A comparison of a genetic algorithm and a depth first search algorithm applied to Japanese nonograms

Wouter Wiggers
Faculty of EECMS, University of Twente
w.a.wiggers@student.utwente.nl

ABSTRACT

In this paper the performance of a genetic algorithm and a depth first algorithm are compared by implementing both algorithms and solving the Japanese Nonogram. Both algorithms are popular because of their generic structure, which makes them easy to apply to a wide range of problems. Problems such as how to make a correct representation of the Japanese Nonogram so it can be used by the genetic algorithm will be addressed. In the last section the results of the experiments show that the genetic algorithm can outperform the depth first search algorithm.

1. INTRODUCTION

A genetic algorithm [RN95] is an algorithm that can be easily applied to solve difficult problems because of its generic structure. Often one does not have a full understanding of a problem or it is just too difficult to implement an algorithm. This difficulty makes the use of genetic algorithms so attractive. In this article I take a closer look at the performance of this algorithm.

During my research I am going to compare the performance of two different algorithms applied to the Japanese nonogram problem. The two algorithms I want to compare are a depth first search algorithm and a genetic algorithm. I have chosen for the depth first search algorithm because it is a typical example of a brute force algorithm that is straightforward to implement. I want to compare it with the genetic algorithm because both algorithms can be used for almost any problem and it is exciting to know which one is faster in the general case. Because researching the general case is almost impossible I have narrowed my research to a particular problem of interest, the Japanese nonogram.

I will begin this article by first giving an exact definition of the Japanese nonogram and how it can be solved. After this section I will clarify the general working of genetic algorithms. This explanation is an introduction to genetic algorithms and people who already know about them can skip this part. Next are the sections about the implementation of the different algorithms. First the depth first search algorithm is implemented and hereafter the genetic algorithm. Finally, this article will give the results of the performance measurements of both algorithms.

1st Twente Student Conference on IT, Enschede 14 June 2004

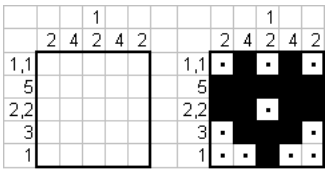

Figure 1: The Japanese nonogram

2. JAPANESE NONOGRAM

The Japanese nonogram is a type of problem that can best be explained using a picture. In figure 1 a simple nonogram is given. The purpose of the puzzle is to color the correct cells. The numbers alongside the rows and columns give information about how many cells need to be filled in that row or column respectively. Take for example the first row of the puzzle in figure 1. Here 1,1 means that in this row two cells need to be painted with one or more white spaces between them.Unfortunately there are six different positions of that row, these permutations are also shown in figure 4 , so it is not clear which cells need to be painted. The second row of the puzzle has the number 5 attached to it, so this row can be entirely painted with black cells. As gradually more cells are being filled more information becomes available which cells must be filled and which must be left empty. When all the rows and columns are painted according to their numbers the puzzle is solved. More information about Japanese nonograms can be found in [EL04].

The Japanese nonograms that one finds in popular puzzle magazines are always solvable using simple logic. This means that the solution of a puzzle can be found with elementary reasoning and without complex deductions. Although it is comprehensible such puzzles are popular my research will cover a larger superset of Japanese nonograms. These puzzles are sometimes not solvable using simple logic and require making assumptions that may prove wrong later on.

The fact that this puzzle is difficult to solve for humans is not only an empirical result but has been proven as well. An

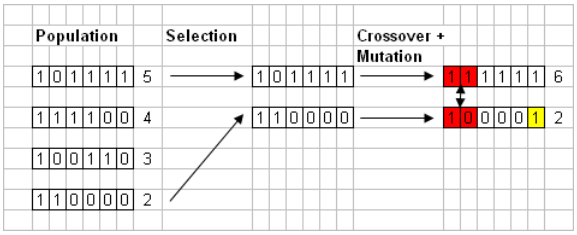

Figure 2: The genetic algorithm

article of N. Ueda and T. Nagao[UN96] shows that the decision problem that must be solved in a japanese nonogram is NP-complete and thus not solvable in polynomial time. This gives an impression of how hard the problem is that I am trying to solve.

