Solving Sudoku with a Genetic Algorithm

Maulik Shah
Institute of Artificial Intelligence, The University of Georgia.
Athens, GA. 30602, USA.
Email: mns28652@uga.edu

This paper discusses about a Genetic Algorithm (GA) designed for solving the Sudoku puzzles. The GA uses adaptive recombination and adaptive mutation with informed operators to solve the given game playing problem effectively and efficiently. Use of island model in this GA has improved performance significantly. As result, the GA is able to solve the easy Sudoku puzzles with ease, although it staggers to find solutions as the difficulty level increases.

Keywords— Sudoku, Genetic Algorithm, Island Population Model, Intelligent Operators.

I. INTRODUCTION (SUDOKU)

Game playing has recently become a prominent area in Artificial Intelligence(AI) for research and development of different algorithms. From rock-paper-scissor to Checkers [1,2], there are many different games, used as a subject to prove the efficiency of AI algorithms, designed specifically from Evolutionary Algorithms family. Sudoku is also a well-known puzzle around the world [3] and have a very straight forward logic to find a solution, which is a suitable quality for being applied with a GA.

Sudoku is made of 9*9 square grid, with 9 sub-partitioned grids, which will be called as boxes throughout this paper, where a box is a 3*3 square grid. The solution to the problem is to have numbers in the range from 1 to 9 filled in the main grid, such that neither a row nor a column nor a box would repeat any number. Indeed, the solution will have 9 pairs of 1-9 digits, which are neatly paired to avoid any repetition of elements within a row or a column or a box at same time.

Generally, in a Sudoku puzzle, there are always some fixed numbers, which cannot be removed from their base positions. Also, the solution must contain these numbers intact. As shown in the **Figure 1,** the Sudoku puzzle is given with some fixed numbers, while the remaining blank squares are to be filled by the player as a seeker to the solution. **Figure 2** gives the most suitable solution to the given problem.

Sudoku in many way is an idle problem for game playing, due to its moderate difficulty level in the game playing domain as well as a clear direction to solve it. Efficiency of any algorithm designed for Sudoku can be measured by its solution probability and computation cost to find a solution. Different levels of difficulty can be created, just by adding or removing some fixed numbers.

Figure 1: Sudoku Puzzle

	5	2			6			
1	6		9					4
	4	9	8		3	6	2	
4						8		
	8	3	2		1	5	9	
		1						2
	9	7	3		5	2	4	
2					9		5	6
			1			9	7	

Figure 2: Solution to the given Sudoku Puzzle

3	5	2	4	7	6	1	8	9
1	6	8	9	5	2	7	3	4
7	4	9	8	1	3	6	2	5
4	2	5	6	9	7	8	1	3
6	8	3	2	4	1	5	9	7
9	7	1	5	3	8	4	6	2
8	9	7	3	6	5	2	4	1
2	1	4	7	8	9	3	5	6
5	3	6	1	2	4	9	7	8

II. Introduction (Genetic Algorithms)

Genetic Algorithms(GA) are from the family of Evolutionary Computation research area in computer science. These algorithms get their inspiration from natural evolution. [4] As mentioned by Holland, the father of GA, Genetic Algorithms are computer based optimization methods, which uses Darwin's theory of evolution.

GAs use a primary structure as a representation of the problem, which is called as a chromosome. There are several individual solutions in the initial population created for optimization. The solution evolves over time using recombination and mutation; this process creates another set of improved gen-set. The algorithm stops when either solution is found or some termination condition is matched.

III. Related Work

There has been some work done for playing Sudoku using Evolutionary Computation algorithms. Timo Mantere and Janne Koljonen from University of Vassa have given a straight forward GA approach for solving the Sudoku puzzle with a representation of array segments. They also rated the Sudoku puzzle and generated puzzles using GAs. [5] They used GA with simple mutation and recombination operators, but the algorithm lacked variety to achieve the optimum results.

