Solving sudoku's by evolutionary algorithms with pre-processing

Metadata of the chapter that will be visualized in SpringerLink

Book Title	Recent Advances in Soft Computing	
Series Title		
Chapter Title	Solving Sudoku's by Evolutionary Algorithms with Pre-processing	
Copyright Year	2019	
Copyright HolderName	Springer Nature Switzerland AG	

Author	Family Name	**Amil**
	Particle	
	Given Name	**Pedro Redondo**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	University of Córdoba
	Address	C2, Rabanales Campus, C.P. 14071, Córdoba, Spain
	Email	
Corresponding Author	Family Name	**Mantere**
	Particle	
	Given Name	**Timo**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	University of Vaasa
	Address	PO Box 700, 65101, Vaasa, Finland
	Email	timan@uva.fi

Abstract	This paper handles the popular Sudoku puzzle and studies how to improve evolutionary algorithm solving by first pre-processing Sudoku solving with the most common known solving methods. We found that the pre-processing solves some of the easiest Sudoku's so we do not even need other methods. With more difficult Sudoku's the pre-processing reduce the positions needed to solve dramatically, which means that evolutionary algorithm finds the solution much faster than without the pre-processing.

Keywords (separated by '-')	Ant colony optimization - Cultural algorithms - Genetic algorithms - Hybrid algorithms - Puzzle solving - Sudoku

Solving Sudoku's by Evolutionary Algorithms with Pre-processing

Pedro Redondo Amil[1] and Timo Mantere[2](✉)

[1] University of Córdoba, C2, Rabanales Campus, C.P. 14071 Córdoba, Spain
[2] University of Vaasa, PO Box 700, 65101 Vaasa, Finland
timan@uva.fi

AQ1

Abstract. This paper handles the popular Sudoku puzzle and studies how to improve evolutionary algorithm solving by first pre-processing Sudoku solving with the most common known solving methods. We found that the pre-processing solves some of the easiest Sudoku's so we do not even need other methods. With more difficult Sudoku's the pre-processing reduce the positions needed to solve dramatically, which means that evolutionary algorithm finds the solution much faster than without the pre-processing.

Keywords: Ant colony optimization · Cultural algorithms · Genetic algorithms Hybrid algorithms · Puzzle solving · Sudoku

1 Introduction

In this paper we will report our newest developments with solving Sudoku's with evolutionary algorithms. This research project started as a hobby back in 2006 when we published our first Sudoku paper [1]. Since then we have published other papers, and got many citations to them and still many people inquire us of our Sudoku papers and software. Therefore, we felt that now is time to update our Sudoku experiments.

This time we decided to program the most common Sudoku solving methods and use them to pre-process Sudoku's before applying evolutionary algorithms to solve them. We also add some literature and background information that we have not discussed in our earlier Sudoku papers.

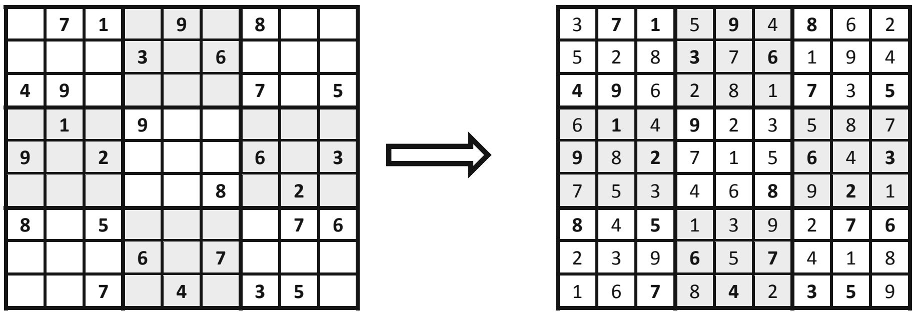
According to Wikipedia [2], Sudoku, originally called Number Place, is a logic-based, combinatorial number-placement puzzle (Fig. 1). The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub grids that compose the grid (also called "boxes", "blocks", "regions", or "sub squares") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a unique solution.

We will try to solve the Sudoku with different algorithms: Genetic Algorithms, Cultural Algorithms and Genetic Algorithm/Ant colony optimization hybrid.

Genetic algorithms (GAs) [3] are computer based optimization methods that use the Darwinian evolution [4] of nature as a model and inspiration.

Cultural algorithms (CAs) were introduced by Reynolds [5], they are a branch of evolutionary computation methods, where algorithms include knowledge component,

R. Matoušek (Ed.): MENDEL 2017, AISC 837, pp. 1–13, 2019.
https://doi.org/10.1007/978-3-319-97888-8_1



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

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

Fig. 1. Sudoku and its solution

belief space. Cultural algorithm is usually an extension of conventional genetic algorithm that has an added belief space. Ant colony optimization algorithm (ACO) [6] in turn is a probabilistic technique for solving computational problems, which can be reduced to finding good paths through graphs, ant colony also gather cultural knowledge.

