A Retrievable GA for Solving Sudoku Puzzles

Kedar Nath Das kedar_nath_dash@yahoo.com
Department of Mathematics, Indian Institute of Technology Roorkee,
Roorkee - 247 667, India

Sumit Bhatia[*] sumit@cse.psu.edu
Department of Computer Science and Engineering, The Pennsylvania State University
University Park, PA-16802, USA

Shubhin Puri shubhin@gmail.com
Department of Chemical Engineering, Indian Institute of Technology Roorkee,
Roorkee – 247 667, India

Kusum Deep kusumfma@iitr.ernet.in
Department of Mathematics, Indian Institute of Technology Roorkee,
Roorkee – 247 667, India

Abstract

In this paper we propose a modification to Genetic Algorithm which we call as *Retrievable Genetic Algorithm* (*Ret-GA*) and use it to solve a given Sudoku puzzle. Although a number of approaches exist for solving a given Sudoku puzzle, very few papers with heuristic approaches are available for obtaining its solution. In this work, the fitness function is modeled in a new way considering puzzle-character-dependent constraints. The GA is named as *Retrievable*, since the population is reinitialized after a certain number of generations in order to escape the local maxima and retrieve the solution in reduced computational time. A set of 9 sample puzzles have been considered for comparison with the previously published results by GA and it is concluded that Ret-GA performs better than GA in most of the cases. In addition we consider a set of other 75 puzzles namely *Easy*, *Medium* and *Hard* (25 of each type) to further test the effectiveness of proposed algorithm.

Keywords

Retrievable Genetic Algorithm, Genetic Algorithm, Sudoku, Evolutionary Computation.

1 Introduction

The word *Sudoku* comes from Japan and consists of the Japanese characters Su (meaning 'number') and Doku (meaning 'single'). Today, it is one of the most popular puzzles attracting young and old alike. Due to its addictive and challenging nature, it has spread like a wildfire throughout the globe and has attracted the attention of many researchers who are trying to design algorithms to solve it by applying varied approaches. Not only the deterministic techniques, but efforts are on for designing heuristics like simulated annealing (Lewis, 2007) and Genetic Algorithms (Mantere and Koljonen, 2006) to solve Sudoku puzzles.

The basis of Sudoku can be dated back to 18th century when great Swiss mathematician Leonard Euler introduced the idea of *Latin Squares* in 1783. A Latin Square is

[*] Work done when author was an undergraduate student in the Department of Electrical Engineering, Indian Institute of Technology Roorkee.

an $N \times N$ table filled with N different symbols in such a way that each symbol occurs exactly once in each row and exactly once in each column. On the other hand, a Sudoku is an $N \times N$ square that is divided into $\sqrt{N}$ sub-squares, each of size $\sqrt{N} \times \sqrt{N}$. Here N is a perfect square and is known as the order of the Sudoku. In the beginning, there are some static numbers (called givens) in the puzzle. The game is to fill all non-givens such that each row, column and sub-square contains each integer from 1 to N exactly once. The difficulty level of the Sudoku puzzle is determined not only by the number of givens (Semeniuk, 2005), but is also dependent on around 20 other factors (Mantere and Koljonen, 2006).

Figure 1 is an example of a Sudoku puzzle of order 9 and Figure 2 represents its solution. The static numbers given in Figure 1 retain their positions and values in the solution. In the solution, each row, column and sub-square of solution contains integers from 1 to 9 exactly once. Also, it has been reported that the total number of unique 9 × 9 Sudoku puzzles that can be generated are 6,670,903,752,021,072,936,960 ($\sim 6.67 \times 10^{21}$) (Felgenhauer et al, 2006) each having a unique solution.

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

Figure 1: A Sudoku puzzle with 26 givens.

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

Figure 2: Solution to the Sudoku puzzle in Figure 1.

Genetic Algorithms (GA) are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome like data structure and apply recombination operators to these structures so as to preserve critical information (Whitley, 1994). Recently Mantere and Koljonen (2006) proposed a method to solve and generate Sudoku puzzles using GA. In this paper we present a new algorithm which we call as *Retrievable Genetic Algorithm (Ret-GA)* and use it for solving Sudoku puzzles. The results are compared with the results given in Mantere and Koljonen (2006).

The rest of the paper is organized as follows. In section 2, a review of literature on solving the Sudoku puzzles using Genetic Algorithms is presented. In section 3, the proposed Retrievable Genetic Algorithm is described. In section 4, the numerical results on instances of various difficulty levels are discussed and analyzed. Finally, the conclusions are drawn in section 5.

2 Literature Review

Sudoku is an *NP*-complete problem (Yato and Seta, 2003).As a result, we cannot hope to find a polynomial time algorithm for all possible problem instances, unless P = NP (Garey and Johnson, 1979). This implies that there will be some instances (possibly many) that cannot be solved without some sort of search also being necessary (Lewis, 2007). Recently various researchers have made good efforts to solve Sudoku puzzles using various approaches including GA. Gold (2005) has developed software which generates full solutions from a blank grid and some of the entries are then removed in order to make a Sudoku puzzle. Moraglio et al. (2006) designed a product Geometric Crossover incorporating the distance of the search space treated as metric space, to solve Sudoku puzzles and concluded that on *Medium* and *Hard* problems, the geometric crossovers perform significantly better than hill-climbers and mutation alone.

