International Review on Computers and Software (I.RE.CO.S.), Vol. 10, N. 1
ISSN 1828-6003 January 2015

Multiple-Population Genetic Algorithm for Solving Min-Max Optimization Problems

Mohammad Alshraideh[1], Luay Tahat[2]

Abstract – *A min-max optimization problem was originally designed for simultaneous maximization of the same object functions during the same optimization run. The existing approaches for solving min-max problem using genetic algorithms is mainly focused on maintaining a single-population of candidate tests. In this paper, we explore a new approach for using genetic algorithms (GAs) to solve min-max problems. The approach uses a two-population GA to find Maximum and Minimum goals of separate search processes using distinct island populations. The advantage of the suggested approach is that its ability to explore a greater variety of execution paths increases the search efficiency under certain conditions. By applying this to a collection of benchmarks problems, it has been shown experimentally that the proposed multiple-population algorithm out performs the single-population algorithm in terms of the number of executions, execution time, performance improvement, and efficiency.*

Keywords: *Single-Population, Multiple-Population, Genetic Algorithm, Island, Min-Max Problems*

Nomenclature

GAs	Genetic Algorithms
Min	Minimum
Max	Maximum
F'	First derivative
F''	Second derivative
ACO	Ant colony optimization
SSCLI	Shared Source Common Language Infrastructure
pop_i	Population i
Ind_j	Individual J
T_m	Execution Time of the Multiple-Population Algorithm
T_{cp}	Computation Time
T_{cm}	Communication Time
T_s	Execution Time of the Single-Population
P_i	Performance Improvement

I. Introduction

Genetic algorithms are computer-based optimization methods that use the Darwinian evolution of nature as a model [1]-[29]. The solution-base of the problem is encoded as being similar to individual chromosomes consisting of several genes. The GAs are thus simplified models derived from natural genetics. For example, in GAs, the individual (phenotype) is usually exactly derived from the chromosome (genotype), whereas in nature, the phenotype is not directly derived from the genotype, but the age, living conditions, diseases, accidents etc. of an individual have effects on the phenotype.

In general, genetic algorithms tend to work better than traditional optimization algorithms because they are less likely to be off track by local optima [7], [20].

This is because they do not make use of single-point transition rules to move from one single instance in the solution space to another. Instead, GAs take advantage of an entire set of solutions spread throughout the solution space, all of which are experimenting upon many potential optima.

In this paper, the problem is finding the Maximum and Minimum values of the function. Although the problem of satisfying Max-Min has been tackled before, the method has usually been focused on one limit (max or min) at a time, using a single-population of solutions [8], [9], [11], [14].

The goal of this paper is the search for input to find a particular value as Maximum and another value as Minimum where these values cannot be solved easily using simple mathematics. As shown in Equation (1) below, since there are two paths (Max& Min), two searches would be conducted; one would seek input data to find Max and the other would seek input data to find Min:

$$f(x)=\frac{1}{4}\left(x^3-6x^2+9x=7\right) \quad \text{where } 0\le x\le 2 \tag{1}$$

From Fig. 1 below which represents the function in Eq. (1), we can see that the maximum value is $\frac{11}{4}$ at $x=1$ and

the minimum value is $\frac{7}{4}$ at $x = 0$.

In this paper, the effectiveness of a genetic algorithm that searches for multiple paths and maintains distinct "island" populations, each of which is evolved with the objective of satisfying a particular path, is compared with that of a genetic algorithm that maintains a single-population which, at any one time, is evolved with the objective of satisfying one value only (Max or Min).

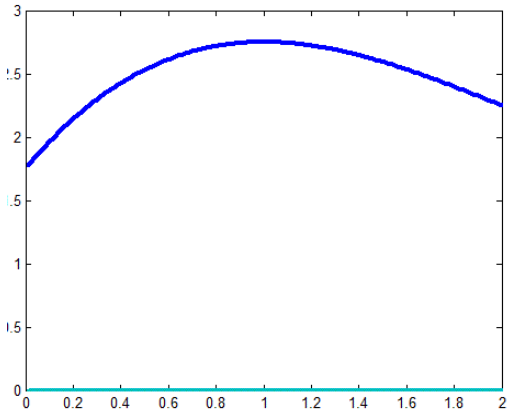


Fig. 1. The landscape of equation (1)

The remainder of this paper is organized as follows: Section 2 presents background and related works.

The proposed multiple-population genetic algorithm is presented in Section 3. In Section 4, the experimental environment from hardware, software and objects is described. The results of the empirical study are presented in Section 5. Concludes and outlines future works presented in Section 6.

II. Background and Related Work

Fig. 2 summarizes some basic terminology. Since we are talking about points on the graph of a function, there are two numbers for each point: the first and the second coordinate. In the case of these extreme points, i.e. either maximum or minimum, the first coordinate is called the max (min) point and the second coordinate is called the max (min) value.

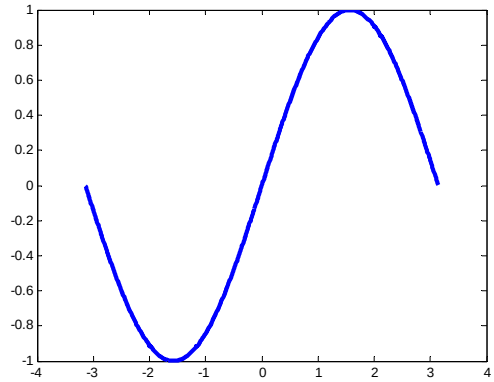


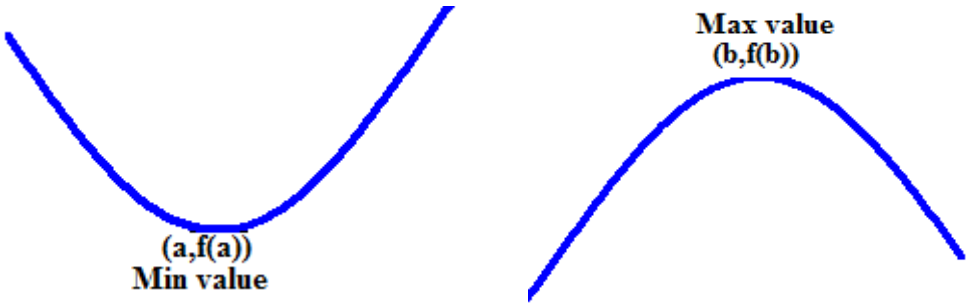
Fig. 2. Drawing Sinefunction lines to the graph limits at the extreme points

The global maximum function value[s] is/are the function values[s] that are greater than other function values. Similarly for global minimum function values are the function values that are less than other function values.

