SOLVING THE SUDOKU WITH THE DIFFERENTIAL EVOLUTION

Urszula Boryczka, Przemysław Juszczuk

Institute of Computer Science, Silesian University, ul. Bedzinska 39, Sosnowiec, Poland

Abstract: In this paper, we present the application of the Differential Evolution (DE) algorithm to solving the combinatorial problem. The advantage of the DE algorithm is its capability of avoiding so-called „local minima" within the considered search space. Thanks to the special operator of the adaptive mutation, it is possible to direct the searching process within the solution space. The DE algorithm applies the selection operator that selects from the child population only the offspring with the greater value of the fitness function in comparison to their parents. An algorithm applied to a combinatorial optimization problem: Sudoku puzzle is presented. Sudoku consists of a nine by nine grid, divided into nine three by three boxes. Each of the eighty-one squares should be filled in with a number between one and nine. In this article we show, that the mutation schema has significant impact on the quality of created solution.

Keywords: differential evolution, sudoku, discrete optimization

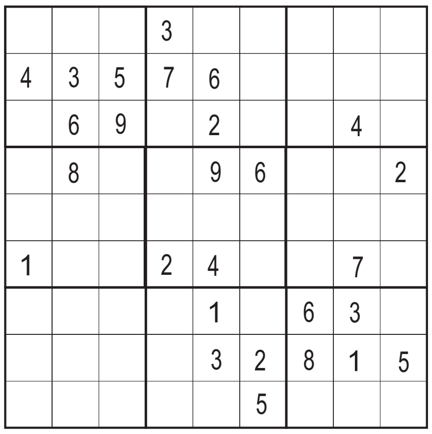
1. Introduction

This paper presents the Differential Evolution algorithm for solving a combinatorial problem - the Sudoku puzzle. Sudoku is a Japanese logical game which achieved international popularity in 2005. A classical Sudoku puzzle consists of 9 by 9 matrix partitioned into nine 3 by 3 sub-matrices. The rules for completing Sudoku are very simple:

- Every subgrid contains all 9 digits.
- Every row contains all 9 digits.
- Every column contains all 9 digits.

At the beginning of game the grid only contains some of the „start values" - selected cells from 9 by 9 matrix.

Zeszyty Naukowe Politechniki Białostockiej. Informatyka, vol. 9, pp. 5-16, 2012.



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

Fig. 1. Example of a Sudoku puzzle

These are then used to fulfill remaining empty cells. Also set of start squares is determined in such way, that it provides a unique solution to the Sudoku. Rules of this game are very simple and the objective is clear. But in the elementary 9x9 puzzle version of the game there are about $6.671x10^{21}$ valid grids, and only one appropriate solution for some given cells is satisfied. Moreover the general case of the problem has been proven to be the NP-Complete [21], which makes it one of the most interesting problems, that can be solved by approximate algorithms.

When generating these puzzles it is important that the set of start squares ensure a unique solution, so in general, puzzles are created by calculating a valid solution, and then removing a subset of the grid values, leaving just enough that the configuration has a unique solution. In general a puzzle require at least 17 start squares to maintain its uniqueness, although this problem remains formally unsolved [15].

The standard Sudoku puzzle is 9 by 9, but any *n* by *n* grid can be made into Sudoku puzzle. For example rather often found in literature problem consist of 16 by 16 cells. In 16 by 16 puzzle the rows, columns and boxes are filled with the numbers 1 to 16. Such a puzzle will be referred to as size four in reference to the value of *n*. The standard Sudoku will be referred to as a size three puzzle. Similarly a 4 by 4 puzzle will be called a size two puzzle.

Main objective of this paper is to test if the Differential Evolution is an efficient method for solving combinatorial problems - especially the Sudoku puzzle. Proposed algorithm will be evaluated on different sets of example problems and different difficulty puzzle levels. Also authors will try to evaluate difficulty of puzzles depending on number of iterations needed to solve the problem and then estimate the puzzle rating.

The paper is structured as follows: first, authors discuss some related works in section 2 and give description of the differential evolution algorithm in section 3. In section 4 authors describe the proposed method - especially an evaluation function and a mutation schema. Next section provides experimental results and finally we end with some conclusions.

