World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:8, No:5, 2014

A Retrievable Genetic Algorithm for Efficient Solving of Sudoku Puzzles

Seyed Mehran Kazemi, Bahare Fatemi

***Abstract*—**Sudoku is a logic-based combinatorial puzzle game which is popular among people of different ages. Due to this popularity, computer softwares are being developed to generate and solve Sudoku puzzles with different levels of difficulty. Several methods and algorithms have been proposed and used in different softwares to efficiently solve Sudoku puzzles. Various search methods such as stochastic local search have been applied to this problem. Genetic Algorithm (GA) is one of the algorithms which have been applied to this problem in different forms and in several works in the literature. In these works, chromosomes with little or no information were considered and obtained results were not promising. In this paper, we propose a new way of applying GA to this problem which uses more-informed chromosomes than other works in the literature. We optimize the parameters of our GA using puzzles with different levels of difficulty. Then we use the optimized values of the parameters to solve various puzzles and compare our results to another GA-based method for solving Sudoku puzzles.

***Keywords*—**Genetic algorithm, optimization, solving Sudoku puzzles, stochastic local search.

I. Introduction

SUDOKU is a Japanese word consisting of two parts, su and doku, where the first part means number and the second means single. It has been originally called Number Place which indicates the nature of the game, in which numbers should be placed in appropriate places in a grid. Indeed, it is a logical game which has attracted both young and old people.

The first Sudoku was published in a puzzle magazine in USA, 1979 [1]. Having a very challenging and addictive nature, it has spread wildly all over the world. The objective of the game is to take a 9×9 grid and fill in the open spots with numbers from 1 to 9 so that each column and each row of the grid contains each of the numbers only once. Furthermore, each of the nine 3×3 sub-grids that together compose the total 9×9 grid (also called boxes, blocks, regions, and sub-squares) must contain all of the digits from 1 to 9 only once.

Fig. 1 is an example of a Sudoku puzzle where 21 of the cells initially contain a number. Fig. 2 represents the solution to the Sudoku puzzle in Fig. 1.

As in Fig. 1, grids come with partially filled columns and rows depending on the difficulty level of the puzzle. For instance, in a beginner puzzle several numbers will be already in the grid. In more advanced puzzles, only a few numbers are in the grid. But this is not the only factor for determining the level of difficulty. There are also other factors such as the number of occurrences of each number in the initial grid, the position of numbers in the initial grids, etc. which are very effective in the difficulty level of a Sudoku puzzle.

In this paper, we will examine the problem of proposing an algorithm to solve Sudoku puzzles. We will introduce genetic algorithms and then propose a way of applying this algorithm to the problem of solving Sudoku puzzles efficiently. We evaluate out model on Sudoku puzzles having different difficulties and compare our results with another GA-based method in the literature.

The rest of the paper is organized as follows: Section II reviews the works available in the literature. Section III describes our formulation of genetic algorithm to solve Sudoku puzzles. Section IV is devoted to optimizing the parameters of the GA. In Section V, we represent our experiments and results and compare them to another existing work which also uses genetic algorithm. Finally, Section VI concludes the paper and points out some future directions.

II. Literature Review

Solving the generalized Sudoku problem is NP-complete, as has been shown in [2]. Therefore, we cannot hope to find an algorithm with polynomial time for all puzzles, unless P = NP [3]. This means that there will be possibly many instances that cannot be solved without one kind of search also being necessary [4]. Researchers have tried to propose heuristic algorithms or use existing local search methods such as Genetic Algorithm (GA) to solve Sudoku puzzles.

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

Fig. 1 An example of a Sudoku puzzle

Seyed Mehran Kazemi is with the Computer Science Department, University of British Columbia (e-mail: smkazemi@cs.ubc.ca).
Bahare Fatemi is with the Computer Engineering and IT Department, Amirkabir University of Technology (e-mail: b.fatemi@aut.ac.ir).

