Solving and Rating Sudoku Puzzles with Genetic Algorithms

Timo Mantere and Janne Koljonen
Department of Electrical Engineering and Automation
University of Vaasa
FIN-65101 Vaasa
firstname.lastname@uwasa.fi

Abstract

This paper discusses solving and generating Sudoku puzzles with evolutionary algorithms. Sudoku is a Japanese number puzzle game that has become a worldwide phenomenon. As an optimization problem Sudoku belongs to the group of combinatorial problems, but it is also a constraint satisfaction problem. The objective of this paper is to test if genetic algorithm optimization is an efficient method for solving Sudoku puzzles and to generate new puzzles. Another goal is to find out if the puzzles, that are difficult for human solver, are also difficult for the genetic algorithms. In that case it would offer an opportunity to use genetic algorithm solver to test the difficulty levels of new Sudoku puzzles, *i.e.* to use it as a rating machine.

1 Introduction

This paper studies if the Sudoku puzzles can be optimized effectively with genetic algorithms. Genetic algorithm (GA) is an optimization method that mimes the Darwinian evolution that happens in nature.

According to Wikipedia (2006) Sudoku is a Japanese logical game that has recently become hugely popular in Europe and North-America. However, the first puzzle seems to be created in USA 1979, but it circled through Japan and reappeared to west recently. The huge popularity and addictivity of Sudoku have been claimed to be because it is challenging, but have very simple rules (Semeniuk, 2005).

Sudoku puzzle is composed of a 9×9 square that are divided into nine 3×3 sub squares. The solution of Sudoku puzzle is such that each row, column and sub square contains each integer number from [1, 9] once and only once.

In the beginning there are some static numbers (givens) in the puzzle that are given according to the difficulty rating. Figure 1 shows the original situation of the Sudoku puzzle where 30 numbers for 81 possible positions are given. The number of givens does not determine the difficulty of the puzzle (Semeniuk, 2005). 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 (Wikipedia, 2006).

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

Figure 1. A starting point of the Sudoku puzzle, where 30 locations contains a static number that are given.

Figure 2 shows the solution for the puzzle in fig. 1. Note that each column, row and sub square of the solution contains each integer from 1 to 9 once. The static numbers given in the beginning (fig. 1) are in their original positions.

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

Figure 2. A solution for the Sudoku puzzle given in figure 1.

The objective of this study is to test if genetic algorithm is an efficient method for solving Sudoku puzzles, and also if it can be used to create generate Sudoku puzzles and test their difficulty levels. If the difficulty rating of the Sudoku puzzles in the newspapers is consistent with their difficulty for GA optimization, the GA solver can also be used as a rating machine for Sudokus.

1.1 Genetic Algorithms

All Genetic algorithms (Holland, 1982) are computer based optimization methods that use the Darwinian evolution (Darwin, 1859) of nature as a model and inspiration. The solution base of our problem is encoded as individuals that are chromosomes consisting of several genes. On the contrary to the nature, in GAs the individual (phenotype) is usually deterministically derived from the chromosome (genotype). These individuals are tested against our problem represented as a fitness function. The better the fitness value an individual gets, the better the chance to be selected to 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 genes for a new chromosome are selected from two parents using some preselected practice, e.g. one-point, two-point or uniform crossover. In mutation, random genes of the chromosome are mutated either randomly or using some predefined strategy. The GA strategy is elitist and follows the "survival of the fittest" principles of Darwinian evolution.

1.2 Related work

Sudoku is a combinatorial optimization problem (Lawler *et al*, 1985), 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 column and row of the solution are equal. Therefore, as a problem Sudoku is obviously related to the ancient magic square problem (Latin square), where different size of squares must be filled so, that the sum of each column and row are equal. The magic square problem has been solved by GAs (Alander *et al*, 1999, Ardel, 1994) and also the Evonet Flying circus (Evonet, 1996) has a demonstration of a magic square solver.

Another related problem is a generating threshold matrix for halftoning (Kang, 1999) grayscale images. 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 at the threshold matrix. The values in the threshold matrix should be evenly distributed. Therefore the sums of the threshold values in each row and column of the threshold matrix should be nearly equal. Furthermore, the demand of homogeneity holds also locally, *i.e.* any fixed size sub area should have a nearly evenly distributed threshold value sum. This guarantees that the resulting image does not contain large clusters of black or white pixels. Threshold matrices have been previously optimized by GAs *e.g.* in (Alander *et al*, 1998 and 1999; Kobayashi and Saito, 1993; Newbern and Bowe, 1997; Wiley, 1998).