2. Related Works

Mathematical foundations of sudoku are well discused in [2,3]. For example, one of aims is to compute the number of valid Sudoku grids. The sudoku puzzle were deeply studied as a constraint programming problem [16,6]. Other proposed method is a transformation of the sudoku puzzle to the SAT problem [8,20].

Sudoku is a combinatorial optimization problem [7], It is obviously related to the ancient magic square problem (Latin square), where different size of squares must be filled in way that the sum of each column and row are equal to 45. The magic square problem has been solved by GAs [1].

Evolutionary methods are not popular methods for solving Sudoku puzzle. There are only a few articles which presents this approach: genetic algoritms were so far used to generate optimal solutions [10]. In [4] authors focus on using the genetic algorithm for generating new instances of the Sudoku puzzle. Other population based methods are ant systems [15] and the Particle Swarm Optimization [12,11,13].

3. The Differential Evolution Algorithm

The Differential evolution is a stochastic technique which was developed by K. Price and R. Storn in 1995 [17]. It is a population–based optimization method which can be used for example to numerical optimization [19], neural network training [9], filter design [18] or image analysis [5]. DE is conceptually similar to the evolutionary algorithm, but there are also quite big differences. First of all, the mutation is the major genetic operator. It is not a trivial process and it also provides the algorithm's convergence. Moreover, the mutation is performed before the crossover process.

The pseudocode of the general DE algorithm:

```
1. Create the initial population of genotypes $P_0 = \{X_{1,0}, X_{2,0}, ..., X_{n,0}\}$,
2. Set the generation number $g = 0$
3. Until the stop criterion is not met:
```

(a) Compute the fitness function for every genotype in the population $\{f(X_{1,g}), f(X_{2,g}), ..., f(X_{n,g})\}$
(b) Create the population of trial genotypes V_g based on P_g
(c) Make crossover of genotypes from the population P_g and V_g to create population U_g
(d) Choose the genotypes with the highest fitness function from the population U_g and P_g for the next population
(e) $generation = generation + 1$, go to step a.

The DE algorithm begins with the initialization of the population $P(0)$ which consist of n_X individuals. Mutation is a process that adds randomly-generated values to the selected genes. Each increment moves selected individuals towards the global optimum. Note that the mutation is used to every individual in the population. Trial individual $u_i(t)$ is created as follows:

$$u_i(t) = x_{i_1}(t) + F \cdot (x_{i_2}(t) - x_{i_3}(t))$$

Individual $x_i(t)$ is called the target vector. $(x_{i_2}(t) - x_{i_3}(t))$ is a differential vector created from the two random individuals $x_{i_2}(t)$ and $x_{i_3}(t)$.

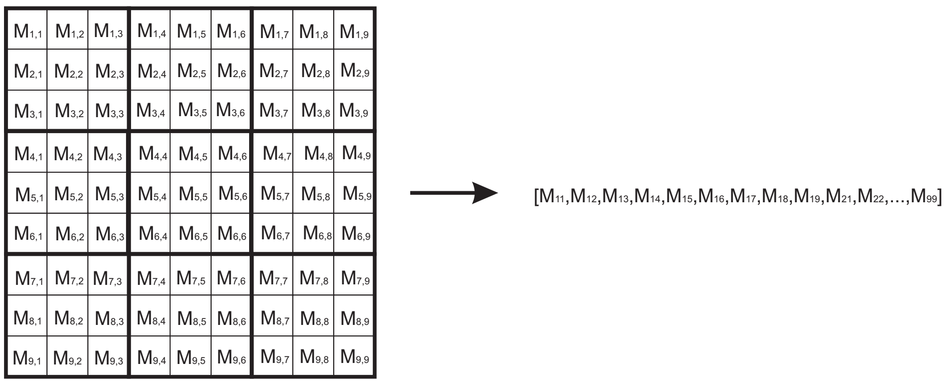
Crossover uses both the genotype from the population P_g and the trial genotype (population V_g). This operator will be described deeply in next section. After the crossover process the offspring is compared with its parent. Next, the better one of these individuals is added to the new population. The last step of the algorithm is the increment of the generation counter t. The best individual from the last generation is the result of the DE algorithm.