Hybrid algorithm [7, 8] refers to the combination of two or more algorithms. We will use here the Genetic Algorithms/Ant colony optimization hybrid algorithm.

The pre-processing consists in filling sure number in different squares of the Sudoku through several methods, we will use this pre-processing with every algorithm that we have described before.

The objective of this study is to test if Genetic algorithms, Cultural algorithms and Genetic algorithm/Ant colony optimization hybrid algorithm are efficient methods for solving Sudoku puzzles; some Sudoku's generate by our Genetic Algorithm and some of them from a newspaper or website.

The document is organized as follows. In Sect. 1, we have included the objectives of this study. In Sect. 2 we describe the Sudoku. In Sect. 3 we have defined the different algorithms. In Sect. 4 we explain the result of the experiments and finally in the Sect. 5 we say our conclusions.

2 Sudoku

Sudoku puzzles are related to Latin squares [9], which were developed by the 18th century Swiss mathematician Leonhard Euler. Latin squares (Fig. 2) are square-grids of size n × n where each of the character (numbers, letters, symbols…) from 1 through n appears in every column and in every row precisely once (rank n Latin squares).

Magic squares [10] are square grids that are filled with (not necessarily different) numbers such that the numbers in each row and column add to up to the same sum.

In the late 19th century a Paris-based daily newspaper, Le Siècle (discontinued) published a partially completed 9 × 9 magic square that had 3 × 3 sub grids. The object of the game was to fill out the magic square such that the numbers in the grids also sum to the same number as in the rows and columns.

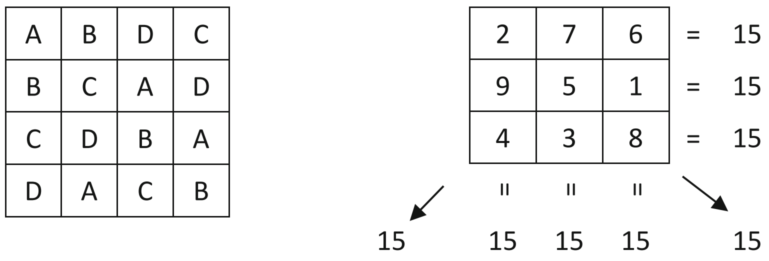


Fig. 2. Examples of Latin square (left) and magic square (right)

The name of the game is of Japanese origin, the word "SuDoku" is abbreviation of "Suuji wa dokushin ni kagiru" that it means, "The digits must remain single". It was not until 1986 when the Japanese company Nikoli, Inc. started to publish a version of the Sudoku at the suggestion of its president, Mr. Maki Kaji. He gave the game its now famous name. Almost two decades passed before (near the end of 2004) The Times newspaper in London started to publish Sudoku as its daily puzzle due to the effort of Wayne Gould, who spend many years to develop a computer program that generates Sudoku puzzles. By 2005, it became an international hit.

A complete Sudoku solution may be arrived at in more than one ways, as we can start from any of the given clues that are distributed over the sub grids of a given Sudoku (to be solved). There is no known technique that we surveyed to determine how many different starting squares (the first square that we can fill) there are. Removal of a single number given may generate another Sudoku different for which other solutions may exist and the solution is no longer unique.

The puzzles require logic, sometimes intricate, to solve but no formal mathematics is required. However, the puzzles lead naturally to certain mathematical questions. For example: Which puzzles have solutions and which do not? If a puzzle has a solution, is it unique? A couple of years ago it was proven that there has to be minimum of 17 givens in order to Sudoku have unique solution [11]. With the experiments of our Sudoku puzzle generator (yet unpublished paper), we have observed that there are several Sudoku's that do have two or more valid solutions even if the number of givens is 17 or more.

There could be Sudoku's with multiple solutions, but those that are published in newspapers usually have only one unique solution. Therefore, we can say that, a subset of Sudoku puzzles may have one and only one valid solution, but in general, a Sudoku might have two or more solutions as well. Sudoku is a combinatorial optimization problem [12], where each row, column, and 3×3 sub squares of the problem must have each integer from 1 to 9 once and only once. This means that the sum of the numbers in each row and each column of the solution are equal to 45 and the multiply of the numbers in each row and each column of the solution are equal to 9! (362880).

2.1 Pre-processing the Sudoku

To solve the Sudoku, we can pre-process it before applying some algorithm. The pre-processing consists in filling sure number in different squares of the Sudoku through several methods: full house, naked singles, hidden singles, lone rangers and naked pairs.

