Revista Varia Scientia
v.10, n.17, p. 147-162

Ano de Impressão 2011

Ademir Aparecido Constantino[1]
Andréia Alves dos Santos[2]
Sílvio Alexandre de Araujo[3]

NOVO ALGORITIMO GENÉTICO PARA PROBLEMAS DE COBERTURA DE CONJUNTOS

RESUMO: O Problema de Cobertura de Conjuntos (PCC) é bastante importante em Pesquisa Operacional, pois, pode ser encontrado como parte de vários problemas reais. Neste trabalho é reportado o uso de um algoritmo genético para resolver o PCC. O Algoritmo inicia com uma população gerada por uma heurística gulosa randômica. Um novo operador de cruzamento e um novo operador adaptativo de mutação foram incorporados para intensificar a busca. Nosso algoritmo foi testado para uma classe de exemplos "non-unicost", obtido da "OR-Library", sem aplicar técnicas de redução. O Algoritmo encontrou boas soluções em termos de qualidade e tempo computacional. Os resultados revelam que o algoritmo proposto é capaz de encontrar soluções de boa qualidade de maneira mais rápida do que algumas abordagens recentemente publicadas na literatura usando as instâncias da OR-Library.

PALAVRAS CHAVES: Cobertura de conjunto, algoritmos genéticos.

A NEW GENETIC ALGORITHM FOR THE SET COVERING PROBLEM

ABSTRACT: The Set Covering Problem (SCP) plays an important role in Operational Research since it can be found as part of several real-world problems. In this work we report the use of a genetic algorithm to solve SCP. The algorithm starts with a population chosen by a randomized greedy algorithm. A new crossover operator and a new adaptive mutation operator were incorporated into the algorithm to intensify the search. Our algorithm was tested for a class of non-unicost SCP obtained from OR-Library without applying reduction techniques. The algorithms found good solutions in terms of quality and computational time. The results reveal that the proposed algorithm is able to find a high quality solution and is faster than recently

[1] Professor titular, Departamento de Informática, Universidade Estadual de Maringá, Maringá/PR.
[2] Bacharel em Informática, Departamento de Informática, Universidade Estadual de Maringá, Maringá/PR.
[3] Professor adjunto, Departamento de Ciências da Computação e Estatística, Universidade Estadual Paulista.

published approaches algorithm is able to find a high quality solution and is faster than recently published approaches using the OR-Library.

KEYWORDS: set covering problem, genetic algorithms.

INTRODUCTION

The Set Covering Problem (SPC) is a classical optimization problem that has applications in crew scheduling, assembly line balancing, facilities locations, information retrieval (Al-Sultan *et al*. 1996). It can be defined as:

- Let $I = \{1,2, ..., m\}$ and $J = \{1, 2, ..., n\}$ sets of indexes.
- Let $\Gamma = \left\{P_j \subseteq I, j \in J\right\}$, such as, $I = \bigcup_{j \in J} P_j$.
- Let c_j the cost associated to the set P_j.
- We can define a *cover* as a subset $J^* \subseteq J$ such as $I = \bigcup_{j \in J^*} P_j$ with cost $C^* = \sum_{j \in J^*} c_j$.

The SCP consists of finding a cover with minimum cost. This problem can be represented as a mathematical programming model:

Minimize

$$\sum_{j=1}^{n} c_j x_j$$

subject to: $\sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = 1, 2, ..., m$

$c_j \in \{0,1\}; x_j \in \{0,1\} \quad j = 1, 2, ..., n$

Let A a $m \acute{} n$ matrix of zeros and ones, where a_{ij} Î?A and a_{ij}=1 if and only if IÎP_j and a_{ij} = 0 otherwise. If the cost c_j is always equal to 1 so the SCP is named unicost SCP otherwise it is a non-unicost SCP. A variable x_j is a decision variable of the problem and shows if the column j was selected (x_j = 1) or not (x_j = 0). Note that I and J represent the matrix A sets of rows and columns, respectively. Moreover, P_j represents the column j of the matrix A. More details about this problem we suggest to read GHVÀTAL (1979) and GAREY AND JOHNSON (1979)

It is well-known that the decision problem of the SCP is NP-complete (Garey and Johnson, 1979). Several applications of the SCP

can be found in the literature, such as: urban transit crew scheduling problem (Desrochers and Soumis, 1989), location of emergency service facilities (Toregas *et al*, 1971), assembly line balancing problem (Salveson, 1955) and test set compaction (Flores *et. al*., 1999).