The Japanese nonogram is a NP-complete problem and the consequence is that this research will not only show how genetic algorithms perform on Japanese nonograms but also how genetic algorithms perform on other NP-complete problems. This effect is caused by the fact that all NP-complete problems are reducible to each other [SB00] [JS89].

3. GENETIC ALGORITHMS

In this section it is explained what genetic algorithms are. First I will show the relation between biological evolution and genetic algorithms. Next I will use an example of a genetic algorithm to explain the different operators that are used and finally I will give the general outline of a genetic algorithm using a flowchart.

A genetic algorithm uses many biological-derived techniques such as inheritance, natural selection, recombination and mutation. These functions are all necessary parts of the evolution process in nature. Because this evolution process is so important in nature people have tried to understand it and are trying to simulate it. A genetic algorithm is just an abstraction of the evolution process of nature and also uses the same functions but with a slightly different name.

The genetic algorithm that I will explain and use to solve the Japanese nonogram works with the three operators: selection, crossover and mutation [Gol94] The functioning of these three operators is shown in figure 2. These operators work on the data sets of the algorithm, which are also called chromosomes. Another important part of the algorithm is the fitness function. This function calculates the fitness of a chromosome using the genes of the chromosome. The goal of a genetic algorithm is to optimize this fitness function and find the individual that has the "best" fitness value.

Figure 2 shows an example of a genetic algorithm. The process starts with a population of 4 chromosomes or individuals. The genes of the chromosomes are binary encoded en thus can contain the values 0 and 1. The values behind the chromosomes correspond to their fitness. In this example the fitness-function just counts the number of ones in the chromosome.

After calculating the fitness of all the chromosomes in the population pool of 2 two chromosomes must be selected for

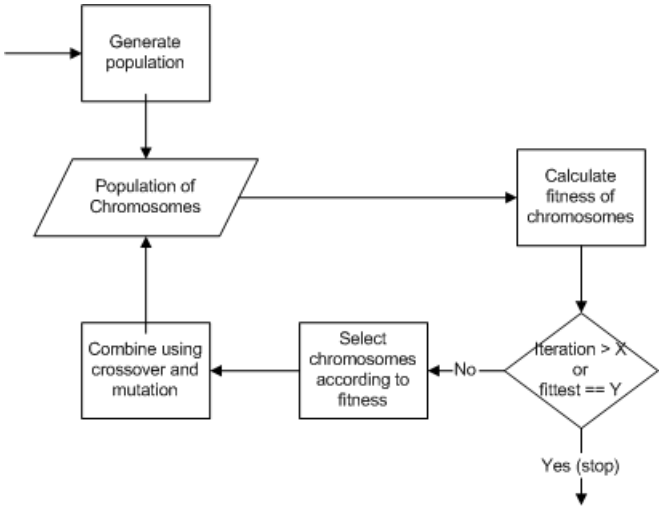

Figure 3: Flowchart of the genetic algorithm

reproduction. This selection process is often random but with a bias for the chromosomes with higher fitness. Following the selection process the crossover and mutation operators will work on the two chromosomes. The crossover operation exchanges a number of genes from one chromosome with another. The mutation operator just changes the value of one gene in a chromosome. In the right part of figure 2 the left two genes of the chromosomes are exchanged and the mutation occurs on the right side of the lower chromosome. Crossover and mutation do not always happen however, in [JS89] a cross-over rate of 60% and a mutation-rate of 0.1% are used. This means that 60% of the selected chromosomes recombine with each other and only 0.1% of the selected chromosomes mutate.

The general outline of the genetic algorithm is given in figure 3. First the algorithm starts with a population or generates one. Next the fitness of the chromosomes is calculated. Hereafter a check is made whether the fitness is high enough or there have been too many iterations and no solution was found. After this check the selection, cross-over and mutation operators are used to generate the new population. When the generation of the new population is finished the process in figure 3 starts all over again. The goal of this procedure is to find a chromosome with a fitness that is "good enough". More information about genetic algorithms can be found in [Gol89].