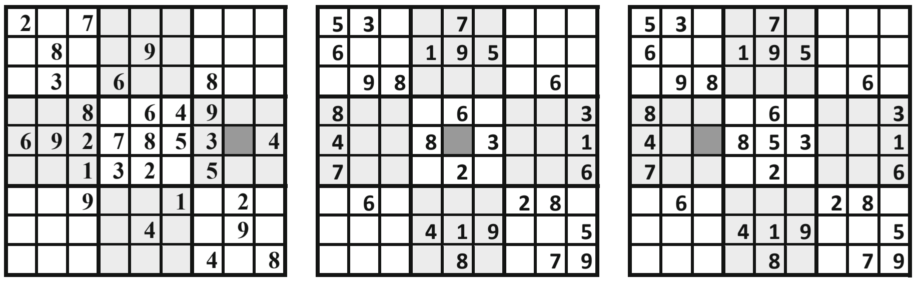


Fig. 3. Examples of full house (left), naked singles (middle) and hidden singles (right)

We used only these methods [13] because the many of the rest of methods take more time than to solve the Sudoku, because they perform several checks.

After applying the pre-processing, we use the different evolutionary algorithms to solve rest of the Sudoku, if it was not already solved.

The "Full house" method [3] consists in filling an empty cell where its value can be deduced when its row, column or 3 $\times$ 3 block has eight numbers, so we can get the last remaining number. In this example (Fig. 3), we check the central row and we know that the only number missing from this row is {1}, so we can fill this square (marked dark) with this number.

The "Naked singles" method [3] consist in filling an empty cell with the number not present in the set union of those in the same row, column and 3 $\times$ 3 sub block. This technique is applicable only when there is only one missing number. In this example (Fig. 3), we check the central square (dark square) and we know that the set from its row is {1, 3, 4, 8}, the set from its column is {1, 2, 6, 7, 8, 9} and the set from its 3 $\times$ 3 sub block is {2, 3, 6, 8}. We get a unique set joining all the sets and the result is {1, 2, 3, 4, 6, 7, 8, 9}. The only number missing from the set is {5}, so we can fill this square with this number.

The "Hidden singles" method [14] consists in filling an empty square if there is a value that can be placed in only one cell of a row, column, or 3 $\times$ 3 sub block. Then, this cell has to hold that value. In this example (Fig. 3), we check the fourth, and sixth rows and we see that the number {6} is in the fourth and sixth row, but it is not in the fifth one. This number must appear in the fifth row and in the left central 3 $\times$ 3 sub block. Since number {6} is already in first and second column, there is only one empty cell there where it fits (dark square), so it must hold this value, so we can fill this square with the number {6}.

The "Lone rangers" method [14] consists in filling an empty square that seems to have more than one possible candidate. Whereas a careful observation across the possible candidate lists of row, column or 3 $\times$ 3 sub block neighbours of that empty square reveals the exact candidate, because one of the values appear just in one cell of the row, column or 3 $\times$ 3 sub block. In this example (Fig. 4), we check the third row and we see that the number {6} only appears in one square of this row (with circle) like a candidate, so we can fill this square with this number.

The "Naked pairs" method [14] consists in removing candidates from the candidate list of squares. If two cells in a row, column or 3 $\times$ 3 sub block have the same pair of

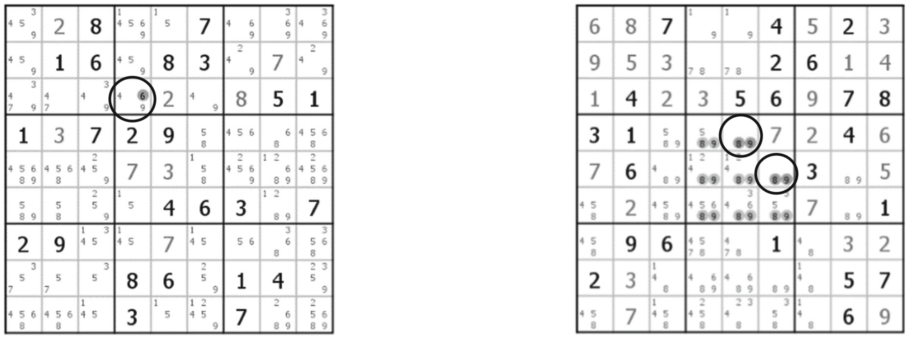


Fig. 4. Examples of lone rangers (left), and naked pairs (right)

candidates then they cannot be housed in any other cell of the same row, column, or 3 × 3 sub block, and therefore they can be removed from their candidate lists. In this example (Fig. 4), we check the central square and we see that two squares have the same candidates (marked with circle) that they are {8, 9}, then we can delete these numbers from the candidate lists of the other cells in the same sub block. Subsequently, another technique might possibly be applied to deduce the value of a cell. For instance, by the "Naked single", so we get the number {5} is the number of the first square of this 3 × 3 square because it is the only candidate number.

3 The Algorithms Used

We used different algorithms for solving the Sudoku's. The first one is a Genetic Algorithm [3], using crossover and mutation operators. The second one is a method that pre-process the Sudoku before applying the Genetic Algorithm, filling the squares that have only one possible value. The third one is a Cultural Algorithm [5]. The fourth one is a method that pre-process the Sudoku before applying Cultural Algorithm. The fifth one is a hybrid algorithm of Genetic Algorithm with the ant colony optimization [6]. The sixth one is a method that pre-process the Sudoku before applying the hybrid algorithm.