In these real cases it is very easy to find problems with hundreds of rows and thousands of columns. Thus, there are lots of researches using heuristic algorithms to solve the SCP. One of the first papers was Chvàtal (1979) that implemented a greedy algorithm. Balas and Ho (1980) also developed a greedy algorithm based on five new greedy functions. Vasko and Wilson (1984) explored the Chvàtal (1979) and Balas and Ho (1980) heuristics, with new greedy functions and a procedure to remove redundant columns of a feasible solution. Feo and Resende (1989) proposed a non-deterministic variation of the Chvàtal (1979) heuristic where a local search was developed. Beasley (1990a) and Haddadi (1997) developed algorithms based on Lagrangean heuristics. Jacobs and Brusco (1993) implemented a simulated annealing algorithm. Beasley and Chu (1996), Al-Sultan *et al.* (1996) and Aickelin (2002) developed procedures based on Genetic Algorithms. Recently, Lan *et al.* (2007) developed a meta-heuristic based on randomized priority search and Yagiura *et al.* (2006) developed a 3-flip neighborhood local search. A relatively recent review on algorithms for SCP can be found in Caprara *et al.* (2000) where several exact algorithms were compared showing that the solver Cplex is the best exact algorithm. Ren *et al*. (2008) proposed a ACO-based algorithm, but they only compare with previous ACO-based algorithms and results quality are not better when we comparing with other approaches.

In the present paper, a procedure based on Genetic Algorithm was implemented. The algorithm reached good results comparing run time of the best-known results from the literature Yagiura *et al.* (2006) using the OR-Library benchmark. The procedure and the computational results are described in the following sections.

The remainder of the paper is structured as follows. The algorithm description is given in the Material and Methods section, including the codification and the genetic operators. Experiments are presented and discussed in the Results section while we conclude the paper in the Conclusions section.

MATERIAL AND METHOD

In this section we present the computational encoding and the Genetic Algorithm parameters. For a complete overview of Genetic

Algorithm see Beasley *et al.* (1993a), Beasley *et al.* (1993b), Goldberg (1989) and Reeves (1993).

FITNESS FUNCTION

The fitness function of an individual is defined as the sum of the cost of columns that belong to this individual. We denote by:

$C^* = \sum_{j \in J^*} c_j$ where c_j is the cost of the column *j, j ÎJ.*

REPRESENTATION

A well-known representation of a chromosome to the SCP is a binary array with the same size of the number of columns of the matrix *A* (Al-Sultan *et al.* 1996) where the position *j* has value 1 if the column *j* is in the solution, and 0 otherwise.

However, many SCP have a low density (sparse) matrix in the literature and in practical applications. Therefore, in this paper we implemented a compact and more efficient representation through a vector $M=[m_j]$ where $m_j=P_j$, *jJ*;. We use a linked list structure to implement m_j *as* an example in Figure 1 where we show the representation of a coverage matrix *A*.

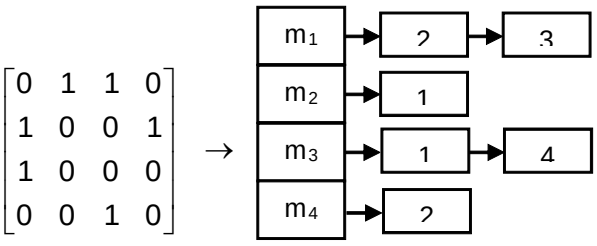

Figure 1 A matrix represented by a linked list structure.

