THE UNIVERSITY OF BRITISH COLUMBIA

NP-Game: Minesweeper

Group 4

Carmen(Jiawen) Xu (40307688)

Yuecheng Cai (66794462)

Tian Le Chen (32135329)

Adriana Sangulin (49289168)

Weixi Long (11932084)

The University of British Columbia

Math 441: Mathematical Modelling: Discrete Optimization Problems

Professor: Jozsef Solymosi

Abstract

The purpose of this paper is to examine Minesweeper's solvability by explaining its relation to the subgroup of NP problems, and how its membership within the group relates to the ways its solutions can be found. The conclusions of NP-completeness, polynomial time reduction and solution algorithm discussed in this paper heavily rely on and refer to the findings of Richard Kaye, who published a direct proof of Minesweeper's NP-completeness. Using the method of polynomial time reduction, we have shown that minesweeper can be reduced to an SAT problem, and that an SAT problem can be reduced to Minesweeper thus implying that Minesweeper can be solved in at most the same time as it takes to solve an SAT problem. The existence of an algorithm which solves an SAT problem in polynomial time still remains to be proved, however the polynomial time reduction of Minesweeper to SAT problem indicates the existence of an algorithm which checks the satisfiability of Minesweeper's solution which runs in polynomial time. Therefore, Minesweeper belongs to the group of NP-complete problems. Cellular automation could be applied as the solver of the minesweeper problem. In this article, three states were defined to identify different types of squares. Based on this solver, two approaches could be applied to solve the minesweeper game: Naive Single Point Strategy and Double Set Single Point Algorithm. Also, in this article, we proved that any logic circuit problem could be represented using a minesweeper board.

1. Introduction

Minesweeper is a video game whose early origins emerge in the 1960s, and has gained widespread popularity through the Windows operating system in 1992. The emerging of such puzzle games through the PC has shaped generations by detecting, inspiring, and challenging their interest in solving logical puzzles. As members of this generation, we wanted to further our understanding of successful strategies, and their mathematical properties utilizing a more sophisticated mathematical background, which we obtained motivated by such games. In this paper, we will argue that minesweeper is NP-complete and provide a summary of proofs that have been published on this matter, as well as explain the properties of a solver which could solve this game in polynomial time.

2. Minesweeper Rules

Minesweeper is a single player puzzle game whose objective is to determine whether each uncovered square on a n x m board is a mine or an integer. Each integer denotes the number of mines adjacent to that square, and if a discovered square is blank, the integer representation of that square is 0. The integer value of a square ranges from 0 to 8, as there are a maximum of 8 neighboring squares. The game starts with a board in which some squares are discovered and some are not (see Fig. 1). Lets focus on the discovered square at position (3,2) whose integer value 1 indicates there is one mine adjacent to it. As there is only one undiscovered square (4,1) adjacent to it, and none of the discovered squares adjacent are mines, we can conclude (4,1) is a mine. Using the same logic, we can conclude (6,2) is a mine and the square (5,2) with the integer value 2 has 2 mines adjacent to it, and all other undiscovered neighbours of (5,2) must be integers (see Fig. 2). The game ends successfully when all undiscovered squares have been discovered (see Fig. 3). If a mine has been discovered as an integer, the game ends immediately, and the player loses.

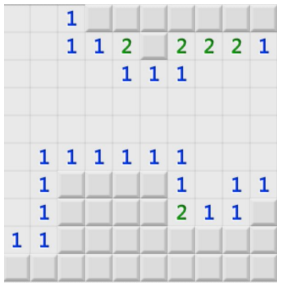


Fig. 1 : An example of a 9 x 9 Minesweeper game.

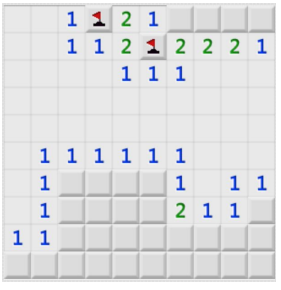


Fig. 2 : An example of a correct move.

Fig. 3 : An example of a solved game, with (1,9) a mine.

3. Polynomial Reduction and NP-Completeness

According to the computational complexity theory, a problem is NP complete when a non-deterministic Turing machine can solve it in polynomial-time, a deterministic Turing machine can verify its solutions in polynomial time, and can be used to simulate other problems with similar solvability.[1] Richard Kaye proved minesweeper is NP complete by reducing the Boolean Satisfiability Problem (SAT) to Minesweeper, and his findings were published in *The Mathematical Intelligencer* in 2000. In 1971 Steven Cook showed that SAT is NP-complete, and therefore by reducing the SAT problem to Minesweeper, Kaye's proved Minesweeper is indeed NP-complete, as minesweeper is at most as difficult to solve as an SAT problem. Kaye firstly showed Minesweeper can be reduced to an SAT problem (see Fig. 4). For each square in Minesweeper it is true that it is either a mine or some integer k ranging from 0-8 . If it is false that a square is a mine, and it is true it is an integer, then for precisely k squares adjacent to that square, it is true they are a mine . Reducing minesweeper to these true/false values indicates it is possible to connect all squares into a logical circuit and minesweeper becomes equivalent to a SAT problem. Hence it is possible to check whether for a given value of a square, are there truth values that can be assigned to all other squares that make the entire circuit true, precisely are there truth values which make the solution valid and the board consistent.

