Solving and Analyzing Sudokus with Cultural Algorithms

Timo Mantere and Janne Koljonen

Abstract—**This paper studies how cultural algorithm suits to solving and analyzing Sudoku puzzles. Sudoku is a number puzzle that has recently become a worldwide phenomenon. It can be regarded as a combinatorial problem, but when solved with evolutionary algorithms it can also be handled as a constraint satisfaction or multi-objective optimization problem. The objectives of this study were 1) to test if a cultural algorithm with a belief space solves Sudoku puzzles more efficiently than a normal permutation genetic algorithm, 2) to see if the belief space gathers information that helps analyze the results and improve the method accordingly, 3) to improve our previous Sudoku solver presented in CEC2007. Experiments showed that proposed the cultural algorithm performed slightly better than the previous genetic algorithm based Sudoku solver.**

I. Introduction

THIS paper is a part in a series of studies, where Sudoku puzzles are solved and analyzed with evolutionary algorithms [1], [2]. Previously Sudokus were solved with genetic algorithms (GA) [3]. The main novelty of this study was to add a belief space to the previous GA based Sudoku solver and to produce a cultural algorithm (CA) [4].

We compare the solving efficiencies of CA and GA and analyze the solving mechanisms of the added cultural part. In order to evaluate if CA/GA can be used as Sudoku rating machine, we estimate how well the measured CA/GA efficiencies correlate with the alleged difficulty ratings given in the original publications of the test Sudoku puzzles. Yet another goal was to analyze the information gathered by the belief space to see if it can be employed to further develop the method. Furthermore, solving Sudokus with CA is expected to assist understanding of the capabilities of CA in constraint combinatorial problems.

As test cases, Sudoku puzzles published in newspapers [5], [6] and in www.sudoku.com [7] are used. In addition to the published ones, new Sudokus generated by a GA-based Sudoku maker [2] are also used.

Next in this section, we introduce the problem and related work. Section II introduces the proposed method, Section III the obtained results, and section IV discusses the findings and their implications.

A. Sudoku

When According to Wikipedia [8] Sudoku is a Japanese logical game that has recently become hugely popular in Europe and North-America. However, the first Sudoku puzzle was published in a puzzle magazine in USA as early as 1979, after which it circled through Japan, where it became popular in 1986. Finally, Sudokus became a phenomenon in the western world ca. 2005 [9]. Sudokus have been claimed to be highly popular and even addictive, because they are challenging but consist of relatively simple rules [10].

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) that cannot be changed or moved. However, the number of givens does not determine the difficulty of the puzzle [10], [11]. The rating of puzzles is one of the most difficult things in Sudoku puzzle creation, and there are 15 to 20 factors that are claimed to affect the difficulty rating [8].

The givens can be symmetric or nonsymmetric. In the symmetric case, all the givens are located symmetrically with respect to the centre position.

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

Fig. 1. An example of Sudoku puzzles, 32 positions contain a given number, the other positions should be solved.

Fig. 1 shows a Sudoku puzzle example. It was generated by our GA-based Sudoku maker [2]. Now the puzzle contains 32 nonsymmetric givens; the other 49 positions should be solved. This Sudoku is referred as GA-Medium b in the results of this paper. The Sudoku explainer [12] rates it with difficulty value 1.5.

The solution of this Sudoku is shown in fig. 2. Note that givens has remained in their original positions (comp fig. 1).

B. Related Work

The Sudoku problem seem to be rarely studied in the

Manuscript received December 31, 2007.

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

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

978-1-4244-1823-7/08/$25.00©2008 IEEE

technical sciences, since the IEEE Xplore search engine [13] finds only nine papers mentioning Sudoku. Nevertheless, the Sudoku problem could be used as a test bench for algorithms design. According Aaronson [14] it is based on NP complete problems. He also stated that the Sudoku craze may even end up in breakthroughs in computer science.

The Sudoku problem is studied in constraint programming and satisfiability research [15], [16], and [17]. Those methods are also efficient to solve Sudokus, but do not provide solution for every Sudoku puzzle.

There seems to be relatively few scientific papers about solving Sudoku with evolutionary algorithm (EA) methods. Moraglio et al. [11] 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 [7] efficiently, but has difficulties with the medium and hard Sudokus. They also stated that EAs are not the most efficient method when solving Sudokus, but that Sudoku is an interesting case study for algorithm development.

Nicolau and Ryan [18] have used quite a different approach to solve Sudokus: their Genetic Algorithms using Grammatical Evolution (GAuGE) optimizes the sequence of logical operations that are then applied to find the solution.

Gold [19] has used GA for generating new Sudoku puzzles. Software called SudokuMaker [20] is also available. It is claimed to use GA internally and 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. For more related work see refs. [1], [2].

