A Genetic Algorithm for the Multiple Knapsack Problem in Dynamic Environment

Ali Nadi Ünal

***Abstract*—The 0/1 Multiple Knapsack Problem is an important class of combinatorial optimization problems, and various heuristic and exact methods have been devised to solve it. Genetic Algorithm (GA) shows good performance on solving static optimization problems. However, sometimes lost of diversity makes GA fail adapt to dynamic environments where evaluation function and/or constraints or environmental conditions may change over time. Several approaches have been developed for increasing the diversity of GA into dynamic environments. This paper compares two of these approaches named Random Immigrants Based GA (RIGA) and Memory Based GA (MBGA). Results show that MBGA is more effective than RIGA for The 0/1 Multiple Knapsack Problem in a changing environment.**

***Index Terms*—Genetic Algorithms, Multiple Knapsack Problem, Dynamic Environments**

I. Introduction

MKP is a well-known NP-hard optimization problem since any dynamic programming solution will produce results in exponential time [1]. Other names given to this problem in related literature are "the multi-constraint knapsack problem", "the multi-knapsack problem", "the m-dimensional knapsack problem" and "the multidimensional knapsack problem" [2]. Lots of researchers also include "zero-one" in their name for the problem. The problem can be represented as follows:

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} r_{ij} x_j \le b_i, \; i=1,2,\ldots,m, \tag{2}$$

$$x_j \in \{0,1\}, \; j=1,2,\ldots,n. \tag{3}$$

Constraints are defined in (2) for *m* knapsacks. Each knapsack has a capacity b_i. We have *n* objects, each has profit p_j. Unlike the simple version in which the weights of the objects are fixed, the weight of the *jth* object in the multiple knapsack problem takes *m* values. The *jth* object weighs r_{ij} when it is considered for possible inclusion in the *ith* knapsack of capacity b_i. As shown in (3), the *x* can only takes binary values. We are interested in finding a vector $\vec{x}=(x_1, x_2, \ldots, x_n)$ that guarantees no knapsack is overfilled and that yields the maximum profit.

Several approaches have been suggested to solve MKPs. Different proposed algorithms can be broadly grouped into two classes; (i) exact algorithms, and (ii) heuristics and metaheuristics. A good review of these approaches is given in [3].

Both practitioners and theoreticians are interested in the knapsack problems. Practitioners enjoy the fact that these problems can model many industrial opportunities such as cutting stock, cargo loading, capital budgeting, menu planning, project selection and processor allocation [1,4]. On the other hand theoreticians think that these simple structured problems can be used as sub problems to solve more complicated ones [4].

There is a huge amount of literature on genetic algorithms for various problems and different phases of the algorithm. However, this project is restricted to a specific benchmark problem (i.e. weingartner2 multiple knapsack problem). For this reason only necessary explanation (used techniques in this paper) is given about genetic algorithms and the given problem (i.e. dynamic MKP) through following paragraphs.

The first step of designing a genetic algorithm is creating an initial population that consists of individuals (chromosomes). We need to devise a suitable representation scheme to represent individuals in the population. It's proper to use standard GA 0-1 binary representation for the MKP since it represents the underlying 0-1 integer variables [2]. An example chromosome of the seven-item knapsack problem is illustrated in Fig.1.

1	2	3	4	5	6	7
0	1	0	1	0	1	0

Fig. 1. Binary code for knapsack problem [5].

The upper numbers in Fig. 1 represent the order number of the items and the lower numbers represent the selection of the items. But this code scheme might generate infeasible solutions. An infeasible solution means that at least one of the knapsack constraints is violated.

There are number of standard ways of dealing with constraints and infeasible solutions in GAs [2] such as; (i) using a representation that automatically ensures that all solutions are feasible, (ii) separating the evaluation of fitness and infeasibility, (iii) designing a heuristic operator which guarantees to transform any feasible solution into a feasible one, and finally (iv) applying a penalty function to penalize the fitness of any infeasible solution without distorting the fitness landscape.