Also, the problem of determining all of the extreme values is more complicated. Now let's figure out which computation has to be done in order to find the extreme points. Let's draw *sin* lines to the graph limits at the extreme points as shown in Figure 2. By looking to the concavity at points in Figure 2, it is clear that the graph has to be concave downward at the maximum and concave upward at the minimum. So the second derivatives value must be negative at the maximum and positive at the minimum. This is summarized in Figures 3 below.

In order to locate the extreme points and classify which points are maximum and which are minimum the following steps are required:

1. Compute the derivative $f'(x)$.
2. Solve the equation $f'(x) = 0$. There might be several solutions.
3. Compute the second derivative $f''(x)$.
4. Evaluate $f''(x)$ for each solution obtained in step2.
5. Classify each point as local maximum or local minimum.
6. Compute the function value for each point obtained in step 2.



f '(a) =0
f ''(a) >0

f '(b)=0
f ''(b) <0

Figs. 3. Describing concave downward at the maximum and concave upward at the minimum

It is not always possible to find the min and max values by using first and second derivative, for example: in Eq. (2) (the landscape of this equation is shown Fig. 4), it is very difficult to find max or min values by using first and second derivative, but we can use heuristic technique (e.g. GA) to find these values:

$$f(x,y) = x^3 - 3x^2y + y^3 \quad \text{where } -10 \le x \le 10,\ -10 \le x \le 10 \tag{2}$$

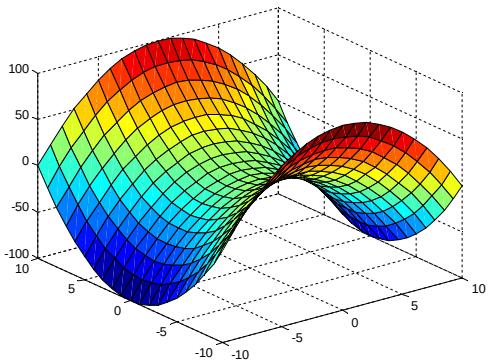


Fig. 4. The landscape of Eq. (2)

Another example in Eq. (3) from [22]:

$$f(x,y) = exp\left(-\frac{1}{3}x^3 + x - y^2\right) \tag{3}$$

Fig. 5 below plots the surface z=*f* (*x*, *y*). Notice the relative maximum at (*x*=1, *y*=0). (*x*=-1, *y*=0) is a relative maximum if one travels in the y direction and a relative minimum if one travels in the x-direction. Near (-1, 0) the surface looks like a saddle.

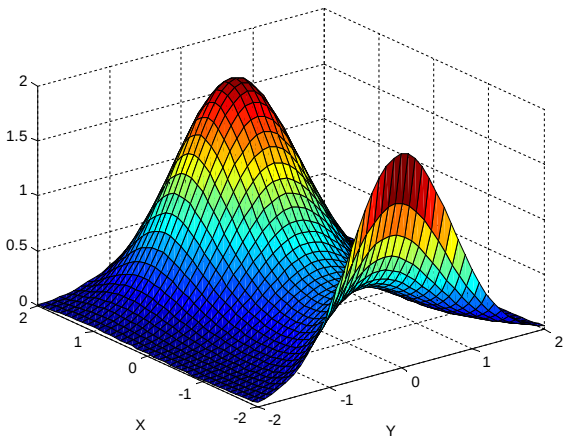


Fig. 5. Landscape of Eq. (3)

In general, the min-max problem can be defined as:

$$\min_x \ f(x) \tag{4}$$

where:

$$f(x) = \max_{i=1 \longrightarrow m} f_i(x) \quad \text{with} \quad f_i(x) : S \to R^n,\ i = 1,...,m \tag{5}$$

Such problems are encountered in numerous optimal controls, engineering design, discrete optimization, Chebyshev approximationand game theory applications [18], [19].

Specifically, in the Chebyshev approximation, given a function $g : Y^{(0)} \subset R^m \to R$, the Chebyshev approximate p_z of g in p_n solves the following minimax problem [19]:

$$\min_x \ \max_{y \in y^{(0)}} \left(g(y) - p_z(y) \right)^2$$

In game theory, a game is defined as a triple (*Y; Z; k*) where *Y, Z*, denotes the spaces of strategies for player I and II, respectively and *k* is a real-valued pay-off function of $y \in Y$ and $z \in Z$. Under natural conditions, theoptimal strategies for both players solve the saddle point problem [19]:

$$\min_{z \in Z} \max_{y \in Y} k(y,z) = \max_{y \in Y} \min_{z \in Z} k(y,z)$$

In numerous engineering design problems, one is interested in minimizing the largest eigenvalue of an $n \times n$ symmetric matrix-valued function $A(y)$ of a variable y inR^n.

