AMO - Advanced Modeling and Optimization - Volume 8, Number 2, 2006

Elitist genetic algorithm approach for Assignment Problem

J. Majumdar[1*] and A.K. Bhunia[2**]

[1] Department of Mathematics, Durgapur Government College, Durgapur-713214, India.

[2] Department of Mathematics, The University of Burdwan, Burdwan-713104, India.

Abstract

The name 'Assignment Problem' (AP) originates from the classical problems where the objective is to find the optimum assignment of a number of tasks (jobs) to an equal number of facilities (or persons) at a minimum cost (or time) or maximum profit. In this paper, a new approach for solving an assignment problem (balanced) is proposed with the help of an elitist genetic algorithm (EGA) and also its computational behaviour is reported. The mathematical formulation of the problem indicates that this problem is a 0-1 programming problem. To solve this problem, an EGA with new types of initialization, crossover, mutation and existing rank-based selection has been developed. As special cases, different types of assignment problems like unbalanced assignment problem, maximization assignment problem, restricted assignment problem have been reported and illustrated with some numerical examples.

Keywords : Assignment Problem, Genetic Algorithm, Elitism, Heuristic.

1 Introduction

The assignment problem (AP) is one of the most-studied, well-known and important problem in mathematical programming in which our objective is to assign a number of jobs (tasks) to an equal number of facilities (persons) so as to minimize the total assignment cost. This problem can be formulated as a linear programming problem (L.P.P.) particularly 0-1 programming problem. This problem can be solved using different methods like enumeration method, Simplex Method, Branch and Bound Method and Hungarian Method. However, as an assignment problem is highly degenerate, it will be frustrating to attempt to solve it by Simplex Method or Branch-bound Method. In fact, a very convenient and efficient iterative procedure for solving an assignment problem is the Hungarian Method. But for large assignment problems, this method will be very much laborious and therefore unsuitable.

Genetic Algorithm (GA) is a class of computerized adaptive heuristic search and optimization algorithm based on natural genetics. It is an iterative optimization procedure and it maintains a population

** Corresponding author: e-mail: bhuniaak@rediffmail.com*

* e-mail: majumdarjayanta@rediffmail.com*

of probable solutions within a search space over many simulated generations. The population members are string entities of artificial chromosomes. In natural terminology, we say that each chromosome (or string) is composed of genes, the basic building blocks. In each iteration (generation), three basic genetic operations viz., selection, crossover and mutation are performed. The basic concepts of GA were primarily developed by [Holland, 1976]. Thereafter, a number of researchers have contributed to the development of this field. Detailed reviews on the development of the subject can be obtained in the books of [Golberg, 1989], [Michalawicz, 1999], [Deb, 1995], [Sakawa, 2002], [Gen and Cheng, 1997], [Devis, 1991] and others.

In the last decade of 20th century, several researchers developed different methodologies for solving generalised assignment problems. Among them, one may refer to the works of [Catrysse and Van Wassenhove, 1992], [Chu and Beasley, 1997], [Lorena and Narciso, 1996], [Ross and Soland, 1975], [Wilson, 1997]. Again as special cases of the generalised assignment problem, manpower scheduling, nurse-scheduling, employee-scheduling etc. problems have been solved by [Bradley and Martin, 1990], [Easton and Mansour, 1993], [Tanomaru, 1995], [Dowsland, 1998], [Dowsland and Thompson, 2000]. Recently, [Aickelin and Dowsland, 2004] solved a nurse-scheduling problem by genetic algorithm using an indirect coding based on permutations of the nurses, and a heuristic decoder that builds schedules from these permutations. Again, [Harper et.al., 2005] developed a project assignment problem with the help of genetic algorithm. Among the above-mentioned researchers, [Aickelin and Dowsland, 2004], [Chu and Beasley, 1997], [Easton and Mansour, 1993], [Harper et.al., 2005], [Tanomaru, 1995] and [Wilson, 1997] used genetic algorithm for solving either generalised assignment problem or manpower scheduling problem or project assignment problem. In their genetic algorithms, initialization, crossover and mutation processes have been reported differently.

In this paper, an assignment problem has been solved by a new approach using elitist genetic algorithm (EGA). Initially, the problem has been formulated as 0-1 programming problem. Then EGA has been developed to solve the problem. In our proposed GA, new methodologies for initialization, crossover and mutation have been developed instead of existing methodologies. However, for the reproduction/selection, existing rank-based selection has been used. Finally, different special cases of assignment problems like unbalanced, maximization and restricted assignment problems have been mentioned and the proposed method has been illustrated with some numerical examples.

2 Assumptions and notations

The following assumptions and notations are used in developing the proposed model:

(i) There are n jobs in a factory and the factory has n machines to process the jobs.

(ii) Each job can be associated with one and only one machine.

