Comparing Methods for Solving Kuromasu Puzzles

Leiden Institute of Advanced Computer Science

Bachelor Project Report

Tim van Meurs

Abstract

The goal of this bachelor thesis is to examine different methods for solving a Kuromasu puzzle, a Japanese binary determination puzzle with the goal of finding all the black and white cells in a puzzle to form a unique solution. The Kuromasu puzzle can be compared to the Nurikabe puzzle, which was invented by the same company as the Kuromasu puzzle. The different methods will be compared on success-rate and speed. Finally a conclusion will be given on which is the best of the compared methods for solving a puzzle using a computer and some possible fields for further research will be suggested.

December 16, 2008

Contents

1 Introduction

We will start by describing the Kuromasu puzzle and the rules in Section 2. Section 3 will describe the working of the bruteforce method and explain the complexity of bruteforce solving a Kuromasu puzzle. Section 4 will give a description of the solving of a Kuromasu puzzle using logical rules, where each specific rule will be described in detail. Section 5 will give a description on the working of a Genetic Algorithm and the way it can be adapted to be used to solve Kuromasu puzzles. Section 6 will give a description on a number of special cases that can occur in puzzles which may need special solutions but may also have default solutions. Section 7 will give an overview of the results of solving Kuromasu puzzles using the three different methods mentioned. Finally, Section 8 will give a conclusion on which of the methods is most successful at solving Kuromasu puzzles and suggests a number of possible fields for further research.

2 Kuromasu

2.1 General

Kuromasu [1] is a fairly new binary-determination logic puzzle published by the Japanese company Nikoli [4], who also created the immensely popular Sudoku puzzle. The name Kuromasu is translated by Nikoli to "Where is Black Cells". This is a fairly straightforward translation of its Japanese name, as Kuro stands for black and Masu stands for square in the Japanese language.

2.2 The rules

Kuromasu is played on a rectangular grid. Some of these cells have numbers in them. Each cell may be either black or white, though they all start out as being white. The object is to determine what type each cell is.

The following rules determine which cells are which:

1 Each number on the board represents the number of white cells that can be seen from that cell, including itself. A cell can be seen from another cell if they are in the same row or column, and there are no black cells between them in that row or column.

2 Numbered cells may not be black.

3 No two black cells may be horizontally or vertically adjacent.

4 All the white cells must be connected horizontally and/or vertically through other white cells.

The terminology above will be used throughout this paper with the following definitions:

1 white cell: cell in the puzzle that should be white. *(displayed as a white cell)*

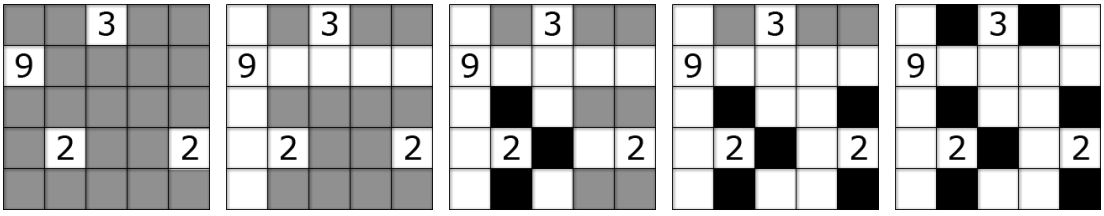
2 black cell: cell in the puzzle that should be black. *(displayed as a black cell)*

3 empty cell: cell that is yet to be decided to be white or black. *(displayed as a grey cell)*

4 numbered cell: cell that is numbered, and thus can be counted as white. These are given at the start of a puzzle. *(displayed as a white cell with a number in it)*

2.3 Example puzzle

To further explain the above-stated rules an example puzzle and a possible stepwise solution is shown below (Figure 1).



(a) Start (b) Step 1 (c) Step 2 (d) Step 3 (e) Finished

Figure 1: Example of a puzzle

How these steps are reached is further described in the Logical Rules (Section 4).

3 Bruteforce

3.1 Working of bruteforce

A simple way of solving a puzzle is by trying every single option possible. Obviously this would take too long to do with a pencil and paper. Luckily, computers are a lot faster and are therefore the way to go for doing such big calculations.To be able to explain the method of bruteforce for the puzzle we will enumerate the empty cells as shown below (Figure 2).