Thus, if $\lambda_i(y)$, $i = 1,\ldots, n$, is the i-th eigenvalue of $A(y)$ and by setting $f(I, n) = \lambda_I(y)$, then the following min-max problem is obtained [19]:

$$\min_{y \in Y^{(0)}} \ \max_{i=1,...,n} f(i,y)$$

Moreover, a nonlinear programming problem, with in equality constraints, of the form:

$$\min F(x), \quad \text{subject to} \quad g_i(x) \ge 0, i = 2,...,m \tag{6}$$

Can be transformed into the following minimax problem:

$$\min_x \ \max_{1 \le i \le m} f_i(x) \quad f_1(x) = F(x), \quad f_i(x) = F(x) - \alpha_i g_i(x), \alpha_i > 0, \tag{7}$$

For $2 \le i \le m$. It has been proved that for sufficiently large αi, the optimum point of the minimax problem, coincides with the optimum point of the nonlinear programming problem [15]-[18]. In addition to the above, numerous other applicationsinvolve solving minimax problems, justifying the ongoing interest for the

development of techniques that cancope efficiently with it. However, the nature of the minimax objective function $f(x)$ of Eq. (4), may pose difficulties in the process of solving minimax problems. Specifically, at points where $f_j(x) = f(x)$ for two or morevalues of $j \in \{1,\ldots,m\}$, the first partial derivatives of $f(x)$ are discontinuous, even if all the functions $f_i(x), i = 1,\ldots,m$, have continuous first partial derivatives.

This difficulty cannot be addressed directly by the well-known and widely used gradient-based methods, andseveral techniques have been proposed to cope with it [19]. Moreover, globally optimal solutions are frequently not only desirable but also indispensable.

II.1. Genetic Algorithm

Genetic algorithms (GA) were first introduced by John Holland in the 1970s [10], [12] as a result of investigations into the possibility of computer programs undergoing evolution in the Darwinian sense. GAs is part of a broader soft-computing paradigm known as evolutionary computation. The seat tempts to arrive at optimal solutions through a process similar to biological evolution ([22]-[29]).

This involves following the principles of 'survival of the fittest', cross breeding and mutation to generate better solutions from a pool of existing solutions. Genetic algorithms have been found to be capable of finding solutions for a wide variety of problems for which no acceptable algorithmic solutions exist. The GA methodology is particularly suited for optimization, a problem solving technique in which one or more very good solutions are searched for in a solution space consisting of a large number of possible solutions.

GAs reduce the search space by continually evaluating the current generation of candidate solutions, discarding the ones ranked as poor, and producing a new generation through cross breeding and mutating those ranked as good. The ranking of candidate solutions is done using some pre-determined measure of goodness or fitness.

A genetic algorithm is a probabilistic search technique that computationally simulates the process of biological evolution. It mimics evolution in nature by repeatedly altering a population of candidate solutions until an optimal solution is found. The GA evolutionary cycle, as shown in Fig. 6, starts with a randomly selected initial population.

The changes to the population occur through the processes of selection based on fitness, and alteration using crossover and mutation.

The application of selection and alteration leads to a population with a higher proportion of better solutions. The evolutionary cycle continues until an acceptable solution is found in the current generation of population, or some control parameter such as the number of generations is exceeded.

The smallest unit of a genetic algorithm is called a *gene*, which represents a unit of information in the problem domain.

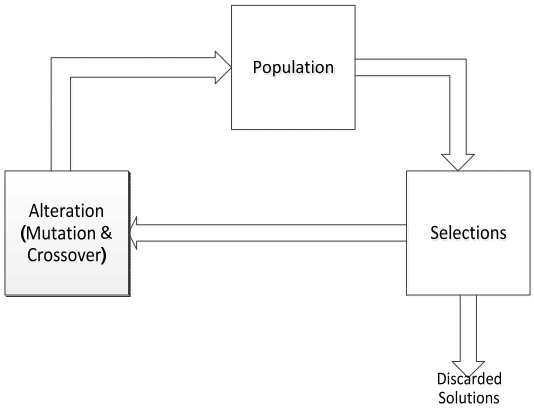


Fig. 6. Genetic algorithm evolutionary cycle

A series of genes, known as a *chromosome*, represents one possible solution to the problem. Each gene in the chromosome represents one component of the solution pattern. The most common form of representing a solution as a chromosome is by using a string of binary digits. Each bit in this string is a gene. The process of converting the solution from its original form into the bit string is known as *coding*. The specific coding scheme is used as an application dependent.

The solution bit strings are decoded to enable their evaluation using a fitness measure. In biological evolution, only the most fitted survives and their gene pool contributes to the creation of the next generation.

Selection in GA is also based on a similar process. In a common form of selection, known as *fitness proportional selection*, each chromosome's likelihood of being selected as a good one is proportional to its fitness value. The alteration step in the genetic algorithm refines the good solution from the current generation to produce the next generation of candidate solutions. It is carried out by performing crossover and mutation.

II.2. Crossover

Crossover may be regarded as artificial mating in which chromosomes from two individuals are combined to create the chromosome for the next generation. This is done as shown in Fig. 7, by splicing two chromosomes from two different solutions at a crossover point and swapping the spliced parts. The idea is that some genes with good characteristics from one chromosome may as a result combine with some good genes in the other chromosome to create a better solution represented by the new chromosome.

II.3. Mutation

Mutation is a random adjustment in the genetic composition. It is useful for introducing new characteristics in a population; something that cannot be achieved through crossover alone. Crossover only rearranges existing characteristics to give new combinations.

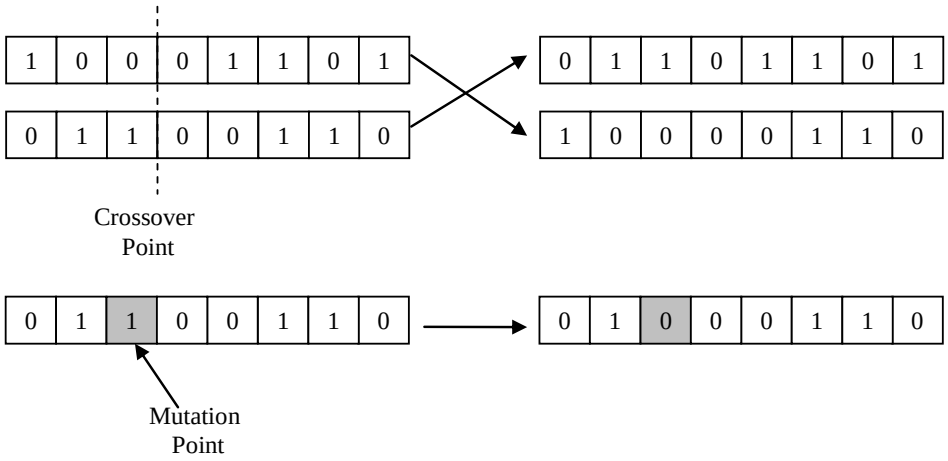


Fig. 7. Crossover and mutation in GA

For example, if the first bit in every chromosome of a generation happens to be a 1, any new chromosome created through crossover will also have 1 as the first bit.

