Solver of 8-Puzzle with Genetic Algorithm

Cheng Min Yang
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP064404@mail.apu.edu.my

Vincent Pek
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP063671@mail.apu.edu.my

Sim Hui Ling
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP061282@mail.apu.edu.my

Tan Choon wei
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
TP063860@mail.apu.edu.my

Zailan Arabee Abdul Salam
School of computing
Asia Pacific University of Technology and Innovation (APU)
Kuala Lumpur, Malaysia
zailan@apu.edu.my

***Abstract*—Genetic algorithm is a search heuristic that takes Charles Darwin's theory of natural evolution as its source of inspiration that can be used in solving 8 puzzle problems. One of the earliest known problems in mathematics is the "N puzzle problem.". In this paper, we will look at how to solve 8 puzzle problems by using genetic algorithm. The parameters will be changed to see the different result and time used. The three parameters that will be modified are the crossover rate, mutation chance and population len. The results showed that the problem will be solved faster when the crossover rate and mutation chance are higher but when the population size becomes more, the time taken to solve the problem will be longer.**

Keywords—8 puzzle problem, genetic algorithm, N puzzle problem

I. Introduction

This documentation is about using genetic algorithms to solve 8-puzzle. 8-puzzle is a game invented by Noyes Palmer Chapman in 1870s. (Shahzad, 2022) Nowadays, it is a popular game which almost everyone has played before, the rules is simple you can only move the block beside the block number 0 (change the position with it) and your goals is rearrange the blocks from number 0 to number 8. The boards will randomly generate when the game starts. Hence, you won't face the same board in a short time unless you are very lucky. Genetic algorithms also came from a field of study that is known as evolutionary computation in which the algorithm was used to copy the reproduction process and select the fittest solutions (Hor et al., 2022; Yuen et al., 2021).

There are many different algorithms that can be used to solve problems, for example the A* search algorithm, which will always find the best solution to solve the specific problem. But we decided to use genetic algorithm as it is an algorithm which we didn't learn before. This documentation will include a few different parts to explain what genetic algorithms are and how the problem is solved by the algorithm.

II. Literature Review

8 puzzle or N-puzzle is a common problem in artificial intelligence. There are many research using different approaches to solve this problem. Most of the research are using breath-first-search, death-first-search, or A* algorithm to solve 8 puzzle problem. In the research of (Shaban et.al, 2010) they were using genetic algorithm to solve the problem.

In the research, they had applied all the stages of genetic algorithm. 8 puzzle is a 3x3 grid which contains nine blocks (Richard, n.d.). In the research, they convert the puzzle into one dimension array and this array will be the chromosome. The program will first generate a new population of tiles based on the number of available moves of the block which do not match the goal state (Melanie, 1999). The available move is counted by it position, for example, the block at the top-left of 8 puzzle will only have two available moves which are moving down or to the right. After that, the fitness function values of each chromosome will be calculated using the fitness function provided in the research paper. The final goal will also have a fitness value which is used to compare with the chromosome. If the comparison of fitness values of chromosome and final goal is satisfied, the program will then record the chromosome. Then, tournament selection method was being used to choose two individuals as parent from the filtered parents (Melanie, 1999).

The fourth phase will be cross over, in this stage, two chromosomes (previously selected parent) will be recombined (Goldberg, 1989). The cross point will be selected randomly, then parent 1 and parent 2 will recombine to form a new child. As a reason of the chromosome is 1D array, it will cause that the child may have duplicate values after cross over operator, thus, after cross over operator, the program will remove the duplicate number and add the missing number at the cross over point of chromosomes. The main objective of this stage is to generate new population. Last, all the new chromosomes will have a probability to carry out mutation. The mutation operator will swap the position of two random number and preventing the chromosome become too similar to each other to avoid local minima (Koza, 1992). Iteration will be carried out for repeating these procedures until the program cannot give out a better solution.

