Solving, rating and generating Sudoku puzzles with GA

Conference Paper · October 2007

DOI: 10.1109/CEC.2007.4424632 · Source: IEEE Xplore

2 authors:

Timo Mantere
University of Vaasa
57 PUBLICATIONS **506** CITATIONS

SEE PROFILE

Janne Koljonen
University of Vaasa
21 PUBLICATIONS **239** CITATIONS

SEE PROFILE

Solving, Rating and Generating Sudoku Puzzles with GA

Timo Mantere and Janne Koljonen

***Abstract*—This paper studies the problems involved in solving, rating and generating Sudoku puzzles with genetic algorithms (GA). Sudoku is a number puzzle that has recently become a worldwide phenomenon. Sudoku can be regarded as a constraint satisfaction problem. When solved with genetic algorithms it can be handled as a multi-objective optimization problem. The three objectives of this study was: 1) to test if genetic algorithm optimization is an efficient method for solving Sudoku puzzles, 2) can GA be used to generate new puzzles efficiently, and 3) can GA be used as a rating machine that evaluates the difficulty of a given Sudoku puzzle. The last of these objectives is approached by testing if puzzles that are considered difficult for a human solver are also difficult for the genetic algorithm. The results presented in this paper seem to support the conclusion that these objectives are reasonably well met with genetic algorithm optimization.**

I. INTRODUCTION

This paper studies if the Sudoku puzzles can be solved, rated and generated effectively with genetic algorithms. Genetic algorithm (GA) is an optimization method that mimes the Darwinian evolution [1] that occurs in nature.

According to Wikipedia [2] Sudoku is a Japanese logical game that has recently become hugely popular in Europe and North-America. However, the first puzzle was published in a puzzle magazine in USA 1979, then it circled through Japan, where it became popular in 1986, and later it become a phenomena in the western world circa 2005 [3]. Sudoku has been claimed to be very popular and addictive because it is very challenging but has very simple rules [4].

Sudoku puzzle is composed of a 9×9 grid, total 81 positions, that are divided into nine 3×3 subgrids. The solution of Sudoku puzzle is such that each row, column and subgrid contains each integer {1, 2, … , 9} once and only once.

The puzzle is presented so that in the beginning there are some static numbers, givens, in the grid that are given in advance and cannot be changed or moved. The number of givens does not determine the difficulty of the puzzle [4], [5]. Grating puzzles is one of the most difficult things in Sudoku puzzle creation, and there are about 15 to 20 factors that have an effect on difficulty rating [2].

The givens can be symmetric or nonsymmetrical. In the symmetric case, there are pairs of givens located symmetrically with respect to centre position.

Figure 1 shows one example of Sudoku puzzle generated by our GA. It contains 38 given numbers, and the correct number for the other 43 positions should be solved. This puzzle has nonsymmetrical givens, since the givens are not symmetrically located respect to the central point. The solution of this Sudoku is shown in fig. 2. The static numbers given in the beginning (fig. 1) have remained exactly in the same positions where they originally were.

In this study, we try to evaluate, how effectively genetic algorithms solve Sudoku puzzles presented in newspapers and in www.sudoku.com [6]. Furthermore, we evaluate if the GA efficiency correlates with the alleged difficulty ratings of these Sudoku puzzles. The third objective is to utilize the information obtained from the first two objectives in generating new puzzles with GA and to evaluate the difficulty of the new puzzle.

In the Section I we introduce the problem, genetic algorithms and related work, Section II introduces the proposed method, Section III the obtained results, and section IV discusses on the findings and their implications.

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

Fig. 1. A starting point of the Sudoku puzzle, where 38 locations contains a static number that are given

A. Genetic Algorithms

All Genetic algorithms [7] are computer based optimization methods that use the Darwinian evolution [1] of nature as a model and inspiration. The solution base of a problem is encoded as individuals that are chromosomes consisting of several genes. On the contrary to nature, GAs the individual (phenotype) of GA is usually deterministically derived from the chromosome (genotype). The age or environment does not alter the phenotype during the GA

Timo Mantere is with the Department of Electrical Engineering and Automation, University of Vaasa, FIN-65101 Vaasa, Finland (corresponding author to provide phone:+358 6 324 8679; fax: +358 6 324 8467; e-mail: timan@uwasa.fi).

Janne Koljonen is with the Department of Electrical Engineering and Automation, University of Vaasa, FIN-65101 Vaasa, Finland (e-mail: jako@uwasa.fi).

1-4244-1340-0/07/$25.00 ©2007 IEEE

individual "life time". These virtual individuals are tested against a problem represented as a fitness function. The better the fitness value individual gets, the better is its chance to be selected as a parent for new individuals. The worst individuals are killed from the population in order to make room for the new generation. Using crossover and mutation operations GA creates new individuals. In crossover, we select the genes for a new chromosome from the parents using some preselected practice, *e.g.* one-point, two-point or uniform crossover. In mutation, we change random genes of the chromosome either randomly or using some predefined strategy. The GA strategy is often elitist and therefore follows the "survival of the fittest" principles of the Darwinian evolution.