II. The Proposed Method

An integer coded elitist GA is used. The size of the GA chromosome is 81 integers, divided into nine sub-blocks of nine numbers (building blocks) that correspond to the 3×3 subgrids, from left to right and from top to bottom. Uniform crossover operation is applied only between sub-blocks, and the sequences of swap mutations only inside the sub-blocks. Therefore the crossover point cannot be inside a 3×3 subgrid. All new individuals are generated by applying first crossover and then mutations to the crossover result.

A. GA Parameters and Implementation

The population size (`POP`) was 11, and the amount of elitism (`ELIT`) was one individual. These parameters were chosen with preliminary testing, to which we randomly selected five test Sudokus. First we solved those Sudokus with a population size 30 and changing the elitism between [0, 20]. Fig. 3 shows results, according to which the best amount of elitism with 4 out of 5 test Sudokus was 1. The easiest test Sudoku was solved most effectively without elitism. Number of trials, i.e. fitness function evaluations, needed to solve a Sudoku was used as measure of GA efficiency. Next we tested the same five test Sudokus with different population sizes between [2, 30], while elitism was 1. Fig. 4 shows the results, according to which the easiest Sudokus were solved efficiently even with a population size of 5, but with 4 out of 5 test Sudokus the optimal population size was between 7 and 10. Only with the most difficult Sudokus the best population size was 15. We decided to use population size 11.

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

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

We favored the best individuals as parents by selecting the mating individuals `p1` and `p2` using the following algorithm (Java code):

```
for(i=POP-1; i>=ELIT; i--){
  ii=ord[i];
  p1 = ord[i*Math.random()];                    (1)
  p2 = ord[i*Math.random()];
  crossover(indiv[ii], indiv[p1], indiv[p2]);
  mutation(indiv[ii]);}
```

The population is first sorted so that the best individual is in order vector (`ord[]`) index 0, and the worst is in index POP-1. The favoring is stronger than the linear favoring, but still gives even the worst individuals a small change to become a parent. Note that there is a chance of selecting p1=p2, when only the mutation operation changes the new individual genotype. Search was stopped (stopping condition) when a solution found, since our method never failed to solve the Sudoku.

In addition to basic rules of Sudoku, the givens must be observed during the solving process. Therefore a Sudoku solver must obey four different 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 3×3 subgrid must contain each integer from 1 to 9,
4) the given numbers must stay in the original positions.

By selecting an appropriate solving strategy, condition 4) is always fulfilled. Additionally, one of the conditions 1) to 3) can be controlled. Hence, only two conditions are subject to optimization. We chose to implement our EAs so that conditions 3) and 4) are intrinsically fulfilled and only the conditions 1) and 2) are optimized. Sudoku has 9 rows and 9 columns, so we have 18 equality constraints that must the solution must fulfill.

The EAs (GA and CA) used in this study were customized to this problem or other grid type combinatorial problems, and could not directly be used to other type of problems. The EAs did not use any direct mutations or crossovers that could generate illegal situations for 3×3 subgrids. However, the genetic operators allow rows and columns to contain integers

more than once in the non-optimal situation. The genetic operators were not allowed to move the givens. That was guaranteed by a help array, which indicates whether a number is a given or not. We also tested a version, where the givens were allowed to move during the solving process, provided that they were in correct position in the solution. That strategy was not efficient, so it was abandoned.

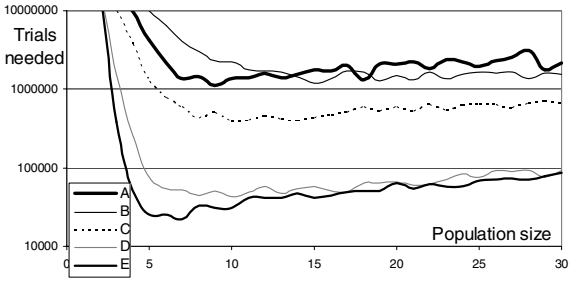

Fig. 3. Number of trials (log scale) needed to solve some test Sudokus as a function of elitism, when the population size was 30.

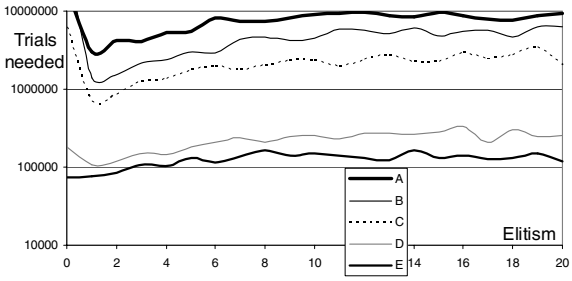

Fig. 4. Number of trials (log scale) needed to solve the same test Sudokus, as in fig. 3, as a function of population size, when the elitism was 1.

