Baltic J. Modern Computing, Vol. 5 (2017), No. 4, 351-361
http://dx.doi.org/10.22364/bjmc.2017.5.4.02

Solution of a Generalized Problem of Covering by means of a Genetic Algorithm

Natela ANANIASHVILI

Faculty of Exact and Natural Sciences,
I. Javakhishvili Tbilisi State University, Tbilisi, Georgia

Natela.Ananiashvili@tsu.ens.edu.ge

Abstract: An algorithm of solution of a generalized problem of covering is offered. Generalization is made on the basis of subsets with diverse values of minimal summary weight of the given set. In difference from a standard problem of covering, this problem implies selection of particular covering, when rank sum of covering columns (from the covering matrix) gives the particular vector and corresponding values of this vector exceeds not a singular vector, but pre-existing vector *b* (with non-singular components). Offered algorithm is based on a general genetic algorithm. It uses variable operator of non-standard mutation.

Keywords: minimal covering, genetic algorithm, cross-breeding, mutation

1. Introduction

A problem of covering is famous and many works have been devoted to its solution. However, because of its NP-complexity, solution of this problem in real time is often impossible. Therefore, it is still actual to develop efficient algorithms of solution of the standard problem of covering and its modifications.

A generalized problem of covering may be practically useful in solution of everyday problems. For instance, we may need to select location of service objects such that consumer has access to one of the several nearest objects, i.e. for him/her service object must be one or more. We may need to avoid queues in the banks, for instance, determine number of operators and many other practical problems may arise. If we use language of graphs, such problem may be used for solution of problem of multiple centers (Christofides, 1975).

A standard problem of covering is a NP-complex problem of combinatoric optimization (Garey and Johnson, 1979). Obviously, a generalized problem of covering is also NP-complex. Algorithms of precise solution of the standard problem of covering utilize techniques of branches and limits, time of their realization rapidly grows and it becomes impossible to get optimal solutions in real time, when scale of the problem increases (Christofides, 1975; Minieka, 1978). The standard and generalized problems of covering represent mathematical models of frequent and realistic everyday problems. Therefore, it is important to solve them in a real time. Heuristic algorithms are often used for solution of such problems, because they find almost optimal solutions in reasonable time intervals (Jacobs and Brusco, 1993). Approximate algorithms often use partial selection of covering sets and currently popular genetic algorithms (Holland, 1992) also belong to this class. Indeed, these algorithms often give near optimal solutions (Goldberg and Holland, 1988; Beasley and Jörnsten, 1992; Haupt and Haupt, 2004; Rutkovskaya et al., 2008).

One of the first and the best of approximate algorithm is offered by Chvatal (Chvatal, 1979). It solves the standard problem of covering in polynomial time. The work of Grossman and Wool (1997) considers the standard problem of covering, offers a heuristic algorithm and argues that it gives better results than previous techniques of solution. Lan et al. (2007) present meta-heuristic algorithm of standard covering. Ananiashvili (2015a) gives precise solutions of problems of the least division and covering and uses technique of branches and limits. According to this work, by means of compact insertion ("packaging") of columns of a covering matrix, a volume of random access memory (used for calculations) and number of operations is decreased approximately 32 times. Ananiashvili (2015b) offers approximate solution of the standard problem of covering by means of modified genetic algorithm and represents results of test problems. In their paper, Azar et al (2009) considered the general problem and gave a logarithmic approximation algorithm for it. In their paper, Bansal et al. (2010) improved their result and gave a simple randomized constant factor approximation algorithm for the generalized min-sum set cover problem. Lim et al. (2014) offer greedy algorithm of solution of problem of minimal covering. Yang and Leung (2003) and Umetani et al (2013) consider the generalized problem of weighted covering and require multiple covering of every element. For comparison, the authors use commercial software CPLEX (ILOG COMPLEX 7.0 – User's Manual, ILOG) which is used for solution of problems of integer programming and they use it for their own test problems. Motivation is that they don't know results of other authors. Yesipov and Muraviev (2014) offer two algorithms: additive and genetic. The efficiency of these algorithms is tested on randomly developed matrices and vector of weights is also filled randomly.