Details about the fitness-function, the type of selection, cross-over and mutation rate that were used in the genetic algorithm for solving the Japanese nonogram will be explained in a later chapter.

4. DESIGN OF THE DFS

Solving the Japanese nonogram using a depth first search algorithm is very straight forward. One takes the first row of the puzzle and generates all the different positions for that row. In figure 4 this is done for the first row of that puzzle. When all the possible positions for the first row have been generated the second row generates a new position and the process starts all over again until all possible solutions of

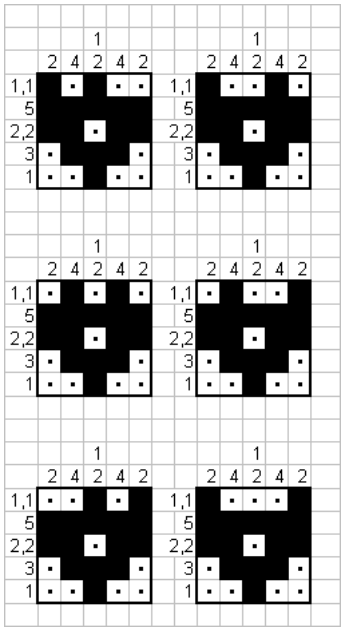

Figure 4: Result of a DFS

the Japanese nonogram have been generated. It is called a depth first search because all the possible positions for the first row are generated first.

The different rows in the puzzle together create a solution. Each different solution is then checked for correctness using the columns of the puzzle as a reference. When every column of the Japanese puzzle is correct the puzzle is solved. This is because each generated position of a row is already correct. In figure 5 this process is shown for the different positions of the first row. The third solution in figure 5 is the correct one. The solution of the puzzle in figure 5 is found quickly because the positions of the other rows were already correct.

The depth first search algorithm searches for the correct solution of the puzzle and returns the correct one if it exists and the number of checks that were necessary to find it. A check is defined as the process of evaluating a proposed solution of the puzzle. Because the steps the depth first search algorithm takes are irreversible and the state space is finite (there are only a finite number of permutations of the rows) a solution will be found if it exists.

Although the method of generating the state space during the depth first search and the evaluation function are both very fast the overall performance of this algorithm is very poor. This is because the are so many possible states that need to be checked. The actual performance of this algorithm is presented later in this article. The depth first search algorithm is implemented in the Japanese Nonogram Solver Prototype which is available at [Wig04].

5. DESIGN OF THE GA

In this section a genetic algorithm is developed that is suitable for solving Japanese nonograms. In figure 3 the outline of the algorithm is given and in this section I will decide upon the exact implementations for the fitness function and

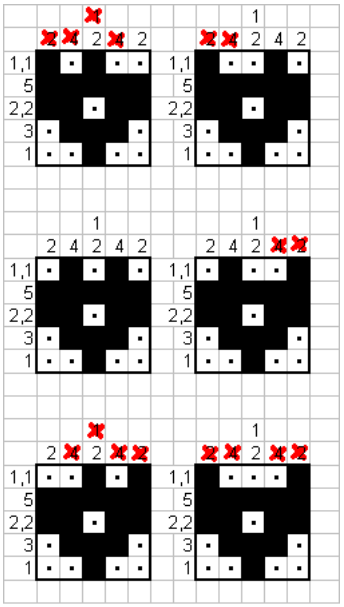

Figure 5: Verification

SAT-problem: (X AND Y) OR (Z AND X)

X	Y	Z	fitness:
0	0	0	0
0	0	1	0.5
0	1	0	0.5
0	1	1	0.5
1	0	0	0.5
1	0	1	1
1	1	0	1
1	1	1	1

Figure 6: Calculating the fitness

the crossover-, mutation- and selection-operators.

The most important part of the genetic algorithm is the fitness function which will ultimately decide which chromosome is the best. In an article of De Jong and Spears [JS89] a technique is given for making a fitness function for a Satisfiability problem. I used this technique to generate a fitness function for every possible Japanese nonogram by first transforming the Japanese nonogram into a SAT-problem.