B. Genetic Operators and Heuristics

Mutations were applied only inside the 3×3 subgrid. Originally, we used three different mutation strategies [1], but this time only a sequence of 1 to 5 (the length drawn randomly) swap mutations inside a subgrid was applied.

The fact that many swap attempts are omitted (explained below) caused the real distribution of swap sequence lengths {1, 2, 3, 4, 5} to be {62.5, 30.4, 6.6, 0.5, 0.01} %.

The swap mutation probability was 0.1 for each gene location. This does not equal to the actual amount of mutation due to some problem specific issues: In swap mutation, the values of two positions are exchanged. Each time the mutation is tried inside a subgrid, the help array of givens is checked. If it is illegal to change the numbers in randomly chosen positions, the mutation operation is omitted; this decreases the mutation probability. But, we also tested if the new trial was identical with one of its parents. If so, the mutation operation was called again until different trial was obtained. This increased the mutation probability.

Therefore the actual probability of mutation could only be determined empirically during the solve run. We measured that 88.5% of new trials were affected by mutation; 11.5% were changed only by crossover. The probability of each gene location to experience mutation was 3.7%. Note, that the swap mutation always affects two gene locations.

We also implemented another heuristic to control whether mutation is applied or omitted. The inference behind this rule is that every time mutation is attempted, it affects one or two columns, and one or two rows, at total 3 or 4 row and column vectors (fig. 6). In an optimal swap situation, the two digits that are swapped should appear in these vectors zero times before the swap (or 2 times in the case of 3 vectors).

When the puzzle is solved, any randomly selected two digits should appear in any randomly selected two rows and two columns total of 4 times.

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

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

Fig. 5. Attempts to swap digits. Left: Swap is attempted between 4 and 1: the corresponding columns and rows are scanned to count the occurrence frequencies of these digits. Now the digits occurred four times (highlighted) -> swap omitted. Right, top corner: now 1 and 4 are in wrong positions. Now the corresponding columns and rows have no occurrence of these digits -> swap performed. Right, bottom corner: 4 and 9 are to be swapped. Three vectors involved are now scanned. The digits appeared four times in these vectors, since their current positions now also count -> swap omitted.

Henceforth, we decided to use another help array to indicate how many times each digit appears in each row and column, and when the swap mutation is attempted, it is allowed only if the digits to be swapped appear in the destination rows and columns 0 to 3 times. After the swap, they will appear 2 to 5 times. Recall that 4 is the optimum, but we give the system some 'slack', otherwise the condition is too tight and it is difficult to perform swaps. This rule prevents a large number of useless swaps that would lead to an excess of repeated digits in rows and columns. This rule takes some time to calculate, but the overall performance was enhanced. We also tested a tighter rule (<3), table 1, but then we needed a lot more trials in order to solve Sudoku. A looser rule (<5) was also tested, table 1, and with it the number of trials needed to solve Sudoku is significantly higher. Note that the difficulty order with these is not the same as with the normal rule; we found that Sudokus perform differently with different setups.

Fig. 6 shows the distribution of how many times the digits of attempted swap positions appeared in the destination row and column vectors, in logarithmic scale. In most cases (85%), these digits already appear four times (optimum). If these digits appear five times or more (7.8% of cases) on these vectors, there is too many of these digits. If these digits appear three times or less, it indicates that these digits are not

optimized in these vectors yet, and the proposed swap is reasonable. Only 7.2% of the swaps were allowed.

A special genetic operator that was applied was population reinitialization or cataclysmic mutation [21]. In combinatorial problems, the optimization process often gets stuck, and it is more efficient to restart with a new initial population than to continue from the stuck point. Reinitialization also means that a small population size and elitism can be used, since even the inescapable stuck points can be handled.

TABLE 1
COMPARISON OF HOW EFFECTIVELY CA WITH DIFFERENT SETUPS FOUND SOLUTIONS FOR SOME SUDOKU PUZZLES

Sudoku	*Our version*	*Without aging*	*Tighter <3 limit*	*Looser <5 limit*
E*b*	271	290	1315	1280570
E*c*	549	702	2173	3209492
E*a*	553	850	3739	4449784
1*b*	665	1070	6318	5703074
1*a*	1141	3461	24170	*Too many*
GA-E*c*	3057	7700	13582	11033764
3*a*	17123	179608	5826266	*Too many*
2*b*	33116	781994	1694321	*Too many*
5*a*	62813	950079	*Too many*	*Too many*
SD*a*	422542	5969736	*Too many*	*Too many*

The average trials needed to solve Sudoku with different CA setups. The versions we used in this paper, a version without the aging rule (see chapter 2*C* for details), and versions with tighter and looser swap allow limits (with aging). Sudoku names as in table 2.