2. Formulation of a Problem

The generalized problem of minimal covering requires finding of a covering, when function is once again minimized

$$f(x) = \sum_{j=1}^{M} c_j \cdot x_j \tag{1}$$

with the following constraints:

$$\sum_{j=1}^{M} a_{ij} \cdot x_j \geq b_i,\ i = 1,\ldots,N,\ x_j \in \{0,1\},\ j = 1,\ldots,M, \tag{2}$$

where $a_{ij} \in \{0,1\},\ i = 1,\ldots,N,\ j = 1,\ldots,M$ are elements of the given matrix $A(N \times M)$ of coefficients, $c_j \geq 0,\ j = 1,...,M$ corresponds to weights or values of columns of matrix *A* and $b_i,\ i = 1,...,N$ are given natural numbers with small range (less than *N*). Note that in the standard problem of covering, every element of standard vector *b* equals to 1. $x_j \in \{0,1\},\ j = 1,\ldots,M$ are solutions of the given problem.

Vector R covers any vector *b*, if $R \geq b$, i.e. $R_i \geq b_i, i = 1,...,N$. In the standard problem of minimal covering, if logical rank sums of elements of selected columns of matrix *A* are equal or greater than 1, then it is enough to find covering. However, the generalized problem of minimal covering implies selection of such columns from the matrix *A*, when their rank arithmetic sum gives vector *R*, which covers the given non-singular vector *b*. In both problems, a function $f(x)$ of goal (1) is minimal for the selected columns.

3. The General Design of Genetic Algorithm

The proposed algorithm of solution of a generalized problem (1)-(2) of minimal covering implies general principles of genetic algorithms (Goldberg and Holland,1988). It can be described on the Fig. 1.

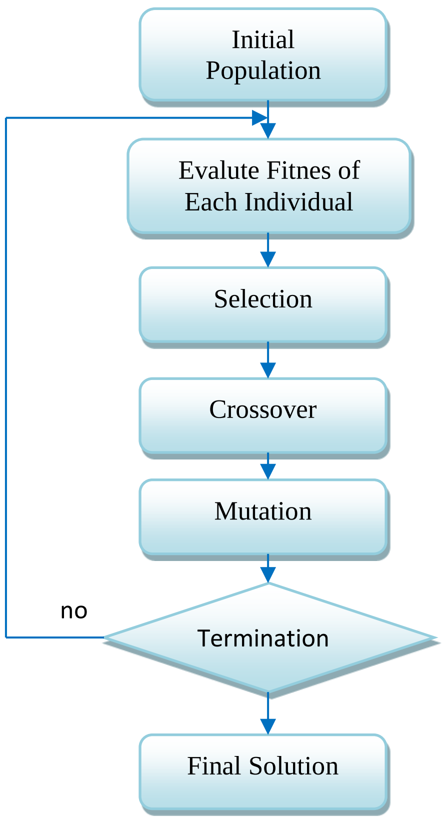

Fig. 1. General Genetic Algorithm

Let us consider a design that is used in the proposed algorithm. The first step of genetic algorithm is selection of scheme of encoding. Binary encoding is selected, i.e. every chromosome is n -dimension vector $x^k = \left(x_1^k, x_2^k, ..., x_n^k\right)$, where x_j^k th element is equal to 1, if column A_j is into kth chromosome and $x_j^k = 0$, if the same column is not in the kth chromosome. It means that $x^k, k = 1, 2, ..., s$ values are chromosomes that correspond to the individuals. x_j^k Values are genes that can be equal to 0 or 1. A_j column correspond to genotypes. Quantity *S* of individuals (chromosomes) in a population depends on the scale of problem.

