Greedy Algorithm and String Matching in Battleship Game Strategy Using Probability Density Matrix

Faiz Ilham Muhammad (13511080)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
faizilham@students.itb.ac.id

***Abstract*—Battleship game is a classic mathematical game that use logical reasoning and some luck to play. There are some algorithm to use for playing battleship game efficiently, one of them is a greedy algorithm based on probability density matrix of current game state. Aside of the greedy property, the algorithm also use a string matching algorithm to generate the probability density matrix.**

***Index Terms*— battleship, greedy, string matching, probability density**

I. Introduction

Battleship is one of many popular classic pencil-and-paper game. The game objectives is to sink all opponent's battleships with minimum numbers of shot as possible. Aside of the probabilistic nature of the game, it is possible for a player to logically deduce which area to shoot to maximize the probability of hitting an opponent ship. This paper explains a greedy algorithm using probability density matrix for playing battleship efficiently and string matching algorithm for generating the probability density matrix.

II. Battleship Game Definition & Terms

Battleship game, as explained before, is a classic pencil-and-paper guessing game. This game is played by two player like chess and othello. Each player's objective is to sink all opponent's ship by "shooting" it.

2.1. Gameplay

Battleship uses four grids, usually a 10x10 square, to play. Each player are given two grids, one for recording his/her shot to the opponent's ships and one for recording his/her opponent's shot to his/her ships.

Before the game starts, each player must place their ships on the grid. Each ship may not overlaps other ships, but may touch other ship. There are five kinds of ships on the game, with each ship's dimension is 1xL where L is its length.

Table 2.1. Types of ship in battleship game

Name	Length
Aircraft Carrier	5
Battleship	4
Submarine	3
Cruiser	3
Destroyer	2

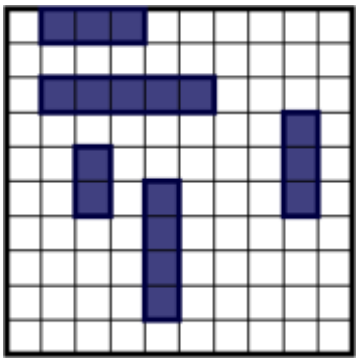

Figure 2.1. Ships positioning example

Everytime a player get a turn, he/she may shoot a certain tile on the opponent's grid. The opponent must reply with a "hit" if the shot hit a ship or a "miss" if not. If every tile that belongs to a ship was hit, the ship is sink and the opponent must also declared that the ship sinks, i.e. "my cruiser sinks!". The last player who still have a ship will be the winner.

III. Basic Algorithm Theories

3.1. Greedy Algorithm and Principle

A greedy algorithm is an algorithm that employs greedy principle "take what can you take now"; that is to optimize global solution by taking current optimal solution. Since greedy algorithm take current optimal solution, the global solution it creates may be not optimal and only close to the real optimal solution.

Greedy algorithms are consist of several elements:

1. Candidate Set
   Set of elements that may be a part of the solution. Usually, any object that are related directly to the problem may be regarded as a member of candidate set.
2. Solution Set
   Set of selected elements from the candidate set to

build up the solution.

3. Selection Function
   Function that is used to select an element of candidate set. Selection function typically selects element that optimize current state.
4. Constraint Function
   Function that is used to check whether the selected element satisfy certain constraints; the selected element will be included if it satisifies the constraints.
5. Objective Function
   Funtion that is used to determine the optimization of the solution.

All greedy algorithm have a generic scheme:

1. Initialize an empty solution set
2. Select an element from the candidate set using the selection function
3. Remove the element from candidate set
4. Check whether the solution set will satisfy the constraints if the selected element is added into it using constraint function. If yes, add the selected element to solution set
5. Check whether the solution set is complete. If yes, finish. If not, repeat to step (2).

3.2. String Matching Algorithm

A string matching algorithm is an algorithm to check whether a certain pattern is exist within a text. There are many kinds of string matching algorithm, some of them are brute-force string matching and KMP (Knuth-Morris-Pratt) algorithm.

3.2.1. Brute-Force String Matching Algorithm

The brute-force string matching algorithm is defined as below:

1. Align the pattern with the beginning of the text
2. For each character in pattern, check the whether it is the same with the character in the text. If all character matches, finish. If not, proceed to step (3).
3. Shift the pattern by one character, so that the first character of pattern aligned with the next character of text. If the number of remaining character of text >= length of pattern, repeat to step (2).