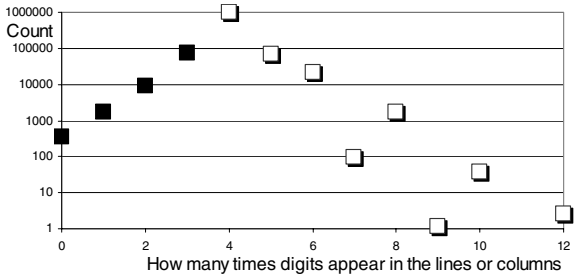

Fig. 6. Distribution (log scale) of how many times the digits attempted to be swapped in the swap mutation already appeared in the line and column vectors where they are supposed to be relocated. The swap is allowed only if they appear three times or less (7.2%, ■); most commonly they appear four times (85%). If they appear four or more times, the attempted swap mutation is not performed (92.8%, □).

We did preliminary tests which showed that with all Sudokus a near-optimum solution (max. 4 positions wrong) is reached always without reinitialization and usually with less than 2000 trials. Solutions with max. 2 positions wrong were not always reached without reinitialization. With both difficult and easy Sudokus, a near-optimum solution was found relatively fast. The crucial difference between the easier and the difficult Sudokus is the fact that the difficult Sudokus do not advance from the near-optimum solutions.

When Sudoku solving does advance from the near-optimal solution to the optimum, it does it quite fast. For instance, if a near-optimal solution with 4 (2) positions wrong was reached in T trials, the optimum was reached in 4T (4T) trials after that with more than 50% of cases.

The difficult Sudokus seem to have a lot of different near-optimal solutions. Backtracking or branch and bound might be efficient, but since EA reaches Sudoku near-optimal solutions quite fast, we decided to use reinitialization, after which the solution probably approached from different path, thus avoiding the same stuck point.

We used a self-adapting reinitialization rule, which was triggered when the best solution first time reaches fitness value 4 (4 numbers in the wrong position).

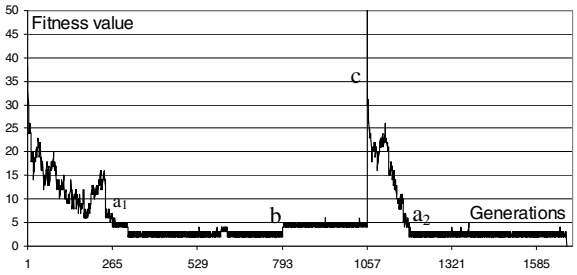

Fig. 7. The development of fitness value as a function of generations. Note that in generation 266 the fitness value reached 4 (point a) for first time, which triggers the counter. In point b (=$3a_1$), we add penalty for those Sudoku solutions where the same digit as given appears in the vector. In point c (=$4a_1$), we reinitialize the population with the help of the belief space. The next time that the solution reaches the fitness value 4 (a_2) we set the possible penalty increase point as $b_2=a_2+2(a_1+a_2)/2$ and the next possible reinitialization point as $c_2=a_2+3(a_1+a_2)/2$. However, in this fig. we find the solution before b_2 or c_2 are needed.

Fig. 7 shows how this reinitialization rule works. When the best fitness value reaches 4 for the first time (point a_1) we start to count generations, and in the point b_1 (which is $3a_1$) we add penalty for those Sudokus, where the same digit as a given appears in the same vector (row or column). In the point c_1 (which is $4a_1$), we reinitialize the population. The general formulas for these points are: $b_n=a_n+2(\sum a_i)/n$, $c_n=a_n+3(\sum a_i)/n$, where n is the number of reinitialization period, start from 1.

These points were selected after exhaustively testing and analyzing the Sudoku solving behavior. The reason why we do not penalize all the time if the same digit as some given appears in the same vector is that a near-optimal solution is found much faster this way. Moreover, if we penalize from the beginning, search may get stuck before even reaching a near-optimal solution.

However, when we have reached the near-optimal solution, some Sudokus are solved faster, if this penalty is forced, therefore it is reasonable to force it at some point. The added penalty is removed after the reinitialization. The reinitialization in this case is not a complete restart of the solving run, because it is guided by the cultural belief space. There exists a link between earlier solutions and the reinitialized population via the belief space (see chapter 2D).

C. Fitness Function

Designing a fitness function that would aid the GA search

is often difficult in combinatorial problems [22]. In this case, the conditions that every 3×3 subgrid contains integers from 1 to 9 and the fixed numbers was guaranteed intrinsically and penalty functions are used to force the other conditions.

Originally, we used a somewhat complex fitness function [1] that penalized different constraint violations differently. In this study the fitness function is derived from the set theory. It requires that each vector, row, x_i, and column, x_j, is considered as a set that must be equal to the set A, which contains integers from 1 to 9:

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

The The functions (2) 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 every row and column sets, and fitness function value becomes zero. Otherwise the penalty value corresponding to the count of missing digits in each row or vector set is aggregated to the fitness value. This means that if some digit is missing from one row or column a penalty value 1 is added, if two digits are missing, penalty value two is added, and so on.