First of all, let us select an initial population and then compute value of fitness function for chromosomes:

$$f_k = \sum_{j=1}^{n} c_j \cdot x_j^k, k = 1, 2, ..., s \tag{3}$$

Then we select the parents. When the problems are solved by means of genetic algorithms and the parents are selected, technique of roulette or some other techniques are used (Haupt and Haupt, 2004). In the proposed article, selection is made with the following technique. At the odd iteration two individuals are selected from the population that has minimal values of fitness function. At the even iteration we select one chromosome with minimal value of fitness function and another chromosome with maximal value of fitness function.In this way we avoid a rapid summation that brings us to the solution which is very different from minimal. Then we breed selected individuals with operator of one-point crossover. Individuals derived after crossover are mutated. Fixed value $\frac{1}{N}$ of mutation operator is often used in genetic algorithms, but we choose variable value of mutation, because purpose of crossover and mutation operators is to derive individuals that are different from individuals of population. Besides, when we approach local extreme, individuals are slightly different from each other. Therefore, we must use mutation operator with variable value to avoid selection of only kindred individuals. This value will depend on individual, as well as characteristics of genotype of these individuals, particularly quantity of 1s and 0s in the genotype.

From the chromosomes derived by means of crossover and mutation, we select the best one that has minimal value of function of fitness/usefulness. Then we look for individual with maximal fitness in the initial population and replace it with individual selected from offspring. The process of selection of parent chromosomes, crossover, mutation and replacement of parent individual with offspring individual is repeated until the end of iterations.

4. Algorithm

4.1. Formation of initial population

Let us assume that columns of matrix *A* are arranged according to growth of costs.

Quantity of individuals of population is denoted with *S*, *L* is the matrix of population with $s \times n$ size. (**Note:** In the algorithms the array indexes are specified in parentheses).

Algorithm 1: Formation of initial population.

Step 1. Let us take: $i = 1$; $t = 1$; $L(k, j) = 0, k = 1, 2...s, j = 1, 2, ..., n$; $R(l) = 0, l = 1, 2, ..., m$;

Let us form the first row of matrix L, i.e. the first chromosome of population in the following way:

Step 2. Let us find the first columns with numbers $j_1 = \{j_{i_1}, j_{i_2}, ..., j_{i_u}\}, u = b(i)$ of matrix *A* that covers r_ith node, i.e. $a(1, j_k) = 1, k = 1, 2, ..., u$ and take it into covering. Let us assume:

$$L(1, j_1) = 1, \; R(l) = R(l) + a(l, j_{i_1}) + a(l, j_{i_2}) + ... + a(l, j_{i_u}), \; l = 1, 2, ..., m \cdot$$

Step 3. Let us find number i_1 of the first element of *R*, for which: $R(i_1) < b_{i_1}$. Let us take $i = i_1$ and jump to step 2. If such element is absent, then jump to step 4.

The following rows of matrix *L*, i.e. remaining chromosomes of population are formed in the following way:

Step 4. Let us take $t = t + 1$. If $t > s$, then finish, otherwise take $R(l) = 0, l = 1, 2, ..., m$, $i = 1$;

Step 5. Let us find numbers $\{j_1, j_2, ..., j_{h_i}\}$ of columns of matrix A that cover r_i th node. Then randomly select some number j_u from $\{j_1, j_2, ..., j_{h_i}\}$, i.e. $a(1, j_u) = 1$ and take it into t th chromosome: $L(t, j_u) = 1$. Let us assume $R(l) = R(l) + a(l, j_u), l = 1, 2, ..., m$.

Step 6. Let us find number i_1 of the first element of *R* for which: $R(i_1) < b(i_1)$. If such element exists, then assume $i = i_1$ and jump to step 5. If such element does not exist, the jump to step 4.

When a population is formed with such technique, it is possible to find excessive genes in each population, i.e. after deleting some or several columns from each covering, set $R = \{r_1, r_2, ..., r_m\}$ can be covered again. We call *J* dead-end covering, if $J \setminus \{j\}$ is not covering for any $j \in J$. Therefore, it is necessary to delete excessive columns from each individual of given population and make them dead-end. For this purpose the following heuristic algorithm is used:

Algorithm 2: Deletion of excessive columns from *k*th individual.

Step 1. Let us define $j \in J$ for every $l = 1, 2, ..., m$ as $R(l) = h_l$, where h_l is quantity of columns that cover r_l th node in k th chromosome. It means that we must find quantity h_l of covering columns $\{j_1, j_2, ..., j_{h_l}\}$ in k th chromosome that is represented in k th row of matrix *L*, when $a(l, j_p) = 1, p = 1, 2, ..., h_l$.