In addition to the column list, we keep another auxiliary structure in order to facilitate the manipulation on the algorithm. This auxiliary structure keeps, for each row, the number of columns that covers this row and the indexes of these columns.

Following this idea, each individual (solution) is represented by a set of index (colunms) implemented as a linked list structure, instead a binary representation (Al-Sultan *et al.* 1996 and Beasley, 1996), using similar structure to represent m_j , but in this case, each node means a column used by the solution. This is a compact way to represent a

solution, once a solution uses a small number of columns when we compare to number of columns in the matrix A.

Initial Population

Considering the population as a set $Pop=\{S_l \subseteq J, l=1,2,...,NPop\}$, where $NPop=|Pop|$. We also implemented the population representation as a dynamic list with every solution (structures) in non-increasing order of the costs (fitness).

To generate a solution for the initial population a randomized constructive method based on a greedy heuristic was implemented, where at each step, one row is covered by one column chosen by a greedy algorithm. Therefore, only feasible solutions are constructed. The algorithm to generate an individual of the initial population is described in the following figure:

α_i is a set of columns that cover the row i, $i \in I$; β_j is a set of rows covered by the column j, $j \in J$; c_j is the cost of the column j;
1. Initialize $S := \varnothing$; $U := I$; $w_i := 0, \forall i \in I$;
2. While ($U \neq \varnothing$) do
   (a) Select randomly a row i, $i \in U$;
   (b) Select randomly a column $j \in \alpha_i$ that minimize $c_j / |U \cap \beta_j|$;
   (c) $S := S \cup \{j\}$;
   (d) $w_i := w_i + 1, \forall i \in \beta_j$;
   (e) $U := U - \beta_j$;
Return S.

Figure 2 Individual generation

Once a solution is constructed, an algorithm is applied to eliminate redundant columns, i.e. columns that can be removed without feasibility loss.

1. $S' := S$;
2. While ($S' \neq \phi$) do
   (a) Select randomly a column $j \in T$;
   (c) $S' := S - \{j\}$;
   If ($w_i \geq 2, \forall i \in \beta_j$) then
      $S' := S - \{j\}$;
      $w_i := w_i - 1, \forall i \in \beta_j$
Return S.

Figure 3 Redundancy elimination

At the end of the process we have an individual that is evaluated and included in the initial population. These steps are repeated until NPop individuals have been generated, where NPop is the size of the population.

SELECTION TECHNIQUES

The selection criterion is probabilistic and based on a proportional selection (ranking) where the classification considers the relative position (associated with the fitness function) of the individual in the population. Thus, the population is classified according to the fitness function and the better individuals will be at the beginning of the list. The following function is used:

$$p(S_l) = \frac{l}{\sum_{k=1}^{Npop} k} = \frac{2l}{Npop(Npop+1)},$$

Where S_l represents an individual belongs to the population; *Npop* represents the population size and *l* represents the individual's position on the population. This function does not need to be actualized for each iteration since the population size is fixed.

CROSSOVER OPERATOR

The crossover operator is similar to the Individual Generation. The steps are described following:

1. X:= Selection an individual;
2. Y := Selection an individual;
3. $Z := X \cup Y$;
4. Initialize $S := \phi$;
5. Initialize $U := I$;
6. Initialize $w_i := 0, \forall i \in I$;
7. While ($U \neq \phi$) do
   - (a) Select randomly a row ;
   - (b) Select randomly column $j \in Z$ that minimize $c_j / \left|U \cap \beta_j\right|$;
   - (c) $S := S \cup \{j\}$;
   - (d) $w_i := w_i + 1, \forall i \in \beta_j$;
   - (e) $U := U - \beta_j$;

Return S.

Figure 4 Crossover operator.

In steps 1 and 2 the fathers are selected to crossover according to the selection criterion. Following, the columns are united to be chosen in step 7 according to the heuristic procedure.