Consider a three-by-three block of squares labelled as shown.

a	b	c
d	e	f
g	h	i

Let a_{m} denote "there is a mine at a," and for $0 \leq j \leq 8$ let a_j denote "there is no mine at a and precisely j mines in the neighbouring squares around a"; and similarly for b, c, d, . . . , i. Then the rules of Minesweeper for the centre square e can be described by the following statements:

1. precisely one of e_{m}, e_0, e_1, . . . , e_8 is true;
2. for $k = 0, 1, \ldots, 8$, if e_k is true then precisely k of a_{m}, b_{m}, c_{m}, d_{m}, f_{m}, g_{m}, h_{m}, i_{m} are true;

and these can all be expressed (in a rather cumbersome fashion) by boolean circuits in the 90 inputs a_{m}, a_0, . . . , i_8. If we let C be the circuit consisting of all of these circuits for all points in the rectangular grid in place of e, the outputs of all these being combined into a single AND gate, then the Minesweeper problem becomes equivalent to an instance of SAT: given certain inputs for C being true or false, are there truth values for the other inputs that makes the output of the whole circuit C true?

Fig. 4 : Reducing Minesweeper to SAT, source, (Kaye, 2020, p.12.)

Reducing minesweeper to SAT is not sufficient for proving NP-completeness, as every algorithmic problem can be represented using booleans, hence we must reduce

[1] "NP-Completeness," Wikipedia (Wikimedia Foundation, November 25, 2020), https://en.wikipedia.org/wiki/NP-completeness.

SAT to minesweeper to prove NP-completeness. An arbitrary boolean formula can be converted to a minesweeper configuration if and only if that formula is satisfiable, and such circuit-SAT problems answer the question of the following form: given an input, can an output be created? These boolean expressions are converted using wires, and not gates and will be further explained in the Solution Plan portion of this paper.

4. Minesweeper is NP-Complete

As we know from the above, the player will need to click the location on the gameboard to reveal numbers, and the numbers correspond to the number of adjacent squares that contain mines. If the player clicks a square with a mine, the player loses; otherwise, the player wins for the round.

To determine if a solution exists for the given m-by-m board, you either check to see if the given answer shows logically after you failed the game, or you can correctly guess all the location of mines and eventually fill in the entire board. A computer algorithm will run through the squares on the board and verify that these mines produce the numbers seen as neighbors to ensure that the solution is consistent. With the given Minesweeper grid with m squares, the runtime will be 8*t*m, where t is the time it takes to check the surrounding of one square, since each square will only have a maximum of 8 adjacent squares to check. As we can see, the strategy stated above is valid and this algorithm clearly runs in polynomial time, the Minesweeper problem is considered as NP.

On the other hand, If we want to say that Minesweeper is NP-Complete, we also have to show that Minesweeper is NP Hard. Suppose we have an arbitrary boolean circuit with input variables: x1,x2,...,xn and output value y. We can make “wires” out of Minesweeper in order to give it either true or false value, and we assume this wire is traveling from left to right. We can also form AND, OR, NOT gates which will make it easier to combine signals, and eventually assign either true or false to y depending on the initial values of our variables. We would like to see if a given Boolean circuit has an assignment of its inputs that makes the output true[2] and by definition, this is our “circuit satisfiability”. Since we are able to construct wires and form the AND, OR, NOT gates in Minesweeper, we can build a corresponding minesweeper grid to our circuit and create an algorithm to make a random boolean formula. A mine to the value of y is true will be

[2] "Circuit Satisfiability Problem". En.Wikipedia.Org, 2020, https://en.wikipedia.org/wiki/Circuit_satisfiability_problem#:~:text=In%20theoretical%20computer%20science%2C%20the,that%20makes%20the%20output%20true.

assigned to our constructed Minesweeper grid. We are assuming the algorithm is consistent if and only if the random boolean formula is satisfiable in polynomial time. In this way, circuit satisfiability can be determined using Minesweeper Consistency and the output of the circuit is verifiable in polynomial time[3]. Since we know that circuit satisfiability is NP Hard, Minesweeper Consistency is NP-Hard as well.

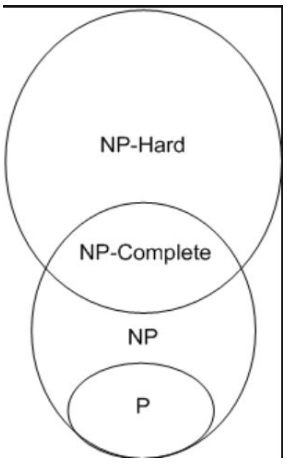


Fig. 5 : NP-Completeness

In conclusion, since Minesweeper consistency has been shown to be in NP and is NP-Hard, it is NP-Complete.