After point cn, an extra penalty of two is added, if a same digit as some given appears in the same vector, that is added at is two. In solution search, fitness function is minimized.

Aging was included in the fitness function. A penalty term (3) is added to the best individual's fitness value, if the same individual was the best one also in the previous generation:

```
if (Best[generation[i]] == Best[generation[i-1]])
        Value[Best]+=1;                          (3)
```

We studied the effect of aging with some randomly selected Sudokus. The average numbers of trials needed with and without aging are presented in table 1. With easy Sudokus the performance was almost the same regardless of aging, but the more difficult the Sudokus were, the more beneficial the aging was. This operation was added in order to generate more variation to the population, and our tests also showed that it increases the solving speed significantly.

D. Cultural Genetic Algorithm with a Belief Space

The main difference of this paper to [2] is that this time we added a belief space model to our GA. The belief space in this case was simple: it was a 9×9×9 cube (fig. 8), 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

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 1 in the belief space. This model also means that the belief space is updated only with near-optimal solutions.

The belief space gathers information from near-optimal solutions (only 2 positions wrong). This information is used only in the population reinitialization 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, *i.e.* those positions that always contain the same digit in the near-optimal solutions.

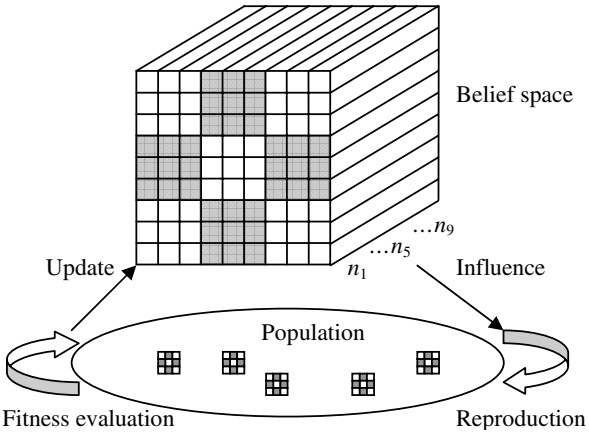

Fig. 8. The cultural algorithm model used in this study. When the fitness function is evaluated, the best individual updates the belief space. The belief space also has an influence on the population reproduction process.

This also connects the reinitialized population to the previous generations, since the new initial population is not formed freely but according to the information learned in the whole solve run until then. The information gathered by the belief space was also used when analyzing the results and explaining of why some Sudokus are easy and some difficult.

III. Experimental Results

We tested the proposed EA methods (GA and CA) by solving 45 different benchmark Sudokus.

15 Sudokus were taken from newspaper Helsingin Sanomat [5] labeled with difficulty ratings 1-5 `stars`. These had 28 to 33 symmetric givens (table 4). We also tested 12 Sudokus taken from newspaper Aamulehti [6]. They were labeled with difficulty ratings: `Easy`, `Challenging`, `Difficult`, and `Super difficult`. They contained 23 to 36 nonsymmetrical givens.

We also have 9 Sudokus taken from Pappocom [7] labeled as `Easy`, `Medium` and `Hard`, and 9 Sudokus we generated with GA-based Sudoku maker [2]. The latter ones were evaluated by a GA-based Sudoku solver [2] and rated as `GA-Easy`, `GA-Medium` and `GA-Hard`.

We used two parameter setups: unlimited number of trials (chapter A) and max. 100 000 trials (chapter *B*).

A. Sudoku Solving

We started by solving the whole set of 45 benchmark Sudokus 100 times with both genetic algorithm and cultural

algorithm. We repeated the operation two more times in order to see if the results are consistent.

In these three solve runs, GA and CA needed on average {5560474, 5696622, 5785018} and {5146725, 5189095, 5227965} trials, respectively. As can be seen, there was some variation between the solve runs, but CA was every time better than the best GA run; the results seem consistent. As for the grant mean, CA was 8.7% more efficient than GA.

TABLE 2
COMPARISON OF HOW EFFECTIVELY GA AND CA (BETTER **BOLD**) FIND SOLUTIONS FOR THE SUDOKU PUZZLES WITH DIFFERENT DIFFICULTY

