FUNCTION OPTIMIZATION OVER INTEGRAL DOMAIN: COMPARATIVE PERFORMANCE OF ELITE GENETIC ALGORITHM FOR SMALL ITERATIONS AND SMALL GENERATION SIZE

S. K. Amponsah and K. F. Darkwah
Mathematics Department,
Kwame Nkrumah University of Science and Technology,
Kumasi

ABSTRACT

In this paper we present a version of the Genetic Algorithm (GA), which we call XSGA to find integral suboptimal global solution of one variable multimodal functions. XSGA makes use of local search and restarts. Our proposed XSGA works better than the elite Genetic Algorithm for small generation size and small number of iterations. Both versions of the GA work well for larger generation size and larger number of iterations.

Keywords: *Heuristics, Genetic Algorithm, Schema, multimodal function.*

INTRODUCTION

In a previous article, "Exact versus heuristics methods in Operations Research" (Amponsah and Darkwah, 2005) showed that real life problems are difficult to solve by exact algorithms that find optimal solutions in finite number of steps. Hence the need for heuristic methods.

Among the heuristic methods discussed was Genetic Algorithm (GA) and Ant Colony Optimization (ACO). These are biologically inspired search algorithms and use population of solutions to search greater number of points in the search space. However, GA is an evolutionary algorithm while ACO is an algorithm of emerging intelligence. We also considered heuristics algorithms like Simulated Annealing (SA) and Tabu Search (TS).

These are point-wise local search algorithms that search the neighbourhood of current solution to find the next solution. They use one solution at a time

This paper focuses on the use of GA in integral function optimization.

Genetic algorithm is an evolutionary algorithm inspired by Darwin's theory of 'the survival of the fittest'. The following principles in biological evolution inspired GA.

Species live in a competitive world.

i) The continued survival of the species depends on fitness competition and having offsprings who are stronger or equally as strong as their parents.

ii) The offsprings genetically take the characteristics of their parents,
iii) The offsprings are however unique and there is probability of slight variations in some of their genes, and
iv) In the competitive environment less fit individuals die off and may not become parents for breeding. If they should breed they have only a small probability for breeding.

Rechenberg introduced Evolutionary Computing (Rechenberg, 1960) and in further development, Holland invented Genetic Algorithm as an adaptive search procedure (Holland, 1975). Koza used GA to develop programmes with graph networks and trees and called them Genetic Programming (Koza, 1992). As shown in Table 1 below the Genetic Algorithm introduced by

Table 1: Simulation of evolutionary principles by Genetic Algorithm

Evolution	Genetic Algorithm
• An **individual** is a genotype of the species.	• An **individual** is a solution of the optimization problem.
• **Chromosomes** define the structure of an individual.	• **Chromosomes** are used to represent the data structure of the solution.
• Chromosome consists of sequence of cells called **genes** which contain the structural information.	• Chromosome consists of a sequence of gene species which are **placeholder boxes** containing string of data whose unique combination give the solution value.
• The genetic information or trait in each gene is called an **allele.**	• An **allele** is an element of the data structure stored in a gene placeholder.
• **Fitness** of an individual is an interpretation of how the chromosomes have adapted to the competition environment.	• **Fitness** of a solution consists in evaluation of measures of the objective functions for the solution and comparing it to the evaluations for other solutions.
• **A population** is a collection of the species found in a given location.	• **A population** is a set of solutions that form the domain search space
• A **generation** is a given number of individuals of the population identified over a period of time.	• A **generation** is a set of solutions taken from the population (domain) and generated at an instant of time or in an iteration.
• **Selection** is pairing of individuals as parents for reproduction	• **Selection** is the operation of selecting parents from the generation to produce offsprings.
• **Crossover** is mating and breeding of offspring by pairs of parents whereby chromosome characteristics are exchanged to form new individuals.	• **Crossover** is the operation whereby pairs of parents exchange characteristics of their data structure to produce two new individuals as offsprings.
• **Mutation** is a random chromosomal process of modification whereby the inherited genes of the offspring from their parents are distorted.	• **Mutation** is random operation whereby the allele of a gene in a chromosome of the offspring is changed by a probability p_m.
• **Recombination** is a process of nature's survival of the fittest.	• **Recombination** is the operation whereby elements of the generation and elements of the offspring form an intermediate generation and less fit chromosomes are taken from the generation.

Holland had the following simulations of the evolutionary principles.