To solve a Kuromasu puzzle through bruteforce, the tactic is fairly straightforward. We make cell 1 a black cell and check whether the puzzle is solved. If this is not the case, we move the black cell to cell 2, and check again. This can be repeated for all empty cells in the puzzle. If this doesn't solve the puzzle, a second black cell is introduced. They are placed on cells 1 and 2. If this doesn't solve the puzzle, the black cell on cell 2 is then moved to cell 3, etc., until the last empty cell is reached. The black cell on cell 1 is then moved to cell 2, and the other cell is placed on cell 3. This process can be continued until

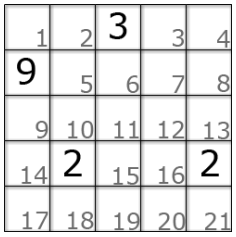


Figure 2: Augmented puzzle (enumerated cells in gray)

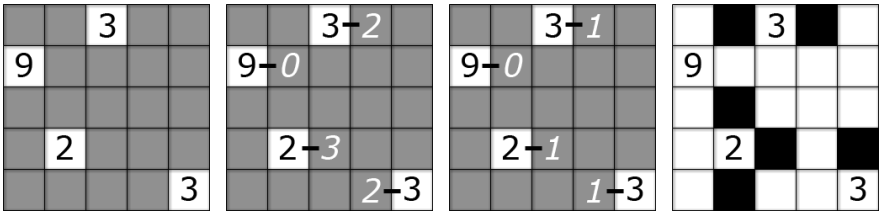
both cells are on the last 2 empty cells of the puzzle. If no solution is found, a 3rd black cell is introduced. This keeps going until a solution is found.

3.2 Amount of black cells

As we have seen, the bruteforce method starts by placing a single cell in all possible locations in a puzzle, followed by 2 black cells, etc. This is not very efficient, so we look for a method to find the minimum amount of black cells in a puzzle. We start by finding, for each numbered cell, the minimum amount of black cells needed to finish the cell. This means, all 4 sides are checked whether the side of the puzzle might be reached by that cell by only placing white cells, with a maximum of the cells number minus 1 (for the cell itself). If it does, that side doesn't count. By doing this for all 4 sides, we end up with the amount of sides which need a black cell for the numbered cell to be finished. An example puzzle with the amount of black cells necessary for each numbered cell is shown in the figure below (Figure 3).

After doing this for all numbered cells, we go through all numbered cells and see whether a black cell might be combined with the black cell of another numbered cell. This is simply done by marking all cells horizontally and vertically from a numbered cell, with a maximum distance from that cell, equal to the cell's value. This is done for all numbered cells except the one that is being checked. By now checking all 4 sides from that cell to a distance equal to its number, we can see whether that side of the cell might combine a black cell with another numbered cell. We now take the number of sides that might have a combined black cell and substract that amount from the number of black cells necessary to finish the numbered cell, described above. The remaining number is the amount of black cells that will be unique for that particular numbered cell, and is therefore certain to be placed in the puzzle without being combined with another numbered cell. The number described here is shown in the example below (Figure 3).

We can now see, that the puzzle below will be calculated to have a minimum of 3 black cells, while the total amount necessary to finish the puzzle is 6.



(a) Empty puzzle

(b) Amount of black cells needed to finish the numbered cell, shown in italic font, next to the numbered cell

(c) Amount of black cells needed, after checking possible combinations, shown in italic font, next to the numbered cell

(d) Solution

Figure 3: Example of a puzzle

3.3 Complexity of bruteforce

A major downside of the bruteforce method is the amount of options that will be checked. The amount of options that will be checked can be calculated through a simple formula: $options = \frac{n!}{r!(n-r)!}$ where n is the amount of cells that is not numbered and r is the amount of black cells. For a puzzle with sides of length 9 and 11 numbered cells, n would be 70. If we now calculate the amount of options for 3 black cells we get $\frac{70!}{3!(70-3)!} = 54{,}740$. For 4 black cells we get $\frac{70!}{4!(70-4)!} = 916{,}895$. As we can see, the number of options grows fast as the amount of black cells increases. How this affects the implementation of bruteforce in a computer program can be see in the subsection with results (Section 7.1).

4 Logical rules

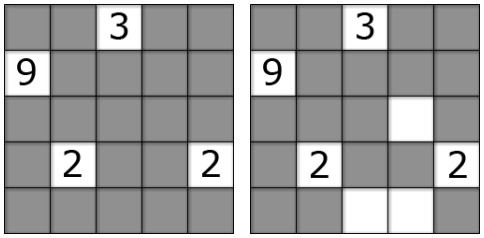
These are the logical rules that are used to solve (part of) the puzzle. They are all based on a system which could be applied by a person using a pencil and paper.

4.1 Method 0

This method is only used once, in the very beginning of the solving process. It runs through the puzzle to find cells that can not be "reached" by any numbered cells. If a numbered cell has the value 3, it can only "reach" cells that are 3 cells away from itself in a horizontal or vertical direction. As the cell itself is white and it can see a maximum of 3 white cells, this means the third cell can be black, but there is no direct influence on the fourth cell. A cell in the puzzle that can not be "reached" by any numbered cell can be made white without consequences. The preceding statement is only true for puzzles with a unique solution. If

a puzzle has multiple solutions, these cells might, in some cases, end up being either black or white. Making the cell white will never prohibit the program from finding a solution, for non-unique puzzles, it might limit the amount of solutions that can be found.