The method for calculating the fitness of a solution of the SAT-problem given in [JS89] is actually very simple. Instead of giving a solution a fitness of 1 or 0 the fitness is averaged. A fitness that can take more values then only 1 or 0 is necessary because it gives more information about the fitness of the solution, this information is required for making a correct selection of chromosomes later in the algorithm. This averaging of the fitness is accomplished by redefining the AND-operator by an AVG-operator which takes the average of the operands. The OR-operator is redefined as a MAX-operator which returns the maximum of the operands. An example is shown in figure 6

This leads us to the problem of transforming the Japanese nonogram to the Satisfiability problem in order to get a

AND
X_{00} AND $\neg X_{01}$ AND X_{02} AND $\neg X_{03}$ AND $\neg X_{04}$
OR
X_{00} AND $\neg X_{01}$ AND $\neg X_{02}$ AND X_{03} AND $\neg X_{04}$
OR
$\neg X_{00}$ AND X_{01} AND $\neg X_{02}$ AND X_{03} AND $\neg X_{04}$
OR
$\neg X_{00}$ AND X_{01} AND $\neg X_{02}$ AND $\neg X_{03}$ AND X_{04}
OR
$\neg X_{00}$ AND $\neg X_{01}$ AND X_{02} AND $\neg X_{03}$ AND X_{04}
OR
X_{00} AND $\neg X_{01}$ AND $\neg X_{02}$ AND $\neg X_{03}$ AND X_{04}
AND
X_{10} AND X_{11} AND X_{12} AND X_{13} AND X_{14}
AND
X_{20} AND X_{21} AND $\neg X_{22}$ AND X_{23} AND X_{24}
AND
X_{30} AND X_{31} AND X_{32} AND $\neg X_{33}$ AND $\neg X_{34}$
OR
$\neg X_{30}$ AND X_{31} AND X_{32} AND X_{33} AND $\neg X_{34}$
OR
$\neg X_{30}$ AND $\neg X_{31}$ AND X_{32} AND X_{33} AND X_{34}
AND
X_{40} AND $\neg X_{41}$ AND $\neg X_{42}$ AND $\neg X_{43}$ AND $\neg X_{44}$
OR
$\neg X_{40}$ AND X_{41} AND $\neg X_{42}$ AND $\neg X_{43}$ AND $\neg X_{44}$
OR
$\neg X_{40}$ AND $\neg X_{41}$ AND X_{42} AND $\neg X_{43}$ AND $\neg X_{44}$
OR
$\neg X_{40}$ AND $\neg X_{41}$ AND $\neg X_{42}$ AND X_{43} AND $\neg X_{44}$
OR
$\neg X_{40}$ AND $\neg X_{41}$ AND $\neg X_{42}$ AND $\neg X_{43}$ AND X_{44}
AND
X_{00} AND X_{10} AND $\neg X_{20}$ AND $\neg X_{30}$ AND $\neg X_{40}$
...

Figure 7: SAT-problem of a nonogram

mutation: 5%
cross-over: 60%

size nonogram:	dfs:	avg. ga:	1st run:	2nd run:	3rd run:
05x05	113	5800	3900	7400	6100
06x06	22140	65000	98200	36800	60100
07x07	961818	308100	141000	520900	262400
08x07	1555379	655066	37700	626000	1301500
10x10	46417433	1148533	1474700	1359600	611300

Figure 8: Number of evaluations

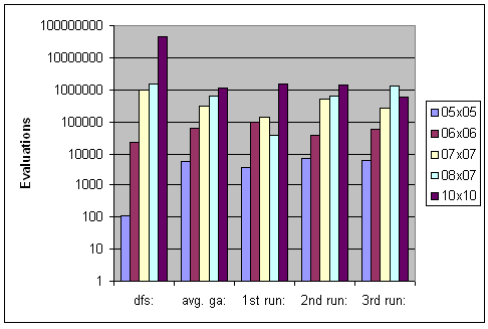

Figure 9: Graphical comparison

suitable fitness function. Of the puzzle shown in figure 1 the corresponding Satisfiability problem is given in figure 7. Every X of the Satisfiability problem belongs to a matrix position of the puzzle in the figure 1, in this example X_{00} is the left-uppermost square and X_{44} is the right-bottommost square of the nonogram. If one can find a solution for the SAT-problem given in figure 7 the solution for the Japanese nonogram of figure 1 can be made by filling the correct squares of the puzzle. If for example X_{00} must be true in order to get a SAT-solution, the corresponding square of the Japanese nonogram must be filled too. The SAT-problem in figure 7 is made by calculating the different permutations of the rows and columns of the Japanese nonogram. The problem is then formed by putting all the different permutations together with a $\wedge$ or a $\vee$.

