UMGAtoolbox1.0 User's Guide

Created by: Andrew Goupee, M.S. Candidate
University of Maine
Department of Mechanical Engineering
Boardman Hall, Room 228
Orono, Maine 04469

Table of Contents

1. Intro to Genetic Algorithms (GAs) and UMGAtoolbox1.0
2. GA Working Principles
   2.1. Real-Coded GA Representation
   2.2. Reproduction with Tournament Selection and Niching
   2.3. Constraint Handling with GAs
   2.4. Crossover with Simulated Binary Crossover
   2.5. Mutation with Parameter Based Mutation
3. Using UMGAtoolbox1.0
   3.1. GA.m m-file; the Input Page
   3.2. GA Parameter Descriptions and Recommended Values
      3.2.1. max_gen and n_pop parameters
      3.2.2. n_genes, ub_1, lb_1, ub_2 and lb_2 parameters
      3.2.3. elite and best parameters
      3.2.4. pc, pcg and nc parameters
      3.2.5. pm, pmg and nm parameters
      3.2.6. d_nich and nf_f parameters
      3.2.7. drop and dyn parameters
      3.2.8. tolerance and span parameters
      3.2.9. grad_switch parameter
      3.2.10. plot_switch parameter
   3.3. GA Objective and Constraint Function Inputs
      3.3.1. Objective Function Construction
      3.3.2. Constraint Function Construction
   3.4. Running the UMGAtoolbox1.0 from Matlab®
4. References

1. Introduction to GAs and UMGAtoolbox1.0

The genetic algorithm (GA) was first conceived by John Holland, who outlined its basic working principles in his 1975 work *Adaptation in natural and artificial systems* (Holland, 1975). In short, a GA is an optimization method which mimics the chromosomal processing of natural genetics to constitute a search and optimization technique. Since the introduction of the GA, Holland's students, as well as numerous other individuals interested in this unique and powerful optimization tool, have continued to refine the method. Engineers have taken a liking to the method for its ability to find optimal solutions to very challenging problems, problems for which standard gradient based methods (The Mathworks, 2000) perform poorly. Problems that fall under this umbrella are ones in which there are a large number of parameters to be optimized, problems with numerous local minima (multi-modal) and problems with highly nonlinear constraints, just to name a few.

The UMGAtoolbox1.0 is a general purpose implementation of a GA for use in the Matlab® programming environment. This toolbox will work on a wide variety of problems, and will work especially well on those types of problems for which a GA excels. To use this program, the user must only input the values of various parameters on the GA.m m-file page, construct simple objective and constraint functions, and then run the program. Much greater detail on the aforementioned steps is given in this guide; however, after reviewing this manual (which is highly encouraged, especially if one does not want garbage!), use of toolbox should be very easy.

2. GA Working Principles

In this section, the fundamentals of the GA process will be outlined, and then, elaborated upon. For more extensive explanations on GA fundamentals, please see (Goldberg, 1989; Deb, 1998; Deb, 2001) or check out other readily available sources which cover GAs in the engineering literature.

As is stated previously in this manual, a GA borrows heavily from the ideas of chromosomal processing in natural genetics. Many biologists believe the main driving force behind natural evolution is Darwin's so called 'survival of the fittest' principle. In short, this principle states that if above-average offspring are created, they will have a greater chance of survival. On the other hand, if a sub-average offspring is created, its chances of survival are slim.

A GA implements this idea in the sense of a numerical optimization problem. A GA starts out with a set of randomly generated solutions, each solution being called an individual. The group of individuals is called a population. Each individual in the population possesses one chromosome (unlike humans which possess 46!). An individual's chromosome consists of genes, and for a real-coded GA as used here in the UMGAtoolbox1.0 implementation, there is one gene for each design parameter to be optimized. For instance, if we have 3 design variables, each individual's chromosome will contain 3 genes. These genes contain actual values of the parameters to be optimized. These parameter values can be used in the

evaluation of our objective function we wish to minimize, and from the resulting value, we can prescribe a fitness to the individual corresponding to how good the solution is.

