NP-Completeness TheoryImplementations in Minesweeper Problems

Thea Olivia[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13511001@std.stei.itb.ac.id

***Abstract*–This paper represents the effectiveness and implementations of NP-Completeness Theory forMinesweeper Problems. From a sample problem of Minesweeper, Flood-Fill Algorithm is required for completing the non-mine space. For solving the problem,**

***Index Terms*–NP-Completeness Theory, Minesweeper, Flood-Fill**

I. INTRODUCTION

Minesweeper has its origins in the earliest mainframe games of the 1960s and 1970s. The earliest ancestor of Minesweeper was Jerimac Ratliff's *Cube*. The basic gameplay style became a popular segment of the puzzle game genre during the 1980s, with such titles as *Mined-Out* (Quicksilva, 1983), *Yomp* (Virgin Interactive, 1983), and *Cube*. Cube was succeeded by *Relentless Logic* (or *RLogic* for short), by Conway, Hong, and Smith, available for MS-DOS as early as 1985; the player took the role of a private in the United States Marine Corps, delivering an important message to the U.S. Command Center. *RLogic* had greater similarity to *Minesweeper* than to *Cube* in concept, but a number of differences exist.

Figure 1. Condition if Player Loses

The objective of Minesweeper is to not choose the mine-containing box of a problem. If the box choosed containing a mine, the player will lose and end the game at the same time. If not, the chosen box may contains blank space or number , usually range one until six. If it is a blank space, then the blank space will expand and open other nearby boxes containing blank spaces or even reveal boxes containing numbers. On the other hand, if it is a number, then the box only open the chosen one. The numbers are indicating how many mine in 9x9 radius box unit field. For instance, in a case the number is two, then in the 9x9 radius of the number, two mines, or bombs, are placed randomly. So with this numbers, players can take them as clues for solving the puzzles.

Figure 2. Condition If Player Wins

There are multiple solutions to solve one problem, thus Non-determinisitic is the right word to describe Minesweeper Algorithm in general, specifically for responding to Player control by choosing an unopened box. Then, the solutions will be completed by NP-Completeness principles. NP-Completeness is chosen because it's use non-deterministic options for determine

The aim of this paper is to determine whether NP-Completeness theory satisfies Minesweeper problems, particularly in determining whether the box is a blank space, number or mine, and Flood-Fill Algorithm in expanding of mines, numbers and blank spaces.

II. BASE THEORIES

2.1. Box Matrices

Box matrices is used for creating the field containing boxes. Every element in the matrix containing boxes, for the example representation is as following:

$$\begin{bmatrix} 1 & 2 & *2 & 1 & 0 \\ * & 3 & 3* & 2 & 0 \\ * & 2 & 2* & 2 & 0 \end{bmatrix}$$

Asteriks(*) are the symbol of mines, while zero is the symbol of blank spaces, and numbers are symbolizing the representative number in the game. The mechanism is when player choose a box, the Gaussian Elimination will be used to eliminate chosen box. If the player choose a mine box, there will be loop occured in the game, and stop until Flood-Fill algorithm reach its boundary to end the game. On the other hand if its a zero, there will be Flood-Fill Algorithm for expansion of the blank space to its homologue neighbours.

2.2. Flood-Fill Algorithm

Flood-fill Algorithm is an Algorithm used for determining area connected to a given node in a multi-dimensional array. Also dubbed as Seed Algorithm, usually used for coloring. In this Minesweeper case, Stack-based Flood-Fill Algorithm is used for item expansion, such as numbers, blank spaces, and mines.

Here below is the stack-based step of Flood-Filling Algorithm:

```
Flood-fill (node, target-color, replacement-color):

 1. If target-color is equal to replacement-color, return.

 2. If the color of node is not equal to target-color, return.

 3. Set the color of node to replacement-color.

 4. Perform Flood-fill (one step to the west of node, target-color, replacement-color).

    Perform Flood-fill (one step to the east of node, target-color, replacement-color).

    Perform Flood-fill (one step to the north of node, target-color, replacement-color).

    Perform Flood-fill (one step to the south of node, target-color, replacement-color).

 5. Return.
```

Hopefully, this step is going to obtain best case time complexity, O(n).

