Solving the 3-SAT problem using genetic algorithms

Conference Paper · September 2015

DOI: 10.1109/INES.2015.7329708

Jana Lovíšková
Slovak Academy of Sciences

SOLVING THE 3-SAT PROBLEM USING GENETIC ALGORITHMS

Ing. Jana Lovíšková, PhD.
Department of modeling and control of discrete processes
Institute of Informatics SAS
Bratislava, Slovakia
jana.loviskova@savba.sk

Abstract - **The goal of the article is to compare efficiency of solving a problem of determining satisfiability of logical formulas SAT (3-SAT), which belongs to a class of NP-complete problems, either by using simple genetic algorithms or using genetic algorithms with suggested modified mechanism of stepwise adaptation of weights (SAW). Two possible ways of dealing with constraint - either based on a penalization or based on a remuneration are compared here. It is shown, on the basis of repeated and evaluated experiments, that a genetic algorithm with suggested mechanism of stepwise adaptation of weights (SAW) is much more efficient for solving the 3-SAT problem.**

Keywords - NP-complete problem; 3-SAT problem; back-tracking; local searching; genetic algorithm (GA); stepwise adaptation of weights (SAW)

I. INTRODUCTION

A problem of determining satisfiability of logical formulas - SAT (satisfiability problem) [1] is one of the first tasks, for which it has been demonstrated that it belongs to a class of NP-complete problems [7]. This task occurs in many practical applications in the field of logical inference, combinatorial problems (N-queens problem, sudoku, etc.), image interpretation, diagnostics, planning and scheduling tasks (e.g. a problem of planning and scheduling of manufacturing operations). In case, that a problem of planning and scheduling of manufacturing operations is understood as a SAT problem, a basic idea consists of encoding of the scheduling problem in a form of propositional formulas. The problem of determining the satisfiability of logical formulas (satisfiability problem - SAT) is defined as follows: Let a logical formula in a conjunctive normal form be

$$F(x_1,...,x_n) = \bigcap_{i=1}^{m} \bigcup_{j=1}^{m_i} a_{ij}\ , \tag{1}$$

where each a_{ij} is either x_k or $\neg x_k$ for a suitable *k*.

The aim is to determine, whether it is possible to find such truth values of Boolean variables x_1, ..., x_n , for which the formula F (x_1, ..., x_n) is satisfied [2], [8]. It can be relatively easy to show that any formula in a conjunctive normal form can be converted into a form, in which each clause contains exactly three literals. Therefore, especially for testing characteristics of individual algorithms (as well as in the literature), a 3- SAT problem is more frequently considered, instead of the above-defined problem, where

$$F(x_1,...,x_n) = \bigcap_{i=1}^{l}\left(a_i \cup b_i \cup c_i\right), \tag{2}$$

every a_i, b_i, c_i , $i = 1, \dots, l$, are either x_k or $\neg x_k$ for a suitable *k* [2].

In this work, we focus on the comparison of the effectiveness of a simple genetic algorithm and a genetic algorithm with suggested mechanism of stepwise adaptation of weights in solving the 3-SAT problem. At the same time, we choose two different methods dealing with constraints using either penalization or remuneration, used in a simple genetic algorithm as well as in a genetic algorithm with suggested mechanism of stepwise adaptation of weights (SAW). Besides a comparison of the effectiveness of a simple genetic algorithm and a genetic algorithm with the suggested mechanism of a stepwise adaptation of weights, our goal is to choose such a way of dealing with constraints, which is preferable for the solving of the 3-SAT problem.

Fot this purpose, we created four different variants of genetic algorithms. Variant 1 – a simple genetic algorithm with penalization, variant 2 – a simple genetic algorithm with remuneration, variant 3 – a genetic algorithm with suggested mechanism of stepwise adaptation of weights with penalization, variant 4 – a genetic algorithm with suggested mechanism of stepwise adaptation of weights with remuneration. We gradually explored and evaluated the effectiveness of each variant with regard to solving the 3-SAT problem, in another procedure.

