On Genetic Algorithm with Non-Binary Representation for Set Covering Problem

Anton V. Eremeev, Omsk Branch of Sobolev Institute of Mathematics, RAS

Summary: A genetic algorithm (GA) for the set covering problem is developed using a non-binary solution representation. The paper describes the GA and the new problem-specific operators of crossover and local improvement that we have implemented in it. Usage of these operators leads to a substantial improvement of the algorithm. Finally, some promising results of computation experiments and comparison to other algorithms are reported. In addition to what was presented in [9] here we describe some new experiments with the Steiner triple systems.

1. Introduction

The set covering problem (SCP) can be stated as follows. *Consider a set* $M = \{1, \ldots, m\}$ *and the subsets* $M_j \subseteq M$*, where* $j \in N = \{1, ..., n\}$*. A subset* $J \subseteq N$ *is a cover of* M *if* $\bigcup_{j \in J} M_j = M$*. For each* M_j*, a positive cost* c_j *is assigned. The SCP is to find a cover of minimum summary cost.*
Sometimes it is more convenient to consider SCP as an integer linear programming problem:

$$\min\{cx : Ax \geq e, x \in \{0,1\}^n\}, \tag{1}$$

where A is an $m \times n$ matrix of 0s and 1s, c is an n-vector of costs, e is the m-vector of 1s, and $a_{ij} = 1$, if and only if $i \in M_j$. Using this formulation one can regard SCP as a problem of optimal covering of rows by the columns in matrix A. In this paper we assume that the columns are ordered according to nondecreasing values of c_j-s and if two columns have equal costs then the one which covers more rows stands first.

Different algorithms have been proposed for the exact solving of SCP using branch and bound, cutting planes [1], L-class enumeration [9,13] and other techniques. However, SCP is an NP-hard problem and usually application of exact algorithms to the large-scale instances is very time-consuming.

A number of heuristic algorithms are developed for approximate solving the problem within relatively short running time. The solutions of good quality may be obtained using Lagrangian relaxation heuristics [4], neural networks [11] or metaheuristics such as GA [3].

2. Genetic Algorithm

GA is a probabilistic search method based on biological analogy of selective breeding in nature, used for the search of near-optimal solutions. The basic components of GA are a *population* of *individuals* and random operators that are introduced to model mutation and crossover in nature. An individual is a pair of *genotype* g and *phenotype* $x(g)$ corresponding to a search point in the space of solutions D to a given optimization problem. Here g is a fixed length string of symbols (called *genes*) from some alphabet $\mathbf{A}$. The function $x(g)$ maps g to its phenotype $x(g) \in D$, thus defining a *representation* of solutions in GA. The search is guided by the results of evaluating the *fitness* function $\Phi(g) = \phi(f(x(g)))$ for each genotype in population (here $f : D \to R$ is the goal function, and $\phi : R \to R$ is a monotone function which is often introduced to intensify the GA search).

The genotypes of the initial population are randomly generated according to some a priori defined probability distribution. The GAs discussed in this paper implement the *steady-state* reproductive strategy [6], in which a single new individual is generated at each iteration. The genotype of a new individual replaces the least fit one in the population, thus the population size s remains constant during the run of GA. Usually the stopping criterion is the limit on the number of iterations t_{max}. The best solution found is returned as the final result when GA stops.

The construction of a new genotype (offspring) starts with choosing a pair of parents from the current population with a help of probabilistic *selection* operator. Then the *crossover* operator replaces some genes of one parent with genes of the other. The resulting string undergoes *mutation*, where a randomly chosen subset of genes is substituted with random symbols from $\mathbf{A}$. In this paper we consider a GA where a gene is mutated with a fixed probability p_m. The settings of p_m, s and the probability of crossover usage are very important for efficient work of GA. Usually they are chosen empirically for the specific class of optimization problems. In many successful implementations the genotype is processed by a heuristic of local improvement or by a repair operator which makes the solution feasible before it is added to a population.

An efficient GA for SCP was proposed by Beasley and Chu [3]. The algorithm uses a convenient binary representation of solutions as n-bit strings (i.e. $\mathbf{A} = \{0, 1\}$). A column j is included into phenotype $x(g)$ if and only if the bit j in genotype g equals 1. The only inconvenience with this representation is that one has to deal with infeasibile phenotypes that may be produced by mutation and crossover. A greedy-like heuristic is used in [3] to repair the infeasible solutions before adding them to population. In the next section we describe a GA, which uses a similar reproduction strategy and provides only feasible solutions due to a non-binary representation.