5. Minesweeper can be solved in polynomial Time

Since We known that Minesweeper is Np-completeness problem. To find out the polynomial time of minesweeper, a DFA(Deterministic Finite Automata) ,which is for each input symbol, one can determine the state to which the machine will move, will figure out that the running time of 2-dimensional Minesweeper from 1-dimensional Minesweeper. [4]

As Kasper mentions that some properties of 1-dimensional Minesweeper are considered, let use a 1-D array to represent the game like below:

X_1	X_2	X_3	X_4	X_5	X_6	X_7
1	*	*	2	*	1	blank

*Fig. 6 : array s1**21bs.*

Here, it call array *s1**21bs*(see Fig. 6). *S* represents the point of start to end, *b* indicates a blank and * indicates a mine. According to the logic inference, each square have chosen

[3] "Circuit Satisfiability Problem". En.Wikipedia.Org, 2020, https://en.wikipedia.org/wiki/Circuit_satisfiability_problem#:~:text=In%20theoretical%20computer%20science%2C%20the,that%20makes%20the%20output%20true.

[4] "The complexity of Minesweeper and strategies for game playing".Kasper Pedersen,2003-2004, ,University of Warwick. Page 16.

as 0-2(0 is blank) or a mine. If it's not a mine, it will get an equation like $X_2 + X_3 + X_5 \geq 2$, and then it will have another equation when getting another clear square, which is x2>=1. After it combines two equations, then $X_3 + X_5 \geq 2$. Thus, each variable represents a mine. In other words, it indicates that X_3 and X_5 will be mine. After doing this the equation, it must do the whole procedure once again until all the squares have been checked. Thus, the worst case of 1-dimensional Minesweeper is Θ(n) where n is the array's length.

Obviously, the normal Minesweeper is a 2-dimensional Minesweeper, where n, m is the map's length and width. Therefore, the worst case of 2-dimensional Minesweeper is $m * \Theta(n)$, which is Θ(nm), also is Θ(n^2). Taking into account all these factors, it may reasonably come to the conclusion that, the Minesweeper can be solved in Θ(n^2).

More details for 2-dimensional Minesweeper, We assumed a $m * n$ Minesweeper game. An algorithm, called *updateBoard(char[ ][ ] board, int[ ] click)* in Java, checks the result of next click location showing the same time complexity as well.

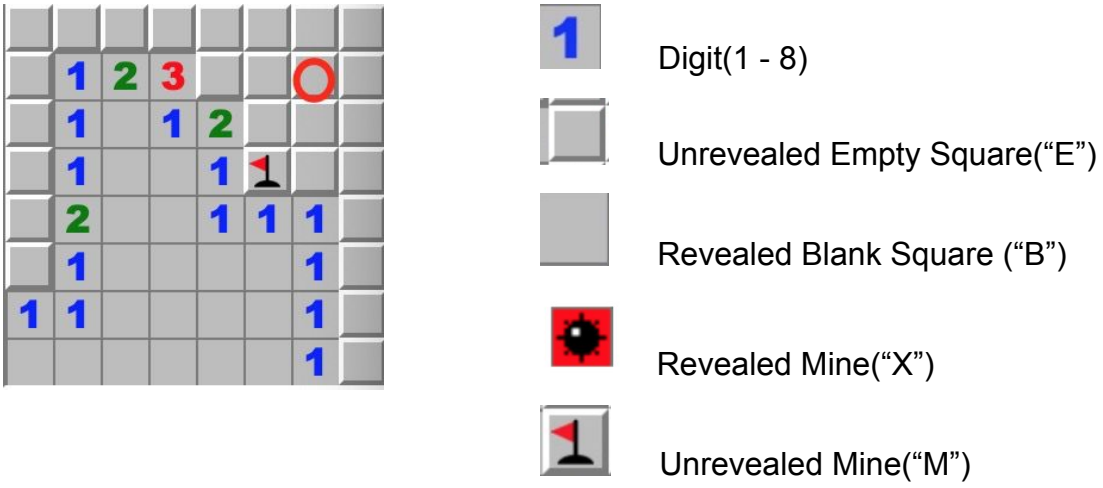


Fig. 7: A case of game waiting for next click.

Every time clicked on a square, if it's not a mine, the algorithm needs to judge its own and the surrounding square's state, in other words, the judgment will gradually spread out until it encounters a boundary or mine. This process can be simulated by a recursion. The algorithm is following:

A. Let $Next = board[x][y]$ be the next clicked (Next is the red circle in Fig. 7). If *Next* meets a mine(M), it sets Next to X and returns back.

B. If *Next* is not a mine M, call $turnOver()$ method and pass the *Next* coordinates to open the empty *square E*: first traverse 8 elements around the *Next*, jumping across

the border, count the number of mines around it, and if the mine count *cnt* is 0, it means *Next* meets a square without adjacent mines, continues to call the $turnOver()$ method, passes into the surrounding elements as click coordinates, and the recursive depth takes precedence for traversal.