(iii) $C_{ij} \geq 0$ be the cost which is incurred when a job i $(i = 1, 2, ..., n)$ is processed by the machine j $(j = 1, 2, ..., n)$.

(iv) The crisp number x_{ij} denotes that the i^{th} job is assigned to the j^{th} machine.

(v) Each machine can perform each job but with varying degree of efficiency.

3 Mathematical formulation

Mathematical formulation of this problem is given by

$$\text{Minimize } Z = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, ..., n \tag{2}$$

and

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, ..., n \tag{3}$$

where $x_{ij} = 1$, if the i^{th} job is assigned to the j^{th} machine

= 0, if the i^{th} job is not assigned to the j^{th} machine.

The constraint (2) ensures that only one job is assigned to one machine while the constraint (3) ensures that only one machine should be assigned to one job.

4 Genetic Algorithm structure

Now, we shall develop an elitist GA (EGA) for solving the above mentioned constrained minimization problem involving n^2 integer variables x_{ij} (whose values are either 0 or 1). The working steps of our developed elitist GA are :

Step-1. Generate randomly a family of initial population of chromosomes (potential solutions).

Step-2. Calculate the fitness of each chromosome.

Step-3. Obtain the best found result from the initial population.

Step-4. Reproduce a new population for the next generation from the population of the earlier generation using ranking selection.

Step-5. Alter the chromosomes by crossover and mutation operations.

Step-6. Calculate the fitness of the population of child chromosomes of the next generation.

Step-7. (Perform elitist operation)
Obtain the best found result from the population of the current generation and compare that with the same of the previous generation and replace the worst result of the current population by the best found result of the previous generation if it is better than that of the current generation.

Step-8. Repeat Step-4 to Step-7 until the termination criterion is satisfied.

Step-9. Print the best found result.

Step-10.Stop

For implementing the above GA the following basic components are to be considered.

- Parameters of GA
- Representation of chromosomes
- Initialization
- Evaluation function
- Selection process
- Genetic operators (crossover and mutation)

4.1 Parameters of Genetic algorithm

Genetic Algorithm (GA) depends on some parameters like population size (p_{size}), maximum number of generation (m_{gen}), probability of crossover (p_c) and probability of mutation (p_m). About the population size of GA, though there is no clear indication, but for large population size, there arises a difficulty in storing of data. However, for small population size there may not be sufficient chromosomes for good crossover as well as mutation.

4.2 Representation of chromosomes

The first problem met during the implementation of GA is the appropriate representation of the chromosomes. This representation is connected to the specifications of the concerned optimization problem. Although binary strings have been favoured by many GA researchers for their chromosome representation, [Chu and Beasley, 1997] use the string of decimal numbers, in their approach to the generalised assignment problem (GAP). Our representation is similar to Chu and Beasley's GAP representation and specifies, for each job (task), the machine (person) number to which it is assigned. Thus,

in order to design an appropriate chromosome representation of solutions of our proposed GA, we have used square matrix of order *n* containing n^2 genes whose values represent the values of the crisp variable x_{ij} (either 0 or 1). This representation ensures that the constraints (2) and (3) are automatically satisfied.

4.3 Initialization

In GA, generally, each component (gene) of a chromosome of the population is generated randomly within the boundary of the component, satisfying all the constraints given in the problem. However, in the assignment problem, each gene in the chromosomes is defined for each job by randomly assigning a machine from the set of n machines. So, each component (gene) of chromosome of the assignment problem may be either 0 or 1. To generate all the components of a chromosome, we have proposed a new scheme for initialization of chromosomes by which all the constraints will be satisfied automatically. The steps are as follows:

Step-1. Set '0's to all the n^2 components (genes) of a chromosome.
Step-2. Randomly select a gene of the chromosome.
Step-3. Set a '1' in each row and in each column according to constraint equations (2) and (3).
Step-4. Stop.

In this way, p_{size} number of such chromosomes $(V_1, V_{2,}.....,V_{p_{size}})$ are generated randomly. All these chromosomes constitute the initial population of GA.

4.4 Evaluation function

After getting a population, we need to find out a chromosome which gives the better value (minimum) of the objective function. For this purpose, we have to calculate the fitness for each chromosome. In our case, the fitness of a chromosome is defined to be the value of the objective function due to the chromosome which is nothing but the sum of the costs (times) in the assignment cells represented by the chromosome.

4.5 Selection Process

The selection process is one of the most important factor in the genetic search. This process is stochastic and biased towards the best solutions. It depends on the evolutionary principle "Survival of the fittest". During the selection, the "parent" chromosomes aimed at producing the "child" chromosomes are chosen. Several methods exist in the literature to achieve this task. Initially, [Holland, 1976] used the selection process based on spinning the Roulette wheel (known as Roulette Wheel selection). However,

this selection has some limitations. In the proposed algorithm, rank-based selection method has been used. This selection procedure is not dependent on the actual values of the objective function, it is dependent on the position of arrangement of the fitness values from best to worst. The probability of the i^{th} chromosome being selected in this method is defined by

$$\text{P(select the } i^{th} \text{ chromosome)} = p(1-p)^{i-1}$$

where ' p ' is the probability of selecting the best chromosome and ' i ', the rank of the chromosome.