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

Fig. 2. A solution for the Sudoku puzzle given in fig 1. The given numbers marked in **bold**.

B. Related Work

The Sudoku problem is widely studied in constraint programming and satisfiability research [8], [9], and [10]. Those methods are also efficient to solve Sudokus, but do not provide solution for every Sudoku puzzle. However, in this study we concentrate on Sudoku solving with evolutionary methods. The main reason for solving Sudokus with GA is to learn more about capabilities of GA in constrained combinatorial problems, and hopefully to learn new tricks to make it more efficient also in this field of problems.

There seems to be a few scientific papers about Sudoku with EA methods. Moraglio *et al.* [5] have solved Sudokus using GA with product geometric crossover. They claim that their geometric crossover perform significantly better than hill-climbers and mutations alone. Their method solves `easy` Sudokus from [6] efficiently, but has difficulties with the `medium` and `hard` Sudokus. They also acknowledged that evolutionary algorithms are not the most efficient technique for solving Sudokus, but that Sudoku is an interesting study case for algorithm development.

Nicolau and Ryan [11] has used quite a different approach to solve Sudokus: their GAuGE (Genetic Algorithms using Grammatical Evolution) optimizes the sequence of logical operations that are then applied to find the solution.

Gold [12] has used GA for generating new Sudoku puzzles, but the method seems to be inefficient, since in their example their GA needed 35700 generations to come up with a new open Sudoku solution. In our results we create a new open Sudoku solution, in average, with 101 generations (2020 trials).

There is also a Sudoku Maker [13] software available that is said to use genetic algorithm internally and claimed that the generated Sudokus are usually very hard to solve. Unfortunately, there are no details how GA is used and how quickly a new Sudoku is generated.

The rules of Sudoku imply that the sum of the numbers in each column and row of the solution are equal. Therefore, Sudoku obviously is related to the ancient magic square problem (Latin square), where the given size of square must be filled so, that the sum of each column and each row are equal. The magic square problem has previously been solved with GA [14], [15] and also the Evonet Flying circus [16] has a demonstration of a magic square solver.

Sudoku can also be seen as constraint satisfaction problem, where all the row and column sums must be equal to 45. Constrained optimization problems have been efficiently optimized with evolutionary algorithms *e.g.* in [17], [18], [19].

Another related problem is the generation of threshold matrices for halftoning grayscale images [20]. In the halftoning, the gray values of an image are converted into two values, black or white, by comparing the value of each pixel to the value of the corresponding position in the threshold matrix. The values in the threshold matrix should be evenly distributed; the threshold value sums in each row and column of the threshold matrix should be nearly equal. The most well known threshold matrix by Bayer [21] has equal threshold sums in one dimension (either rows or columns), while in the other dimension (columns or rows respectively) they differ in a regular fashion. Furthermore, the demand of homogeneity of the threshold values also holds locally, *i.e.* any fixed size subareas should have nearly evenly distributed threshold value sums. This guarantees that the result image does not contain large clusters of black or white pixels. Threshold matrices have been previously optimized by GA *e.g.* in [15], [22], [23], [24], and [25].

II. The Proposed Method

In order to test the proposed method we decided to use an integer coded elitist GA. The size of the GA chromosome is 81 integer numbers, divided into nine sub-blocks of nine numbers. The uniform crossover operation were applied only between between sub-blocks, and the sequences of swap mutations only inside the sub-blocks.

The swap mutation probability was 0.6 (the number of tried mutations, which does not equal the actual mutations made in this case, see mutations details to follow).

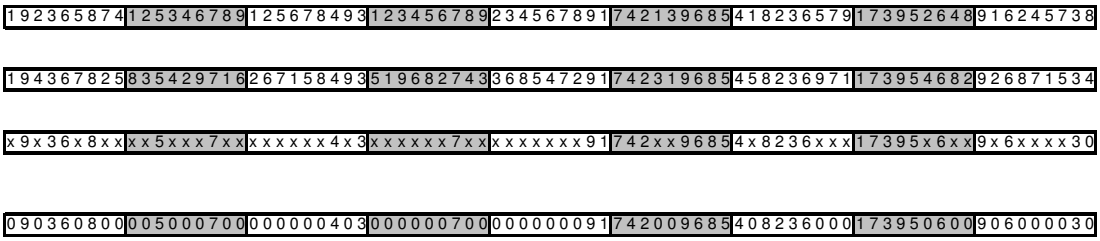

Fig. 3. The representation of Sudoku puzzles in our GA. One possible solution (individual) is an array of 81 numbers that is divided into nine sub blocks of nine numbers. The crossovers points can only appear between sub blocks (marked as vertical lines). The help array is used for checking fixed positions, if there is a number that is not equal to zero that number cannot be changed during the optimization, only the positions that have zero are free to change.