Difficulty Rating	Average amount of trials needed to solve with GA			Average amount of trials needed to Solve with CA			Improve by
	a	*b*	*c*	*a*	*b*	*c*	%
1	1264	**634**	4787	**1141**	665	**4732**	2.19
2	8765	**32618**	**12828**	**8187**	33116	14515	-2.97
3	17841	70214	**35450**	**17123**	**65068**	42332	-0.82
4	**47057**	70994	71539	50083	**67691**	**68229**	1.89
5	66813	**49802**	**101691**	**62813**	54625	114180	-6.10
E	**535**	**252**	600	553	271	**549**	1.05
C	**24656**	**80486**	50406	26330	84761	**41034**	2.20
D	281519	90496	**66810**	**250518**	**83608**	71503	7.56
SD	**413450**	241184	218102	422542	**222883**	**207893**	-0.22
Easy	11261	2976	**3340**	**11109**	**2800**	3520	0.84
Med	66183	**191627**	**53365**	**63676**	199871	53806	-1.99
Hard	1419023	90883	627091	**1232282**	**81677**	**530257**	13.70
GA-E	4128	**4523**	3100	**4065**	4596	**3057**	9.31
GA-M	36735	19186	32651	**33808**	**17242**	**29536**	9.02
GA-H	**163636**	**104389**	785814	193622	104655	**601404**	14.63
Sum		5680705			**5187928**		8.67

There are three different Sudokus (a, b, and c) from each of the 15 difficulty classes 1-5 stars [5], Easy, Challenging, Difficult and Super Difficult [6], Easy, Medium, Hard [7] and GA-Easy, GA-Medium, GA-Hard [2]. Each Sudoku was solved 300 times and the table shows mean values of how many trials was needed on average. The mean improvement percentages of CA against the GA with each difficulty class are also given.

Table 2 summarizes the average results with GA and CA. These results are combined from all 300 solve times of each Sudoku. Table 2 shows that CA was more efficient only with 24 test Sudokus out of 45. However, if we divide them to difficulty rating classes, CA was more efficient in 11 rating classes out of 15 tested. Most importantly, CA is usually more efficient with the most difficult Sudokus.

There exists a logical explanation for CA performing better with difficult Sudokus. The belief space gathers and uses information from near-optimal solutions quite slowly. If the Sudoku is very easy the belief space information is not used at all. The longer the solve run lasts, the more information is collected, and it is also more likely that the information also becomes more relevant and accurate.

Note that the CA improvement over GA is not very large, and with most Sudokus a T-test would support the null-hypothesis that the means are equal. However, since we have an improvement with most of the test Sudokus and difficulty classes, and this improvement seems to be consistent, we can claim that at least CA is not less efficient. With a more effective use of a better belief space it would probably be possible to increase the efficiency further. The actual mean processor time for the 45 Sudokus was 6.1 seconds (CA written in Java, using 3 GHz Pentium4). When the same benchmark Sudokus as in ref. [2] were used, the mean processor time was 3.9 sec. (compare to 4.1 sec. in [2]).

B. Comparison to Other Studies

The study in ref. [11] presents results with a large number of different methods, totally 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 presented in [11]. 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 that 100,000 trials were used. For the sake of comparison, we tested five Sudokus from [7] with our GA to obtain comparable results. We cannot be totally sure, if the Sudokus are exactly the same as they used, but we chose the first three from the Easy category and the first one from Medium and Hard categories, exactly as they described that they selected them.

Table 3 shows our results with unlimited trials and with 100,000 trials. The latter method should be compared to hill climbers, where we that number of trials was used [11]. Both GA setups (unlimited and 100,000 trials) performed better than the best GA in [11]. Only with Hard our 100,000 trials GA run performed worse than their "Swap space" method.

TABLE 3
OUR RESULTS AND THE BEST RESULTS REPRESENTED IN [11]

Sudoku problems from [7]	Our CA		The best results represented in [11]		
	Unlimited trials	100 000 Trials	Hamming Space	Swap Space	Hill Climbers
Easy 1	30	30	5	28	30
Easy 2	30	30	8	21	30
Easy 3	30	30	14	30	30
Medium	30	22	0	0	0
Hard	30	2	0	15	0
Total	150	114	27	94	90

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

With our unlimited trials GA setup the longest solve run, with the Hard Sudoku, lasted 5,630,221 trials.

Nicolau and Ryan [18] have also presented very good result by GAuGE system. They had taken their benchmark Sudokus from a book that was unavailable for us. Thus, we cannot directly compare our results with their method. For 17 out of 20 benchmark Sudokus, they find the solution every time in 30 test runs, but for two problems their method was unsuccessful to find the solution within 320 000 trials.

Our method has never failed to find a solution of Sudoku. We tested our method to AI Escargot [23], a Sudoku that is claimed to the most difficult Sudoku. Without any trial

limitation it was solved every time, but when using a limit of 100,000 (320,000) trials it was solved only 5 (16) times out of 100 test runs. In the fastest solve run, it was solved with 16,271 trials, on average it required 1,542,376 trials, and the longest solve run required 10,394,690 trials.

C. Analyzing Sudokus with Belief Space

We try to analyze, if the Sudoku difficulty and the CA improvement over GA can be explained with some Sudoku properties. Table 4 shows some properties of the benchmark Sudoku set: the number of givens of each Sudoku, the minimum number of hidden givens, and maximum number of different near-optimal solutions found (2 positions wrong).