4.6 Crossover

The crossover operator is acknowledged as one of the main causes of the efficiency of GA: it generates improved offspring combining the beneficial features of their parents. It operates on two or more randomly selected highly fitted chromosomes at a time and generates offspring by recombining the parent chromosomes features. For this operation, expected $p_c.p_{size}$ number of chromosomes will take part. We have observed that in matrix representation for chromosomes, standard crossover schemes, like partially matched crossover (PMX), order crossover (OX) cannot produce chromosomes better than the parents in maximum cases. For this reason, we have tested three new crossover schemes, viz., (i) modified form of whole arithmetical crossover (MWAX) given in [Michalewicz, 1999], (ii) matrix binary crossover (MBX) and (iii) row exchange crossover (REX). Now we shall describe these schemes in details.

(i) Modified form of whole arithmetical crossover (MWAX)

Step-1. Select two chromosomes(in matrix form) $V_1 = (v_{ij}^{'})$ and $V_2 = (v_{ij}^{''})$ randomly from the population.

Step-2. Generate two temporary matrices $D = (d_{ij})$ and $R = (r_{ij})$ given by

$$d_{ij} = (v_{ij}^{'} + v_{ij}^{''})/2 \text{ [Integer division]}$$

and

$$r_{ij} = (v_{ij}^{'} + v_{ij}^{''}) \bmod 2 \text{ [Remainder division]}$$

Note that the matrix R has an interesting property for assignment problems. This property has been described in Theorem -1.

Theorem -1. For matrix R, the values of the marginal sums of rows or columns will be either '0' or '2'. (For Proof Ref. **Appendix A**)

Step-3. Decompose the matrix R into two matrices R_1 and R_2 such that

$$R = R_1 + R_2 .$$

where every row of both R_1 and R_2 has at most one '1' and all the elements of the column containing that element '1' must be 0s.

Step-4. Produce two offspring V_3 and V_4 given by

$$V_3 = D + R_1$$

$$V_4 = D + R_2$$

It is to be noted that decomposition of the matrix R into two matrices R_1 and R_2 is a formidable task in most of the cases. In those cases, offspring V_3 and V_4 will be infeasible. To make them feasible, we have to apply repair algorithm.

(ii) Matrix binary crossover (MBX)

Matrix binary crossover (MBX) is a natural extension of the conventional 1-point or 2-point crossover on strings and deals with column positions rather than bit (i.e., 0 or 1) positions. The steps are as follows:

Step-1. Randomly select two parent chromosomes (matrices) $P_1 = (p'_{ij})$ and $P_2 = (p''_{ij})$ from the population.

Step-2. Randomly select two crossover sites $s_1, s_2 \in \{2, 3, ..., n-1\}$ with $s_2 \geq s_1$. [n being the order of the matrix]

Step-3. Exchange all the entries of P_1 and P_2 determined by s_1 and s_2 to produce two child chromosomes $C_1 = (c'_{ij})$ and $C_2 = (c''_{ij})$ given by

$$c'_{ij} = p''_{ij},$$
$$c''_{ij} = p'_{ij}$$

where $i = 1, 2, ..., n$; $j = s_1, s_1 + 1, ..., s_2$.

(iii) Row exchange crossover (REX)

Step-1. Select two chromosomes (matrices) $P_1 = (p'_{ij})$ and $P_2 = (p''_{ij})$ randomly from the population as parents.

Step-2. Exchange successively the first row of P_1 with the last row of P_2, the second row of P_1 with the last but one row of P_2 and so on to produce two offspring $C_1 = (c'_{ij})$ and $C_2 = (c''_{ij})$ given by

$$c'_{ij} = c''_{n-i-1,j},$$
$$c''_{ij} = c'_{n-i-1,j}$$

where $i = 1, 2, ..., n; j = 1, 2, ..., n$.

4.7 Repair algorithm

In the earlier mentioned crossover methods (i) and (ii), as most of the offspring chromosomes (solutions) are infeasible, the said crossover methods need a scheme to repair infeasible solutions to feasible ones from time to time. For this purpose, we have used the following steps:

Step-1. Find rows containing duplicate '1's and rows containing no '1's (i.e., vacant rows).

Step-2. Remove the duplication by moving a '1' from each row with duplicate '1's into each vacant row.

Step-3. Set a '0' in the position of the row with duplicate '1's wherefrom '1' is removed.

In our study, this repair algorithm is treated as if it is a part of the crossover operation, although it is applied separately after offspring generated by crossover operation.