In this paper, we shall use the computational ability of genetic algorithm to find the optimal or sub optimal solution of a function selected from the suite of test functions for GA. One of the motivations for the paper is to show the usefulness of function evaluation optimizer like the GA over calculus based algorithms where derivatives need to be calculated in the process of optimization.

The remainder of the paper is organized as follows:

Section 2, discusses the study problem, the evolutionary strategy of

(Holland, 1975) and the simple genetic algorithm (SGA) introduced by (Goldberg, 1989).

Section 3, discusses Elitist Genetic Algorithm (EGA)

Section 4 introduces our proposed algorithm which uses the SGA together with restarts and local search while keeping track of the best global solution. We call the procedure XSGA. Sections 5 and 6 provide the results of the EGA and XSGA algorithms and the conclusion respectively.

STUDY PROBLEM

Genetic algorithm has been used to solve a variety of combinatorial (discrete) optimization problems. They include function optimization, evaluation of algebraic expressions, the traveling salesman problem (TSP), the bin packing problem and the knapsack problem.

In this paper we shall use the optimization of a function as a case study to compare the elitist genetic algorithm (EGA) and the XSGA.

We use the single variable form of the multimodal Whitley test function

$$f(x) = x * \sin\left((0.4) * \left(\sqrt{\left(\frac{(x+47)}{2}\right)}\right)\right) + (x+47) * \sin\left(\sqrt{\left(\frac{(3x+94)}{2}\right)}\right) + 3560$$

to find the global minimum, $x = 0,1,2,3, \ldots,$ 2050 is an integral domain. Since the domain is integral the optimal solution may not be found if there is decimal placement in the optimal solution. In such a case we shall obtain suboptimal solution. Since the domain is discrete the problem is combinational and also an NP-hard problem. The solution *(x)* will be given as a chromosome representation of a string of binary numbers and for any *(x)* the evaluation of the measure of fitness will be based on the objective function *f(x)*.

Genetic Algorithm

Calculus based algorithms for solving function optimization involves computing and solving derivatives and the use of possible solutions to evaluate the function. Computing derivatives could be complex and sometimes intractable. Implementing an algorithm for such method becomes more difficult as the problem assumes greater complexity. However, heuristic methods such as Genetic Algorithm solve such problems through function evaluation and hence the relevant algorithm is easier to implement. Genetic Algorithm was introduced by (Holland et al, 1975) as a computational analogy of adaptive systems, which find the global solution irrespective of the initial solution. The result was an evolutionary strategy, which was called genetic plan and later named Genetic Algorithm (Rana, 1999). Genetic Algorithm is a discrete population event simulation. Given a population at time *t*, genetic operators are applied to produce a new population at time $t + 1$. A step-wise evolution of the population from time *t* to $t+1$ is called a generation. The Genetic Algorithm for a single generation is based on the general GA framework of Selection, Crossover, Mutation and Recombination.

The evolution of a population at time *t* to time $t+1$ is analysed by schema. A schema is a tem-

plate made up of concatenation of the set {0,1,*} into a binary string of length L. The total number of 0 and 1 bits in the schema is the order of the schema. '*' is a wild card used to represent 0 or 1. The instances of a schema are the binary string realisation of the schema by replacing '*' with 0 or 1. One of the eight instances of the six character schema H=1**0*0 is the string element 101010.

It is assumed that a generation of population has a proportional representation of the total number of schemata available that corresponds to the length *(L)* of the chromosome representation of the individuals (solutions) of the optimization problem. The GA framework processes schemata by using the current population to hold competition between schemata implicitly through the population.

The result of the competition is that lower order schemata with higher fitness are processed into higher order schemata with yet higher fitness.

Higher order schema will mean the wild cards '*' have been replaced by specific binary digits. Subsequent iterations will lead to the repetition of the same schema which implies convergence to the schema representation of the optimal solution.

Let $p(H,t)$ be the proportion of schema H found in the population at time t then the average proportion of schema H found in the population at time t+1 is $p(H,t+1)$ and is given by the Schema Theorem (Holland, 1975).

$$p(H,t+1) \geq \left[p\frac{f(H,t)}{\bar{f}}\right]\left[1-p_c\left(\frac{\Delta H}{L-1}\right)\left(1-p(H,t)\frac{f(H,t)}{\bar{f}}\right)\right] * \left[(1-p_m)^{0(H)}\right]$$

The formula is a lower bound for computing the proportional representation of a single schema *(H)* in the next generation, *(t+1)*.

