Comparative Performance Analysis of Genetic Algorithm Variants on Solving 0/1 Knapsack Problem

Ilamparithi Yaazharasu CR*[1],
Department of Computer Science and Engineering,
Amrita School of Computing, Coimbatore,
Amrita Vishwa Vidyapeetham, India.
[1]cb.en.u4cse19325@cb.students.amrita.edu,

Ashwin R[2],
Department of Computer Science and Engineering,
Amrita School of Computing, Coimbatore,
Amrita Vishwa Vidyapeetham, India.
2cb.en.u4cse19345@cb.students.amrita.edu,

Pavan Teja Ramana[3],
Department of Computer Science and Engineering,
Amrita School of Computing, Coimbatore,
Amrita Vishwa Vidyapeetham, India.
[3]cb.en.u4cse19343@cb.students.amrita.edu,

Paladugu Shilpa[4],
Department of Computer Science and Engineering,
Amrita School of Computing, Coimbatore,
Amrita Vishwa Vidyapeetham, India.
4cb.en.u4cse19342@cb.students.amrita.edu,

Jeyakumar G[5]
Department of Computer Science and Engineering,
Amrita School of Computing, Coimbatore,
Amrita Vishwa Vidyapeetham, India.
[5]g_jeyakumar@cb.amrita.edu

***Abstract*—Practical optimization issues are primarily and commonly addressed by *EAs* (Evolutionary Algorithms). A well-liked subset of *EA*, the Genetic Algorithm (*GA*), is renowned for its capacity to solve various optimization problems. This research work examines the possibility of the Genetic Algorithm (*GA*) to resolve the 0/1 Knapsack problem. In this study, a number of mutation and crossover types are used to solve 0/1 Knapsack problem. The effectiveness of *GA* is compared by utilizing various crossover and mutation combinations. Positively, *GA* may solve the problems substantially faster and with a reasonable degree of accuracy as compared to traditional methods. In a careful assessment of the stated behavior of *GA*, combination on three mutations and three crossover methods are explored in this study.**

Keywords—Genetic algorithm; Mutations; Crossovers; 0/1 Knapsack; Optimal solution

I. Introduction

Evolutionary Computing (*EC*) is a subfield of Artificial Intelligence (*AI*)'s "soft computing" concepts. *EC* contains a collection of methods called Evolutionary Algorithms (*EAs*) to address the problems with optimization. Technically speaking, *EAs* are stochastic, iterative, population-based metaheuristic problem solvers. *EAs* employ the biological processes of recombination, mutation, and selection. The fitness functions created for the specific optimization problem are used to generate the individual's fitness score. The best candidates are selected during the evolutionary process in order to reach the best outcome.

The Genetic Algorithm (*GA*), a component of *EA*, is used to solve optimization and search problems. The *GA* parameters include the initial solutions, the maximum number of generations, the kind of mutation, and the type of crossover. These parameters' values are either initialized or supplied by the user. In order to discover the best outcome for the population, *GA* creates an initial random population and employs a fitness function. The *GA* process is carried out up until an acceptable solution is found or the permitted restrictions are satisfied.

An example of a combinatorial optimization problem is the Knapsack Problem. The goal is to choose some objects from a list of items with weight and value to fit into a knapsack with a defined capacity so that the overall weight of the items is less than or equal to the capacity but the value is as high as possible. When there are financial limits on resource allocation, problems frequently develop. The knapsack problem comes under the NP-complete problem.

This paper analyses various combinations of different crossover and mutation functions to find the best-suited combination for solving knapsack problems using *GA*.

The remaining part of the paper has multiple sections, as mentioned below. The Section II explains the studies reported related to the work presented in this paper, Section III describes the design of experiments conducted as part of this study, and Section IV discusses the performance of different *GA* variants and finally the Section V concludes the paper. The experiment was implemented in IntelliJ IDE using java as the base language.

The objective of this paper is to analyze the behavior of *GA* with different selection and crossover functions.