The population size (POP) was 21, and elitism ratio (ELIT) was 1. The best individuals were favored by selecting the mating individuals x1 and x2 with the following algorithm:

```
for( i=POP-1; i>=ELIT; i--){
  x1 = ( int )(i * Math.random() );
  x2 = ( int )(i * Math.random() );
  . . .
  }
```

All new individuals were generated by first applying crossover and then mutations to the crossover result. However, there is a chance of selecting x1=x2, where only the mutation operation changes the new individual genotype. A found solution was the only stop condition, since our method never failed to find a solution.

In addition to basic rules of Sudoku, the fixed numbers ("givens") must be observed during the solving process. Therefore a Sudoku solution obeys four conditions:

1) Each row has to contain each integer from 1 to 9,
2) each column has to contain each integer from 1 to 9,
3) each 3x3 subgrid must contain each integer from 1 to 9,
4) the given numbers must stay in the original positions.

Condition 4) can be omitted a new open Sudoku solution that is used for creating a new Sudoku puzzle is looked for. When a given Sudoku puzzle is solved condition 4) has to always fulfilled. By an appropriate solving approach one of conditions 1) to 3) can be controlled, hence only three conditions are subject to optimization.

We chose to program the GA so that conditions 3) and 4) are automatically fulfilled and only the conditions 1) and 2) are optimized. Sudoku has 9 rows and 9 columns, so we have 18 equality constrains that must be fulfilled when we have the solution.

The GA used in this study is customized to this problem or other grid type combinatorial problems. This GA was a modification of a combinatorial GA originally programmed for magic square solver [15] and other combinatorial problems.

This GA does not use direct mutations or crossovers that could generate illegal situations for 3x3 subgrids. On the contrary rows and columns can contain integers more than once in non-optimal situation. The genetic operators are not allowed to move the fixed numbers that are given in the beginning of the problem. This is guaranteed by an help array, which indicates whether a number is fixed or not.

We decided to represent a Sudoku puzzle solution as an integer array of 81 numbers. The array is divided into nine sub-blocks (building blocks) of nine numbers corresponding to the 3×3 subgrids from left to right and from top to bottom. Crossovers operated only these whole sub-blocks between individuals. Therefore the crossover point cannot be inside a building block (fig. 3).

A. *The mutation operator*

Mutations are applied only inside a sub-block. Originally, we used three different mutation strategies that are commonly used in combinatorial optimization: swap mutation, 3-swap mutation, and insertion mutation. However, since the 3-swap and insertion mutations in practice are actually only sequences of swap (two-swap) mutations, we later replaced them with a sequence of 1 to 5 swap mutations inside a sub-block. This was done, because our test runs showed that the GA with only swap mutations performed better than GA with swap, 3-swap and insertion mutations.

In swap mutation, the values of two positions are exchanged. Each time mutation is tried inside a sub-block, the help array of givens is checked. If it is illegal to change the randomly chosen position, mutation is omitted. Additionally, if we have randomly drawn a sequence of *e.g.* 5 mutations we will perform the mutations only so far as both positions of swap mutations are legal. Hence, despite the equal chance to draw a 1-, 2-, 3-, 4-, or 5-swap sequence, most of the time the sequence is interrupted. Sometimes the very first swap attempt is illegal and no mutation is performed, this means that the actual mutation percentage is much lower than the 0.6 mentioned earlier.

The test runs with using different number of sequential swaps also showed that the GA with just one swap mutation performed almost as well as GA with 1 to 5 swap mutation sequences, so the fact that most of the longer swap sequences are aborted prematurely is not harmful.

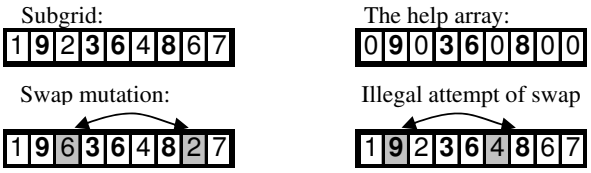

Fig. 4. The swap mutation used in Sudoku optimization. Up left is one sub block and up right the static values of that sub block (6 and 7 are givens). The mutation is applied so that we randomly select positions inside the sub block, and then compare the selected positions to the help array in order to see if the positions are free to change.

There was one more special rule controlling whether the mutation was performed or abandoned. In this rule we have another help table that tells how many times each digit appears in each row and column. When a mutation attempt is tried, it affects one or two columns, and one or two rows, totally 3 or 4 row and column vectors. In the optimal situation, the two digits that are swapped should appear in these vectors only twice in total. We give system some slack and do not require that a digit cannot have multiple occurrences in the same row or column. Instead, if after the attempted mutation these digits appear three times or less the mutation is done. This means that one too many is forgiven in order to help GA to swap positions by the help of other positions. We measure the solving speed and it is about 5 times slower, if no slack is given. If we forgive two instead of one, the solving speed is over 10 times slower.

These rules take some time to calculate, but the overall performance was enhanced. This rule also decreases the real mutation percentage from 0.6 even more.