2.3. NP –Completeness

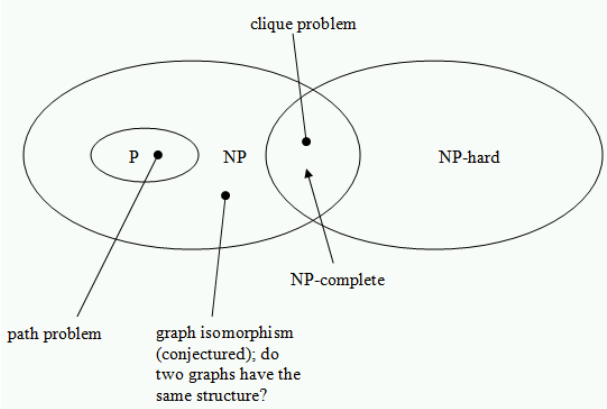

Figure 13. Euler's Diagram of P, NP and NPC Dependencies

An algorithm to NP-Completeness is needed for checking verifications on steps are polynom-oriented algorithm. Example of NP-Completeness can be used to solve Integer Knapsack Decision problem. In order, to show whether a problem can be solved through NPC can be done by following these steps :

1. *Show whether X is a member of NP.*
2. *Choose instance, say Y, from NPC random problems*
3. *Show an algorithm in polynomial time for*

$$transforming\ Y\ become\ X.$$

Flood-Filling Algorithm is one of good sample for this. In this case, X is the node will be colored. X is the member of NP, because to coloring the nodes have many options that resulting the solving style is non-deterministic. Choose the blank node as Y, then the Y become transformed into X.

And then the solution of the puzzle. The selected box undoubtly non-deterministic, because player are not ought to choose the same box in every game. The selection listener connect the box, which is analogue to X, and the Y is the positions of the mines.

2.4. The Items

In this game, containing many items to be used in the game

2.4.1. Mines

Mines are the trigger of this game, and must be avoided in order to win the game. However, Mines in the program representated by asterisk (*) mark in the matrix.

2.4.2. Numbers

The numbers, ranged from one to six, is the number of mines sorrounding the number in a 3x3 matrix, while the positions of the number is in the center (2,2) matrix index.

2.4.3. Blank-Spaces

Blank spaces are properties for clearing the boxes on the field. It used Flood-Fill algorithm for the expansion, and reveal any blank spaces within it neighbour. However, the boundary for the expansion has set on the mine, thus the blank space only uncover other boxes containing blank spaces.

2.4.4. Flags

Flags are used to mark boxes, especially when player preference is memorizing the position of the mines. The Flags

```
Function NP-Completeness(n node, IsND :
boolean) -> boolean
Variables
ALGROITHM
For [1..n] do
       If IsND = true then
       Return NP-Completeness(true);
```

The pseudo code described above assumes that every actions' cost is 1. Because the algorithm traverses a 2D matrix, the algorithm will run in the speed of $O(mn)$ with the required space of $O(n)$. A possible improvement is to reduce the space complexity from $O(n)$ to $O(m)$, observing that the algorithm only requires that the previous row and current row can be stored at any one time.