II. Related Works

This section explores the applications of *GA* in solving toy problems viz 0/1 knapsack problem, n-queen problem, sudoku, etc.

The author in [1] has focused on the selection function for the application of the 0/1 Knapsack problem. He analyzed four different selection algorithms and how they affect the genetic algorithm. It is concluded that the 0/1 knapsack problem which takes exponential time to solve can be solved in linear time using genetic algorithms. In [2], the 0/1 knapsack problem is solved using *GA* with different types of crossovers, mutation, encoding, and decoding, and using Dynamic Programming (*DP*). In the two methods, the results are compared based on the running time, the iteration count, and the accuracy. It is found in the comparison that the *GA* shows improvements in its results over the iterations.

The author in [3] solves the n-queens problem using two ways where *GA* can be used, which is by using simple heuristics and using existing evaluation heuristics by mapping onto the n-queens problem. Heuristics are developed in order to solve them using *GA*. The author used the satisfiability heuristics developed by Dejong and Spears to get a working evaluation function. Reference [4] presents a study of using *GA* for solving sudoku puzzles. It also discussed whether *GA* can be used to create new puzzles as well as how it can summarize about the difficulty rate. *GA* with integer encoding and elitist is used with chromosome length of 81 integers, broken down into nine subblocks of nine numbers. The swap mutations are used inside the sub-blocks, and the uniform crossover operation is only used between adjacent sub-blocks.

In [5], the authors discuss implementing *GA* for solving the 0/1 knapsack problem. Various methods with which each part of the *GA* can be implemented are mentioned. Three types of selection operations and three types of crossover operations are used. Various crossover functions viz., one-point crossover, two-point crossover, order crossover, partially mapped crossover, and cycle crossover are analyzed in [6]. The analysis is done on different datasets. A novel crossover function (named Sinusoidal Motion Crossover), in which the new gene that is being generated has alternating chromosomes from parent 1 and parent 2, is introduced.

It's interesting to note that *GA* is tested in [7] for a toy maze problem. GAmaze is the name of the suggested framework. The interesting aspect of GAmaze is how it continuously updates the population. The GAmaze uses fitness data from potential solutions to help people learn more effectively and get to their destination by teaching them how to navigate challenges and find a way. The applicability of GA in solving the famous subset problem is tried out in [8], and its performance on this problem is compared with the dynamic programming approach. Similar to this, the performance of GA on Sudoku problem solving with its different variants is experimented and the reports are presented in [9].

GA is also used to solve several sensitive real word optimization problems around us. A few such works are presented next. Reference [10] focuses on one of the main issues facing the online community: computer malware or, more particularly, cyber security. The two fields of biologically inspired computing and computer malware are combined in this article. This paper explores the potential, difficulties, and opportunities associated with merging evolutionary computing methods with malware generation under the guidance of proactive defense. In [11], the author implies a *GA*-based method for dividing up slices of brain *MRI* (Magnetic Resonance Imaging) images. In this study, the various *GA* parameters have been examined, and a slice-type-based optimum threshold value has been established. [12] presents a thorough analysis of the use of *GA* for feature selection. In order to address issues with course allocation in educational institutions, the authors of [13] adopted GA. Thus, the wide applicability of *GA* (on its own and by hybridizing with other algorithms [14, 15]) is evident in the literature on solving a variety of global optimization algorithms. *GA* is also used for optimizing the design parameters of neural networks. In [16], the authors have used GA to train the spiking neural networks.

In [17] the authors have explained different methods of implementing different steps of a *GA*, while [18] discusses different types of GA that have been found and used in recent years and all the operators that have been used till now in solving different steps of a *GA*. [19] works on how *GA* can be implemented to solve a real-life problem which in this case was a different variant of the Job Scheduling Problem.

Reference [20] discusses an approach to solve the 0/1 knapsack problem using an evolutionary multi objective genetic algorithm where the simple point mutation and simple point crossover are used.

On understanding the nature and benefits of *GA*, this paper presents a comparative performance analysis of *GA* in solving the 0/1 knapsack problems with variety of mixing of its mutation and crossover.