The three factors of the formula respectively model selection, crossover and mutation operations

$\left[p(H,t)\frac{f(H,t)}{\bar{f}}\right]$ models the effect of selection,

$f(H,t)$ is average fitness of schema,

$\bar{f}$ is average fitness of population and

$\left[1-p_c\left(\frac{\Delta H}{L-1}\right)\left(1-p(H,t)\frac{f(H,t)}{\bar{f}}\right)\right]$ models the effect of crossover, where

P_c is crossover probability, ΔH = defining length of the schema, L=length of chromosome string P_m *is* mutation probability, $0(H)$ = order of the schema,

$[(1-p_m)^{0(H)}]$ models mutation,

Based on schema processing, (Goldberg, 1989) presented a standard genetic algorithm, which was called simple genetic algorithm (SGA). SGA was based on Holland's schema theorem. The steps of the SGA are given in Table 2.

Table 2: Algorithm steps of Simple Genetic Algorithm

Step 1: Code the individual of the search space
Step 2: Initialize the generation counter (g=1)
Step 3: Choose initial generation of population (solution)
Step 4: Evaluate the fitness of each individual in the population
Step 5: Select individuals of best fitness ranking by fitness proportionate probability.
Step 6: Apply crossover operation on selected parents
Step 7: Apply mutation operation on offspring
Step 8: Evaluate fitness of offspring
Step 9: Obtain a new generation of population by combining elements of the offspring and the old generation by keeping the generation size unchanged
Step 10: Stop, if termination condition is satisfied
Step 11: Else g = g + 1

Termination conditions

The algorithm terminates when a set of conditions are satisfied. At that point the solution with highest fitness among the current generation of the population is taken as the global solution or the algorithm may terminate if one or more of the following are satisfied.

i) A specified number of total iterations are completed.
ii) A specified number of iterations are completed within which the solution of best fitness has not changed.
iii) The standard deviation of the generation of the population approaches a given large value.
iv) The average fitness of the generations of population do not differ significantly from the solution of best fitness.

Illustration

The implementation of the SGA on the basis of the schema theorem will involve

a) Using fitness proportionate selection procedure.
b) Using single crossover point for the crossover operation

We provide an example of function optimization by processing a population to get a new population. In order to illustrate the selection procedure used in the XSGA procedure, we take the crossover probability to be P_c =1such that all selected parents will have the crossover operation.

$$\min f(x) = \frac{x^4}{4} - \frac{7}{3}x^3 + 7x^2 - 8x + 15,\ x = 0,1,2,3,...,10$$

Step 1: Encoding: The search space is x= 0, 1, 2,…,10. We encode elements of the search space in a binary sequence. Express x=10 and x=0 in binary sequence to obtain

$10=1010_2$ and $0 = 0000_2$. Thus x = 10 is an individual and 1010 is its chromosome representation. The chromosome has 4 genes placeholders for the alleles such that x =

x =	*	*	*	*

The allele information in the genes will be the binary numbers 0 and 1

The chromosome for x = 9 is therefore

1	0	0	1

The objective evaluation is $f(9)$ = 449.25

There are 2^4 permutations for a binary string of length 4. These 2^4 permutations consist of both infeasible and feasible solutions. There are 11 feasible solutions, which constitute our search space and the rest form the infeasible set. Since the solution set is restricted to the integers we look for suboptimal solution.

Step 2: Generation counter for populations: we set g=1 to set the counter tag for the initial population. Since we are processing only the initial population there will be no further increment of the counter.

Steps 3 and **4**: Initial population: we select at random 4 individuals (solutions). We choose the number to be the same as the length of a chromosome string.

Take $x_1 = 5,\ x_2 = 1,\ x_3 = 3,\ x_4 = 9$

Table 3, gives the serial number, the chromosome representation, the solution and objective evaluation of each solution.

Table 3: Serial listing of Chromosomes and their objective values

Serial No	Chromosome $g(x)$	Solution (x)	Objective $f(x)$
1	0101	5	14.58
2	0001	1	11.92
3	0011	3	11.25
4	1001	9	449.25

Step 5: Selection of parents: Since the problem is a minimization problem we use the fitness formula $F=(f_{max} - f_i)+ 1$ (Cox, 2005) to reverse the magnitude of the objective. We then use fitness proportionate selection rule that requires the calculation of:

i) Probability of selection $p(select)=\frac{F(x)}{\sum F(x)}$

ii) Expected count $= \frac{F(x)}{\overline{F(x)}}$, $\overline{F(x)}$

is the average fitness of the population

iii) Actual count = Round up of expected count to nearest integer

Table 4 provides basis for the determination of the number of times an individual will be selected as a parent. It is obtained by adding 1 to the individual's actual count. This means x = 5, 1, 3 will be selected twice while x = 9 will be selected once.