In this paper, infeasible solutions are allowed to join the population by adding a penalty term to its fitness, as in [4]. In their work [4], they emphasize that the farther away from feasibility the string is, the higher its penalty term should be.

Manuscript received July 19, 2013; revised August 12, 2013. Ali Nadi Ünal is with the Aeronautics and Space Technologies Institute (ASTIN) , İstanbul, Türkiye (Phone: +90-530-520-0950; e-mail: anunal@ hho.edu.tr).

Taking this explanation into consideration, the fitness function to be maximized is:

$$f(\vec{x}) = \sum_{j=1}^{n} p_j x_j \text{ -s.max}\{p_j\} \quad (4)$$

where *s* denotes the number of overfilled knapsacks.

For Weingartner 2 benchmark problem, there are 2^{28} possible solutions of the problem regardless of the feasibility assumption. But it can be said that, the more 1s in any chromosome the higher probability to violate the constraints. For this reason, it's suggested biasing the random number generator so as to produce strings in which the number of zeros is greater than the number of ones [4]. This suggestion is adopted in this paper to deal with the benchmark problem.

For selection phase of genetic algorithms, several methods are used in literature such as roulette wheel, stochastic universal sampling, sigma scaling, rank selection, tournament selection and steady state selection. For a detailed explanation for these methods, reader should refer to [6].

In this paper the tournament selection is used for parent selection. The tournament selection strategy provides selective pressure by holding a tournament competition among individuals. The best individual (the winner) from the tournament is the one with the highest fitness value,. The tournament competition is repeated until the mating pool for generating new offspring is filled. The mating pool comprising of the tournament winner has higher average population fitness. The fitness difference provides the selection pressure, which drives GA to improve the fitness of the succeeding genes. This method is more efficient and leads to an optimal solution [7]. For this paper, tournament size is determined as 5.

After creating the mating pool, the next phase is crossover. For binary representations, several methods are explained in [8]. Among these crossover operators (i.e., one point crossover, N-point crossover and uniform crossover), one point crossover is used in this paper. It works by choosing a random number in the range [0, l-1] (l denotes the length of encoding) and then splitting both parents at this point and creating the two children by exchanging the tails [8]. Fig. 2 illustrates the process.

Parents							Children						
0	1	0	1	***1***	***0***	***0***	0	1	0	1	0	0	0
1	0	0	1	0	0	0	1	0	0	1	***1***	***0***	***0***

Fig. 2. One point crossover.

Once crossing over is applied to the population, a mutation procedure is performed. The chosen procedure (e.g., bitwise mutation, creep mutation, random resetting, uniform mutation, swap mutation and etc.) depends on the encoding used [8]. The most common mutation operator used for binary encodings is called bitwise mutation, for this reason in this paper bitwise mutation is implemented to the random individuals through random bits. Fig. 3 illustrates an example for bitwise mutation.

Before	1	0	1	0	0	0	0	1	0
After	1	0	0	1	0	0	0	0	0

Fig. 3. Bitwise mutation

As shown in Fig. 3, the 3rd, 4th, and 8th bits are changed.

Mutation is often interpreted as a "background operator" supporting the crossover operator [4, 9]. For this reason the rate of mutation is generally set to be a small value (e.g., 1 or 2 bits per string) [2]. In this paper mutation probability for an individual is 0.01 and, 2 bits per string.

MKPs can be solved using genetic algorithms like many other combinatorial optimization problems (COPs)[10]. Because of their ability to sample the search space, their ability to simultaneously manipulate a group of solutions, and their potential for adaptability, evolutionary algorithms (EAs) have been successfully applied to most COPs [11]. This judgment is mostly true for stationary environments.

When the environment changes over time, resulting in modifications of the fitness function from one cycle to another, we say that we are in the presence of a dynamic environment [12]. Genetic algorithms' tendency to converge rapidly to an optimum is a disadvantage in dynamic environments. Lack of diversity is another problem. Maintenance of the diversity is an essential requirement to apply the GA to dynamic environments [13].

Several approaches have been developed into GAs to address dynamic optimization problems, such as maintaining diversity during run via random immigrants, increasing diversity after a change, using memory schemes to reuse old good solutions and multi-population approaches [14]. In this paper the random immigrants approach and memory based approach are implemented separately.

