GENETIC ALGORITHMS APPLIED TO REAL TIME MULTIOBJECTIVE OPTIMIZATION PROBLEMS

Z. BINGUL, A. S. SEKMEN, S. PALANIAPPAN AND S. SABATTO
Tennessee State University, Nashville, TN, 37209
e-mail: sabatto@harpo.tnstate.edu

ABSTRACT

Genetic algorithms are often well suited for multiobjective optimization problems. In this work, multiple objectives pertaining to the THUNDER software were concerned to optimize the war results obtained from the software. It is a stochastic, two-sided, analytical simulation of military operations. The simulation is subject to internal unknown noises. Due to these noises and discreetness in the simulation program, GA approach has been applied to this multiobjective optimization problem. This method is capable of searching for multiple solutions concurrently in a single run. Transforming this multiobjective optimization problem to a form suitable for direct implementation of GA was the major challenge that was achieved. Three different kinds of fitness assignment methods were implemented and the best one was chosen. The THUNDER software may be considered as a black box since very less information about its internal dynamics was known. The problem with THUNDER software is expensive running time. In order to optimize the time involved with THUNDER software, autocorrelation techniques were used to reduce the number of THUNDER runs. Furthermore, the GA parameters were set optimally to yield smoother and faster fitness convergence. From these results, GA was shown to perform well for this multi-objective optimization problem and was effectively able to allocate force power for the THUNDER software.

Keywords: Intelligent Systems, Soft Computing, Genetic Algorithms, and Optimization.

INTRODUCTION

THUNDER software is a very large campaign simulation model, which was built based on Monte-Carlo simulation. This software is a stochastic, two-sided, analytical simulation of military operations developed by System Simulation Solutions Inc. (S3I) for the Air Force Studies and Analyses Agency (AFSAA). This simulation was designed in order to examine issues involving the utility and effectiveness of air and ground power in a theater-level joint warfare context. The Thunder software can define approximately 25 air missions grouped under air-to-ground missions, air-to-air missions, air defense suppression missions, reconnaissance, anti–tactical ballistic missile, and air refueling. This software automatically plans military moves and actions in a rule-based manner. It also judges the outcome of their interactions and then it dynamically incorporates the results and uses this information into the on-going perception, planning, and execution of military operations. This software is capable of supporting campaign analysis involving the integration of effects over time and space. This means automatic distribution of threats and targets, the number of target kills, ground movement and deployments. This simulation also applies constraints to the problems like defining the inventories, sortie rates, mission allocations etc.

In most of the real world problems associated with several objectives, it is needed to optimize them simultaneously based on some given criteria. However, there is no single optimal solution in many of these cases, but a set of alternative solutions exists. These solutions are called pareto-optimal solutions. Selecting one particular pareto optimal solution requires using one optimization method with some additional rules. To find solutions to this kind of optimization problem, one of potential ways is to use genetic algorithms because they process a set of solutions in parallel, possibly exploiting similarities of solutions by recombination.

A general multi-objective optimization problem with inequality constraints can be stated as a vector function f that maps a set of *m* parameters (decision variables) to a set of *n* objectives. This can be formulated as

$$Y = f(x) = (f_1(x), f_2(x), f_3(x), \ldots\ldots f_4(x)) \qquad (1)$$

$$\text{subject to} \quad x = (x_1, x_2 \ldots\ldots x_m) \in X$$
$$y = (y_1, y_2 \ldots\ldots y_n) \in Y$$

where *x is* set of the decision vectors and *X* is the parameter space, *y* is the objective vector, and *Y* is the objective space. The set of solutions of a multiobjective optimization problem consists of all decision parameters for which corresponding objective vectors cannot be improved in any dimension without degradation in another. Such solutions are known as pareto-optimal solutions. A Pareto optimal solution is not unique, but is a member of a set of such points, which are considered equally good in terms of the vector objective. This space may be viewed as a space of compromise solutions in which each objective could be

improved, but at the expense of at least one other objective. In other words these solutions are optimal in a way such that no other solution in the search space are superior to them when all objectives are concerned.

Genetic algorithms

Many practical optimization problems have mixed (continuous and discrete) variables and discontinuities in their search space. If standard non-linear programming techniques were to be used in such cases, then they would be computationally very expensive and inefficient. Genetic algorithms are a good solution to such situations. They were first introduced by Holland [12] who describes how to apply the principles of natural evolution to optimization problems. Holland's theory has been further developed and now Genetic Algorithms (GAs) stand up as a powerful adaptive method to solve search and optimization problems.

Genetic algorithms are search algorithms that come under the range of techniques, collectively known as "evolutionary computing ". They are based on the principles of natural genetics and natural selection. The major benefits of these algorithms is that they provide a robust search in complex spaces and are usually less expensive, as far as computation is concerned, when compared to most other optimization solutions. They are also resistant to getting trapped in local optima. This leads to a wide range of applications in large-scale optimization problems of various fields.