Table 4 is then sorted in ascending order according to P(select) and the column for cumulative of F(x) is added to obtain Table 5. The roulette operation is used to obtain the order of parent pairings. The roulette was generated by using the Matlab random function

prb = (m-n)*rand + n where m = max(CumF(x)) and n = min(CumF(x)).

The sequence of values (rv) generated by the random function and the corresponding individuals selected are:

1. (rv =769, x = 1);
2. (rv = 352, x = 5);
3. (rv = 1262.3, x = 3);
4. (rv = 286.63, x = 5);
5. (rv = 854.24, x = 1);
6. (rv = 912.59, x = 3);
7. (rv = 0, x = 9)

The order of pairings of parents are then (1,5), (3,5), (1,3), (9,9). The individual x = 9 is paired to itself since it is the last to be selected and being an odd (7th) selection it cannot be paired with any other individual. This unfortunately breaks the original selection rule

Step 6: Apply crossover operation: The pairings (1, 5), (3, 5), (1, 3), (9, 9) are used and the space

Table 4: Determination of the actual count of chromosomes

Number	chromosome	x	$F(x)$	P (select)	Expected count	Actual count
1	0101	5	435.6667	0.3316	1.3262	1
2	0001	1	438.3333	0.3336	1.3343	1
3	0011	3	439.0000	0.3341	1.3364	1
4	1001	9	1.0000	0.0008	0.0030	0
Sum			1314	1	4.0000	3
Average			328.50	0.25	1	0.75
Max			439.0000	0.3341	1.3364	1

Table 5: Chromosomes are arranged in their cumulative positions

Number	Chromosome g(x)	x	F(x)	P (select)	Ecount	Acount	Cum. F(x)
4	1001	9	1	0.0008	0.0030	0	1.00
1	0101	5	435.6667	0.3316	1.3262	1	436.67
2	0001	1	438.3333	0.3336	1.3343	1	875.00
3	0011	3	439.0000	0.3341	1.3364	1	1314.00

number 2, representing the crossover point between the second and third binary digits is chosen. Table 6 below shows the results of the crossover operations.

Table 6: Results of crossover operations

x	Parents	Offsprings
1	00 ↕ 01	0001
5	00 ↕ 01	0101
3	00 ↕ 11	0001
5	01 ↕ 01	0111
1	00 ↕ 01	0011
3	00 ↕ 11	0001
9	10 ↕ 01	1001
9	10 ↕ 01	1001

Step 7: Apply mutation operation on offsprings: A random number is assigned to each allele of a chromosome offspring. Table 7 below lists the chromosome and random numbers between 0 and 1. Mutation (exchange of 0 and 1) is done if Rand(i,j) < 0.3; where 'i' is serial number of chromosome and 'j' is the loci of the allele or binary bit. The last column shows the mutated offsprings.

Step 8: Evaluation of mutated offspring: Table 8 shows the mutated offspring, the individual it represents and the evaluation of the objective $f(x)$.

$x = 13$ and $x = 11$ are not evaluated because they fall outside the domain or solution space. They are infeasible solutions.

We also delete repeated mutated chromosomes and join the rest to members of the old generation to get the recombination. In Table 9 the serial numbers with asterisks are for chromo-

Table 7: Results of mutating offspring chromosomes

Serial Number	Offspring chromosome	Allele 1	Allele 2	Allele 3	Allele 4	Mutated chromosome
1	0101	0.539	0.0008	0.341	0.69	0001
2	0001	0.412	0.2	0.857	0.888	0101
3	0111	0.49	0.609	0.717	0.431	0111
4	0001	0.275	0.029	0.498	0.972	1101
5	0011	0.325	0.671	0.464	0.292	0010
6	0001	0.478	0.449	0.682	0.73	0001
7	1001	0.660	0.3419	0.2897	0.3412	1011

Table 8: Objective evaluation of mutated offsprings

Number	Mutated chromosome	x	$f(x)$
1	0001	1	11.92
2	0101	5	14.58
3	0111	7	101.92
4	1101	13	-
5	0010	2	12.33
6	0001	1	11.92
7	1011	11	-