In their research, they had provided two results after running the program. In the first run of the program, we can see the graph showing the greater the number of generations, the greater the number of solutions that can be found. However, the second run of the program show that the number of solutions become lower when the number of generation higher than 300. From the graph we can also find that the program cannot give out a satisfy result when the number of generations is below 100 while the number of solution start growing when the number of generations is greater than 100. From the two examples, we can know that genetic algorithm is able to solve 8 puzzle problem and provide different solutions. However, the examples and results provided is not enough since that the first example are having only 2 different blocks compare to the goal while the second example having 3 differences. The example should have more difference block compared to the final goal so that we can have more accurate result.

In conclusion, genetic algorithm is able to solve 8 puzzle problem and from the research of Sha'ban et al, we can find that the optimal number of generations to find out the solutions would be around 100 until 300.

III. Material and Method

A genetic algorithm is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are a particular class of evolutionary algorithms (also known as evolutionary computation) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover also called recombination. From the idea inspired by evolutionary biology, genetic algorithms will maintain the chromosome with good genes and weed out the chromosome with bad genes. As the diagram shows in Fig. 1, it will select two chromosomes as parents and crossover two of the chromosomes to generate a new chromosome. Then, the new chromosome will have the probability to mutate either in a bad way or in a good way. After this, it will maintain the chromosome with good genes and weed out the chromosome with bad.

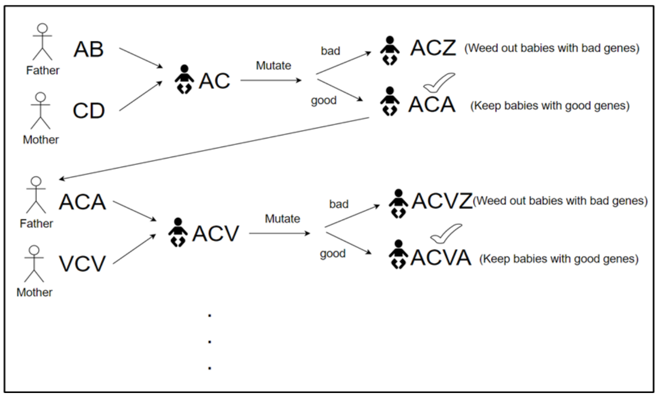
AB
Father
CD
Mother
AC
Mutate
bad
good
ACZ (Weed out babies with bad genes)
ACA (Keep babies with good genes)
ACA
Father
VCV
Mother
ACV
Mutate
bad
good
ACVZ (Weed out babies with bad genes)
ACVA (Keep babies with good genes)

Fig 1. Example of evolutionary biology

A genetic algorithm operates through a simple cycle of stages (Konar, 2018).

- Creation of a "population" of strings.
- Evaluation of each string.
- Selection of best strings.
- Genetic manipulation to create a new population of string.

Fig 2. illustrated the cycle of a genetic algorithm (Konar, 2018).

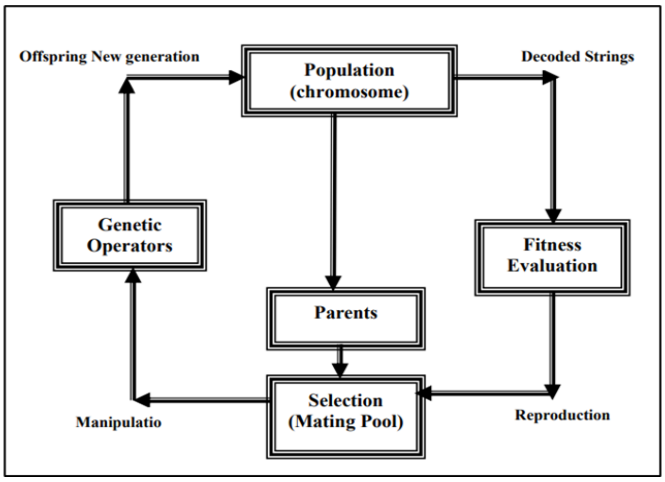

Fig.2 Cycle of a genetic algorithm (Konar, 2018).