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

Fig. 2 The solution to the Sudoku puzzle in Fig. 1

Pencil and pen [5] algorithm is one of the proposed methods in the literature for solving Sudoku puzzles. This algorithm is a tree-based search algorithm which backtracks on a tree until a solution is found. Another well-known work in the literature uses a meta-heuristic technique [4]. It is a stochastic search based algorithm, which uses simulated annealing and is able to solve logic-solvable puzzle-instances efficiently. Stochastic optimization is another approach which researchers have tried to apply to this problem. In this approach, the Sudoku puzzle is solved using stochastic local search techniques. Cultural Genetic Algorithm (CGA), Repulsive Particle Swarm Optimization (RPSO), Quantum Simulated Annealing (QSA) and the Hybrid method that combines Genetic Algorithm with Simulated Annealing (HGASA) [6] are some of the methods which use stochastic optimization approaches.

One of the methods to solve Sudoku puzzles, as well as any other problem which requires searching a huge state space, is genetic algorithm [7]. Genetic algorithms are computing algorithms based on Darwinian evolution [8] which are constructed in analogy with the process of evolution [9]. They seem to be useful for searching very general and poorly defined spaces. It is not, however, completely known for what kinds of spaces they work best, but there are efforts to figure it out [10]. First pioneered by John Holland in the 60s, GA has been widely studied, experimented and applied to many fields in the real world [11]. Not only does GAs provide alternative methods to solve problems, they consistently outperform other traditional methods in most of the problems. Many of the real world problems involve finding optimal parameters, which might be quite difficult for traditional methods, but ideal for GAs.

KedarNath Das [12] has used GA to solve Sudoku puzzles. In his paper, he proposed a GA algorithm, in which he models his fitness function in a way that considers the puzzle-character-dependant constraints and used GA to solve the problem. He called his method retrievable GA, or simply ret-GA, because after a certain number of iterations, the population is reinitialized in his method. Since we also use this idea of reinitializing the population, like in [12], we also use the term retrievable for our GA method.

Timo Mantere and Janne Koljonen [1] also used GA to solve Sudoku puzzles. In their paper, they create chromosomes of the size of the Sudoku puzzle with two special conditions. The first condition is that the numbers initially given in the grid should be fixed in their corresponding positions in the chromosome. The second one is that no number can appear twice in each row. While these two conditions cause chromosomes to contain some information, they didn't get promising results. One possibility for this may be the fact that their chromosomes are not sufficiently informed. Generating more-informed chromosomes may help solving Sudoku puzzles using GA more efficiently.

In this paper, we try to generate chromosomes containing more information. We add one more condition to the conditions specified in [1] which is, in addition to each of the rows; chromosomes are also forbidden to have common numbers in each 3×3 sub-square.

In the next section, we represent how different genetic algorithm operations, such as crossover and mutation, can be applied to these chromosomes. We use some parameters in different operations and try to optimize our parameters empirically so as to make the proposed genetic algorithm more efficient.

III. Proposed Method

In this section, we describe how we use genetic algorithm to solve a Sudoku puzzle. We explain different functions and operations which are required to specify how genetic algorithm is used for a problem.

A. Representation

We can see a Sudoku puzzle as a 9×9 matrix, where each row, column and each 3×3 sub-square must contain all numbers from 1 to 9 only once.

For representing the Sudoku puzzle in GA program, we represent each chromosome as a two dimensional integer array of size 9×9, with these conditions: 1) the numbers predefined by Sudoku setter should be fixed in their corresponding position in the chromosome and cannot be changed, 2) each row of the chromosome should contain numbers from 1to 9 only once, and 3) each 3×3 sub-square of the general grid should also contain numbers from 1 to 9 only once.

B. Fitness Function

After testing different functions, we considered the fitness function to be the maximum possible number of mismatches in a chromosome minus the total number of mismatches in the given chromosome, where by mismatch we mean a case where a number appears more than one time in a column. We can see that the maximum possible number of mismatches that a Sudoku puzzle can have in the worst case is 81. Using this fitness function, it is obvious that as the number of mismatches reduce the fitness of the chromosome increases, which is intuitively reasonable. A chromosome with a fitness

of 81 is the solution to our Sudoku puzzle and the process will be stopped once such a chromosome is found.

Using this fitness function, most chromosomes tend to have fitness values in a small range. That's because random chromosomes hardly have lots of (or few) mismatches, having an initial fitness ranging from 40 to 60. This can deteriorate the efficiency of the GA by giving high probabilities of being selected as a parent to all the chromosomes.