The random immigrants approach is a quite natural and simple way around the convergence problem [14]. It maintains the diversity level of the population by replacing some individuals of the current population with randomly created individuals for every generation. The outgoing individuals from the current population may be either the worst ones or randomly chosen individuals. In this paper the replacement rate is 0.1. The worst individuals in the current population are replaced with randomly generated immigrants. According to [12] the best results for RIGA approach achieved with a replacement rate of 0.1.

The memory based approach works by storing useful information from the current environment. The stored information can be reused later in changing environments [14]. Storing useful information may be either implicitly through redundant representations or explicitly by storing good solutions in an extra memory space.

For explicit memory scheme, when the environment changes, old good solutions in the memory that well fit the new environment will be reactivated. Especially, when the environment changes periodically, the memory approach can work very well because with time going, an old environment will reappear exactly and the associated solution in the memory, which exactly remembers the old environment, will instantaneously move the algorithm to the reappeared environment [14]. The memory retrieval can be done periodically every generation or only when the environment changes.

For updating memory, several replacement strategies can be used [14]. These strategies are; (i) replacing the least important one with respect to the age, contribution to

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

diversity and fitness, (ii) replacing the one with least contribution to memory variance, (iii) replacing the most similar one if the individual is better or (iv) replacing the less fit one of a pair of memory points that have the minimum distance among all pairs.

In this paper, explicit memory scheme is used. The memory size is 10 (0.1*population size). Firstly, the memory is initialized choosing the best 10 solutions from the initial population. This memory is now fixed, and will not be updated through generations. When the environment changes, the memory is mounted to the population (i.e., replacing the last 10 orders of the population matrix with memory).

II. METHODS

Experiments were carried out to compare RIGA and MBGA in a dynamic environment. Weingartner 2 benchmark multiple knapsack problem was used. This problem can be downloaded from http://elib.zib.de/pub/Packages/mp-testdata/ip/sac94-suite/. In this problem there exist 28 objects and 2 knapsacks. Each knapsack has a capacity of 500. The optimal solution for this problem is reported as 130.883. MATLAB 7.6.0 was used for coding the algorithm.

For each of the two approaches, parameters were set as follows: generational GA with elitism of only 1 individual, one point crossover with probability 0.7, bitwise mutation with probability 0.01 and 2 bits per individual and tournament selection to create mating pool (parent selection). Population size was 100 and tournament size for selection was 5. Generation number was set to 2000. For 3 different statement (i.e., no action for changing environment, RIGA and MBGA), 50 independent runs with the same seed values were executed.

Environmental change was implemented periodically for 3 different period values for both GAs and the results were compared. Period values are 10, 100 and 500. It means that when the change period value is 10, at every 10 generation the first knapsack's capacity oscillates between 400 and 500 (i.e., after the 10th generation the capacity is 400, after the 20th generation the capacity is 500 again and so on so forth). For the period value 10, 2000/10=200 changes are counted totally. At each iteration, the best fitness values of the population through the 50 runs were recorded.

The overall performance of the approaches is formulated as follows [14]:

$$F_{BOG} = 1/g \sum_{i=1}^{g} [1/n \sum_{j=1}^{n} F_{BOG_{ij}}] \quad (5)$$

Where g is the number of generations for a run, n=50 is the total runs, F_{BOG_j} is the best-of-generation fitness of generation i of run j, and F_{BOG} is the offline performance, i.e., the best-of-generation fitness averaged over the 50 runs and over the data gathering period.

III. RESULTS AND DISCUSSION

First of all, for the benchmark static 0/1 Multiple Knapsack Problem, 50 runs were executed. The best result achieved through 50 runs is 130883 (items included: 3,5,7,8,10,11,14,19,21,23,24). This shows that the algorithm can find the optimal solution within 50 runs. The optimal value was reached at 40th run. The seed value for this run was set to 40.

When the algorithm find the optimal solution, the path of algorithm with respect to fitness values through 2000 iterations is shown in Fig. 4.