II. POSSIBILITIES OF SOLVING 3-SAT PROBLEM

Let us show a concrete example for understanding the 3-SAT problem. Let us consider 50 Boolean variables $x_1, \dots, x_{50}$ and a logical formula F in a conjunctive normal form

$$F(x_1,...,x_{50}) = \\ = (\neg x_1 \cup x_{17} \cup x_{25}) \cap (x_{13} \cup \neg x_8 \cup x_{30}) \cap \cdots (x_{16} \cup \neg x_7 \cup x_{29}). \tag{3}$$

This work has been supported by the Slovak scientific grant agency VEGA under a grant No. 2/0194/13.

The aim of the task is to decide, whether there is such truth evaluation of variables x_1 to x_{50}, for which

$$F\left(x_1,\ldots,x_{50}\right) = 1 . \tag{4}$$

A number of algorithms solving the problem of determining the satisfiability of logical formulas using some form of back-tracking are known. Using these deterministic methods, all possible truth evaluations of the variables $x_1, \ldots, x_n$ are stepwise generated, and at the same time, a testing, whether the formula under consideration for the given evaluation is satisfied, is conducted. There are total of 2^n different truth evaluations, therefore it is possible to solve only trivial tasks with a very small number of variables *n,* this way. More efficient methods use the fact that the formulas are in conjunctive normal forms and each formula is satisfied only if all clauses are satisfied. It is possible to use a branch and bound method [2], when evaluation of the variables x_1 to x_n are stepwise generated in a specific order (instead of generating the truth evaluation of all variables). Once, after assigning the value to variables $x_1, \ldots, x_k$ $(1 \le k \le n)$, any of the clauses of the formula $F(x_1, \ldots, x_n)$ is not satisfied, it does not make sense to continue in generating truth evaluation of variables $x_{k+1}, \ldots, x_n$. It is clear that it is necessary to go back and look for other evaluation of the first *k* variables. The best known deterministic algorithm for the problem SAT (3-SAT) is an algorithm also based on the method of back-tracking. Its essential advantage, however, is the use of resolution to the possible elimination of some literals, so that the corresponding logical formula becomes simpler [2]. The problem of any of the methods based on the back-tracking is that, in the practice, it is applicable for only very small values of parameter *n*, because size of the search space increases exponentially according to the expression 2^n with the increasing number of variables. On the other side, it is clear that using any of these methods will surely find such truth evaluation of variables $x_1, \ldots, x_n$, where the formula $F(x_1, \ldots, x_n)$ will be satisfied or it will be possible to state confidently that such evaluation does not exist and formula is not satisfiable. On the ground of usability of these methods in practice only for a very small number of variables *n,* a great attention is currently devoted to algorithms using different heuristics (metaheuristics).

Within heuristic (metaheuristic) algorithms, very good results in solving the problem SAT (3-SAT) can be achieved by using an algorithm based on a local searching [2], [8]. This algorithm starts with a randomly generated truth evaluation of variables. The algorithm consequently performs local searching in a way that it changes the truth evaluation of each of the variables stepwise and keeps a record of how many clauses remained, after each of these changes, unsatisfied. After checkout of a change of the truth value for each of *n* variables, previous solution is replaced by a new solution that has the least number of unsatisfied clauses among currently generated solutions. This iterative process is repeated as long as such truth evaluation of variables $x_1, \ldots, x_n$ is found, for which the formula $F(x_1, \ldots, x_n)$ is satisfied or until a maximum of permissible and forward specified number of iterations is reached. If the solution of the problem is not found within a given number of iterations, a new initial solution will be generated randomly and the whole algorithm is reiterated. The number of new starts is usually limited by suitable constants due to finality of the algorithm [2]. The stated algorithm can be further improved so that some weight, which increases while this clause remains unsatisfied, is assigned to each clause. Thus often unsatisfied clauses stepwise acquire higher weights as easily satisfiable clauses and any solution, which satisfies the worst satisfiable clauses, is preferred, because of the weights.