Another special procedure we added was a restart or cataclysmic mutation [26]. In combinatorial problems, optimization is often stuck and it is more efficient to restart it with a new initial population than try to continue from the stuck situation. We did a test series where we reinitialized the population after 500, 1000, 2000, 5000, 10000, 20000 and 50000 generations and came to conclusion that an optimal interval is 2000 generations if no solution is found.

B. *Fitness function*

To design a fitness function that would aid the GA search is often difficult in combinatorial problems [27]. In this case, we originally [28] used a somewhat complex fitness function that penalized different constraint violations differently. In the first tests, we required that each row and column sums must be equal to 45 and also that each row and column product must be equal to 9!. The third requirement was derived from the set theory. It required that the each row, x_i, and column, x_j, was considered as set that must be equal to the set A, which contains integers from 1 to 9; if not then a penalty was added to the fitness value. The system worked somewhat well, but by removing parts from the fitness function we came to a conclusion that a much simpler fitness function performs just as well.

The conditions that every 3×3 subsgrid contains integers from 1 to 9 and the fixed numbers was guaranteed intrinsically and penalty functions are used in order to trying to force the other conditions. The fitness function used in this paper has only the set theory part left.

Consider the set A that contains each integer from 1 to 9:

$$A = \{1,2,3,4,5,6,7,8,9\}$$
$$g_{i3}(x) = |A - x_i| \quad (1)$$
$$g_{j3}(x) = |A - x_j|$$

The functions (1) calculate the number of missing digits in each row (x_i) set and column (x_j) set, where $|\cdot|$ denotes for the cardinality of a set.

In the optimal situation all digits appear in the row and column sets, and fitness function value becomes zero. Otherwise digit penalty value $|n\text{-}1|$ is added to fitness function. This means that if some digit is missing a penalty value 1 is added, and if some number appears n times, where $n>1$, the penalty value n-1 is added. Therefore each mistake in some row or column set equals at least a penalty value 2. The fitness function is to be minimized.

```
If Best(generation_i) = Best(generation_i-1)
   then Value(Best)+=1;                    (2)
```

We also added a special penalty term (2) that adds penalty value 1 to the best solution each generation if the same solution holds the best position. This means that when a new solution becomes best, its value is the value it gets from the fitness function. If it is still the best solution in the next generation we added value 1 to its fitness value. This operation can be seen as some kind of aging process of the individual's phenotype. It is not as fit and strong if it lives longer. This operation was added in order to make more variation to the population, and our preliminary tests also showed that it is beneficial and it resulted a faster solving of Sudoku.

III. Experimental Results

We start by testing five Sudoku puzzles taken from the newspaper Helsingin Sanomat [29] marked with their difficulty rating 1-5 `stars`. These had 28 to 33 symmetric givens. We also tested four Sudokus taken from newspaper Aamulehti [30]. They were marked with difficulty ratings: `Easy`, `Challenging`, `Difficult`, and `Super difficult`. They contain 23 to 36 nonsymmetrical givens.

One objective of this study was also to improve Sudoku solving from the previous version [28]. From table 1 it is clear that the changes we made significantly improved the solving rate of Sudokus compared to the previous version. We used max. 5 000 000 trials in [28] and this time unlimited and max. 100 000 trials versions. The Sudokus rated as 4 `stars` and 5 `stars` seemed to be in wrong

order in [28], since the 5 stars Sudoku was solved more often than the 4 stars Sudoku, this time they are in right order. With unlimited trials all Sudokus were always solved with new version, whereas in [28] only two of them were always solved. The longest run with the new version was with Super difficult Sudoku and it needed 4 065 900 trials. If only 100 000 trials were used, the new version results were still better than with the former version.

TABLE I
THE COMPARISON OF PREVIOUS AND NEW GA

Difficulty rating	1 st	2 st	3 st	4 st	5 st	Ea-sy	Chal leng.	Dif ficu l	Supe r d.
GA, old	100	69	46	26	23	100	30	4	6
GA, new	100	100	100	100	100	100	100	100	100
GA 100 000	100	100	96	63	47	100	60	10	8

The comparison of how often the previous GA [28] and the new versions found solutions for the nine Sudoku puzzles with different difficulty rating out of 100 test runs. The 1-5 stars Sudokus have with symmetric givens and they and their difficulty ratings taken from the newspaper Helsingin Sanomat. The Sudokus with ratings Easy to Super difficult are taken from newspaper Aamulehti and they had nonsymmetrical givens.

A. *Sudoku Solving and Rating*

The findings in [28] seem to suggest that the Sudoku rating seemed to be nonlinear and sometimes in wrong mutual order in respect to their GA hardness. To test these properties we need study more Sudokus. Therefore, this time we decided to test 27 given Sudokus, three representatives from each of the categories mentioned earlier.