III. Design of Experiments

The experiments conducted as part of this study is intended to analyze the variations in the performance of *GA* in solving the 0/1 knapsack problem with different combination of mutation and crossover operations. The logic followed during the implementation of *GA* for its various algorithmic components is described below.

- The genes of the initial population are generated randomly with a 50% chance of each chromosome being either 0 or 1.
- The initial population is with the size of either 100 (for n = 10) or 200 (for n = 200).
- The selection function is choosing the best two genes present in the current generation to produce the next generation.
- Crossover rate set to 0.8 and mutation rate set to 0.05.

- Fitness value for each gene represents the sum of values of items included in the knapsack denoted by that gene.
- Value efficiency used in analyses means that the best fitness value obtained is divided by the solution found using the standard DP approach.
- Runtime for each iteration is calculated and represented in nanoseconds.

A brief summary of different mutation and crossover operations used in the experiment is presented next.

A. Crossover Operations

(1) *One Point crossover:* A random point in selected as crossover point, then the contents of the parent strings are swapped beyond that point.

Parent 1: 1 0 1 1 1 0 0 1; Parent 2: 0 0 1 0 1 1 0 0

Considering a crossover point at 3, swap all the genes after the crossover point between the parents.

Offspring 1: 1 0 1 0 1 1 0 0; Offspring 2: 0 0 1 1 1 0 0 1

(2) *Two Point crossover:* Two points are selected at random. All data in between the selected two points are swapped between the two parent genes.

Parent 1: 1 0 1 1 1 0 0 1; Parent 2: 0 0 1 0 1 1 0 0

Considering a crossover between index 3 and 6, swap all the genes after the crossover point between the parents.

Offspring 1: 1 0 1 0 1 1 0 1; Offspring 2: 0 0 1 1 1 0 0 0

(3) *Uniform crossover:* The number of swaps and positions of chromosomes to be swapped are generated randomly.

Parent 1: 0 0 0 0 0 0 0 0; Parent 2: 1 1 1 1 1 1 1 1

Offspring 1: 0 0 1 0 1 1 0 0; Offspring 2: 1 1 0 1 0 0 1 1

B. Mutation Operations

(1) *Bitflip* mutation: Randomly selected bits are flipped.

Before mutation: 0 0 1 0 1 1 1 1
After mutation: 1 0 0 1 0 1 0 1

(2) *Inverse* mutation: A subset of genes are selected and inverted in the subset.

Before mutation: 1 0 1 1 0 0 0 1 0 1
After mutation: 1 0 0 0 1 1 0 1 0 1

(3) *Swap* mutation: Two positions on the chromosome are selected at random and their genes are interchanged.

Before mutation: 1 0 1 0 1 0 0
After mutation: 1 0 0 0 1 0 1

IV. Results and Discussion

Considering 3 crossover operations (One point, two-point and uniform) and 3 mutation operations (bitflip, inverse, swap), 9 different variants of *GA* are formulated and their performance in solving the 0/1 knapsack problem is analyzed. The nine *GA* variants and their mutation and crossover operations are presented in Table I.

TABLE I. The GA Variants

***GA* Variant**	**Crossover**	**Mutation**
GA_{ob}	One point	Bitflip
GA_{oi}	One point	Inverse
GA_{os}	One point	Swap
GA_{tb}	Two point	Bitflip
GA_{ti}	Two point	Inverse
GA_{ts}	Two point	Swap
GA_{ub}	Uniform	Bitflip
GA_{ui}	Uniform	Inverse
GA_{us}	Uniform	Swap

The empirical results obtained for GA_{ob} (One point crossover and bit flip mutation) are shown in Table II. The GA_{ob} could produce the result within 10 generations with high efficiency when *N* is comparatively less. As the value of *N* increases, GA_{ob} takes more generations to find the solution. Here, *N* is the total number of available items that can be chosen to fill the knapsack i.e., the gene size. The *value* efficiency is the maximum optimal value obtained by the *GA* that is used, divided by the actual optimal value obtained using the *DP* method. *Runtime* is the total time taken by a *GA* to reach the maximum optimal solution it could achieve.