C. If the mine count*(cnt)* is greater than 0 (it is impossible to be less than 0), it means that the empty square *E* is adjacent to at least one mine, and changing its element to the mine count cnt means the number of adjacent mines.

D. If no more squares can be revealed, it means it has already skipped all boundaries and non-empty squares in the process of revealing squares.

In conclusion, the worst case is it needs to go through all the elements of the 2-dimensional array *board[ ][ ].* Thus, the worst case of the time complexity and space complexity both are Θ(nm).

```java
class 441_project_Minesweeper_solution {
    //Increment exploring eight grids
    //around the exposed point.
    int[] scX = {-1, -1, -1, 0, 0, 1, 1, 1};
    int[] scY = {-1, 0, 1, -1, 1, -1, 0, 1};
    public char[][] updateBoard(char[][] board, int[] click) {
        //row x column y
        int x = click[0], y = click[1];
        //Rule 1: if Next meets a mine, otherwise turnOver()
        if (board[x][y] == 'M')
            board[x][y] = 'X';
        else
            turnOver(board, x, y);
        return board;
    }
    //Recursively expose unrevealed square
    public void turnOver(char[][] board, int x, int y) {
        //the mine count
        int cnt = 0;
        for (int i = 0; i < 8; i++) {
            //To set eight grids around Next as newX, newY
            int newX = x + scX[i];
            int newY = y + scY[i];
            //Skip the border
            if (newX < 0 || newX >= board.length
            || newY < 0 || newY >= board[0].length)
                continue;
            //Number of mines
            if (board[newX][newY] == 'M')
                cnt++;
        }
        //Rule 2: No adjacent mines, and check Next
        if (cnt == 0) {
            board[x][y] = 'B';
            for (int i = 0; i < 8; i++) {
                int newX = x + scX[i];
                int newY = y + scY[i];
                //Skip border and empty square
                if (newX < 0 || newX >= board.length || newY < 0
                || newY >= board[0].length || board[newX][newY] != 'E')
                    continue;
                //Rule 4: Recursion
                turnOver(board, newX, newY);
            }
        } else {
            //Rule 3: If meets adjacent mines and modified cnt.
            board[x][y] = (char)(cnt + '0');
        }
    }
}
```

Fig. 8: Java code for solution.

6. Minesweeper solver

One of the solvers that could solve the minesweeper problem is cellular automaton (CA). A cellular automaton defines a set of cell states Q and a transition function f which takes a state at time t to the state t + 1. Q is a finite nonempty set which could be represented as the formula shown here:

$$Q = \{\#, \bullet, \circ\} \times \{0, 1, \ldots, 8\}$$

Therefore, the status of a cell has three cases, contain a mine, covered but without a mine, and uncovered square, and those three cases were represented as '#', '$\circ$', and '$\bullet$', respectively. Also, each covered cell will be assigned a value to indicate the surrounding number of mines, and this value was denoted as 'label'. Specifically for the minesweeper problem, the rules for cell state transition is defined by Andrew[5], where three cases were defined and the formula of the transition function f is defined as below:

$$x^{t+1} = \begin{cases} \circ, & (x^t = \cdot) \wedge \Big(\exists\, y \in u_1(x)\colon y^t = \circ \\ & \qquad \wedge \Big(\nu(y) = 0 \vee \nu(y) = \sum_{z \in u_1(y)} \chi(z^t, \#)\Big)\Big) \\ \#, & (x^t = \cdot) \wedge \Big(\exists\, y \in u_1(x)\colon y^t = \circ \wedge \sum_{z \in u_1(y)} \chi(z^t, \cdot) = 1 \\ & \qquad \wedge \Big|\nu(y) - \sum_{z \in u_1(y)} \chi(z^t, \#)\Big| = 1\Big) \\ \cdot, & \text{otherwise} \end{cases}$$

Three cases were considered in this transition function that could transfer the state of cell x at time t to time t+1. In the first case, if and only if for any neighbour 'y' of cell x, the label of y is equal to the number of the surrounding mines. Then we could say the cell x has no mine. This case was also being denoted as all free neighbours (AFN) and an example of AFN is shown below:

(a) *(b)*

Figure 9. (a) An instance of All Free Neighbours (b) An instance of All Marked Neighbours

[5] Becerra, David J. 2015. Algorithmic Approaches to Playing Minesweeper. Bachelor's thesis, Harvard College.

For the second case, for any neighbour ‘y’ of cell x, the uncovered neighbour of cell ‘y’ is 1 and the unmarked number of neighbours is 1. Therefore, this cell ‘x’ could be identified as having a mine and the figure of showing this case is shown above. In this case, all marked neighbours (AMN) was used to indicate the minesweeper local board. Otherwise, we still keep this cell uncovered. As the information provided in this case is not enough of the players to determine the states of the cell.