TABLE 2
THE COMPARISON OF HOW EFFECTIVELY GA FINDS SOLUTIONS FOR THE SUDOKU PUZZLES WITH DIFFERENT DIFFICULTY RATING

Diffi-culty Rating	Median of Solve generations			Generations needed (a, b, c)			
	a	*b*	*c*	*Min*	*Mean*	*Med*	*Max*
1	110	58	1006	20	519	125	8157
2	1084	4703	1295	60	3331	1471	28324
3	1437	5667	5121	52	6159	3541	52352
4	4751	7094	4134	95	7572	5100	54380
5	6099	6656	9862	68	11753	7355	63422
E	62	34	65	16	64	50	311
C	3410	8904	4249	92	7878	4852	57180
D	27433	11979	6348	153	24354	12601	203295
SD	23615	20869	17815	132	29078	20285	159491

There are three different Sudokus (a, b, and c) from each of the nine difficulty classes 1-5 stars, Easy, Challenging, Difficult and Super Difficult. Each of the Sudokus was solved 100 times and the table shows median value of how many generations was needed for each Sudoku. There is also the minimum, average, median and maximum number of generations needed to find solution for the three Sudokus with the same difficulty rating.

Table 2 shows that the GA hardness of Sudokus is relatively consistent with their difficulty rating. However, *e.g.* 5 stars Sudoku b seems to easier than 4 stars Sudoku b. The average and median solving speed of difficulty classes seems to correlate well with the difficulty ratings, but sometimes the upper and lower bounds do not.

Comparing the Sudokus from different source, the Easy from Aamulehti is the easiest, followed by 1-4 stars Sudokus from Helsingin sanomat before Challenging from Aamulehti, 5 stars from HS, and finally Difficult and Super difficult from Aamulehti as the most difficult ones.

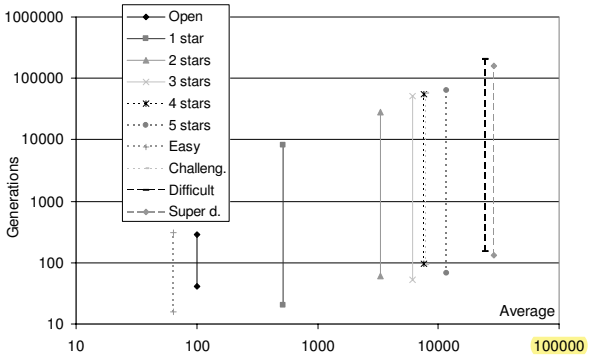

Fig. 5. The difficulty order of tested Sudokus. The minimum and maximum generations needed to solve each Sudoku with 100 test runs as a function of average generations needed.

The Sudokus in Helsingin Sanomat have the symmetrical givens and Sudokus in Aamulehti has nonsymmetrical givens, so the findings may suggest that the Sudokus with nonsymmetrical givens are more difficult. However, this conclusion would require more evidence.

Figure 5 shows the difficulty order of tested Sudokus, the minimum and maximum generations needed to solve each Sudoku from 100 test runs as a function of average generations needed. From fig. 5 it is easy to see that the Easy is even easier to solve than find a new open Sudoku solution. Otherwise the difficulty order seems to divide smoothly to the whole logarithmic average scale. The 1 star to 5 stars Sudokus from [29] are smoothly divided in the right order. The Sudokus from [30] Easy to Super difficult divides into larger space relatively smoothly and in the right order. The Challenging locates together with 4 stars Sudoku and Difficult and Super difficult seems to form their own group.

The solving of all these 27 Sudoku puzzles 100 times each took 185 minutes with Java programmed GA and using 3000 MHz Pentium4, *i.e.* in average 4.11 seconds for solving one Sudoku.

B. *The New Sudoku Puzzle Creation*

We also generated new Sudoku puzzles by first searching a new open Sudoku solution, not any given numbers in the beginning, and then randomly drawing 20-50 givens. The new puzzle was then solved max. 10 times in order to test that only a single solution exists.

The preliminary tests seemed to suggest that if the same solution is found more than four times in successive solve

runs, it is likely the unique solution. We ran the Sudoku generation program 100 times. It needed to find from 1 to 251 open Sudoku solutions (average 55) before printing a new puzzle. Table 3 shows how the first solve attempt inside of these 100 puzzle generations resulted to a different solution than the original open solution 5293 times, and sometimes it found the same solution 1-4 times. All these 5458 puzzle candidates were rejected. Another 100 times the solve program found the same solution 10 times in a row. Since the existence of multiple solutions was always detected in 4 solve runs, we can assume that 10 solve runs with identical solutions is enough to suppose that only a single solution exists.

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

Fig. 6. Example of our GA generated `medium` difficult Sudoku

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

Fig. 7. Example of our GA generated `hard` Sudoku

TABLE 3
THE TIMES THAT THE SAME SOLUTION FOUND IN SUGGESSION

N times	0	1	2	3	4	5-9	10
Count	5293	157	33	7	1	0	100

This by no means confirms that we definitely have an unique solution. Main reason for trying to "proof" the uniqueness this way was that we want to find out, if GA can also be used for uniqueness testing. It is much faster and maybe more reliable to test uniqueness with some faster method. We also tested half of our GA generated Sudokus with SudokuExplainer software [31] and it confirmed that each of the Sudokus generated by our method had a unique solution.