Using genetic algorithms to solve a problem of determining the satisfiability of logical formulas seems to be quite straightforward. Each Boolean variable can be represented as one binary gene (element of chromosome) such that the *i*-th gene represents the variable x_i and its allele determines the truth evaluation of a variable x_i. The whole chromosome thus can be interpreted as the truth evaluation of variables $x_1, \ldots, x_n$ and it may be seen that the formula $F(x_1,\ldots, x_n)$ can be used without problems as a cost (fitness) function. This seemingly correct procedure would be quite wrong, because the algorithm will lose any information about how near or how far is a currently examined individual from the feasible solution by such a choice of evaluation function [2], [9], [10]. Therefore, the evaluation of individual binary strings according to the number of clauses that are satisfied by the corresponding truth evaluation, was proposed.

Within the solution of the problem of satisfiability of logical formulas through genetic algorithms we always cope with constraints through the specific penalization or remuneration. However, a number of disadvantages come out from the above listed approaches. The most important of them is a fact that various clauses are considered quite uniformly, without further separation of the extension to which it is difficult the satisfying of the individual clauses. This problem can be tackled by assigning different weights to individual clauses, so that often unsatisfied (worst satisfiable) clauses acquire stepwise higher weights than clauses that are relatively easily satisfiable.

Then every solution that satisfies just worst satisfiable clauses due to its weight, is preferred. An idea to supplement a genetic algorithm by mechanism that would allow changing the weights of individual clauses comes exactly from these reasons [2], [9], [10]. The ideal seems to be a possibility to incorporate a capability of adaptation the weights of individual clauses directly into a genetic algorithm and improve it so much that the stepwise adaptation of weights occurs in an actual running of the genetic algorithm.

III. USING GENETIC ALGORITHMS FOR SOLVING THE 3-SAT PROBLEM

A logical formula consisting of $m = 300$ clauses, each clause contains literals x_i, $i=1,2,...,n;\ n=50,$ have been chosen for the experiment. As mentioned above, it is valid for the 3-SAT problem that each clause contains exactly 3 literals x_a (or $\neg x_a$), x_b (or $\neg x_b$), x_c (or $\neg x_c$), $\{a,b,c\}$ □ $<1;n>$, literals can be repeated in individual clauses. A population size in GA was set to $N_{pop}= 30$ individuals, each individual contains $n = 50$ elements (the truth values of individual literals).

Variant 1 - simple GA with penalization - consists of the following steps:

1. generate random population of binary coded individuals of the size N_{pop}
2. calculate the value of the fitness function for each individual in the population, where the penalty of value 1 is added to an individual into the calculation of the value of fitness function for each unsatisfied clause
3. create a new population for the next generation (selection, mutation, crossover) on the basis of algorithm GA
4. repeat the procedure from step 2 until the specified number of generations is reached, or until the global optimum (which is known) is achieved.

Variant 2 - simple GA with remuneration

1. generate random population of binary coded individuals of the size N_{pop}
2. calculate the value of the fitness function for each individual in population, where a value of 1/10 is subtracted from the calculation of the value of individual´s fitness function for each satisfied clause, making it actually a reward
3. create a new population for the next generation (selection, mutation, crossover) on the basis of algorithm GA
4. repeat the procedure from step 2 until the specified number of generations is reached, or until the global optimum (which is known) is achieved.

In order to achieve a higher efficiency of solving the 3-SAT problem using genetic algorithms, the following modifications of this mechanism with implemented mechanism SAW (variant 3, variant 4) were proposed.