Harmony Search(HS) for Sudoku is also a GA technique to solve the Sudoku puzzle described by Zong Woo Geem from John Hopkins University. The idea of HS can also be applied to different constraint satisfaction problems. [6] Dr. John M Weiss has also tried to use GAs with a representation of n 1-D arrays for n*n Sudoku puzzle. He used Generational GAs to produce solution population, while 'restart' strategy in case of finding a local optimum. [7]

While the GA used in this paper uses the group of 1-D arrays to solve the puzzle, Sato, Yuji, and Hazuki Inoue [8] used a 2-D representation for solving it. They considered a box as the building block of the problem and performed operations on boxes to find a solution. Although, the representation was unique from the previously mentioned approaches, it did not excel the capability of the GA significantly.

Perez Meir and Tshilidzi Marwala [9] tried to compare efficiencies Cultural GAs, simulated annealing and repulsive particle swarm optimization on Sudoku. Their hybrid approach to use Simulated Annealing and Genetic algorithms to speed up the process was found best among the used methods. In the GA used here, there is an attempt to boost the parallelism using the island population model.

IV. The Proposed Methodology in GA

In GA terminology, Sudoku is a special kind of constraint satisfaction problem. Primary areas of constraint satisfaction problem are faced in Engineering design, where for completing a job, one must satisfy all the constraint for a given environment. This GA is specifically designed in terms of solving a 9*9 Sudoku puzzle, although the approach can be applied to different game playing and engineering problems, as just mentioned. Primary objective of it is to create a player that can efficiently and effectively solve the different levels of the Sudoku puzzles.

The GA used here is a Steady State GA(SSGA), which produces one child at a time and replace it with an individual in the current population. One more thing about the algorithm is that, it is not a learner; meaning it does not use past data to solve the current problem.

The salient feature of this GA is the heuristic mutations for guiding the algorithm, while the island model increases parallelism in GA; detailed description of the same is as given below:

A. Representation

The representation or chromosome in this GA is a set of permutation. In technical terms, there are total 9 arrays of size 9, where an array does not repeat any element and which also represents a row in a Sudoku puzzle. As we are already taking care of no repetition constraint in the rows by considering them as permutations, the computation required for finding a solution, would be reduced by a dimension. The representation for the puzzle given in the introduction section can be given as:

Figure 3: Representation of individual

	Elements								
Arrays	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	3	5	2	4	7	6	1	8	9
2	1	6	8	9	5	2	7	3	4
3	7	4	9	8	1	3	6	2	5
4	4	2	5	6	9	7	8	1	3
5	6	8	3	2	4	1	5	9	7
6	9	7	1	5	3	8	4	6	2
7	8	9	7	3	6	5	2	4	1
8	2	1	4	7	8	9	3	5	6
9	5	3	6	1	2	4	9	7	8

B. Fitness Function

As the representation reduces the 'row' dimension for constraint satisfaction, there are only two other dimensions, column and boxes remain to satisfy. To accommodate this situation, the fitness function is given a structure of penalty calculation for each violation in the constraint, which is occurrence of any duplication in either a column or a box would add one in the penalty calculation.

Precisely the penalty calculation is done in the following two ways:

- Add one point penalty for each number occurring more than once in a column for its each unique occurrence. Also, add one point penalty for each number, which does not occur in a column.
- Same way for the 3*3 sub-grid or a box, add one point penalty for each number occurring more than once in a box for its each unique occurrence. Also, add one point penalty for each number, which does not occur in a box.

As already mentioned, when the GA finds an individual with a 0-fitness value, our quest for the solution is over.

C. Initialization

The population size for this GA is 100 individuals, which is static through the whole search of solution. The population is initialized randomly without any bias. Also, as already defined in the problem, the fixed numbers in the given puzzle cannot move from their original positions; the initialization process also takes care of this constraint.

D. Recombination

The recombination operator uses 2 or more parents to create one or two children, but in this GA, only 2 parent strategy is being used to produce one offspring after each recombination. The GA uses the recombination operators which are suitable for the permutation representation specifically.

Apart from that, using multiple recombination operator decreases the chances of finding a local optimum in the solution. The recombination operators used here are applied partially to the array, as the fixed elements can't be moved from their positions. So, one new array of non-fixed numbers from the original array is created, which is the only part in array that pass through the recombination operation. In the end, while creating a new child, the fixed numbers are copied first into the array and then the newly created sub-array is copied.