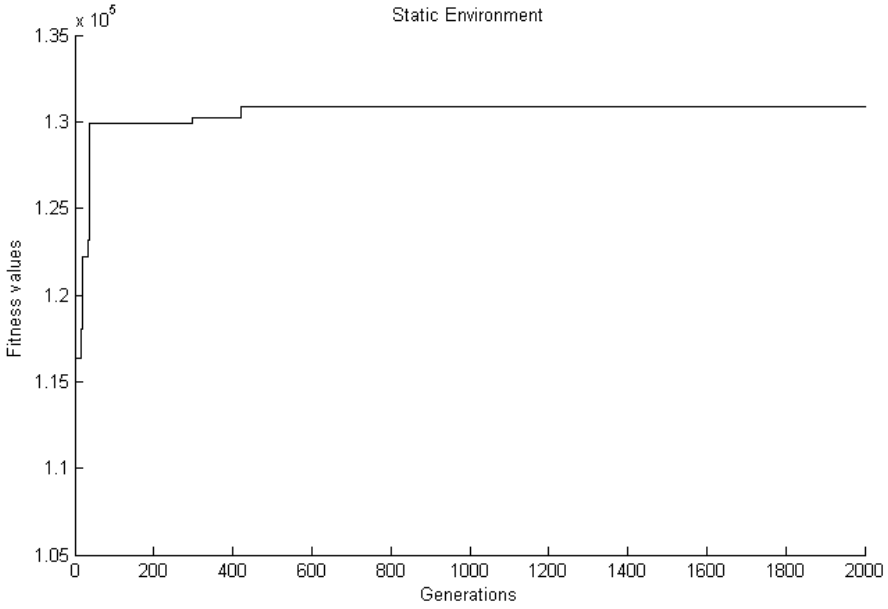

Fig. 4. Fitness value through generations for static problem.

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

After checking the algorithm works properly for static environments, it's time to run it for the dynamic environment that was mentioned above in Section 3. To compare the "RIGA approach", the "MBGA approach" and "no action" for changing environment each other, 50 runs for each algorithm was executed with same parameters and same random seeds for 3 different environment. The offline performances of the used approaches are compared, and the results are shown in Table 1.

TABLE 1
THE OVERALL PERFORMANCE OF 50 RUNS.

	Period Values		
Approaches	*10*	*100*	*500*
No Action	106744	107870	115069
RIGA	110102	110771	115476
MBGA	119035	118837	119799

As shown in Table 1, both methods that was used to increase diversity seem to work well compared to doing nothing (i.e., no action). Furthermore, the MBGA outperforms the RIGA for all period values. It was necessary to test whether the results were statistically significant. To this end, hypothesis testing was applied to the data gathered.

When the population variances unknown, the sample variances can be used to test a hypothesis for large samples (n>40) using Z_0 test statistic, and also the normality assumption is not necessary (Montgomery & Runger, 2003, p. 329). Results for hypothesis testing are shown in Table 2.

Test statistic values in Table 2 confirm the claims mentioned above. Both MBGA and RIGA are better than doing nothing for changing environment, but the case the period value equals 500. When the period value is 500, the change in environment is considerably slow. The MBGA outperforms the others again, but we don't have strong evidence to conclude that RIGA is more effective than doing nothing. For such a slow changing environment, using the RIGA and "no action" preference does not seem to have a difference at all.

From Fig. 5 to 7, it can be seen the dynamic behavior of the GA for different environmental change periods. In these figures, the average fitness values of each run through iterations are plotted, (i.e., the sum of first iterations' fitness values over 50 runs is divided by 50). As shown in the figures below, the MBGA outperforms the others for all environments.