Variant 3 - GA with mechanism of stepwise adaptation of weights with penalization

1. Generate random population of binary coded individuals of the size N_{pop}
Initialize all elements of vector Y of dimension m by the values 0. Each j-th element of the vector represents the frequency of satisfying the j-th clause through all individuals of the population.
Initialize all elements of vector W of dimension m by the values 1. Here the values of weights of individual clauses during the solving GA will be saved. The higher weight corresponds to the higher difficulty of satisfying the given clause during the solving GA.
Initialize counter generations by $g = 0$.
Initialize the matrix A of dimension 5 rows, m-columns to values 0. Matrix A represents a tray of past 5 vectors W.
2. Increment counter generations $g = g+1$
3. Evaluate the truth value $f(y)$ - the entire logical formula F in a conjunctive normal form consists of logical clauses (containing exactly 3 literals) in line with (3) for the entire population. Calculate the value of the fitness function for each individual in the population according to the equation

$$fit(i) = \left(1 - f(y_i)\right) . P_i\ , \tag{5}$$

where y_i is a vector of truth values of the i-th individual (representative satisfying or non-satisfying individual clauses of formula F through i-th individual) and $f(y_i\ )$ presents the conjunction of all truth values of the vector y_i (satisfying or non-satisfying the entire logical formula F through i-th individual).

$$P_i = \sum_{j=1}^{m} W(j).(1 - y_i(j))\ , \tag{6}$$

where P_i is the penalty function, y_i is a vector of truth values of the i-th individual, and $y_i(j)$ is the j-th element of the vector y_i. For each j-th unsatisfied clause ($y_i\ (j)=0$) in the truth vector y_i of the given individual is the value $W(j)$ added to penalty P_i for given individual. The penalty P_i is considered by calculating the value of the fitness function in agreement with (5).
4. Calculate the vector Y

$$Y = \sum_{i=1}^{N_{pop}} y_i\ , \tag{7}$$

where y_i is a vector of truth values of all m clauses of the i-th individual
5. Create a new population for the next generation (selection, mutation, crossover) on the basis of algorithm GA
6. Fill all elements of vector W of dimension m by the values “1”. Find 5 least successful clauses a,b,c,d,e of the vector Y (serial numbers of clauses which acquire the lowest values): $\{a,b,c,d,e\}$ □ $<1;m>$. Assign the values: $W(a)=5,\ W(b)=5,\ W(c)=5,\ W(d)=5,\ W(e)=5$ to the elements of vector W.
7. If $g \leq 5$, fill the k-th row of the matrix A with elements of the vector W, where $k = g$. If $g > 5$, move the rows 2-5 of the matrix A to rows 1-4. Fill the 5-th row of matrix A by the elements of the vector W.
8. Retype all elements of the vector W with new values according to the equation

$$W = \sum_{r=1}^{5} A_{r,j} \qquad j=1\ ,2,...\ ,m. \tag{8}$$

9. Fill all the elements of the vector Y of dimension m by the values “0”

10. Repeat the procedure from step 2 until the specified number of generations is reached, or until the global optimum (which is known) is achieved.

Variant 4 - GA with mechanism of stepwise adaptation of weights with remuneration

1. Generate random population of binary coded individuals of the size N_{pop}
Initialize all elements of the vector Y of dimension m by the values 0. Each j-th element of the vector represents the frequency of satisfying the j-th clause through all individuals of the population.
Initialize all elements of the vector W of dimension m by the values 1. The values of weights of individual clauses will be saved here, during the solving GA. The higher weight corresponds to the higher difficulty of satisfying the given clause during the solving GA.
Initialize counter generations to $g = 0$.
Initialize the matrix A of dimension 5 rows, m-columns to values 0. The matrix A represents a tray of the past 5 vectors W.
2. Increment counter generations $g = g+1$
3. Evaluate the truth value $f(y)$ - the entire logical formula F in a conjunctive normal form consists of logical clauses (containing exactly 3 literals) in agreement with (3) for the entire population. Calculate the value of the fitness function for each individual in the population according to the equation

$$fit(i) = -1000\, f(y_i) - (R_i \div 10), \tag{9}$$

where y_i is a vector of truth values of the i-th individual (representative satisfying or non-satisfying individual clauses of formula F through i-th individual) and $f(y_i)$ presents the conjunction of all truth values of the vector y_i (satisfying or non-satisfying the entire logical formula F through the i-th individual)