Table 9: Combination of offsprings and parent generation

Serial No	Chromosome g(*x*)	Solution (*x*)	Objective *f(x)*
1	0101	5	14.58
2	0001	1	11.92
3	0011	3	11.25
4	1001	9	449.25
5*	0001	1	11.92
6*	0101	5	14.58
7*	0111	7	101.92
8*	0010	2	12.33

somes from the offsprings. The chromosome $g(x) = 0010$ representing $x=2$ will be added to the old generation of population to form a new generation. The chromosome $g=(x) = 0001$ representing $x = 1$ and which has the lowest $f(x)$ is already part of the old generation of population, so is the chromosome $g(x) = 0101$ representing $x = 5$.

We further delete repeating chromosomes from the recombination to get the intermediate generation listed in Table 10 and ordered in ascending order.

Step 9: Update of generation
Chromosome 4 in the old generation is deleted and chromosome 5 in the offspring list used replaces it. The new generation is then given by Table 11.

Table 10: Intermediate generation

Serial No.	Chromosome g(*x*)	Solution (*x*)	Objective *f(x)*
1	0011	3	11.25
2	0001	1	11.92
3	0010	2	12.33
4	0101	5	14.58
5	0111	7	101.92
6	1001	9	449.25

Table 11: New generation of population

Serial No.	Chromosome g(*x*)	Solution (*x*)	Objective *f(x)*
1	0011	3	11.25
2	0001	1	11.92
3	0010	2	12.33
4	0101	5	14.58

ELITIST GENETIC ALGORITHM (EGA)

This is a modification of the SGA obtained by introducing intensification of search. This is achieved by retaining a given number of the best fit individuals from the previous generation to be reused in the current generation. This inhibits the use of greater number of offsprings who could be in the new generation and therefore could become parents and thereby participate in the three ingredients of selection, crossover and mutation. Crossover and Mutation are processes that produce diversification of movement in the search space and which is necessary for the GA framework to exploit the domain space. Thus the EGA needs to balance intensification which tends to preserve the solution(s) of best fit in the old generation so as to be automatically included in the new generation with diversification resulting from the production of offsprings from crossover and mutation and thereby need to replace solutions of the old generation.

Even though the implementation of elitism has been successful in finding global optimum, the potential exist for getting trapped in a local optimum especially for multimodal functions (Godberg, 1989).

Pseudocode for EGA

Steps 1 to 8 and Steps 10 to 11 are implemented as found in the SGA .Step 9 is implemented as follows:

Set aside the best solution from the old generation. Combine elements of the rest of the old generation to elements of the offspring. Add to the previous best solution so as to get a new generation of same size as the previous generation.

XSGA

The literature shows a variety of GA's that use selection, crossover operations in varying ways to solve combinatorial optimization problems. The genetic algorithm proposed here uses the SGA as basis with modification of restarts and local search. It is established that as a GA converges to an optimal solution the crossover operation is ineffective in producing diversification and mutation remains the active operation for search diversification. Thus if an intensification procedure such as elitism (Dejong, 1992) is added then the effect of diversification will be greatly diminished and the algorithm can easily be trapped in a local optimum especially for multimodal problems. The present algorithm saves the global best solution into a variable and for any iteration the current best solution replaces the global best solution if it is better. As many offsprings will replace individuals in the old generation that are less fit than the offsprings The generation is re-initialized and a local search for local optimum performed for a given number of times in the course of the iterations. This is to exploit the observation made by (Hansen and Mladenovic, 2003) that for many problems local optimum with respect to one or several neighbourhoods are relatively close to one another.

Termination of the algorithm is made after a certain number of iterations. The steps of our proposed algorithm are:

Step 1: Code the individual of the search space as L-bit chromosomes.

Step 2: Initialize the generation character (g=1) .

Step 3: Randomly choose initial generation of population of n individuals.

Step 4: Evaluate the fitness of each individual in the generation. Store the individual of the best fit as global solution and as current best solution.

Step 5: Select individuals of best fitness rank by fitness proportionate probability to be paired.

Step 6: Use tournament selection with replacement to pair parent for mating. Apply one point crossover operation on selected pairs of parents to produce offspring.

Step 7: Apply mutation operation on offsprings.

Step 8: Evaluate the offsprings.