3.2.2.KMP (Knuth-Morris-Pratt) Algorithm

The KMP algorithm is an improvement of the brute force algorithm. The KMP algorithm allows shifting pattern by more than one character depend on the border function. The border function of a pattern is defined as the maximum length of the same prefix and suffix of the pattern. The KMP algorithm is defined as below:

1. Calculate border functions of the pattern
2. Align the pattern with the beginning of the text
3. For each character in pattern, check the whether it is the same with the character in the text. If all character matches, finish. If not, proceed to step (4).
4. Shift the pattern by *len – b(len)* character(s), with len the length of matching pattern and b(len) the border function of pattern for length len. If the number of remaining character of text >= length of pattern, repeat to step (3).

IV. BATTLESHIP GAME ALGORITHM USING PROBABILITY DENSITY MATRIX

There are some algorithm that can be used for playing battleship game, one of them is using a probability density matrix. A probability density matrix is a matrix that describes the probability of a tile in the grid contains a part of a ship. By using this, it is possible to make a greedy algorithm that select a tile in the grid that have the highest probability. However, because of the game's probabilistic nature, it is impossible to make any algorithm that always yields most optimum solution; that is to shoot all ships without miss.

The greedy algorithm elements are:

1. Candidate Set
   Every tile on the opponent's grid that haven't been shot.
2. Solution Set
   The selected tile to be shot so that the number of shot to sink all ships minimal.
3. Selection Function
   Select a tile that have the highest probability value based on current calculated probability density matrix. The calculation algorithm will be discussed more on Section V.
4. Constraint Function
   Since all tile in candidate set is valid, no constraint function is needed
5. Objective Function
   Minimize the number of shots used to sink all ships

The algorithm is defined as below:

1. Set the probability calculation mode to "hunt" mode (more on Section V)
2. Calculate the probability density matrix.
3. Select a tile which haven't been shot and have the highest probability
4. Check opponents reply. If current mode is "hunt" and the reply is "hit", change mode to "target". Else if the current mode is "target", the reply is "sink", and there is no more "hit" tile, change mode to "hunt".
5. Check whether all opponent's ship is sunk. If yes then finish. If not repeat to step (2).

After implementating the algorithm (including the matrix calculation algorithm), the following are some of the program results. Notice that the initial ship positions are randomized.

```
F:\kuliah\Stima\paper>python battleship.py

shot: 38
miss: 21
hit rate: 44%

F:\kuliah\Stima\paper>python battleship.py

shot: 42
miss: 25
hit rate: 40%

F:\kuliah\Stima\paper>python battleship.py

shot: 45
miss: 28
hit rate: 37%

F:\kuliah\Stima\paper>python battleship.py

shot: 34
miss: 17
hit rate: 50%
```

Figure 4.1. Some result of the battleship program in Python. Execution time under 1 second for each run.

```
3 3 : hit
1 3 : sink 3
5 9 : miss
9 5 : miss
1 5 : miss
1 8 : miss
2 6 : miss
2 9 : miss
3 10 : miss
6 10 : miss
9 2 : miss
9 9 : miss
10 6 : miss
3 5 : miss
4 1 : hit
4 3 : miss
5 1 : hit
6 1 : sink 3

shot: 43
miss: 26
hit rate: 39%
```

Figure 4.2. Battleship program result (truncated) with information of each shot done by the program.

V. Probability Density Matrix Calculation Algorithm Using String Matching

Probability density matrix is basically a matrix that describe each tile chance of having a part of a ship. One of the method to calculate it is by enumerate all possible position of every ship given the current state of the grid. If a ship can be placed at a certain position, every tile that coincide with the ship will have its probability value incremented by one.

Since all ships have a dimension of 1xL with L the length of ship, the enumeration process can be decomposed into a string matching problem: given character "0" represents a tile that haven't yet been shot and "1" tile that have been shot, search all position of pattern consists L character(s) "0" on a certain row or column.

The general scheme of the calculation is defined as below.

1. For each ship that haven't yet sunk, do the following.
2. Let L as current ship length
3. For each row and column in the grid, do the following.
4. Find the position of pattern consisting L character of "0" on current row / column, using a certain string matching algorithm.
5. If there is no match, continue to next row / column.
   If there is a match, increase probability of each covered tiles by one, shift the pattern to the right and repeat to step (4).

5.1. Calculation Mode

In calculating probability density matrix, there are two kind of mode to use, "hunt" and "target" mode. In hunt mode, the program "hunts" a ship by shooting tiles until a shot hit. At this point, the program interested in which tile that have highest probability of having a part of ship. Hence, the matrix is calculated by enumerating all ship position on grid, as explained before.

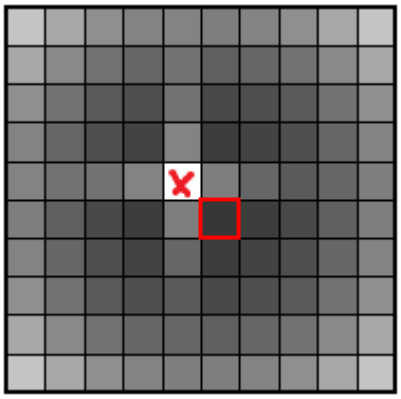

Figure 5.1. The matrix calculated in hunt mode. Notice the red X marks the previous miss shot.