$$R_i = \sum_{j=1}^{m} W(j).\, y_i(j), \tag{10}$$

where R_i is a reward function, y_i is a vector of truth values of the i-th individual, and $y_i(j)$ is the j-th element of the vector y_i. This means that for each j-th satisfied clause ($y_i(j)=1$) in the truth vector y_i of given individual is a value $W(j)$ added to the reward R_i for the given individual . The final value of the reward R_i for the given individual is then in the calculation of the value of fitness function (for given individual) divided by 10 and subtracted in agreement with (9).
4. Calculate the vector Y

$$Y = \sum_{i=1}^{N_{pop}} y_i\,, \tag{11}$$

where y_i is the vector of truth values of all m clauses of the i-th individual

5. On the basis of algorithm GA create a new population for the next generation (selection, mutation, crossover)
6. Fill all elements of the vector W of dimension m by the values “1”. Find 5 least successful clauses a,b,c,d,e of the vector Y (serial numbers of clauses which acquire the lowest values): $\{a,b,c,d,e\}$ □ $<1;m>$. Assign the values: $W(a)=5$, $W(b)=5$, $W(c)=5$, $W(d)=5$, $W(e)=5$ to the elements of vector W.
7. If $g \le 5$, fill the k-th row of the matrix A with elements of the vector W, where $k = g$. If $g > 5$, move the rows 2-5 of the matrix A to rows 1-4. Fill the 5-th row of the matrix A by the elements of the vector W.
8. Retype all elements of the vector W with new values according to the equation

$$W = \sum_{r=1}^{5} A_{r,j} \qquad j=1\,,2,\dots,m\,. \tag{12}$$

9. Fill all the elements of the vector Y of dimension m by the values “0”
10. Repeat the procedure from step 2 until the specified number of generations is reached, or until the global optimum (which is known) is achieved.

The parameters for tuning values, i.e. a number of worst satisfiable clauses to select, increment of their weight and number of rows of the matrix A, were chosen in a heuristic way on the basis of simulation results. We have chosen in this way such values of parameters mentioned above, which provide the best efficiency of the proposed variants 3 and 4 (GA with mechanism SAW).

Because the values of these parameters for tuning are depending on the number of clauses, it means that the optimal values of these parameters are around 1 % - 2 % from a given number of clauses.

A. Case study for the 3-SAT problem

The following parameters of the algorithm were considered while verifying the proposed tuning algorithm (variant 3, variant 4):
The number of logical clauses $m = 300$, number of generations GA = 500; number of iterations the whole GA = 100, population size N_{pop} = 30 individuals, string size = 50 elements.

Performance analysis of the proposed tuning algorithm (variant 3 and variant 4) was performed on the 3-SAT problem of the relatively high intensity (50 logical variables - literals, 300 logical clauses), which represents a sufficient testing problem, on which it is possible to verify fairly and reliably the efficiency of the proposed tuning algorithm.

Variant 1 – simple GA with penalization
global minimum of the fitness (cost) function = 0

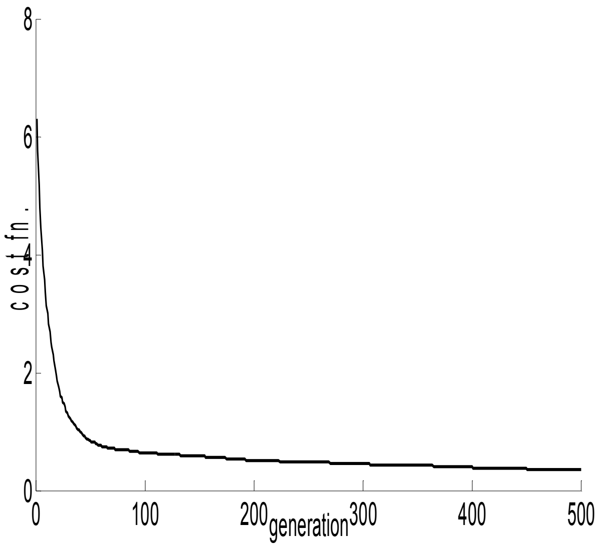

Figure 1. Algorithm variant 1, simple GA with penalization. Cost function evolution, average of 100 runs.
Global optimum is 0

