Learner Support Services

The University of Bradford Institutional Repository

http://bradscholars.brad.ac.uk

Author(s): Dahal, K. P. and McDonald, J. R.

Title: Generator maintenance scheduling of electric power systems using genetic algorithms with integer representations.

Publication year: 1997.

Conference title: Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, Glasgow 2-4 Sept 1997 (GALESIA 97)

ISSN: 0-85296-693-8

Publisher: IEEE

Link to original published version:
http://ieeexplore.ieee.org/servlet/opac?punumber=5586

Citation: Dahal, K. P. and McDonald, J. R. (1997) Generator maintenance scheduling of electric power systems using genetic algorithms with integer representations. In: Second International Conference On Genetic Algorithms in Engineering Systems: Innovations and Applications, Glasgow 2-4 Sept 1997 (GALESIA 97). New York: IEEE. Conf. Publ. No. 446. pp.456-461.

GENERATOR MAINTENANCE SCHEDULING OF ELECTRIC POWER SYSTEMS USING GENETIC ALGORITHMS WITH INTEGER REPRESENTATION

K.P.Dahal, J.R.McDonald

Centre for Electrical Power Engineering, University of Strathclyde, Glasgow, UK

(presented at International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'97).

ABSTRACT

The effective maintenance scheduling of power system generators is very important to a power utility for the economical and reliable operation of a power system. Many mathematical methods have been implemented for generator maintenance scheduling (GMS). However, these methods have many limitations and require many approximations. Here a Genetic Algorithm is proposed for GMS problems in order to overcome some of the limitations of the conventional methods.

This paper formulates a general GMS problem using a reliability criterion as an integer programming problem, and demonstrates the use of GAs with three different problem encodings: binary, binary for integer and integer. The GA performances for each of these representations are analysed and compared for a test problem based on a practical power system scenario. The effects of different GA parameters are also studied. The results show that the integer GA is a very effective method for GMS problems.

1. INTRODUCTION

It is very important for the effective operation of a power system to determine when its generators should be taken off line for preventive maintenance. This is primarily because other planning activities are directly affected by such decisions. In modern power systems the demand for electricity has greatly increased with related expansions in system size, which has resulted in higher numbers of generators and lower reserve margins making the generating maintenance scheduling (GMS) problem more complicated. The goal of GMS is to allocate a proper maintenance timetable for generators while maintaining a high system reliability, reducing total production cost, extending generator life time etc., subject to some unit and system constraints. The maintenance schedule of each generating unit should be optimised in terms of the particular objective function under a series of constraints.

Several deterministic mathematical methods and heuristic techniques are reported in the literature for solving these problems [1,2,3,4]. General solution methods are based on integer programming, branch-and-bound technique, dynamic programming, etc. However, such approaches are severely limited by the 'curse of dimensionality' and are poor in handling the non-linear objective and constraint functions that characterise the GMS problem. The heuristic approach uses a trial-and-error method to evaluate the maintenance objective function in the time interval under examination. This requires significant operator input and in some situations it fails to produce even feasible solutions [3,4].

Genetic algorithms may be an effective alternative method for finding optimal or near optimal solutions of these complex problems. This paper describes the procedure for implementing GAs for solving the GMS problem. A steady state GA approach has been applied to a medium sized test GMS problem, which includes many features that characterise real systems. The GA is implemented for three types of problem encoding: binary, binary for integer and integer. The results have shown that the integer representation is very efficient in finding good solutions for GMS problems.

The following section describes the formulation of a general GMS problem using a reliability objective and general unit and system constraints. Section 3 details the implementation of GAs to the GMS problem with the three types of problem encoding. The performances of the GA with these different problem representations for the test problem are compared and discussed in Section 4. This section also presents the effects of GA parameters on the performance of the GA and the results obtained using an improved mutation operator. Our conclusions are presented in Section 5.

2. PROBLEM FORMULATION

There are generally two categories of objective functions in GMS, based on reliability and economic cost [4]. This paper uses the reliability criteria of levelling reserve generation throughout the period under examination. This can be realised by minimising the sum of squares of the reserve over the entire operational planning period. The problem has a series of unit and system constraints to be satisfied, which in general include the following:

- Maintenance window and sequence constraints - define the earliest and latest time, the duration and the restriction of maintenance for each unit.
- Crew and resource constraints - consider the manpower availability and the limits on the resources needed for maintenance activity at each time period.
- Load and reliability constraints - consider the demand and the risk level on the power system during the scheduling period .