TABLE 2
HYPOTHESIS TESTING (ONE-TAILED)

	μ_1	μ_2	H_0	H_a	$\bar{x}_1$	$\bar{x}_2$	s_1	s_2	Test Statistic z_0	Reject/Fail to reject H_0
Period values 10	No action	MBGA	$\mu_1 = \mu_2$	$\mu_1 < \mu_2$	106744	119035	8695	3563	-9.2490	Reject
	No action	RIGA	$\mu_1 = \mu_2$	$\mu_1 < \mu_2$	106744	110102	8695	6411	-2.1980	Reject
	RIGA	MBGA	$\mu_1 = \mu_2$	$\mu_1 < \mu_2$	110102	119035	6411	3563	-8.6121	Reject
100	No action	MBGA	$\mu_1 = \mu_2$	$\mu_1 < \mu_2$	107870	118837	6952	3757	-9.8135	Reject
	No action	RIGA	$\mu_1 = \mu_2$	$\mu_1 < \mu_2$	107870	110771	6952	6338	-2.1805	Reject
	RIGA	MBGA	$\mu_1 = \mu_2$	$\mu_1 < \mu_2$	110771	118837	6338	3757	-7.7411	Reject
500	No action	MBGA	$\mu_1 = \mu_2$	$\mu_1 < \mu_2$	115069	119799	5608	3434	-5.0862	Reject
	No action	RIGA	$\mu_1 = \mu_2$	$\mu_1 < \mu_2$	115069	115476	5608	5005	-0.3829	*Fail to reject*
	RIGA	MBGA	$\mu_1 = \mu_2$	$\mu_1 < \mu_2$	115476	119799	5005	3434	-5.0361	Reject

$n_1=n_2=50$, $\alpha=0.05$,- $z_\alpha=-1.645$

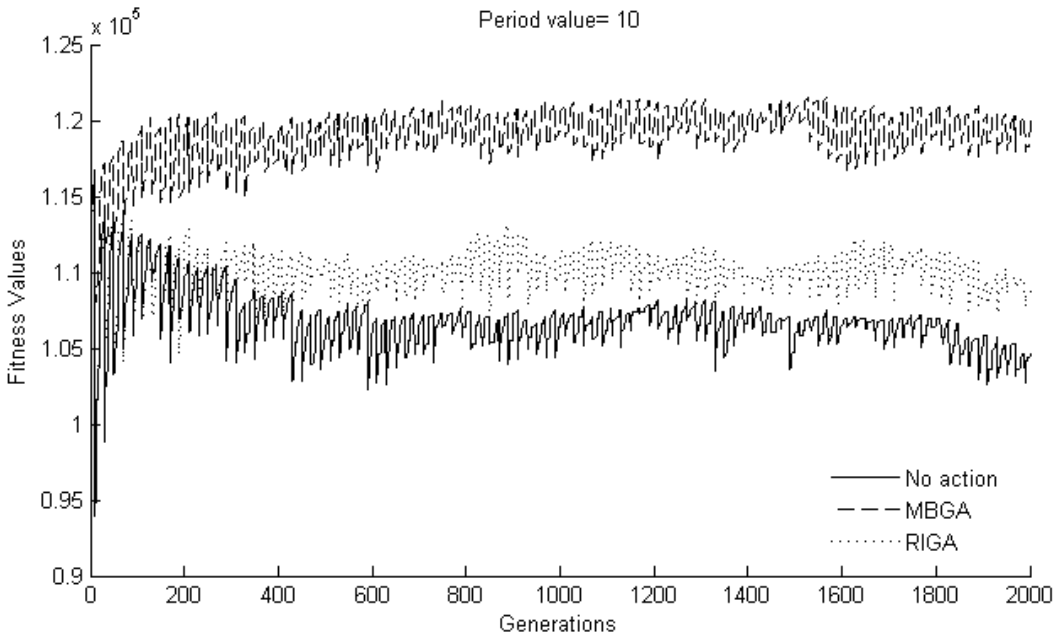

Fig. 5. Dynamic behavior of the GA for period value 10.