4. The Proposed Methodology for Solving the Sudoku Puzzle

Sudoku is a combinatorial problem, where each row, column and also each of nine 3 by 3 small square must have number digit from set $\{1,2...9\}$ exactly once. Our method assumes, that the single individual genotype is the one dimension vector, where every gene holds value from set $\{1,2,\ldots,9\}$. The array is divided into nine sub blocks (building blocks) where each block consists of nine digits.

Proposed method of transforming Sudoku square to linear vector implies some conditions. First we assume that every digit may occur only once in every of 9 genotype parts (each part is 9 genes long and corresponds to small Sudoku square).

Two mutation schemas are presented in this paper. First mutation schema applied for our algorithm is based on standard swap operator used in combinatorial problems. The mutations are applied only inside a sub block. We are using the mutation strategy



$M_{1,1}$ … $M_{9,9}$
$[M_{11},M_{12},M_{13},M_{14},M_{15},M_{16},M_{17},M_{18},M_{19},M_{21},M_{22},\ldots,M_{99}]$

Fig. 2. Transforming Sudoku into the genotype

that is common in the combinatorial optimization - swap mutation (Fig. 3). Of course some genes in genotype are given from the beginning of the algorithm. This positions are related to the Sudoku puzzle. As it was said in section 1 each puzzle requires at least 17 start squares to maintain its uniqueness. Mutation schema allows to swap this position, but strong penalty function is used while evaluating each individual.

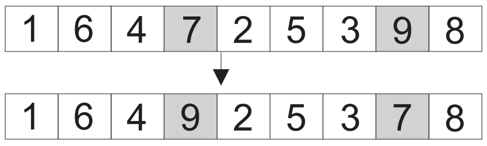


Fig. 3. Swap mutation

Other mutation schema is close to approach used in [14] and it is reffered as the geometric operator [14,13].

$$diffvector = \begin{bmatrix} x_{11}\ x_{12}\ \ldots\ x_{1p} \\ x_{21}\ x_{22}\ \ldots\ x_{2p} \\ \ldots\ \ldots\ \ldots\ldots \\ x_{n1}\ x_{n2}\ \ldots\ x_{np} \end{bmatrix} - \begin{bmatrix} y_{11}\ y_{12}\ \ldots\ y_{1p} \\ y_{21}\ y_{22}\ \ldots\ y_{2p} \\ \ldots\ \ldots\ \ldots\ldots \\ y_{n1}\ y_{n2}\ \ldots\ y_{np} \end{bmatrix}$$

where x and y are individuals from population P, n is number of rows in Sudoku (equal to 9) and p is number of columns in Sudoku (also equal to 9). In the next step, only a few randomly selected cells are copied to the trial vector.

The first interpretation of the mutation schema is close to the general one used in the Differential Evolution algorithm for the continuous optimization. Unfortunately, as we show in the next section, this approach gives worse effect than the second option.

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 core of this algorithm is the fitness function. It is required that the each row x_i and column x_j must be equal to set $A = \{1, 2, ..., 9\}$. So we can calculate number of missing digits in each row x_i and column x_j set:

$$f_1 = |A - x_i|$$

$$f_2 = |A - x_j|$$

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

$$f_3 = f_3 + 100 \text{ if } (fixed_{i,j} \neq 0) and (fixed_{i,j} \neq x_{i,j})$$

where $x_{i,j}$ denotes for cell at the intersection of row i and column j, and $fixed$ denotes for matrix of given cells (matrix cell is filled only if cell is given, otherwise cell value is equal to zero). So the fitness function is equal to:

$$fit(x) = f_1 + f_2 + f_3 \tag{1}$$

The fitness function is computed for each offspring from the population U_g, and the comparison of the computed fitness function with the value of the fitness function of the parent (from the population P_g) follows. The genotype with the higher fitness function value is transferred to the next population. Our proposed method is based on two simple rules which allows properly evaluate population:

- rule 1 is derived from the set theory. It is required that the each row and column of Sudoku must be equal to set $A = \{1, 2, ..., 9\}$.
- second rule is linked with mutation operator. If mutation affects on cell that is considered as constant value in puzzle - the fitness function is strongly decreased (this case explicitly rejects individual as unsatisfactory).