2.1 General Scheme of the Non-Binary Genetic Algorithm

Let's denote the set of columns that cover the row i by $N_i = \{j \in N : a_{ij} = 1\}$. The non-binary GA (NBGA) proposed in this paper is based on a non-binary representation where the genotype consists of m genes $g^{(1)}, g^{(2)}, \ldots, g^{(m)}$, such that $g^{(i)} \in N_i$, $i = 1, 2, \ldots, m$. Here each gene contains a column index that is assigned to cover the row.

In this representation x maps a genotype g to the phenotype $x(g) \in \{0, 1\}^n$, where ones correspond to the columns present in genes of g. Obviously, it is a feasible solution to the problem (1) and no repair operator is needed. However, as it was noted in [3], the implementation of GA with the non-binary representation requires an additional operator to eliminate the redundant columns from the solution after crossover and mutation. For this local improvement we use greedy heuristics that find approximate solutions to a corresponding reduced version P_g of the given SCP. The reduced problem P_g has a matrix that consists of the columns of A represented in genes of g.

An improved genotype is added to the population only if there are no individuals with the same phenotype in it yet. The genotypes of the initial population are generated independently and each gene is uniformly distributed over the set of indices of columns covering the gene.

Let's consider the current population on iteration t as a vector of genotypes of its individuals

$G = (g_1, g_2, \ldots, g_s)$. Then the fitness function on iteration t is defined as

$$\Phi_t(g) = cx(g_{l(t)}) - cx(g),$$

where $l(t)$ is the index of the individual of the largest cover cost in the population on iteration t. The selection operator in our GA implements the proportional selection scheme, where the probability to choose the k-th individual ($k = 1, 2, \ldots, s$) equals

$$p(g_k) = \Phi_t(g_k) \left(\sum_{l=1}^{s} \Phi_t(g_l) \right)^{-1}.$$

The general scheme of the NBGA

1. While the initial population is not complete do
   1.1. Generate a random genotype g.
   1.2. Apply the column elimination procedure *Prime* to g and add g to the population.
2. For t:=1 to t_{max} do
   2.1. Choose the parent genotypes g_u, g_v with the help of the proportional selection.
   2.2. Produce an offspring g from g_u and g_v using a crossover operator.
   2.3. Mutate each gene of g with probability p_m.
   2.4. Obtain genotype g', applying column elimination procedures *Greedy* and *Dual Greedy* to g.
   2.5. If there are no individuals in G with the phenotype $x(g')$, then
      2.5.1. substitute the least fit genotype in G by g',
      2.5.2. otherwise substitute the least fit genotype in G by g.

The operators of crossover, mutation and the local improvement procedures *Prime*, *Greedy* and *Dual Greedy* will be described in detail in the following sections. The condition in step 2.5 is introduced to prevent undesirable premature convergence of the population to local optima. Thus, the population in this GA also works as a tabu list of tabu search heuristics. In steps 1.2 and 2.5.2 an individual is added without any tests because duplicate solutions are very rare in these cases.

2.2 Crossover and Mutation

In this section we propose a crossover operator which is specially designed for SCP. Here we refer to it as *LP-crossover*. The goal of this operator is to find the best possible combination of the given parent genotypes g_u and g_v, if it is possible without extensive computations.

Let's consider a problem of optimal crossover P_{oc}, which is a reduced version of the initial SCP but with the covering subsets restricted by the set of indices $N' = \{j : x(g_u)_j = 1\} \cup \{j : x(g_v)_j = 1\}$.

First, in *LP-crossover* a trivial reduction is applied to P_{oc}. Denote $S = \{i \in M : |N_i \cap N'| = 1\}$. Then each row $i \in S$ may be covered by a single column $j(i)$ in P_{oc}. Let's call $Q = \{j : j = j(i), i \in S\}$ a set of fixed columns. For each $i \in \bigcup_{j \in Q} M_j$, one of the columns that cover the gene $g^{(i)}$ in Q is assigned. Let's refer to these genes as fixed too. As a result of the reduction we obtain a new subproblem P_r, that consists of the rows and columns that were not fixed during this procedure.