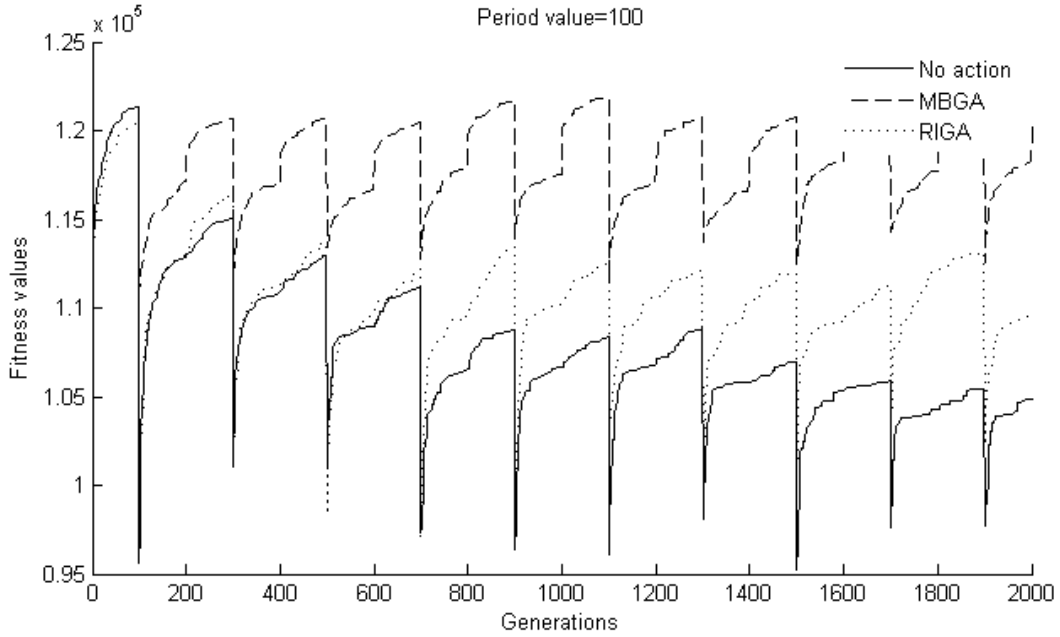

Fig. 6. Dynamic behavior of the GA for period value 100.

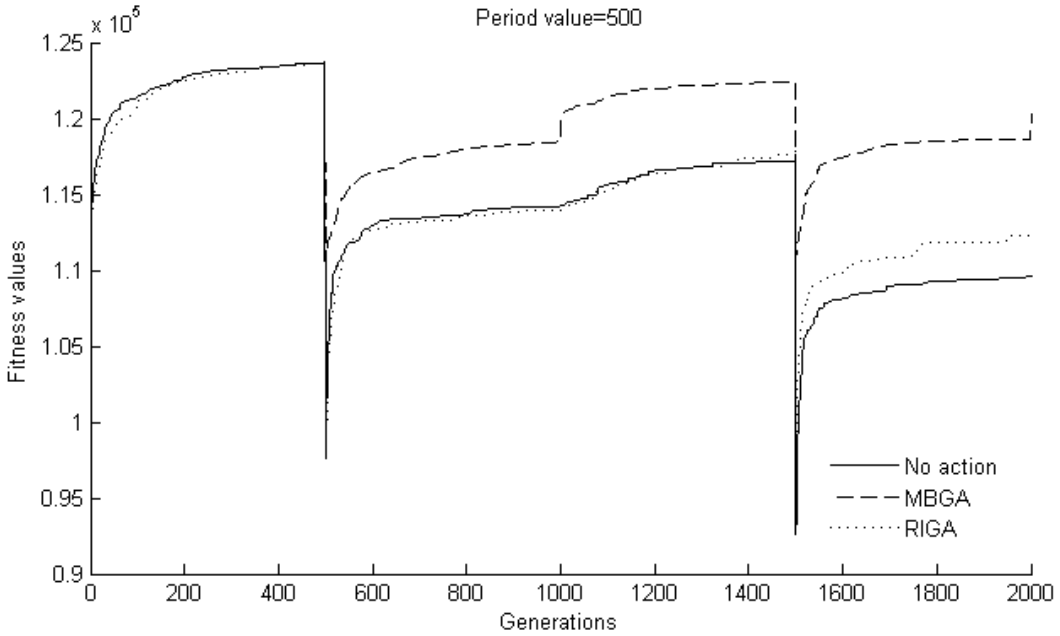

Fig. 7. Dynamic behavior of the GA for period value 500.