The empirical results obtained for GA_{oi} (one point crossover and inverse mutation) are shown in Table III. The GA_{oi} could produce results with 100% value efficiency for almost every run, within 10 generations regardless of the value of *N*.

The empirical results obtained for GA_{os} (One point crossover and swap mutation) are shown in Table IV. The GA_{os} could produce results with 100% value efficiency within 10 generations when *N*=10 but value efficiency reduces by a small amount when *N*=20.

TABLE II. Performance of GA_{OB}

Run	**No of items**	**Value Efficiency**	**Generation**	**Run time (ns)**
1	10	100	6	24878800
2	10	62.44	1	212216500
3	10	100	6	24198300
4	10	100	2	7749000
5	10	100	8	23981800
6	20	100	19	29282900
7	20	99.53	2000	678551700
8	20	100	38	45498500
9	20	100	20	31097400
10	20	94.11	24	1227537300

TABLE III. PERFORMANCE OF GA_{OI}

Run	No of items	Value Efficiency	Generation	Run time (ns)
1	10	100	2	1893200
2	10	100	1	2450900
3	10	100	4	26495500
4	10	100	8	23423100
5	10	100	6	9039200
6	20	100	21	20275600
7	20	100	28	11526400
8	20	100	17	14649300
9	20	100	76	45655300
10	20	99.39	202	863513600

TABLE IV. PERFORMANCE OF GA_{OS}

Run	No of items	Value Efficiency	Generation	Run time (ns)
1	10	100	2	6029600
2	10	100	2	1926900
3	10	100	10	20741200
4	10	100	6	22839800
5	10	100	2	2967100
6	20	99.03	429	198329300
7	20	100	35	7987900
8	20	100	17	11632700
9	20	98.71	2000	617254900
10	20	93.90	30	798178400

The empirical results obtained for GA_{tb} (Two-point crossover and Bit flip mutation) are shown in Table V. The GA_{tb} could produce results with almost 100% value efficiency within 10 generations when N=10, but value efficiency reduces as N is increased to 20.

TABLE V. PERFORMANCE OF GA_{TB}

Run	No of items	Value Efficiency	Generation	Run time (ns)
1	10	100	11	11394600
2	10	100	3	3713900
3	10	100	8	18084600
4	10	100	3	6487300
5	10	97.25	10	603970100
6	20	99.03	2000	325541600
7	20	100	49	7890000
8	20	99.15	2000	380399800
9	20	98.71	2000	268859700
10	20	100	10	5954300

The empirical results obtained for GA_{ti} (Two-point crossover and Inverse mutation) are shown in Table VI. The GA_{ti} was unable to produce results with near 100% value efficiency within 10 generations regardless of the value of N.

TABLE VI. PERFORMANCE OF GA_{TI}

Run	No of items	Value Efficiency	Generation	Run time (ns)
1	10	100	7	18488100
2	10	70.05	2	384978800
3	10	100	10	4843100
4	10	100	5	11300700
5	10	98.28	5	283972000
6	20	98.06	2000	231981300
7	20	100	99	14772000
8	20	95.72	2000	223123500
9	20	100	127	26166300
10	20	98.17	64	168314700

The empirical results obtained for GA_{ts} (Two-point crossover and Swap mutation) are shown in Table VII. The GA_{ts} could produce results with 100% value efficiency within 10 generations when N=10 but value efficiency reduces by a small amount when N=20.

TABLE VII. PERFORMANCE OF GA_{TS}

Run	No of items	Value Efficiency	Generation	Run time (ns)
1	10	100	3	8075800
2	10	100	43	5856600
3	10	100	14	4770400
4	10	100	3	3731200
5	10	100	7	2931500
6	20	100	6	1263600
7	20	99.53	2000	272221900
8	20	100	46	5581200
9	20	100	39	8300900
10	20	92.89	162	400176900

The empirical results obtained for GA_{ub} (Uniform crossover and Bit Flip mutation) are shown in Table VIII. The GA_{ub} could produce results with 100% value efficiency within 10 generations when N=10 but there is a significant reduce in the value efficiency when N=20.