The mutation operator changes the current value of a gene to a different one as shown in Fig. 7. For bit string chromosome this change amounts to flipping a 0 bit to a 1 or vice versa. Although useful for introducing new traits in the solution pool, mutations can be counterproductive, and are applied only infrequently and randomly.

The steps in the typical genetic algorithm for finding a solution to a problem are listed below:

1. Create an initial solution population of a certain size randomly.
2. Evaluate each solution in the current generation and assign it a fitness value.
3. Select "good" solutions based on fitness value and discard the rest.
4. If acceptable solution(s) found in the current generation or maximum number of generations is exceeded, then stop.
5. Alter the solution population using crossover and mutation to create a new generation of solutions.
6. Go to step 2.

In the literature P.E. Amiolemhen [2] used genetic algorithms for the determination of optimal machining parameters in the conversion of a cylindrical bar stock into a continuous finished profile. This work proposed an optimization technique based on genetic algorithms for the determination of the cutting parameters in multi-pass machining operations by simultaneously considering multi-pass roughing and single-pass finishing operations.

The optimum machining parameters are determined by minimizing the unit production cost of converting a cylindrical bar stock into a continuous finished profile involving seven machining operations; with each operation subject to many practical constraints.

The cutting model developed for each machining operation is a non-linear, constrained problem.

Experimental results show that the proposed technique is both effective and efficient. Ant colony optimization [3] (ACO) uses many ants (or agents) to traverse the solution space and find locally productive areas. While usually inferior to genetic algorithms and other forms of local search, it is able to produce results in problems where no global or up-to-date perspective can be obtained, and thus the other methods cannot be applied.

R.K. Sahoo [4] uses genetic algorithms for multi-component aromatic extraction. This work applies GAs which leads to globally optimal binary interaction parameters from multi-component liquid-liquid equilibrium data, has been recently demonstrated for some ternary, quaternary and quinary systems.

The binary interaction parameters are related to each other through the closure equations. In this work, the binary interaction parameters based on non-random two-liquid (NRTL) activity coefficient model have been estimated using GA, with and without closure equations for 65 multicomponent aromatic extraction systems: 53 ternary, 9 quaternary and 3 quinary systems. Parameters that satisfy the closure equations exhibit better root mean square (RMS) deviations than those that do not satisfy the closure equations. The RMS deviation value without implementation of closure equations is 0–80% better than in the literature as compared with0–90% with implementation of closure equations. Aromatics, such as benzene, toluene and xylene are considered essential in the chemical industry because they are the source of many organic chemicals.

These aromatics are presented in naphtha. High purity aromatics are difficult to separate using ordinary distillation operation, since they form several binary zoetrope's with non-aromatics. Extraction, therefore, is a better choice to separate the aromatics from naphtha, as they are preferentially soluble in a variety of solvents. To predict the separation, it is necessary to know the liquid–liquid equilibrium (LLE) data for a particular system. Various activity coefficient models, such as the universal quasi chemical (UNIQUAC) and the non-random two-liquid (NRTL) can be used to predict the LLE. Each of these models requires proper binary interaction parameters that can represent LLE for the highly non-ideal liquid mixtures usually encountered in aromatic extraction.

These parameters are usually estimated from the known experimental LLE data via optimization of a suitable objective function.

B. M. Kariuki [5] used genetic algorithms for solving crystal structures from powder diffraction data. This work used GA to tackle crystal structure solution from powder diffraction data in the case of a previously unknown structure-ortho-thematic acid.

In this structure solution calculation, the structural fragment was subjected to combined translation and rotation within the unit cell, together with variation of selected intermolecular degrees of freedom under the control of a genetic algorithm, in which a population of trail structures is allowed to evolve, subject to well defined procedures; formatting, mutation, and natural selection. Importantly, the genetic algorithm approach adopts the 'direct-space' philosophy for structure solution, and implicitly avoids the problematic step of extracting the intensities of individual reflections from the powder diffraction data. The structure solution was found efficiently in the genetic algorithm calculation, and was then used as the initial structural model in Riveted refinement calculations.

III. A multiple-Population Genetic Algorithm for Min-Max Problem

In genetic algorithms, the virtual individuals are tested against the problem represented as a fitness function. The better the fitness value, the more chance for the individual to be selected as a 'parent' for new individuals. The worst individuals are removed from the population in order to make room for the new generation. A basic genetic algorithm conducts a search by selecting candidate solutions from the population and using them to produce new candidate solutions.

The selection is random, but biased towards the most promising candidates as estimated by the cost function.

The size of the population is usually fixed and so as new candidates are produced, the least promising are discarded. This is called 'survival of the fittest'. A multi-population genetic algorithm [4] extends the basic genetic algorithm by including number of populations, in order to speed up the computation. In this paper, each population is evolved with its own cost function. This is done to direct the search in different populations to different regions of the input. The use of a multiple-population is a simple method of maintaining diversity in the set of inputs. If only a single-population is used, survival of the fittest can lead to the elimination of all individuals except those exploring the single lowest cost input region.

In general, a multi-population genetic algorithm may also allow individuals to migrate from one population to another. Migration is normally limited in order to maintain the differences between populations, but in this paper migration is unrestricted. There are two reasons for this: firstly, each population has its own cost function, which is the overriding determinant of which individuals remain in a population irrespective of the number of migrants from other populations.