A selection schema used in this algorithm guarantee that individuals promoted to the next generation are not worse than individuals in actual generation.

$$X_{i,t+1} = \begin{cases} U_{i,t} \text{ when } f(U_{i,t}) \leq f(X_{i,t}), \\ X_{i,t} \text{ else.} \end{cases}$$

5. Experimental results

The aim of this research work is to determine if the Differential Evolution algorithm is capable to solve an example of the difficult combinatorial problem - the Sudoku puzzle. For all experiments described in this section we assume the following settings:

- size of the population $P = 200$,
- crossover $CR = 0.5$,
- swap mutation,
- the length of the gene was 81,
- the maximum number of iterations was set to 40000.

For our experiments we used a group of Sudoku puzzle divided in dependence of their difficulty. Every puzzle was run 100 times. In the Table 1 we put information about the given number of cells at the beginning of the algorithm, minimum, maximum and average number of iterations needed to achive solution, median and standard deviation. We also used the Differential Evolution to generate Sudoku puzzle from scratch. Afterwards newly generated puzzle is solved by our algorithm and rated. Unfortunately first mutation schema based on geometric operators provides far worse solutions and the algorithm is capable to solve within a reasonable time only easy puzzle (less than 20000 iterations), results shown below concern only the Differential Evolution algorithm with the swap mutation. Also, as a comparison, in the Table 2 we present results for solving Sudoku puzzle with the Genetic Algorithm [10]. We used the same Sudoku problems, and the same number of individuals in the population. As it may be seen, it is difficult to clearly rate newly generated puzzle. Four sets generated from the scrach, had 30 cells filled. It is rather unclear, if the newly generated puzzles should be assigned to the „easy” or „medium” classes. The Differential Evolution seems to be far more effective in solving the Sudoku problems. Most likely, cause of this is the selection schema. In the Differential Evolution only the best adapted individuals are moved to the next generation. In the Genetic Algorithm we used the proportional selection, which allows worse adapted individuals to move into the next generation. Premature convergence in the Sudoku is not the problem, and the more greedy schema is better. Interesting is fact, that both algorithms seems to be sometimes insensitive to the Sudoku difficulty level. It may be seen in the GA results, where average number of iterations needed to find solution i similar for the Medium and Hard problems. We also present distribution of population in subsequent iterations for selected „easy”, „medium” and „hard” puzzle.

Two interesting population features may be seen at Figures 4, 5 and 6. In all three cases, at the first step of the algorithm significant improvement of fitness function can