For individuals going through the recombination operation, each row is applied to it separately. Also, a row of an individual only recombines with same ordered row of the other individual. The recombination operators used in this GA are mentioned as below:

- Partial Match Crossover (PMX): The PMX recombination is generated by first copying some portion of the first parent into the child. For the same portion in the 2nd parent, the elements are replaced with the first parent's position of similar values in a cyclic fashion and copied into the child with relative positions. The remaining elements from 2nd parent is copied directly to the child with the same positions as in the parent.

- Uniform-Order Crossover: This is a unique kind of crossover which uniformly copies the alternative elements of the first parent to the child, while finds the remaining ones from the 2nd parent by scanning through the whole row and copying the elements in the vacant places in the child.

E. Mutation

The mutation operators change or flip some values in the chromosome to evolve individuals. Here, there are total 4 mutation operators used for evolving the individual including two informed operators. [10] None of the mutation is applicable to the fixed element of the puzzle. All of them are as mentioned below:

- Swap Mutation: The swap mutation is simply swapping some values in the permutation except the fixed ones. If this mutation is applied to an individual, all the rows of that individual are applied with this mutation separately.

- Scramble Mutation: The scramble mutation is scrambles a certain portion of the permutation in the chromosome, to unorder the given region. If this mutation is applied to an individual, all the rows of that individual are applied with this mutation separately.

- Column Mutation(Informed): After some analysis, there has been a conclusion that, the columns with the maximum number of fixed elements are always chosen to address first by the Sudoku players. Adding this idea as a heuristic in the GA would give a direction to the search. This mutation tries to minimize the penalty of the columns, which has fixed numbers more than half of their size. In this process the elements occurring more than once in a column of an individual are replaced with the elements which does not occur in the same. The replacement of an element is done within the permutation of that element.

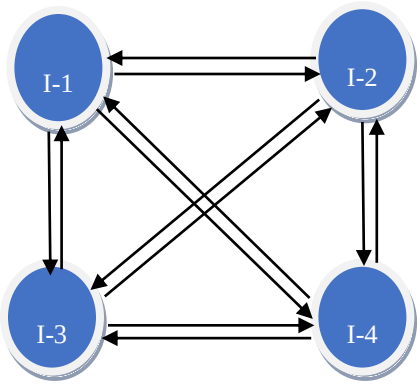
- Box Mutation(Informed): Same as column mutation, Box mutation tries to minimize the penalty of boxes in an individual, which have fixed numbers more than half of their size. In this process the elements occurring more than once in a box are replaced with the elements which does not occur in the same. The replacement of an element is done within the permutation of that element.

F. Population Model

The population is divided into certain numbers of sub-populations called 'islands', which is holistically termed as "Island Model" in GA. [11] The island model converts the GA into a **linearly separable** space, which reduces the computation time significantly. The most important parameters of the island models are migration interval and migration size. At some regular interval in execution, the islands exchange their individuals amongst each other, which is migration interval; while the number of individuals exchanged are termed as the migration size.

In this GA, the migration interval is $1/4^{th}$ of the population size, with 4 islands in the model. Also, in case of migration size, the GA follows the elite strategy; at every migration interval, each island transmits best individuals to the other islands with a migration size of $1/10^{th}$ of the island population, while the other islands replace them with the worst individuals in their population.

Figure 4: Island Model. The islands exchange some individuals (migration size) at every migration interval.



The GA discussed here uses combinations from 4 different mutation strategy on each island for evolution, while the recombination is chosen randomly. The first island uses scramble mutation with a 90% probability and column mutation with the remaining 10% probability; others use the remaining combination of different mutation operators, same as the first island. It must be noted that, the heuristic or informed mutations are used with only 10% probability to prevent the algorithm to get stuck in a local minimum. Also, the probabilities mentioned here are relative to the overall mutation probability, which is the mutation probability mentioned specific to each island exists provided mutation occurs in an evolution cycle.