The results of GA_{ui} (Uniform crossover and Inverse mutation) are shown in Table IX. The GA_{ui} could produce results with 100% value efficiency within 10 generations when N=10 but value efficiency reduces by a small amount when N=20.

The results of GA_{us} (Uniform crossover and Swap mutation) are shown in Table X. The GA_{us} could produce results with 100% value efficiency within 10 generations for almost all the runs when N=10. Value efficiency is almost 100% for all the runs when N=20.

Analysis of an instance with 20 items is shown in Table XI. The optimal solution found for this set using DP (Dynamic Programming) is 938. With this, the algorithm is run against all 9 combinations of functions. Evaluated with overall efficiency as the sum of value efficiency and run-time efficiency. Value efficiency is defined as the fitness obtained for that run divided by the optimal solution found using DP (in this case, 938) times 100, while run-time efficiency is defined as the individual GA's run-time obtained compared to the shorted runtime among all the nine combinations of functions. The lesser the run time efficiency the longer it took to run those GA variants.

TABLE VIII. PERFORMANCE OF GA_{UB}

Run	No of items	Value Efficiency	Generation	Run time (ns)
1	10	100	9	8333100
2	10	100	1	221800

3	10	100	11	17584400
4	10	100	22	11055100
5	10	100	16	13216300
6	20	100	23	3798600
7	20	97.16	2000	262905400
8	20	97.65	2000	268394300
9	20	100	47	11185700
10	20	83.74	21	376165800

TABLE IX. PERFORMANCE OF GA_{UI}

Run	No of items	Value Efficiency	Generation	Run time (ns)
1	10	100	12	4177000
2	10	100	2	520300
3	10	100	9	6981200
4	10	100	1	279300
5	10	100	14	7025000
6	20	100	198	26928500
7	20	99.29	586	72697300
8	20	95.50	2000	175345100
9	20	100	148	24340000
10	20	100	37	12206700

TABLE X. PERFORMANCE OF GA_{US}

Run	No of items	Value Efficiency	Generation	Run time (ns)
1	10	100	10	3197200
2	10	76.14	1	9419900
3	10	100	4	1606300
4	10	100	7	2697800
5	10	100	6	2125300
6	20	99.03	2000	266313700
7	20	99.53	2000	196692200
8	20	100	12	1440400
9	20	100	16	2664000
10	20	100	32	10235000

Table XII shows the *average value efficiency* and *runtime* of all the 9 variants. *Average value efficiency* and *runtime* are the summations of respective values divided by the total number of runs the algorithm was tested with.

Figure 1 shows the plot between the 9 variants with the x-axis as *average value efficiency* and y-axis as *average runtime.* Each point is marked with the respective variant. *One-point crossover* with *bit flip mutation* had the lowest *average value efficiency* and highest *average runtime*, while *one-point crossover* with *inverse mutation* had the highest *value efficiency*, and *uniform crossover* with *inverse mutation* had the least average runtime of all.

TABLE XI. COMPARATIVE ANALYSIS OF *GA* VARIANTS WITH 20 ITEMS

GA	Fitness Obtained	Value Efficiency (%)	Generation	Run time (ns)	Run time efficiency (%)	Total Efficiency
GA_{ob}	853	90.94	20	12841200	89.99	90.46
GA_{oi}	833	88.81	20	5715100	96.10	92.45
GA_{os}	847	90.29	20	3563600	97.95	94.12
GA_{tb}	855	91.15	20	1973500	99.31	95.23
GA_{ti}	816	86.99	20	2032300	99.26	93.13
GA_{ts}	794	84.65	20	2068800	99.23	91.94
GA_{ub}	893	95.20	20	1317700	99.87	97.54
GA_{ui}	844	89.98	20	1447900	99.76	94.87
GA_{us}	872	92.96	20	1166600	100	96.48