III. IMPLEMENTATION AND EXPERIMENT

3.1. Implementation

For the experiment, an implementation of the Flood-Fill Algorithm has been written in JavaScript. Below are the implementation of the algorithm.

```
index: function(x, y){


// If square is not revealed, is
within boundaries and exists

if(x >= 0 && y >= 0 && x <=
globals.squaresX && y <=
globals.squaresY &&
globals.mineMap[x] !== undefined){


var l = (globals.revealedMap[x][y]) ?
1 : -1;


if(!util.is('revealed', x, y)){


// Add revealed square to the
revealed array

globals.revealedMap[x][y] = 1;


if(globals.mineMap[x][y] !== -1){

// 'remove square', by drawing a
white one over it

var alpha = 0.1,

squareFade = setInterval(function(){

globals.context.strokeStyle =
'white';

globals.context.fillStyle =
'rgba(255,255,255,' + alpha + ')';

util.roundRect(x, y);
```

```
if(globals.mineMap[x][y] !== -1){


alpha = alpha + .1;

if(alpha > 1){

window.clearInterval(squareFade);

}

}, 50);

// jika kotak yang dipilih tidak
dikelilingi mine

}else{

// trigger jika player memilih kotak
yang terdapat mine

var mine = new Image();

mine.src = defaults.mineImg;

mine.onload = function() {

action.revealMines(mine);

};

}

if(globals.mineMap[x][y] === 0){

// membuang seluruh kotak hingga
tersisa yang mengelilingi mine

for(var i = -1; i <= 1; i++){

for(var j = -1; j <= 1; j++){

// looping jika sesama blank space

if(l < 0 && x + i >= 0 && y + j >= 0
&& x + i <= globals.squaresX && y + j
<= globals.squaresX){

action.index(x + i, y + j);

}

}

}
```

3.2. Source Code For Generating Items

A problem with the above implementation of Flood-Fill algorithm is that it have to set boundaries to process.Without that boundaries, the program will return the mines and other items crashing in one box.

To make the way around, an observation of source code reveals that more than half of the characters in the source code are whitespaces. These whitespaces can be safely ignored, reducing the length of the source code string.

Below are the code for generate and reveal boxes with mine;

```
generateMines: function(){

// Untuk setiap kotak

for(var i = 0; i < globals.squaresX;
i++){

globals.mineMap[i] = new
Array(globals.squaresX);

// makin rendah tingkat kesulitan,
makin banyak mine

for(var j = 0; j < globals.squaresY;
j++){

globals.mineMap[i][j] =
Math.floor((Math.random() *
defaults.difficulty) - 1);

if(globals.mineMap[i][j] > 0){

globals.mineMap[i][j] = 0;

}

}

}


action.calculateMines();

},



calculateMines: function() {

var mineCount = 0;

globals.totalMines = 0;

// cek kotak

for(var i = 0; i < globals.squaresX;
i++){

for(var j = 0; j < globals.squaresY;
j++){
```

```
if(globals.mineMap[i][j] === -1){

var xArr = [i, i + 1, i - 1],

yArr = [j, j + 1, j - 1];

/*

Ilustrasi iterasi kotak:

-------------------------

| i - 1 | i | i + 1 |

| j - 1 | j - 1 | j - 1 |

-------------------------

| i - 1 | i | i + 1 |

| j | j | j |

-------------------------

| i - 1 | i | i + 1 |

| j + 1 | j - 1 | j + 1 |

-------------------------

*/

for(var a = 0; a < 3; a++){

for(var b = 0; b < 3; b++){

if(util.is('mine', xArr[a],
yArr[b])){

globals.mineMap[xArr[a]][yArr[b]]++;

}

}

}

globals.totalMines++;

}

}

}

},
```

3.3. Experiment

The test cases for the experiment is taken from several source codes from assignments of a JavaScript course, thus making all test case source codes are written in JavaScript.

1. Test case where player choose the difficulty of the game, whether it is Easy, Medium, or Expert.
2. Test case where the first attempt reveals mine and game over.
3. Test case where the first attempt a blank space, then expanding to neighbouring blank spaces.
4. Test case where the first attempt reveals number, then expanding to neighbouring same number.

The test case (1) serves no control for Flood-Fill Algorithm while the rest serves as the variable for Flood-Fill Algorithm.

IV. ANALYSIS

4.1. Game Analysis

Basically, Minesweeper are the collection of the matrices in a game. The matrices have logical relationships to the listener. If player choose a box and an another, there will be a logical comparison within the game. If the one other is selected, the next step will be different from the first box.

The main difficulties in putting these gadgets together are concern firstlyhow to make other standard gates (such as OR, and so on) out of the onesalready found, and secondly how to cross wires over one another.