For this reason, unrestricted migration does not lead to the loss of diversity that it might in other multi-population genetic algorithms. Secondly, it is efficient to re-use executed tests wherever possible, since the time required to execute the program under test is usually the most important factor that determines the speed with which test data is generated. Once an input has been executed and the cost function data collected, an evaluation of the input against any specific cost function can be produced relatively quickly.

The following proposed algorithm presents the main iterative procedure of the search method. For each min and max path, a population and associated cost function are created. The initial populations are constructed randomly and executing a number of inputs, then the algorithm is applied. The proposed algorithm operation is shown in Fig. 8a.

Algorithm Name: Min-max Optimization Problem using multi-population GA

Input: No. of generations, Mathematical function, and Function scope

Output: Numbers that achieves the biggest fitness value in each population.

1. For each Population i (pop_i) do in parallel, i=1,2
2. Initialize pop_i
3. For each individual j (Ind_j) in pop_i $i \in P(g)$, evaluate its fitness f_j
4. Sort $pop_i$by fitness
5. While termination condition not reached do
6. Select two parents from pop_i by Roulette- wheel selection
7. Create child solution using cross over
8. Apply crossover with a probability P_c
9. Apply mutation with a probability P_m to child solution
10. Replace child solution with the worst member of the pop_i
11. Sort pop_i i by fitness
12. End while

The best child (fitness) achieved in each population is a solution.

Fig. 8a. Multi-population Genetic Algorithms for Min-Max problem

IV. Experiment and Results

In this section, the specifications of the experimental environment utilized by this work are presented. These specifications include both the hardware and software modules used in implementing the simulator. More specifically, the hardware specifications include a Dual-Core Intel Processor (CPU 2.66 GHz), 2 MB L2 Cache per CPU, and 1 GB RAM. The software specifications include windows 7.

Further, the tested objects that have been used to evaluate our proposed multiple-population versus single-population algorithm are described in this section.

In order to assess the reliability of the proposed algorithm introduced in the previous section, an empirical investigation is presented. A number of benchmark problems were assembled.

The set of constrained benchmark functions used in this paper have been used by many researchers; we compare our proposed multiple-population versus single-population algorithm in test problems that are introduced in Runarsson and Yao, (2000), see Table I. Each functions formula is given and the characteristics of the corresponding problem are described.

Each of the cost functions and associated search operators were implemented in a prototype test data generation tool. The tool has been constructed by modifying the JScript (JavaScript) language compiler within the Shared Source Common Language Infrastructure (SSCLI), and can therefore be used to test functions within programs written in the JScript language. The program under test must include directives to specify any input domain constraints. The program is then parsed and semantically analyzed [14].

The tool then inserts instrumentation code at each branch in the function. This instrumentation code calculates the fitness of each path.Since it is not known which population will produce a solution, each population is evolved for only one input in turn before moving on to the next population. This can be done using a steady-state genetic algorithm such as Genitor [33]. Reproduction takesplace between two individuals who produce one or two offspring (depending on the choiceof reproduction operator).

These offspring are then evaluated and either inserted into theoriginal population, expelling the one or two least fit, or discarded if the offspring are the least fit. The population is kept sorted according to cost, and the probability of selectionfor reproduction is based on its rank in this ordering.

An important consideration is determining when a population is no longer evolving towards a solution. In this work, search progress in a single population is considered tohave stopped when a sequence of input cost values of a given length k (equals a positiveconstant s, set to 100 for this work) have been accumulated and in comparing each costwith the cost l, where $l \cdot k=2$, the majority of comparisons do not show a cost decrease.

Such a population is said to be stagnant. Stagnant populations are not evolved, but theirsearch goals are extended and used to evolve other populations. Whenever all populationsare stagnant and the maximum execution count have not yet been reached, then in order to continue searching for inputs, the stagnant status of all populations is cleared.

This is done by simply emptying the sequence of accumulated cost values. This ensures that aformerly stagnant population is evolved for at least k inputs before there is the possibilityof once again becoming stagnant. Note that, in this scheme, since the most effective populations will take longer to stagnate, they will take more computation time.

A search goal for a population is a set of max or min search goals. In this work, a fixed population of size 100 was used. This parameter was not *tuned* to suit any particular problem under test. In a steady-state updated style of geneticalgorithms (as used in this work); new individuals that are sufficientlyfit are insertedin the population as soon as they are created. Genetic Algorithms (GAs) generate inputs for the function containing the current structural target. A vector of floating point and integer variable values corresponding to the input data is optimized.

The ranges of each variable are specified.The test subject is then called with this input data. The criterion to stop the search was set up to terminate the search after 1,000 executions of the program under test. Individuals were recombined using binary and real-valued (one-point and uniform) recombination, and mutated using real-valued mutation.

Real-valued mutation was performed using *Gaussian distribution*and *number creep*.The performance of the multiple-path island population genetic algorithm searches forpathswas compared to the performance of a genetic algorithm using a single-population, pursuing paths in turn. The subject programs under test are used to compare the genetic algorithms.

Both genetic algorithms used the same overall populationsize, i.e. the size of the single-population was equal to the sum of the sizes of the islands, set at 50, as previously mentioned. This was set large enough to ensure that each islanddid not have too small a population.

For each trial, both genetic algorithms started with thesame set of initial random tests. Random inputs were generated with a random floating-point or integer number (depending on the problem parameters data types) generator producing numeric character sequences of varying lengths.

V. Results and Discussion

Experimental results of the number of executions of the problem under test (Table I) to find the max and min values (Tables II, III) averaged over 30 trials, using single- and multiple-population genetic algorithms.

The results according to Best, Mean, and Worst values are the same, using single- and multiple population.