However, the drawback of cellular automation is weak handling of nondeterminism. As the configuration at time t equals the configuration at time t+1, the cellular automation operator can not move forward and is stuck into the stationary configuration. Therefore, in order to completely finish a minesweeper game, an approach that could handle the stationary configuration is needed.

7. Minesweeper Solution Plan:

7.1 Naive Single Point Strategy

The first algorithm that could solve the minesweeper game is the Naive Single Point strategy. The figure shown below is the basic working flow of this approach.

Algorithm 1. Naive Single Point [6]

```
S ← {}
while game is not over do
  if S is empty then
    x ← SELECT-RANDOM-SQUARE()
    S ← {x}
  end if
  for x ∈ S do
    probe(x)
    if x = mine then
      return failure
    end if
    U_x ← UNMARKED-NEIGHBORS(x)
    if isAFN(x) = True then
      for y ∈ U_x do
        S ← S ∪ {y}
      end for
    else if isAMN(x) = True then
      for y ∈ U_x do
        mark(y)
      end for
    else
      Ignore x
    end if
  end for
end while
```

The two deductions introduced in the previous section AFN and AMN are mainly used in this algorithm in order to solve the minesweeper game. The general approach of

[6] ”The complexity of Minesweeper and strategies for game playing”.Kasper Pedersen,2003-2004, ,University of Warwick. Page 16.

the strategy is to maintain a set S which is a collection of safe cells that the algorithm will probe. If S is empty, the algorithm will randomly select a cell and insert it into set S. As a result, since no information was provided at the beginning, the first move will be a randomly chosen square from the entire board. Because at the first beginning of the minesweeper board, zero information could be used by the player. For each cell in S, the solver will determine if the cell falls into AFN or AMN. The solver will then proceed to probe or mark all adjacent unknown squares depending on which case was found (AFN and AMN). In the case where the cell is neither of the two above options, this algorithm typically abandons this cell. However, this step introduces an ordering problem. More specifically, the sequence of deciding which cell needs to be probed becomes important. It cannot be guaranteed that the sequence of choosing the probing square is optimal. In other words, a square may not be a case of AMN or AFM when considered the first time. However, if considered again with more board information, we may be able to identify whether this cell contains a mine or not.

7.2 Double Set Single Point Algorithm

Another approach that shows a better performance of solving minesweeper is the double set single point algorithm (DSSP). Different from the Naive single point strategy, the double set single point algorithm maintains two sets S and Q. Similar to the NSP, S contains safe squares the solver will probe. The additional set Q is how this algorithm deals with the ordering problem that was introduced in the NSP algorithm. Rather than abandoning probed squares which do not fall into AFN or AMN, this approach classifies these points as questionable and inserts them into Q. In this approach, once enough information is provided, the algorithm will go back to the set Q and re-analysis the squares in Q. In other words, the set Q could also be considered as the frontier of the covered square. Since set Q is a combination of the uncovered square that does not fall in to AFN and AMN, the algorithm will firstly probe the squares in set S, the information provided by this step will be used to identify whether any squares in set Q becomes solvable. However, in the case if S is empty and Q is nonempty, the algorithm will go back to the original stage by randomly selecting a square to probe. The figure below showing the set Q of a minesweeper board, and the red outlined covered squares are the elements in Q.

Algorithm 2. Double Set Single Point Algorithm [7]

```
opener = FIRST-MOVE()
S ← {opener}
Q ← {}
while game is not over do
  if S is empty then
    x ← SELECT-RANDOM-SQUARE()
    S ← {x}
  end if
  while S is not empty do
    x ← S.remove()
    probe(x)
    if x = mine then
      return failure
    end if
    if isAFN(x) = True then
      S ← S ∪ UNMARKED-NEIGHBORS(x)
    else
      Q ← Q ∪ {x}
    end if
  end while
  for q ∈ Q do
    if isAMN(q) = True then
      for y ∈ UNMARKED-NEIGHBORS(q) do
        mark(y)
      end for
      Q.remove(q)
    end if
  end for
  for q ∈ Q do
    if isAFN(q) = True then
      S ← S ∪ UNMARKED-NEIGHBORS(q)
      Q.remove(q)
    end if
  end for
end while
```