C. The Difficulty of New Sudokus

A found open Sudoku was solved 100 times in order to declare its difficulty rating. We decide to declared the difficulty rating with the help of earlier results represented in fig. 5. Since the 9 easiest Sudokus where solved in average less than 3500 generations, we decide that solving averagely less <3500 generations is `easy`, and correspondingly `hard` was more than 10000 generations and those in the middle are `medium`. Table 4 shows that out of these 100 puzzles generated 43 were `easy` and 21 were `hard`. It seems that our GA generates Sudokus relative evenly to each category. The proportion of Sudokus generated to different difficult classes can be changed by changing the expected value of givens or by removing extra givens from the generated Sudoku, while keeping the solution unique.

TABLE 4
THE COMPARISON OF HOW DIFFICULT GA GENERATED SUDOKUS ARE IN COMPARISON WITH THE SUDOKUS TESTED FOR TABLE 2

Type of Sudokus	Average solve generations			Total
	<3500	*3500-10000*	*>10000*	
`Original`	9	9	9	27
`GA generated`	38	41	21	100

The example of our GA generated `easy` Sudoku (GA average was 839) can be found from fig. 1 and examples of `medium` (GA avg. 9100) in fig. 6, and `hard` (GA avg. 52211) from the fig. 7. The SudokuEplainer [31] software gave these Sudokus the difficulty ratings 1.2, 2.0 and 7.8 respectively.

D. Comparing our results with other studies

We found only three studies about Sudoku optimization with GAs [5], [11], and [12]. In [12] GA is used GA for generating new Sudoku puzzles. In their given example GA needed 35700 generations to come up with a new open Sudoku solution. In our results, we create a new open Sudoku solution, in average in 101 generations (2020 trials).

Another study [5] represents the results with a large number of different methods, total of 41 tables with different strategies. These strategies are divided into three groups: Hamming space crossovers, Swap space crossovers and Hill climbers. The worst results in each group never reach the solution. We decided to compare our results with the best results for each groups represented in [5]. Unfortunately, we do not know how many fitness evaluations they used, since their stopping criterion was 20 generations with no progress (50000 trials with population size 5000, and elitism 2500).

With hill climbers they reported using 100000 trials.

TABLE 5
OUR RESULTS AND THE BEST RESULTS REPRESENTED IN [5]

Sudoku problems from [6]	Our GA Unlimited trials	Our GA 100 000 trials	The best results represented in [5] Hamming Space crossovers	Swap Space crossovers	Hill Climbers
Easy 1	30	29	5	28	30
Easy 2	30	30	8	21	30
Easy 3	30	30	14	30	30
Medium	30	10	0	0	0
Hard	30	2	0	15	0
Total	150	101	27	94	90

The numbers represents how many times out of 30 test runs each method reach the optimum with each problem.

For comparison purposes we tested five Sudokus from [6] with our GA to obtain comparable results. However, we do not know if the Sudokus taken from [6] are exactly the same as they used, so we chose the first three from the Easy category and the first one from Medium and Hard categories.

Table 5 shows our results with unlimited trials and with 100 000 trials. The 100 000 trials version should be compared to hill climbers, where we know that [5] used that amount of trials.

Our both version (unlimited and 100 000 trials) GA performed better than their best GA version with each category when comparing the total numbers. Only with Easy1 our 100 000 trials version performed worse than their "hill climbers" and with Hard Sudoku worse than their "Swap space crossovers" version.

With our unlimited trials version the longest solve run was with the Hard Sudoku and it lasted 8 373 740 trials.

The [11] have also presented very good result by GAuGE system (Genetic Algorithms using Grammatical Evolution). They had taken their benchmark Sudokus from a book that was unavailable for us. Thus, we cannot directly compare our results with their method. However, out of 20 benchmark Sudokus they find the solution every time out of 30 test runs for 17 problems, but for two problems their method was unsuccessful of finding a solution with 320 000 trials.

Our method has never failed to find a solution of Sudoku. The hardest Sudoku we tested was AI Escargot by Arto Inkala [32]. Without a trials limit it was solved every time. When using a limited number of trials it was solved 8 times out of 100 test runs with 100 000 trials and 18 times out of 100 with 320 000 trials. In the fastest solve run it was solved with only 7220 trials, in average it required 1 238 749 trials. The longest solve run required 5 901 680 trials, which was less than Hard Sudoku from [6] needed in the worst case.

IV. CONCLUSIONS AND FUTURE

In this paper, we studied if Sudoku puzzles can be solved, rated and generated with a combinatorial genetic algorithm. The results show that GA can solve Sudoku puzzles relatively effectively. However, there exists some more efficient algorithms to solve Sudoku puzzles *e.g.* [8], [9] and [10] are fast, but in the results reported, all these methods fail to solve some Sudoku puzzles they tested. In any case, our results stand well the comparison with other known results with evolutionary algorithms.