Making the cells white will help the solving process as it can help "reaching" white cells from other white cells, which is useful for method 6 (Section 4.7). The cells marked white in the example (Figure 4) are cells that can not be reached by the numbered cells, and can therefore be made white. To find the cells described above, a temporary copy of the puzzle is made in which all cells the can be "reached" from the numbered cells are marked. After this is done for each numbered cell, the cells that are not marked cannot be "reached" by a numbered cell and are therefore made white.



(a) Empty puzzle (b) Puzzle after applying Method 0

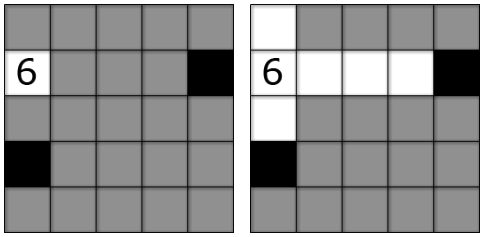
Figure 4: Example of Method 0

4.2 Method 1

This method applies rule number 1: every numbered cell on the board must see that number of white cells. Take a number on the board and count the number of cells to one side until a black cell or the side of the puzzle is reached. Repeat this for all sides. Add 1 to the total of all 4 sides for the cell of the number itself. If the total matches the number of the cell, all cells that can be seen from this cell will have to be white cells. An example of this method is shown in the figure below (Figure 5). This process is repeated for all numbered cells in the puzzle.

4.3 Method 2

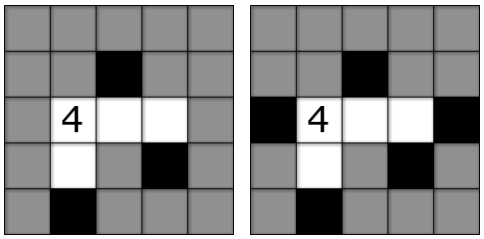
This method also applies rule number 1: every numbered cell on the board must see that number of white cells. This method does approximately the same as method 1, except in this case, only the connecting white cells are counted to all sides. If the number of white cells match the number in the cell, the cells next to the last white cell on each side will have to be a black cell, or the side of the puzzle. An example of this method is shown in the figure below (Figure 6). This process is repeated for all numbered cells in the puzzle.



(a) Numbered cell can see the same amount of empty cells as the number it has

(b) All cells that can be seen will be made white

Figure 5: Example of Method 1



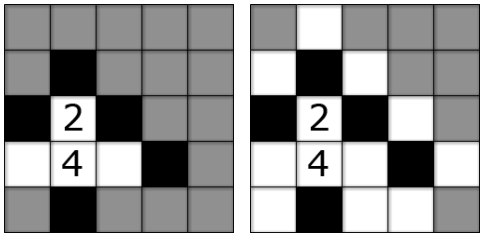
(a) Numbered cell can see the same amount of white cells as the number it has

(b) The rows and colums that don't end with a black cell or the border of the puzzle will be enclosed by black cells

Figure 6: Example of Method 2

4.4 Method 3

This method applies rule number 3: no two black cells may be horizontally or vertically adjacent. A scan is done through the entire puzzle. For each black cell, all 4 sides are checked. If one of the cells adjacent to the black cell is empty, it is made white, as no black cells can be adjacent to other black cells. An example of this method is shown in the figure below (Figure 7). This process is repeated for all black cells in the puzzle.



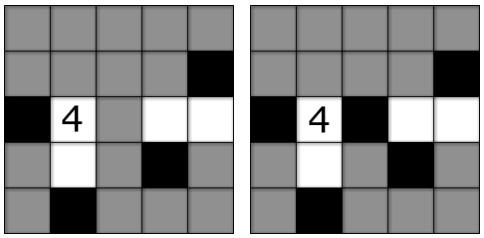
(a) Black cells may not be vertically or horizontally adjacent to another black cell

(b) All cells vertically and horizontally adjacent to a black cell will be made white

Figure 7: Example of Method 3

4.5 Method 4

This method applies rule number 1: every numbered cell on the board must see that number of white cells. This method counts all the white cells that can be seen from a numbered cell. If on one side a row/column of white cells is interrupted by an empty cell, this cell is counted to be white. If the total number of white cells is now larger than the number in the numbered cell, this cell will have to be black. As the example (Figure 8) shows, the number 4 can see 2 white cells, one below, and the cell itself. If the cell to the right of the 4 would be a white cell, it would see 5 white cells, which is too much. Therefore, the cell to the right of the 4 must be a black cell.