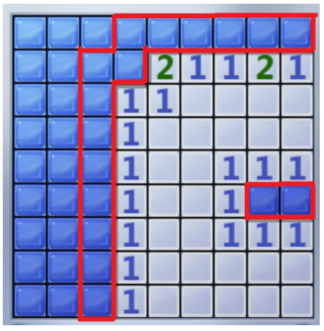


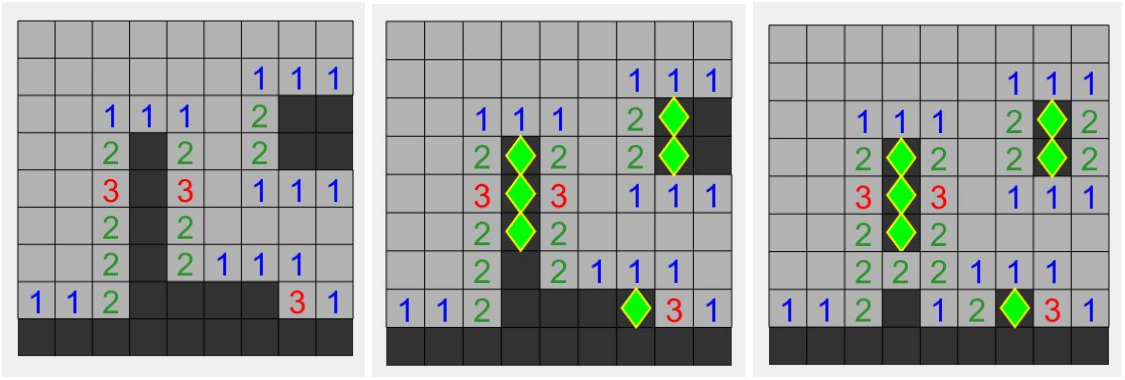
Figure. 10: An example of set Q

Different from NSP, this approach shuffles between probing and marking stages. The first stage is a probing stage. The solver iterates and probes through each square in S. Like previous implementations, if S is empty the algorithm randomly selects a covered square to insert into S.

The second stage is a marking stage. However, in this stage the algorithm will only search for the squares that satisfy the AMN. If any cells belonging to Q are found to be AMN, then the neighbours are marked and the cell will be removed from Q.

[7] Andrew Adamatzky. How cellular automaton plays minesweeper. Applied Mathematics and Computation, 85(2–3):127–137, 1997.

The last stage in DSSP is another probing stagep. Unlike the first probing stage, the solver iterates through Q rather than S searching which searches for instances belonging to AFN. If any square is found to be falling in AFN, all its neighbour squares will be added into S. Then the algorithm will keep shuffling from probing phase and marking phase until the minesweeper problem is completely finished. Also, in the middle of this approach, if S is empty and Q is nonempty and no squares in Q could be identified as AFN or AMN. The solver returns back to stage one and randomly selects a cell from Q and continues the process until the minesweeper is completely finished. The figure below shows the specific steps of the DSSP algorithm.



(a) *(b)* *(c)*

Figure. 11: (a) Initial probing step (b) Marking step (c) Intermediate probing step

8. Using minesweeper solution plan solve other problem

Based on the solver and algorithms introduced before, other problems like logic circuit problems could be solved. As introduced before, a 3-SAT problem could be reduced to a minesweeper problem. In other words, a logic circuit problem could be represented using a minesweeper board. Following are some examples of how to represent logic gates using minesweeper boards.

First of all, the figure shown below is an example of how to represent a wire using a minesweeper. Based on the basic fact of consistency introduced in the previous sections, either the square X or X' has a mine. This is a basic fact of minesweeper that mines' locations and information on one side of the grid can influence the information on another side of the grid. As we all knew, wires are a way to transfer the same value from one

location to another. In Minesweeper, we can use the configuration in order to transfer the "truth" value of mine or no mine down the length of the wire.

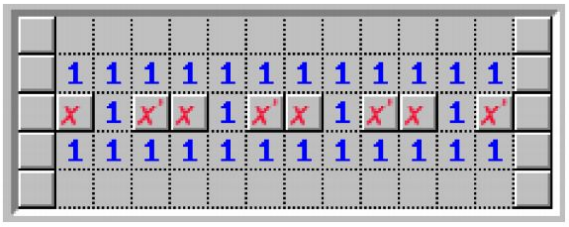


Figure. 12: Potential mines of a wire in a minesweeper board.[8]

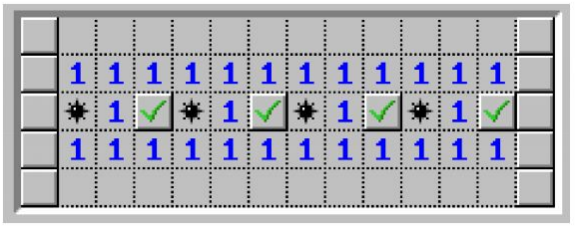


Figure. 13: An example of wire with a boolean value as 'true'.[9]

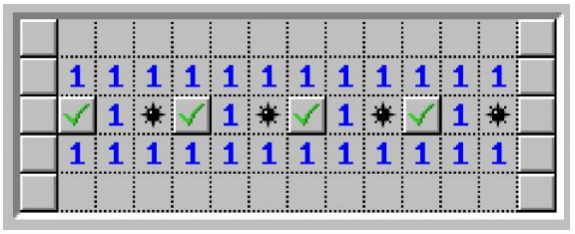


Figure. 14: An example of wire with a boolean value as 'false'.[10]