Mathematically, the GMS problem can be formulated as an integer programming problem by using integer variables associated with answers to "When does

maintenance start?" or alternatively by using conventional binary variables associated with answers to "When does maintenance occur?" [1]. However the first formulation takes care of the constraints on the periods and duration of maintenance and hence the number of unknowns is reduced. The answer to the first question automatically provides the answer to the second.

Notation:

i	index of generating units
I	set of generating unit indices
N	total number of generating units
t	index of periods
T	set of indices of periods in planning horizon
e_i	earliest period for maintenance of unit i to begin
l_i	latest period for maintenance of unit i to end
d_i	duration of maintenance for unit i
P_{it}	generating capacity of unit i in period t
L_t	anticipated load demand for period t
M_{it}	manpower needed by unit i at period t
AM_t	available manpower at period t

Suppose $T_i \subset T$ is the set of periods when maintenance of unit i may start, $T_i=\{t \in T: e_i \le t \le l_i-d_i+1\}$ for each i. We define

$$X_{it}=\begin{cases}1 & \text{if unit i starts maintenance in period t,}\\ 0 & \text{otherwise,}\end{cases} \quad i \in I, t \in T_i$$

to be the maintenance start indicator for unit i in period t. Let S_{it} be the set of start time periods k such that if the maintenance of unit i starts at period k that unit will be in maintenance at period t, $S_{it}=\{k \in T_i: t-d_i+1 \le k \le t\}$. Let I_t be the set of units which are allowed to be in maintenance in period t, $I_t=\{i: t \in T_i\}$. Then the problem can be expressed mathematically as below.

The objective is to minimise the sum of squares of the reserve generation

$$\min_{X_{it}} \left\{ \sum_t \left(\sum_i P_{it} - \sum_{i \in I_t} \sum_{k \in S_{it}} X_{ik} P_{ik} - L_t \right)^2 \right\}, \qquad (1)$$

subject to the maintenance window constraint

$$\sum_{t \in T_i} X_{it} = 1 \qquad \forall i, \qquad (2)$$

the crew constraint

$$\sum_{i \in I_t} \sum_{k \in S_{it}} X_{ik} M_{ik} \le AM_t \qquad \forall t, \qquad (3)$$

the load constraint

$$\sum_i P_{it} - \sum_{i \in I_t} \sum_{k \in S_{it}} X_{ik} P_{ik} \ge L_t \qquad \forall t. \qquad (4)$$

In general a GMS problem may include alternative or additional constraints.

3. GA IMPLEMENTATION

Different types of approaches can be taken in the basic design of a GA. This paper applies a 'steady state' GA to GMS problems. With this approach new offspring are introduced immediately into the population on an individual basis, abandoning the standard generational structure. In each iteration step two individuals are selected from the population pool according to some selection procedure. A new offspring is created in the population pool replacing a less fit individual. Hence, the parents and offspring can co-exist in the same population pool for the next iteration step.

A GA software package called GENITOR [5] which uses this steady state structure has been modified for use on GMS problems. The GENITOR algorithm explicitly uses the ranking selection method. Parents are selected according to their ranked fitness score.

3.1. Problem encoding

The encoding of the problem using an appropriate representation is a crucial aspect of the implementation of a GA for solving an optimisation problem. Different types of candidate solutions may be used to encode the set of parameters for the evaluation function. GMS problems can be solved using three types of representations:

- binary representation,
- binary for integer representation,
- integer representation.

In the binary representation the GMS problem (1) - (4) is encoded by using an one-dimensional binary array as follows.

$$[X_{1,e1}, X_{1,(e1+1)}, \dots, X_{1,(l1-d1+1)}, X_{2,e2}, X_{2,(e2+1)}, \dots \\ \dots, X_{2,(l2-d2+1)}, \dots, X_{N,eN}, X_{N,(eN+1)}, \dots X_{N,(lN-dN+1)}]$$