MUTATION

The mutation operator defines a number of rows to be uncovered and then removes a small set of columns from the solution according to the frequency of these columns in the population. After that, the uncovered lines are covered again with the *Individual Generation Procedure (Figure 2)*. Finally, the *Redundant Elimination Procedure* (*Figure 3*) is applied.

1. Initialize $\lambda, \lambda \in [0,1]$; / / percentage of rows to be uncovered
2. Do $k := 0$;
3. Do $U := \phi$;
/ / To remove a small set of columns
4. While $(k < \lambda * |I|)$ do
   (a) Select a line i, $i \in I$;
   (b) Select a column j, $j \in S$ (which cover line *i*) with probability proportional to the number of times *j* to appear on the population;
   (c) Do $S := S - \{j\}$;
   (d) Do $w_i := w_i + 1, \forall i \in \beta_j$;
   (e) Do $U := U \cup \{i\}, \forall i \in \beta_j$;
/ / Cover again the uncovered lines using a greedy procedure
5. While $(U \neq \phi)$ Do
   (a) Select randomly a row $i \in U$;
   (b) Select randomly a column $j \in J$ that minimize $c_j / |U \cap \beta_j|$;
   (c) Do $S := S \cup \{j\}$;
   (d) Do $w_i := w_i + 1, \forall i \in \beta_j$;
   (e) Do $U := U - \beta_j$;
6. *S := Reduntant Elimination* (*S*) (*Figure 3*);
7. Return S.

Figure 5 Mutation.

To apply the mutation the child must be worse than the best individual of the population and the probability of mutation must be positive. Generally, the mutation rate is constant. However, following the ideas from Beasley and Chu (1994) we implemented a variable mutation rate:

$$TxM(t) = \frac{TxMinM}{1 - e^{\left(-c_1(t) - c_{Npop}(t)\right)/c_1(t)}}$$

Where:

TxMinM = minimum mutation rate (parameter);

$c_1(t)$ = Cost of the worst individual at each iteration (or generation) t; $c_{Npop}(t)$ = Cost of the best individual at each iteration (or generation) t;

The mutation rate increases when the fitness of the worst individual of the population gets closer to the fitness of the best individual, which means an intensification of the search.

POPULATION REPLACEMENT

We use the steady-state replacement technique where only one individual is included in the population at each generation. To do so, some conditions must be satisfied:

(a) if a mutation was applied to a child and the child does not have new columns then this child replaces an individual that is above the average of the population otherwise it replaces an individual worse than the child.

(b) if the mutation was not applied and the child is better than the more similar father then this child replaces this father.

(c) if the child is equal to other on the population then it is disposed to avoid degeneration.

Stop Criterion

The stop criterion is applied after the convergence of the algorithm and consists of a parameter *NMax* that define the maximum number of iterations without an improvement of the fitness function.

THE ALGORITHM OVERVIEW

The main steps of the algorithm are described in the following figure:

t = iterations number; *S* = possible solution; ρ = random variable; *TxM* = mutation rate; *NMax* = maximum number of iteration without improvement of the solution.

1. Initial Population Generation and *t*:=0;

2. *S*:= Crossover (Figure 5);

3. Select randomly a ρ, $\rho \notin [0,1]$;

4. If (ρ < TxM(t)) then *S*:= Mutation (*S*) (Figure 6);

5. *R* receive the solution with the greatest cost from *Pop*;

6. If (Evaluate (*S*) < Evaluate (*R*)) then replace the individual *R* by *S*, keeping the population classification.

7. If The population has been modified then *t*:=0, else *t*:=*t*+1;

8. Repeat the steps 2 to 7 until $t > NMax$;

Figure 6 Algorithm overview.