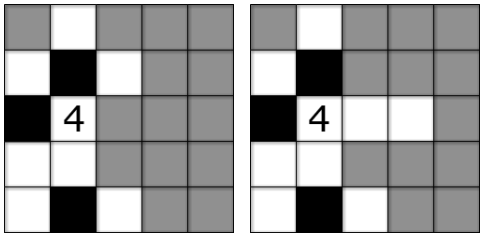
(a) The numbered cell can only see 4 white cells

(b) The cell to the right of the numbered cell must be made black, else the numbered cell will see too many white cells

Figure 8: Example of Method 4

4.6 Method 5

This is a method which applies rule number 1. If a numbered cell has 3 sides that are "fixed", the 4th side is known. A side is fixed if it contains a row of white cells, followed by a black cell, or the side of the puzzle; it can also just be a black cell or the side of the puzzle, without a row of white cells in between. If this happens to 3 sides of a numbered cell, the 4th side can be calculated. Shown in the example (Figure 9), the numbered cell 4, is "fixed" on 3 sides, as none of the sides can be changed. Only the side to the right of the 4 can be changed. It currently sees 2 white cells, so the right will have to be 2 more white cells followed by a black cell.



(a) 3 sides of the numbered cell are "fixed"

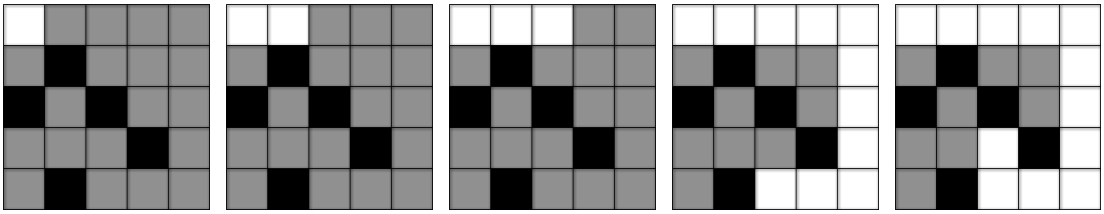
(b) White cells are added to the remaining side

Figure 9: Example of Method 5

4.7 Method 6

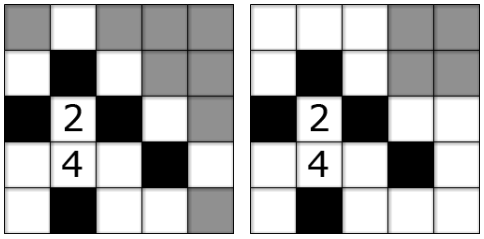
Method 6 is the only method that applies rule 4: all the white cells must be connected horizontally or vertically. A system was implemented that can test whether all white cells are connected. This is done by making a copy of the puzzle with only the black cells in it. Next, a cell is selected and marked, this cell will than check whether the cell to the right is black or marked, if it isn't black or marked, it is marked and this cell will do the same thing. If the cell to the right is black, marked, or the side of the puzzle is reached, the cell below it will be checked and marked if it is white and not yet marked. The same is done for left and up. If a cell has 4 adjacent cells which are either black or marked, it will go back to the cell that marked the current cell, and continues with checking where it stopped. An example of how the method "walks" through the puzzle is shown in the figure below (Figure 10). Because all 4 sides are checked, all adjacent white cells will be found, and all adjacent cells of the adjacent cells will be found, etc. After all cells that can be reached from the original cell are marked, they are counted. If the amount of marked cells is the same as the amount of white cells that was started with, all white cells can be reached horizontally or vertically. To apply this in a puzzle is fairly straight-forward. A cell

in the puzzle is made black and the above method is tested, if making a cell black causes the puzzle to not be able to reach all white cells horizontally or vertically, this means the cell will have to be white. An example of this method is shown in the figure below (Figure 11).



(a) Start (b) Step 1 (c) Step 2 (d) Step 11 (e) Step 12

Figure 10: Method 6; "walking" through a puzzle, marking all empty cells



(a) A couple of white cells will be "cut off" from the other white cells if certain empty cells will become black

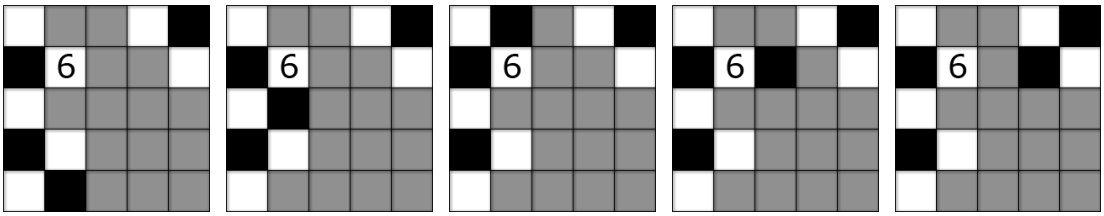
(b) The empty cells that will "cut off" white cells from other white cells are made white

Figure 11: Example of Method 6

4.8 Method 7

This method again applies rule number 1: every numbered cell on the board must see that number of white cells. It does this by a fairly simple method, simply trying a small amount of possibilities. A numbered cell which is not done yet is taken. For each side the method will try all empty cells, by making them black and finding out whether the amount of cells that can be seen from the number is smaller than, larger than or equal to the number itself. If all the cells that are empty and can be seen by the numbered cell have been tested, the results are checked. If just one empty cell has a result of at least as many white cells as can be seen from the numbered cell, and all the other empty cells have a lower number of

white cells that can be seen, the single cell with at least as many cells will have to be a black cell. An example of this method is shown in the figure below (Figure 13).



