ITM Web of Conferences **6**, 02008 (2016)
DOI: 10.1051/itmconf/20160602008


# Use of genetic algorithms for solving problems of optimal cutting

Maxim Sergievskiy[1, 2, a] and Sergey Syroezhkin[1]

[1]*National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), 115409, Moscow, Russia*
[2]*Moscow Technological Institute, 119334 Moscow, Russia*

**Abstract.** Cutting and packing problem is one of the most common optimization problem. Even a small space or material savings allow obtaining substantial advantages on an industrial scale. This paper proposes the genetic algorithm to solve this problem. It includes multipoint operators of crossing, mutation and selection. To use these operators, the special encoding of cutting card is applied, that can be transformed to the real coordinates by using decoder. This is a block typed decoder, which substitution strategy is "first-fit". Efficiency of the solutions, obtained by the algorithm, depends on its parameters and on dimension of a task, but on average it decreases under the logarithmic law from dimension of a task. Temporary complexity of algorithm shows square dependence on the task dimension.

## 1 Introduction

Obtaining the effective solution of the packing rectangular objects problem in a two-dimensional container is one of the most important task of discrete optimization [1]. The necessity of finding efficient algorithms for this task is caused by its broad practical application in various branches of production. Even a small saving of resources while filling container and cutting material results in very tangible saving.

Complexity of this packing problem is caused by its belonging to a class of NP difficult problems of combinatory optimization, i.e. there are no methods and algorithms that can find the exact solution for polynomial time. The studied task is NP difficult in strong sense, because it contains NP difficult task as its subtask [2, 3].

There are many different methods for solving orthogonal packing problems: methods of mathematical programming, branch and bound methods, asymptotically exact methods, single-pass and multi-pass heuristics, metaheuristic, etc [1, 4]. Most of the methods are developed in two directions. Methods of the first direction searches for locally optimal solutions in some neighborhood of the initial acceptable solutions. They use decoders, which restore the packing map and calculate objective function value.

Another direction is development of the constructive methods. They deal with component-wise construction of the packaging, where method adds new components to the partially constructed pack while packing is not full. Offered in this paper method is the typical representative of the first class.

## 2 Problem statement

In this paper we consider the problem of packing rectangular objects in rectangular container (2 Dimensional Bin Packing Problem, 2DBP). Both value, the length ($L$) and the width ($W$) of the container are known. It is required to find the packing, which allows to pack the greatest number of items into one container.

Thus, the input data for the selected task is a set of values: $<W, L, m, \mathbf{w}, \mathbf{l}, e>$, where $W$, $L$ – the length and width of the rectangular container; $m$ – number of rectangular elements for packing; $\mathbf{w} = (w_1, w_2, ..., w_i, ..., w_m)$ – the vector of widths of elements; $\mathbf{l} = (l_1, l_2, ..., l_i, ..., l_m)$ – the vector of lengths of elements; $\varepsilon$ - the sign of possibility of changing elements orientation. Let's introduce a rectangular coordinate system, where $OX$ axis matches to horizontal edge of the object and the axis $OY$ - to vertical. Then task's solutions can be represented as follows: $<W, L, m, \mathbf{w}, \mathbf{l}, \mathbf{X}, \mathbf{Y}, \mathbf{E}>$, where $\mathbf{X}=(x_1, x_2, \ldots, x_i, \ldots, x_m)$, $\mathbf{Y}=(y_1, y_2,....y_i, ..., y_m)$ – vectors of the minimum coordinates of elements; $\mathbf{E}=(e_1, e_2,....e_i,..., e_m)$, $e_i$=1, if the element $i$ is rotated on 90 degrees, else $e_i$=0.

The acceptability of rectangular packing (RP) means the fulfillment of the following conditions:

- orthogonal arrangement of rectangles in the packing: for $(x_i , y_i)$, $i=1..m$ and any other vertex $(x_i^k , y_i^k)$ of rectangle $i$:

$$[(x_i^k = x_i) \vee (x_i^k = x_i + l_i)] \wedge [(y_i^k = y_i) \vee (y_i^k = y_i + w_i)]$$

- not overlapping of rectangles: for $i \neq j$; $i,j=1,..,m$:

$$[(x_i \geq x_j+l_j) \vee (x_j \geq x_i+l_i)] \vee [(y_i \geq y_j+w_j) \vee (y_j \geq y_i+w_i)]$$