Let us assume $i = 1$;

Step 2. Let us find minimal number i_1: $i \le i_1 \le m$, when $R(i_1) > b(i_1)$. If such number does not exist, then the process is finished.

Step 3. If every $R(l) - a(l, j_p) > b(l), l = 1, 2, ..., m$, then deletion of column j_p from k th chromosome is possible.

Let us assume $R(l) = R(l) - a(l, j_p), l = 1, 2, ..., m$ and $L(t, j_p) = 0$.

Step 4. Let us assume $p = p + 1$. If $p \le u$, then jump to step 3.

Step 5. Let us assume $i = i_1 + 1$. If $i \le m$ then jump to step 2, otherwise the process is finished.

By means of this algorithm we can delete excessive columns from k th individual $(k = 1, 2, ..., s)$ of population *L* and find dead-end coverings.

4.2. Crossover

After formation of population, we can compute values of fitness function for each chromosome by means of equation (3). We select two parent chromosomes on the basis of rule that is described in the third paragraph. Let us assume these chromosomes are:

$$x' = \left(x_1', x_2', ..., x_n'\right)$$

$$x'' = \left(x_1'', x_2'', ..., x_n''\right)$$

Let us define location of crossover with random value $k \in \{1, 2, ..., n-1\}$. After crossover we'll get offspring chromsomes:

$$x^{ch_1} = \left(x_1', x_2', ..., x_k', x_{k+1}'', ..., x_n''\right)$$

$$x^{ch_2} = \left(x_1'', x_2'', ..., x_k'', x_{k+1}', ..., x_n'\right)$$

After such crossover offspring chromosomes may not cover $R = \{r_1, r_2, ..., r_m\}$. So, it is necessary to find and add covering sets for uncovered nodes of those chromosomes. We propose algorithm that guarantees this process:

Algorithm 3: Addition of subsets that cover uncovered nodes to chromosomes.
For legibility, chromosome x^{ch_1} is selected.

Step 1. Let us find numbers of non-zero elements in chromosome x^{ch_1}. $\{j_1, j_2, ..., j_h\}$ are numbers of corresponding covering sets of this chromosome. To satisfy condition $l = 1, 2, ..., m$ for every l th row of columns $\{j_1, j_2, ..., j_h\}$ of covering matrix A, let us count number k_l of columns that cover r_l th node, i.e. when $a\left(l, j_p\right)$, $p = 1, 2, ..., k_l$ and $\left\{j_{i_1}, j_{i_2}, ..., j_{i_{k_l}}\right\} \subset \{j_1, j_2, ..., j_h\}$. Let us assume $R(l) = k_l$ and $i = 1$.

Step 2. Let us find minimal number i_1: $i \le i_1 \le m$, for which $R(i_1) < b(i_1)$. If every element of $R(i) \ge b(i)$, , for every i th $i = 1, 2, ..., m$ then the process is finished.

Step 3. Let us find numbers $\{j_1, j_2, ..., j_\upsilon\}$ of those columns of matrix A that cover r_{i_1} node. Let us select randomly any number j_u from $\{j_1, j_2, ..., j_\upsilon\}$, i.e. $a(l, j_u) = 1$; take column j_u into the current chromosome: $x^{ch_1}\left(j_u\right) = 1$.
Let us assume $R(l) = R(l) + a(l, j_u)$, $l = 1, 2, ..., m$.

Step 4. Let us assume $i = i_1 + 1$ If $i \le m$ then jump to step 2, otherwise the process is finished.

By means of described algorithm chromosomes x^{ch_1} and x^{ch_2} will cover again set $R = \{r_1, r_2, ..., r_m\}$. We offer the fragment of a program that enables us to make x^{ch_1} covering:

```
t=true;
  while t
        k0=numel(find(R<b,1,'first'));
      if k0==0
          break
      else
          k=find(R<b,1,'first');
      end
        r=find(A(k,:)==1,1,'first');
for ik=1:M
        if (A(k,ik)==1)&&(xch1(ik)~=1)
            r=ik; break;
        end
   end

        R=R+A(:,r);
        xch1(r)=1;
  end
```

Note: `find(R<=b,1,'first')` searches the first *k*th element of vector *R*, where $R(k) \le b(k)$. If we do not have such element in vector *R*, then the result will be empty. Function *numel* determines the length of an array. If an argument of function *numel* is empty array, then it will return 0.