4.8 Mutation

Mutation introduces random variations into the population thereby increases genetic diversity. It is applied to a single chromosome only with lower probability. It attempts to bump the population gently into a slightly better one. In this operation, expected $p_m.p_{size}$ number of chromosomes will take part. In our GA, we have proposed three types of mutation as follows:

(i) Row/Column exchange mutation (REM/CEM)

For any randomly selected chromosome (in matrix form), randomly select any two rows/columns and interchange the values of the corresponding two rows/columns. In our experiments, we have used REM.

(ii) Inversion mutation (IM) given in [Gen and Cheng, 1997]

For any randomly selected chromosome (in matrix form), randomly choose two mutation sites along the rows/columns and invert the rows/columns of the sub-matrix specified by the sites. Here, we have chosen the sites along columns.

(iii) Displacement mutation (DM) given in [Gen and Cheng, 1997]

For any randomly selected chromosome (in matrix form), randomly choose two mutation sites along the rows/columns and insert the sub-matrix specified by the sites in a random position. In our study, we have chosen the sites along columns.

5 Various types of Assignment Problems:

5.1 Unbalanced Assignment Problem

Whenever the cost matrix [C_{ij}] is not a square one, such assignment problem is known as Unbalanced Assignment Problem. An unbalanced problem can be modified to a balanced one by simply introducing fictitious (dummy) jobs or machines whichever are necessary to convert [C_{ij}] into a square one with zero elements for dummy jobs or machines. After this, we can apply our method to the resulting problem.

5.2 Maximization Assignment Problem

Instead of minimizing the cost, an assignment problem may be concerned with the maximizing profit. A maximization problem can be converted into the usual minimization problem by the relation

$$\text{Max } Z = -\text{Min } (-Z) = -\text{Min } Z^*.$$

5.3 Restricted Assignment Problem

There are assignment problems having restrictions that one or more job(s)/operator(s) cannot be assigned to some particular machine(s). In our method, C_{ij} has been assigned a suitably large value for those cells of the matrix where such case(s) occur.

Table-1 GA version for different combinations of crossover and mutation

Crossover	Mutation	Version name
MWAX	REM	GA-1
MWAX	IM	GA-2
MWAX	DM	GA-3
MBX	REM	GA-4
MBX	IM	GA-5
MBX	DM	GA-6
REX	REM	GA-7
REX	IM	GA-8
REX	DM	GA-9

6 Numerical illustrations with discussion

To illustrate our algorithm, we have solved a large number of assignment problems using EGA which is coded in C++ programming language and the whole computational work has been done in Pentium IV PC with 1 GB RAM in LINUX environment. Among these, nine (9) problems are listed in the **Appendix B**

ranging from 25 variables to 100 variables. To solve the problems, we have developed 9 versions of our GA based on different types of crossover and mutation (Ref. Table-1). Because of the stochastic nature of GAs, all the results reported in this paper are obtained using 50 executions per problem.

Table-2 Parameters used for different problems

Prob. Set		AP-1	AP-2	AP-3	AP-4	AP-5	AP-6	AP-7	AP-8	AP-9
GA-1	p_{size}	147	500	72	500	50	400	729	700	640
	m_{gen}	200	600	30	600	40	600	600	800	600
	p_c	0.8	0.8	0.6	0.8	0.6	0.8	0.7	0.8	0.8
	p_m	1/49	0.01	1/36	0.01	1/25	0.01	1/81	0.01	1/64
GA-2	p_{size}	637	1000	72	800	70	700	891	100	1088
	m_{gen}	500	800	90	600	50	800	700	1200	1000
	p_c	0.8	0.6	0.8	0.8	0.6	0.7	0.7	0.8	0.8
	p_m	1/49	0.01	1/36	0.01	1/25	0.01	1/81	0.01	1/64
GA-3	p_{size}	588	1000	432	800	50	900	1053	1200	960
	m_{gen}	500	1200	200	1000	80	1200	1000	1200	2200
	p_c	0.8	0.7	0.8	0.8	0.8	0.8	0.7	0.8	0.8
	p_m	1/49	0.01	1/36	0.01	1/25	0.01	1/81	0.01	1/64
GA-4	p_{size}	637	800	288	800	60	800	810	800	896
	m_{gen}	600	900	200	900	60	900	900	900	900
	p_c	0.8	0.8	0.8	0.8	0.7	0.8	0.8	0.8	0.8
	p_m	1/49	0.01	1/36	0.01	1/25	0.01	1/81	0.01	1/64
GA-5	p_{size}	680	1200	288	1200	50	1000	810	1000	896
	m_{gen}	700	1500	200	1200	70	1200	900	2000	1000
	p_c	0.6	0.7	0.8	0.8	0.8	0.8	0.7	0.8	0.8
	p_m	1/49	0.01	1/36	0.01	1/25	0.01	1/81	0.01	1/64
GA-6	p_{size}	735	1200	576	1800	50	1200	972	1200	960
	m_{gen}	600	1000	300	1500	60	1000	1000	2200	1000
	p_c	0.7	0.7	0.7	0.7	0.7	0.8	0.7	0.6	0.6
	p_m	1/49	0.01	1/36	0.01	1/25	0.01	1/81	0.01	1/64
GA-7	p_{size}	147	900	432	1000	70	1000	729	1200	832
	m_{gen}	300	900	400	1200	50	1000	800	1500	900
	p_c	0.8	0.8	0.8	0.7	0.6	0.6	0.6	0.7	0.7
	p_m	1/49	0.01	1/36	0.01	1/25	0.01	1/81	0.01	1/64
GA-8	p_{size}	1029	1000	288	1200	60	1000	1053	1500	1024
	m_{gen}	1000	1200	200	1500	60	1200	900	1800	1500
	p_c	0.8	0.7	0.7	0.8	0.8	0.7	0.7	0.8	0.7
	p_m	1/49	0.01	1/36	0.01	1/25	0.01	1/81	0.01	1/64
GA-9	p_{size}	882	1500	576	1800	70	1500	1215	1200	1216
	m_{gen}	1000	1800	500	2000	80	1500	1600	2000	2200
	p_c	0.8	0.7	0.7	0.8	0.8	0.8	0.7	0.7	0.8
	p_m	1/49	0.01	1/36	0.01	1/25	0.01	1/81	0.01	1/64