[a] Corresponding author: sermax@yandex.ru

- not overlapping rectangles edges by elements: for every *i=1..m:*

$$(x_i \geq 0) \wedge (y_i \geq 0) \wedge (y_i + w_i \leq W) \wedge (y_i + w_i \leq L)$$

An objective function can be represented as the ratio of the area occupied by the packaged objects to the area of the container:

$$Q = (\Sigma S_i)/(W \cdot L) \rightarrow 1, \text{ for } i=1..m.$$

## 3 Methods and algorithms

Due to the task complexity, there are no methods to check every possible solution. Therefore, all existing approaches to this problem are to limit the set of checked solutions. Genetic algorithm, which applied in this paper, also uses this approach.

Genetic algorithm is an evolutionary algorithm that includes a group of metaheuristics for finding and selecting the best solutions [5, 6]. Its core contains basic concepts of the evolution theory: inheritance, mutation, selection. The main idea is to construct some set of optimization problem's solutions (population) and by getting through some transformations obtain new solutions instead of bad fit old ones. Storing information for multiple valid solutions at each iteration creates the effect of parallel computing, and increases the chances of finding the optimum.

To evaluate the effectiveness of the obtained solutions, we should convert it to a form that would allow us to calculate the objective function value. For this purpose, a special decoder is used. There are different approaches for individual decoding, but decoders of block type are the most widespread. They use simple strategies: "the next fit" (NF), "the first fit" (FF) and "best fit" (BF) [1–3, 7–14]. Based on the complexity of the strategies, in this work we apply the FF decoder. The decoder with the strategy "Substitution of first fit" (SubFF) at each step places the first matching object (its width does not exceed the width of the residual width of the block) in a partially filled block. If a suitable object is not found, it generates a free position of the next block and places there the first suitable object. Such algorithm (Sub(FF)) requires $O(m^2 \log m)$ of time.

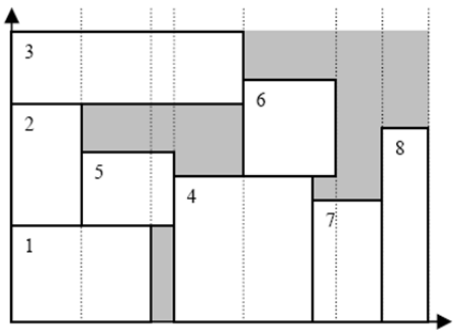


**Figure 1.** Example of SubFF executing for the sequence: {1,2,3,4,5,6,7,8}.

Decoder gets the priority list (replacement) of elements through input parameters and returns the packing - list of element's minimum coordinates. Unlike most of the existing block decoders, which use packing block structure for encoding rectangular, and based on already compiled pair of block structures (one packing) calculate the coordinates of the packing elements, our decoder operates differently. It fills a container in accordance with the priority sequence of rectangular objects, thus it operates like tiered decoders, however in the search for solutions it uses the classic division into blocks. This decoder was developed to combine the advantages of the both decoder types: blocked and tiered. Generally, the block type decoder uses spatial resources more effectively, while a decoder of tiered type has less temporal complexity because it uses just the identifications sequence for filling container instead of block structures.

In operation, the decoder stores information about free sections of the container, and when the element is installed in such section (it is installed to the lower left corner of the section), the decoder stores the coordinates of the lower left point of the element, and "forget" about the portion of the container, which was occupied. This is implemented through the list of block-levels. Block-level is a structure consisting of three coordinates: $X_{Left}$, $Y_{Begin}$, $Y_{End}$. $X_{Left}$ – the coordinate of the left edge of the block level (the same as the coordinate of the left edge of the block); $Y_{Begin}$ – coordinate of the lower edge of the block-level (the same as the y-coordinate of the lower boundary of the free area in the block); $Y_{End}$ – coordinate of the top edge of the block-level (the same as the y-coordinate of the upper boundary of the free area in the block). Figure 2 shows decoder's algorithm in more detail.