IV. Flowchart for Proposed Genetic Algorithm to Solve the 8-Puzzle Problem

A. *Initial state and Goal State*

According to the flow chart below, the initial state will be generated randomly while the goal state will be fixed from 0 to 8. Chromosomes will be generated randomly from the initial state to form a population. The number of chromosomes that will be generated will be according to the value input in the population length. For example, if the input for population length is 1000, then there will be 1000 chromosomes generated from the initial state to form the population. After this, it will calculate the fitness value (fitness function) for each chromosome in the generation for the proposed heuristic genetic algorithm and compare the fitness function for each chromosome with the fitness function of the goal and compare the tile's value of chromosome with tile's value of the goal. If the two conditions are satisfied, then record the generation's index and chromosome index. Else go to the next chromosome.

B. *Selection, Crossover and Mutation operations*

The next generation is produced by executing Selection, Crossover, and Mutation operations, respectively. The crossover operation was to crossover the chromosomes according to the input of crossover rate. For the mutation operation, used the order changing method as two numbers are selected and exchanged them. The probability for chromosomes to mutate is according to the input of mutation chance. After this, it will calculate fitness value (fitness function) again for each chromosome in the new generation and compare the fitness function for each chromosome with fitness function of the goal, also compare the tile's value of chromosome with tile's value of the goal. It will terminate if the optimal solution is found. Else, it will go back to the step of selection and repeat the following steps until the optimal solution is found.

Fig 3. shows the Flowchart for proposed genetic algorithm to solve the 8-puzzle problem

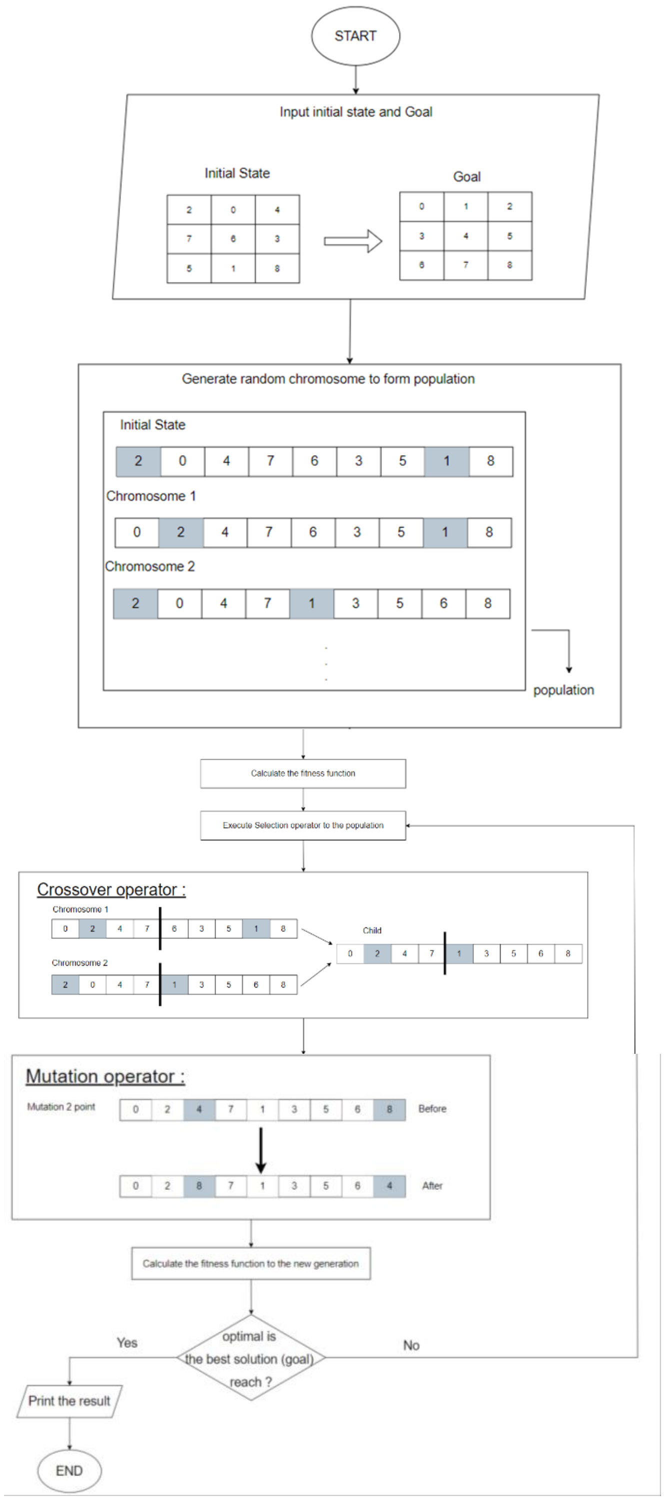
START
Input initial state and Goal
Initial State
Goal
Generate random chromosome to form population
Initial State
Chromosome 1
Chromosome 2
population
Calculate the fitness function
Execute Selection operator to the population
Crossover operator :
Child
Mutation operator :
Mutation 2 point
Before
After
Calculate the fitness function to the new generation
optimal is the best solution (goal) reach ?
Yes
No
Print the result
END