This binary string (chromosome) consists of sub-strings which each contain the variables over the whole scheduling period for a particular unit. The size of the GA search space for this type of representation is

$$2^{\sum_{i=1}^{N}(l_i-d_i-e_i+2)}.$$

For each unit i=1,2,...,N, the maintenance window constraint (2) forces exactly one variable in $\{X_{it}: t \in T_i\}$ to be one and the rest to be zero. The solution of this problem thus amounts to finding the correct choice of positive variable from each variable set $\{X_{it}: t \in T_i\}$, for i=1,2,...,N [6]. The index t of this positive variable indicates the period when maintenance for unit i starts. In order to reduce the number of variables the indices of the positive variables from $\{X_{it}: t \in T_i\}$, for i=1,2,...,N, can be taken as new variables. The advantage of this approach is the possibility of using an integer encoding for these new variables in a genetic structure consisting of a string of integers, each one of which represents the maintenance start period of a unit. For this representation the string length is equal to the number of units (N) and the string is

$$t_1, t_2, ..., t_i, ..., t_N,$$

where t_i is an integer, $e_i \le t_i \le l_i - d_i + 1$, for each i=1,2,...,N, which indicates the maintenance start period for unit i. This type of representation automatically considers the maintenance window constraint (2) and greatly reduces the size of the GA search space to

$$\prod_{i=1}^{N} (l_i - d_i - e_i + 2).$$

The integer formulation of the problem can also be encoded by using binary (or Gray) code to represent the integer variables in the GA structures [7]. For example, with t_i defined as above, suppose the number of possible values of t_i is $l_i - d_i - e_i + 2 = 32$, then a 5 bit binary pattern may be used to represent the possible variable values. We call this representation 'binary for integer'. In this case if the number of variable values is not a power of 2, some of the binary values will be redundant. To overcome this problem, some integer values are represented by two or more bit patterns. The string length in this situation is $b_1 + b_2 + ... + b_N$ and the GA search space is

$$2^{\sum_{i=1}^{N} b_i},$$

where b_i is the number of the binary bits used to represent the integer variable values for unit i and equals the least positive integer greater than or equal to $\log_2(l_i - d_i - e_i + 2)$. This redundancy increases the size of the search space since

$$2^{\sum_{i=1}^{N} b_i} \ge 2^{\sum_{i=1}^{N} \log_2 (l_i - d_i - e_i + 2)} = \prod_{i=1}^{N} (l_i - d_i - e_i + 2).$$

A test GMS problem has been encoded using each of the three representations described above and the performance of the GA investigated.

3.2. Evaluation function

Along with the coding, the procedure for evaluation of new structures is another important aspect of GAs. To take care of the various constraints imposed on GMS problems, we have taken a penalty function approach [6]. The penalty value for each constraint violation is proportional to the amount by which the constraint is violated. The evaluation function is the sum of penalty values for each constraint violation and the objective function itself with some weighting coefficients, hence

$$\text{evaluation} = \sum_{c} \omega_C V_c + \omega_O F, \qquad (5)$$

where ω_C and ω_O are the weighting coefficients,

V_c is the amount of the violation of constraint c,

F is the objective value.

The weighting coefficients are chosen in such a way that the violation of harder constraints gives a greater penalty value than for the soft constraints. In general the penalty value for the constraint violations dominates over the objective function. Feasible solutions with low evaluation measures have high fitness values while unfeasible solutions with high evaluation measures take low fitness values.

In the test problem described below the crew constraint was assigned a low penalty coefficient. This is because a solution with a high reliability but requiring more manpower may well be accepted for a power utility as the unavailable manpower may be hired.

4. GA PERFORMANCE ANALYSIS

A number of small problems have been tested with the proposed GAs with different objectives and constraints. GAs with both generational and steady state approaches yield the optimum solution for small problems when appropriate GA parameters are chosen. Here we present the results of applying a steady state GA to a test problem comprising 21 units over a planning period of 52 weeks, which was loosely derived from the example presented in [2] with some simplifications and additional constraints. The problem is described below.

TABLE 1 -Data for the test system.