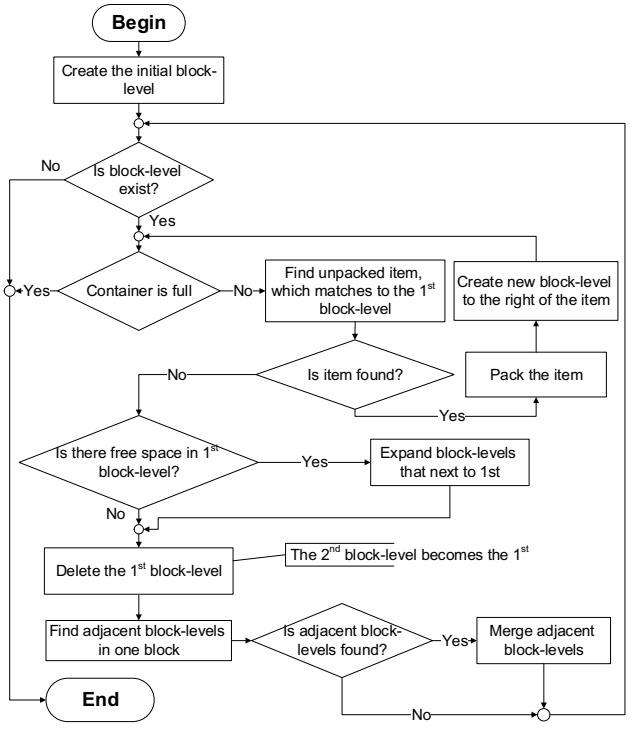


**Figure.2.** Block diagram of decoder

The solution in the population is a numerical sequence of object identifier for the package. The negative value of the element's identifier in the solution

means the element's rotation. Solution may contain several identical identifiers; it means that the packaging contains several identical elements. The number of repetitions of the elements is specified in the input parameters of the algorithm.

In the beginning, the algorithm randomly generates the initial population, that is, each solution in the population is created as a sequence of random numbers. Further, operators of crossing, mutation and selection change solutions at each iteration.

**Crossover operator** crosses pairs of solutions with a given probability. If task's dimension is large, solutions may contain long sequences of numbers. Single-point crossover operator is not effective in this case. For these circumstances, n-point crossover operator was developed.

By default, the crossing points are selected uniformly: the solution breaks into $n$ length-equal ranges (where $n$ – number of crossing points), and each crossing point gets out of the range corresponding to it. If crossing operator forms solution, in which the number of an element occurrences exceeds the maximum permissible value, operator will replace redundant elements with any other remained ones.

The developed crossing operator has several operating modes:

1) classic $n$-point. The number of crossing points $n$ is specified by the user. In this mode, each solution in the list of new solutions formed by the crossing of two old solutions at $n$ points.

2) gradually increasing number of crossing points. In this mode, after each crossing of two solutions, obtained solutions are stored in the list of new solutions. Thus, if the number of crossing point is $n$, the list of new solutions will contain new solutions formed by crossing of two old solutions in *1,2,3, ...,* and $n$ points. This mode allows you to get more different solutions, and it should increase the probability of finding more effective packing, however, this mode has a higher time complexity.

3) mixed mode with the specified ratio (default is 30%). This mode combines the first two modes, so you can get an expanded set of solutions by using the second mode, but only with a given probability, to reduce the computation time.

Obviously, to get a better solution, you need to check more different solutions. Therefore, the crossover operator has a special mode in which the count of crossing points is selected randomly and closer to the top of the solutions. Experimentally it was found that the crossing points located at the beginning of the solution, allow to obtain the better solutions. Probably, the reason is that the first packaged elements mainly define the configuration of the entire package.

**Mutation operator** makes random changes in solution, allowing you to avoid "getting stuck" in a local extremum. Implemented operator consists of two parts. The first part carries out a swap of the elements within solution, the second part performs rotation of elements by 90 degrees about its center in the plane of the container. For the same reasons as in the crossing operator, here we use multi-points mutation.

**Selection operator** is probably the most important part of genetic algorithm. It performs analysis of solutions, calculation of the objective function and the formation of a new population. It is formed on the basis of all individuals existing on the current iteration: solutions concluded in the current population and a set of the new solutions obtained after operators of crossing and a mutation. Every individual can take its place in a new population with sufficiently high probability $P_{sel}$. To ensure the impact of individuals, which are not included in the new population, replacement is made: $n$ solutions with the smallest value of the objective function of the new population is replaced by $n$ solutions with the smallest value of the objective function of the entire set of solutions to the current iteration (the most "bad" solutions). $n$ is a small number equal to about 10% of the number of solutions in the population.

At the end of the algorithm, the decoder restores the packaging by using a solution with the highest value of the objective function, and the algorithm terminates.