RESULTS

Since our method is a Genetic Algorithm (NGA) like the Beasley and Chu (1996) (BeChu) method, which is a classic paper in this area, their method was also implemented and the same machine was used to compare the results. The machine is a Pentium III 900 MHz, 512 MB of RAM and both methods were implemented using Operational System Windows XP and the language Object Pascal. Moreover, the results were also compared with other recent methods from the literature developed by Lan *et al.* (2007) (Meta-RaPS) where the experiments were carried out on a Intel Pentium IV 1.7 GHz PC and Yagiura *et al.* (2006) (3-FNLS) where the algorithm was coded in C and run on a workstation Sun Ultra 2 Model 2300 (two Ultra SPARC II 200 MHz processors with 1 GB memory). To the best of our knowledge these references present the best results with OR-Library benchmenark.

The data base was the OR-Library that is an operational research library of data for computational tests (Beasley, 1990). This library has data tests for the non-unicost SCP. In this work, we consider only non-unicost problems. The library has 11 classes of problems: 4, 5, 6, A, B, C, D, E, F, G e H, with a total of 65 test files with different sizes. The classes 4 and 5 have ten problems and the others classes five problems each. Moreover, the optimal solution is unknown for some problems. The density of the problems (related to the density of the matrix) is variable in each class. For the test problem details see Beasley and Chu (1996).

For the methods BeChu, 3-FNLS and NGA each problem was solved ten times using different random seeds (there are 65 x 10=650 instances solved) and the computational results presented are the average of the ten instances. However, for the Meta-RaPS the results presented are not an average but the results of only one resolution to each problem (there are only 65 instances solved).

Table 1 shows the results of our new genetic algorithm (NGA) for the classes 4, 5, 6, A, B, C, D in which the optimal solutions are known. Observe that we compare the quality solution considering the three methods described at the beginning of this section (Bechu, 3FNLS and Meta-RaPS). Moreover, BKS means the Best Known Solution.

We observe that for ten instances of each problem our method got the optimal solution to every problem except A.1. Comparing the mean it is possible to see that, for this set of problems, the four method results were very close each other.

Table 1 Results to the classes 4, 5, 6, A, B, C and D

	BKS	BeCh	3-FNLS	Meta-RaPS	NGA
Instances	**Solution**	**Solution**	**Solution**	**Solution**	**Solution**
4.1	429	429.9	429	429	429
4.2	512	512	512	512	512
4.3	516	516	516	516	516
4.4	494	494	494	494	494
4.5	512	512	512	512	512
4.6	560	560.1	560	560	560
4.7	430	430	430	430	430
4.8	492	492	492	492	492
4.9	641	641	641	641	641
4.10	514	514	514	514	514
5.1	253	253	253	253	253
5.2	302	302.3	302	302	302
5.3	226	226	226	226	226
5.4	242	242.5	242	242	242
5.5	211	211	211	211	211
5.6	213	213	213	213	213
5.7	293	293.2	293	293	293
5.8	288	288.1	288	288	288
5.9	279	279	279	279	279
5.10	265	265	265	265	265
6.1	138	138	138	138	138
6.2	146	146	146	146	146
6.3	145	145	145	145	145
6.4	131	131	131	131	131
6.5	161	161	161	161	161
A.1	253	253.8	253	253	254
A.2	252	252.5	252	252	252
A.3	232	232.3	232	232	232.8
A.4	234	234	234	234	234
A.5	236	236	236	236	236.5
B.1	69	69	69	69	69
B.2	76	76	76	76	76
B.3	80	80	80	80	80
B.4	79	79	79	79	79
B.5	72	72	72	72	72
C.1	227	227	227	227	227
C.2	219	219	219	219	219.8
C.3	243	244.3	243	243	243
C.4	219	219.3	219	219	219
C.5	215	215	215	215	215
D.1	60	60	60	60	60
D.2	66	66	66	66	66
D.3	72	72	72	72	72
D.4	62	62	62	62	62
D.5	61	61	61	61	61
Mean	**253.7778**	**253.8956**	**253.7778**	**253.7778**	**253.8467**

In Table 2, we compare the computational time when the best solution is found considering the difference between the machines. It is possible to see that our method is much faster than BeCh method and is very competitive comparing to the recent methods 3-FNLS and Meta-RaPS.