After crossover offspring chromosomes are mutated.

4.3. Mutation

Let us select some number *k* from set $k \in \{1,2,...,n\}$ for offspring chromosomes x^{ch_1} and x^{ch_2} and mutate *k*th gene of this chromosome. Note that specification of gene in this problem is its weight, as well is number of 1s and 0s in corresponding genotype. Let us calculate quantity $p_1(j)$ of 1s and quantity $p_0(j)$ of 0s for each *j*th columns of matrix *A*, when $j=1,2,...,n$. Besides, let us calculate the following values for each column:

$$E(j) = -p_0(j)\cdot\log\left(p_0(j)\right) - p_1(j)\cdot\log\left(p_1(j)\right),\ \ j=1,2,...,n$$

then develop vector of probability of mutation:

$$pmt(j) = \frac{\dfrac{1}{E(j)}}{\sum_{u=1}^{n}\dfrac{1}{E(j)}}\ ,\ \ j=1,2,...,n$$

Let us select gene τ for mutation, where τ is random normalized integer from range $1 \le \tau \le n$. Let us define condition of mutation:

1. If the following conditions are met: $pmt(\tau) > \frac{1}{n}$, $x^{ch_1}(\tau) = 0$ and $p_0(\tau) > p_1(\tau)$, then gene τ mutates and we'll get $x^{ch_1}(\tau) = 0$.
2. If the following conditions are met: $pmt(\tau) > \frac{1}{n}$, $x^{ch_1}(\tau) = 0$ and $p_0(\tau) < p_1(\tau)$, then gene τ mutates and we'll get $x^{ch_1}(\tau) = 0$.

Similarly we can mutate the second chromosome x^{ch_2} . As in the case of crossover, after mutation offspring chromosomes may not cover set $R = \{r_1, r_2, ..., r_m\}$. Let us use above mentioned algorithm #3 for x^{ch_1} and x^{ch_2} . Then add columns that cover every uncovered node. Chromosomes derived after addition may not represent dead-end coverings. Let us use algorithm #2 and delete excessive columns. This process is repeated until depletion of iterations.

5. The results of experiment

The corresponding algorithm is realized with Matlab. For experiments we used standard test problems of covering from Or-Library, but because these problems imply that $b_i = 1$, $i = 1, \ldots, N$, we used random numbers $b_i = \{1,2,3\}$, $i \in = 1, \ldots, N$. I developed AMPL-models for the same problems and solved them by means of NEOS Server (solvers Gurobi and CPLEX). Table 1 shows the results of solution. Fmin denotes the best value given by the above-mentioned genetic algorithm and Fgurobi denotes the value given by solver Gurobi. For those problems, where the dimensions of coverage matrix are 100 × 1000, 200 × 1000 or 200 × 2000, 400 × 4000, 500 × 5000 , the results obtained with proposed algorithm and the Gurobi solver are closed to each other. I have got better results with my algorithm compared to GUROBI-solver for scpd1(D,1) problem. For the dimensions 1000 × 10000, I have got results with my algorithm, though Gurobi solver did not get the result.

Table 1. The results of experiments for a generalized problem of covering

Problem	Fmin	Number of iterations	Time of calculation (seconds)	Fgurobi	Time of calculation by Gurobi(seconds)
4,1	1359	877	9.5218	1206	0.000546
4,2	1601	965	11.3247	1448	0.000571
4,3	1631	535	7.4521	1417	0.0005
5,1	841	5500	123.5824	718	0.005
5,2	851	575	10.7950	775	0.000625
5,3	959	895	7.1852	847	0.000584
6,1	360	575	10.9263	330	0.000595
6,2	381	775	10.0854	346	0.000576
6,3	440	9575	115.4195	375	0.000622
A,1	741	1575	50.0961	655	0.000584
B,1	189	874	7.5625	167	0.000533
C,1	771	885	8.2353	598	0.000633
D,1	150	875	8.9659	154	0.000569

E,1	10	1278	4.0702	9	0.000589
F,1	27	875	10.2762	25	0.000564
G,1	447	1876	524.7170	-	-
H,1	149	2276	864.1087	-	-

Fig. 2 shows the diagram for the problem scpd1. We compared $b_i, i=1,...,N$ values of selected constraints with calculated of $R(i), i=1,...,N$ values of covering.