In Table-2, different values of parameters for different GAs used for 9 problems have been presented. Here it is seen that

- population size (p_{size}) is proportional to the number of variables,
- maximum number of generations (m_{gen}) is different for different problems,
- the crossover probability (p_c) is its classical value viz., 0.6 to 0.8,
- the mutation probability (p_m) is equal to the reciprocal of the number of variables.

We also notice from Table-2 that p_{size} varies between 50 and 1216 while m_{gen} varies from 40 to 2200.

Table-3: Computational results of 9 different problems for different GA versions

Prob. Set		AP-1	AP-2	AP-3	AP-4	AP-5	AP-6	AP-7	AP-8	AP-9
GA-1	Best	215	61	333	283.75	37	55.25	228.25	155.5	101
	Worst	230	64	-	268.5	-	58.25	220	154	106
	Suc-rate(%)	86	85	100	75	100	70	98	70	96
	Time(s)	0.133	3.569	0.013	3.608	0.006	2.548	5.709	28.374	5.131
GA-2	Best	215	61	333	283.75	37	55.25	228.25	155.5	101
	Worst	232	78.25	-	272.375	-	71.125	216.5	148.75	109
	Suc-rate(%)	70	52	100	43	100	42	56	48	54
	Time(s)	3.263	14.043	0.028	23.484	0.012	6.010	9.113	21.526	20.153
GA-3	Best	215	61	333	283.75	37	55.25	228.25	155.5	101
	Worst	234	70	326	275	-	58.25	220	147	111
	Suc-rate(%)	60	62	92	64	100	62	60	64	60
	Time(s)	2.001	18.236	0.493	25.586	0.008	21.326	2.593	22.374	21.925
GA-4	Best	215	61	333	283.75	37	55.25	228.25	155.5	101
	Worst	-	64	-	273.625	-	58.25	-	154	106
	Suc-rate(%)	100	70	100	70	100	70	100	66	98
	Time(s)	4.726	12.198	0.344	12.317	0.007	12.351	10.999	12.147	12.861
GA-5	Best	215	61	333	283.75	37	55.25	228.25	155.5	101
	Worst	232	69	312	279	-	66.625	225.5	151.75	105
	Suc-rate(%)	80	66	88	75	100	70	80	68	66
	Time(s)	5.776	39.520	0.314	32.521	0.006	23.801	10.519	38.825	13.430
GA-6	Best	215	61	333	283.75	37	55.25	228.25	155.5	101
	Worst	235	68	322	275	-	64.5	220	148.25	109
	Suc-rate(%)	60	64	88	60	100	60	72	64	60
	Time(s)	3.412	35.352	0.993	49.974	0.005	16.777	9.682	35.879	8.947
GA-7	Best	215	61	333	283.75	37	55.25	228.25	155.5	101
	Worst	234	65	326	273.375	-	58.25	218	154	106
	Suc-rate(%)	74	66	64	60	100	60	98	66	88
	Time(s)	0.186	13.672	1.043	19.328	0.007	16.287	6.706	32.156	8.458
GA-8	Best	215	61	333	283.75	37	55.25	228.25	155.5	101
	Worst	227	71.75	323	272.375	-	71.125	221.5	150.5	109
	Suc-rate(%)	64	62	64	62	100	64	62	64	60
	Time(s)	15.847	18.119	0.269	28.457	0.007	18.943	14.958	31.224	9.584
GA-9	Best	215	61	333	283.75	37	55.25	228.25	155.5	101
	Worst	236	72	322	268.5	-	71	218	147.5	108
	Suc-rate(%)	60	64	66	60	100	62	60	66	62
	Time(s)	12.654	22.235	0.665	35.232	0.010	20.572	19.653	32.937	23.257

AP : Assignment Problem ; Best : Best solution obtained from GAs ; Worst : Worst solution obtained from GAs
Suc-rate(%) : Success rate of the best solution ; Time(s) : Average coputation time (in seconds) ; - : Nil.