A Genetic Algorithm to solve Sudoku puzzles of order 9 has been proposed by Mantere and Koljonen (2006). In their method, they have modified the constrained optimization problem into an unconstrained one. It is claimed that their software can generate the Sudoku puzzles of different difficulty levels and can solve the puzzles up to some extent.
Mantere and Koljonen (2006), in their paper have defined the fitness function as

$$f(x)=10*\left(\sum_i g_{i1}(x)+\sum_j g_{j1}(x)\right)+\left(\sum_i \sqrt{g_{i2}(x)}+\sum_j \sqrt{g_{j2}(x)}\right)+50*\left(\sum_i g_{i3}(x)+\sum_j g_{j3}(x)\right) \quad (1)$$

where, for row-wise operation i,

$$g_{i1}(x)=\left|45-\sum_{j=1}^{9} x_{i,j}\right|,\ g_{i2}(x)=\left|9!-\prod_{j=1}^{9} x_{i,j}\right| \quad \textit{and}$$

$$g_{i3}(x)=\textit{Cardinality}\left(\{1,2,3,4,5,6,7,8,9\}-x_i\right)$$

Similar notations follow for the column-wise operation j.

The first term of Equation (1) requires that each row and column sum should be equal to 45. The second term requires that each row and column product should be 9!. The third term requires that each row and column must contain each integer from 1to 9 exactly once.

The results reported by them clearly indicate the potential of Genetic Algorithms to solve Sudoku puzzles. However, their algorithm works very well for puzzles of lower difficulty level but its efficiency decreases rapidly with increasing difficulty level. Further, it was reported that this fitness function may not be the best for Sudoku puzzles.

In the present work, we have used an entirely new fitness function incorporating puzzle-character-dependent constraints. Further, as evolution often proceeds in *punctuated equilibrium*, the fitness value ceased to improve after certain number of generations. Also, it was observed that all the chromosomes tend to converge to a point very close to the desired solution. It may be possible that after a very large number of generations we may obtain the exact solution but in order to reduce computational time we introduced a random restart mechanism where after a certain number of generations (which depends on the difficulty level of the puzzle) the population is again reinitialized.

3 The Proposed Retrievable Genetic Algorithm

The proposed Retrievable Genetic Algorithm (Ret-GA) starts with the creation of a *Blueprint Matrix* (of same size as the Sudoku puzzle) in which a particular entry is assigned a value 1 if the corresponding entry in the Sudoku puzzle is given, otherwise it is assigned the value 0. The Blueprint matrix thus defined keeps track of the static given entries which are kept unaltered throughout. The algorithm proceeds with the generation of initial population which consists of 10**N* individuals, where *N* is the order of the given Sudoku puzzle. Each individual is an $N \times N$ array. The entries corresponding to the non-givens in the Sudoku puzzle are assigned randomly generated values from 1 to *N*. This step is important because GA yields robust optimal solution due to its random character and approaches the solution by excluding the unfit individuals. Hence, we utilize this random character starting from the initial population itself unlike Mantere and Koljonen (2006), where they generate the initial population intrinsically (no repetition of digits from [1, 9] in each sub-square). As a result our algorithm randomly explores the larger search space and selects the better individual for each consecutive generation.

The initial population thus generated is subjected to Row-wise Uniform Crossover and the crossover probability was selected as 0.8 after extensive experimentation. The crossover operation swaps the whole rows between two consecutive individuals. The possible crossover sites are indicated by dotted lines in Figure 3. It is to be noted that this operation ensures that the givens are not disturbed as they are same in all the individuals.

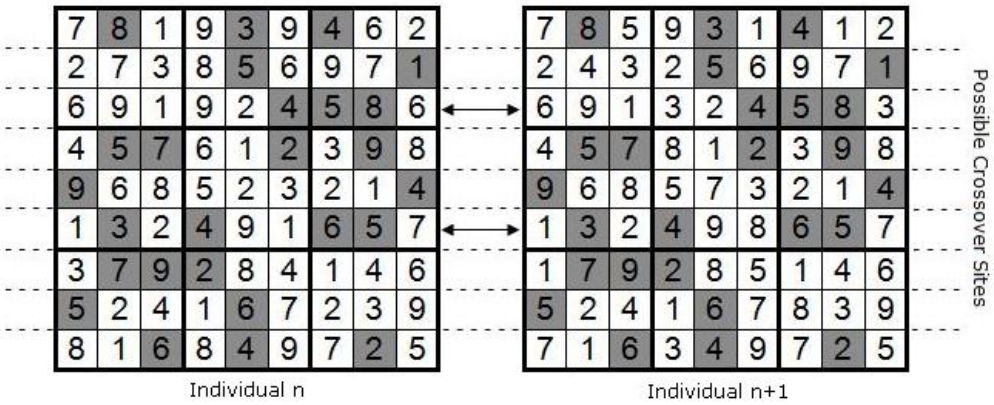


Figure 3: Row-wise Uniform Crossover.

For mutation we follow a different approach where instead of performing a row wise operation, we operate bit wise. Each of the non-given entry is replaced by a randomly generated number (from 1 to *N*) with a probability of 0.2, which has been selected after extensive experimentation (Figure 4).