Genetic algorithms are different from more normal optimization and search procedures in four ways:

1. GAs search with a population of points (candidate solutions), not a single point. Thus, they are less likely to be trapped in a local optimum
2. GAs use only the values of the payoff (objective function) information, and not the derivatives or other auxiliary knowledge.
3. GAs work with a coding (representation) of a parameter set not the parameters themselves. Thus the search method is naturally applicable for solving discrete and integer programming problems.
4. GAs use randomized parents selection and crossover from the old generation. Thus they efficiently explore the new combinations with the available knowledge to find a new generation with better fitness values.

Blackbox optimization and genetic algorithms

Black box optimization may be considered as the presence of little or no knowledge about the existing domain and such situations are becoming more and more popular in the recent years. There are several algorithms to optimize a black box and they may be classified as follows.

- Deterministic approaches
- Stochastic approaches
  - Blind random search methods
  - Adaptive sampling search methods.

Deterministic enumeration methods become highly impossible with the growth of the search spaces and amongst the stochastic approaches, the adaptive sampling search methods work well with black box optimization techniques. Genetic algorithms are one of the most popular and effective tools amongst the adaptive sampling search methods. The mechanics of a simple genetic algorithm are very simple, involving nothing more complex than copying strings and swapping partial strings. GAs are very popular due to their simplicity of operation and computational efficiency. GAs use the representation to implicitly divide the search space into several non-overlapping classes often called as the schemata.

Let us denote the input and output spaces as follows *X* and *Y* respectively. Then the general blackbox optimization may be defined as follows given a blackbox that computes $\phi(x)$ for input x.

$$\phi : X \rightarrow Y$$

The objective of a maximization problem is to find some $x^* \in X$, such that $\phi(x^*) \geq \phi(x)$ for all $x \in X$.

Most of the simple genetic algorithms which yields good results to many practical problems and are composed of three main operators and they are:

1. Selection
2. Crossover
3. Mutation

Selection

Selection is one of the most fundamental genetic operators. Selection operation may be modeled as follows:

$$P_{select}(n) = f(n) / \sum_{k=1}^{pop} f(k) \qquad (2)$$

Where n is n^{th} string, *pop* is the population size and *f (n)* is the fitness function. This first population must offer a wide diversity of genetic materials. The gene pool should be as large as possible so that any solution of the search space can be engendered. Generally, the initial population is generated randomly. Some of the most commonly used selection operators are roulette wheel selection, tournament selection, Ranking selection etc.

Crossover

This is the most powerful genetic operator, and may be considered as the main engine for exploration in a GA. This operator is responsible for the shuffling and recombination of building blocks.

The simplest form of crossover is that, a single point is chosen on two equal length chromosomes and they are crossed at that particular point. It is possible to select two or more points for cross over, to get more genetic mixing but sometimes while using multipoint crossover it degrades the performance. Crossover can be shown as follows.

```
1 1 1 1   1 1          1 1 1 1 0 0
  X                 →
0 0 0 0   0 0          0 0 0 0 1 1
```

Crossover generally consists of forming a new solution by taking some parameters from one solution and exchanging it with another at the very same point. Thus we get new offspring. Some crossover operators use complex geometric methods to generate the off springs of two parents.

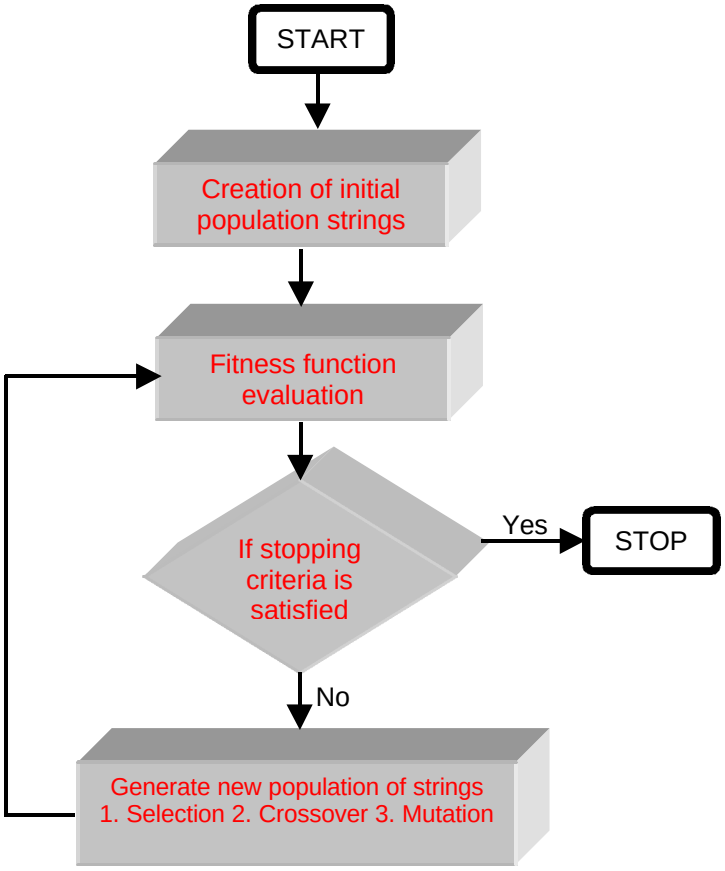
Mutation

This is a common genetic manipulation operator, and it involves, the random alteration of genes during the process of copying a chromosome from one generation to the next. Mutation simply involves the incorrect copying of some parameters, which make up a solution. It may be illustrated as follows.

```
1 1 1 1 1 1  →  1 1 0 1 1 1
```