The algorithms used are the same as in papers [15, 16] where we have explained them with details. The algorithms used in [16] were used as a base and pre-processing methods were programmed to the source codes used in that study. Also all GA, CA and GA/ACO results without pre-processing are obtained with algorithms presented in [16].

3.1 Genetic Algorithms

Genetic Algorithms (GAs) [3] are computer based optimization methods that use the Darwinian evolution [4] of nature as a model and inspiration. It can be defined also like a search heuristic that mimics the process of natural selection.

This heuristic (also sometimes called a metaheuristic) is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection and crossover.

When solving Sudoku's we need to use a GA that is designated for combinatorial optimization problems. That means that it will not use mutations or crossovers that could generate illegal situations, like: rows, columns, and sub squares would contain some integer from {1, …, 9} more than once, or some integers would not be present at all. In addition, the genetic operators are not allowed to move the static numbers that are given in the beginning of the problem (givens), so we need to represent the Sudoku puzzles in GA program so that the givens will be static and cannot be moved or changed with genetic operations.

The crossover operation is applied so that it exchanges whole sub blocks of nine numbers between individuals, thus, the crossover point cannot be inside a building block. The mutations contrarily are applied only inside a sub block.

We use the technique of Swap mutation. In swap mutation, the values in two positions are exchanged. In this example (Fig. 5), we change the numbers 2 and 5. Each time mutation is applied inside the sub block (3 × 3 blocks), the array of givens is referred. If it is illegal to change that position, we randomly reselect the positions and we recheck until legal positions are found. We are using the mutation strategy of swap mutation with an overall mutation probability of 0.12. In swap mutation, the values in two positions are exchanged.

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

Fig. 5. Swap mutation

3.2 Cultural Algorithms

Cultural algorithms (CAs) were introduced by Reynolds [5], they are a branch of evolutionary computation methods, where algorithms includes knowledge component, belief space. A CA is usually an extension of conventional GA that has added belief space.

The belief space of a cultural algorithm is divided into distinct categories. These categories represent different domains of knowledge collected from the search space. Belief space could collect, *e.g.* domain specific knowledge, situational knowledge, temporal knowledge, or spatial knowledge. Belief space acts as an extra reproduction component that somehow affects to the generation of new individuals.

If we compare our CA with GA, the main difference is that we added a belief space model, which in this case is simple: it is a 9 × 9 × 9 cube, where the first two dimensions correspond to the positions of a Sudoku puzzle, and the third dimension represents the nine possible digits for each location.

After each generation, the belief space is updated if:

(1) The fitness value of best individual is 2 (two numbers in wrong positions).
(2) The best individual is not identical with the individual that updated the belief space the previous time.

The belief space is updated so that the value of the digit that appears in the best Sudoku solution is incremented by one in the belief space. This model also means that the belief space is updated only with near-optimal solutions.

The belief space collects information from the near-optimal solutions (only two numbers in "wrong" positions), and this information is used only in the population re-initialization process. When the population is reinitialized, positions that have only one non-zero digit value in the belief space are considered as givens. These include the actual givens of the problem, but also so called hidden givens that the belief space have learned, that is, those positions that always contain the same digit in the near-optimal solutions. This also connects the reinitialized population to the previous generations, since the new initial population is not formed freely.

3.3 GA/ACO Hybrid Algorithm

In computer science and operations research, the Ant Colony Optimization algorithm (ACO) is a probabilistic technique for solving computational problems, which can be reduced to finding good paths through graphs, initially proposed by Marco Dorigo 1992 in his PhD thesis [6]. The original idea was to search for an optimal path in a graph based on the behaviour of ants seeking a path between their colony and a source of food. The method has since diversified to solve a wider class of numerical problems, and simulating on various aspects of the behaviour of ants. The basic idea is that ants leave trail of pheromone, which is stronger, if the path is good.

In our Genetic Algorithm/Ant colony optimization hybrid algorithm, the GA part acts as a heuristic global searcher that generates new paths and most of the new individuals, the GA also finds the initial population for this hybrid. The ACO part acts like a greedy local searcher that tries quickly to converge into the strongest path. In addition, ACO collects cultural information stored in pheromone trails and it inserts new individuals generated by ACO, that is, by weighted random generator, where weights are proportionate to the pheromone strength. The ACO parts act in this hybrid algorithm somewhat similarly as a belief space in CAs.

In our algorithm, we generate the initial generation randomly. After each generation, the old pheromone paths are weakened 10%. Then the best individuals update the pheromone trails by adding strength value.

3.4 Fitness Function

To design a fitness function that would aid a GAs, CAs or hybrid algorithms search is often difficult in combinatorial problems. In this case, we decided to use to a simple fitness function that penalizes different constraint violations differently.

The condition that every 3×3 sub block contains a number from 1 to 9 is guaranteed intrinsically, because of the chosen solution encoding. Penalty functions are used to evaluate the other conditions. Each row and column of the Sudoku solution must contain each number $\{1, \ldots, 9\}$ once and only once. This can be transformed to a set of inequality constraints.