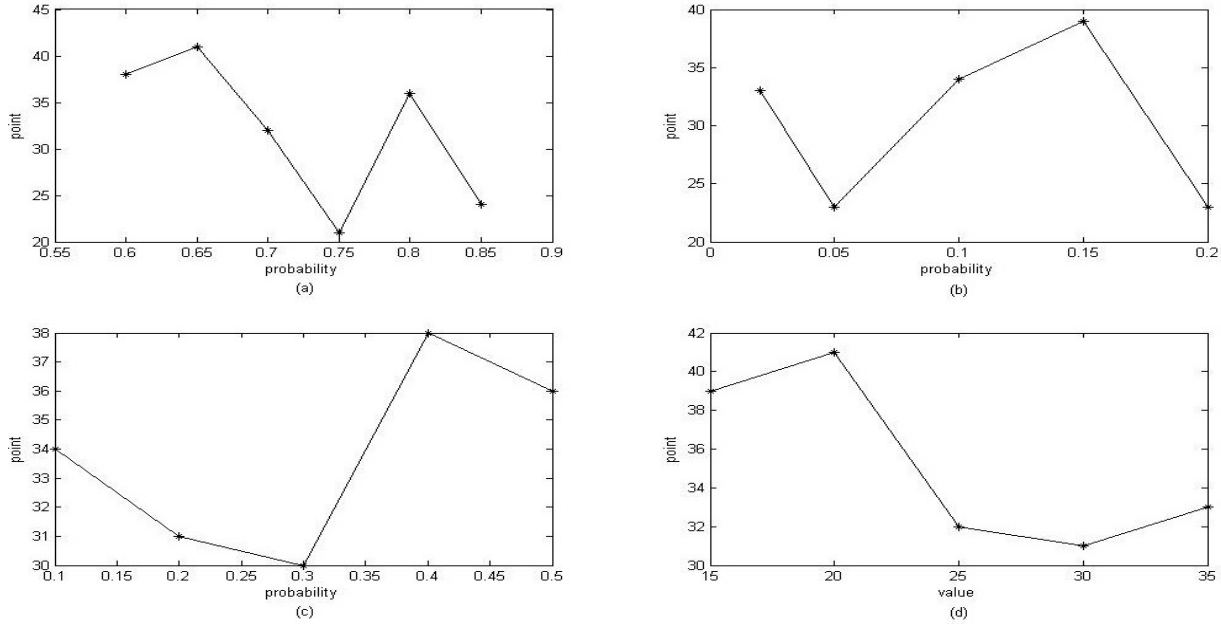

Fig. 3 The results obtained for optimizing (a) crossoverProb, (b) mutationProb1, (c) mutationProb2, and (d) maxValue

Therefore, we use max-min normalization to project the fitness values of chromosomes to a smaller interval of [5, max Value], where we optimize the parameter maxValue empirically. Having fitness values in this interval, chromosomes with lower fitness values are less likely to be chosen as a parent.

C. Crossover

Before explaining about crossover operation, it is necessary to see that we can consider a Sudoku puzzle as a 3×3 matrix of sub-squares. In the rest of the paper, we call each row of this matrix a sub-square row. Because of the constraints we imposed on the chromosomes, our chromosomes have no mismatches in each sub-square row. Therefore, to do the crossover operation, first of all we choose two chromosomes from our population as parents, with the probability of selection being proportional to their fitness values. Then we generate two daughters with the first one having randomly two of her sub-square rows from one of the parents and one from the other, and the second daughter having the sub-square rows of the parents not used by the first daughter. These two daughters go to the next generation.

Note that crossover operation happens by a probability crossoverProb which we optimize empirically. If the operation does not happen, the parents go to the next generation without any changes.

D. Mutation

We use two different mutation operations in our formulation. The first one is applied to a gene (a cell in the matrix) of the chromosome by probability mutationProb1. If this mutation is happening to a gene, another random non-predefined gene in the same row is selected and the values in these two genes are swapped. Then if the new chromosome is not valid according to our constraints, we change the same numbers of these two genes in the other two matrix rows of the same sub-square row to get a valid chromosome.

The second mutation is applied to each of the sub-square rows with probability mutationProb2. Each time a mutation is happening to a sub-square row, that row is completely replaced by a newly generated row.