Mutation is usually used to avoid premature convergence, which is a common problem in GAs, which use fixed length, binary codlings. When proportional selection is used, all the individual chromosomes in the population become very similar before a nearly optimal solution is reached, thus preventing any further progress. In such cases mutation is essential. Mutation acts against this, by constantly generating new chromosomes, this helps in preventing the population from getting trapped in a local maximum in a search space. However, mutation sometimes also result in loss of good individual, thus the need to prevent premature convergence has to be balanced against the loss of efficiency due to the damage of good genetic material. Thus there is a payoff between exploitation and exploration illustrated here.

Genetic algorithm process (flow chart)



Representation

This is an important aspect in determining the success or failure of a GA. Representation is a way by which the chromosome is generated, so as to reflect a potential problem solution. A representation should have as little epistasis as possible and should represent the solutions in such a way so that the genetic operators could be used on them. There is a strong link between the representation scheme and the genetic operators to be used. The representation may also be highly dependent on the problem to be solved. The most suitable representation is based upon the type of application .The two most common representations, binary and real number coding differ in how the recombination and mutation operators are performed.

The main purpose of this research is to determine how to effectively allocate force power using genetic algorithms (GA). The reason why genetic algorithms is used to determine the force allocations for war simulation is because they provide robust search procedures for many types of functions including those exhibiting discontinuities,

multi-modality, high dimensionality, huge search spaces and noise.

The war allocations are made based on the capabilities of the threat forces, conditions of the war, and capabilities of the friendly forces, all simulated by THUNDER software. The input war allocation file is generally given by the user. Currently, these war allocation are being made by an analyst judgement, and this is a very time consuming task, and does not always produces consistent results. Each mission requires an apportionment. In this research, only four missions were used as inputs and 15 day war was used. The missions were Offensive Counter Air (OCA), Strategic Target Interdiction (STI), Long Range Air Interdiction (INT), and Lethal Direct Air Defense Suppression (DSEAD). OCA, STI and INT are air-to-ground missions. OCA is against airbases and INT is against units moving on the network and in garrison, logistics facilities, transportation network transshipment points, checkpoints, supply convoys, and air defense complexes. STI is against strategic targets. DSEAD is a suppression of enemy air defense missions and it is against air defense sites. The THUNDER software can be viewed more like a two-player game in which blue represents the friendly side and red is the enemy side. According to these definitions, our objectives for these scenarios would be to

1. Minimize the territory that blue side losses
2. Minimize the blue side aircraft lost
3. Maximize the number of red side strategic targets killed
4. Maximize the number of red side armor killed

This is a typical multiobjective optimization problem because it is desired to optimize all the four objectives simultaneously. The procedure followed to solve the allocation problem using GA and THUNDER Software is illustrated in Figure 1.

RESULTS AND DISCUSSION

Thunder software is more like a black box. In order to understand its behavior and to optimize its the running time, optimum number of replication were to be determined. This reduces the time considerably for the entire process. In this regard, THUNDER runs were replicated several times and the results were obtained by assigning random seeds, this was stored in a separate output file. The maximum values of the parameters (FLOT penetration, Red STI kills, A-G armor + G-G armor and Blue aircraft losses) were taken from its output file for further analysis. Using this data, time series plots were drawn for all four parameters under consideration. These plots are shown in the figure 2. It can be noticed that there is a lot of noise, due to the random behavior of THUNDER. .Differences in the set of data for each parameter (FLOT penetration, Red STI kills, A-G