Fitness function is combined of three different rules (Eqs. 1–3). The first part (1) requires that all digits {1, …, 9} must be present in each row and column, otherwise penalty P_x is added to the fitness value.

$$P_x = \sum_{i=1}^{8}\sum_{j=1}^{8}\sum_{ii=i+1}^{9}\sum_{jj=j+1}^{9}\left[\left(x_{i,j} \equiv x_{ii,j}\right) + \left(x_{i,j} \equiv x_{i,j}\right)\right] \tag{1}$$

```
if Best(generation(i))=Best(generation(i-1)) then Value(Best)+= 1;
```
(2)

$$P_g = \sum_{i=1}^{9}\sum_{j=1}^{9}\left(x_{ij} \equiv g_{ij}\right) \tag{3}$$

Equation (1) count the missing digits in each row (x_i) and column (x_j). In the optimal situation, all digits appear in all the row and column sets and value of this fitness function part becomes zero. The second part of the fitness function (2) is for aging the best individual, it increments its fitness value by one on each generation, when it remains the best. This is easy to perform since our sorting algorithm only sorts indexes, if the same individual remains best, also, the index to it remains same.

The third component (3) of the fitness function requires that the same digit (x_{ij}) as some given (g_{ij}) must not appear in the same row or column as that given. Otherwise, penalty P_g is added.

Note, due to lack of space we do not explain all the details about the old versions of our GA, CA and GA/ACO Sudoku solvers in this paper. Instead, we are trying to summarize the main details needed in order to follow this paper separately from our previous papers. Those who are interested to learn more can are encouraged to read [15, 16]. Our genetic encoding is also explained in our Sudoku webpage [17]. Algorithms in this paper are identical to those in [16, 17] with the exception of added preprocessing part.

4 Experiments

We test five Sudoku puzzles taken from the newspaper Helsingin Sanomat marked with their difficulty rating 1–5 stars, where 1 star is the easiest. We test four Sudoku's taken from newspaper Aamulehti, they are marked with difficulty ratings: Easy, Challenging, Difficult, and Super difficult and we also tested three Sudoku's made with Genetic Algorithms from the Sudoku webpage [17] marked with difficulty ratings: Easy (GA-E), Medium (GA-M), Hard (GA-H).

We tried to solve each of the ten Sudoku puzzles 100 times. The stopping condition was the optimal solution found, which was always found with maximum seven hundred thousand trials (fitness function calls) for GA and CA, and four million trials for GA/ACO hybrid algorithm.

To compare which method is better, we will use the generations required to find the solution and the time spent in seconds. For this reason, we will get from each execution of the code the minimum, maximum and average of generations and the minimum, maximum and average of the time spent.

We done every test with a laptop with Windows 8 and the following hardware: Intel core i5-4200U 1.6 GHz with turbo boost up to 2.6 GHz, 4 GB DDR3 of Memory ram and 750 GB of capacity of Hard disk.

4.1 Genetic Algorithm

The results of the GA are given in Table 1 where the columns (left to right) stand for: difficulty rating, number of givens, minimum, maximum and average of the generations and minimum, maximum and average of the time required (in seconds) to solution.

Table 1 shows that the difficulty ratings of Sudoku's correlate with their GAs hardness. Therefore, the more difficult Sudoku's for a human solver seem to be also the more difficult for the GAs. The Easy from Aamulehti was the most easiest to solve in general. In addition, the average of the generations and the time needed to find the solution increased somewhat monotonically with the rating. The puzzles Difficult and Super Difficult of Aamulehti were much more difficult than any of the puzzles in Helsingin sanomat.

The results with pre-processed GA, also are given in Table 1, are obtained by first pre-processing 5 times (with 5 times the most of the Sudoku's are solved or it is not possible to find any more sure number) and later we use the Genetic Algorithms to solve the Sudoku's.

With respect to the results obtained without pre-processing, we can see that if we use the pre-processing of the Sudoku, we reduce drastically the generations needed to solve all Sudoku's. We reduce drastically the time spent also because the average of the time of every test is less than 1 s. For the Sudoku's that we put inside of the Easy and Medium group, only the pre-processing was enough to solve the Sudoku, so their generations needed and the time spent is very similar in all of them.

Table 1. Results of GA and GA with pre-processing.