In Table-3, the objective values of the best and the worst solution, success rates of the best solution and the average execution times of 50 trials (in CPU seconds) for each problem and also for each version of GA have been reported.

From Table-3, it is seen that the objective values of the best solution for each problem as obtained from different GAs remain the same. Further, it is observed that in general, our GAs perform well, as the success rate is 60% and more for the majority of the 81 results. Moreover, this rate is 100% for 14 cases. The average execution times are all within a reasonable time of 40 seconds which indicates that within a reasonable computation time, our GAs are able to find best found results for all the test problems.

As seen from Table-3, on an average, GA-1, GA-4 and GA-5 are the best, GA-3, GA-6, GA-7, GA-8 and GA-9 are the moderate whereas GA-2 is the worst method.

7 Conclusion:

In this paper, we have proposed an elitist genetic algorithm (EGA) as an aid for solving assignment problems. At first, these problems have been converted to equivalent linear programming problems. After that, to obtain the solutions, an EGA for discrete variables with new types of initialization, crossover and mutation have been developed. These operations together act as a very powerful search mechanism. As indicated from our experiments, modified form of whole arithmetical crossover (MWAX) with row exchange mutation (REM), matrix binary crossover (MBX) with row exchange mutation (REM) and matrix binary crossover (MBX) with inversion mutation (IM) are able to preserve good solutions for all the problems. Also it is found that row exchange mutation (REM) and inversion mutation (IM) are far better than displacement mutation (DM). The proposed algorithm shows good performances for all the test problems. The developed GA for integer variables has efficiently compared to an optimum integer programming approach, both for small and large assignments, as GA works with a set of potential solutions to the problem in each generation. In the existing literature, to design a chromosome of GA, ordinal representation, adjacency representation and path representations have been used. Among these representations, path representation has been used widely. In our approach, a matrix representation has been used to generate n^2 genes of a chromosome. The developed model can be improved in different ways. For future works in this area, one can perform similar analysis for the problems with interval valued assignment cost/profit/time and multi-objective assignment problems.

Acknowledgement:

The authors would like to acknowledge the support of Research Project provided by the UGC, India, for conducting this research work.

References:

1. Aickelin, U., and Dowsland, K.A., (2004) An indirect Genetic Algorithm for a nurse-scheduling problem. Computers and Operations Research, vol.31, pp. 761-778.
2. Bradley, D., and Martin, J. (1990) Continuous personnel scheduling algorithms: a literature review. Journal of the Society for Health Systems, vol.2, pp. 8-23.
3. Catrysse, D., and Van Wassenhove, L. N. (1992) A survey of algorithms for the generalized assignment problem. European Journal of Operational Research, vol.60, pp. 260-272.
4. Chu, P.C., and Beasley, J.E. (1997) A genetic algorithm for the generalized assignment problem. Computers and Operations Research, vol.48, pp. 17-23.
5. Davis, L. (1991) Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York.

6. Deb, K. (1995) Optimization for Engineering Design-Algorithms and Examples. Prentice Hall of India, New Delhi.
7. Dowsland, K.A. (1998) Nurse scheduling with Tabu search and strategic oscillation. European Journal of Operational Research, vol.106, pp. 393-407.
8. Dowsland, K.A., and Thompson, J. M. (2000) Nurse scheduling with knapsacks, networks and Tabu Search. Journal of the Operational Research Society. pp. 825-833.
9. Easton, F., and Mansour, N. (1993) A distributed Genetic Algorithm for employee staffing and scheduling problems. In: Forrest S, editor, Proceedings of the Fifth International Reference on Genetic Algorithms. San Mateo: Morgan Kaufmann Publishers. pp. 360-367.
10. Gen, M., and Cheng, R. (1997) Genetic Algorithms and Engineering Design. Wiley, New York.
11. Goldberg, D.E. (1989) Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, New York.
12. Harper, P.R., Senna, V., Vieira I.T., and Shahani, A.K. (2005) A genetic algorithm for the project assignment problem. Computers and Operations Research, vol.32, pp. 1255-1265.
13. Holland, J.H. (1976) Adaptation of Natural and Artificial system. University of Michigan Press, Ann Arbor, MI.
14. Lorena L., and Narciso, M.G. (1996) Relaxation heuristics for a generalized assignment problem. European Journal of Operational Research, vol.91, pp. 600-610.
15. Michalawicz, Z. (1999) Genetic Algorithms + Data structures = Evolution Programs. Springer-Verlag, Berlin.
16. Ross, G.T., and Soland, R.M. (1975) A branch and bound algorithm for the generalized assignment problem. Mathematical Programming, vol.8, pp. 91-103.
17. Sakawa, M. (2002) Genetic Algorithms and fuzzy multiobjective optimization. Kluwer Academic Publishers.
18. Tanomaru, J. (1995) Staff scheduling by a Genetic Algorithm with heuristic operators. Proceedings of the IEEE Conference on Evolutionary Computation, New York, pp. 456-461.
19. Wilson, J.M. (1997) A genetic algorithm for the generalized assignment problem. Journal of the Operational Research Society, vol. 24.