There seems to be no scientific papers on Sudoku in the research indices, but there are some white papers on the Internet. In (Gold, 2005) a GA is used to generate new Sudoku puzzles, but the method seems inefficient, since in their example the GA needed 35700 generations to come up with a new puzzle. In our results, we created a new Sudoku, in average, in 1390 generations.

There is also a 'Sudoku Maker' (2006) software available that is said to use genetic algorithm internally. It is 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 puzzle is generated.

Sudoku can also be seen as constrain 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 (Li *et al*, 2005; Mantere, 2005; Runarsson and Yao, 2000).

2 The proposed method

Sudoku is a combinatorial optimization problem, where each row, column and also each nine 3×3 sub square must have a number from {1, 2, …, 9} exactly once. Therefore we need to use a GA that is designated for combinatorial optimization problems. That means that it will not use direct mutations or crossovers that could generate illegal situations: 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).

Consequently, we need to represent the Sudoku puzzles in GA program so that the givens will be static and cannot be moved or changed in genetic operations. For this purpose we have an auxiliary array containing the givens. We decided to present Sudoku puzzle solution trials as an integer array of 81 numbers. The array is divided to nine sub blocks (building blocks) of nine numbers corresponding to the 3×3 sub squares from left to right and from top to bottom.

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 (fig. 2).

The mutations are applied only inside a sub block. We are using three mutation strategies that are commonly used in combinatorial optimization: swap mutation, 3-swap mutation, and insertion mutation (fig. 3).

Each time mutation is applied inside the sub block, the array of givens is referred. If it is illegal to change that position, we randomly reselect the positions and recheck until legal positions are found. In swap mutation, the values in two positions are exchanged. In 3-wap mutation the values of three positions are rotated, either clockwise or counterclockwise. In insertion mutation, one or more numbers are inserted before some number and all the freely changeable numbers are then rotated clockwise or counterclockwise.

To design such a fitness function that would aid a GA search is often difficult in combinatorial problems (Koljonen and Alander, 2004). In this case we decided to use to a simple fitness function that penalizes different constraint violations differently.

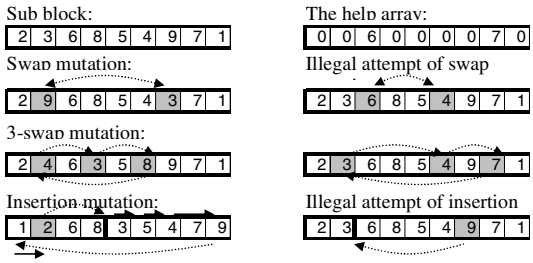

Figure 4. The types of mutation used in Sudoku optimization. Up left is one sub block and up right the static values of that sub block (6 and 7 are givens). The mutation is applied so that we randomly select positions inside the sub block, and then compare the selected positions to the help array if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

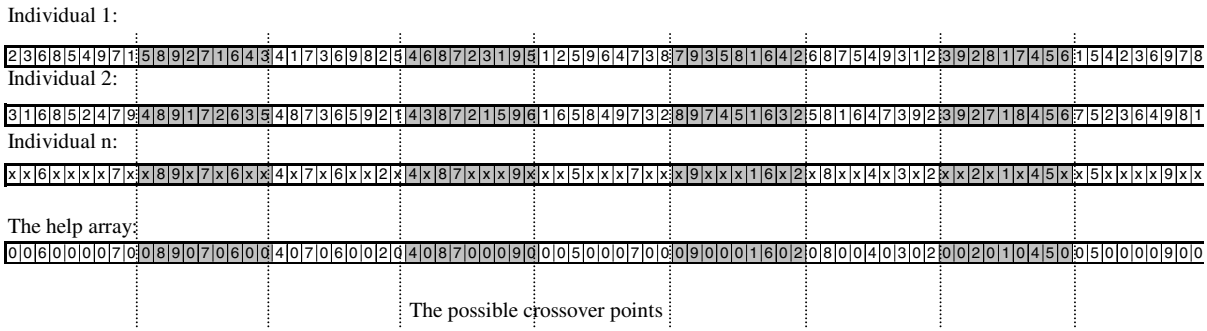

Figure 3. The representation of Sudoku puzzles in GA. One possible solution (individual) is an array of 81 numbers that is divided into nine sub blocks of nine numbers. The crossovers points can only appear between sub blocks (marked as vertical lines). The auxiliary array is used for checking static 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 condition that every 3×3 sub square contains a number from 1 to 9 is guaranteed intrinsically. Penalty functions are used to force the other conditions.