Difficulty		Genetic algorithm						GA with pre-processing					
rating	Givens	Min. gene	Max. gene	Avg. gene	Min. time	Max. tim	Avg. Time	Min. gene	Max. gen	Avg. gene	Min. time	Max. tim	Avg. Time
1 star	33	14	95	23	0.036	0.246	0.062	1	1	1	0.022	0.045	0.035
2stars	30	24	2217	355	0.059	5.326	0.836	1	1	1	0.023	0.046	0.035
3 stars	28	36	1838	405	0.084	4.162	0.927	1	1	1	0.023	0.046	0.035
4 stars	28	39	6426	1139	0.088	15.016	2.701	8	363	92	0.025	1.099	0.289
5 stars	30	59	14805	2837	0.165	35.248	6.831	1	234	20	0.023	1.955	0.166
Easy	36	9	33	19	0.026	0.088	0.052	1	1	1	0.022	0.046	0.035
Challengi	25	38	3595	628	0.082	8.903	1.425	1	1	1	0.023	0.046	0.035
Difficult	23	88	44168	6093	0.197	102.103	13.890	6	379	55	0.059	3.629	0.538
Super dif	22	115	30977	7814	0.250	66.997	17.672	9	213	62	0.054	1.482	0.395
GA-E	32	15	291	74	0.038	0.700	0.187	1	1	1	0.022	0.045	0.035
GA-M	29	35	4436	893	0.083	10.360	2.110	1	1	1	0.023	0.046	0.035
GA-H	27	50	27143	4989	0.118	63.442	12.193	4	654	149	0.026	4.241	0.935
Avg. Gen+sum time		43.50	11335.33	2105.75	1.226	312.591	58.886	2.92	154.17	32.08	0.345	12.726	2.568

In the other hand, when we solve Sudoku's that we consider inside of the level Hard, the pre-processing basically changes their difficulty to Easy level, so when we solve those Sudoku's with the Genetic Algorithms, we do not need many generations.

With respect to the Sudoku's with 4 stars, after the pre-processing, it is still relatively difficult to solve, but easier than the 5 stars, difficult and super difficult Sudoku's.

4.2 Cultural Algorithm

The results given in Table 2 are using Cultural Algorithms. The columns used are the same than the Table 1. The results obtain with this algorithm are very similar to the results obtain with the Genetic Algorithms; in the most of the Easy and Medium Sudoku's this algorithm is a little bit better than the GAs one, but in the most of the hard Sudoku's the GA is a little bit better that CA. Both algorithms have similar average of the generation, average of the time spent, minimum generations and minimum time but the CA has better maximum generations and maximum time spent than the GA.

The results with CA and pre-processing are also given in Table 2. They are obtained using first the pre-processing 5 times and after that CA to solve rest of the Sudoku. The columns used are the same than the Table 1. The results obtained with this algorithm are a little bit better than the results with the pre-processed GA, but there is not a significant difference.

Table 2. Results of CA and CA with pre-processing.

Difficulty		Cultural algorithm						CA with pre-processing					
rating	Givens	Min. gene	Max. gene	Avg. gene	Min. time	Max. tim	Avg. Tim	Min. gene	Max. gen	Avg. gene	Min. time	Max. tim	Avg. Time
1 star	33	10	68	25	0.027	0.173	0.065	1	1	1	0.031	0.042	0.035
2stars	30	24	1240	362	0.056	2.878	0.841	1	1	1	0.032	0.038	0.035
3 stars	28	31	2411	445	0.071	5.390	1.005	1	1	1	0.032	0.039	0.035
4 stars	28	34	5810	908	0.081	13.333	2.132	9	362	87	0.027	1.06	0.259
5 stars	30	31	9129	2301	0.076	21.542	5.422	1	208	18	0.008	1.709	0.142
Easy	36	10	30	18	0.028	0.076	0.049	1	1	1	0.031	0.039	0.035
Challengi	25	40	10266	792	0.087	22.629	1.762	1	1	1	0.032	0.038	0.035
Difficult	23	80	26094	5829	0.170	56.193	12.651	6	230	51	0.039	2.124	0.486
Super dif	22	116	34207	8603	0.250	72.975	18.421	7	253	59	0.065	2.387	0.552
GA-E	32	19	329	84	0.047	0.781	0.204	1	1	1	0.031	0.039	0.035
GA-M	29	28	4059	779	0.064	9.394	1.815	1	1	1	0.032	0.04	0.035
GA-H	27	39	24083	5331	0.090	55.619	12.272	8	796	163	0.048	4.886	1.024
Avg. Gen+sum time		38.50	9810.50	2123.08	1.047	260.983	56.639	3.17	154.67	32.08	0.408	12.441	2.708

4.3 Hybrid Algorithm of GA and Ant Colony Optimization

The results showed in Table 3 are got by using a hybrid algorithm of Genetic algorithms and Ant colony optimization algorithm. The columns are same as in Table 1.

The results show that the average of the generations needed is less than with the GA or CA with every Sudoku tried. On the other hand, the average time spent is longer than with GA or CA, but it is not a great difference. GA/ACO hybrid uses more time per one trial than GA or CA. This might still be considered the best method without pre-processing.

The results showed in Table 3 (right) are using first the pre-processing 5 times and after that, we use a hybrid algorithm of Genetic Algorithms and Ant colony optimization algorithm. The columns used are the same as the Table 1.

Now, with pre-processing, we also obtain smaller average of the generations than with pre-processed GA or CA, but again we obtain longer average of the time spent than those algorithms for the Sudoku's that we consider as Hard level. For the rest of the Sudoku's (Easy and Medium level), we need the same number of generations but more time.

