:
```

```
        return False

    for i in range(move_count):
        if mines[opened_cells[move_count][0]][opened_cells[move_count][1]]:
            return False

    return True
```

This simple brute force algorithm first checks if the number of mines is equal to the number of opened cells. The outer loop has m iterations while the inner loop has n iterations so the entire time would be $m \cdot n$. It also has a nested `for` loop which has at most $m \cdot n$ iterations (the user can't open more cells than there are cells on the grid). Hence it's runtime is $\mathcal{O}(m \cdot n)$ which is polynomial in the number of cells in the board.