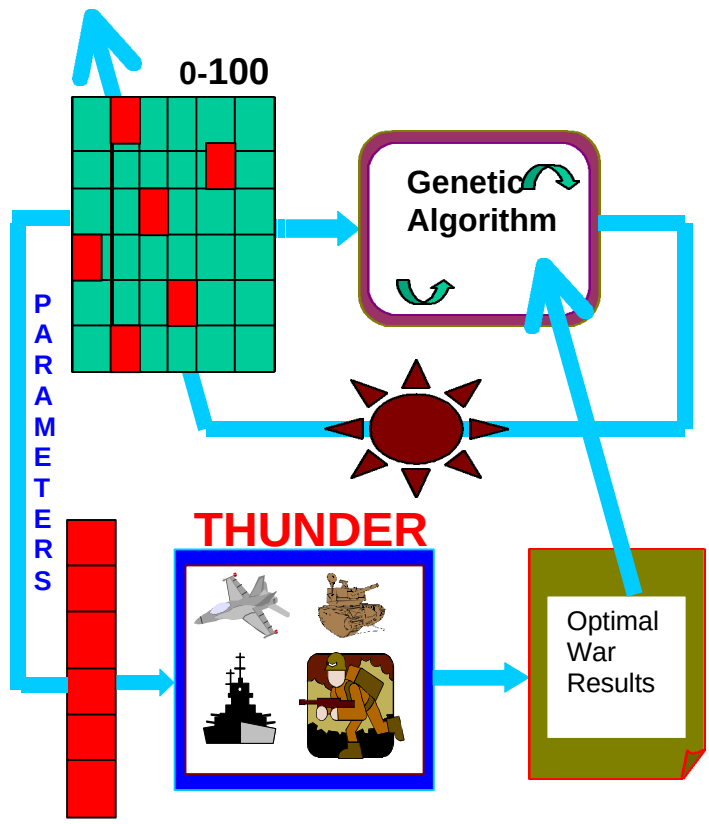


Figure 1. Schematic representation of optimization procedure.

Armor + G-G Armor and Blue aircraft losses) were calculated by subtracting from the previous value of the corresponding set of data. These differences were used to plot the trend analysis. The figure 3 show the amount of variation in the data from the trend. Based on the trend analysis, the data are fitted linearly and the linear regression equation is defined as:

$$y = a + bx \tag{3}$$

The coefficients of the linear regression are summarized in table 1.

Table 1: The coefficients of the linear regression

Thunder Output	a	b
Territory lost	-5.99	0.826
Red strategic targets killed	5.19	-0.49
Red armor killed	-5.19	0.77
Blue aircraft lost	3.56	-0.27

Autocorrelation analysis was made to decide which replication number would be more suitable to represent the entire system behavior. The figure 4 shows the autocorrelation between the set of data for each parameter. These show that lag 3 and lag 4 give higher correlation then

others. This means that the replication number 3 and 4 is much correlated with the general trend of the system. Using these replication numbers (3 and 4) would optimize the running time of THUNDER in an efficient way. The results of the statistical analysis (mean, median, standard deviation, the standard error of the mean, Se-mean, minimum and maximum) are summarized in table 2. From this table, it is found that, standard deviation is very less for FLOT penetration when compared to others, whereas Red armor is highest. It is also seen that the variation between the maximum and minimum values is less for FLOT penetration and high for red armor. These results indicate that red armor is highly dependent on the number of replications.

Table 2. Statistical analysis of the outputs of THUNDER software.

Variable	mean	median
Terrlost	107.29	107.25
Redsti	484.93	486.3
Red armor	1987.4	1989.3
Bl-A-loss	153.25	155.7
Variable	stdev	Se-mean
Terrlost	3.86	1.22
Redsti	6.71	2.12
Red armor	21.9	6.9
Bl-A-loss	8.22	2.6
Variable	max	min
Terrlost	116.9	102.8
Redsti	494	472
Red armor	2025.5	1963.3
Bl-A-loss	162.6	143

Fitness assignment

These results were obtained, by using modified data in the Thunder Software. In other words, new data set was applied to Thunder Software. Thunder Software (territory lost, aircraft lost, the number of strategic targets killed and the number of red armor killed) were assigned a minimum score and a maximum score. They were translated to a minimum score = 1 and a maximum score=2. Scores between the minimum and maximum were interpolated based on the worst case and the best case. These cases were determined by expert knowledge. In general, it is important to improve only the high priority objectives, such as hard constraints, until the corresponding design goals are met, after which improvements may be sought in the lower priority objectives. There are many ways to assign the fitness value. In our case, three different fitness functions were used to calculate the fitness values based on the above results. The first method is that GA tries to improve the worst score first, second worst next, and so forth. In other words, the high priority objective is that, the GA is forced to push up the lower scores. Based on this, fitness function is defined as follows:

$$F_1 = f_1^{\,4} + f_2^{\,3} + f_3^{\,2} + f_4 \qquad (4)$$

Where F_1 is a fitness score, f_1 is the smallest ordered individual score, f_2 is the second smallest ordered individual score, f_3 is the third smallest ordered individual score, and f_4 is the largest ordered individual score. Second method is square-based fitness assignment. In this method, individual scores are squared and their summation gives the assigned fitness values. This function can be written as:

$$F_2 = f_1^{\,2} + f_2^{\,2} + f_3^{\,2} + f_4^{\,2} \qquad (5)$$

Similarly, the third method is squared-error based fitness assignment. This function is:

$$F_3 = E_{\max} - \sum_{i=1}^{4} (f_{\max} - f_i)^2 \qquad (6)$$

where E_{max} (16 for this case) is the maximum value of total fitness score and f_{max} (2 for this case) is maximum value of each fitness score. In these experiments, the following GA parameters were adopted and held constant:

N =50 (population size)

P_c *=0.7 (crossover probability)*

P_m *=0.02 (mutation probability)*

The thunder runs were used online with the genetic algorithm, with all the three different fitness function. After50 generations were evaluated for a scenario, the maximum function values obtained in each case are shown below.