G. Recombination and Mutation Probabilities

The recombination and mutation probabilities are adaptive in nature. By default, the Recombination probability follows this equation:

$$P(\text{recombination}) = 0 + \left(\frac{\text{Current Evolution Cycle}}{\text{Maximum Evolution Cycle} * 2}\right) \quad (1)$$

, while the below equation gives probability of the mutation:

$$P(\text{mutation}) = 1 - \left(\frac{\text{Current Evolution Cycle}}{\text{Maximum Evolution Cycle}}\right) \quad (2).$$

As clearly visible, the recombination probability increases by the time from 0 to 0.5, while the mutation probability decreases by the time from 1 to 0.

One special condition is included in the algorithm in case, the diversity falls below significant level, which is if there are more than 66.33% of individuals holding the same fitness values in an island, then the mutation ratio is increased to 90% for certain iterations to prevent the GA from ending up in local minimum.

H. Parent Selection:

Parent selection in GA is a single tournament selection, while also an adaptive process to the environment. As the GA evolves, the selection pressure increases with time, by increasing the tournament size. Certain number of individuals are selected randomly for a single tournament and two of them are chosen to create the offspring in the end. The formula for deciding the tournament size can be said as

$$\text{Tournament Size} = (4) + \text{Mod}\left(\left(10 * \frac{\text{Current Evolution Cycle}}{\text{Maximum Evolution Cycle}}\right)\right) \quad (3)$$

, which clearly depicts the tournament size can vary from 4 to 14 throughout the search.

Also, a special case to maintain the diversity in the population, if there are more than 66.33% of individuals with similar fitness values, then the parent selection is made free of selection pressure; which is the tournament size is decreased to 2 that gives each selected individual a definite chance to be a parent.

I. Survivor Selection:

Survivor selection uses two different techniques for replacing the offspring with the individual in the population.

The strategy for the replacement mirrors the SSGA approach, which is as described below:

- Replace the worst: As an elite strategy, in this case the newly generated offspring is compared with the worst individual in terms of fitness value. If the offspring's fitness is better than the worst individual, it replaces the later in the current population.
- Replace from the bad half: In this case, the bad half of population or the least fit 50% population is used for survivor selection. One random individual is selected from the 'bad half' of the population and is compared with the newly created offspring, if the offspring has a better fitness than this random individual, then it replaces the later. A noteworthy thing is that, this strategy does not guarantee a replacement, even though the offspring is better than a big chunk of population.

Also, as a special case, if the diversity falls below certain level, which is if more than 66.33% of individuals holds same fitness value, then the newly created child is replaced with any random individual in the population.

J. Termination Condition:

A simple termination condition to this GA is to find a solution, that is to find an individual with a zero fitness. Otherwise, if the GA runs for 200,000 cycles without any solution, then it must terminate. For the hardest puzzles, this number is 400,000.

V. Implementation

The project implementation is done in Java SE 1.8, with no external libraries included. Source code for this implementation is publicly available on the URL:

https://github.com/mauliknshah/sudokuwithGA/tree/multiThreadedFinal

The code base also gives some more versions, which includes the GA without the island model and without recombination.

VI. Results and Performance

As unpredictable in nature, GAs are showing the same properties even though used with many different techniques and an island model. As the difficulty level increases, the GA is finding more local minimums. The premature convergence ratio is as given in the Table 1.

The difficulty level is derived from the 'Sudoku' online application on android smart phone, provided free by Everyone on Google play store. [12]. The 'very easy' problem set are created artificially from the easy puzzles by adding some more fixed numbers, to derive efficiency of the algorithm. The results are derived after taking 25 outputs from each difficulty level puzzles.

Table 1: Premature Convergence Ratio

Problem Type	Premature Convergence Ratio
Very Easy	0%
Easy	4%
Medium	76%
Hard	84%
Very Hard	96%

Conclusively, the Sudoku puzzle have a sharp curve of solution distribution, which tends to find a local minimum on very frequent occasions. One of the best strategy to prevent this from happening is to start from the scratch every time a local minimum is found, although it is computationally very expansive (Current GA does not use this strategy).