Step 9: Obtain a new generation of population by combining element of offspring and the current generation according to fitness and to keep

the generation size unchanged at n. Update the current best solution and the current generation.

Step 10:

i) If current best solution is better than global best solution then replace global best solution with current best solution and set the local search gauge counter at k=0.

ii) If after a prescribed number of iterations (k=m) the current best solution is not better than the global best solution perform local search centered on the global best solution and reinitialize the generation. The current generation, the reinitialize generation and the local search space are combined to select a new generation.

Step 11: g = g+1

RESULTS

Procedure

Matlab programs were written for the Elite Genetic Algorithm (EGA) and the XSGA. using the multimodal function

$$f(x) = x*\sin\left((0.4)*\left(\sqrt{\left(\frac{(x+47)}{2}\right)}\right)\right) + (x+47)*\sin\left(\sqrt{\left(\frac{(3x+94)}{2}\right)}\right) + 3560$$

for x= 0,1,2,3,...,2050. For generation sizes of 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 500, 600 and 700 individuals, programme iterations of 10, 20, 30 were processed for each of the generation sizes. A total of 20 runs were used for any given pair of generation size and programme iteration. From the solutions obtained for the 20 runs the number of appearances of the expected solution is counted.

The results for a given number of iterations were partitioned into 2 sets for the cases where

a) *max prod* is less than the domain size, and
b) *max prod* is equal or greater than the domain size.

prod = (generation size)x(number of iterations) and

max prod = (generation size)x (30 iterations).

The results are illustrated below for mutation probability P_m = 0.25, crossover points *cop* = *6* and the crossover probability P_c = 1

Table of Results

Table 12a: Results for 10 iterations, *max prod* **less than domain size**

Generation size	XSGA : Solution Count	EGA:Solution Count
10	5	0
15	3	0
20	5	0
30	12	1
40	17	1
50	20	0
60	20	2

Table 12b: Result for 10 iterations, *max prod* **greater than domain size**

Generation size	XSGA: Solution Count	EGA: Solution Count
70	20	0
80	20	1
90	20	1
100	20	1
200	20	3
500	20	6
600	20	2
700	20	4

Table 13a: Results for 20 iterations, *max prod* **less than domain size**

Generation size	XSGA: Solution Count	EGA: Solution Count
10	2	1
15	2	1
20	6	0
30	14	0
40	16	0
50	16	0
60	19	0

Table 13b: Result for 20 iterations, *max prod* greater than domain size

Generation size	XSGA:Solution Count	EGA:Solution Count
70	19	2
80	20	0
90	20	0
100	20	0
200	20	3
500	20	7
600	20	2
700	20	8

Table 14a: Result for 30 iterations, *max prod* less than domain size

Generation size	XSGA: Solution Count	EGA:Solution Count
10	4	1
15	5	2
20	11	0
30	17	0
40	18	0
50	17	0
60	20	1

Table 14b: Result for 30 iterations, *max prod* greater than domain size

Generation size	XSGA Solution Count	EGA Solution Count
70	20	0
80	20	2
90	20	1
100	20	0
200	20	1
500	20	7
600	20	4
700	20	9

DISCUSSIONS

The results show that EGA performs absymal for lower population size and lower number of iterations. In no situation was the count of the expected solution greater than 10. This happened even for cases where *prod* was greater than the domain size. This may indicate that the EGA could not cover the domain space as expected and that the EGA may have been stuck at local optima without being able to escape.

It is significant that the EGA was not responding proportionately to the increases in generation size which may suggest that there is larger critical size at which the response of EGA will be noticed.

On the other hand the XSGA did better than the EGA in all cases and was seen to respond to the increases in the generation size until a size was reached where the response gave perfect or near perfect results. When perfect results are reached they are stable. The range of size for reaching such perfect results was between generation sizes of 50 and 60 which gave average *prod* to be about half the domain size. This may indicate that XSGA was able to search the domain even when *prod* was about half the domain size. The results are perfect and stable for iterations with *prod* greater than the domain size.

CONCLUSIONS

The results show that EGA does not perform well for small number of iterations and small generation size. (Earl Cox ,2005), indicated that generation size for GA may be taken to be 5 times the length of the string representation of the chromosome or half the domain size, which ever is smaller. Going by this rule the generation size should be 60 (which is smaller of 60 and 1025), however the EGA fail to respond to such generation size for lower number of iterations.