Table 3. Maximum fitness values using various fitness functions

Method	Max fitness values
F_1	14.033
F_2	11.893
F_3	15.502

GA produces three allocation-input sets for each fitness function correspondingly. These are:

Table 4. Optimum input allocations using various fitness functions

Method	OCA	INT	DSEAD	STI
F_1	20	27	53	0
F_2	0	20	40	40
F_3	14	0	80	6

Thunder Software was run, using these allocation inputs for 15 days war, the results are:

Table 5: Thunder output values based on the best possible allocation

Method	Territory lost	Red strategic targets killed	red armor killed	blue aircraft lost
F_1	87.1	482	1796	144
F_2	96.3	500	2061	136
F_3	108.9	491	2060	135

The fitness values have been plotted for all the three-fitness functions and are shown in figure 5. It is seen that after 20 generations, most of the learning is achieved in all the three cases. After 40 generations again the learning curve rises for fitness function F_1, whereas it increases for fitness function F_2 after 50 generations. But in the case of fitness function F_3 the learning is achieved in 10 generations. It is evident that the learning curve for the GA is not very smooth and has steps, which means that GA often stagnates at some points for few generations until it finds better solution, and when it finds the better solution, it moves over. In order to remove the steps in the learning curve or to optimize the step length so that we get smoother curve, we need to optimize the parameters in the genetic algorithm. This will improve the performance of the existing genetic algorithms.

The genetic algorithm was then used with random seed and the results are shown in the figure 5. The fitness function F_3 was not used with random seed because its fitness performance is low when compared to other two fitness functions, according to the following performance analysis. The maximum and minimum values of F_1, F_2 and F_3 functions are 12-16, 4-16, and 4-30, respectively. In order to compare performance of these fitness functions, their fitness values were scaled based on the ranges of the maximum and minimum. Based on this analysis, table 3 can be rewritten in table 6. Only the fitness functions F_1 and F_2 were used with random seed and we find that the fitness value has decreased for the same 50 generations because of the randomness in the system. However the genetic algorithm required more number of generations to achieve same results. Figure 5 shows the plot of various fitness functions with and without the random seed. In this plot, we clearly see the difference in the value of fitness for the same fitness function. Fitness_4 and fitness_5 are slightly less than fitness_1 and fitness_2, respectively. This shows that the GA has to put more efforts to achieve same results as it is trying hard to figure out the randomness in the system. Tables 7 and 8 show the best possible allocations and their corresponding THUNDER output values.

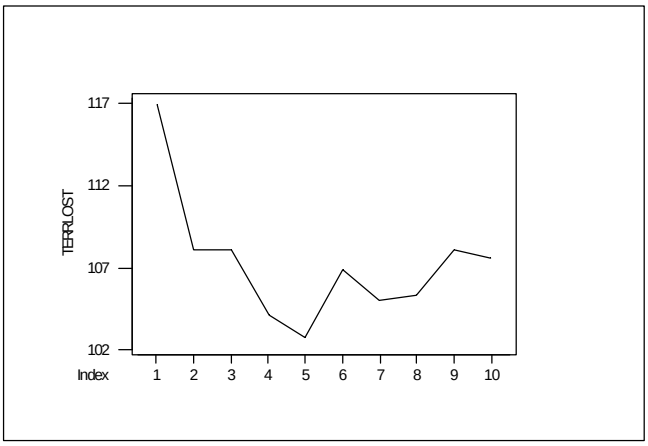


Figure 2. The blue side territory lost for changed random seeds.