Table 2 Mean Computational Time (in seconds) to the classes 4, 5, 6, A, B, C and D

	BeCh	**3-FNLS**	**Meta-RaPS**	**NGA**
Hardware	Pentium III 900 MHz, 512 MB of RAM	Sun Ultra 2 Model 2300 (two Ultra SPARC II 200 MHz processors with 1 GB RAM).	Pentium IV 1.7 GHz	Pentium III 900 MHz, 512 MB of RAM
Mean	18.31111	2.009111	3.017333	2.911111

In Table 3, we present the results to classes E, F, G and H, where the optimal solution is not known for most instances. The lower bounds (according to Yagiura *et al.* (2006)) are presented in column (LB).

We should observe that there are some small differences comparing the results of our implementation of BeChu´s algorithm and the original paper results. Basically, we have a small improvement in the solution quality to the problems E to H.

For these classes of problems, our method did not find (in ten instances) the best known solution to the problems G.2, G.3, H.1, H.2 which is similar to BeChu results. The 3-FNLS and the Meta-RaPS found the best known solution to all these problems. Considering the total mean, it is possible to observe that the results are still very close. However, our computational time (see Table 4) is better than BeChu and much better than Meta-RaPS even with a worse machine.

Table 3: Results to the classes E, F, G and H

	LB	BKS	BeCh	3-FNLS	Meta-RaPS	NGA
Instances	**solution**	**solution**	**solution**	**solution**	**solution**	**Solution**
E.1	29	29	29	29	29	29
E.2	28	30	30.9	30	30	30
E.3	27	27	27	27	27	27
E.4	28	28	28	28	28	28
E.5	28	28	28	28	28	28
F.1	14	14	14	14	14	14
F.2	15	15	15	15	15	15
F.3	14	14	14	14	14	14
F.4	14	14	14	14	14	14
F.5	13	13	13.1	13	13	13.6
G.1	165	176	177.2	176	176	176.3
G.2	147	154	156.2	154	154	155.6
G.3	153	166	168	166	166	168.2
G.4	154	168	170.8	168	168	169
G.5	153	168	168.1	168	168	168
H.1	52	63	64.1	63	63	64
H.2	52	63	64	63.3	63	64
H.3	48	59	59.6	59.9	59	60.8
H.4	47	58	58.5	58	58	58.2
H.5	46	55	55	55.4	55	55
Mean	**61.3**	**67.1**	**67.725**	**67.18**	**67.1**	**67.585**

Table 4: Mean Computational Time (in seconds) to the classes E, F, G and H

Hardware	BeCh	3-FNLS	Meta-RaPS	NGA
Mean	296.3	180*	351.461	23.25

* The computational time to 3-FNLS method, is not clear in the paper Yagiura *et al.* (2006). The authors wrote that the time limit were set to 180 seconds for instances E-H.

CONCLUSIONS

In this paper we proposed a straightforward new genetic algorithm to the Set Cover Problem and presented a comparative study of four different methods using non-unicost and and without reduction technique. The results shows that in terms of quality solution our method is better than the Genetic Algorithm developed by Beasley and Chu in 1996 but it is worse than two recent methods in the literature. However, in terms of computational time our method (which is very simple) is much better than the Beasley and Chu method and it seems

to be better (considering the difference in the machine) than the other recent methods.

For future work we intend to develop a local search to try to improve the quality solution without greatly increasing the computational time. Another suggestion for future work is to use the tournament method for the individual selection which allows the use of a data structure based on binary tree since it is not necessary to keep the classification of the solution. Finally, other sets of data used recently in the literature should be tested.

REFERENCES

AL-SULTAN, K.S., HUSSAIN, M.F., NIZAMI, J.S. A genetic algorithm for set covering problem. **Journal of the Operational Research Society**, 47, pp. 702-709, 1996.