Variant 3 – GA-SAW with penalization
global minimum of the fitness (cost) function = 0

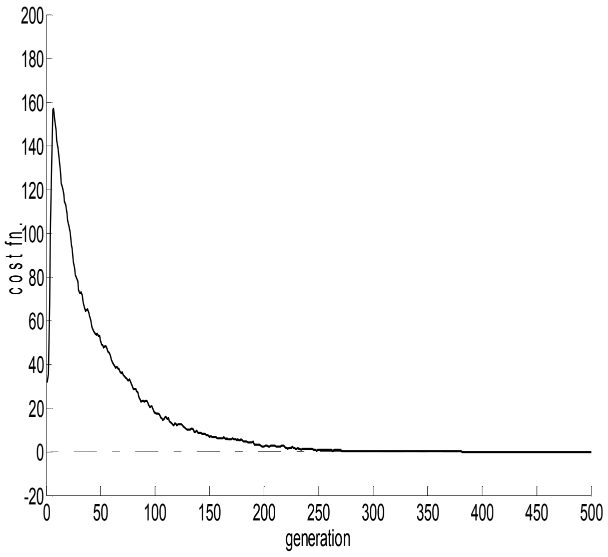

Figure 3. Algorithm variant 3, GA-SAW with penalization. Cost function evolution, average of 100 runs.
Global optimum is 0

Variant 2 – simple GA with remuneration
global minimum of the fitness (cost) function = -1030

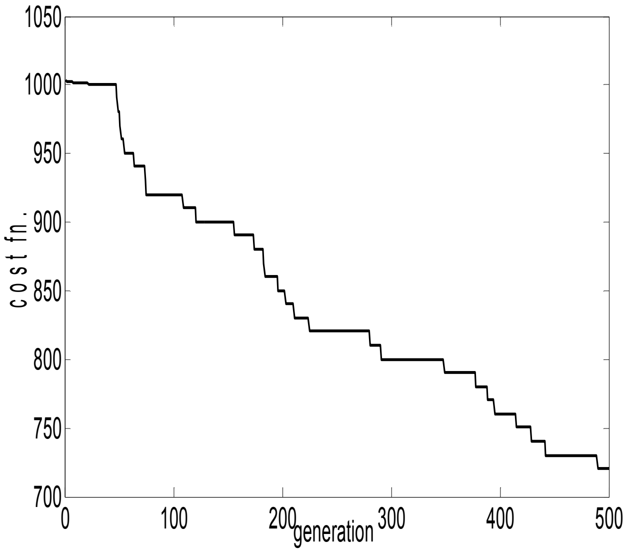

Figure 2. Algorithm variant 2, simple GA with remuneration. Cost function + 1030, average of 100 runs.
Global optimum is -1030

Variant 4 – GA-SAW with remuneration
global minimum of the fitness (cost) function = -1160

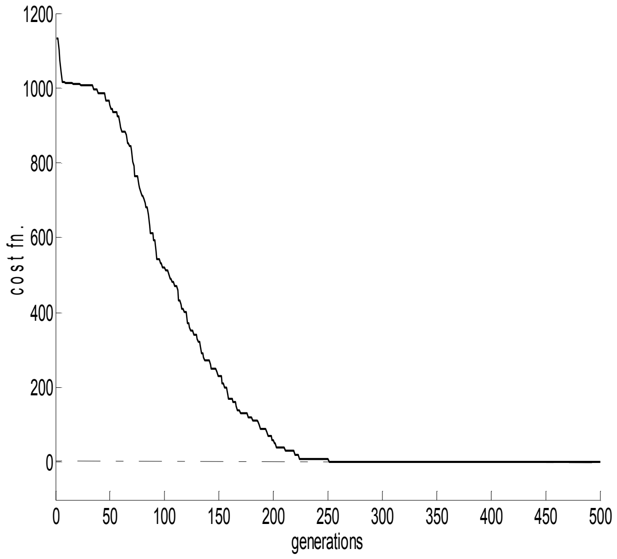

Figure 4. Algorithm variant 4, GA-SAW with remuneration. Cost function + 1160, average of 100 runs.
Global optimum is -1160

Based on the obtained results (see Table 1), we can conclude that among all the proposed variants, the most effective for solving the 3-SAT problem are the variants 3 and 4. The way of dealing with constraints (penalization, remuneration) does not play a significant role in this case.