Second, the dual simplex method is used to solve the linear relaxation of P_r, where the Boolean constraints are replaced with the conditions $x_j \geq 0$ for all j. If the solution x' obtained by simplex method turns out to be integer, then it is used to complete the genotype g by an assignment of the non-fixed genes that corresponds to the solution x'.

To avoid time-consuming computations we don't start the simplex method if the number of rows in P_r exceeds a certain threshold μ (we use $\mu = 150$). If this is the case or if the number of simplex iterations exceeds its limit (equal to 300 in our experiments), or if the solution x' is fractional, then *LP-crossover* returns an unchanged genotype g_u.

We designed the mutation operator similar to the proportional selection procedure. Suppose, i-th gene is to be mutated. Then the probability to assign a column $j \in N_i$ to it equals

$$p_i(j) = \frac{1}{c_j} \left(\sum_{k \in N_i} \frac{1}{c_k} \right)^{-1}.$$

2.3 Redundant columns elimination

We use three greedy-type heuristics to exclude the redundant columns from the solution.

The most simple and fast heuristic starts with a given cover and discards the columns in decreasing order of indices. A column is discarded only if the remaining solution is still a cover. We refer to this algorithm as *Prime*. It finds a prime cover, i.e. a solution without redundant columns.

The second heuristic is the well-known *Greedy* algorithm [5]. It sequentially builds a cover to the problem P_g by adding at every step a column maximizing the ratio w_j/c_j, where w_j is the number of unsatisfied inequalities that use the variable x_j. Ties are broken lexicographically (i.e. the variable with the smallest index is picked first). This algorithm might find a solution which is not minimal, therefore *Prime* is run after it to eliminate the redundant columns.

The next heuristic we shall call the *Dual Greedy* algorithm. It combines the successive columns discarding of *Prime* and the adaptive columns pricing similar to *Greedy*. Let's denote the set of columns in the subproblem P_g by $N' := \{j \in N : x(g)_j = 1\}$. A cover J is obtained as follows.

The Dual Greedy Algorithm

1. Set $N_i' := N_i \cap N'$ for all $i = 1, ..., m$, $M' := M$, $J' := N'$, $J := \emptyset$.
2. While $J' \neq \emptyset$ do

    2.1. If there is $i \in M'$ such that $|N_i'| = 1$, then

    using the $j \in N_i'$, set $J := J \cup \{j\}$, $M' := M' \backslash (M_j \cap M')$.

    Otherwise choose $j \in J'$ such that $c_j |M_j \cap M'|^{-1} \geq c_k |M_k \cap M'|^{-1}$ for all $k \in J'$.

    2.2. Set $J' := J' \backslash \{j\}$, $N_i' := N_i' \backslash \{j\}$ for all $i \in M_j$.

In step 1.2 of NBGA we use the *Prime* algorithm. The choice is motivated by our computation experiments, which show that the usage of more complex algorithms like *Greedy* and *Dual Greedy* is not justified at this stage. Firstly, they are too time-consuming compared to *Prime*, secondly, for some problems they construct the initial populations with individuals of a similar structure which is locally optimal. This bias might hinder the GA from attaining the global optimum.

Greedy and *Dual Greedy* are used in step 2.4 of NBGA. Both heuristics are called and the best of the two solutions x' is accepted to construct a new genotype g', which differs from g in the genes that represent the columns absent in x'. These genes are switched in g' to other columns that are still in x'.

3. Computation Experience

The NBGA was implemented in Borland Pascal on 100 MHz Pentium and tested on the OR-Library [2] benchmark problem sets *4-6, A-H*, and a set of combinatorial problems *CLR*. The sets *4-6* and *A-H* consist of non-unicost problems with c_j taken from [1,100]. Before solving these problems we apply a core-refining reduction which is similar to the approach used in [3]. The reduction keeps only the columns belonging to the set $N_{core} = \bigcup_{i=1}^{m} \alpha_i^{10}$. Here α_i^r is the set of the least r indices in $N_i, i = 1, \ldots, m$. In our experiments this reduction permitted the NBGA at least once out of 10 trials to find the optimal solutions to all problems of *4-6, A-D* and the best-known solutions to the problems in *E-H* except for H1 and H2. Using the reduction with $r = 5$, the NBGA attained the best-known solution to the problem H2 also. The dimensions of problems before and after the core-refining reduction as well as their density (i.e. the percentage of ones in the original SCP matrix), and the number of problems in set are given in Table 1.