Table 3. Results of GA/ACO without and with pre-processing.

Difficulty		Genetic algorithm/Ant colony hybrid						GA/ACO hybrid with pre-processing					
rating	Givens	Min. gene	Max. gene	Avg. gene	Min. time	Max. time	Avg. Time	Min. gene	Max. gen	Avg. gene	Min. time	Max. time	Avg. Time
1 star	33	10	50	22	0.043	0.309	0.082	1	1	1	0.04	0.061	0.044
2stars	30	28	2199	199	0.098	7.531	0.794	1	1	1	0.037	0.051	0.043
3 stars	28	37	3280	347	0.135	12.458	1.235	1	1	1	0.038	0.061	0.046
4 stars	28	38	6438	905	0.132	22.925	3.391	6	229	71	0.027	1.431	0.399
5 stars	30	42	6571	1515	0.141	22.636	5.345	1	11	4	0.009	0.103	0.036
Easy	36	9	31	17	0.032	0.165	0.059	1	1	1	0.038	0.05	0.043
Challengi	25	41	3720	581	0.153	11.617	1.939	1	1	1	0.037	0.046	0.042
Difficult	23	77	17673	5104	0.269	67.874	13.040	6	221	35	0.043	3.877	0.534
Super dif	22	123	24199	6867	0.394	85.914	15.529	10	375	52	0.056	3.905	0.564
GA-E	32	15	338	65	0.055	1.353	0.343	1	1	1	0.055	0.13	0.082
GA-M	29	34	3865	606	0.128	17.392	2.240	1	1	1	0.05	0.108	0.071
GA-H	27	51	15073	4136	0.172	69.384	16.755	7	554	91	0.05	4.88	0.887
Avg. Gen+sum time		42.08	6953.08	1697.00	1.752	319.558	60.752	3.08	116.42	21.67	0.480	14.703	2.791

In the end of each Tables 1–3, we have calculated the average of the generations and the sum of the time spent to solve the different Sudoku's. According to these results, we can say that the best algorithm (without using the pre-processing) is GA/ACO hybrid algorithm because this algorithm need fewer generations to solve every level of difficulty but this algorithm has a problem. This problem is that it needs more time to solve the Sudoku but it is not a great difference.

The next best algorithm is GA because needs less generations to solve every level of difficulty and it needs less time also in every level of difficulty except in the Hard level, although the difference between GA and CA is not as high as the difference between GA/ACO and GA. Therefore, the worst algorithm studied was CA.

About the pre-processing, there are not differences between solving Sudoku's with GA, CA or GA/ACO in Easy or Medium level. For Hard level, GA/ACO is the best algorithm with slightly more time than GA but less time spent than CA. GA uses the same generations as CA but it needs slightly less time (thanks to no belief space updates) to solve the Sudoku's. So, GA is a little bit better than CA when we use pre-processing.

5 Conclusions

We studied if Sudoku puzzles can be solved with a combinatorial Genetic Algorithms, Cultural Algorithms and a hybrid algorithm with Genetic Algorithms and Ant Colony optimization. The results show that all these methods can solve Sudoku puzzles, but with slightly different effectivity. We find out that the hybrid algorithm GA/ACO is the most efficient with every Sudoku's, even more efficient than GAs or CAs. In this study we for the first time applied pre-processing of Sudoku's by first applying the most well known Sudoku solving methods. When we apply the pre-processing we obtain really good results and get a really efficient algorithm for solving Sudoku's, where first the easy parts of Sudoku is solved and then heuristic evolutionary algorithm is applied to solve rest instead of brute force used e.g. in SudokuExplainer.

The other goal was to study if difficulty ratings given for Sudoku puzzles (human difficulty) are consistent with their difficulty for our algorithms. The answer to that question seems to be positive: those Sudoku's that have a higher difficulty rating proved more difficult also for our algorithms. This also means that our algorithms can be used for testing the difficulty of a new Sudoku puzzle: Easy, Medium and Hard depending of the algorithm we used for testing and the generations needed for solving Sudoku.

We are using the same algorithms as here with Sudoku for other combinatorial optimization purposes, e.g. evenly loading the vehicle etc. technical balancing problems. We found Sudoku solving as a good benchmark problem for creating the better and more efficient algorithm. With real world problems, we have found that evaluating and comparing our algorithms and their evolution versions sometimes more uncertain than with exactly defined combinatorial balancing problem as Sudoku.