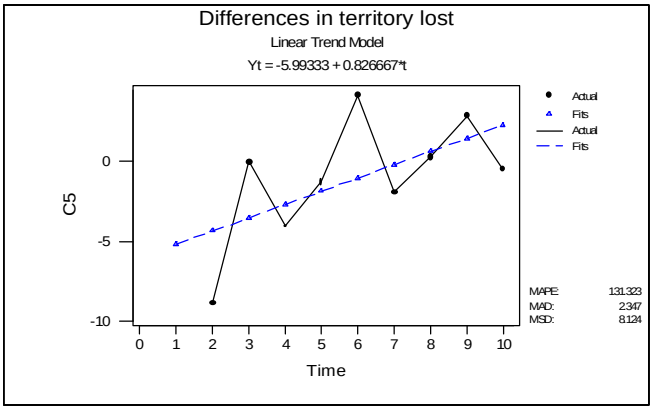


Figure 3. Trend analysis for differences in territory lost

Table 6:Performance analysis

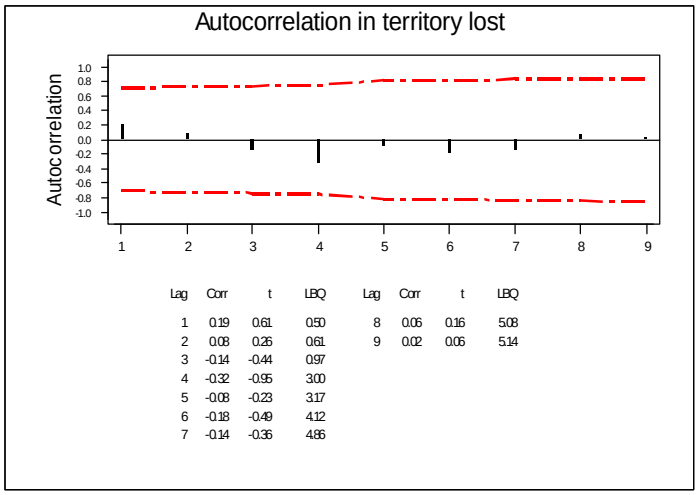
Method	Performance values
F_1	14.033*0.87*0.53=6.51
F_2	11.893*0.75=8.91
F_3	15.502*0.25=3.87

Table 7: Optimum input allocations using various fitness functions with random seed

Method	OCA	INT	DSEAD	STI
F_1	20	7	53	20
F_2	60	13	7	20

Table 8: Thunder output values based on the best possible allocation with random seed

Method	Territory lost	Red strategic targets killed	Red armor killed	Blue aircraft lost
F_1	137.1	452.3	1948	168.7
F_2	123.3	548.7	1885	152.7



Lag	Corr	t	LBQ	Lag	Corr	t	LBQ
1	0.19	0.61	0.50	8	0.06	0.16	5.08
2	0.08	0.26	0.61	9	0.02	0.06	5.14
3	-0.14	-0.44	0.97				
4	-0.32	-0.95	3.00				
5	-0.08	-0.23	3.17				
6	-0.18	-0.49	4.12				
7	-0.14	-0.36	4.86				

Figure 4. Autocorrelation in territory lost

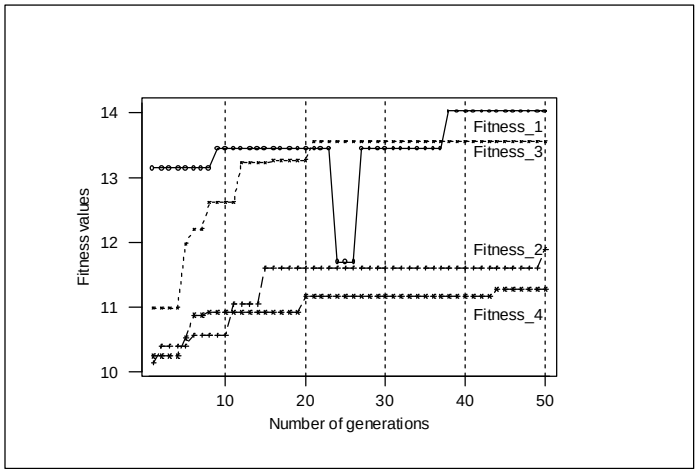


Figure 5. Comparison of fitness values with and without the random seed for the various fitness functions.

Setting of GA Parameters

Genetic algorithms work very efficiently, when good values for parameters such as mutation rate, crossover rate and population size, is chosen. There will be a wide range of parameters, which produce GA performance, not significantly different from the best possible performance. It is very beneficial, when these parameters are set optimally, because the GA will yield better or similar fitness values, in comparatively lesser number of generations. It is generally recognized that most GAs are fairly robust to the settings of crossover rates and mutation rates.

De Jong [7] has investigated the effect of varying the GA paramaters over a range of problems, who comes to the conclusion that a rule of thumb is that, the crossover rate should be 0.6, and the mutation rate should be 0.005. Setting of these GA parameters optimally would also yield a smooth fitness convergence. In this work, one of these three parameters was varied, while the others were kept constant and their effects are shown in the corresponding figures. By setting these parameters optimally we find a considerable improvement in the performance of the GA.

Mutation Rate

In order to see the effect of mutation rate on the fitness values of the GA, the crossover rate was set to 0.7. The mutation rate was then varied between 0.001 to as high as 0.1.The figure 6, shows the settling values of the fitness function for various mutation rates.

Despite the fact that mutation is often regarded as a secondary operator to crossover, it can be seen here that it can produce high rates of learning, if the mutation rate is right. It can be observed that, when the mutation rate is very high such as (0.1) the GA has a faster learning rate in the earlier stage, but it settles down at a lower fitness value indicating that we could have got a better optimum. When the mutation rate is very low such as 0.001, the learning rate itself is very slow and this could be computationally very expensive. It is seen from the figures 1-2, that a mutation rate of 0.1 gives fairly good results. Thus we can conclude that, when the mutation rate is too low then the learning rate is poor, and when the mutation rate is too high, the probability of disruption is also high and the fitness function would settle at a lower value. An intermediate value however gives a high learning rate and better performance for this problem, when crossover is fixed. Using a value of 0.02 gives a good learning rate and better fitness value, so this value was used as a standard through out for further analysis.