TABLE I
PROBLEMS DESCRIPTIONS FROM [13].
(THE AMOUNT OF PARAMETERS, *N*, AND
THE AMOUNT OF INEQUALITY CONSTRAINTS, *M*,
OF EACH TEST PROBLEM G01 TO G13)

Function Name	*n*	*M*
G01	13	9
G02	20	2
G03	10	1
G04	5	6
G05	4	5
G06	2	2
G07	10	8
G08	2	2
G09	7	4
G10	8	6
G11	2	1
G12	3	729
G13	5	3

TABLE II
EXPERIMENTAL RESULTS ON THIRTEEN BENCHMARK FUNCTIONS (AVERAGED OVER 30 TRIALS) (MAX)

Function Name	Optimal	GA		
		Best	Mean	Worst
G01	5.00	5.00	5.00	5.00
G02	-0.70299	-0.703102	-0.775181	-0.803529
G03	-1.00	-1.00	-1.00	-1.00
G04	-23068.746	-23068.746	-23068.746	-23068.746
G05	6112.242	6112.226	6107.460	6000.321
G06	-1206.083	-1206.136	-1283.584	-1533.473
G07	4761.138	4761.320	4762.919	4763.325
G08	-0.095825	-0.095825	-0.095825	-0.095825
G09	10013748.521	10013747.810	10013704.318	10013480.071
G10	30000.0	30000.0	30000.0	30000.0
G11	1.000	1.000	1.000	1.000
G12	-1.000	-1.000	-1.000	-1.000
G13	21.6648	21.663703	20.789860	16.710397

TABLE III
EXPERIMENTAL RESULTS ON THIRTEEN BENCHMARK FUNCTIONS (AVERAGED OVER 30 TRIALS) (MIN)

Function Name	Optimal	GA		
		Best	Mean	Worst
G01	-15.000	-1500	-1500	-1500
G02	0	0	0	0
G03	0	0	0	0
G04	-30665.539	-30665.539	-30665.509	-30664.695
G05	5126.498	5126.498	5128.812	5141.625
G06	-6961.814	-6961.814	-6893.446	-6405.774
G07	24.306	24.315	24.375	24.578
G08	0.105460	0.105460	0.105460	0.105460
G09	680.630	680.630	680.656	680.784
G10	7049.331	7051.361	7631.016	9063.089
G11	0.750	0.750	0.750	0.751
G12	-0.48752	-0.484741	-0.484735	-0.484734
G13	0.053950	0.054032	0.085221	0.438940

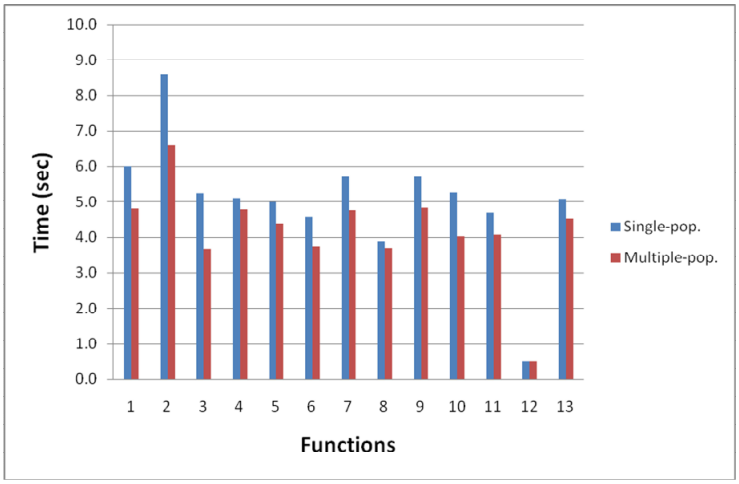


Fig. 8b. A comparison of Experimental results on thirteen benchmark functions (averaged over 30 trials) between single- and multiple-population genetic algorithms (Min & Max)

The execution time of the different functions (Table I) using the multiple-population genetic algorithm is less than using single-population, as shown in Fig. 8b, because the multiple-population algorithm requires less number of executions to find max and min but using single-population we need to run the program twice one to find the min and other time to find max.

For example, in Fig. 8b the execution time of G02 function is an average of about 6.57 seconds using multiple-population algorithm, whereas it is about 8.59 seconds using the single-population algorithm for the same program. However, using a multiple-population requires communication time between populations, because there is a migration between populations. This is done because re-using the existing tests is usually efficient since once a test is found to execute a given path, it need not be "rediscovered" if it is required for a later path. So, the execution time of the multiple-population algorithm (T_m) consists of computation time ($T_{cp.}$), which is searching time, plus communication time ($T_{cm.}$), as shown in Equation (8), whereas the execution time of the single-population (T_s) consists only of

computation time, which is searching time. Fig. 9 shows the computation and communication time of multiple-population for different programs. It is clear from this figure, that the multiple-population algorithm spends time on communication. For example, the G02function spends about 4 seconds on communication and about only 3 seconds on computation, as shown in Fig. 9:

$$T_m = T_{cp+}T_{cm} \tag{8}$$

The performance improvement (P_i) of using the multiple-population algorithm over the single-population algorithm under a different set of functions is presented in Fig. 10, where it is defined as the execution time of using the single-population algorithm (T_s) over the execution time of using the multiple-population algorithm (T_m), as shown in Eq. (9):

$$P_i = T_s / T_m \tag{9}$$

Fig. 10 shows that the multiple-population algorithm outperformed the single-population algorithm by about 1.2 times on average under different programs

It is clear that for the sample programs used in the experiments, the performance of the multiple-path island population genetic algorithms is significantly better than that of the single-population genetic algorithm in convergent speed of searching. More specifically, the use of multiple-population genetic algorithm outperforms the use of the single-population genetic algorithm in terms of execution time, performance improvement, cost, and search effectiveness.