TABLE I.

ASSESSING OF THE EFFICIENCY OF INDIVIDUAL VARIANTS OF GENETIC ALGORITHMS

Variant GA		Number of generations	Number of iterations	Number of achieving the global optimum per 100 iterations	Rate of achieving the global optimum (average per 100 iterations)
1	Simple GA – penalization	500	100	77	cannot be determined (only 77% performance)
2	Simple GA – remuneration	500	100	76	cannot be determined (only 76% performance)
3	GA – SAW with penalization	500	100	100	253. generation
4	GA – SAW with remuneration	500	100	100	253. generation

IV. CONCLUSION

A new method of solving the 3-SAT problem was proposed. This method is a genetic algorithm using suggested modified mechanism of stepwise adaptation of weights, which converges faster compared to conventional methods of penalization an objective function in GA.

The aim was not to compare this proposed tuning algorithm (variant 3 and variant 4) with previously proposed genetic / evolutionary methods based on SAW mechanism for solving the 3-SAT problem [3], [4], [5], [6], but to create it, and then to compare its efficiency with a simple GA.

The suggested approach is suitable for a number of practical applications in a field of logical inference, combinatorial problems (N-queens problem, sudoku, travelling salesman problem, etc.), image interpretation, diagnostics and/or planning and scheduling tasks.

Since each NP problem can be transformed to a problem SAT or 3-SAT,, the suggested approach is also applicable to other NP problems.

In general, we can state that in comparison with simple methods of penalization (remuneration) in genetic algorithms, the objective function on the basis of SAW mechanism has a good exploitation, especially with regard to its use in the context of increasing efficiency of solving some types NP and NP-complete problems.

REFERENCES

[1] T. J. Schaefer, " The complexity of satisfiability problems," in *STOC'78: Proceedings of the tenth annual ACM symposium on Theory of computing,* New York, NY, 1978, pp. 216-226.

[2] J. Hynek, *Genetic algorithms and genetic programming* (in Czech), Praha, CZ: Grada Publishing a.s., 2008.

[3] A. E. Eiben, and Jan K. van der Hauw, "Solving 3-SAT by GAs adapting constraint weights," in *Evolutionary Computation, 1997., IEEE International Conference on*, Indianapolis, IEEE, 1997, pp. 81-86.

[4] T. Bäck, A. E. Eiben, and M. E. Vink, "A superior evolutionary algorithm for 3-SAT," in *Evolutionary Programming VII.,* Springer Berlin Heidelberg, 1998, pp. 123-136.

[5] B. G. W. Craenen, and A. E. Eiben, "Stepwise adaption of weights with refinement and decay on constraint satisfaction problems," in *Proceedings of the genetic and evolutionary computation conference (GECCO 2001),* San Francisco, CA, 2001, pp. 291-298.

[6] R. Shalom, M. Avigal, and R. Unger, "A conflict based SAW method for constraint satisfaction problems" in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on,* Trondheim, 2009, pp. 373-380.

[7] K. Devlin, *Problémy pro třetí tisíciletí* (in Czech), Praha, CZ: Argo a Dokořán, 2005.

[8] V. Kvasnička et al., *Umelá inteligencia a kognitívna veda I.* (in Slovak), Bratislava, SVK: STU, 2009.

[9] M. Mach, *Evolučné algoritmy, prvky a princípy* (in Slovak), Košice, SVK: TU Košice, 2009.

[10] I. Sekaj, *Evolučné výpočty a ich využitie v praxi* (in Slovak), Bratislava, SVK: IRIS, 2005.