Fig.3 Flowchart for proposed genetic algorithm with 8-puzzle problem

V. Algorithm Implementation

Genetic algorithm is an algorithm which was developed by John Holland and his student in 1975. And they were inspired by the theory of evolution which was introduced by Charles Darwin. This algorithm is based on a combination of genetic crossover, mutations, and nature selection, which can keep generate better answer until the generation reaches max generation.

From the source code of (Noury, 2020) some tools such as numpy and random are called to generate the board which could randomly generate a new board when a new game is started. For the board, the function numpy.zero will be used to generate an empty matrix which will fill by 0. The matrix will be filled by combination of two for loop. Firstly, the index number will be set as 0 because the index number will be used to fill the matrix which is generated before. The for loop will fill the block [I, j] with the index number every time the second for loop is run after the second for loop are run the index number will automatic plus one. The for loops will run until all empty blocks in the matrix are filled by the index number.

After the matrix is filled, the matrix will be arranged from 0 to 8. Hence, the board needs to be shuffled before the game starts, the function shuffle will be called to do this action. The function will be used for loop to randomly move the numbers in the matrix 100 times to make sure the matrix is disordered. It will use four functions which will also be used when the algorithm tries to arrange the number in the matrix which are move_up, move_down, move_left and move_right.

These four functions are almost the same, but the direction of the move isn't the same. Fistly, the code will used where function which is called from numpy to find out the 0 in the block. After that, it will check if the number 0 can move to that specific direction for example, in the move_left function the code will check if the number 0 is at the third (2 in the code cause start with 0) position because if the number 0 is at that position it is not able to move to the left-hand side than the code will return false and try to move it to other direction. Same for the others move function. Until here, the game board is generated, and the algorithm can start to solve the puzzle.

The class chromosomes are also very important as they will affect the output of the algorithm. The update error function in this class will use deepcopy from module copy, this is because you didn't want the affect the regular board. Every update in the deepcopy will not cause the original object to be changed because if you are using deepcopy the code will create a new object. The code will also use apply_chain from the class pad (generate board and allows movement). The apply_chain function will be used to print out the movement or the board.

After selecting the parents, the function cross_over will run. In this function the function random will be called and if the number provided by the function random is smaller than 0.5 the evolution of the gene will happen. The mutate chance is also using random. random function, if the mutate chance is lower than the number generated by the function than the mutate will not happen else the gene will be mutated.

The class solver will import two modules which are math and random. This class will use the output of the class pad and the chromosomes. The function select best will run to choose the best gene in the generation and let them become parents of the next generation. The function of the calculate_error will show us the total error of the iteration.

The mutate function will be run after the parents are chosen. If the number generated by function random. random are smaller than mutate chance the mutate function in the class chromosomes will execute. The last function is solved, it will run 1000 times repletely, unless the best solution is generated. Then the display function will run, to print out the iteration.

In this code there are 4 parameters which can be changed which are Mutate_chance, Cross_over_rate, Population_len and Max_iteration. The function of each parameter will be discussed later on.

VI. Result and discussion

A. *Mutation Chance*

The tests conducted here involve the mutation chance that will have lesser iteration. The max iteration for the runs here is 1000, the population length is 100 and the crossover rate is 0.5.