Unit	Capacity (MW)	Allowed period	Outage (weeks)	Manpower required for each week
1	555	1-26	7	10+10+5+5+5+5+3
2	555	27-52	5	10+10+10+5+5
3	180	1-26	2	15+15
4	180	1-26	1	20
5	640	27-52	5	10+10+10+10+10
6	640	1-26	3	15+15+15
7	640	1-26	3	15+15+15
8	555	27-52	6	10+10+10+5+5+5
9	276	1-26	10	3+2+2+2+2+2+ 2+2+2+3
10	140	1-26	4	10+10+5+5
11	90	1-26	1	20
12	76	27-52	3	10+15+15
13	76	1-26	2	15+15
14	94	1-26	4	10+10+10+10
15	39	1-26	2	15+15
16	188	1-26	2	15+15
17	58	27-52	1	20
18	48	27-52	2	15+15
19	137	27-52	1	15
20	469	27-52	4	10+10+10+10
21	52	1-26	3	10+10+10

Schedule the maintenance outages of generators to minimise the sum of squares of reserves and satisfy the following constraints:

- Maintenance window: Each unit must be maintained exactly once and the maintenance for each unit must occupy the required time duration without interruption.
- Load constraint: The system's peak load is 4739 MW.

- Crew constraint: There are only 20 people available for the maintenance work each week.

The data for the test problem is given in Table 1. Due to its complexity the optimum solution for this problem is unknown.

A number of GA runs have been done using the three representations (binary, binary for integer and integer), taking different values of the GA parameters and employing different GA operators. A brief analysis of the results is presented below.

4.1. Crossover operator

The crossover operator used here is a simple two-point crossover. This operator first chooses two points at which to break each of the two selected parent strings, and then combines fragments from each string to build offspring which contain information from each of the parent strings. The crossover is applied in each iteration when the exchanged information is unique to each parent. For the 'binary' and 'binary for integer' representations, the crossover points may be within a gene (a sub-string of a genetic structure which represents one particular unit). Hence the crossover operator may split genes and introduce changes within them. Theoretically, the splitting of genes by the crossover operator seems undesirable. In the case of integer representation, this sub-string splitting does not occur and the individual variable values are preserved in crossover. In this case only the mutation operator is responsible for creating a new integer value for a gene.

4.2. Effect of mutation probability on GA performance with different problem encodings

In order to observe the effect of the mutation probability for each of the representations described above, GA runs were carried out with varying mutation probability (MP), while taking all other GA parameters as constant. The population size was taken to be 50. The selection bias, which parameterises the selection of parents for reproduction, was taken as 2.5. We discuss this further in Section 4.4. The total number of trials for each run was fixed at 30000.

Table 2 presents the test results for the GMS problem from a total of 75 runs of the genetic algorithm using different values of mutation probability for the three types of representation. Each case presents the minimum, average and maximum evaluation measures of the best solutions obtained for 5 GA runs, each using a different random seed.

TABLE 2 -Effect of mutation probability (MP).

Type	MP	0.001	0.005	0.01	0.05	0.1
Binary	Min	2474	1143	4398	5.8e6	8.7e6
	Avg	3937	2138	1.3e5	6.3e6	9.4e6
	Max	6647	2600	3.0e5	7.2e6	1.0e7
Binary for integer	Min	167	156	159	156	239
	Avg	211	175	185	174	264
	Max	256	201	196	200	287
Integer	Min	191	144	141	138	147
	Avg	200	176	160	144	155
	Max	227	194	198	157	170

The computational results in Table 2 show that the effect of the mutation probability depends on the particular representation. For the binary representation, the GA achieves a better solution in a smaller number of GA trials with a lower mutation probability, whereas the higher mutation probabilities are recommended for 'binary for integer' and 'integer' representations. The variation of the performance on the mutation probability is much more sensitive for the binary representation than for the other two representations.

As explained in Section 3, using a binary representation, a string corresponding to a maintenance window feasible solution has only one '1' for each unit over the entire scheduling period with the remaining bits being '0'. Therefore, a maintenance window feasible genetic structure contains many more '0' bits than '1' bits. For our test problem, only 21 out of 496 bits in the string are '1' and the rests are '0'. A high mutation probability increases the chance of changing these '0's into '1's, dragging the solution into the maintenance window unfeasible region. Hence the search space is very large and most of it represents the unfeasible solutions. Therefore, a high mutation probability has the potential to disrupt and degrade the search process using the binary representation of the GMS problem. With higher mutation probabilities the GA could not find a maintenance window feasible solution even in 30000 trials. However, with lower mutation probabilities the GA found maintenance feasible solutions but converged prematurely. With the lower mutation probability there is a high chance of being trapped in a local minima.

The 'binary for integer' and 'integer' encodings of the GMS problem result in every candidate solution being maintenance window feasible, which causes a great reduction in the search space. The GA search is thus limited within the maintenance feasible region. In this case a higher mutation probability increases the exploration for the global minima within this limited region reducing the chance of premature convergence. However, a very high mutation probability causes more randomness in the GA search reducing the exploitation of the solutions previously found.