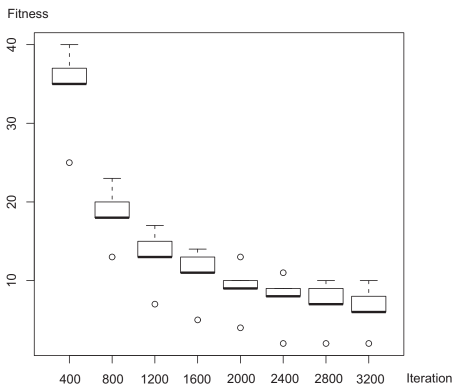


Fig. 4. Distribution of population in subsequent iterations for „easy” puzzle

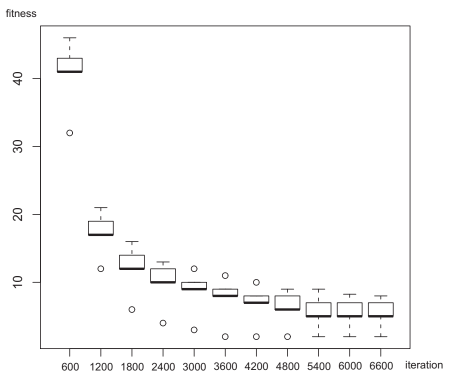


Fig. 5. Distribution of population in subsequent iterations for „medium” puzzle

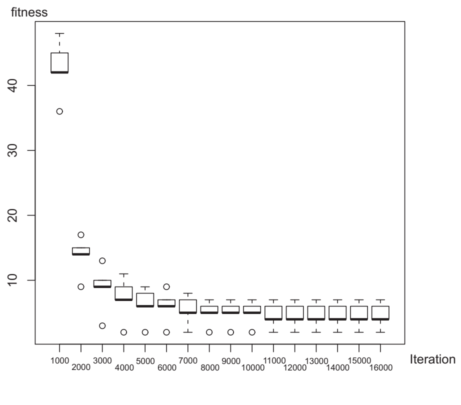


Fig. 6. Distribution of population in subsequent iterations for „hard” puzzle

Table 1. The comparison of how effectively DE finds solutions for the Sudoku puzzles with different difficulty ratings.

Difficulty rating	Cells given	Minimum number of iterations	Maximum number of iterations	Average number of iterations	Median	Std dev
New	0	855	1638	1258.714	1354	281.4378
Easy	34	2344	5344	3701.9	3377	1026.246
Easy	30	2270	5040	3801.9	3824	949.161
Medium	30	4571	12343	6833.9	5665	3537.016
Medium	26	5709	16743	10713.5	9946.5	3548.422
Hard	26	7276	29686	15206.4	12564.5	7361.502
Hard	24	6635	30105	19849.43	19701	7624.729
New generated 1	30	2461	6875	4976.571	5805	1642.706
New generated 2	30	2483	4164	3203.857	2923	652.475
New generated 3	30	3007	7447	4915.286	4238	1829.077
New generated 4	30	3480	9493	5583.571	4946	2153.169

be observed. But further, solutions have been improved very slowly. Second feature of the algorithm is clearly seen at Fig. 6. Solutions for the puzzle have been found in about 17 thousands iterations, but from 11th thousand iteration no improvements are reported. Minimal fitness value of population remains at 2 (0 is optimal solution).

6. Conclusions and future work

After the analysis of all test sets, the following conclusions may be reached:

- The Differential evolution is capable to solve a very difficult Sudoku puzzles. For example „hard” instance in 15 thousand iterations.
- The Proper mutation schema is very difficult to implement even with strong theoretic analysis of geometric operators.
- The Simple swap mutation brings very good results.
- The Algorithm is far better with finding solutions close to the optimal solution (for example the fitness function equal 2).

To summarize it should be observed as well that the Differential Evolution and other evolutionary algorithms are capable not only to find solutions for Sudoku puzzle but also to generate new puzzles from scratch. The new mutation schema seems to be very effective tool for solving dicrete problems. Our next goal is to adapt similar mutation into the continuous optimization problem. One of the most difficult things, will

Table 2. The comparison of how effectively GA finds solutions for the Sudoku puzzles with different difficulty ratings.

Difficulty rating	**Cells given**	**Minimum number of iterations**	**Maximum number of iterations**	**Average number of iterations**	**Median**	**Std dev**
New	0	6831	40802	20261.4	17462	11870.7
Easy	34	6251	15421	9111.2	7462	4397.7
Easy	30	4918	12572	6472.6	5150	4056.1
Medium	30	6253	26255	13921.4	7124	10559.5
Medium	26	7171	33498	19846.6	17463	9409.9
Hard	26	11563	21462	16312	16857	3987.2
Hard	24	10831	40802	20261.4	17462	11870.7
New generated 1	30	7513	24524	16598.4	16356	6578.3
New generated 2	30	8457	18436	11427	9135	4242.9
New generated 3	30	3476	31673	15062.6	12452	10409.1
New generated 4	30	4339	28463	13036	9687	9753.8

be descrption of the mutation schema, which will allow to modify only small part of the genotype even in the continuous optimization problems. The key issue should be the proper representation of the individual. We try to show, that appropriate problem transformation is one of the main problems for the described mutation schema.

References

[1] D. Ardel, TOPE and magic squares: a simple GA approach to combinatorial optimization, 1994, pp. 1–6.