With the fitness function defined it follows that a chromosome of the genetic algorithm must consist of ones and zeros. These genes of a chromosome correspond to the squares of the Japanese nonogram and tell whether they are filled or not.

Now we only need to design the mutation, crossover and selection operator of the genetic algorithm. Because the fitness function is based on the SAT problem in [JS89] I have also tried to get the other operators of the algorithm the same as in [JS89]. The mutation operator is straight forward and just flips a bit of the chromosome at a random position. The cross-over operator is the same as in [JS89] and uses 2-point cross-over instead of 1-point where the chromosome is cut in two parts. The selection operator uses roulette wheel selection where the chromosomes' segments of the wheel are as large as their corresponding fitnesses, this provides a random selection with a bias for the fittest individuals. The genetic algorithm is implemented in the Japanese Nonogram Solver Prototype which is available at [Wig04].

6. RESULTS OF BOTH ALGORITHMS

In this section the performance results of the two algorithms will be given. The performance of an algorithm is defined as the number of evaluations it needs to solve the Japanese nonogram. An evaluation of the depth first search algorithm takes place when it checks whether the different rows of a possible solution together form a correct solution. The genetic algorithm makes an evaluation each time the fitness function calculates the fitness of a chromosome. Because of the stochastic nature of the genetic algorithm the performance will be averaged over the first 3 succesful runs. Although this provides a bias in the measurements I was not able to design a different way of measuring performance. This was caused by the fact that the genetic algorithm often got stuck during solving. More about this problem will be clarified later in this article.

I have researched different settings for the cross-over and mutation-rate of the genetic algorithm and I got the best results with a cross-over rate of 60% and a mutation-rate of 5%. The setting of the mutation-rate is actually very high for a genetic algorithm, but this was necessary to avoid staying stuck in local optima of the nonogram. The population size did not make any obvious differences for the performance if it was set at 100 chromosomes or a larger size. But if the population size was set at for example 10 chromosomes the algorithm began to perform very poorly. Because of these results I have choosen to use the population size of 100 chromosomes in my genetic algorithm, this size is also used in other researches such as [JS89].

The two different algorithms were compared using five different sized nonograms. These nonograms were ofcourse chosen without a particular bias for an algorithm. It can be proven that the complexity of a nonogram depends on the total number of permutations of the rows and columns.

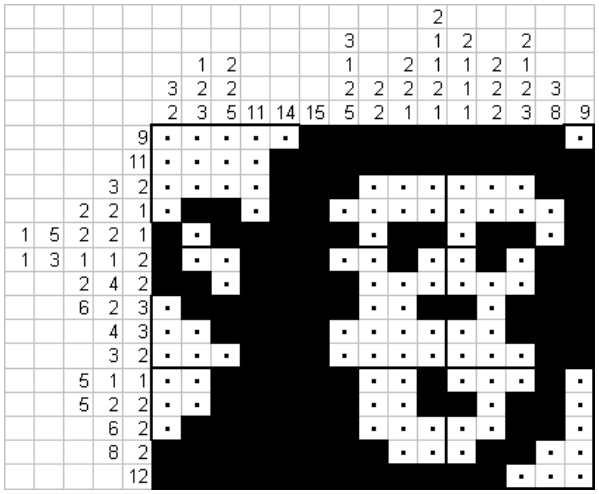

Figure 10: 15x15 nonogram

Furthermore, the number of permutations of a nonogram is dependend on the size of the puzzle, that's why I have choosen five in size succeeding nonograms.