Each row and column of the Sudoku solution must contain each number between [1, 9] once and only once. This can be transformed to a set inequality constraints. The first two equations (1) require that row and column sums should be equal to 45 and the other two equations (2) require that each row and column product should be equal to 9!.

$$g_{i1}(x) = \left| 45 - \sum_{j=1}^{9} x_{i,j} \right|$$
$$g_{j1}(x) = \left| 45 - \sum_{i=1}^{9} x_{i,j} \right| \quad (1)$$

$$g_{i2}(x) = \left| 9! - \prod_{j=1}^{9} x_{i,j} \right|$$
$$g_{j2}(x) = \left| 9! - \prod_{i=1}^{9} x_{i,j} \right| \quad (2)$$

Another requirement is derived from the set theory. It is required that the each row x_i and column x_j, must be equal to set A = {1, 2, …, 9}. The functions (3) calculate the number of missing numbers in each row (x_i) and column (x_j) set.

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

where $|\cdot|$ denotes for the cardinality of a set.

In the optimal situation all constraints (1), (2), and (3) should be equal to zero. The overall fitness function is a linear combination of the partial cost functions (1–3):

$$f(x) = 10*(\sum_i g_{i1}(x) + \sum_j g_{j1}(x)) + \sum_i \sqrt{g_{i2}(x)} + \sum_j \sqrt{g_{j2}(x)} + 50*(\sum_i g_{i3}(x) + \sum_j g_{j3}(x)) \quad (4)$$

The fitness function (4) was chosen after a series of tests of different ideas for constraints and parameter values. It penalizes most heavily if some numbers of set A are absent in the row or column set. It also penalizes if sums and products of a row or column are not equal to the optimal situation. This fitness function setting worked satisfactorily, but in the future we need more consideration whether or not this is a proper fitness function for the Sudoku problem.

3 Experimental results

In order to test the proposed method we decided to use an integer coded GA. The size of the GA chromosome is 81 integer numbers, divided into nine sub blocks of nine numbers. The uniform crossovers were applied only between sub blocks, and the swap, 3-swap and insertion mutation with relative probabilities 50:30:20, respectively, inside sub blocks, with an overall mutation probability 0.12. The population size was 100, and elitism 40.

We tested five Sudoku puzzles taken from the newspaper Helsingin Sanomat (2006) marked with their difficulty rating `1-5 stars`. These had 28 to 33 symmetric givens. We also tested four Sudokus taken from newspaper Aamulehti (2006), they were marked with difficulty ratings: `Easy`, `Challenging`, `Difficult`, and `Super difficult`. They contained 23 to 36 nonsymmetrical givens.

We also generated new Sudoku puzzles (no given numbers in the beginning), marked with difficulty rating: `New Sudoku`. We tried to solve each of the ten Sudoku puzzles 100 times. The stopping condition was the optimal solution found, or max. 100000 generation, *i.e.* max. six million trials (fitness function calls).

The results are given in table 1. The columns (left to right) stand for: difficulty rating, the count of how many of the 100 optimization runs found optimal solution, and minimum, maximum, average, median as well as standard deviation calculated only from those test runs that found the solution (showed in count section). The statistics represent the amount of generations required to find the solution.

Table 1 shows that the difficulty ratings of Sudoku's correlate with their GA hardness. Therefore, the more difficult Sudokus for a human solver seem to be also the more difficult for the GA. The GA found the solution for Sudoku's with difficulty rating `1 star` and `Easy` every time. The `Easy` from Aamulehti was the most easiest to solve in general, even easier than to generate a `New Sudoku`.

With other ratings the difficulty increased somewhat monotonically with the rating, and the count of those optimization runs that found a solution decreased. In addition, the average and median GA generations needed to find solution increased.

Table 1. The comparison of how effectively GA finds solutions for the Sudoku puzzles with different difficulty ratings. The rating `New` means no givens (new Sudoku) and numbers 1-5 are Sudoku puzzles with symmetric givens and their difficulty ratings taken from the newspaper Helsingin Sanomat (2006). The Sudokus with ratings `Easy` to `Super difficult` are taken from newspaper Aamulehti (2006) and they had nonsymmetrical givens. *Count* shows how many of the 100 GA optimization runs found the solution and other columns are the descriptive statistics of how many GA generations was needed to find the solution.