Name	*4*	*5*	*6*	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*
Rows (m)	200	200	200	300	300	400	400	500	500	1000	1000
Columns (n)	1000	2000	1000	3000	3000	4000	4000	5000	5000	10000	10000
Rows in core	640-696	628-672	293-315	694-724	312-329	720-737	328-341	171-180	91-96	808 - 817	353-370
Density (%)	2	2	5	2	5	2	5	10	20	2	5
Num. of probl.	10	10	5	5	5	5	5	5	5	5	5

Table 1.

Tables 2 and 3 compare the performance of the NBGA to the performance of GA by Beasley and Chu [3] (BC), and the Lagrangian heuristic by Caprara, Fischetti and Toth [4] (CFT). In our experiments 10 trials of GA were made for each problem with the settings $s = 100$, $p_m = 0.1$, $t_{max} = 10000$. The column σ represents the average percentage deviation $\sum_{k=1}^{10} \frac{z_k - z^*}{10z^*} \cdot 100\%$, where z_k is the solution found in k-th trial and z^* is the optimal or the best-known value. Let f_{bst} be the frequency of obtaining the best solution found by a GA in a series of 10 trials. The average time T_{sol} spent until the final solution occurred, the average time of obtaining the best solution found in 10 trials T_{bst}, and the average execution time T_{ex} are given in 100MHz Pentium seconds. The results obtained by other authors are converted by estimating the performance of the computers using Dongarra report [7]. In Table 2 the mentioned above values are averaged over all instances of problem set for the sets *4-6* and *A-D*.

The algorithm CFT was run once for each problem [4], thus f_{bst} and σ are not available. The optimal or the best-known solutions to all problems in *4-6* and *A-H* are found by it.

Name	CFT		BC				NBGA				
	T_{bst}	T_{ex}	f_{bst}	σ	T_{sol}	T_{ex}	f_{bst}	σ	T_{bst}	T_{sol}	T_{ex}
4	2.88	2212.4	0.88	0.07	44.95	315.06	0.86	0.09	35.3	33.5	369.2
5	1.42	2212.4	0.76	0.17	27.21	550.80	0.7	0.27	40.9	37.7	318.0
6	4.13	2212.4	0.94	0.07	25.23	319.10	0.86	0.24	40.05	41.0	486.0
A	47.15	2212.4	0.86	0.20	65.98	787.60	0.44	0.35	82.0	71.8	1693.0
B	3.34	2212.4	1.00	0.00	68.63	938.30	1.00	0.00	20.8	20.8	343.9
C	29.23	2212.4	0.68	0.41	87.93	1141.10	0.74	0.26	53.5	52.4	721.3
D	7.64	2212.4	0.96	0.06	101.70	1372.63	0.94	0.08	26.62	23.33	582.7

Table 2.

The optimal solutions to all problems in *4-6* and *A-D* except 5.3 are obtained by BC at least once in 10 trials. The current best-known solutions for G2, H1, H2 are better then those found by BC in [3].

Table 3 gives more detailed information on experiments with the sets *E-H*. Here BK is the best-known solution value. The lower bound found by solving SCP linear relaxation is denoted by LP.