III. Conclusion

In this paper a GA for the 0/1 Multiple Knapsack Problem (MKP) is developed and its performance for dynamic environments is compared using 2 different approaches, namely random immigrants based genetic algorithm (RIGA) and memory based genetic algorithm (MBGA). Offline performance is used a measure to compare two approaches, among 3 different changing environment (period values, 10, 100, 500).

The GA that is developed using simple genetic operators (binary encoding, one point crossover, bitwise mutation, tournament selection) is highly effective for 0/1 MKP. For dynamic environments both RIGA and MBGA is useful to increase diversity. The results show that MBGA outperforms RIGA for all statements. For the period value 500 using the RIGA does not seem effective at all, compared to doing nothing.

References

[1] A. Zaheed and I. Younas, "A Dynamic Programming based GA for 0-1 Modified Knapsack Problem" *International Journal of Computer Applications* , vol. 16, No. 7, pp. 1-6, 2011.

[2] P.C. Chu and J. Beasley, "A Genetic Algorithm for the Multidimensional Knapsack Problem", *Journal of Heuristics* , vol. 4, No.1, pp 63-86, June 1998.

[3] M. Varnamkhasti, "Overview of the Algorithms for Solving the Multidimensional Knapsack Problems" *Advanced Studies in Biology* , vol. 4, No. 1, pp. 37-47, 2012.

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

[4] S. Khuri, T. Baeck and J. Heitkötter, "The Zero/One Multiple Knapsack Problem and Genetic Algorithms", in Proc. *ACM Symposium on Applied Computing (SAC'94)*, New York, 1994, pp. 188-193.
[5] X. Yu and M. Gen, *Introduction to Evolutionary Algorithm,* London: Springer Verlag-London Limited, 2010, p. 272.
[6] M. Mitchell, *An Inroduction to Genetic Algorithms,* Cambridge: The MIT Press, 1998, pp. 124-127.
[7] S. N. Sivanandam and S.N. Deepa, *Introduction to Genetic Algorithms,* Berlin: Springer-Verlag, 2008, pp. 48-49.
[8] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, Berlin: Springer, 2003, ch. 4.
[9] T. Baeck, D. B. Fogel, D. Whitley, and P. J. Angeline, "Mutation Operators" in *Evolutionary Computation 1: Basic Algorithms and Operators,* T.Baeck, D.B. Fogel and Z. Michalevicz, Eds., Bristol: Institute of Physics Publishing, 2000, pp. 237-255.
[10] M. Gen and R. Cheng, R *Genetic Algorithms and Engineering Optimization,* New York: John Wiley and Sons, 2000, p. 78.
[11] A. Younes, S. Aeribi, P. Calamai, and O. Basir, "Adapting Genetic Algorithms for Combinatorial Optimization Problems in Dynamic Environments" in *Advances in Evolutionary Algorithms,* W. Kosinski, Ed., Viena: I-Tech Education and Publishing, 2008, pp. 207-230
[12] A. Simoes and E. Costa, " Using Genetic Algorithms to Deal with Dynamic Environments: A Comparative Study of Several Approaches Based on Promoting Diversity", in Proc. Genetic and Evolutionary Computation Conference (GECCO'02), San Francisco, California: July, 2002, p. 698.
[13] N. Mori and H. Kita, " Genetic Algorithms for Adaptation to Dynamic Environments-A Survey", in Proc. Industrial Electronics Society, 2000. IECON 2000. 26th Annual Conference of the IEEE, Nagoya: 2000, pp. 2947-2952.
[14] S. Yang, "Memory-Based Immigrants for Genetic Algorithms in Dynamic Environments", in Proc. Genetic and Evolutionary Computation Conference (GECCO'05), Washington DC: June, 2005, pp. 1115-1122.
[15] D.C. Montgomery and G. C. Runger, *"Applied Statistics and Probability for Engineers 3rd ed",* New York: John Wiley and Sons Inc, 2003, p. 329.

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)