## 4 Results

In this paper, the dimension of the source data is a number of rectangular objects to be packed. This approach is most suitable for the tested algorithm, since just the number of rectangular objects has the value for it, not the sizes of objects. To assess the effectiveness of the genetic algorithm, experiments were performed on a special sets of input data. Their peculiarity lies in the fact that for each of them there is at least one solution with objective function value of 100%.

Based on recommendations from a variety of sources [12–14] and on the results of our experiments the "default values" for genetic algorithm parameters were set:

- the population size – 20
- number of iterations - 1000
- probability of the crossing operator performing - 30%
- probability of performing of the swap operation- 10%
- probability of performing of the turn operation - 50%.

For the given parameters values the algorithm showed the better results in most cases. To see this, here are some graphs (Figs. 3.1-3.4), which show different dependencies obtained in experiments.

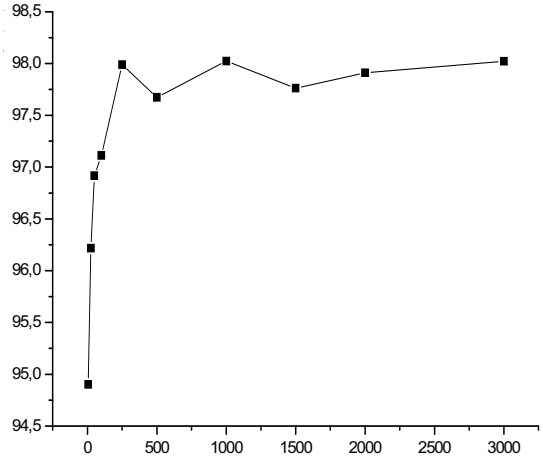


**Figure. 3.1.** The average value of the objective function on the iterations number

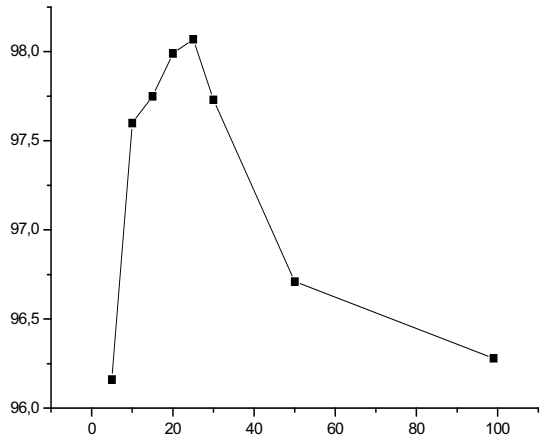


**Figure. 3.2.** The average value of the objective function on the size of the population

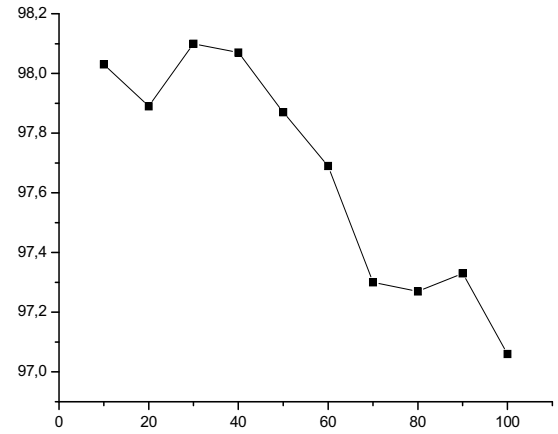


**Figure.3.3.** The average value of the objective function on swap probability

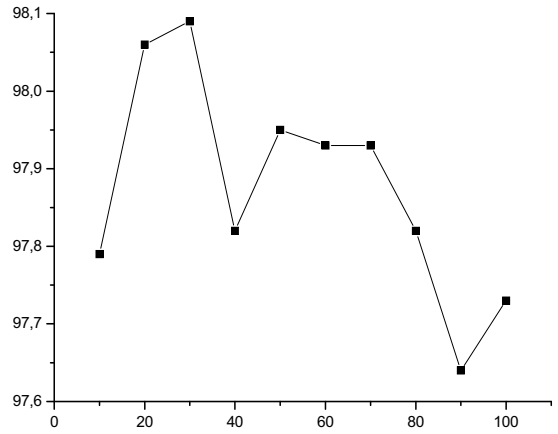


**Figure.3.4.** The average value of the objective function on crossing probability

Assessments of the solution's quality and searching time were carried out for two classes of containers: wide and long. Wide container is container with ratio of length to width is less than three, in other case container is long. Such division of containers into categories was required to test flexibility of algorithm. Figures 4.1–4.2 show results of experiments.