Name	BK	LP	CFT			BC					NBGA					
			Best	T_{bst}	T_{ex}	Best	f_{bst}	σ	T_{sol}	T_{ex}	Best	f_{bst}	σ	T_{bst}	T_{sol}	T_{ex}
E1	29	21.3	**29**	11.5	2212	**29**	1.0	0.0	16.9	3014.1	**29**	1.0	0.0	1.0	1.0	562.6
E2	30	22.4	**30**	180.5	2212	**30**	0.4	2.0	266.9	3101.5	**30**	1.0	0.0	94.8	94.8	1768.4
E3	27	20.5	**27**	41.7	2212	**27**	0.3	2.6	85.1	3110.4	**27**	1.0	0.0	23.1	23.1	1345.3
E4	28	21.4	**28**	11.6	2212	**28**	1.0	0.0	238.5	3093.5	**28**	1.0	0.0	11.0	11.0	1234.4
E5	28	21.3	**28**	16.2	2212	**28**	1.0	0.0	15.5	3122.4	**28**	1.0	0.0	2.1	2.1	1067.2
F1	14	8.8	**14**	14.7	2212	**14**	1.0	0.0	33.8	5026.0	**14**	1.0	0.0	7.9	7.9	1698.6
F2	15	10.0	**15**	13.8	2212	**15**	1.0	0.0	34.5	5535.5	**15**	1.0	0.0	1.3	1.3	1548.5
F3	14	9.5	**14**	110.0	2212	**14**	1.0	0.0	117.9	5318.1	**14**	1.0	0.0	55.4	55.4	1765.1
F4	14	8.5	**14**	13.7	2212	**14**	1.0	0.0	92.6	5710.3	**14**	1.0	0.0	20.4	20.4	5686.0
F5	13	7.8	**13**	89.0	2212	**13**	0.3	5.4	67.1	5760.0	**13**	0.3	5.4	497.4	151.2	8290.3
G1	176	159.9	**176**	65.0	2212	**176**	0.2	1.0	451.3	3222.0	**176**	0.7	0.3	115.0	96.0	795.0
G2	154	142.1	**154**	346.6	2212	155	0.5	1.5	159.3	3628.0	**154**	0.5	0.65	318.3	226.6	836.6
G3	166	148.3	**166**	432.7	2212	**166**	0.1	1.1	312.1	3151.5	**166**	0.1	0.8	627.6	319.1	972.5
G4	168	148.9	**168**	167.5	2212	**168**	0.2	1.4	665.4	3503.9	**168**	0.4	0.7	160.0	172.5	884.8
G5	168	148.2	**168**	105.0	2212	**168**	0.2	0.8	242.6	3696.4	**168**	0.7	0.2	161.2	170.4	952.2
H1	63	48.1	**63**	642.1	2212	64	1.0	1.6	743.0	5383.4	64	1.0	1.6	90.5	90.5	1573.7
H2	63	48.7	**63**	392.5	2212	64	1.0	1.6	234.3	5750.0	64	1.0	1.6	34.7	34.7	9649.5
H3	59	45.2	**59**	690.4	2212	**59**	0.9	0.2	796.6	5703.6	**59**	1.0	0.0	493.2	493.2	5619.8
H4	58	44.0	**58**	105.1	2212	**58**	0.4	1.6	962.9	5534.9	**58**	1.0	0.0	218.2	218.2	3458.3
H5	55	42.4	**55**	68.8	2212	**55**	0.9	0.2	198.6	5610.1	**55**	1.0	0.0	25.2	25.2	1049.4

Table 3.

We have also tested the NBGA with uniform crossover (where each gene of the offspring is copied from one of the parents with equal probabilities) and with more simple local improvement schemes, but these versions turned out to be less efficient.

The NBGA was also tested on a set of combinatorial problems *CLR* derived from a question of P.Erdös: *"what is $m_d(r)$, the minimal number of r-tuples that can be chosen so that every coloring of d elements with 2 colors contains at least one monochromatic r-tuple?"* (here $d \geq 2r - 1$)[8]. This problem can be stated as a unicost SCP [11], where M is the set of colorings of d elements (due to symmetry only a half of colorings needs to be considered), N is the set of all r-tuples. Assume that an r-tuple covers the colorings for which it is monochromatic. We consider the case $r = 4$. Table 4 shows the dimensions of SCP instances, the best-known upper bounds on $m_d(4)$ from literature and the values of the solutions found by NBGA. The columns f_{bst} and T_{sol} have the same meaning as in Table 3. Here we again use $s = 100$, $t_{max} = 10000$. The values of p_m are indicated in the column p_m. For the cases $d = 12$ and 13 the best solutions known in literature are improved by NBGA, and for the cases $d = 9, 10, 11$ it found the solutions of previously known values (see e.g.[11]). In fact, simple solutions of 23 tuples could be constructed for all $d' > 11$ without a GA. To do this, one should take the existing solution for $d = 11$ and use the same set of tuples for all $d' > d$ (similarly it can be proved in general that $m_d(r)$ is a monotone non-increasing function in d).

The Table 4 also contains the results of NBGA computations on a series of problems *Stein* that arise from Steiner triple systems. These instances were proposed in [10] as the examples of hard problems that can be used for evaluation of different algorithms.