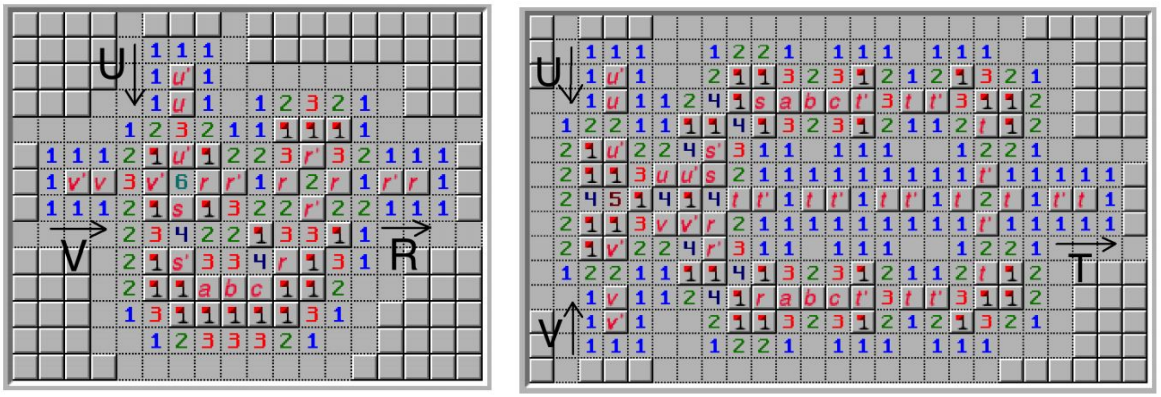
In figure 12 shown above, we first assume that the square in position X has a mine. This assumption would then necessitate that the entire grid looks like the arrangement shown in figure 13 where the output boolean value of this wire is 'true'.

In contrast, if the square in position X' has mine, the arrangement becomes the figure 14, where the boolean value of that wire is 'false'. This indicates that the wire is carrying the information about the location of the mines down the length of the wire.

Because circuits have a series of logic gates like AND, OR, NOT, XOR, etc. Those gates could be represented using minesweeper as well. The figure 15 (a) is the OR gate and the figure 15 (b) is an example of an AND gate. Both gates require two boolean inputs U and V in this case. For the OR gate, if either of U and V is 'true', in other words, if either

[8] Carini, R. (2015). Circuits , Minesweeper , and NP Completeness. 1–9.
[9] Carini, R. (2015). Circuits , Minesweeper , and NP Completeness. 1–9.
[10] Carini, R. (2015). Circuits , Minesweeper , and NP Completeness. 1–9.

U and V has mines, then the output R has a mine. Similar to the AND gate, if both U and V have a mine, then the output T has a mine.



(a) *(b)*

Figure. 15: (a) Example of OR gate (b) Example of AND gate. [11]

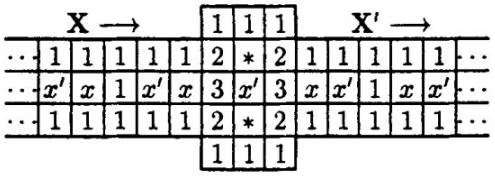


Figure. 16: Example of NOT gate. [12]

Figure 16 shows an example of how to construct a NOT gate using minesweeper. This is important for the logic circuit as we input the 'true' boolean value, this gate will output the 'false' boolean value. Also, we are able to bend wires and to split them. The figure below shows how to do this. Figure 17 (a) is a simple 90-degree bend in the wire and figure 17 (b) shows how a wire can be terminated.

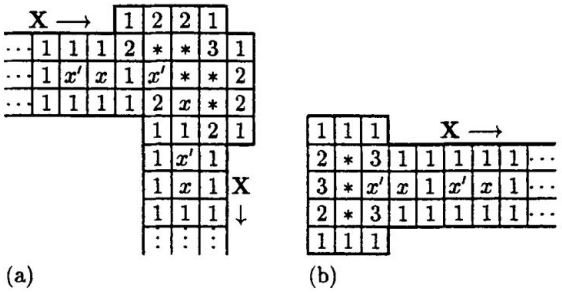


Figure.17: (a) Example of bent wire. (b) Example of terminated wire. [13]

[11] Carini, R. (2015). Circuits , Minesweeper , and NP Completeness. 1–9.
[12] "Circuit Satisfiability Problem". En.Wikipedia.Org, 2020, https://en.wikipedia.org/wiki/Circuit_satisfiability_problem#:~:text=In%20theoretical%20computer%20science%2C%20the,that%20makes%20the%20output%20true
[13] "Circuit Satisfiability Problem". En.Wikipedia.Org, 2020, https://en.wikipedia.org/wiki/Circuit_satisfiability_problem#:~:text=In%20theoretical%20computer%20science%2C%20the,that%20makes%20the%20output%20true

Other more complicated gates like XOR could be constructed using AND and NOT gates which is shown in figure 18.