One point to be noted for the 'binary for integer' representation is that the actual mutation probability for changing integer values is much greater than the prescribed mutation probability. As explained above, in the 'binary for integer' representation the variable states (integers) are denoted by a binary (or Gray) code with a number of binary bits in a string, for example 5 bit strings are used for each unit for our test problem. The mutation operator takes each bit and decides whether or not to change that bit with the given mutation probability. In particular, the given mutation probability is the probability of mutating each binary bit. However,

a change in at least one of these 5 bits by the mutation operator results in a change in the corresponding integer value. The actual mutation probability m_a of changing the integer value for a given binary mutation probability m can be calculated as $m_a=1-(1-m)^{b_i}$, where b_i is the number of bits used to represent the integer variable for unit i. For example, if the given mutation rate m=0.05 and a 5 bit representation is used, the actual mutation probability is $m_a=0.23$. In order to have the actual mutation probability $m_a=0.05$, the mutation probability m should be taken as about 0.01. However, the distribution of the new integer values following mutation is not uniform in this case.

In the integer encoding of a GMS problem, each gene is an integer, which is the number of the time period in which maintenance work begins for a unit. The mutation operator takes each integer and with the given mutation probability changes the value within the allowed integer interval. The distribution of the new integer value within the interval is approximately uniform during mutation.

4.3. Comparison of performance for different representations

Table 3 presents a performance comparison for each of the three representations with the mutation probabilities chosen to give the best performance from Table 2.

TABLE 3 -Comparison of GA performance for different problem encodings.

Type	Binary	Binary for Integer	Integer
MP	0.005	0.05	0.05
Evaluation value of best solution	1142.88	156.41	137.91
Computational time	45.01s	21.07s	16.81 s
Size of GA search space	$2^{496}= 2.05 \times 10^{149}$	$2^{105}= 4.06 \times 10^{31}$	6.23×10^{28}

The GA with the integer representation found a solution with evaluation measure 137.91 for the test GMS problem, which is better than the solution found by the GA with the binary for integer representation (156.41), and significantly better than that of the GA with binary representation (1142.88). The binary GA did not find feasible solutions for load and crew constraints up to 30000 trials. It requires a large number of GA trials to obtain feasible solutions.

The computational times for the three GAs for one run with 30000 trials on a DEC Ultrix 5000/260 workstation are shown in Table 3. The time taken by the GA with the integer representation is shorter than that for the other two representations. The sizes of the search space for the three GAs are also shown in Table 3.

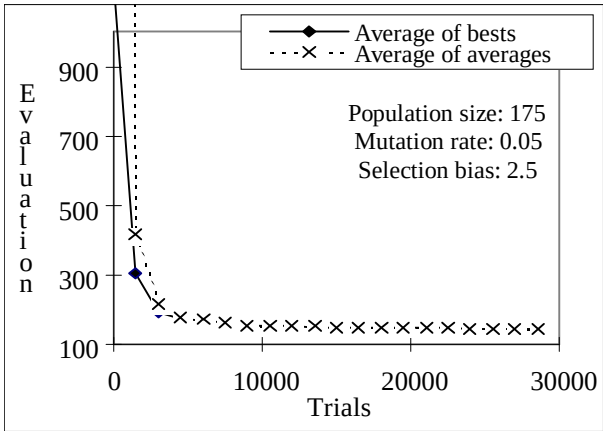

Figure 1: Average performance of GA with integer representation for 5 GA runs.

The average performance over 5 runs of the integer GA is depicted in Figure 1. Using the integer representation the best solution was found within 24000 trials and was not bettered before the GA terminated after 30000 trials. It is apparent from the above that the integer representation is the best choice for GMS problems both in terms of speed and quality of the solution, further discussion is concentrated on this representation alone.

4.4. Effect of selection bias and population size

The selection bias (SB) value specifies the amount of preference to be given to the superior individuals in the population. If SB=2, for example, then the selection probability for the best individual is twice that of the mean individual. If SB>2, a number of the least fit solutions in the genetic pool are assigned zero probability of selection.