References

1. Mantere, T., Koljonen, J.: Solving and rating Sudoku puzzles with genetic algorithms. In: Hyvönen, E., et al. (eds.) Proceedings of the 12th Finnish Artificial Conference STeP 2006, Espoo, Finland, 26–27 October, pp. 86–92 (2006)
2. Wikipedia: Sudoku. http://en.wikipedia.org/wiki/Sudoku
3. Holland, J.: Adaptation in Natural and Artificial Systems. The MIT Press, Cambridge (1992)
4. 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)
5. Reynolds, R.G.: An overview of cultural algorithms. In: Advances in Evolutionary Computation, McGraw Hill Press, New York (1999)
6. Colorni, A., Dorigo M., Maniezzo, V.: Distributed optimization by ant colonies. In: actes de la première conférence européenne sur la vie artificielle, pp. 134–142. Elsevier, Paris (1991)
7. Lozano, M., García-Martínez, C.: Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: overview and progress report. Comput. Oper. Res. **37**(3), 481–497 (2010)
8. Blum, C., Puchinger, J., Raidl, G., Roli, A.: Hybrid metaheuristics in combinatorial optimization: a survey. Appl. Soft Comput. **11**(6), 4135–4151 (2011)
9. Wikipedia: Latin square. http://en.wikipedia.org/wiki/Latin_square
10. Wikipedia: Magic square. http://en.wikipedia.org/wiki/Magic_square

11. McGuire, G., Tugemann, B., Civario, G.: There is no 16-clue Sudoku: solving the Sudoku minimum number of clues problem via hitting set enumeration. Exp. Math. **23**(2), 190–217 (2014)
12. Lawler, E.L., Lentra, J.K., Rinnooy, A.H.G., Shmoys, D.B. (eds.): The Traveling Salesman Problem – A Guided Tour of Combinatorial Optimization. Wiley, New York (1985)
13. Saha, S., Rajeev Kumar, R.: Unifying heuristics and evolutionary computing for solving and rating Sudoku puzzles. Communicated 0 (2013) AQ2
14. Sudokuwiki: Sudoku. http://www.sudokuwiki.org/sudoku.htm
15. Mantere, T., Koljonen J.: Ant colony optimization and a hybrid genetic algorithm for Sudoku solving. In: MENDEL 2009 – 15th International Conference on Soft Computing, Brno, Czech Republic, 24–26 June, pp. 41–48 (2009)
16. Mantere T.: Improved ant colony genetic algorithm hybrid for Sudoku solving. In: Proceedings of the 2013 3rd World Congress on Information and Communication Technologies (WICT 2013), Hanoi, Vietnam, 15–18 December, pp. 276–281 (2013)
17. Mantere, T., Koljonen, J.: Sudoku page. http://lipas.uwasa.fi/~timan/sudoku/

Author Query Form

Book ID :	**462158_1_En**
Chapter No :	**1**

Springer
the language of science

Please ensure you fill out your response to the queries raised below and return this form along with your corrections.

Dear Author,
During the process of typesetting your chapter, the following queries have arisen. Please check your typeset proof carefully against the queries listed below and mark the necessary changes either directly on the proof/online grid or in the 'Author's response' area provided below

Query Refs.	Details Required	Author's Response
AQ1	This is to inform you that corresponding author has been identified as per the information available in the Copyright form.	
AQ2	Kindly provide complete details for Ref. [13].	

MARKED PROOF

Please correct and return this set

Please use the proof correction marks shown below for all alterations and corrections. If you wish to return your proof by fax you should ensure that all amendments are written clearly in dark ink and are made well within the page margins.

Instruction to printer	*Textual mark*	*Marginal mark*
Leave unchanged	• • • under matter to remain	Ⓥ
Insert in text the matter indicated in the margin	⋏	New matter followed by ⋏ or ⋏ⓧ2
Delete	/ through single character, rule or underline or ⊢—⊣ through all characters to be deleted	δ/ or δ/ⓧ2
Substitute character or substitute part of one or more word(s)	/ through letter or ⊢—⊣ through characters	new character / or new characters /
Change to italics	— under matter to be changed	⌣
Change to capitals	≡ under matter to be changed	≡
Change to small capitals	= under matter to be changed	=
Change to bold type	∼ under matter to be changed	∼
Change to bold italic	≂ under matter to be changed	≂
Change to lower case	Encircle matter to be changed	≢
Change italic to upright type	(As above)	⊥
Change bold to non-bold type	(As above)	⊥
Insert 'superior' character	/ through character or ⋏ where required	⋎ or ⋏ under character e.g. $\overset{2}{⋎}$ or $\overset{2}{⋏}$
Insert 'inferior' character	(As above)	⋏ over character e.g. $\underset{2}{⋏}$
Insert full stop	(As above)	⊙
Insert comma	(As above)	,
Insert single quotation marks	(As above)	$\overset{‘}{⋎}$ or $\overset{‘}{⋏}$ and/or $\overset{’}{⋎}$ or $\overset{’}{⋏}$
Insert double quotation marks	(As above)	$\overset{“}{⋎}$ or $\overset{“}{⋏}$ and/or $\overset{”}{⋎}$ or $\overset{”}{⋏}$
Insert hyphen	(As above)	⊢⊣
Start new paragraph	_⌐	_⌐
No new paragraph	↩	↩
Transpose	⊔⊓	⊔⊓
Close up	linking ⁀‿ characters	⁀‿
Insert or substitute space between characters or words	/ through character or ⋏ where required	Y
Reduce space between characters or words	| between characters or words affected	⊤