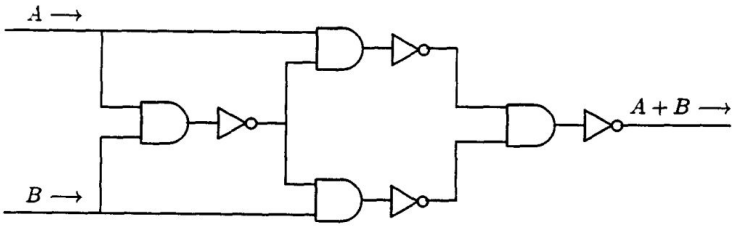


Figure.18: Making XOR gate with AND and NOT gates. [14]

Also, figure 19 shows that a crossing of two wires can be simulated in the plane by using three splitters and three XOR gates.

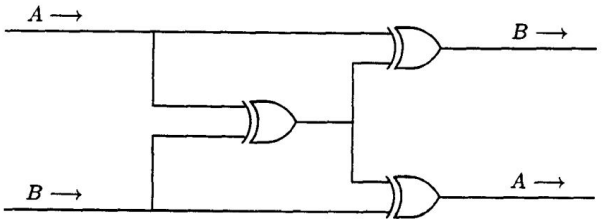


Figure.19: Crossing two wires with XOR gates. [15]

So now we could use those logic gates to construct any logic circuit until there's only one output circuit. Like the figure shown below, we could perfectly represent this boolean circuit using minesweeper.

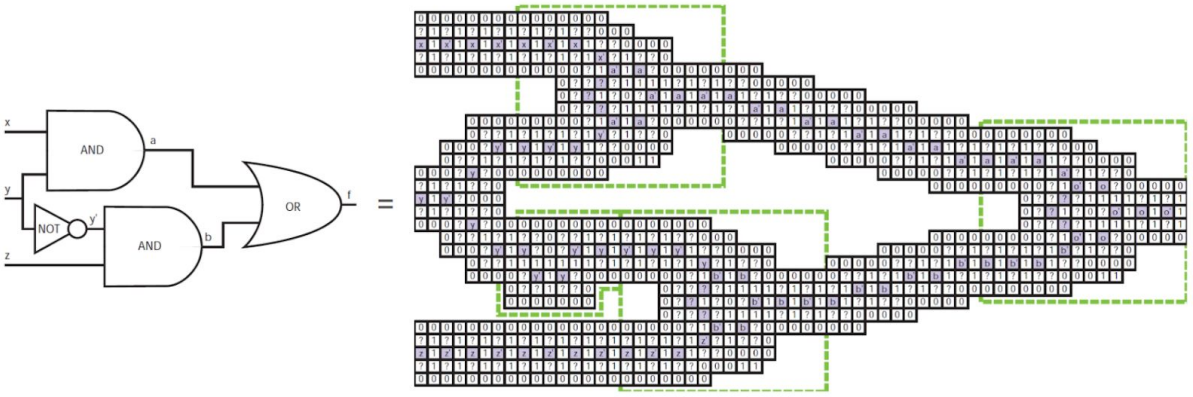


Figure.20: Representing a boolean circuit using a minesweeper board. [16]

[14] "Circuit Satisfiability Problem". En.Wikipedia.Org, 2020, https://en.wikipedia.org/wiki/Circuit_satisfiability_problem#:~:text=In%20theoretical%20computer%20science%2C%20the,that%20makes%20the%20output%20true

[15] "Circuit Satisfiability Problem". En.Wikipedia.Org, 2020, https://en.wikipedia.org/wiki/Circuit_satisfiability_problem#:~:text=In%20theoretical%20computer%20science%2C%20the,that%20makes%20the%20output%20true

[16] "Kaye, R. (2000). Minesweeper is. Problems of Information Transmission, 22(2).

At this point, we can layout any logic circuit on the minesweeper board. Solving this logic circuit problem is equivalent to solving a minesweeper problem. And we could achieve it by using the minesweeper solver and algorithms.

Reference

[1] Andrew Adamatzky. How cellular automaton plays minesweeper. Applied Mathematics and Computation, 85(2–3):127–137, 1997.

[2] Becerra, David J. 2015. Algorithmic Approaches to Playing Minesweeper. Bachelor's thesis, Harvard College.

[3] Carini, R. (2015). Circuits , Minesweeper , and NP Completeness. 1–9.

[4] "Circuit Satisfiability Problem". En.Wikipedia.Org, 2020, https://en.wikipedia.org/wiki/Circuit_satisfiability_problem#:~:text=In%20theoretical%20computer%20science%2C%20the,that%20makes%20the%20output%20true.

[5] Kaye, R. (2000). Minesweeper is. Problems of Information Transmission, 22(2).

[6] Kaye, R. (2000). Minesweeper is NP-complete. *The Mathematical Intelligencer*, *22*(2), 9–15. https://doi.org/10.1007/bf03025367

[7] Kaye, R. (2000). Minesweeper Is NP-Complete. Springer-Verlag, vol. 22, no. 2, 2000, pp. 9–15.

[8] Kasper Pedersen. (2003-2004). *The complexity of Minesweeper and strategies for game playing*. Department of Computer Science. University of Warwick.

[9] Wikimedia Foundation. (2020, November 25). *NP-completeness*. Wikipedia. https://en.wikipedia.org/wiki/NP-completeness.