(a) This would block the white cell in the bottom-left corner

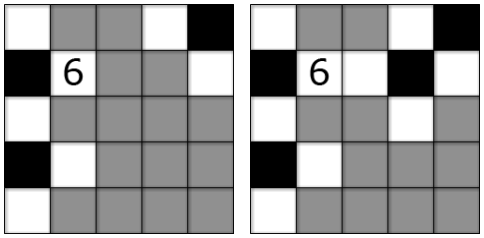
(b) This would block the white cell to the left, and leave too little cells for the numbered cell to see

(c) This would block the white cell in the top-left corner

(d) This leaves too little cells for the numbered cell to see

(e) Only possible position for the black cell

Figure 12: Method 7; trying all possible positions for a black cell



(a) A possible situation where Method 7 could be applied

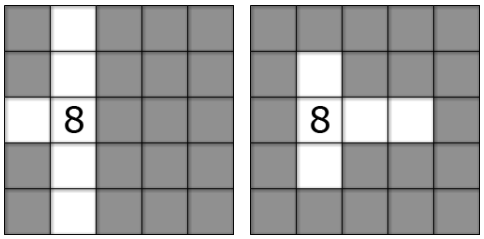
(b) The only possible option that won't disrupt the puzzle is marked black, as this is the only place the black cell can be placed

Figure 13: Example of Method 7

4.9 Method 8

Method 8 is particularly useful for large numbers. A large number can give a lot of information without even knowing any other white or black cells. This is done through a fairly simple trick. All empty and white cells that can be seen on 3 sides are counted. For the example below (Figure 14) this would be 5 for the left, up and down side; 1 is added for the cell itself, making it 6. This would mean that if all cells that might be seen from the

numbered cell to the left, top and bottom are all white, it will only see 6 cells. This means that at least 2 cells to the remaining right side will have to be white cells. This process is repeated for the other 3 sides to come up with the example shown below (Figure 14). This method will also work if there are black cells in the puzzle.



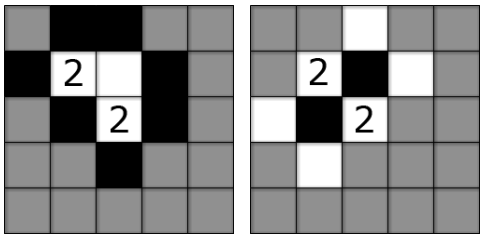
(a) White fields visible to top, bottom and left

(b) Result after checking all 4 sides

Figure 14: Example of Method 8

4.10 Method 9

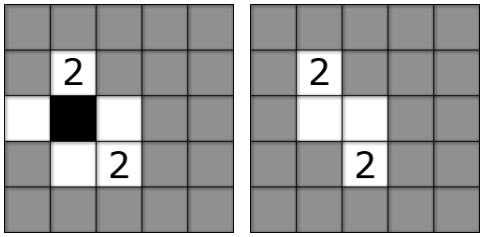
Using all the methods described before, two very common situations will not be covered. This method will make sure that these situations will also be solved. Situation 1 is as follows; 2 cells which are diagonally connected while both have the number 2. In this case, the cells that are connected horizontally and vertically to both cells must be made black. The figures below (Figure 15) shows why. Situation 2 applies when two cells with the number 2 are a knight's move away from each-other. In this case the cells between the numbered cells will be white. The figures below (Figure 16) shows why.



(a) Adjacent black cells due to bad solution

(b) Therefore, both cells adjacent to both numbered cells will have to be black cells

Figure 15: Method 9, situation 1



(a) If one of the cells in between the numbered cells is made black, the other cell will see too many white cells

(b) Therefore, both cells in between the numbered cells will have to be white cells

Figure 16: Method 9, situation 2

4.11 Guessing the next step

If all the methods described before do not fully solve the puzzle, a final step is applied. This step involves making an empty cell black. The empty cell is selected through a fairly simple system. The difference between the amount of white cells that can be directly "seen" by a numbered cell and the number of the cell itself is checked. All numbered cells that do not "see" sufficient white cells are selected. From these numbered cells, all four sides are checked to see whether they are "ending" (they can only see white cells followed by a black cell or the side of the puzzle, or they are adjacent to a black cell or the side of the puzzle). The amount of sides that are "ending" (1,2 or 3) is stored. Next, the list of all numbered cells that were selected will be sorted descending on the amount of "ending" sides. If the amount of "ending" sides is equal, they are also sorted descending on the difference between the white cells that can be directly "seen" from the numbered cell and the number of the cell itself. This will ultimately give us a list of numbered cells which is sorted from cells that are close to being solved (e.g., 1 or 2 open sides and a small amount of white cells missing) to cells that are less close to being solved (e.g., 2 or 3 open sides and a large amount of white cells still missing). We can now take the first cell from the list (which is most likely to be solved easily) and check all sides for empty cells where we might place a black cell. We take the first empty cell we find and make it a black cell. Now, we have a puzzle that has one extra black cell that we can apply all described methods (1 through 9) to. After applying all methods, 3 possible situations can arise:

1 The first situation is fairly straight-forward; by making the empty cell black, the puzzle could be solved using the methods.