It is significant that for generation size of 60 and onwards the XSGA generally gave stable perfect solutions. The authors therefore recommend the XSGA over EGA for the computation of integral global optimization

APPENDIX A

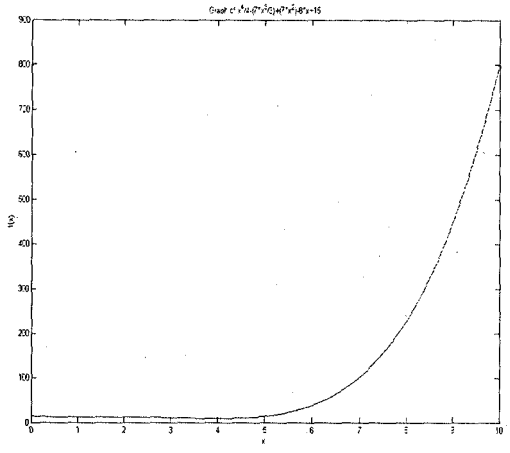

Fig. 1: Graph of function used in illustrative example

APPENDIX B

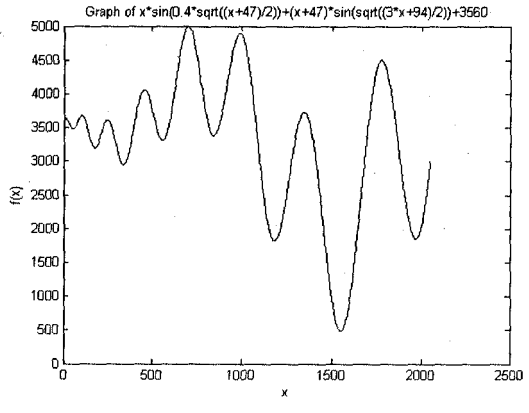

Fig. 2: Graph of Whitley's one-dimensional multimodal function

APPENDIX C

Matlab code for the main XSGA programme

```
function[xglobal,fxglobal,k,g,matigene]=mga
(objFun,sdomain,...
ipopsize,mup ,cop,iter);
clear global;
%----------------------------------------
%--Defintion of variables
%----------------------------------------
%objFun    = objective function
%sdomain   = domain
%ipopsize  = initial population
%mup       = mutation probability
%cop       = crossover point
%elite     = number of elite individuals
%m         = gauge iterations
%----------------------------------------
%Initializing a Generation of n-individuals
%----------------------------------------
partpop=round(ipopsize/6);
partdom = round((length(sdomain))/200);
m=round(iter/5);
mindom=min(sdomain);
maxdom=max(sdomain);
igene=createpop(sdomain,ipopsize);
sym x;
objFun;
xglobal=-1  ;        %[For Initial Generation]
fxglobal=-1;
matigene=[ ];
%---------------------------
%Set Counter here
%---------------------------
g=1;
k=0;
while g<iter;
   matigene=[matigene,igene];
    %Perform Calculations
    fx=subs(objFun,'x',igene);
afx=mean(fx);
sfx=sum(fx);
[maxfx,imaxfx]=max(fx);
xmax=igene(imaxfx);
ps=p_select(fx,sfx);  %Probability for Selection
```

```
ec=e_count(fx,afx);
ac=round(ec);
%--This array contains all values of the above
calculations
 tbl=[igene fx ps ac];
 %---------------------------
%preparing table for roulette
%---------------------------
ostbl=sortM(tbl,1,3);
stbl=flipud(ostbl);
stbl(:,5)=stbl(:,4);
stbl(:,4)=cumsum(stbl(:,3));
%--getting the parents::FUNCTION ROU-
LETTE::
parents=roulette(stbl);
%--get pair::FUNCTION CHOOSEPAIR::
spairs=choosepair(parents);
%--crossover::FUNCTION VCROSSOVER::
cparents=vcrossover(spairs,cop);
%--mutation::FUNCTION MUTATE::
b_cparents=dec2bin(cparents);
mut=mutate(b_cparents,mup);
mut(isspace(mut))='';
new_offspring=bin2dec(mut);
%start monitoring from here
someoffspring=outsidedom
(new_offspring,maxdom);
n_igene=[igene;someoffspring];
fit_igene=fitness(n_igene,objFun);
%--Remove repeating new generation from
NEW and OLD offspring
n_igene=removeRepeat(fit_igene);
 igene=n_igene(1:ipopsize,:);
 %--------------------------------
 %TERMINATION CONDITION
 %--------------------------------
  if maxfx>fxglobal;
     xglobal=xmax;
     fxglobal=maxfx;
     k=0;
     g=g+1;
  else
     k=k+1;
  end;
  if (k>0)&(k<m)
     g=g+1;
 elseif(k>0)&((k==m)|(k==2*m)|(k==3*m))
      if partpop<partdom
      upbnd=xglobal+partpop;
      lobnd=xglobal-partpop;
      else
      upbnd=xglobal+partdom;
      lobnd=xglobal-partdom;
      end
      if upbnd>maxdom;
        upbnd=maxdom;
      end
      if lobnd<mindom;
         lobnd=mindom;
      end
      subdom=[lobnd:upbnd];
      subfx=subs(objFun,'x',subdom);
       [fxlocal,ixlocal]=max(subfx);
        xlocal=subdom(ixlocal);
      if fxlocal>fxglobal;
         xglobal=xlocal;
      end
      igene;
      newigene=createpop(sdomain,ipopsize);
      comigene1=compelite(igene,newigene);
      comigene2=[igene;comigene1];
      comigene3=compelite
(comigene2,subdom');
      comigene=[comigene2;comigene3];
      gogene=fitness(comigene,objFun);
      igene=gogene([1:ipopsize],:);
      g=g+1;
  elseif k>0
     g=g+1;
    %-----------------------------
  %NEW CALCULATION HAPPENS HERE
    %-----------------------------
     %[For Initial Generation]
  end;
end;
xglobal;
fxglobal;
k;
g;
matigene;
return;
```