With a population of individuals in hand, we wish to improve the fitness of the population and more importantly, find more fit individuals (which in turn means finding better solutions). To do this, a few simple operators are applied to the population to create a new population with hopefully better solutions. The first operator is called the reproduction or selection operator. This operator carries out Darwin's 'survival of the fittest' principle in a sense. It creates an intermediate population called a mating pool which contains more copies of the good solutions in the population and fewer copies of the bad solutions in the population. With a mating pool created, two more operators are applied to create a new population. These are called the crossover operator and the mutation operator. These mimic the basic mechanics of chromosomal processing to create new individuals, which in the sense of an optimization routine, explores our search space in hopes of creating better solutions. The process is repeated, creating new generation after new generation until some predetermined termination criteria is met. A flowchart of the overall process is shown on the following page in Fig. 1. Following the figure, more detailed explanations of the concepts touched upon here are given.

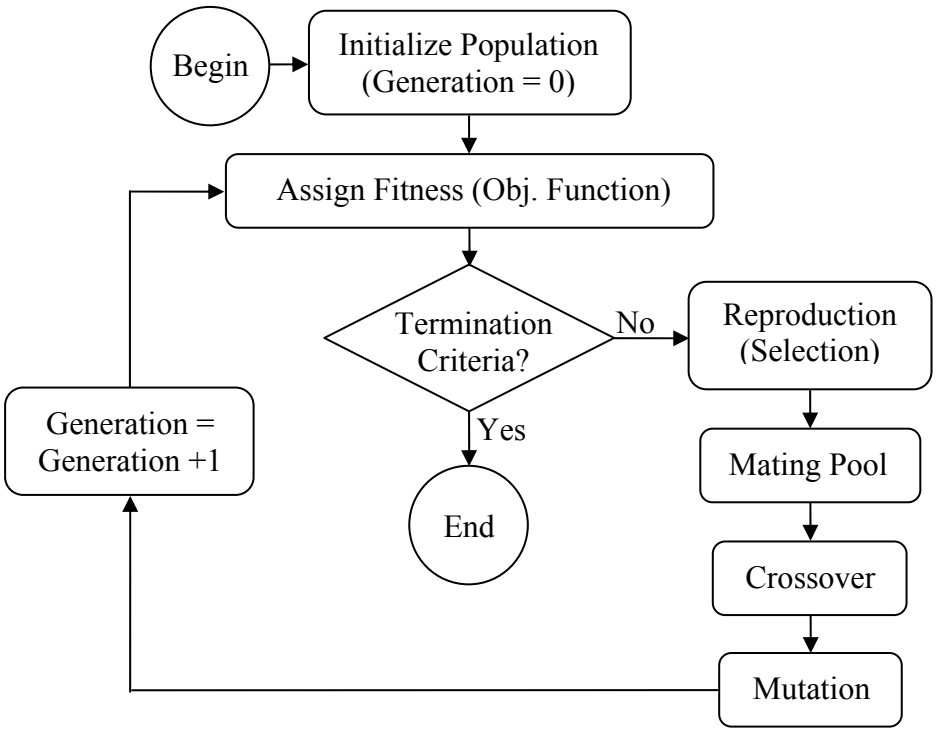

Figure 1: GA process flowchart

2.1. Real-Coded GA Representation

In this section the subject of GA representation is discussed. In short, representation is the means of representing a possible solution as an individual that the GA operators (reproduction, crossover and mutation) can work upon. Each individual possesses one chromosome. That chromosome is essentially a vector with one entry for every parameter to be optimized. In essence, if the objective function to be minimized as four parameters, an individual's chromosome may look like the following:

$$chromosome = [0.3456 \quad 1.2347 \quad 4.0010 \quad 0.0023]$$

As it is shown, the chromosome above is in a form that can be manipulated by the crossover and mutation operators. For the reproduction or selection operator, the chromosome itself is not really needed. What is needed is the fitness

of the individual. To calculate the fitness, the chromosome vector is inserted into the outlined objective and constraint functions and fitness is calculated. For instance, if the objective function is f(**x**) and the constraint function is g(**x**), where **x** is a vector of length n, then we set **x** equal to the chromosome vector (also of length n) and evaluate the functions. Determining the fitness from these objective and constraint function values is discussed in detail in section 2.3. Constraint Handling with GAs.