In Table I, the average of iterations with the mutation chance of 0.9 is lesser compared with the mutation chance of 0.1 and the mutation chance of 0.5. This is because the chance of mutation refers to the probability that a chromosome will mutate. Therefore, the mutation chance of 0.9 means that each chromosome will have a 90 percent probability to mutate into a new chromosome and it will generate a greater number of new chromosomes in one iteration. Hence, with a greater number of new chromosomes that will be generated in one iteration, the greater number of mutation chances can have a lesser number of iterations.

TABLE I. Mutation chance result

No	Number of Iteration		
	Mutation chance = 0.1	Mutation chance = 0.5	Mutation chance = 0.9
1	544	1	2
2	680	7	54
3	369	97	21
4	235	564	23
5	179	244	69
6	4	268	77
7	9	6	39
8	149	127	145
9	567	329	143
10	9	342	41
11	431	5	63
12	999	913	8
13	196	178	1
14	131	237	30
15	11	154	7
16	761	2	48
17	173	4	105
18	407	74	38
19	899	272	111
20	8	549	10
Average:	338.05	218.65	51.75

B. *Crossover rate*

The test conducted here involves the crossover rate that will have lesser iteration. The population length is set as 1000, the mutation chance is 1 and the max iteration will be 3000. The maximum value of crossover rate is 0.5 because this is the swapping rate of two chromosomes. When the swapping rate is 0.6, it is actually the same as 0.4 since two chromosomes rate will be separated into two.

In Table II, the number of iterations in different crossover rates are shown. When the crossover rate equals to 0, the only chance to change the chromosome will be only mutation chance. This makes the iteration become many and time taken become longer. While crossover rate is higher, the good chromosome will remain and get to the next generation. So, the number of iterations will become lower.

TABLE II. Crossover rate result

NO	NUMBER OF ITERATIONS		
	CROSSOVER RATE = 0.1	CROSSOVER RATE = 0.3	CROSSOVER RATE = 0.5
1	323	732	27
2	2650	0	104
3	128	426	422
4	2890	30	1058
5	953	703	290
6	647	1529	26
7	2975	505	382
8	6	1032	198
9	1523	1299	1
10	86	376	359
11	705	49	621
12	2808	403	10
13	1379	3	926
14	76	830	603
15	849	499	431
16	2677	2031	185
17	902	1394	68
18	740	2245	239
19	2034	38	0
20	92	127	493
AVERAGE	1222.15	712.55	322.15

C. *Population Length*

The tests conducted here involve the population length that will have lesser iteration and the population length that has a higher chance to have a chromosome representing the optimal solution in the initial state. The max iteration for the runs here is 1000, the mutation chance is 1 and the crossover rate is 0.5.

In Table III, the average of iterations with the population length of 1000 is lesser compared with the population length of 10 and the population length of 100. This is because when you set crossover rate to 0.5 and mutate chance to 1, you can compare fitness values for 500 new chromosomes generated from the population length of 1000 in one iteration. However, you can only compare fitness values for 5 new chromosomes generated from the population length of 10 in one iteration. Therefore, with the greater number of chromosomes that will be compare fitness values in one iteration, the greater number of population length can have lesser number of iterations.

Moreover, the population length of 1000 has a higher chance that the population of chromosomes generated from the initial state will contain a chromosome representing the optimal solution. This is because the number of chromosomes that will be generated from the initial state for the population length of 1000 is greater compared with the population length of 10 and the population length of 100.

TABLE III. Population length result

No	Number of Iteration		
	Population Length = 10	Population Length = 100	Population Length = 1000
1	237	0	615
2	481	182	473
3	323	460	681
4	27	84	242
5	695	605	9
6	2	89	3
7	63	362	0
8	4	624	178
9	890	56	67
10	17	67	0
11	236	4	14
12	884	614	5
13	85	1	149
14	616	751	531
15	833	41	557
16	23	224	66
17	712	21	265
18	58	32	384
19	12	127	177
20	961	372	0
Average:	357.95	235.8	220.8

D. *Discuss on implementation*

From the output which are shown in the table above, we found that the largest the population the less of the iteration will be generate which mean if we assign the largest number of populations to the code, the less time will be needed to wait before the solution are found by the algorithm. This is because the more population means the more chromosomes can be selected by the algorithm in the same generation which will let the algorithm find the elite in the generation. For example, China is a big country which are having 1.47 billion of population, hence if only 1% of the people will be an elite, they are still having a lot of elites. Which means the more population you have the more possibility the generation will have an elite.