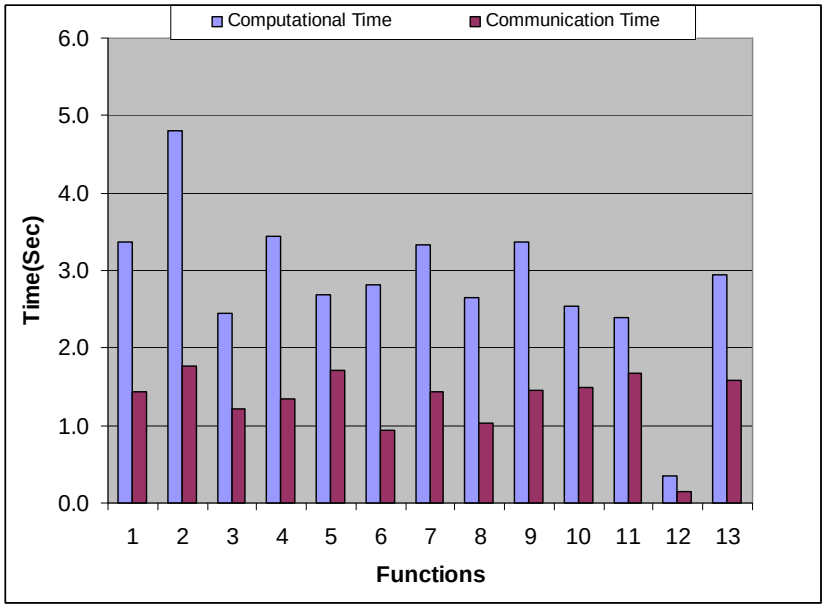


Fig. 9. Computation and communication time of multiple-population for different programs

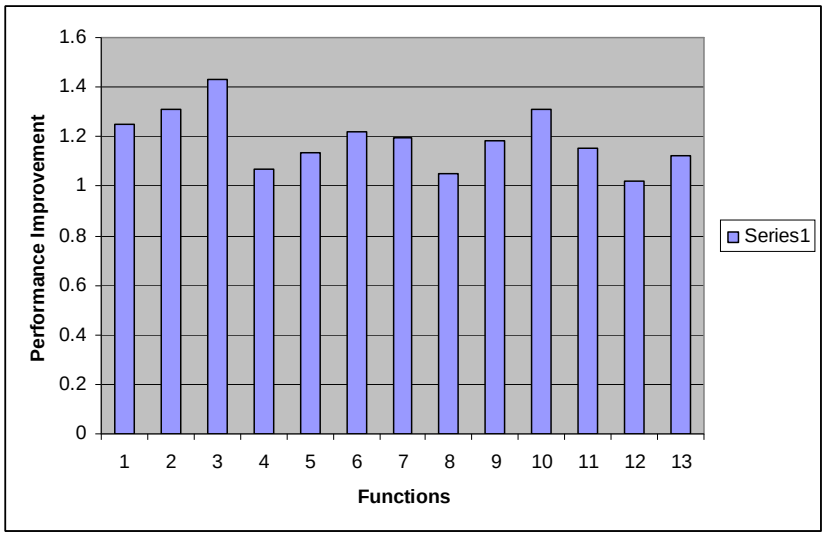


Fig. 10. Performance improvement of multiple-population over single-population for different programs

VI. Conclusion and Future Work

A single-population and a multiple-population genetic algorithm have been presented to find maximum and minimum values. A sample of collection functions has been used to carry out the evaluation. The results for these sample problems are clearly in favor of the multiple path island genetic algorithms.

The single and multiple- population have been compared and evaluated in terms of the following performance metrics under different functions: execution time, performance improvement, cost, and search effectiveness.

Furthermore, the experimental results show that the proposed multiple-population algorithm outperformed the single-population algorithm for the above performance metrics. For example, the multiple-population algorithm outperformed the single-population algorithm by about 1.2 times on average under different functions.

References

[1] T. Mantere, A Min-Max Genetic Algorithm with Alternating Multiple Sorting for Solving Constrained Problems, *http://www.stes.fi/* scai2006/proceedings/061-067.pdf, accessed August 2009.

[2] P. E. Amiolemhen, and A. O. A. Ibhadode, Application Of Genetic Algorithms- Determination Of The Optimal Machining Parameters In The Conversion Of A Cylindrical Bar Stock Into A Continuous Finished Profile, *International Journal of Machine tools & Manufacture*, Vol. 44, n. 1, pp. 1403 – 1412, 2004.

[3] B. Baudry, F. Fleurey, J. M. Jezequel, and Y. L. Traaon, Automatic Test case Optimization: A Bacteriological Algorithm, *Proceeding of 17 IEEE International Conferences on Automated Software Engineering*, pp. 76-82, 2005.

[4] R. K., Sahoo, T. Banerjee, S. A. Ahmad, and A. Khanna, Improved Binary Parameters using GA for Multi-Component Aromatic Extraction: NRTL Model Without and With Closure Equations, Journal of Fluid Phase Equilibria Vol. 239, n. 5, pp.107–119, 2006

[5] B. M.Kariuki, , H. S. González, R. L. Johnston, K. D. M. Harris, The Application of a Genetic Algorithm for Solving Crystal Structures from Powder Diffraction Data, Journal of Chemical Physics Letters, Vol. 280, pp. 189 – 195, 1997.

[6] Wikipedia," MinMax problem", http http://en.wikipedia.org/wiki/Minimax, accessed July2014.

[7] M. Alshraideh., B. Mahafzah, E. Salman, H. S., Salah I, Using Genetic Algorithm as Test Data Generator for Stored PL/SQL Program Units, *Journal of Software Engineering and Applications*, Vol. 6, n. 2, pp. 65-73, 2013.

[8] M. Alshraideh, B. Mahafzah, S. Al-Sharaeh , A Multiple-Population Genetic Algorithm for Branch Coverage Test Data Generation, *Software Quality Control*, Vol. 19, n. 3, pp. 489-513, 2011,

[9] M. Alshraideh , L. Bottaci, Using Program Data-State Diversity in Test Data Search, *Proceedings of the Testing: Academic & Industrial Conference on Practice And Research Techniques*, pp.107-114, 2006.