E. Other Criteria

We set the maximum number of generations for solving a single Sudoku puzzle to be 100000. Since we tend to compare our results with the results of [1] and the maximum number of generations in [1] was set to be 100000, we chose the same number to make our comparisons more reasonable.

As we mentioned earlier, we use a retrievable GA, in which after each ten thousand generations, if we haven't found a solution, all chromosomes are killed and a new population of them is produced.

TABLE I
OBTAINED RESULTS FROM USING OUR PROPOSED GA FOR SOLVING SUDOKU PUZZLES

Difficulty rating	Count	Min	Max	Median	Average	Stddev
1 star	100	12	6691	276	548.43	963.85
2 stars	100	70	10593	697.5	1584.8	2439.8
3 stars	100	84	70146	45965.5	1183.4	15599
4 stars	97	809	90432	40540	35871	23576
5 stars	79	1021	65659	36001	34911.4	19881
Easy	100	11	507	78	97.51	76.6
Challenging	51	135	72202	32444	40141	15011
Difficult	17	1991	90282	27345	33273	24689
Super difficult	20	309	76111	22187	25843	21846

TABLE II
OBTAINED RESULTS FROM USING THE GA METHOD IN [1] FOR SOLVING SUDOKU PUZZLES

Difficulty rating	Count	Min	Max	Median	Average	Stddev
1 star	100	184	23993	917	2466.6	3500.98
2 stars	69	733	56484	7034	11226.8	11834.68
3 stars	46	678	94792	14827	22346.4	24846.46
4 stars	26	381	68253	22297	22611.3	22429.12
5 stars	23	756	68991	17365	23288	22732.25
Easy	100	101	6035	417	768.6	942.23
Challenging	30	1771	89070	17755	25333.3	23058.94
Difficult	4	18999	46814	26162	20534.3	12506.72
Super difficult	6	3022	47352	6722	14392	17053.33

IV. PARAMETER OPTIMIZATION

We have four parameters in our proposed method which we need to find the best values for them. These parameters are: crossoverProb, mutationProb1, mutationProb2, and maxValue. In order to find the best values for these parameters, we tested different combinations of values for them over twenty Sudoku puzzles from www.websudoku.com with different levels of difficulty. Five of these puzzles were easy, five of them were medium, five of them were hard and the other five were evil.

At the beginning, we tested some random combinations of values for these parameters and came up with this initial guess for the parameters: crossoverProb = 0.8, mutationProb1 = 0.2, mutationProb2 = 0.02 and maxValue = 20.

Then to find the optimized values of the parameter, each time we fixed all the parameters except one of them and tried to solve all the twenty Sudoku puzzles for different values of this parameter. We assigned points to the values of this parameter according to the number of puzzles it could solve where an easy puzzle had only one point, medium had two points, hard had three points, and finally an evil puzzle had four points. After summing up the points for each of the values, the value with the maximum total points was selected as the optimized value for that parameter. Then we used this value for this parameter and tried to optimize the rest.

We started optimizing the parameters in this order: crossoverProb, mutationProb1, mutationProb2, and maxValue. Fig. 3 demonstrates the results achieved in the experiments where the horizontal axis is the value to the parameter and the vertical axis is the total points.

We can see that the best values for the parameters are crossoverProb = 0.65, mutationProb1 = 0.15, mutationProb2 = 0.04 and maxValue = 20. We use these values in the next section to run our experiments and compare our results with obtained results from another GA-based method for solving Sudoku puzzles in [1].