Difficulty rating	*Givens*	*Count*	*Min*	*Max*	*Average*	*Median*	*Stdev*
New	0	100	206	3824	1390.4	1089	944.67
1 `star`	33	100	184	23993	2466.6	917	3500.98
2 `stars`	30	69	733	56484	11226.8	7034	11834.68
3 `stars`	28	46	678	94792	22346.4	14827	24846.46
4 `stars`	28	26	381	68253	22611.3	22297	22429.12
5 `stars`	30	23	756	68991	23288.0	17365	22732.25
`Easy`	36	100	101	6035	768.6	417	942.23
`Challenging`	25	30	1771	89070	25333.3	17755	23058.94
`Difficult`	23	4	18999	46814	20534.3	26162	12506.72
`Super difficult`	22	6	3022	47352	14392	6722	17053.33

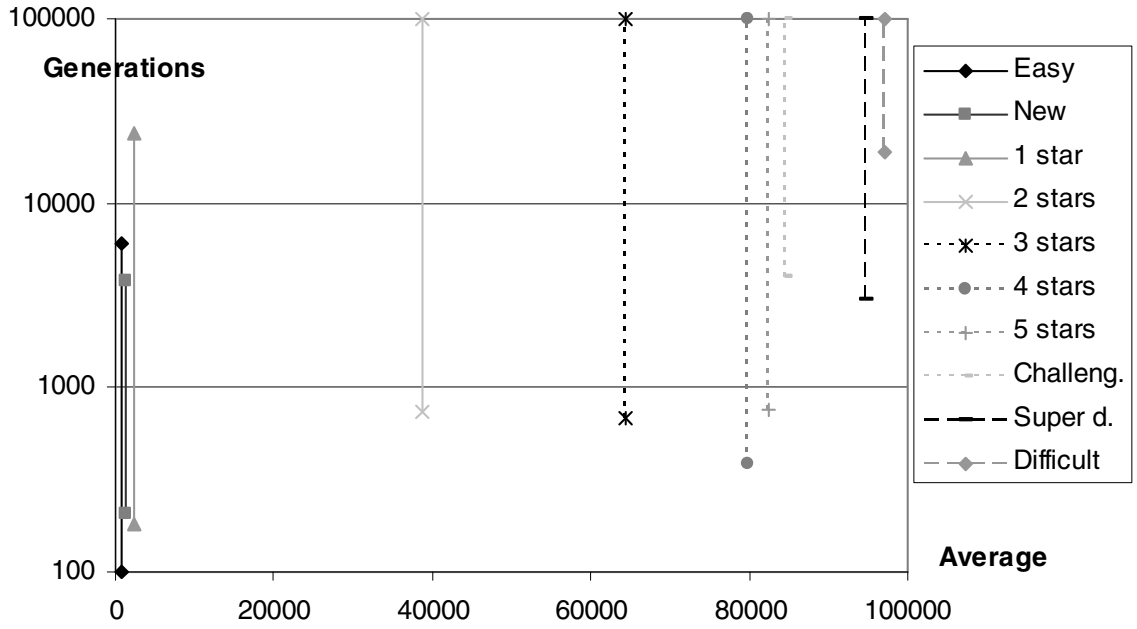

Figure 5. The difficulty order of tested Sudokus. The minimum and maximum generations needed to solve each Sudoku from 100 test runs as a function of average generations needed.

However, the most difficult puzzle (`5 stars`) from Helsingin sanomat was solved 23 times out of 100 test runs, which was almost as often as `4 stars` puzzle (26 times). With the exception of `Easy` the other Sudokus in Aamulehti were found to be more difficult than Sudokus in Helsingin sanomat. The Sudoku with rating `Challenging` was almost as difficult as `4` and `5 stars` Sudokus in Helsingin sanomat.

The puzzles `Difficult` and `Super Difficult` of Aamulehti were much more difficult than any of the puzzles in Helsingin sanomat. The difficulty rating of these two also seemed to be in wrong mutual order (fig. 5), since `Difficult` was harder for GA than `Super Difficult`. However, both were so difficult that GA found the solution only a few times, so the difference is not statistically significant.