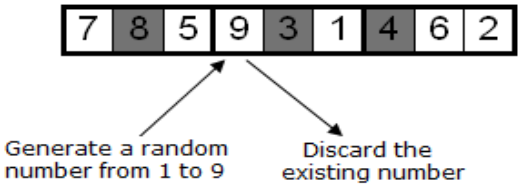


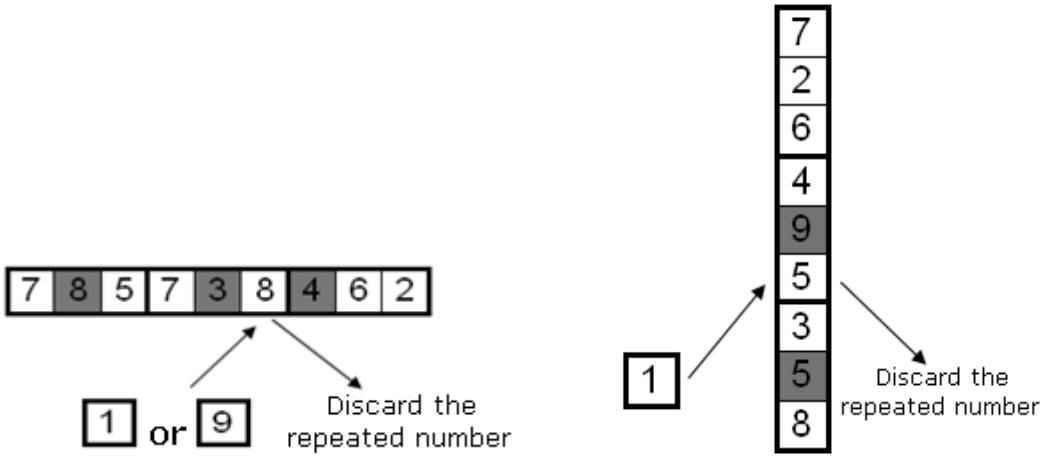
Figure 4: Bit-wise Mutation.

While conducting the experiments, neither complete nor partial elitism was found to be efficient. Hence, we followed an approach in between the two. The population before crossover and after mutation were combined together to form a population of size 20**N* with the individuals arranged in ascending order of their fitness values. To maintain the diversity in the population, the alternate individuals are then selected for the next generation and the cycle continues with the population of the original size 10**N*. We call this step *Alternate Elitism*.

In order to ensure faster convergence, we employ a strategy to utilize the givens of the Sudoku puzzle. We call this step "Remove Repetition". The operation is first performed on each row and then on each column where any repetition of a given is replaced by a randomly selected number from the set ({1,2,3,4,5,6,7,8,9} – G_i), where G_i is the set of numbers present in the i^{th} row or column under consideration (Figure 5). During column wise operation some of the givens may be repeated in a row but such instances were found to be very few, hence the overall fitness of the individuals improves. It should be noted that repetition of non-givens is not removed by this process. This technique is applied after generation of initial population, crossover and mutation.

It is often difficult to design a fitness function in combinatorial problems (Koljonen and Alander, 2004). However, in this paper, an attempt is made to design a simple objective function for a generalized $N \times N$ Sudoku puzzle. We use a simple *uniqueness* technique to design the fitness function. It consists of three different fitness terms, namely row-fitness, column-fitness and sub-square-fitness. These are the only constraints taken with equal penalty. Hence the overall fitness function is defined as:

Fitness function = Row fitness + Column fitness + Sub-square fitness (2)



(a) Row-wise (b) Column-wise

Figure 5: Remove repetition of givens.

Each of the above three functions attains maximum value only when the solution is reached. In the following lines we derive the expression for maximum overall fitness value. Each row entry is compared with all the remaining entries to its right. If the two entries are not equal, row-fitness value is incremented by 1 otherwise it remains same. Thus for the solution the contribution from each row is $N(N-1)/2$ (sum of first $N-1$ natural numbers). Hence for N rows, it will be $N^2(N-1)/2$. Similar results hold for column and sub-square. Hence for a $N \times N$ Sudoku puzzle, the maximum fitness value is $3N^2(N-1)/2$ (which comes out to be 972 for a 9 X 9 Sudoku puzzle). Expressing this concept in a mathematical form, the functions defining the fitness values of any individual can be defined as follows:

$$f(i,j,k,l) = \begin{cases} 0, & \text{if } (i,j) = (k,l) \\ 1, & \text{otherwise} \end{cases} \tag{3}$$

where, (i,j) and (k,l) refer to two entries of $N \times N$ Sudoku puzzle.

The fitness function for row is defined as follows,

$$\text{Row fitness} = \sum_{i=1}^{N} \sum_{j=1}^{N-1} \sum_{l=j+1}^{N} f(i,j,i,l) \tag{4}$$

The fitness function for column is defined as follows,

$$\text{Column fitness} = \sum_{j=1}^{N} \sum_{i=1}^{N-1} \sum_{l=i+1}^{N} f(i,j,l,j) \tag{5}$$

The fitness function for sub-square is defined as follows,

$$\text{Sub-square- fitness} = \sum_{i=1}^{N} \left[\sum_{q=1}^{\sqrt{N}} \left\{ \sum_{j=1+(q-1)\sqrt{N}}^{q\sqrt{N}-1} \sum_{l=j+1}^{q\sqrt{N}} f(i,j,i,l) + \sum_{\substack{r=1+(q-1)\sqrt{N} \\ i \neq t\sqrt{N},\ t \in Z^+}}^{q\sqrt{N}} \sum_{k=i+1}^{i+\sqrt{N}-i(\bmod \sqrt{N})} \sum_{s=1+(q-1)\sqrt{N}}^{q\sqrt{N}} f(i,r,k,s) \right\} \right] \tag{6}$$

where, Z^+ is the set of all positive integers,

Hence by (2) the fitness function is the sum of (4), (5) and (6).

In order to tap the solution as soon as it occurs, we check for the maximum fitness value in initial population and after crossover and mutation. Attaining the maximum fitness value plays an important role as it is being used as the stopping criteria in the algorithm. However, it is observed that as the fitness value of an individual tends to the maximum value, it generally gets trapped in local maxima. This is common with GA which gives near optimal solution. Hence to overcome this shortcoming of GA, we reinitialize the population after a *reset point* in the same run and thus try to approach the solution through a different path. The reset point is defined as the number of generations after which the algorithm attempts to regenerate the initial population if the solution is not found. We set different reset points for different difficulty level Sudoku puzzles. Higher the difficulty level (or, fewer is the number of givens, in general), greater is the reset point. After an extensive experiment, we define reset points as 2000 if there are 27 or less givens, 350 if there are 28 or 29 givens, 300 if there are 30 or 31 givens and 200 if there are 32 or more givens. Thus our algorithm tries to retrieve the solution if it gets stuck in local maxima, overcoming the problem faced by GA. Hence we call it as a **Retrievable Genetic Algorithm** *(Ret-GA)*. The step-by-step procedure followed by Ret-GA is depicted in Figure 6.