The final two properties (hidden givens and near solutions) are collected from the belief space. In the analysis, we also use the collected other information, like average number of hidden givens and average number of near solutions found in the test runs, and also count all different near-optimal solutions and hidden givens over all test runs.

Table 4 shows that, while the number of givens does not explain the difficulty of the Sudoku, there seems to be less givens in the difficult Sudokus. Information collected by the belief space seems to imply that there are more near-optimal solutions and less hidden givens in the difficult Sudokus than in the easy Sudokus.

TABLE 4
SUDOKU PROPERTIES KNOWN OR COLLECTED BY BELIEF SPACE

Diff	*Givens (G)*			*Hidden givens(H)*			*Near solutions(N)*		
rating.	*a*	*b*	*c*	*a*	*b*	*c*	*a*	*b*	*c*
1	33	36	32	34	34	15	41	16	51
2	30	28	28	27	11	16	46	58	64
3	28	26	27	17	14	7	89	116	107
4	28	27	28	9	11	* 7	88	119	123
5	30	28	26	11	3	8	126	118	234
E	36	39	36	32	35	33	21	8	8
C	25	25	25	19	10	11	99	126	122
D	22	23	22	* 5	11	18	319	118	89
SD	23	22	22	6	* 13	11	140	249	147
Easy	31	31	32	20	32	21	36	50	50
Med	28	26	28	11	8	19	118	140	174
Hard	22	26	23	* 2	6	* 5	263	107	198
GA-E	32	33	37	12	17	19	62	35	38
GA-M	29	32	31	8	11	10	63	104	96
GA-H	27	27	24	10	10	7	174	136	148

G presents how many givens each Sudoku instance has. *H* is minimum number of hidden, and *N* the maximum number of near-optimal solutions collected by the belief space in 100 solves runs. Those marked with * actually posses zero hidden givens, when analyzed of all 100 solve runs.

In order to see if these findings explain the Sudoku solving efficiency we performed a correlation analysis between several Sudoku properties. Table 5 shows linear correlations between some Sudoku properties, GA and CA results, and CA improvement over GA. It also shows that correlation between the CA and GA results is almost 1; GA and CA perform almost identically with the same Sudoku.

A notable thing is that all properties have a little bit higher correlation with CA results than with GA results. This might indicate that CA have learned and employed some problem properties to achieve better results. The improvement of CA over GA does not have high correlation with any Sudoku properties; this means that none of the properties solely explains why CA performs better. Nevertheless, almost every property has equals signs of correlation with GA, CA, and the improvement. This might imply that all properties are slightly employed by CA and together they establish the overall improvement.

TABLE 5
CORRELATIONS BETWEEN SUDOKU PROPERTIES, GA AND CA SOLVING EFFICIENCIES, AND IMPROVEMENT OF CA OVER GA

	GA	*CA*	*Improvement*
CA	0.996		0.391
G	-0.512	-0.533	-0.168
N	0.462	0.486	0.111
N_{all}	0.601	0.624	0.187
M_{avg}	0.501	0.516	0.109
H	-0.439	-0.457	-0.099
H_{all}	-0.382	-0.410	-0.115
H_{avg}	-0.205	-0.224	0.014
Ha_{min}	-0.420	-0.438	-0.103
Ha_{all}	-0.371	-0.397	-0.117
Ha_{avg}	-0.410	-0.435	-0.085

G, N and H as in table 4. N_{all} is all different found in 100 solve runs, N_{avg} is average found in 100 solve runs. *Ha* is the number of hidden givens adjusted with the number of free locations, *i.e.* $Ha=H/(81-G)$. H_{all} and H_{avg} formed similarly as with *N*.

The highest correlation is found between CA results and the overall number of the near-optimal solutions that a Sudoku instance possesses. The number of near-optimal solutions is the most important factor to define Sudoku puzzle difficulty. This was employed by the CA as intended.

However, the overall number of near-optimal solutions (N_{all}) is unknown during a Sudoku solve run. Thus it cannot be used. N_{all} is estimated after the series of 100 solve runs.

The second highest correlation is between the numbers of givens and solving efficiency. This means that although the number of givens does not implicitly define the difficulty of the Sudoku, it does have a large influence.

The amount of hidden givens does not have high correlation with the results. Hidden givens are also utilized by our CA. The number of hidden givens adjusted with the number of free locations $Ha=H/(81-G)$ in the Sudoku does not explain results better than unadjusted, except with the average numbers (H_{avg} vs. Ha_{avg}) the correlation is almost twice as high. Some Sudokus were found to posses zero hidden givens (see table 4) when looking the data collected over 100 solve runs (H_{all}). These Sudokus are quite difficult and they possess so many near-optimal solutions (2 positions wrong) that all free positions can have different values in some of the near-optimal solutions.