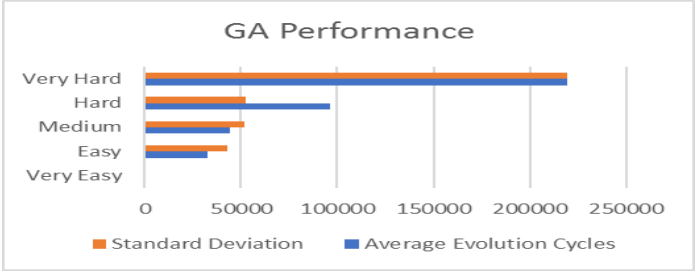
Going ahead with the performance, the measure for performance for any GA is the evolution cycles it takes to find a solution. Apart from that, if the GAs are made parallel, how the computational time is being saved, is also a major observation for performance. For this GA, the number of iterations or evolution cycles increase significantly with the increase in difficulty level. Table 2 shows some statistics about the number of iterations taken by the GA to solve the problem. Notably, the premature convergence cases are not considered here:

Table 2: Statistics of the GA Performance

Note: The cycles are calculated separately for each island. If all parallel cycles are considered as one, the numbers would reduce to the 1/4th (number of islands) of the original cycles.

Problem Type	Average Evolution Cycles	Standard Deviation
Very Easy	544	586
Easy	32445	42896
Medium	44000	51559
Hard	96326	52792
Very Hard	219364	219364

Figure 5: Comparative graph for the statistics of GA Performance



If the comparison is done between the GA without island model and with island model, it would turn out to be very significant in terms of time consumption. The island model reduces the computation time almost to a fraction of the number of islands, compared to the GA without any parallelization. The time consumption is not mentioned, as it depends on the machine where the GA is running.

VII. Conclusion And Future work

GAs are a very powerful tool to solve difficult gaming problems, although they do not guarantee the solution. In this case also, the uncertainty of the results hold to a significant level. Still, the GAs give promising outputs to go forward in this domain. Conclusively, the GAs can be applied to different game playing algorithm with uncertainty and a large search space, which can give some great results.

The island model is very useful to make the GA parallel, while the heuristic(informed) mutations give a whole new direction to the GA. The current implementation is with adaptive parameters, which can be converted to self-adaptive to the environment. Apart from that, the fine-grained population model can also be implemented to check parallelism effectiveness.

References

[1] Fathelalem F. Ali ,"**Playing the Rock-Paper-Scissors Game with a Genetic Algorithm**", Evolutionary Computation, 2000.

[2] Kumar Chellapilla, David Fogel ,"**Evolving an Expert Checkers Playing Program without Using Human Expertise**", IEEE Transactions on Evolutionary Computation (Volume: 5, Issue: 4, Aug 2001), page 422 – 428.

[3] Wikipedia,Sudoku.Link:http://en.wikipedia.org/wiki/Sudoku

[4] Eiben and Smith. Springer-Verlag, "**Introduction to Evolutionary Computing**". Corrected 2nd printing, 2007, ISBN: 978-3-540-40184-1

[5] Timo Mantere and Janne Koljonen, "**Solving and Rating Sudoku Puzzles with Genetic Algorithms**". Evolutionary Computation, 2007. CEC 2007. IEEE Congress.

[6] Zong Woo Geem, "**Harmony Search Algorithm for Solving Sudoku".** 11th international conference, KES 2007 and XVII Italian workshop on neural networks conference on Knowledge-based intelligent information and engineering systems, page 371-378.

[7] Weiss, John M. "**Genetic Algorithms and Sudoku**." (2009).

[8] Sato, Yuji, and Hazuki Inoue. "**Solving sudoku with genetic operations that preserve building blocks**." Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games. IEEE, 2010.

[9] Perez Meir and Tshilidzi Marwala. "**Stochastic optimization approaches for solving Sudoku.**" arXiv preprint arXiv:0805.0697 (2008).

[10] Khaled Rasheed, Haym Hirsh. "**Informed operators: Speeding up genetic-algorithm-based design optimization using reduced models**", Genetic and Evolutionary Computation Conference (GECCO),2000, page628-635

[11] Darrell Whitley , Soraya Rana , Robert B. Heckendorn ,"**The Island Model Genetic Algorithm: On Separability, Population Size and Convergence**", Journal of Computing and Information Technology(1998), page 33-47.

[12] Sudoku by Everyone available on Google Play Store , Link: https://play.google.com/store/apps/details?id=com.http.sudoku&hl=en