2 If in the second situation one of the methods runs into problems, e.g., trying to make a white cell black. This means that the cell that was made black in the

first place is not supposed to be black, so it will be changed back to be empty. Another empty cell will be selected and the process will be repeated.

3 The final situation arises when all the methods have been applied and the puzzle is still not fully solved. In this case, the cell will be made empty again and the next cell will be made black and the process will be repeated.

For the more complicated puzzles, this will not always come up with a solution as situations 2 and 3 can apply for all empty cells. In this case the process is started again from the first empty cell, only in this case, if situation 3 applies, a second empty cell will be found and made black to start the process again from there. Summarizing; if no solution is found using the methods, the program will try to find a solution by making one guess, if no solution is found, a solution is sought by making two guesses, etc.

5 Genetic Algorithms

Genetic Algorithms are a particular class of evolutionary algorithms [2, 6, 3] (also known as evolutionary computation) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination). Although Genetic Algorithms are mostly used to approximate solutions, they can also be used for exact solutions, like in the case of Kuromasu puzzles. In the following sections a step-by-step explanation of the Genetic Algorithm will be given, with examples referencing the algorithm used to solve Kuromasu puzzles.

5.1 Initialization

The algorithm starts by initializing a complete population of 5,000 possible solutions. The original empty puzzle is taken and the amount of numbered cells is counted. This number is multiplied by 1.5 to come up with the number of black cells that will be placed in the puzzles of the population (It is found that nearly all Kuromasu puzzles have a minimum of 1.5 black cells for each numbered cell). Every puzzle is now randomly filled with the amount of black cells counted before. The first generation has been formed.

5.2 Selection

In the selection process every puzzle from the population is graded, the lower the grade, the better the puzzle. A couple of rules are used to determine this grade, or fitness-function. The first rule is the most important, it checks all numbered cells in the puzzle and counts the number of white cells can be seen from that cell. The absolute difference between the numbered cell and the amount of white cells is added to the grade. If a numbered cell with a 4 sees 7 white cells, 3 is added to the total grade, if it sees only 2 white cells, 2 is added

to the total grade. By doing this for all numbered cells, a global idea of how close a puzzle is to being solved is given.

For a numbered cell c, let $v(c)$ be the value of the cell and $white(c)$ the number of white cells the cell can "see" in a particular element from the population. Define:

$$F_1 = \sum_{cell\ c} |v(c) - white(c)|$$

A second rule determines how many black cells are horizontally or vertically adjacent to each other. For every 2 black cells that are adjacent a penalty of 2 is given. Written in formula, this would be:

For a black cell b, let $black(b)$ be the amount of black cells vertically and/or horizontally adjacent to cell b. Define:

$$F_2 = \sum_{black\ cell\ b} black(b)$$

The fitness of this potential solution is now defined as $F_1 + F_2$. After both rules are applied on all puzzles in the population a selection can be made of puzzles that will be used to form the next generation. In order to do this, two possible methods are implemented. (Explanation given is for a selection of 100 puzzles.) The first method is very straightforward, the 100 puzzles with the lowest grades are selected. The second selection is called "tournament selection". This involves splitting the complete population into 100 evenly sized, randomly formed groups of puzzles. After doing so, the puzzle with the lowest grade from each group is selected to form the new population.

5.3 Crossover

After a selection is made from the current generation, a new generation can be formed. This is done by taking the selection and using crossover to form new puzzles. Crossover is a method where elements from a puzzle are combined with elements from another puzzle to form a new puzzle. This program uses 3 methods of crossover. The first combines the left half of a puzzle with the right half of another. The second combines the top half of a puzzle with the bottom half of another. The last combines the uneven rows of one puzzle with the even rows of another. Every method is used as often as the others. A typical selection will consist of (1/6)th of the total amount of the population, the other (5/6)th are generated through the described crossover. By combining the newly generated puzzles with the selected puzzles from the previous generation, a new generation is formed.

5.4 Mutation

Because crossover alone is not very likely to come up with a correct solution, mutation is needed. Mutation on the puzzles is done in three ways. The first involves the rule that two

black cells may not be vertically or horizontally adjacent, a puzzle will be checked and of every combination of two adjacent black cells, one of them will be made empty. The second involves checking a puzzle for the absolute difference that is found. After doing so, every black cell in the puzzle will one-by-one be made empty and the puzzle will be checked again. If the absolute difference does not change, this means the cell has no involvement in the current puzzle. It will remain empty. If the absolute difference has changed, the cell will be turned back to being black. The final method involves going through a complete puzzle and randomly changing a small number of cells from black to empty or vice-versa.