[10] J. Holland, Adaptation in Natural and ArtificialSystems. *The MIT Press, Cambridge*, MA, 1992.

[11] T. P. Runarsson, Y. Stochastic, ranking for constrained evolutionary optimization, *IEEE Transactions on Evolutionary Computation, Vol. 4, n. 3, pp.*284-294, 2000.

[12] E. Mezure-montes, simple multimembered evolution strategy to solve constrained optimization problems. *IEEE Transactions on Evolutionary Computation, Vol.* **9**: 1- 17, 2005.

[13] J. Morovic, Y. Wang, Influence of test image choiceon experimental results, *In Proceedings of 11thColor Imaging Conference, Scottsdale*, pp. 143-148, 2003.

[14] M. Alshraideh, A Complete Automation of Unit Testing for JavaScript Programs, *Journal of Computer Science, V*ol. 4, n.12, pp. 1012--1019, 2008.

[15] J.W. Bandler, C. Charalambous, Nonlinear Programming Using Minimax Techniques , *Journal of Optimization Theory and Applications*, Vol. 13, pp.607-619, 1974.

[16] C. Charalambous, A.R. Conn, An Efficient Method to Solve the Minimax Problem Directly, *SIAM Journal on Numerical Analysis.* Vol. 15, pp. 162-187, 1978.

[17] D.Z. Du, P.M. Pardalos, Minimax and Applications, Kluwer: Dordrecht, 1995.

[18] S. Zuhe, A. Neumaier, M.C. Eiermann, Solving Minimax Problems by Interval Methods, *BIT Numerical Mathematics*, Vol. 30, pp. 742-751, 1990.

[19] Maxima and Minima of Functions of Two Variables, "http://math.oregonstate.edu/home/programs/undergrad/Calculus QuestStudyGuides/vcalc/min_max/min_max.html ", [on line 10/10/2014].

[20] Mary Gladence, L., Ravi, T., Mining the change of customer behavior in fuzzy time-interval sequential patterns with aid of Similarity Computation Index (SCI) and Genetic Algorithm (GA), (2013) *International Review on Computers and Software (IRECOS)*, 8 (11), pp. 2552-2561.

[21] Moustafa, A.A., Alqadi, Z.A., Alduari, M., Alomar, S., Practical approach to genetic algorithm cryptanalysis, (2009) *International Review on Computers and Software (IRECOS)*, 4 (6), pp. 658-663.

[22] Kunaraj, K., Seshasayanan, R., Constrained Cartesian Genetic Programming - A New Paradigm for Evolving Imprecise Multipliers, (2014) *International Journal on Numerical and Analytical Methods in Engineering (IRENA)*, 2(1), pp. 5-8.

[23] Abdelhakem-Koridak, L., Rahli, M., Benayed, F., Genetic Optimization for Combined Heat and Power Dispatch, (2014) *International Journal on Engineering Applications (IREA)*, 2(5), pp. 163-168.

[24] Abdul Jaleel, J., Rekhasree, R.L., A comparative study on AGC of power systems using reinforcement learning and genetic algorithm, (2013) *International Review of Automatic Control (IREACO)*, 6 (4), pp. 404-409.

[25] Hypiusova, M., Kajan, S., Robust controller design using edge theorem and genetic algorithm, (2013) *International Review of Automatic Control (IREACO)*, 6 (2), pp. 194-200.

[26] Tarim, N., İyibakanlar, G., Beamforming the Antenna Arrays in the Localizer Unit of Instrument Landing System by Using Genetic Algorithm, (2013) *International Review of Aerospace Engineering (IREASE)*, 6(3), pp. 179-186.

[27] Omar, H.M., Developing geno-fuzzy controller for satellite stabilization with gravity gradient, (2014) *International Review of Aerospace Engineering (IREASE)*, 7 (1), pp. 8-16.

[28] Bouslama-Bouabdallah, S., Tagina, M., A fault detection and isolation fuzzy system optimized by genetic algorithms and simulated annealing, (2010) *International Review on Modelling and Simulations (IREMOS)*, 3 (2), pp. 212-219.

[29] Rezaie Estabragh, M., Mohammadian, M., Rashidinejad, M., An application of elitist-based genetic algorithm for SVC placement considering voltage stability, (2010) *International Review on Modelling and Simulations (IREMOS)*, 3 (5), pp. 938-947.

Authors' information

[1]Computer Science Department, University of Jordan, Amman, Jordan.

[2]Management Information system Department, Gulf University for Science and Technology, Kuwait.

Mohammad Aref Alshraideh is an Associate Professor of Computer Science at the University of Jordan, Jordan. He received his B.Sc. degree in Computer Science in 1988 from Mu'tah University in Jordan and a Master degree in Computer Science in 2000 from University of Jordan. He obtained his Ph.D. degree in Computer Science from University of Hull, UK,

in 2007. During his graduate studies he obtained a fellowship from the University of Jordan. He was a Head Director Assistant for Computer Technology at the Hospital of the University of Jordan until June 2012. Also he was working as Human Resource Director at the University of Jordan until 2015. Dr. Alshraideh is currently is working as Registrar General at the University of Jordan. His research interests include Software Testing, Artificial Intelligence, and Data Mining.

Luay Tahat is an assistant professor in the Management Information System and Computer Sceince Department at Gulf University for Sceince and technology since 2008. Prior to that, He was the lead Mobile Network Solution Architect at Alcatel-Lucent in Naperville, USA. He has a master's degree in computer science from Northeastern Illinois University in Chicago and a Ph.D. in computer science from the Illinois Institute of Technology (IIT), also in Chicago. In his time at Alcatel-Lucent, Dr. Tahat has held several positions in software development, system engineering, and system architecture and has contributed to several areas in the fields of software engineering. His research interests include software testing, software maintenance, and wireless network solutions. The results of his research were published in several Journals and conference proceedings.