Crossover Rate

In order to see the effect of crossover rate on the fitness values of the GA, the mutation rate was set to 0.2 (which was considered to be ideal). The crossover rate was then varied between 0.1 and to as high as 0.9.The figure 7, shows the settling values of the fitness function for various crossover rates.

It can be seen from Figure 7, that a value of 0.6–0.7 seem to be a good choice for this application. When the crossover rate is set to low values like (0.1 or 0.3), the learning rate is faster in the initial stages, but it eventually settles down at a lower fitness value. When the crossover rate is set to a high value of 0.9, the GA settles down in a lower fitness value, indicating that too high or too low the crossover rate will not yield optimal results. When the crossover rate of 0.6 or 0.7 is used, the GA yields good response and settles in a higher fitness value, in less number

of generations. The crossover rate was thus set to 0.7 for further analysis, as this gives higher fitness value when compared to other values.

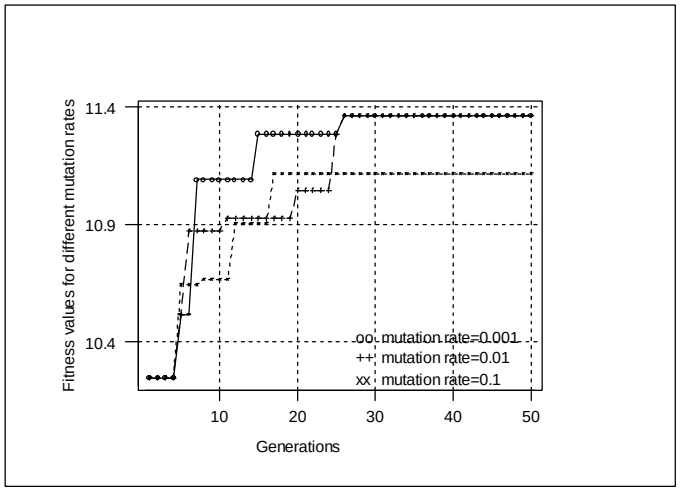


Figure 6. Fitness values for different mutation rates

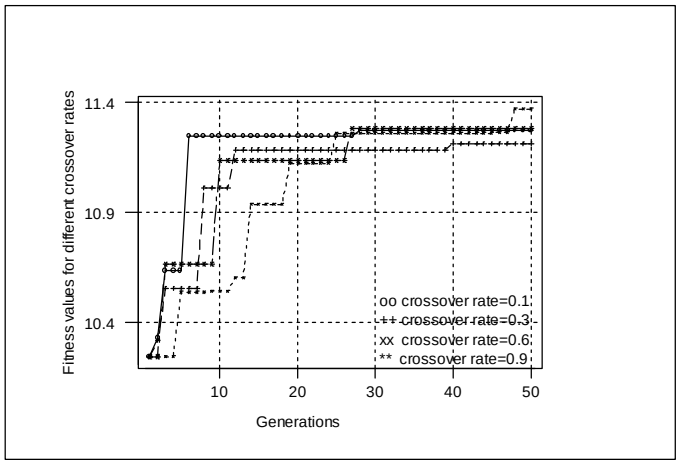


Figure 7. Fitness values for different crossover rates

Population Size

Population is defined as an array of individual solutions. Thus the size of the population is another important parameter that is to be considered for the success of the GA. If the population size is too small, then an insufficient number of samples are sampled and would not yield in the best possible solution. If the population size is too large, the algorithm becomes inefficient as more number of tests is performed than necessary for each generation.

To see the effect of the population size the crossover rate (0.7) and mutation rate (0.2) remains constant. The number of generation was set to be 50. As seen from the figure 8, population size of 50 yields a lower fitness value and when the population size was (100,150 and 200), we notice that they all settle at the same function value except that larger the population size the learning rate is faster but efficiency is lost. So an optimal solution considered was a population size of 100.

It can be seen from these figures that an intermediate value for the population size gives an increase in efficiency and a higher converged score for the same number of generations. Thus the best possible setting was considered to be 100.