Figure 5 shows the difficulty order of the tested Sudokus. The {Easy, `New Sudoku`, `1 star`} seems to form a group of the most easiest Sudokus, while `2` and `3 stars` Sudokus lie by themselves between the previous and the group containing {`4 stars`, `5 stars`, `Challenging`}. The two most difficult Sudokus {`Super difficult`, `Difficult` } are in the final group.

The `New Sudoku` (no givens) is the second easiest for GA in average. This means that GA can be used for generating new puzzles. It takes for GA between [406, 3817] generations ([24400, 229060] trials) to create a new open Sudoku solution that can further be used to create a new puzzle.

If GA is used as puzzle generator, one first searches any possible open solution and then removes some numbers and leave the givens. The difficulty ratings of new Sudoku can also be tested with GA by trying to solve it after the givens are selected.

4 Conclusions and future

In this paper, we studied if Sudoku puzzles can be solved with a combinatorial genetic algorithm. The results show that GA can solve Sudoku puzzles, but not very effectively. There exist more efficient algorithms to solve Sudoku puzzles (Wikipedia, 2006). However, in this study the aim was to test how efficient pure GA is, without problem specific rules, for solving Sudokus. The GA results can of course be enhanced by adding problem related rules.

The other goal was to study if difficulty ratings given for Sudoku puzzles in newspapers are consistent with their difficulty for GA optimization. The answer to that question seems to be positive: Those Sudokus that have a higher difficulty rating proved to be more difficult also for genetic algorithms. This also means that GA can be used for testing the difficulty of a new Sudoku puzzle.

Grading puzzles is said to be one of the most difficult tasks in Sudoku puzzle creation (Semeniak, 2005), so GA can be a helpful tool for that purpose.

The new puzzles are usually generated by finding one possible Sudoku solution and then removing numbers as long as only one unique solution exists. The GA can also be used to select the removed numbers and also testing if there is still only one unique solution. This can be tested e.g. by solving the Sudoku 10 or more times with other GA, and if the result is always identical, we can be relatively sure (but not completely) that a unique solution exists. However, it may be wiser to use more effective algorithms to test the uniqueness of a solution.

The future research may consist of generating a more efficient hybrid GA and algorithms created specifically to solve Sudoku. Another research area would be to study if it is possible to generate a fitness function based on an energy function (Alander, 2004).

It has been said that 17 given number is minimal to come up with a unique solution, but it has not been proved mathematically (Semeniak, 2005). The GA could also be used for minimizing the number of givens and still leading to a unique solution.

Acknowledgements

The research work of the author T.M. was funded by the research grant from the TietoEnator fund of the Finnish Cultural Foundation.

References

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

Alander, J.T. Potential function approach in search: analyzing a board game. 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, 61-75.

Alander, J.T., Mantere T., and 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.

Alander, J.T., Mantere T., and 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, 211-216.

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.

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.

Evonet Flying Circus. *Magic square solver*. Available via WWW: http://evonet.lri.fr/CIRCUS2/node.php?node=65 (1996, cited 11.9.2006)

Gold, M.. *Using Genetic Algorithms to Come up with Sudoku Puzzles*. Sep 23, 2005. Available via WWW: http://www.c-sharpcorner.com/UploadFile/mgold/Sudoku09232005003323AM/Sudoku.aspx?ArticleID=fba36449-ccf3-444f-a435-a812535c45e5 (cited 11.9.2006)

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

Holland, J. *Adaptation in Natural and Artificial Systems*. The MIT Press, 1992.

Kang, H.R.. *Digital Color Halftoning*. SPIE Optical Engineering Press, Bellingham, Washington, & IEEE Press, New York, 1999.

Kobayashi, N., and 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

Koljonen, J. and 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.

Lawler, E.L., Lentra, J.K., Rinnooy, A.H.G., and Shmoys, D.B. (eds.). *The Traveling Salesman problem – A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, New York, 1985.

Li, H., Jiao, Y.-C., and 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.

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, pp. 52-57.

Newbern, J., and 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.

Runarsson, T.P., and Yao, X.: Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation* **4**(3), 2000, pp. 284-294.

Semeniuk, I. Stuck on you. In *NewScientist* **24/31** December, 2005, 45-47.

Sudoku Maker. Available via WWW: http://sourceforge.net/projects/sudokumaker/ (cited 27.1.2006)

Wikipedia. *Sudoku*. Available via WWW: http://en.wikipedia.org/wiki/Sudoku (cited 11.9.2006)

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.