TABLE XII. SUMMARY OF THE RESULTS OBTAINED

Crossover	Mutation	Average Value Efficiency	Average runtime (ns)
One Point Crossover	Bit Flip Mutation	95.61	230499220
	Inverse Mutation	99.94	101892210
	Swap Mutation	99.16	168788780
Two Point Crossover	Bit Flip Mutation	99.41	163229590
	Inverse Mutation	96.03	136794050
	Swap Mutation	99.24	71291000
Uniform Crossover	Bit Flip Mutation	97.85	97286050
	Inverse Mutation	99.48	33050040
	Swap Mutation	97.47	49639180

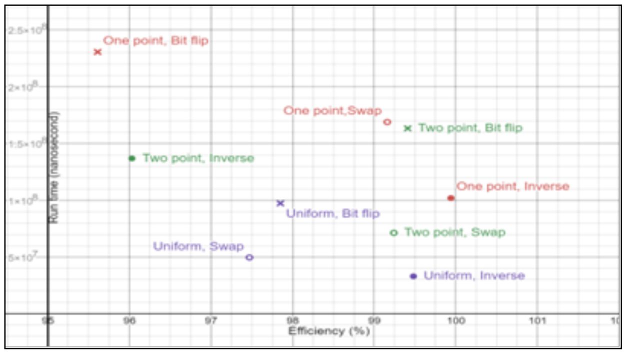

Fig. 1. The combinations with *value efficiency* (x-axis) against *run time* (y-axis)

V. CONCLUSIONS

In order to solve 0/1 Knapsack problem, this research compares the performance of 9 different Genetic Algorithm (*GA*) variants. The manner in which the mutation and crossover process are carried out distinguishes the *GA* variants form one another. Every mutation and crossover resulted in a different performance from *GA*. According to the comparison, uniform crossover with inverse mutation takes less time to execute and achieves an average efficiency of 99.48. Nonetheless, at the average value efficiency of 99.94, the one-point crossover and inverse mutation combination has delivered the best performance.

This study clearly depicts the behavioral changes happening in *GA* on solving the 0/1 knapsack problem for the changes in its crossover and mutation operations.

This study can be extended further adding more toy problems as well as additional performance metrics, to get a generic comparison of crossover and mutation combinations for *GA*. For the current problem in hand i.e., the 0/1 knapsack problem combination of uniform crossover with inverse mutation is the apt method to implement in the GA to get optimal results.

REFERENCES

[1] R. P. Singh, "Solving 0–1 Knapsack problem using Genetic Algorithms," 3rd IEEE international conference on commnuication software and networks, pp. 591-595., 10.1109/ICCSN.2011.6013975, 2011.

[2] Yan Wang, Min Wang, Jia Li, and Xiang Xu, "Comparison of genetic algorithm and dynamic programming solving knapsack problem,", *In 2020 3rd International Conference on Algorithms, Computing and Artificial Intelligence (ACAI 2020).* Association for Computing Machinery, New York, NY, USA, Article 10, pp. 1–5, 2020.

[3] Kelly D. Crawford, "Solving the n-queens problem using genetic algorithms,", *In Proceedings of the 1992 ACM/SIGAPP symposium on Applied computing: technological challenges of the 1990's (SAC '92).,* Association for Computing Machinery, New York, NY, USA, 1039–1047, 1992.

[4] Mantere, Timo and Koljonen, Janne, "Solving, rating and generating Sudoku puzzles with GA,", 2007 IEEE Congress on Evolutionary Computation, CEC 2007, pp. 1382 - 1389. 10.1109/CEC.2007.4424632, 2007.

[5] Veenu Yadav and Ms.Shikha Singh, "A Review Paper on Solving 0-1 knapsack Problem with Genetic Algorithms," International Journal of Computer Science and Information Technologies, Vol. 7., No. 2, pp. 830-832, 2016.

[6] S G. Varun Kumar and R. Panneerselvam, "A Study of Crossover Operators for Genetic Algorithms to Solve VRP and its Variants and New Sinusoidal Motion Crossover Operatorm," *International Journal of Computational Intelligence Research,* Vol. 13., No. 7., pp. 1717-1733, 2017.