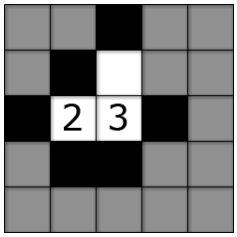
5.5 Termination

The process described above (initialization — selection) is run once. If no solution has been found during selection, the population will go through the rest of the process (crossover — mutation — selection). This process will be repeated until a solution has been found in the selection step of the process. If no solution is found after a number of generations (e.g., 500), the process is restarted from the initialization step. If there is still no solution after doing this multiple times (e.g., 20 times) the algorithm is stopped. As puzzles get more complex when they get bigger in size, the estimate of 1.5 times the amount of numbered cells in a puzzle will be black cells, may be inaccurate. As a result, the number estimated will be changed giving it an upper and lower bound of black cells. These bounds will be respectively raised and lowered after each time initialization is done.

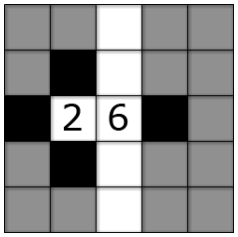
6 Special cases

6.1 The number 2

There are a number of special situations that can occur in Kuromasu puzzles. These situations call for special solutions or sometimes have default solutions that (nearly) always apply. A special case is the number 2 in a cell, especially when it's adjacent to another number. It is a special situation for different reasons. First of all, the number adjacent to the 2 will always be larger than 3. If the number adjacent to the 2 is a 3, black cells will be adjacent, which is not allowed through rule 3. The same applies for a cell with a 2 adjacent to another cell with a 2. The other reason why this is a special situation is because the placement of black cells will always be the same (not taking into account the sides of the puzzle). The 2 will be surrounded with black cells on 3 sides and a black cell will be adjacent to the other numbered cell opposite the side of the 2. An example of why the number adjacent to the 2 must always be larger than 3 and an example of the default situation can be seen in the figure below (Figure 17).



(a) A number adjacent to a 2 will always have to be larger than 3, as black cells will be adjacent if it is a 2 or 3.

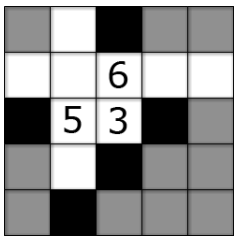


(b) The default solution for a 2 with another numbered cell next to it

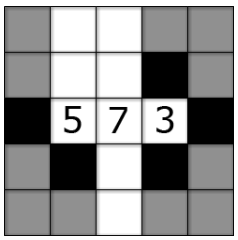
Figure 17: Special case — Number 2

6.2 The number 3

Among with the number 2, the number 3 can also create some special situations. Most common is the situation of a number 3 with 2 numbered cells adjacent to the number 3 or adjacent to each other. In both cases, a default solution is applicable. If the 3 is adjacent to both numbered cells, the two other sides will be black cells, also the cells adjacent to the numbered cells that are opposite the 3 will be black cells. If the 3 is adjacent to a numbered cell which is again adjacent to a numbered cell opposite the 3, the 3 will be surrounded by black cells on the remaining sides and a black cell will be adjacent to the last numbered cell opposite the 3. Examples of both situations can be seen in the figure below (Figure 18).



(a) Number 3 with 2 numbered cells adjacent to it



(b) Number 3 adjacent to a numbered cell, in turn adjacent to a numbered cell

Figure 18: Special case — Number 3

7 Results of solving the puzzles

7.1 Bruteforce

Bruteforce might be a very suitable option to solve the Kuromasu puzzles as it will always find a solution. A huge downside to bruteforce is the fact that the number of options to be tested is far too large to be done within reasonable time. Since there is no calculation to figure out the amount of black cells, bruteforce will always have to start at 1 black cell and work its way up. This will lead to enormous amounts of solutions to be tested. For example, an easy puzzle with 9×9 cells with 11 numbered cells and a solution with 17 black cells will have to check somewhere between $3,5 \cdot 10^{15}$ and $1,1 \cdot 10^{16}$ possible options, as can be seen in the table below.

Testing with the program showed that it takes the bruteforce program approximately 40 seconds to try 100 million options. Though this seems very fast, it would still take the program between 44 and 143 years to solve a simple 9×9 puzzle. With the calculation described in Section 3.2, we find for the puzzle described here, a minimum of 6 black cells. This means that we can start checking from 6 cells up. This seems like a step forward, but it only takes away 13,077,134 options, which is $3,7 \cdot 10^{-7}$ percent of the options, meaning that finding a solution will still take at least 43 years.

This is not good enough. Therefore, bruteforce is not useful to solve puzzles, although it can be used to finish puzzles that have been partially solved by the logical rules as the amount of options will then be a lot smaller.