Appendix A

Proof of the Theorem-1 :

In V_1 and V_2 there is one and only one '1' in each row and in each column. For a particular row (say, i) these '1's lie either in the same column or in the different columns of V_1 and V_2, viz., $v'_{ik}=1,\ v''_{ik}=1$ or $v'_{ik}=1,\ v''_{il}=1$. In the former case, the above mentioned remainder division will give $r'_{ik}=0$ and in the

later case, this will give $r'_{ik} = 1,\ r'_{il} = 1$. Hence, the marginal sum of the elements of i^{th} row will be either '0' (for the first case) or '2' (for the second case) i.e.,

$$\sum_{i=1}^{n} r_{ij} = 0 \text{ or } 2\ ; \quad i = 1, 2, ..., n.$$

Similar analysis will prove the result for a particular column (say, *j*). Hence our proof is complete.

Appendix B

List of matrices $([C_{ij}]_{n\times n})$ used for test problems

Bal. Min. (AP-1) : 7 × 7

$$\begin{pmatrix} 35 & 22 & 60 & 41 & 27 & 52 & 44 \\ 51 & 39 & 42 & 33 & 65 & 47 & 58 \\ 25 & 32 & 53 & 41 & 50 & 36 & 43 \\ 32 & 28 & 40 & 46 & 3 & 55 & 49 \\ 43 & 36 & 45 & 63 & 57 & 49 & 42 \\ 27 & 18 & 31 & 46 & 35 & 42 & 34 \\ 48 & 50 & 72 & 59 & 43 & 64 & 58 \end{pmatrix}$$

Bal. Min. (AP-2) : 10 × 10

$$\begin{pmatrix} 17.75 & 18 & 15 & 13.25 & 11.25 & 10 & 5 & 6.5 & 9.5 & 12 \\ 16 & 17.5 & 16.5 & 14.75 & 12.75 & 11.5 & 9.25 & 5 & 5 & 10.25 \\ 13.5 & 15 & 17.5 & 17 & 15 & 13.75 & 11.5 & 10.25 & 5.25 & 5.25 \\ 12 & 13.5 & 16 & 17.5 & 16.75 & 15.5 & 13.25 & 12 & 9 & 6.25 \\ 9.5 & 11 & 13.5 & 15 & 17.75 & 18 & 15.75 & 14.5 & 11.5 & 9.5 \\ 6 & 4.5 & 11 & 12.5 & 15.25 & 16.5 & 18.25 & 17 & 14 & 12 \\ 4.25 & 5.75 & 4.25 & 9.75 & 12.5 & 13.75 & 15.5 & 17.25 & 16.75 & 14.75 \\ 10 & 4.75 & 5.5 & 8.75 & 11.5 & 12.75 & 14.5 & 16.25 & 18 & 16 \\ 11.5 & 10 & 6 & 5.25 & 10.25 & 11.5 & 13.25 & 15 & 17.75 & 17.5 \\ 14 & 12.5 & 9.5 & 5 & 5.75 & 4.5 & 10.75 & 12.5 & 15.25 & 17.75 \end{pmatrix}$$

Bal. Max. (AP-3) : 6 × 6

$$\begin{pmatrix} 9 & 22 & 58 & 11 & 19 & 27 \\ 43 & 78 & 72 & 50 & 63 & 48 \\ 41 & 28 & 91 & 37 & 45 & 33 \\ 74 & 42 & 27 & 49 & 39 & 32 \\ 36 & 11 & 57 & 22 & 25 & 18 \\ 13 & 56 & 53 & 31 & 17 & 28 \end{pmatrix}$$