The average value of the objective function falls under the law of the natural logarithm with increasing of dimension of the problem, which corresponds to the theory. Also the results demonstrate the difference in the quality of the solutions obtained for the two categories of containers. The algorithm provides a slightly better solution for long containers than for the wide. Most likely, the reason of this result is decoder. Average search time of solution increases according to the law of $n^2$ with increasing of problem's dimension. This result is also close to theory, which states that genetic algorithms using the block decoder with FF strategies have a time complexity $O(m^2 \log m) \sim O(m^{2,5} \log m)$. As expected, the category of the container does not have a significant effect on the solution search time.

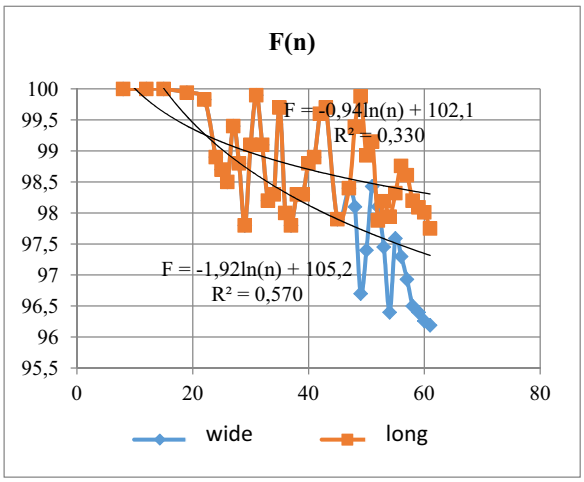


**Figure. 4.1.** Dependency of average value of the objective function on task dimension

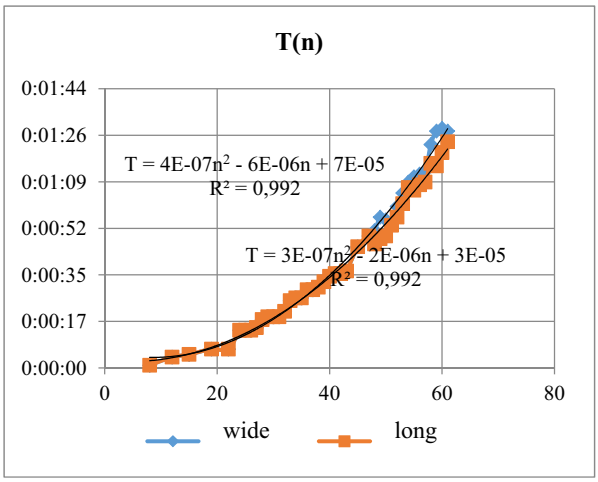


**Figure. 4.2.** Dependency of average value of searching time on task dimension

The developed algorithm uses multi-point crossover and mutation operators, as well as several other heuristics, the purpose of which is to increase a variety of solutions in the population. It would be interesting to determine their impact on the results. To do this, every used heuristic has been turned off, and the number of points of crossover and mutation set to *1*. To compare the performance of these two configurations of genetic algorithms: modified (multipoint and with heuristics) and simple (single-point), we construct the following dependencies:

- a graph of the difference between the average values of the objective functions, that we obtained by the

first and second algorithms on the task dimension (figure 5,1). The graph shows that the average value of the objective function increase will be 2,5%, if we use the modified algorithm.

- a graph of the difference between the average of searching time values on the task dimension (figure 5,2). The graph shows that the searching time is greatly higher in the case of modified algorithm.

The performance improvement, obtained by using modified genetic algorithm, may seem too insignificant in comparison with the increased time cost, and, as a result, the use of this algorithm may seem unjustified. But do not forget that graphs show the average value of the objective function. When using the modified algorithm globally-optimal solution were obtained in 91% of experiments, but in case of simple algorithm only in 56%.