[7] K. Krishnaa, K. Jonathan, K. Saketh, A. Tadi and G. Jeyakumar, "A Genetic Algorithm Framework to Solve Two-Dimensional Maze Problem," 10.1007/978-981-19-2828-4_27, 2022.

[8] Konjeti Harsha Saketh and G. Jeyakumar, "Comparison of Dynamic Programming and Genetic Algorithm Approaches for Solving Subset Sum Problems", Advances in Intelligent Systems and Computing, Vol. 1108., pp. 472-479., Springer, Cham, 2019.

[9] D. Srivatsa, T.P.V.K, Teja, I. Prathyusha and G. Jeyakumar, "An Empirical Analysis of Genetic Algorithm with Different Mutation and Crossover Operators for Solving Sudoku," Lecture Notes in Computer Science, Vol .11941., Springer, Cham. 2019.

[10] M. Ritwik and C.S. Velayutham, "A Conceptual Direction on Automatically Evolving Computer Malware using Genetic and Evolutionary Algorithms", *2020 International Conference on Inventive Computation Technologies (ICICT)*, pp. 226-229., 2020.

[11] K. Vikram, Hema Menon, D. Dhanya, "Segmentation of Brain Parts from MRI Image Slices Using Genetic Algorithm," Lecture Notes in Computational Vision and Biomechanics, Vol. 28., 2018.

[12] C. Pragadeesh, Rohana Jeyaraj, K. Siranjeevi, R. Abishek and G. Jeyakumar, "Hybrid feature selection using micro genetic algorithm on microarray gene expression data", *Journal of Intelligent and fuzzy* systems, Vol. 36., No. 3., pp. 2241-2246, 2019.

[13] S. Abhishek, S.C. Emmanuel, G. Rajeshwar and G. Jeyakumar, G, "A Genetic Algorithm Based System with Different Crossover Operators for Solving the Course Allocation Problem of Universities," New Trends in Computational Vision and Bio-inspired Computing, ICCVBIC 2018. Springer, Cham, pp. 149-160., 2020.

[14] S S. Shinde, S. Thangavelu and G. Jeyakumar, "Mixed Differential Evolution and Genetic Algorithm Hybridization for Solving Global Optimization Problems," Advances in Intelligent Systems and Computing, Vol. 1392., 2021.

[15] A. Harishchander, S. Senapati and DA. Anand, "Analysis of drug resistance to HIV-1 protease using fitness function in genetic algorithm," *BMC Infectious Diseases*, Vol. 12., No. 7., 2012.

[16] V. Sangeetha and J. Preethi. "Cluster analysis of breast cancer data using Genetic Algorithm and Spiking Neural Networks," IEEE 9th International Conference on Intelligent Systems and Control (ISCO), pp. 1-5., 2015.

[17] Lingaraj, Haldurai, "A Study on Genetic Algorithm and its Applications,". International Journal of Computer Sciences and Engineering, Vol. 4., pp. 139-143., 2016.

[18] S. Katoch, S. S. Chauhan and V. Kumar, "A review on genetic algorithm: past, present, and future," Multimed Tools Appl, Vol. 80., pp. 8091–8126., 2021.

[19] Muhammad Kamal Amjad, Shahid Ikramullah Butt, Rubeena Kousar, Riaz Ahmad, Mujtaba Hassan Agha, Zhang Faping, Naveed Anjum, and Umer Asgher, "Recent Research Trends in Genetic Algorithm Based Flexible Job Shop Scheduling Problems," Mathematical Problems in Engineering, Vol. 2018., Article ID 9270802, 32 pages, 2018.

[20] S. N. Mohanty and R. Satapathy, "An evolutionary multiobjective genetic algorithm to solve 0/1 Knapsack Problem," In the procedings of 2nd IEEE International Conference on Computer Science and Information Technology, Beijing, China, pp. 397-399., 2009.