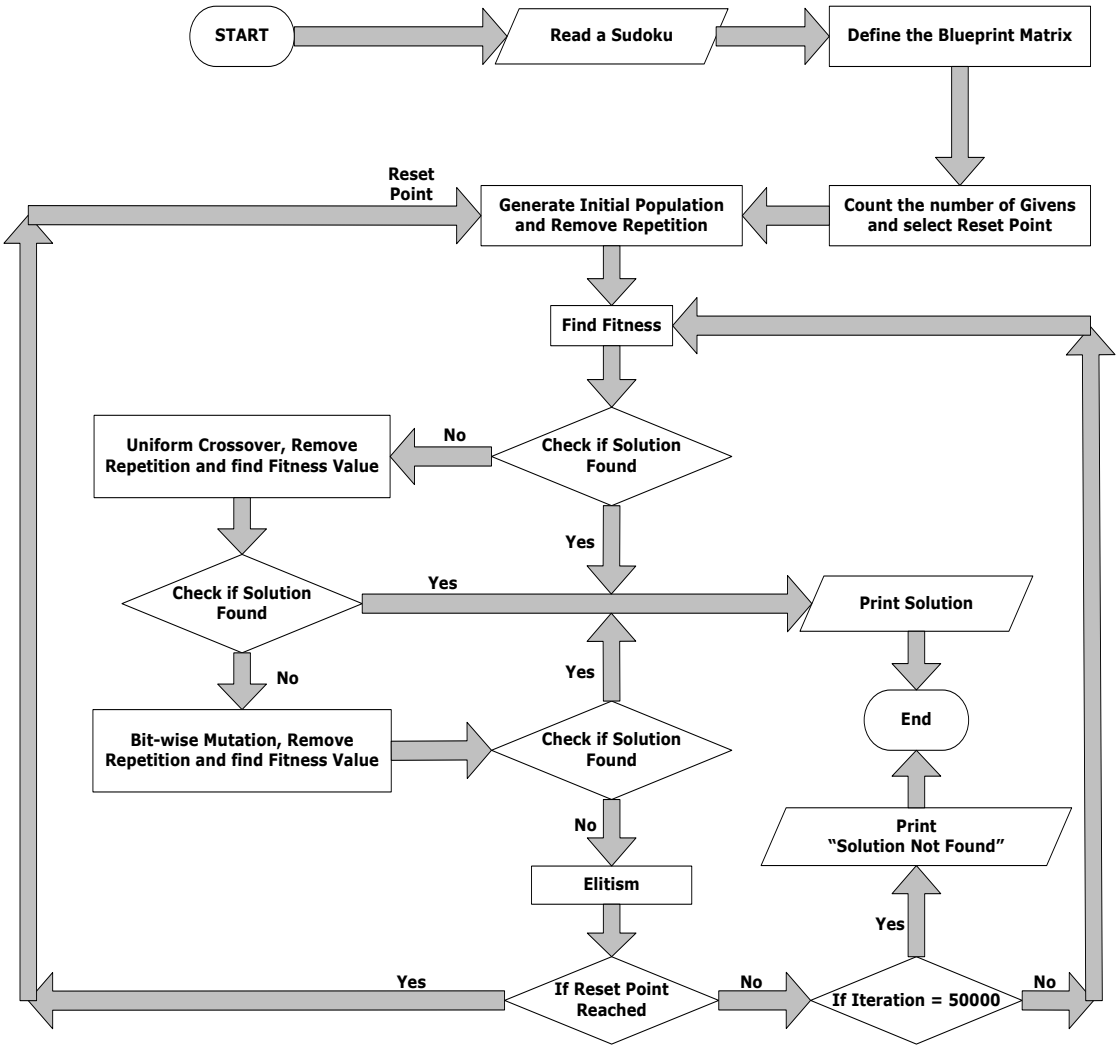


Figure 6: Schematic representation of the proposed Ret-GA.

4 Computational Experiment

4.1 Experimental setup

In order to check the effectiveness of our algorithm, we have considered Sudoku puzzles of order 9 (N = 9), which are most common. The population size is kept fixed at 90 (=10**N*). Uniform crossover and bitwise mutation have been incorporated at a probability of 0.8 and 0.2 respectively. We consider the problem set of 84 Sudoku puzzles. The first nine of them are picked from Mantere and Koljonen (2006) except their first (New) problem, where there are no givens. Out of them, they have taken five Sudoku puzzles from the newspaper Helsingin Sanomat (2006), marked with difficulty rating *1-5 star*, where there are symmetric givens. Rest four problems are from newspaper Aamulehti (2006), marked with difficulty rating: *Easy, Challenging, Difficult and Super difficult*. Apart form these, 75 new puzzles have been taken from www.sudoku.com (2007) marked with difficulty rating: *Easy, Medium and Difficult* (25 of each type). We tested each problem 100 times. The stopping criterion for a run is either the optimum value (972) reached or the maximum number of generations (fixed to 50,000) attained.

4.2 Results and Discussions

The results of Ret-GA are compared with the results obtained by Mantere and Koljonen (2006) in Table 1. The performance of Ret-GA on other 75 puzzles is given in Table 2 (*Easy*), Table 3 (*Medium*) and Table 4 (*Hard*).