The other parameters which can make a lot of impact to the iteration generated will be crossover rate. Table 3 shows that if the others parameter is the same the higher chance of mutation will equal to the less time needed to spend waiting for the result. If the crossover rated is 0, the only chance of change will be the possibility of the mutation rate. It is a very simple math question if the crossover rate is 1 (0% of chance) and the mutate rate is 1.5 (0.5) the possibility of change is just 1.5, however is the crossover rate is 1.5 (0.5) and the mutated rate is maintaining the possibility of the change will become 2.25. Hence, the best mutation rate and crossover rate will all be 0.9 and the length of population will be 1000. It is possible to make the length of population even larger, but the computer's RAM must be large enough to store the population, otherwise we recommended to use 1000 and no larger population.

VII. Conclusion

In this paper, it is shown that genetic algorithm can find out the solution of the 8-puzzle problem. The time taken to solve the problem and the number of iterations will change depending on the parameters changing. Although genetic algorithms can solve the problem effectively, but we can know that the time taken to solve it is much longer according to the result. So, it is not really efficient if the crossover rate or mutation chance is low. Implementation of genetic algorithms will be a good choice for problem solving if time allows.

VIII. Acknowledgment

We would like to thank Mr. Zailan Arabee Abdul Salam for providing us with the information we need and helping us when we are doing our research. Mr. Zailan also gives us some reliable comments which are helping us to improve the quality of our research on genetic algorithms and other AI-related knowledge.

References

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Amsterdam University Press.

Melanie, M. (1999). An Introduction to Genetic Algorithm. In *A Bradford Book The MIT Press* (Vols. 8–9). Massachusetts Institute of Technology. https://www.boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf.

Mofan, M. (n.d.). Evolutionary-Algorithm. MofanPython. Retrieved September 1, 2022, from https://mofanpy.com/tutorials/machine-learning/evolutionary-algorithm/.

Richard, J. (n.d.). *8 Puzzle background*. University of Minnesota Duluth. Retrieved October 12, 2022, from https://www.d.umn.edu/%7Ejrichar4/8puz.htm.l

Shaban, R. Z., Alkallak, I. N., & Sulaiman, M. M. (2010, September 1). Genetic Algorithm to Solve Sliding Tile 8-Puzzle Problem. *JOURNAL OF EDUCATION AND SCIENCE*, *23*(3), 145–157. https://doi.org/10.33899/edusj.2010.58405.

Hor, S. H., Yan, M. K., Sim, Y. S., Tan, S. J., & Abdul Salam, Z. A. bin. (2022). Snake Game: A genetic neural network approach. Journal of Applied Technology and Innovation, 6(1), 51–57.

Yuen, M. C., Yeong, L. W., Kang, E. C. Y., Syed, S. Q., & Abdul Salam, Z. A. (2021). Investigating parameters of genetic algorithm and neural network on classic snake game. Journal of Applied Technology and Innovation, 5(2), 7–11.

Goldberg, D. E. (1989, January 11). *Genetic Algorithms in Search, Optimization and Machine Learning* (13th ed.). Addison-Wesley Professional.

Bhasin, H., & Singla, N. (2012, August). Genetic based Algorithm for N – Puzzle Problem. International Journal of Computer Applications, 51(22), 0975 – 8887. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.258.7927&rep=rep1&type=pdf

Genetic Algorithm (Tutorial). (2017, August 12). [Video]. YouTube. Retrieved September 1, 2022, from https://www.youtube.com/watch?v=9ExCPd918Yk&t=306s

Konar, A. (2018, October 8). *Artificial Intelligence and Soft Computing | Behavioral and Cognitive*. Taylor & Francis. Retrieved October 12, 2022, from https://www.taylorfrancis.com/books/mono/10.1201/9781315219738/artificial-intelligence-soft-computing-amit-konar.

Noury, Z. (2020, February 29). *Genetic Algorithm to Solve Sliding Tile 8-Puzzle Problem*. GitHub. https://github.com/zaraanry/8-Puzzle.