In this study, the aim was to test how efficient pure GA is, without many problem specific rules for solving Sudokus. The GA results can of course be enhanced by adding problem related rules. However, if one adds too much problem specific logic to the Sudoku solving, there will be nothing left to optimize, therefore we decided to omit most problem specific logic and try a more straight forward GA optimization approach.

It is also stated [33] that Sudoku is a good laboratory for algorithms design, and it is based on one of the hardest unsolved problems in computer science – the NP complete problems. They think that Sudoku craze may even end up leading breakthroughs in computer science.

The other goal was to study if difficulty ratings given for Sudoku puzzles in newspapers are consistent with their difficulty in GA optimization. The answer to that question seems to be positive. With some solitary puzzles the rating seems wrong, but the overall trend follows the ratings; those Sudokus that have higher difficulty rating proved also to be more difficult for genetic algorithms. This means that GA can be used for rating the difficulty of a new Sudoku puzzle. Rating puzzles is said to be one of the most difficult things in Sudoku puzzle creation [4], so GA can be a helpful tool for that purpose. However, the other explanation can be that the original puzzles are also generated with computer programs, and since GA is also a computer based method, it is possible that a human solver does not necessarily experience their difficulty the same way.

The new puzzles were generated by finding one possible open Sudoku solution and then removing numbers and testing if only one solution exists. It seems that we can suppose the solution unique if it is found more than four times in a row, since each our GA generated puzzles had a unique solution according the SudokuExplainer software [31]. However, we think it is better to be safe and test the uniqueness more times, and overall it is wiser to use other methods for testing the solution uniqueness.

The future research may consist of generating more efficient hybrid of GA and algorithms created specifically to solve Sudokus.

Recently, we have also tested memetic algorithms [34] based method for this problem. For some Sudokus it worked much better than this GA version, but for some others the average results were worse. The results may be reported later, since for the whole benchmark set they were, in average, about 8 percent better than results represented in this paper.

It has been said that 17 given numbers is minimal needed to come up with a unique solution, but it is not

mathematically proven [4]. GA could also be used for minimizing the number of givens that still leads the unique solution. In this study, we generated Sudokus with 20 to 50 givens and found some new Sudokus with a unique solution that had 21 or 22 givens, but we did not tried to find minimum number of givens.

The fitness function setting in this study worked satisfactorily, but in the future we might study more whether or not this is a proper GA fitness function for the Sudoku problem. We are already considering if it is possible to generate Sudoku a fitness function based on energy functions [27].

REFERENCES

[1] Darwin, C.: *The Origin of Species: By Means of Natural Selection or The Preservation of Favoured Races in the Struggle for Life*, Oxford University Press, London, 1859, A reprint of the 6th edition (1968)

[2] Wikipedia: *Sudoku*. Available via WWW: http://en.wikipedia.org/wiki/Sudoku (cited 16.10.2006)

[3] Sullivan, F.: Born to compute. *Computing in Science & Engineering* **8**(4), July (2006) 88

[4] Semeniuk, I.: Stuck on you. In *NewScientist* **24/31** December (2005) 45-47

[5] Moraglio, A., Togelius, J., Lucas, S.: Product geometric crossover for the sudoku puzzle. In *2006 IEEE Congress on Evolutionary Computation (CEC2006)*, Vancouver, BC, Canada, July 16-21 (2006) 470-476.

[6] Pappocom: *Su|do|ku*. Available via WWW: http://www.sudoku.com (cited 16.10.2006)

[7] J. Holland, *Adaptation in Natural and Artificial Systems*, The MIT Press (1992)

[8] Simonis, H.: *Sudoku as a constrain problem*. In B. Hnich, P. Prosser, B. Smith (eds.), Proc. 4th Int. Works. Modelling and Reformulating Constraint Satisfaction Problems, 2005, 13–27

[9] Lynce, I., Ouaknine, J.: *Sudoku as a SAT problem*. In 9th *International Symposium on Artificial Intelligence and Mathematics AIMATH'06*, January, 2006

[10] Moon, K., Gunther, J.: Multiple constrain satisfaction by belief propagation: An example using Sudoku. In *2006 IEEE Mountain Workshop on Adaptive and Learning Systems*, July 2006, 122 - 126

[11] Nicolau, M., Ryan, C.: Genetic operators and sequencing in the GAuGE system. In *IEEE Congress on Evolutionary Computation CEC 2006*, 16-21 July 2006, 1561 - 1568

[12] Gold, M.: *Using Genetic Algorithms to Come up with Sudoku Puzzles*. Sep 23, 2005. Available via WWW: http://www.c-sharpcorner.com/UploadFile/mgold/Sudoku 09232005003323AM/Sudoku.aspx?ArticleID=fba36449-ccf3-444f-a435-a812535c45e5 (cited 16.10.2006)

[13] *Sudoku Maker*. Available via WWW: http://sourceforge.net/projects/sudokumaker/ (cited 16.10.2006)

[14] Ardel, D.H.: TOPE and magic squares, a simple GA approach to combinatorial optimization. In J.R. Koza (ed.) *Genetic Algorithms in Stanford*, Stanford bookstore, Stanford, CA (1994)

[15] Alander, J.T., Mantere T., Pyylampi, T.: Digital halftoning optimization via genetic algorithms for ink jet machine. In B.H.V. Topping (ed.), *Developments in Computational mechanics with high performance computing*, CIVIL-COMP Press, Edinburg, UK (1999) pp. 211-216

[16] *Evonet Flying Circus*. Available via WWW: http://evonet.lri.fr/CIRCUS2/ node.php?node=65 (1996, cited 16.10.2006)

[17] Runadsson, T.P., Yao, X.: Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation* **4**(3) (2000) 284-294.