4.2.1 Comparison between Ret-GA and GA

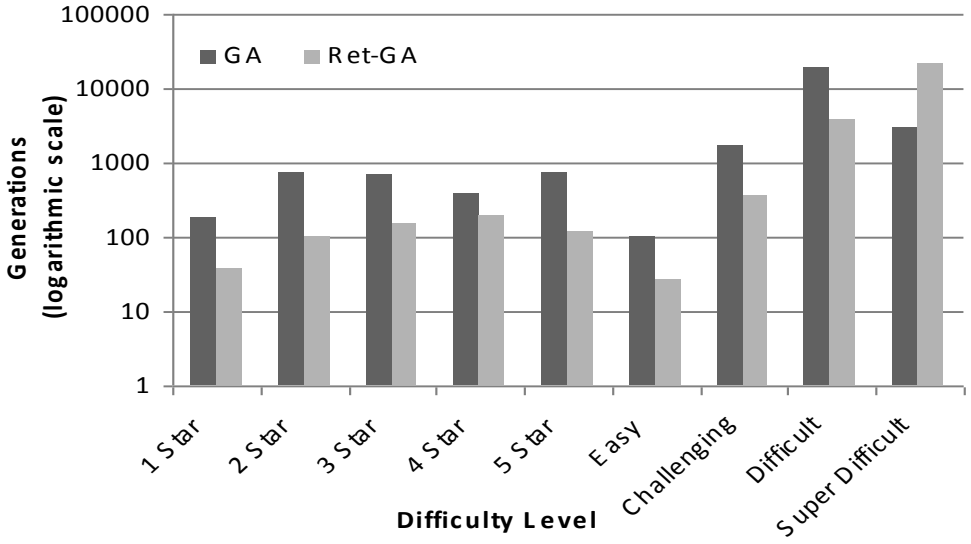
(a) First Method of Analysis

The first goal of the analysis is to observe that whether Ret-GA shows an improvement over GA. The performance of the proposed Ret-GA is realized by comparing with an existing GA in literature, by solving 9 problem instances of varying difficulty levels. The criterion for comparison measures the *reliability*, *efficiency* and *stability* of both the approaches (GA and Ret-GA). In solving the sample puzzles, it is observed that the success rate obtained by Ret-GA is better than that of GA and the standard deviation for Ret-GA is lesser than that of GA in each problem. Hence we conclude that Ret-GA is more reliable and more stable than GA in solving Sudoku puzzles.

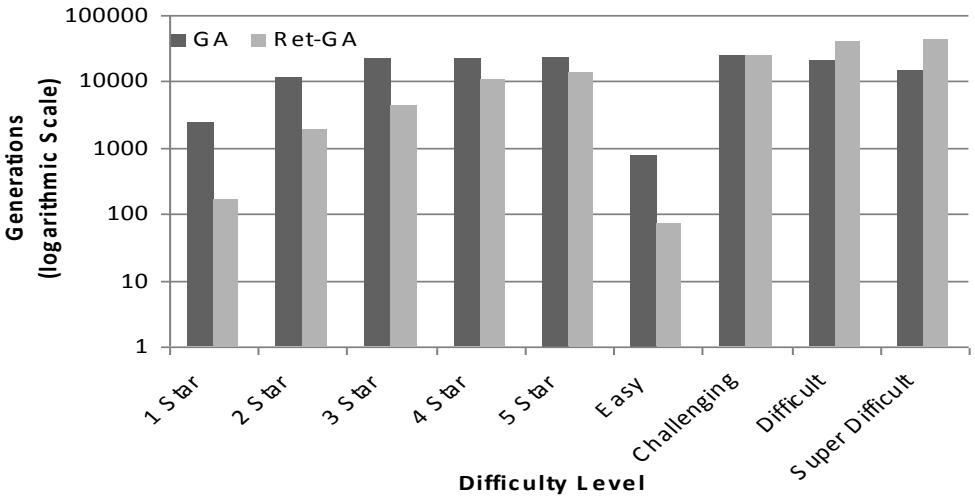
Further it is worthy to note that Ret-GA could solve *1-5 star* and *Easy* rating puzzles with 100% success. It requires lesser number of generations as compared to existing GA. Hence, it can be concluded that Ret-GA is not only more reliable and stable but also more efficient than GA in solving *1-5 star* and *Easy* problems. Though, Ret-GA is not able to solve *Challenging*, *Difficult* and *Super Difficult* problems with 100% success, but the success rate of Ret-GA is always higher as compared to GA. However, the average number of generations is little higher in Ret-GA. Hence, it can be concluded that although Ret-GA is more reliable and stable but is less efficient than GA in solving *Challenging, Difficult* and *Super Difficult* problems.

Level	Givens	Success Rate		Minimum		Maximum		Average		Median		Standard Deviation	
		GA	Ret-GA	GA	Ret-GA	GA	Ret-GA	GA	Ret-GA	GA	Ret-GA	GA	Ret-GA
1 Star	33	100	**100**	184	**39**	23993	**1100**	2466.60	**159.24**	917	**98**	3500.98	**159.95**
2 Star	30	69	**100**	733	**99**	56484	**7459**	11226.80	**1864.94**	7034	**1441**	11834.68	**1728.25**
3 Star	28	46	**100**	678	**152**	94792	**21595**	22346.40	**4338.30**	14827	**2755**	24846.46	**3906.40**
4 Star	28	26	**100**	381	**198**	68253	**42382**	22611.30	**10716.73**	22297	**6966**	22429.12	**11285.80**
5 Star	30	23	**100**	756	**117**	68991	**48336**	23288.00	**13569.76**	17365	**9393**	22732.25	**12832.50**
Easy	36	100	**100**	101	**27**	6035	**305**	768.60	**70.34**	417	**52**	942.23	**55.70**
Challenging	25	30	**94**	1771	**357**	89070	**41769**	**25333.30**	25932.87	**17755**	27786	23058.94	**19153.54**
Difficult	23	4	**16**	18999	**3699**	46814	**45676**	**20534.30**	40466.13	**26162**	33760	12506.72	**11940.66**
Super Difficult	22	6	**9**	**3022**	21424	**47352**	49832	**14392.00**	42354.86	**6722**	36318	17053.33	**11198.42**