In general, if the selection bias is too high, then a superior solution strongly dominates the less fit solutions and this may lead the GA to converge prematurely to a local minimum. Low values of the selection bias cause less preference to be given to the good genetic structures previously found. Therefore, a trade-off needs to be applied in the choice of the bias value. This is demonstrated in our results. Table 4 presents results found using bias values 1.5, 2, 2.5 and 3, with MP=0.05 and population size=50. Taking SB=2.5 gives the best solution.

TABLE 4 -Effect of selection bias (SB).

SB	Best solution in 5 GA runs		
	Minimum	Average	Maximum
1.5	148	156	174
2.0	147	153	165
2.5	138	144	157
3.0	143	151	162

The population size specifies the number of individuals in the genetic pool. A number of GA runs were done using the integer representation for different population sizes between 10 and 500 with other GA parameters

fixed: SB=2.5 and MP=0.05. It was found that the lowest average evaluation measure of the best solutions was achieved with population size 175, though the performance of the GA did not vary greatly over the different cases.

4.5 Use of adaptive mutation operator

During a run of a GA the optimum value of the mutation probability may be varied. Table 5 presents the results from 5 runs of the GA using the integer representation with the adaptive mutation operator. This operator dynamically varies the mutation probability depending on the Hamming distance between parents selected for crossover. The actual mutation probability is always less than or equal to the prescribed value. The GA runs were carried out for the integer representation with three prescribed mutation values 0.005, 0.01 and 0.05. The results show little difference between the performance with the traditional mutation operator and adaptive mutation operator for this representation.

TABLE 5 -GA performance with the adaptive mutation operator.

Given MP	Min	Avg	Max
0.005	146	158	176
0.01	149	161	176
0.05	143	150	156

Tables 2 and 5 indicate that the higher mutation values give better performance as the GA maintains the genetic diversity necessary to sustain the search for the global optimum.

TABLE 6 -The best solution found.

i	1	2	3	4	5	6	7	8	9	10
t_i	6	27	24	26	48	13	2	33	16	18
11	12	13	14	15	16	17	18	19	20	21
1	39	9	5	11	16	42	31	47	43	21

The best solution found by the GA during the above tests, whose evaluation measure is 137.9, is set out in Table 6, where t_i represents the index of the of maintenance start period for unit i. This solution is feasible and better than a heuristic solution (evaluation measure 222) calculated by ranking the generator units in order of decreasing capacity to level the generation. It can be seen from the test results that the integer GA is very stable for a wide range of variations in the GA parameters.

5. CONCLUSIONS

The results presented above show that the GA is a robust and stable technique for the solution of GMS problems for real-sized systems. Good solutions of the problem can be found if an appropriate problem encoding, GA approach, evaluation function and GA parameters are selected for the problem. Although GAs are not guaranteed to find the global optimal solution, it is a significant achievement to obtain a good solution to a complex problem like GMS in a short time.

As the GMS problem variables are integer, representing them directly as integers in a genetic structure has many advantages. The most significant of these is the great reduction in the GA search space. Furthermore, this type of representation is obvious and easy for decoding and a meaningful crossover and mutation operator can be applied. The integer GA is very robust for GMS problems and can find good solutions with a wide range of variations of the GA parameters in a comparatively short time, using traditional operators.

REFERENCES

[1] J.F. Dopazo, H. M. Merrill, “Optimal generator maintenance scheduling using integer programming”, IEEE Trans. PAS-94(5):1537-1545, 1975.

[2] Z. Yamayee, S. Kathleen, "A computationally efficient optimal maintenance scheduling method", IEEE Trans. PAS-102(2):330-338, 1983.

[3] T.G. Gerard, S.D. Tharam, M. Karol, “An experimental method of determination of optimal maintenance schedules in power systems using branch-and-bound technique”, IEEE Trans. SMC-6(8):538-547, 1976.

[4] X. Wang, J.R. McDonald, "Modern Power System Planning", McGraw-Hill, London, 247-307,1994.

[5] Darrel L. Whitley, “GENITOR”, available at ftp site: ftp.cs.colostate.edu/pub/GENITOR.tar, Colorado State University, 1990.

[6] Atidel Ben Hadj-Alouane, James C. Bean, “A genetic algorithm for the multiple-choice integer program”, Technical Report 92-50, Department of Industrial and Operations Engineering, University of Michigan, 1992.

[7] D. Beasley, D.R. Bull, R.R. Martin, “An overview of genetic algorithms: part 2, research topics”, University Computing, 15:170-181, 1993.