Summarizing, to determine whether a given grid of uncovered, correctly flagged, and unknown squares, the labels of the foremost also given, has an arrangement of mines for which it is possible within the rules of the game. The argument is constructive, a method to quickly convert any Boolean circuit into such a grid that is possible if and only if the circuit is satisfiable; membership in NP is established by using the arrangement of mines as a certificate.

4.2. Breaking the Algorithm

The experiments conducted shows that the NP-Theory is a good measurement for solving the puzzle, and Flood-Fill Algorithm for expanding mines, numbers and blank spaces. However, the NP-Completeness can be manipulated to avoid the test case (2) by following these tricks:

1. Adding exception handling in initiating the programs on the source code;
2. Setting conditions for mine in the program

By adding exception, an unwanted case can be aborted by inserted it into exception and throw it when the first box of mine in a first attempt is true. Fortunately, JavaScript is a derivation of Java which is purely Object-Oriented Programming Language. So adding the exception will cause no problems.

However, it can worsen space complexity but increase time complexity at the same time. With exception handling, the algorithm become increased in memory consumption but handle same process with the previous code when not added the handler.

4.3. Algorithm Performance

The time complexity of Flood-Fill algorithm is $O(n)$, where n represents the squares in the game field. However, the hidden constants is small because there are no preprocessing for the algorithm for work and the main loop for matrix traversal consists of a single if-else statement. During the experiment, attempts to measure the speed of algorithm resulted in the speed of 0.002 ms to 0.003 ms on a modern 2,5 GHz processor. The algorithm is fast enough to be implemented on a processing-heavy environment.

The space complexity of NP-Completeness algorithm is $O(n)$, where n represents the mine. While this maximum input length varies over the difficulties, and the best-case of

V. CONCLUSION

The NP-Principles are one of the best solutions to solve Minesweeper Puzzles. There are many methods to solving, notable is Naïve (Brute-Force), but the time and space complexity are worse than Non-Deterministic Polynom Theory.

For the expansion, the Flood-Fill Algorithm is fast and satisfying enough for processing artificial intelligence inside the Minesweeper game. However, the original implementation of the algorithm has a weakness in the high space complexity, especially in Expert difficulty where the Gaussian Matrices are far more complicated.

VI. REFERENCE

[1] Munir, Rinaldi. 2009. *Diktat Kuliah IF2211 Strategi Algoritma*. Program Studi Teknik Informatika ITB.

[2] Arefin, Shamsul Ahmed. 2009. *Art of Programming Contest 2nd Edition*.

[3] Halim, Steven and Halim, Felix. 2013. *Competitive Programming: The New Lower Bound of Programming Contests*.

[4] https://github.com/Joeynoh/HTML5-Minesweeper/

[5] http://stackoverflow.com/questions/1738128/minesweeper-solving-algorithm

[6] http://web.mat.bham.ac.uk/R.W.Kaye/minesw/ordmsw.htm

[7] http://www.minesweeper.info/articles/MinesweeperStatisticalComputationalAnalysis.pdf

VII. ACKNOWLEDGEMENT

This paper is written by Thea Olivia, intended for an assignment in IF2211 Algorithm Strategy course in Institut Teknologi Bandung at 2015. The writer sincerely apologized for grammar mistakes and will be get better on the next opportunities.

The writer wants to express her gratitude first to Our Father in Heaven for all his grace and guidance in writing this paper, and then to Dr. Ir. Rinaldi Munir, M.T. and Nur Ulfa Mauliadewi, as her lecturers on the course. The writer also express her gratitude to the friends and the family of HMIF ITB (*Himpunan Mahasiswa Informatika ITB*) who had given their best for helping the idea of this paper.

The writer also very grateful for GitHub user Joeynoh for such an inspiration.

VIII. NOTES

A repository containing the sample source code for the experiments, along with the test cases and a digital copy of this document is available on the Internet at https://github.com/teaolivia/MySweeper.

IX. DISCLAIMER

I hereby declare that the paper I wrote is my own work. It is not a copy nor a translation of someone else's paper, and not a plagiarism.

Bandung, 5th May 2015

Thea Olivia
NIM. 13511001