[2] B. Felgenhauer, Jarvis F.,Mathematics of Sudoku I, 2006.

[3] B. Felgenhauer, Jarvis F., Mathematics of Sudoku II, 2006.

[4] M. Gold, Using Genetic Algorithms to come up with Sudoku Puzzles, 2005.

[5] K. U. Kasemir, K. Betzler., Detecting ellipses of limited eccentricity in images with high noise levels, Journal Image and Vision Computing, Vol. 21, 2003, pp. 221–227.

[6] J. Lauriere, A language and a program for stating and solving combinatorial problems, Journal of Artificial Intelligence, Vol. 10, 1978, pp. 29–127.

[7] E. Lawler, Kan A. Rinnooy, The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, 1985.

[8] I. Lynce, J. Ouaknine, Sudoku as a SAT Problem, Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics, 2006.

[9] George D. Magoulas, Michael N. Vrahatis, George S. Androulakis, Effective backpropagation training with variable stepsize, Journal of Neural Networks, Vol. 10, No. 1, 1997, pp. 69–82.

[10] T. Mantere, J. Koljonen, Solving, rating and generating Sudoku puzzles with GA, IEEE Congress on Evolutionary Computation 2007, 2007, pp. 1382–1389.

[11] S. McGerty, Solving Sudoku Puzzles with Particle Swarm Optimisation, Final Report, Macquarie University 2009.

[12] A. Moraglio, C. Di Chio, J. Togelius and R. Poli, Geometric Particle Swarm Optimization - Research Article, Journal of Artificial Evolution and Applications, Vol. 2008, 2007.

[13] A. Moraglio, J. Togelius, Geometric Particle Swarm Optimization for the Sudoku Puzzle, GECCO 2007, Genetic and Evolutionary Computation Conference, 2007.

[14] A. Moraglio, J. Togelius, Geometric differential evolution, GECCO 2009, Genetic and Evolutionary Computation Conference, 2009, pp. 1705–1712.

[15] D. Mullaney, Using Ant Systems to Solve Sudoku Problems, University College Dublin.

[16] H. Simonis, Sudoku as a constraint problem, Proceedings 4th Int. Works. Modelling and Reformulating Constraint Satisfaction Problems, 2005, pp. 13–27

[17] R. Storn, K. Price, Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, Journal of Global Optimization, Vol. 11, 1997, pp. 341–359.

[18] R. Storn, Differential evolution design of an IIR-filter, IEEE International Conference on Evolutionary Computation ICEC'96, 1996, pp. 268–273

[19] R. Storn, On the Usage of Differential Evolution for Function Optimization, AFIPS'96, 1996, pp. 519–523.

[20] T. Weber, A SAT-based Sudoku solver, 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR 2005, 2005, pp. 11–15.

[21] T. Yato, Complexity and Completeness of Finding Another Solution and its Application to Puzzles, IEICE - Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. 5, 2003, pp. 1052–1060.

Urszula Boryczka, Przemysław Juszczuk

EWOLUCJA RÓŻNICOWA W ROZWIĄZYWANIU SUDOKU

Streszczenie: W artykule przedstawimy propozycję zastosowania algorytmu ewolucji różnicowej do rozwiązywania problemów kombinatorycznych. Przewagą ewolucji różnicowej jest zdolność do unikania optimów lokalnych w przestrzeni przeszukiwań. Specjalny operator mutacji pozwala ukierunkować proces poszukiwań rozwiązania. W ewolucji różnicowej stosowany jest operator selekcji, który promuje tylko najlepiej przystosowane osobniki z populacji rodziców i potomków. Przedstawimy zastosowanie opisanego algorytmu do problemu rozwiązywania Sudoku. Sudoku składa się z planszy 9 na 9, podzielonej na 9 sekcji - każda o rozmiarze 3 na 3 elementy. Każda z 81 kratek powinna zostać wypełniona wartością z przedziału 1 do 9. W artykule pokażemy, że ewolucja różnicowa pozwala na rozwiązywanie Sudoku.

Słowa kluczowe: ewolucja różnicowa, sudoku, optymalizacja dyskretna