The results of both algorithms are given in figure 8 and 9. I had some problems with measuring performance for bigger nonograms using the genetic algorithm, often the genetic algorithm could not find a solution and I had to restart the algorithm manually. This is caused by the effect that the genetic algorithm converges to local optima instead of the correct solution. However, when the genetic algorithm finds the solution for big nonograms the number of evaluations it requires is less than the number the depth first search requires. On the other hand, the depth first search clearly outperforms the genetic algorithm for little nonograms. This is caused by the effect that a genetic algorithm starts with a population of 100 chromosomes and the algorithm first processes all the individuals of the population before checking whether a solution is found.

As an example of a large nonogram I have tried to solve a 15x15 puzzle using both algorithms. In figure 10 the solution of the 15x15 nonogram is shown. This nonogram was solved using the genetic algorithm and needed more than 2 million evaluations to complete and because it took 20 minutes processing time on a normal personal computer I have only tried to solve the puzzle once using the genetic algorithm. I have not solved this nonogram using the depth first search algorithm because it took at least an hour to complete. This example of a large nonogram is an empircal result which also favours the genetic algorithm in solving the japanese nonogram.

It is also remarkable to see that the genetic algorithm converges very fast to a solution that is almost good but not the correct one. Unfortunately the last steps of the genetic algorithm take a long time. This is probably because the algorithm can no longer rely on the cross-over operator to find the correct solution but needs the mutation operator to find a new species.

7. CONCLUSIONS

This paper shows how a genetic algorithm performs on the japanese nonogram compared to a depth first search algorithm that solves the same problem. To create a suitable genetic algorithm to solve the problem I used a transformation of the Japanese nonogram to the Satisfiability problem.

The actual results of the performance tests are favoring the genetic algorithm. If one looks at large nonograms, the 10x10 sized nonogram for example, the depth first search algorithm uses more than 10 times more evaluations compared to the genetic algorithm. However, the depth first search algorithm outperforms the genetic algorithm when solving small nonograms.

Sometimes the genetic algorithm got stuck because of local optima in the Japanese nonogram, this was solved by manually resetting the algorithm. It is possible to avoid this problem by somehow detecting that the population of the genetic algorithm no longer evolves.

This article shows that a genetic algorithm can perform very well compared to a more robust and predictable algorithm. It is also remarkable to see how fast genetic algorithms can find a solution that is almost correct.

8. DISCUSSION

The performance of the genetic algorithm was measured by averaging the number of evaluations of the first three successful runs. This however presents a possible bias for the genetic algorithm. I could for example initialize the puzzle at random and check whether the solution is correct, this causes the algorithm to only need 1 evaluation for a nonogram. I was not able to design a different way of measuring the performance because the genetic algorithm often got stuck in local optima, as was said before in this paper. Future research is needed to solve this problem.

The population size that was used in the genetic algorithm is too large for small nonograms. A solution would be to change the population size dynamically according to the size of the nonogram. But I have chosen not to implement this improvement but to let the genetic algorithm be as generic as possible. I have made the same consideration for the depth first search algorithm. It is possible to improve some points in this algorithm too, one could for example use the information of the columns of the nonogram to reduce the search space of the algorithm. I think the effects of these improvements are negligible however because the Japanese nonogram problem is NP-complete, but this needs to be investigated by future researches.

9. REFERENCES

[EL04] Hans Eendebak and Jan Lam. *Japanse Puzzels*. Mat Heffels, 2004.

[Gol89] David E. GoldBerg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[Gol94] David E. Goldberg. Genetic and evolutionary algorithms come of age. *Communications of the ACM*, 37:113–119, 1994.

[JS89] Kenneth A. De Jong and William M. Spears. Using genetic algorithm to solve np-complete problems. In *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 24–132, 1989.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. Alan Apt, 1995.

[SB00] Allen Van Gelder Sara Baase. *Computer Algorithms*. Addison Wesley Longman, 2000.

[UN96] N. Ueda and T. Nagao. Np-completeness results for nonogram via parsimonious reductions. Technical Report TR96-0008, Tokyo Institute of Technology, 1996.

[Wig04] Wouter Wiggers. Japanese nonogram prototype. *http://home.student.utwente.nl/w.a.wiggers/prot.zip*, 2004.