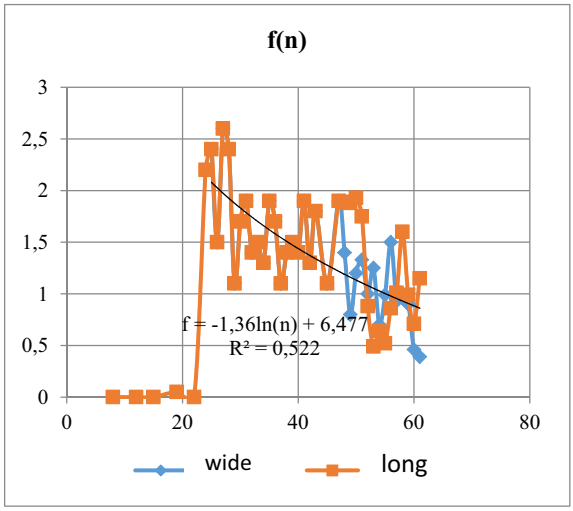


**Figure. 5.1.** Dependency of difference of average values of the objective function on task dimension

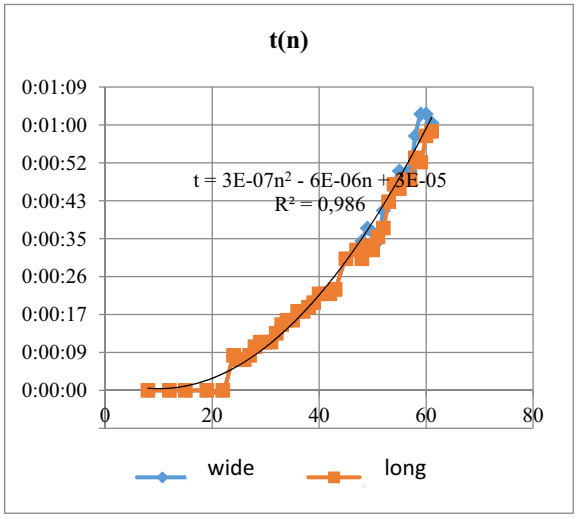


**Figure. 5.2.** Dependency of difference of average value of searching time on task dimension

## 5 Conclusion

Genetic algorithms show good results in solving problems of optimal cutting. With their help it is possible to obtain effective solutions in a short time, looking less than 1% of the total possible combinations of solutions.

The tests have confirmed that the use of multi-point crossover and mutation operators in the modified algorithm allows us to find the solution with the better objective function value and find globally-optimal solution in many cases. However, when choosing an algorithm for solving a specific problem, it also important take into account the available time resources.

## References

1. A.S. Philippova, *Methods of the solution of orthogonal packing problems on the basis of block structures technology* (Ufa, 2007) [In Rus]
2. R.I. Phayzrahmanov *Optimization of process of cutting of industrial materials by criterion of a minimum of material losses in the presence of technological restrictions* (Ufa, 2011) [In Rus]
3. A.I. Lipovetskiy, *Automatic design in mechanical engineering BSSR AN ITK*, **80** (1985) [In Rus]
4. A.Ph. Valeeva, *Constructive methods for solving orthogonal packing and cutting task* (Ufa, 2006) [In Rus]
5. L.A. Gladkov, V.V. Kureychik and V.M. Kureychik *Genetic algorithms* (Moscow, 2006) [In Rus]
6. V.V. Emelyianov, V.V. Kureychik and V. M. Kureychik, *Theory and practice of evolutionary modeling* (Moscow, 2003) [In Rus]
7. E. Hopper and B. Turton, European Journal of Operational Research, **128**, 34 (2001)
8. A.A. Petunin, E.A. Muhacheva and A.S. Philipova, Information technology, **1**, 28 (2008) [In Rus]
9. V.D. Avakumov. Information technology, **5**, 31 (2009) [In Rus]
10. E.A. Muhacheva, A.S. Muhacheva and A.V. Chiglincev, Information technology, **11**, 13 (1999) [In Rus]
11. N.N. Kuzyurin and A.I. Pospelov Diskr. Mat., **18**, 76 (2006)
12. A.A. Petunin, E.A. Muhacheva and A.S. Philippova, Information technology, **1**, 28 (2008) [In Rus]
13. D. Rutkovskaya, M. Pilinskiy and L. Rutkovskiy, *Neural networks, genetic algorithms and fuzzy systems* (Moscow, 2006)
14. A.S. Muhacheva, S.H. Kurelenkov, M.A. Smagin, R.R. Shirgazin, Information technology, **10**, 26 (2002) [In Rus]