In target mode, on the other hand, the program previously has hit a ship, and more interested in which tile that have highest probability of having a part of the *same* ship rather than of any ship. Consequently, there are some modification of the previous calculation algorithm.

First, instead of two, there are three types of tiles: "0" is a free tile, "1" is a miss shot tile, and "2" is a hit shot tile. The algorithm will find a pattern of character "0" or "2" of length L on a certain row or column. If there is a match on certain position, every tile that covered by the pattern will have its probability increased by number of "2" in the pattern.

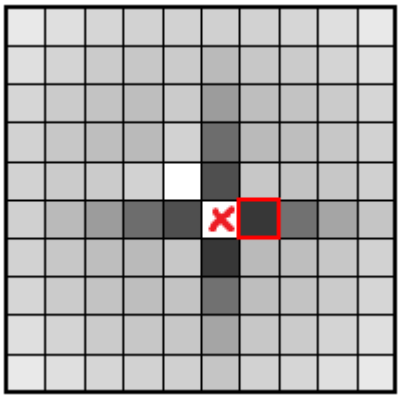

Figure 5.2. The matrix calculated in target mode. Notice the red X marks the previous hit shot

The general scheme of the calculation is defined as below.

1. For each ship that haven't yet sunk, do the following.
2. Let L as current ship length
3. For each row and column in the grid, do the following.
4. Find the position of pattern consisting L character of "0" or "2" on current row / column, using a certain string matching algorithm.
5. If there is no match, continue to next row / column.
   If there is a match, let K as the number of character "2" in matched string. Increase probability of each covered tiles by K, shift the pattern to the right and repeat to step (4).

5.2. Implementation

5.2.1. Brute-Force

The brute-force implementation for both calculation mode are quite straight forward:

1. Visit the first tile on current row / column
2. Read L tiles to the right (if row) or to the bottom (if column) from the currently visited tiles.
3. Check whether the tiles have a "1" or not. If not, increase the probability as explained before.
4. Visit next tile if available and repeat to step (2)

As for the complexity, brute-force implementation will have complexity of O (pn) for each row and column, where p is the average length of ship and n is the number of tiles in a row / column.

5.2.2. KMP

The KMP implementation needs some tweaks for it works. Since there is some pattern, especially for target mode calculation, is not exact, the border function cannot be pre-computed. Instead, the algorithm will check or jump character depends on previous result:

1. Let Prev, the success state of previous check, assigned as false
2. Visit the first tile on current row / column
3. Read L tiles from the currently visited tile
4. If Prev is true, skip checking the first L-1 tiles from the current tile
5. Check whether the tiles have a "1" or not.
   If yes, set N as the position of character "1" relative from current tile
   If not, increase the probability as explained before starting from the current tile, and let N = 0
5. Visit next N + 1 tile if available and repeat to step (2)

As for the complexity, KMP implementation will have complexity of O (n) for each row and column, where n is the number of tiles in a row / column.

The following is the number of checks done by both implementation on some cases.

```
F:\kuliah\Stima\paper\test>pytho

brute-force:  2480 checks
kmp        :  1000 checks
```

Figure 5.3. Number of checks on the case of no shot have been fired before.

```
F:\kuliah\Stima\paper\test>python

brute-force:  2340 checks
kmp        :  984 checks
```

Figure 5.4. Number of checks on the case of 4 shots have been fired before at (0,0), (5,0), (0,5) and (5,5)

V. CONCLUSION

After doing the implementation of both brute-force and KMP algorithm, it can be concluded that the KMP algorithm is better than the brute-force algorithm for generating the probability density matrix. On the other hand, generating probability density matrix can take much computational time if it must be done for every turn. The string matching algorithm can be improved by adding dynamic programming and memoization on it. As for the greedy algorithm, it can be concluded that the greedy implementation is relatively simple to implement yet still give a quite decent result.

REFERENCES

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

[2] http://www.datagenetics.com/blog/december32011/, *Battleship*. Accessed at December 19th 2013.

[3] http://boardgames.about.com/od/salvo/a/salvo_rules.htm, *Salvo – Battleships – Rules*. Accessed at December 19th 2013.

[4] http://www.ics.uci.edu/~eppstein/161/960227.html, *Knuth-Morris-Pratt Algorithm*. Accessed at December 19th 2013.

[5] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Greedy/greedyIntro.htm, *Greedy Introduction*. Accessed at December 19th 2013.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Desember 2013

Faiz Ilham Muhammad
13511080