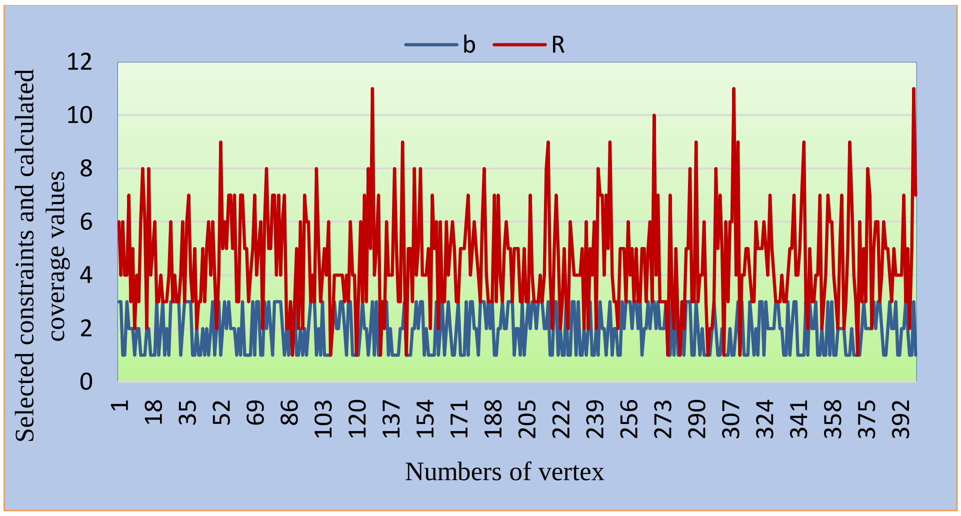

Fig. 2. Comparative diagram of constraints and calculated values of the problem scpd1.

If in the problem (1)-(2) we assume that $b_i = 1, i = 1,...,N$, then we get the standard problem of covering. In the offered algorithm, let us take $b_i = 1, i = 1,...,N$ and use the test problems from Or-Library. Table 2 gives the results. For comparison, in the column (Minimal value f*) of the table, known optimal solutions of these problems are given and the column (Fmin) shows the results of the offered algorithm. These results are quite close to minimal and often margin of error does not exceed 2-5%. Sometimes the results are even precise.

Table 2. The results of experiments for a standart problem of covering.

Problem	Minimal value f*	Fmin	Number of iterations	Time of calculation (seconds)
4,1	429	432	850	5.748834
4,2	512	523	28102	196.452521
4,3	516	521	678	14.437801
5,1	253	257	28124	204.252417
5,2	302	307	23362	108.583021

5,3	226	232	3259	18.085624
6,1	138	145	156	5.745222
6,2	146	150	35250	239.397738
6,3	145	148	5528	123.320145
A,1	253	255	1250	20.120029
B,1	69	76	15566	278.325041
C,1	227	233	36312	886.189983
D,1	60	60	35250	859.387968
E,1	29	29	650	37.089270
F,1	14	16	550	38.747741
G,1	179	185	35250	4163.728506
H,1	64	69	5250	706.016420

On the basis of experiments, we may conclude that offered algorithm operates reasonably efficiently. For test the standard computer was used with specifications Intel(R) Pentium (R) Dual CPU E2220 2.40 GHz, 2.00 GB of RAM.

6. Conclusion

The presented work considers the generalized multicover problem (1-2), which is solved by means of genetic algorithm. Parents are selected according to the rule of proportional selection and correspondingly to the probability determined for each individual. Probabilities are calculated on the basis of fitness. The essence of operator of new cross-breeding is that genes of descendant individuals are determined on the basis of fitness of parent individuals and frequency of recurrence of these genes in a population. In addition, we use variable operator of mutation. After cross-breeding and mutation, descendant new individuals may not be covering anymore and therefore, we take into account a procedure which guarantees that new set is covering. I think that the algorithm and its results which are represented in this work, will be interesting for researchers of this field.