2.2. Reproduction with Tournament Selecting and Niching

The reproduction or selection operator is a fairly simple operator. Once completed, the reproduction operator creates a mating pool, essentially an intermediate population from which the next generation's population is created. The UMGAtoolbox1.0 utilizes a method known as tournament selection (Deb, 2001; Goldberg and Deb, 1991) to carry out this process. In its simplest form, two individuals from a population are selected at random to compete in a tournament. The fitnesses of the two individuals are compared, and the individual with the most desirable fitness is declared the winner. A copy of the winner is placed in the mating pool and the two individuals which competed in the tournament are placed back in the old population. The process is repeated until all the slots in the mating pool are filled. In other words, if a population has n individuals, then this process is repeated n times to create a mating pool with n individuals.

While the basic tournament selection procedure is widely used and considered the best reproduction operator currently available, the addition of a little scheme during this process can facilitate faster convergence of the GA to the optimal solution. This scheme is called niching (Deb, 2000) and is intended to maintain diversity in the population. By maintaining some semblance of diversity amongst individuals in a population, the likelihood of honing in on a suboptimal solution (local minima) is greatly reduced. The technique of niching is implemented in the following fashion. During a tournament for a spot in the mating pool, a first individual is selected at random as usual. This is individual is called individual i. Next, a second individual, individual j, is selected at random. A normalized Euclidean distance between individuals i and j, d_{ij}, is then calculated as follows:

$$d_{ij} = \sqrt{\frac{1}{n}\sum_{k=1}^{n}\left(\frac{x_k^{(i)} - x_k^{(j)}}{x_k^u - x_k^l}\right)^2}$$

where n is the number of optimization parameters (genes), $x_k^{(i)}$ and $x_k^{(j)}$ are the k^{th} gene from individuals i and j respectively and x_k^u and x_k^l are the upper and lower bounds on the k^{th} optimization parameter (gene). If this normalized Euclidean distance (which can range from 0 to 1) is less than a certain threshold value, then the individuals i and j are compared with their fitnesses and a copy of the winner is inserted into the mating pool. If the normalized Euclidean distance is greater than the threshold value, individual j is placed back into the old population and a new individual j is selected at random. The d_{ij} value is recalculated and checked against the threshold value. If the value is too large, a new

individual j is obtained as just mentioned. This process will be repeated until an acceptable individual j is found or until a maximum number of attempts has been met. If the maximum number of attempts to find a suitable individual j has been reached, and yet no suitable one is found, then the individual i is declared the winner and a copy of it is placed in the mating pool.

2.3. Constraint Handling with GAs

Often in realistic engineering problems, one wishes to find an optimal solution **x** subject to a number of constraints. This can be summarized as follows:

$$\begin{aligned} &\text{Minimize} && f(\underline{x}), \\ &\text{Subject to} && g_j(\underline{x}) \le 0, \quad j = 1,...,J, \\ & && x_i^l \le x_i \le x_i^u \quad i = 1,...,n, \end{aligned}$$

where f(**x**) is the objective function, **x** is a vector with n entries, $g_j(\mathbf{x})$ is the j^{th} constraint (of which there are J total) and x_i^l and x_i^u are the respective lower and upper bounds for the i^{th} element of **x**, x_i. Please note that the constraints on the upper and lower bounds on the design parameters x_i (genes) are automatically enforced with crossover and mutation operators utilized in the UMGAtoolbox1.0. See sections 2.4. and 2.5. for more details. The constraints in $g_j(\mathbf{x})$ pose the difficulty in performing a constrained optimization with a GA.