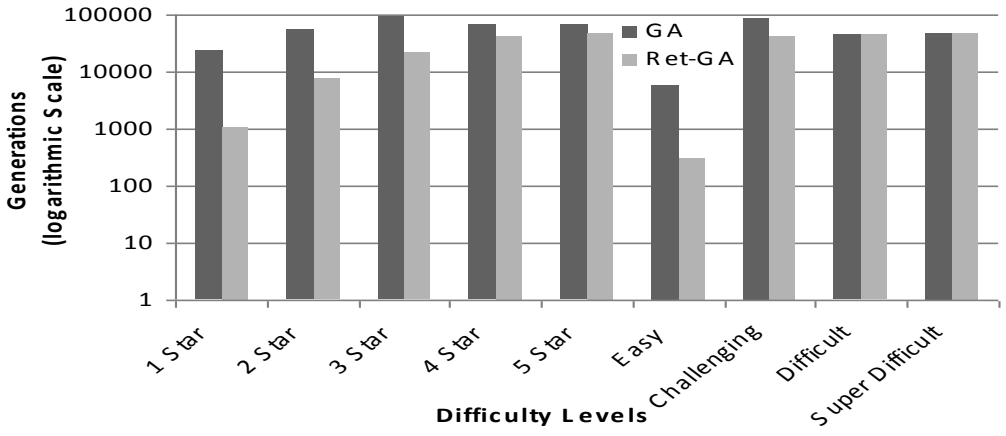
Table 1: Performance Comparison of GA and Ret-GA



(a) Minimum number of generations required.



(b) Average number of generations required.



(c) Maximum number of generations required.

Figure 7: Comparison of GA and Ret-GA.

(b) Second Method of Analysis

In order to compare the performance of both the algorithms GA and Ret-GA, we consider three major factors namely

(i) Success rate to observe the **reliability**
(ii) Average number of generations to observe the **efficiency**
(iii) Standard deviations to evaluate the **stability** of the algorithm.

We have used the performance index (PI) (Bharti, 1994; Mohan and Nguyen, 1999; Deep and Thakur, 2007) in the extended form as below.

$$PI = \frac{1}{N_p}\sum_{i=1}^{N_p}\left(k_1\alpha_1^i + k_2\alpha_2^i + k_3\alpha_3^i\right) \tag{7}$$

where, $\alpha_1^i = \frac{Sr^i}{Tr^i}$,

$$\alpha_2^i = \begin{cases} \frac{Mg^i}{Ag^i}, & if\ Sr^i > 0 \\ 0, & if\ Sr^i = 0 \end{cases}$$

$$\alpha_3^i = \begin{cases} \frac{Msd^i}{Asd^i}, & if\ Sr^i > 0 \\ 0, & if\ Sr^i = 0 \end{cases}$$

.

$i = 1,2,...,N_p$

Sr^i = Number of successful runs of i^{th} puzzle

Tr^i = Total number of runs of i^{th} puzzle

Ag^i = Average number of generations of successful runs used by an algorithm in obtaining the solution of i^{th} puzzle

Mg^i = Minimum of average number of generations of successful runs used by both algorithms in obtaining the solution of i^{th} puzzle

Asd^i = Average of standard deviation of successful runs used by an algorithm in obtaining the solution of i^{th} puzzle

Msd^i = Minimum of standard deviation of successful runs used by both algorithms in obtaining the solution of i^{th} puzzle

N_p = Total number of puzzles analyzed.

k_1, k_2 and k_3 ($k_1 + k_2 + k_3 = 1$ and $0 \le k_1, k_2, k_3 \le 1$) are the weights assigned to percentage of success, mean fitness function value and average number of generations of successful runs, respectively.

From the above definition it is clear that PI is a function of k_1, k_2 and k_3. Since $k_1 + k_2 + k_3 = 1$, one of k_i, $i = 1,2,3$ could be eliminated to reduce the number of dependent variables from the expression of PI. But it is still difficult to analyze the behavior of PI, because the surface plots of PI for all three algorithms are overlapping and it is difficult to visualize them. Equal weights are assigned to two terms at a time in the PI expression. This way PI becomes a function of one variable. The resultant cases are as follows

(i) $k_1 = w,\ k_2 = k_3 = \frac{1-w}{2},\ 0 \le w \le 1$

(ii) $k_2 = w,\ k_1 = k_3 = \frac{1-w}{2},\ 0 \le w \le 1$

(iii) $k_3 = w,\ k_1 = k_2 = \frac{1-w}{2},\ 0 \le w \le 1$

We assign three values 0, 0.5 and 1 to w. By varying k_1, k_2 and k_3 we plot Figures 8 (a),(b) and (c).

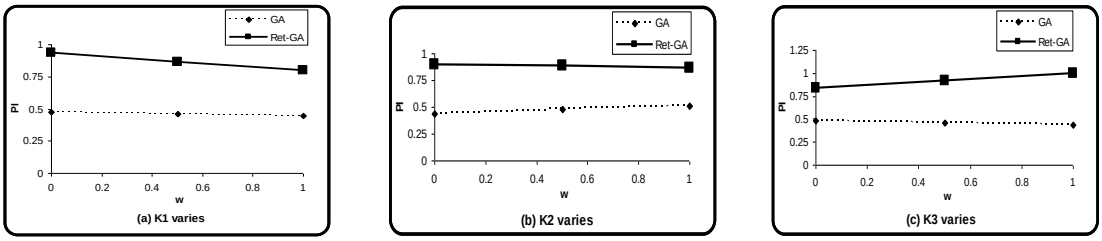


Figure 8: PI graph to compare the performance between GA and Ret-GA.