ACKNOWLEDGEMENT

We wish to acknowledge Mr. Gordon Amoako of the Department of Mathematics, Kwame Nkrumah University of Science and Technology, Kumasi, for writing most of the Matlab programmes.

REFERENCES

Amponsah, S. K. and Darkwah, K. F. (2005). Exact Versus Heuristic Methods in Operations Research, *Journal of Ghana Science Association* 7(1):60 – 67.

Dejong, K. A. (1992). Genetic Algorithm are not functional Optimizers. In: Whitley L D (Ed.), *Foundations of Genetic Algorithm 2*, Morgan Kaufmann, New York, 5 – 18.

Dejong, K. A. (1975). Analysis of the behaviour of a class of Genetic Adaptive Systems, PhD dissertation, Department of Computer and communication Sciences, University of Michigan, Ann Arbor.

Cox, E. (2005). Fuzzy Modeling and Genetic Algorithms for Data Mining and Exploration, Morgan Kaufmann, San Francisco, CA. 376 – 396.

Fogel, D. B. and Ghozail, B.(1997). Schema Processing under proportional selection in the presence of random effect. *IEEEE Transactions on Evolutionary Computation* 1 (4) :290 – 293.

Goldberg, D. E. (1989). Genetic Algorithm in Search, Optimization, and Machine Learning, Addison-Wesley,New York.

Gomez, O and Baran, B.(2004). Reasons of ACO's success in TSP. In:

Dorigo, M., Birattari, M., Blum, C., Gambardella, L. M., Mondada, F. and Stutzle, T.(Eds), Proceedings of ANTS 2004-Fourth International Workshop on Ant Colony Optmisation and Swarm Intelligence, Volume 3172 of LNCS, Springer-Verlag, Brussels, 226-237

Hansen, P and Mladonovic, N. (2001). Variable Neighborhood Search: Principles and Applications, *European Journal of Operations Research* 130: 449-467.

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

Koza, J. (1992). Genetic Programming: On the programming of computers by means of Natural Selection, MIT Press, Cambridge, MA.

Mladonovic, N. and Hansen, P.(1997). Variable Neighbourhood Search, *Computers and operations Research*, 11: 1097 -1100.

Poli, R. and Langdon, W. B. (1998). Schema Theory for Genetic Programming with one-point crossover and point mutation, *Evolutionary computation* 6(3): 231- 252.

Whitley, L. D. (1993). Foundations of Genetic Algorithm 2, Morgan Kaufmann, San Mateo, CA?

Whitley, L. D., Gordon, V. and Mathias, K. (1994). Lamarckian Evolution, the Baldwin Effect and Function Optimization. *Parallel Problem Solving from Nature,* 111: 6 -15.

Whitley L D., Mathias, K., Rana, S. and Dzubera, J. (1996). Evaluating Evolution Algorithms. *Artificial Intelligences Journal,* 85: 245-276.