References

Ananiashvili, N. (2015a). Solution of Minimal Set Partition and Set Covering Problems, *Bull. Georg. Natl. Acad. Sci* 9.1 (2015).

Ananiashvili, N. (2015b). Solution of Problem of Set Covering by Means of Genetic Algorithm, *Computer Science & Telecommunications* 46.2 (2015).

Azar, Y., Gamzu, I., Yin, X. (2009), Multiple intents re-ranking. In STOC '09: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pp. 669-678, New York, NY, USA.

Bansal, N., Gupta, A., Krishnaswamy, R. (2010). A constant factor approximation algorithm for generalized min-sum set cover. In Proceedings of the 21st annual ACM-SIAM symposium on Discrete algorithms, pp. 1539-1545.

Beasley, J. E. (1990). OR-Library: "Distributing Test Problems by Electronic Mail,The Journal of the Operational Research Society". Vol. 41, No. 11 (Nov., 1990), pp. 1069-1072.

Beasley, J. E., Jörnsten, K.. (1992). "Enhancing an algorithm for set covering problems." *European Journal of Operational Research* 58.2 (1992): 293-300.

Christofides, N. (1975). G. Theory. *An algorithmic approach*. Academic Press, Inc., New York, 1975.

Chvatal, V, (1979), A greedy heuristic for the set-covering problem. Mathematics of Oper. Res., vol. 4, no.3, pp. 233-235.

Garey, M. R., Johnson, D. S. (1979). "A Guide to the Theory of NP-Completeness." *WH Freemann, New York* 70 (1979).

Goldberg, D. E., Holland J. H. (1988). "Genetic algorithms and machine learning." *Machine learning* 3.2 (1988): 95-99.

Grossman, T., Wool, A. (1997), "Computational experience with approximation algorithms for the set covering problem." European Journal of Operational Research 101.1, 81-92.

Haupt, R. L., Haupt, S. E. (2004). "The continuous genetic algorithm." *Practical Genetic Algorithms, Second Edition* (2004): 51-66.

Holland, J. (1992). "Holland. Genetic algorithms." *Scientific American* 267.1: 44-50.

ILOG CPLEX 7.0-User’s Manual, ILOG, (2000), Mountain View, California.

Jacobs, L. W., Brusco M. J. (1993). "A simulated annealing-based heuristic for the set-covering problem." *Proc.“Operations Management and Information Systems Department,” Northern Illinois University, Deklab, IL 60115, USA*.

Lan, G., DePuy G. W., Whitehouse G. E.(2007), An effective and simple heuristic for the set covering problem, European Journal of Operational Research 176,1387–1403.

Lim, C.L., Moffat, A., Wirth, A. (2014), Lazy and Eager Approaches for the Set Cover Problem, Proceedings of the Thirty-Seventh Australasian Computer Science Conference, Auckland, New Zealand

Minieka E.,(1978). *Optimization Algorithms For Networks And Graphs,* New York : M. Dekker, c1978.

Rutkovskaya, D., Pilinskiy, M. Rutkovskiy, L. (2008), Neuron nets, genetic algorithms and fuzzy systems (In Russian: Рутковская Д.,Пилинский М., Рутковский Л., "Нейронные сети, генетические алгоритмы и нечеткие системы". М.: *М.: Горячая линия–Телеком* (2008)).

Umetani, S., Arakawa, M., Yagiura, M., (2013), A heuristic algorithm for the set multicover problem with generalized upper bound constraints. In Proceedings of Learning and Intelligent Optimization Conference (LION), 75–80.

Yang, J., Leung, J.Y.T., (2005). A generalization of the weighted set covering problem. Naval Research Logistics 52(2) 142–149.

Yesipov B.A., Muraviev V.V., (2014), Study of Algorithms for Solving the Generalized Problem of the Minimal Covering (In Russian: Б.А.Есипов, В.В.Муравьев, (2014), Исследование агоритмов решеня обобщенной задачи о минимальном покрытии, Известия Самарского научного центра Российской академии наук, т. 16, №4(2))

Received May 9, 2017, revised November 6, 2017, accepted November 14, 2017