A Steiner triple system consists of $m = \frac{n(n-1)}{6}$ subsets of cardinality 3 (called *triples*) of n-element ground set, where $n \equiv 1, 3 (mod\ 6)$, and every pair of elements of the ground set is contained in

$Problem$	$Rows(m)$	$Columns(n)$	$Best-known$	$NBGA$	f_{bst}	T_{sol}	p_m
CLR.9	255	126	26	26	1.0	15.32	0.05
CLR.10	511	210	25	25	1.0	27.0	0.05
CLR.11	1023	330	23	23	1.0	274.6	0.05
CLR.12	2047	495	26	**23**	0.2	979.7	0.02
CLR.13	4095	715	26	**23**	0.3	4615.4	0.01
Stein.27	117	27	18*	18	1.0	1.2	0.1
Stein.45	330	45	30*	30	1.0	12.6	0.1
Stein.81	1080	81	61*	61	1.0	16.6	0.02
Stein.135	3015	135	104	104	0.2	212.4	0.01
Stein.243	9801	243	202	**198**	0.6	536.1	0.005

Table 4.

precisely one of the triples. The problem consists in finding the minimum cardinality subset of the ground set that covers all triples of the system (it is supposed that an element of ground set covers all triples that contain it). During the experiments with these probems the NBGA found the optimal solutions in all cases where the optimum is known (these solutions are marked by '*' in the column $Best-known$). The solution found for the instance Stein.243 improves the best result known to us from the literature [12].

According to the computation experiments, on the set E the NBGA with the proposed settings dominates both other algorithms. The sets F and H include some problems on which NBGA is inferior to CFT and some problems where it is more efficient. The results for both GAs on the sets *4-6* and *A-D* are analogous but the solution times are longer, than for CFT. In terms of the best found solutions the performance of NBGA is between those of BC and CFT.

Further research may address the development of exact hybrid algorithms using e.g. an approach similar to [9]. The possibility of parallel implementation of NBGA might be considered too.

The research was supported in part by RFBR Grant 97-01-00771, and INTAS Grant 96-0820.

Corresponding Author: Anton Eremeev, Omsk Branch of Sobolev Institute of Matematics,
st. Pevtsov 13, 644099, Omsk, Russia.
E-mail: eremeev@iitam.omsk.net.ru

References

[1] E.Balash and A.Ho. Set Covering Algorithms Using Cutting Planes, Heuristics and Subgradient Optimization: A Computational Study. *Math. Programming*, 12:37-60,1980.

[2] J.E.Beasley. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41(11):1069-1072, 1990.

[3] J.E.Beasley, P.C.Chu. A Genetic Algorithm for the Set Covering Problem. *European Journal of Operation Research*, 94(2):394-404, 1996.

[4] A. Caprara, M. Fischetti, P. Toth. Algorithms for the Set Covering Problem. Technical Report No. OR-98-3. *DEIS-Operations Research Group*, 1998.

[5] V.Chvátal. A Greedy Heuristic for the Set Covering Problem. *Mathematics of Operations Research*, 4(3):233-235, 1979.

[6] L. Davis. Handbook of Genetic Algorithms, *Van Nostrand Reinhold, New York,* 1991.

[7] J.J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Technical Report No. CS-89-85. *Computer Science Department, University of Tennessee,* 1998.

[8] P.Erdös. On a Combinatorial Problem, *Nordisk Mat. Tidskrift*, 11:5-10, 1963.

[9] A.V.Eremeev A Genetic Algorithm with a Non-Binary Representation for the Set Covering Problem. In P. Kall, H.-J. Lüthi, eds. *Proceedings of Operations Research (OR'98).* Springer Verlag, 175-181, 1999.

[10] A.V.Eremeev, A.A.Kolokolov. On Some Genetic and L-class Enumeration Algorithms in Integer Programming. In *Proceedings of the First International Conference on Evolutionary Computation and Its Applications,* Moscow, 297-303, 1996.

[11] D.R.Fulkerson, G.L.Nemhauser, L.E.Trotter. Two Computationally Difficult Set Covering Problems that Arise in Computing the 1-Width of Incidence Matrices of Steiner Triple Systems *Math. Programming Study.* 2: 72-81, 1974.

[12] T.Grossman, A.Wool, Computational Experience with Approximation Algorithms for the Set Covering Problem, *Eur. J. Operational Research* 101(1): 81-92, 1997.

[13] C.Mannino, A.Sassano, Solving Hard Set Covering Problems *Oper. Res. Letters.* 18: 1-5, 1995.

[14] L.A.Zaozerskaya. On L-Class Enumeration Algorithm for Set Covering Problem. (in Russian) In *Proceedings of 11-th Baikal International School-Seminar "Optimization methods and their applications"*, 139-142, 1998.