IV. CONCLUSIONS AND FUTURE

In this paper, we studied if Sudoku puzzles can be solved with a combinatorial cultural algorithm, which is a genetic algorithm with a belief space. The results show that EAs can solve Sudoku puzzles relatively effectively, and CA is slightly more efficient than GA. Our results stand well the comparison with other known results with EAs.

However, there exist some more efficient algorithms to solve Sudoku puzzles e.g. [15], [16] and [17], but in the results reported, all these methods fail to solve some puzzles.

In this study, the goal was to test how efficient EA approach is, without too many problem specific rules. Solving the easiest Sudokus could of course be boosted by adding problem specific knowledge. In this study, the belief space was added in order to learn this kind of problem specific knowledge in evolutionary algorithm without any a priory knowledge or logic. If too much problem specific logic is provided to the Sudoku solving, there will be nothing much left to optimize, since Sudoku is a logical puzzle.

The belief space model that was used in this cultural algorithm experiment was quite simple and can possibly be improved in future. It is also likely that the gathered information could be exercised more efficiently. The analysis of the results implies that we should shift the emphasis more to the using of near-optimal solutions than using hidden givens information. Now our method uses both, but it is cannot be exactly estimated in what ratio. In future, we will study if it is possible to generate a fitness function based on energy functions [22]. The CA might also be improved by some kind of energy function based belief space. Bauke [24] have just proposed an interesting approach to solve Sudokus with message-passing algorithm and belief propagation. We might try to apply ideas of his method to our CA.

When some belief spaces were studied manually, it looked like Sudoku puzzles might possess some kind of positional bias: Most of the belief spaces looked like the trials composed based on them would more likely contain small numbers in the left upper corner and larger numbers in right bottom corner. We think that it is possible that Sudoku generators have some kind of positional bias when they generate new Sudoku puzzles. CA belief space could potentially exploit this bias in order to generate better results. We plan to measure the possible positional biases in future and see, if it really appears or not, and if it appears only with some Sudoku generators.

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. For some solitary puzzles the rating seems to be 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 to rate the difficulty of a new Sudoku puzzle. Rating of puzzles is said to be one of the most difficult tasks in Sudoku creation [8]. Hence, 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 the difficulty same way.

The lack of common benchmark Sudokus complicates the comparison of results; everyone has collected their test cases from different sources. We decided to put our 45 test Sudokus available in the web [25]. Hence, anyone interested to compare their results with ours can now use the same benchmark puzzles.

REFERENCES

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

[2] T. Mantere, J. Koljonen, "Solving, Rating and Generating Sudoku Puzzles with GA," in *2007 IEEE Congress on Evolutionary computation – CEC2007*, Singapore, 2007, pp. 1382-1389.

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

[4] R.G. Reynolds, "An overview of cultural algorithms," in *Advances in Evolutionary Computation*, McGraw Hill Press, 1999.

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

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

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

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

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

[10] I. Semeniuk, "Stuck on you.," *NewScientist* **24/31**, 2005, pp. 45-47.

[11] A. Moraglio, J. Togelius, S. Lucas, "Product geometric crossover for the sudoku puzzle," in *2006 IEEE Congress on Evolutionary Comp. (CEC2006)*, Vancouver, BC, Canada, July 16-21, 2006, pp. 470-476.

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

[13] *IEEE Xplore*, Available via WWW: http://ieeexplore.ieee.org/search/advsearch.jsp (cited 31.12.2007)

[14] L. Aaronson, "*Sudoku science,"* IEEE Spectrum **43**(2), 2006, 16-17.

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

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

[17] K. Moon, J. Gunther, "Multiple constrain satisfaction by belief propagation: An example using Sudoku," in *2006 IEEE Mountain Workshop on Adaptive and Learning Systems*, 2006, pp. 122–126.

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

[19] M. Gold, *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).

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

[21] L. Eshelman, "The CHC adaptive search algorithms: how to safe search when engaging in nontraditional genetic recombination," in *Foundations of Genetic Algorithms*, Morgan Kaufmann, 1991.

[22] J. Koljonen, J. Alander, "Solving the "urban horse" problem by backtracking and genetic algorithm – a comparison," in *Step 2004 – The 11th Finnish Artificial Intelligence Conference*, Vol. 3, Vantaa, 1-3 Sept. 2004, pp. 127-13.

[23] A. Inkala, *AI Sudoku 1002 Vaikeaa Tehtävää*, Pressmen Oy, 2006.

[24] H. Bauke, "Passing messages to lonely numbers," in *Computing in Science & Engineering* **10**(2), March/April 2008, pp. 32-40.

[25] T. Mantere, J., Koljonen, *Sudoku project homepage*, Available via WWW: http://www.uwasa.fi/~timan/sudoku/ (cited 14.3. 2008).