V. EXPERIMENTS AND RESULTS

In order to test our proposed method and compare it to other methods, we solved the Sudoku puzzles of different difficulty levels which Timo Mantere and Janne Koljonen [1] used in their paper. These puzzles are from two different sources. The first source categorizes the puzzles into five different groups according to their difficulty levels, where the easiest puzzles are called *1 star* and the hardest ones are called *5 stars*. The second source categorizes the puzzles into four different groups based on their difficulty levels and calls them *Easy*, *Challenging*, *Difficult*, and *Super difficult* to represent their difficulty levels.Like in [1], we ran our GA 100 times for each of the Sudoku puzzles. For each run, if the puzzle could be solved, we recorded the number of generations it took to solve the puzzle. Then we counted the number of times that our GA could solve the puzzle. We also calculated statistical measures such as minimum, maximum, median, average, and standard deviation of number of generations for solving the puzzle in 100 runs. Table I represents the results of our proposed method and Table II represents the results obtained in [1]. We can see from the results that our proposed method is outperforming the previous method in terms of the number of times it has been able to solve a puzzle. While this increment is not very striking for some easier puzzles like *2 stars* puzzle for which the increment is 31%, it is quite striking for more difficult puzzles such as *4 stars* puzzle for which the increment is *71%*. Our method is also outperforming the work in [1] in terms of decreasing the minimum number of generations that has passed to find the solution to a puzzle in most cases. This decrement can be seen for 7 out of 9 puzzles. However, we can see that our algorithm is not totally outperforming the algorithm in [1] in terms of average number of generations. The reason can be the way we optimized our parameters. The only criterion we used for optimizing the parameters was the number of times a puzzle was solved and not the average number of iterations. Including the average as another criterion for optimizing the parameters can result in a decrement in the average number of iterations.

VI. CONCLUSION AND FUTURE DIRECTION

Various methods and algorithms have been proposed to solve Sudoku puzzles. One of these methods is to formulate the problem as a genetic algorithm and try to solve it. Previous attempts of using genetic algorithm seem inefficient because of including only little information in their chromosomes.

In this paper, we proposed a more efficient genetic algorithm by generating more-informed chromosomes. We demonstrated that our method can be used more efficiently than one of the other methods in the literature which also uses

genetic algorithms. We could increase the percentage of solving the puzzles with different difficulties up to 73%.

We could also do a better job in terms of the minimum number of generations which is passed to solve a puzzle. In 7 puzzles out of the 9 puzzles that we used for our experiments, we could decrease this minimum compared to results obtained in [1].

However, we could not do much better than [1] in terms of average number of generations. The reason can be the way we optimized our parameters because we only cared about solving the Sudoku puzzles not the average number of generations.

In future, we can use a different objective function in our parameter optimization which also includes the number of generations it takes to solve the puzzle. We can do the same experiments to see if it can improve our proposed method in terms of the average number of generations for solving the puzzles or not. We may also consider other criterion such as median or standard deviation of they are of high importance to us.

REFERENCES

[1] T. Mantere and J. Koljonen, "Solving and rating sudoku puzzles with genetic algorithms," in New Developments in Artificial Intelligence and the Semantic Web, Proceedings of the 12th Finnish Artificial Intelligence Conference STeP, pp. 86–92, 2006.

[2] Y. Takayuki and S. Takahiro, "Complexity and completeness of finding another solution and its application to puzzles," IEICE transactions on fundamentals of electronics, communications and computer sciences, vol. 86, no. 5, pp. 1052–1060, 2003.

[3] M. R. Garey and D. S. Johnson, Computers and intractability, vol. 174. Freeman New York, 1979.

[4] R. Lewis, "Metaheuristics can solve sudoku puzzles," Journal of heuristics, vol. 13, no. 4, pp. 387–401, 2007.

[5] J. F. Crook, "A pencil-and-paper algorithm for solving Sudoku puzzles," Notices of the AMS, vol. 56, no. 4, pp. 460–468, 2009.

[6] M. Perez and T. Marwala, "Stochastic optimization approaches for solving Sudoku," arXiv preprint arXiv:0805.0697, 2008.

[7] J. H. Holland, Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, 1975.

[8] C. Darwin and J. W. Burrow, The origin of species by means of natural selection: Or, The preservation of favoured races in the struggle for life. Collier Books Nueva York^ eN. YNY, 1962.

[9] J. Heitkoetter and D. Beasley, "The hitchhiker's guide to evolutionary computing: A list of Frequently Asked Questions (FAQ)," USENET: comp. ai. genetic, 1996. Available via anonymous FTP from rtfm. mit. edu:/pub/usenet/news. answers/aifaq/genetic.

[10] E. K. Prebys, "The genetic algorithm in computer science," MIT Undergrad. J. Math, vol. 2007, pp. 165–170, 2007.

[11] J. Li and Z. Zhang, "A learning tool of genetic algorithm," in Education Technology and Computer Science (ETCS), 2010 Second International Workshop on, 2010, vol. 1, pp. 443–446.

[12] K. N. Das, S. Bhatia, S. Puri, and K. Deep, "A retrievable GA for solving Sudoku puzzles," Citeseer, 2012.