Bal. Max. (AP-4) : 10 × 10

$$\begin{pmatrix} 28.625 & 28.875 & 26.375 & 24.875 & 11.125 & 11.125 & 16.375 & 17.375 & 11.25 & 10.75 \\ 27.125 & 28.625 & 28.125 & 26.625 & 11.375 & 11.375 & 11.75 & 16.125 & 16.375 & 11 \\ 24.875 & 26.375 & 29.375 & 29.125 & 26.375 & 25.125 & 12 & 11.5 & 16.875 & 16.375 \\ 11.375 & 24.625 & 27.625 & 29.375 & 28 & 26.75 & 25 & 11.375 & 11.625 & 17.125 \\ 11.25 & 11.25 & 25.125 & 26.875 & 29 & 29.125 & 27.375 & 25.625 & 11.5 & 11 \\ 17.25 & 15.75 & 11.875 & 24.375 & 26.5 & 27.75 & 29.875 & 28.25 & 25.5 & 11.125 \\ 15.375 & 16.875 & 15.875 & 12 & 11.375 & 25 & 27.125 & 28.375 & 28.125 & 25.625 \\ 11 & 15.625 & 16.875 & 11.75 & 11.125 & 11.125 & 25.875 & 27.125 & 20.125 & 26.625 \\ 10.875 & 10.875 & 17.125 & 16.75 & 11 & 11 & 24.375 & 25.625 & 28.625 & 28 \\ 24.625 & 10.75 & 11.25 & 16.375 & 16.375 & 15.125 & 11.25 & 10.75 & 26.125 & 28.125 \end{pmatrix}$$

Unbal. Min. (AP-5) : 5 × 5

$$\begin{pmatrix} 11 & 17 & 8 & 16 & 0 \\ 9 & 7 & 12 & 6 & 0 \\ 13 & 16 & 15 & 12 & 0 \\ 21 & 24 & 17 & 28 & 0 \\ 14 & 10 & 12 & 11 & 0 \end{pmatrix}$$

Unbal. Min. (AP-6) : 10 × 10

$$\begin{pmatrix} 17.625 & 17.875 & 14.875 & 13.125 & 11 & 9.75 & 4.875 & 6.375 & 9.375 & 11.875 \\ 15.875 & 17.375 & 16.375 & 14.625 & 12.5 & 11.25 & 9.125 & 4.875 & 4.875 & 10.125 \\ 13.375 & 14.875 & 17.375 & 16.875 & 14.75 & 13.5 & 11.375 & 10.125 & 5.125 & 5.125 \\ 11.75 & 13.25 & 15.75 & 17.25 & 16.5 & 15.25 & 13.125 & 11.875 & 8.875 & 6 \\ 9.375 & 10.875 & 13.375 & 14.875 & 17.625 & 17.75 & 15.625 & 14.375 & 11.375 & 9.375 \\ 5.875 & 4.375 & 10.75 & 12.25 & 15 & 16.25 & 18 & 16.875 & 13.875 & 11.875 \\ 4.125 & 5.625 & 4.125 & 9.625 & 12.375 & 13.625 & 15.375 & 17.125 & 16.625 & 14.625 \\ 9.875 & 4.625 & 5.375 & 8.625 & 11.375 & 12.625 & 14.375 & 16.125 & 17.875 & 15.875 \\ 11.375 & 9.875 & 5.75 & 5.125 & 10 & 11.25 & 13 & 14.75 & 17.5 & 17.375 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Unbal. Max. (AP-7) : 9 × 9

28.75	29	26.5	25	11.5	11.5	16.5	17.5	0
27.25	28.75	28.25	26.75	11.75	11.75	12	16.25	0
25	26.5	29.5	29.25	26.5	25.25	12.25	11.75	0
11.75	24.75	27.75	29.5	28.25	27	25.25	11.75	0
11.5	11.5	25.25	27	29.25	29.25	27.5	25.75	0
17.5	16	12.25	24.5	26.75	28	30	28.5	0
15.5	17	16	12.25	11.75	25.25	27.25	28.5	0
11.25	15.75	17	12	11.5	11.5	26	27.25	0
11.25	11.25	17.25	17	11.5	11.5	24.5	25.75	0

Unbal. Max. (AP-8) : 10 × 10

16	17.5	16.5	14.75	12.75	11.5	9.25	5	5	10.25
13.5	15	17.5	17	15	13.75	11.5	10.25	5.25	5.25
12	13.5	16	17.5	16.75	15.5	13.25	12	9	6.25
9.5	11	13.5	15	17.75	18	15.75	14.5	11.5	9.5
6	4.5	11	12.5	15.25	16.5	18.25	17	14	12
4.25	5.75	4.25	9.75	12.5	13.75	15.5	17.25	16.75	14.75
10	4.75	5.5	8.75	11.5	12.75	14.5	16.25	18	16
11.5	10	6	5.25	10.25	11.5	13.25	15	17.75	17.5
14	12.5	9.5	5	5.75	4.5	10.75	12.5	15.25	17.75
0	0	0	0	0	0	0	0	0	0

Restricted (AP-9) : 8 × 8

25	15	–	24	26	27	29	28
25	11	15	16	15	12	–	25
27	28	25	27	11	15	16	11
14	12	25	27	20	26	16	–
16	14	12	11	12	15	10	18
13	15	16	18	15	13	11	–
15	25	12	15	16	–	15	18
12	15	20	16	15	18	12	24