BALAS, E., HO, A. Set covering algorithm using cutting planes, heuristics, end subgradient optimization: a computational study. **Mathematical Programming**, 12, pp. 37-60, 1980.

BEASLEY, J.E. A lagrangian heuristic for set-covering problems. **Naval Research Logistic**, 37, pp. 151-164, 1990a.

BEASLEY, J.E. OR-Library: distributing test problems by electronic mail. **Journal of the Operational Research Society**, 41, pp. 1990-1072, 1990b.

BEASLEY, D., D.R. BULL, MARTIN, R.R. An overview of genetic algorithms: Part I. Fundamentals, **University Computing**, 15, 58-69, 1993a.

BEASLEY, D., D.R. BULL, MARTIN, R.R. An overview of genetic algorithms: Part II. Research topics, ***University Computing,*** 15, 170-181, 1993B.

BEASLEY, J.E., CHU, P.C. A. A genetic algorithm for the set covering problem. **European Journal of Operational Research**, 94, pp. 392-404, 1996.

CAPRARA, A., TOTH, P., FISCHETTI, M. Algorithms for the Set Covering Problem. **Annals of Operational Research**, 98, 353-372, 2000.

DESROCHERS, M., SOUMIS, F. A column generation approach to the urban transit crew scheduling problem. **Transportation Science**, 23, pp. 1-13, 1989.

DONGARRA, J.J. Performance of various computers using standard linear equations software. **Computer Architecture News**, 20, pp. 22-

44, 1992.

FEO, T. A., RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. **Operations Research Letters**, 8, pp. 67-71, 1989.

FLORES, P, F., NETO, H.C., Marques-Silva, J.P. On Applying Set Covering Models to Test Set Compaction. **Great Lakes Symposium on VLSI**, pp. 8-11, 1999.

GAREY, M.R., JOHNSON, D.S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. Freeman, New York, 1979

GARFINKEL, R.S., NEMHAUSER, G.L. **Integer Programming**. New York. John Wiley, 1972.

GHVÀTAL, V. A greedy heuristic for the set covering problem. **Management Science**, 21, pp. 591-599, 1979.

GOLDBERG D. **Genetic Algorithms in Search, Optimization and Machine Learning**, Addison-Wesley, New York, 1989.

HADDADI, S. Simple lagrangian heuristic for the set covering problem. **European Journal of Operational Research**, 97, pp. 200-204, 1997.

JACOBS, L. W., BRUSCO, J.J. A simulated annealing-based heuristic for the set-covering problem. **Working paper**, Operations Management and Information System Department, Northern Illinois University, 1993.

LAN, G., DEPUY, G. W., Whitehouse, G. E. An Effective and simple heuristic for the seti covering problem, **European Journal of Operational Research**, v. 176, p. 1387-1403, 2007.

REEVES, C.R. **Modern Heuristic Techniques for Combinatorial Problems**, Blackwell Scientific, 1993.

REN, Z.; FENG, Z.; KE, L.; CHANG, H. A fast and efficient ant colony optimization approach for the set covering problem. In: Evolutionary Computation, 2008, Hong Kong. **Anais...** IEEE World Congress on Computational Intelligence, 2008, p. 1839-1844.

SALVESON, M. E. The assembly line balancing problem. **Journal of Industrial Engineering**, 6, pp. 18-25, 1955.

TOREGAS, C., SWAIN, R., REVELLE, C., BERGMAN, L., 1971. The location of emergency service facilities. **Operations Research**, 19, pp. 1363-1373.

VASKO, F.J., WILSON, G.R. An Efficient Heuristic for Large Set Covering Problems. **Naval Research Logistic Quarterly**, 31, pp. 163-171, 1984.

YAGIURA, M., KISHIDA, M., IBARAKI, T. A 3-flip neighborhood local search for the set covering problem. **European Journal of Operational Research**, v. 172, pp. 472-499, 2006.

VARIA SCIENTIA

Versão eletrônica disponível na internet:

www.unioeste.br/saber