Now from Figure 8, we obtain a clear cut result indicating that Ret-GA is more reliable, more efficient and more stable to solve a Sudoku puzzle in general. Thus Ret-GA outperforms GA in totality.

4.2.2 Performance of Ret-GA

In this section we consider the performance of Ret-GA on a set of 75 new puzzles of difficulty rating *Easy*, *Medium* and *Hard*, 25 of each type. We wish to find out if the puzzles that are difficult for human beings, are also difficult for Ret-GA. The minimum, average and maximum number of generations (of successful runs only) required to solve each puzzle along with the success rate and standard deviation are tabulated in Table 2, 3 and 4 for *Easy*, *Medium* and *Hard* puzzles respectively. The results thus obtained conclude that the success rate decreases and the average number of generations and the standard deviation increases while proceeding form *Easy* to *Hard*. Considering the success rate and the average number of generations required by Ret-GA, we can order the difficulty level of the Sudoku Puzzles form *Easy* to *Hard* as:

Easy < *Medium* < *Hard.*

Thus, we conclude that the problem which is hard for human beings is also hard for Ret-GA to solve.

5 Conclusion

This paper introduces a specialized heuristics, called *Retrievable Genetic Algorithm* (Ret-GA) for solving the popular Japanese Sudoku puzzle. It is named as *Retrievable* since it involves a simple mechanism for escaping the local maxima by reinitializing the population after a specified number of generations. Hence, it overcomes the inherent shortcoming of GA which generally yields near optimal solution, which is irrelevant for Sudoku puzzles having a unique solution. Other significant contribution of this research includes the development of a problem dependent fitness function following a *uniqueness* approach. Crossover, Mutation and Elitism have been designed so that the initially given constant entries are not modified during the simulation. The results thus obtained are found to be encouraging and comparison with the existing GA in literature bolsters the effectiveness of Ret-GA in solving Sudoku puzzles. Though Ret-GA takes a higher number of generations for problems of higher difficulty rating (*Challenging*, *Difficult* and *Super Difficult*) but it is undoubtedly more stable and more reliable than GA in solving Sudoku puzzles. Moreover in problems other than stated above, efficiency of Ret-GA far exceeds than that of GA. Experiments carried out for another set of 75 puzzles show that the number of generations required increases and success rate decreases progressively with increasing difficulty level of the puzzles. Hence it is concluded that the problems which are hard for human beings are also hard for Ret-GA to solve.

Ret-GA is a unique and novel approach to solve Sudoku puzzles. The approach may be extended to larger order Sudoku puzzles and our next paper will report the use of Ret-GA for higher order Sudoku puzzles. Researchers are encouraged to develop new fitness functions

which are more efficient in finding the solution in lesser number of generations with higher success rate for generalized Sudoku puzzles of any order. It is suggested that Ret-GA and Alternate Elitism operator be tested on a wide variety of problems before anything can be said certainly about their applicability in general.

Problem Number	Givens	Success Rate	Minimum	Maximum	Average	Median	Standard Deviation
1	29	100	79	3249	681.78	483	662.21
2	29	100	161	10401	2271.35	1612	2173.31
3	29	100	87	6165	1782.56	1158	1856.46
4	29	100	93	8578	1548.63	1091	1349.43
5	29	100	132	13900	2711.56	2142	2051.33
6	29	100	73	12203	1954.28	1134	1927.07
7	29	100	178	15190	3718.24	2187	3457.85
8	29	100	134	10440	3145.89	2502	2906.97
9	29	100	98	12982	2524.15	1570	2711.83
10	29	100	69	8036	2067.43	1467	1973.55
11	29	100	107	10278	3154.37	2409	3305.29
12	29	100	235	12595	4764.22	2981	3905.17
13	29	100	52	14300	2137.07	1280	2155.35
14	29	100	88	11890	1700.65	1321	1594.82
15	29	100	67	4511	785.11	495	863.05
16	29	100	153	12524	2175.66	1507	1976.34
17	29	100	182	14480	3551.25	2730	3060.28
18	29	100	66	14236	980.45	579	1110.78
19	29	100	127	7584	1545.62	1012	1694.58
20	29	100	76	8931	971.23	780	840.26
21	29	100	87	2957	723.26	463	803.19
22	29	100	113	6830	1044.20	770	849.18
23	29	100	162	12874	1497.33	912	1402.03
24	29	100	44	2326	437.68	321	526.07
25	29	100	89	4021	845.21	702	643.80

Table 2: Performance of Ret-GA for Easy Level Sudoku Puzzles

Problem Number	Givens	Success Rate	Minimum	Maximum	Average	Median	Standard Deviation
1	28	86	271	43918	9810.11	7441	10502.88
2	28	91	362	37491	11180.40	10463	8639.64
3	28	94	307	39048	10254.45	6967	8306.10
4	28	82	420	26253	8130.56	6017	7724.13
5	28	85	391	37782	11705.84	9880	13695.83
6	28	79	540	48711	16750.54	13402	14740.08
7	28	97	186	27805	7324.82	5493	6884.56
8	28	91	291	38042	13330.89	11563	11464.52
9	28	95	368	43714	17836.86	16410	16053.17
10	28	89	840	40549	12015.64	9252	9612.51
11	28	83	447	36265	14645.74	9667	13181.16
12	28	91	260	41795	10897.18	7628	9262.30
13	28	84	352	42609	11589.32	9272	8692.69
14	28	67	1204	49382	21158.73	19043	17984.92
15	28	82	651	38782	9465.72	8045	8822.83
16	28	95	384	44076	13264.57	11142	10963.26
17	28	73	443	48118	18641.65	13608	14883.58
18	28	88	311	37605	14508.32	13058	13246.09
19	28	94	107	29833	12879.79	9659	10690.22
20	28	78	810	46519	16397.21	13117	15464.86
21	28	82	1756	36144	11626.07	9998	13553.67
22	28	90	626	41802	14687.72	10721	13853.60
23	28	98	213	31840	8875.90	7887	7032.84
24	28	89	433	42239	16689.53	14186	15078.16
25	28	91	252	39817	13756.89	10228	11330.24