Black Cells	Number of options	Total
1	70	70
2	2,415	2,485
3	54,740	57,225
4	916,895	974,120
5	12,103,014	13,077,134
6	131,115,985	144,193,119
7	1,198,774,720	1,342,967,839
8	9,440,350,920	10,783,318,759
9	65,033,528,560	75,816,847,319
10	396,704,524,216	472,521,371,535
11	2,163,842,859,360	2,636,364,230,895
12	10,638,894,058,520	13,275,258,289,415
13	47,465,835,030,320	60,741,093,319,735
14	193,253,756,909,160	253,994,850,228,895
15	721,480,692,460,864	975,475,542,689,759
16	2,480,089,880,334,221	3,455,565,423,023,980
17	7,877,932,561,061,644	11,333,497,984,085,620
18	23,196,134,763,125,940	34,529,632,747,211,560

7.2 Logical rules

Logical rules turn out to be very effective in solving Kuromasu puzzles. In order to test the program, a test set of 50 easy and medium puzzles was used, taken from the only available Kuromasu puzzle book [5]. The results were as following:

Depth of guesses	0	1	2	3	4	5
Puzzles solved	25	14	0	6	4	1

Depth of guesses is the number of times a "guess" is made, as described in Section 4.11, where 0 in this table means that no guesses were made. Total calculation time for all 50 puzzles was approximately 6 minutes. Calculation per puzzle varied from 0.1 seconds to approximately 4 minutes. In a smaller test set with hard puzzles, taken from the same book as described above, calculation time per puzzle went up to about 15 minutes, but the program was still able to solve all puzzles.

7.3 Genetic Algorithms

The Genetic Algorithm is fairly effective on smaller puzzles, mostly graded as being easy. For the larger puzzles the success rate is a lot lower. The test set that was used for testing the logical rules was also used to test the genetic algorithm. Every puzzle was tested 3 times. The parameters used:

Parameter	Value
Population size	1,500
Selection size	250
Selection type	standard select
Penalty for adjacent black cells	2
Amount of cells changed in mutation	3
Chance for removing adjacent black cells	.5
Chance for removing "ignored" black cells	.5
Restart after generation	500
Abort after generation	10,000
Amount of black cells	expanding at restart

The algorithm was able to solve 27 of the 50 puzzles at least once during the three attempts. In total, it solved the puzzle in 63 of the 150 attempts. A high success rate was reached on the first group of puzzles, the smaller and easier puzzles, on the larger, harder puzzles, the success rate was extremely low, as can be seen in the list of results below. For a reference to the logical rules, besides the result, also the number of "guesses" is shown in the table.

No.	Result	Guesses	No.	Result	Guesses	No.	Result	Guesses
1	3	0	18	2	0	35	0	3
2	1	0	19	3	0	36	0	0
3	3	0	20	3	3	37	0	1
4	3	0	21	3	0	38	1	4
5	3	0	22	0	0	39	0	1
6	3	0	23	1	1	40	0	1
7	3	0	24	0	1	41	0	4
8	3	0	25	0	3	42	0	1
9	3	0	26	0	1	43	0	1
10	2	0	27	0	0	44	0	0
11	3	0	28	1	0	45	0	3
12	2	0	29	0	0	46	0	1
13	2	0	30	0	1	47	0	3
14	3	0	31	0	1	48	0	1
15	3	1	32	0	1	49	2	4
16	3	0	33	0	3	50	1	5
17	2	0	34	1	4			

8 Conclusion

After testing the three different methods, it must be concluded that the logical rules method is the best way to solve a Kuromasu puzzle, with a success rate of 100% on the test set. Genetic Algorithms are a good second, with a success rate of over 50%. Though the bruteforce method will, in theory, also have a success rate of 100%, it is not fast enough to ever successfully solve a puzzle. Time is also a point where the logical rules method outperforms the Genetic Algorithm.

8.1 Future work

A combination of the methods used in this research could be examined. For example, the logical rules could be applied to the point where no more cells are changed (and the "guessing" would normally begin), and the Genetic Algorithm could be applied. As could be seen in Section 7.3, the Genetic Algorithm performs best on small puzzles, so it is likely to get good results on puzzles that are largely solved by logical rules and only need a small part to be solved.

References

[1] [Wikipedia 2008 — Kuromasu] Wikipedia (2008). Page on Kuromasu. `http://en.wikipedia.org/wiki/Kuromasu` [accessed: 2008.11.26]

[2] [Wikipedia 2008 — Genetic Algorithm] Wikipedia (2008). Page on Genetic Algorithms. `http://en.wikipedia.org/wiki/Genetic_Algorithm` [accessed: 2008.11.26]

[3] [Eiben and Smith — 2003] Eiben, A.E. and Smith, J.E. (2003) *Introduction to Evolutionary Computing (Natural Computing Series),* Springer.

[4] [Nikoli 2008] Website of Japanese company Nikoli, creator of Kuromasu puzzles. `http://www.nikoli.co.jp/en/` [accessed: 2008.11.26]

[5] [Nikoli 2007] Nikoli Company, *Where is black cells*, Nikoli company

[6] [Russell and Norvig, 2003] Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 2nd edition.