[18] Mantere, T.: A min-max genetic algorithm for min-max problems. In Wang, Cheung, Liu (eds.) *Advances in Computational Intelligence and Security - The Workshop of 2005 Int. Conference on Computational Intelligence and Security - CIS 2005*, Xi'an, China, December 15-19, 2005. Xidian university press, Xi'an, China (2005) 52-57.

[19] Li, H., Jiao, Y.-C., Wang, Y.: Integrating the simplified interpolation into the genetic algorithm for constrained optimization problems. In Hao et al. (eds) *CIS 2005*, Part I, *Lecture Notes on Artificial Intelligence* 3801, Springer-Verlag, Berlin Heidelberg (2005) 247-254.

[20] Kang, H.R.: *Digital Color Halftoning*. SPIE Optical Engineering Press, Bellingham, Washington, & IEEE Press, New York (1999)

[21] Bayer, B.: An optimum method for two-level rendition of continuous tone pictures. In *Proc. Of Int. Conference on Communications* 1, NY, June 11-13 (1973) 11-15

[22] Alander, J.T., Mantere T., Pyylampi, T.: Threshold matrix generation for digital halftoning by genetic algorithm optimization. In D. P. Casasent (ed.), *Intelligent Systems and Advanced Manufacturing: Intelligent Robots and Computer Vision XVII: Algorithms, Techniques, and Active Vision*, volume SPIE-3522, Boston, MA, 1-6 November 1998. SPIE, Bellingham, Washington, USA (1998) 204-212

[23] Kobayashi, N., Saito. H.: Halftone algorithm using genetic algorithm. In *Proc. of 4th Int. Conference on Signal Processing Applications and Technology*, vol. 1, Newton, MA 28. Sept. – 1. Oct. 1993, DSP Associates, Santa Clara, CA (1993) 727-731

[24] Newbern, J., Bowe Jr., M.: Generation of Blue Noise Arrays by Genetic Algorithm. In B.E.Rogowitz and T.N. Pappas (eds.) *Human Vision and Electronic Imaging II*, San Jose, CA, 10.-13. Feb. 1997, Vol SPIE-3016, SPIE - Int. society of Optical Engineering, Bellingham, WA (1997) 441-450

[25] Wiley, K.: Pattern Evolver, an evolutionary algorithm that solves the nonintuitive problem of black and white pixel distribution to produce tiled patterns that appear grey. In *The Handbook of Genetic Algorithms*. CRC Press (1998)

[26] Eshelman, L.J.: The CHC adaptive search algorithms: how to safe search when engaging in nontraditional genetic recombination. In G. Rawlinns (ed.) *Foundations of Genetic Algorithms*, Morgan Kaufmann (1991)

[27] Koljonen, J., Alander, J.T.: Solving the "urban horse" problem by backtracking and genetic algorithm – a comparison. In J. Alander, P. Ala-Siuru, and H. Hyötyniemi (eds.), *Step 2004 – The 11th Finnish Artificial Intelligence Conference*, Heureka, Vantaa, 1-3 September 2004, Vol. 3, Origin of Life and Genetic Algorithms (2004) 127-13.

[28] Mantere, T., Koljonen, J.: Solving and rating sudoku puzzles with genetic algorithms. In *Finnish Artificial Intelligence Conference (STeP 2006)*, October 26-27, Finnish Artificial Intelligence Society FAIS, Espoo, Finland (2006) 86-92

[29] Helsingin Sanomat: *Sudoku*. Available via WWW: http://www2.hs.fi/extrat/sudoku/ sudoku.html (cited 11.1.2006)

[30] Aamulehti: *Sudoku online*. Available via WWW: http://www.aamulehti.fi/sudoku/ (cited 11.1.2006)

[31] SudokuExplainer: Available via WWW : http://diuf.unifr.ch/people/juillera/Sudoku/Sudoku.html (cited 12.6.2007)

[32] Inkala, A.: *AI Sudoku 1002 Vaikeaa Tehtävää*, Pressmen Finland Oy, 2006

[33] Aaronson, L.: *Sudoku science*. IEEE Spectrum **43**(2), February (2006) 16-17

[34] Krasnogor, N., Smith, J.: A tutorial for competent memetic algorithms: model, taxonomy and design issues. *IEEE Transactions on Evolutionary Computation* **9**(5), 2005, 474- 488