Table 3: Performance of Ret-GA for Medium Level Sudoku Puzzles

Problem Number	Givens	Success Rate	Minimum	Maximum	Average	Median	Standard Deviation
1	25	32	6547	48758	28274.34	28743	26643.57
2	25	27	8722	48273	37662.83	31357	25955.13
3	25	37	7501	46014	29266.05	26732	28392.72
4	25	25	14311	45994	32038.41	31431	26288.36
5	25	30	8068	48019	28972.87	26042	21338.14
6	25	34	9859	46361	39825.01	31477	22071.71
7	25	23	10913	45994	33742.76	28906	26072.26
8	25	29	12621	45076	31352.97	31502	26299.84
9	25	18	9564	48734	32733.99	26239	23705.45
10	25	36	5185	47225	30009.78	26481	25751.18
11	25	24	7214	49659	31494.43	30251	24514.53
12	25	33	6447	47330	28389.49	28941	20439.15
13	25	31	7154	47093	33697.21	32517	20272.67
14	25	28	10919	49231	36405.64	33479	23127.69
15	25	40	6218	47626	32947.32	32561	19129.71
16	25	31	8382	46013	34608.96	29245	23840.47
17	25	27	9762	48361	28136.72	27060	16831.55
18	25	24	6057	49191	30697.41	29728	20928.21
19	25	39	5355	45098	36750.74	34437	20353.50
20	25	12	14169	48406	35213.06	33124	26124.15
21	25	26	9102	46897	31916.43	29409	26085.82
22	25	32	9936	49159	33517.93	32337	20158.71
23	25	37	5578	47514	36913.68	31989	20164.45
24	25	39	6528	48547	25888.21	24221	21901.66
25	25	23	11132	47144	34043.78	31418	25869.23

Table 4: Performance of Ret-GA for Hard Level Sudoku Puzzles

Acknowledgements

The authors are thankful to the anonymous reviewers for their valuable comments and suggestions.

References

Aamulehti. Sudoku online via WWW: http://www.aamulehti.fi/sudoku/ (cited 11.01.2006)

Bharti (1994), Controlled random search technique and their applications. *Ph.D. Thesis, Department of Mathematics, University of Roorkee*, Roorkee, India, 1994.

Deep, K. and Thakur, M. (2007a). A new crossover operator for real coded genetic algorithms. *Applied Mathematics and Computation, 188(1), pp:895-911*.

Deep, K. and Thakur, M. (2007b). A new mutation operator for real coded genetic algorithms, *Applied Mathematics and Computation*, doi:10.1016/j.amc.2007.03.046.

Garey, M. R. and D. S. Johnson. (1979). *Computers and Intractability*. W. H. Freeman, San Francisco.

Gold, M (2006). Using Genetic Algorithms to Come up with Sudoku Puzzles. Available via WWW:http://www.c-sharpcorner.com/UploadFile/mgold/Sudoku0923005003323AM/Sudoku.aspx?ArticleID-fba36449-ccf3-444f-a435-a812535c45e5 (cited 11.09.2006).

Helsingin Sanomet (2006). Sudoku available via WWW: http://www2.hs.fi/extrat/Sudoku/Sudoku.html (cited 11.01.2006)

Felgenhauer, B. and Jarvis, A. F. (2006). Mathematics of Sudoku I. *Mathematical Spectrum* 39 (2006), pp:15–22

Koljonen, J., and Alander, J. T. (2004): Solving the "urban horse" problem by backtracking and genetic algorithm a comparison. in *Proceedings of the 11th Finnish Artificial Intelligence Conference* (STeP 2004), Jarmo T. Alander, Pekka Ala-Siuru and Heikki Hyötyniemi (eds.), Vantaa (Finland), 1–3 Sep. 2004, Vol. 3, pp: 127–136.

Lewis, R. (2007). Metaheuristics can Solve Sudoku Puzzles. *Journal of Heuristics*, Springer, 13 (4), pp: 387-401.

Mantere, T. *and* Koljonen, J. (2006). Solving and Rating Sudoku Puzzles with Genetic Algorithms. *New Development of Artificial Intelligence and the Semantic Web, Proceeding of the 12th Finnish Artificial Intelligence Conferences, (STeP 2006),* October 26-27, pp: 86-92.

Mohan, C. and Nguyen, H. T. (1999). A controlled random search technique incorporating the simulating annealing concept for solving integer and mixed integer global optimization problems. *Computational Optimization and Applications*. Vol.14, pp: 103-132.

Moraglio, A., Toqelius, J. and Lucas, S. (2006) Product Geometric Crossover for the Sudoku Puzzle, *Evolutionary Computation, ECE 2006, IEEE congress*, pp: 470-476.

Semeniuk, I. (2005). Stuck on you. In *New Scientist 24,* pp: 45-47.

Sudoku generating software. (2007). Available via WWW: http://www.sudoku.com

Whitley, D. (1994) A Genetic Algorithm Tutorial. *Journal of Statistics and Computing*. Vol 4, pp: 65-85.

Yato, T. and Seta, T. (2003). Complexity and Completeness of Finding Another Solution and Its Application to Puzzles. *IEICE Trans. Fundamentals*, Vol. E86-A, No. 5, pp. 1052-1060.