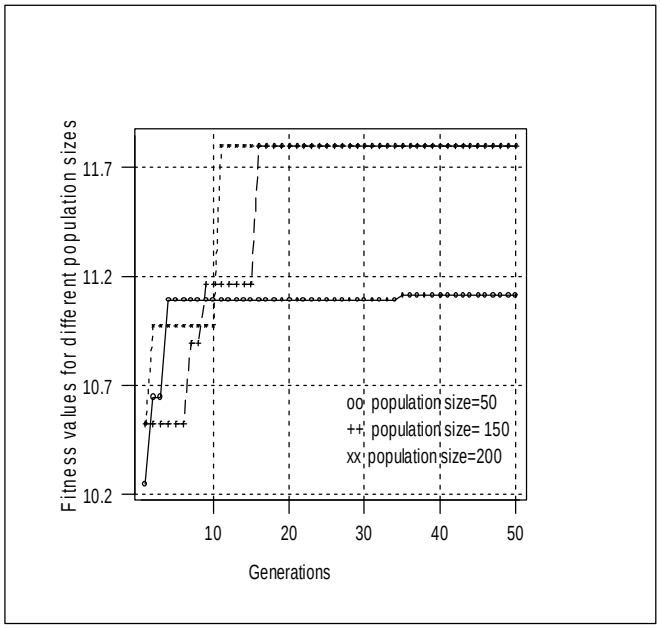


Figure 8. Fitness values for different population size

CONCLUSION

The solution for this multiobjective optimization problem was proposed and developed using GA. A search procedure using the GA was used to provide an optimal or near optimal solutions for this problem. Three different methods for fitness evaluation were tested and the best one was chosen. Autocorrelation techniques were used in order to reduce the number of THUNDER runs and optimize the time consumed. The parameters for the GA were carefully set, by performing a number of trial runs. This increased the efficiency and performance of the GA. But, it can be concluded that, starting the GA with a relatively lower value for crossover and higher value for mutation rate, and then increasing the value of the crossover and decreasing the mutation rate towards the end of the run would yield better results.

ACKNOWLEDGEMENT

Funds for this research effort was provided by Boeing Company. The authors would like to thank Boeing for its

constant support of this research and all previously funded research efforts. The authors would also like to thank the Air Force for providing an unclassified version of THUNDER software.

REFERENCES

[1] L. Altenberg. The Schema Theorem and Price's Theorem. In Foundations of Genetic Algorithms 3, San Francisco, California, USA: Morgan Kaufmann Publishers, 1995.

[2] Brindle, A., "Genetic Algorithms for Function Optimization," Ph.D. Dissertation, University of Alberta, 1981.

[3] Bethke, A. D., "Genetic Algorithms as Function Optimizers," Ph.D. Dissertation, University of Michigan, Ann Arbor, 1981.

[4] B.L. Miller and D.E. Goldberg. Genetic Algorithms, Selection Schemes and the Varying Effect of Noise. IlliGAL report No. 95009. 1995.

[5] Davis, L,. Handbook of Genetic Algorithms. Van Nostrand Reinhold (New York), 1991.

[6] Davis, L, "Genetic Algorithms and Simulated Annealing," Morgan Kaufmann, San Francisco, 1987.

[7] De Jong, K. A., "Analysis of The Behavior of a Class Of Genetic Adaptive Systems," Ph.D. Dissertation, University of Michigan, Ann Arbor, 1975.

[8] De Jong, K. A., "Genetic Algorithms: A 10 Year Perspective," Proc. Int. Conf. On Genetic Algorithms, 1985.

[9] D.E Goldberg., "Genetic Algorithms in search, optimization, and machine learning," Addison-Wesley, 1989.

[10] D.E. Goldberg. A comparative analysis of selection schemes used in genetic algorithms. In Gregory Rawlins, editor, Foundations of Genetic Algorithms, San Mateo, CA: Morgan Kaufmann Publishers. 1991.

[11] D.E. Goldberg and Segrest,P.;"Finite Markov Chain Analysis of Genetic Algorithms"; Proc.2nd Conf. On Genetic Algorithms, pp.1-8, 1987.

[12] J.H. Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press (Ann Arbor), 1975.

[13] J.H. Holland, "Genetic Algorithms and Classifier Systems: Foundations and Future Directions," Proc. 2nd Int. Conf. On Genetic Algorithms, pp. 82-89, 1987.

[14] Hillol Kargupta & David E.Goldberg, "Black box Optimization: Implications of SEARCH", University of Illinois, Urbana-Champaign, 1995.

[15] Schaffer, J. D., "Some Experiments In Machine Learning Using Vector Evaluated Genetic Algorithms," Ph.D. Dissertation, Vanderbilt University, 1984.

[16] Schoen F." stochastic techniques for global optimization: a survey of recent advances." Journal of global optimization, 1991.

[17] Smith, S. F., "A Learning System Based On Genetic Adaptive Algorithms," Ph.D. Dissertation, University Of Pittsburgh, 1980.

[18] Spears,W. M.; "Crossover or mutation?", Foundations of Genetic Algorithms 2, Ed. Whitley,D., Morgan Kaufmann,1993.

[19] Singiresu S.Rao, "Engineering Optimization - Theory and Practice", third edition, A Wisley – Interscience publication, 1996.

[20] Torn A. & Zilinskas A. global optimization, Berlin: Springer-Verlag, 1991.

[21] Yuhui Shi, Russell Eberhart &Yaboin Chen,"Implementation of Evolutionary Fuzzy Systems", IEEE publication, 1999.