There a couple of ways in which to solve constrained optimization problems with GAs. One of the most efficient and effective methods is a constraint handling method created by Deb (Deb, 2000). This is the method which the UMGAtoolbox1.0 implements for constraint handling. The

principle of the method relies on the use of a tournament selection scheme for the reproduction process, which is used in UMGAtoolbox1.0. During a tournament between two individuals of which a copy of the winner enters the mating pool, the following rules are adhered too:

1. Among two individuals which possess feasible solutions (all constraints are met), the individual with the best objective function value (the smallest value, since the search is for a global minimum) is declared the winner.
2. Among two individuals each which possess infeasible solutions, the individual with the smallest constraint violation is declared the winner.
3. Among an individual with a feasible solution and an individual with an infeasible solution, the individual with the feasible solution is declared the winner.

To ensure that the aforementioned rules are followed by merely comparing the fitnesses of two individuals, the fitness $F(\mathbf{x})$ of an individual is constructed in the following manner:

$$F(\underline{x}) = \begin{cases} f(\underline{x}) & \text{if } g_j(\mathbf{x}) \leq 0 \text{ for all } j = 1,2,\ldots,J, \\ f_{\max} + \sum_{j=1}^{J} \langle g_j(\underline{x}) \rangle & \text{otherwise,} \end{cases}$$

where f_{max} is the objective function value of the individual with the worst feasible solution, $\langle g_j(\mathbf{x}) \rangle$ is the value of the constraint violation $g_j(\mathbf{x})$ if is greater than zero and is zero otherwise and the remaining quantities are as described previously in the section. Please note that a lower fitness value is more desirable due to the fact we are performing a minimization process. In other words, an individual with a lower fitness value is deemed more 'fit'

and will be the victor in tournaments with other individuals possessing higher fitness values.

2.4. Crossover with Simulated Binary Crossover

Once a mating pool has been created, the next step is to apply the crossover operator to the chromosomes of the individuals in the mating pool. The crossover operator is the main search mechanism in a GA. The crossover operator is easy to explain and implement in a binary-coded GA (Holland, 1975; Goldberg, 1989; Deb, 2001), but the UMGAtoolbox1.0 does not use this type of coding. Instead, as has been eluded to earlier, this toolbox uses a real-coded GA for its robustness and efficiency. This coding makes the implementation of the crossover a bit messier, but nonetheless, it is still not too difficult to follow.

UMGAtoolbox1.0 uses the Simulated Binary Crossover (SBX) operator, a crossover operator created by Deb and his students (Deb and Agrawal, 1995). With the mating pool established, two individuals are systematically pulled from the mating pool. It is then checked if crossover is to occur between the two individuals or not. The crossover operator is applied with a preselected probability. If that probability is say, 50%, then the crossover operator will be applied, on average, to only half the individual pairs pulled from the mating pool. If crossover does not occur, then the two individuals are placed back into the mating pool, unaltered. If crossover is to occur, then the SBX operator is applied.

The SBX operator creates two new individuals, called children, from the two individuals taken from the mating pool, called parents. Each parent's chromosome has n genes, and so will the children's chromosome. The SBX operator acts directly on the parent genes to create new children genes. However, the SBX operator is not applied to all n genes of the parents to create all new, distinctly different children genes. The SBX operator is applied to each gene in turn with a preselected probability. A common practice is to apply the SBX operator to a pair of parent genes with a 50% probability. This means that on average, the children will retain approximately half of its parents genes unchanged while the rest are different from its parents. Once a pair of parent genes $x^{(1)}$ and $x^{(2)}$ are set to undergo the SBX operation, one proceeds with the following process to create children genes $y^{(1)}$ and $y^{(2)}$. Please note that x^l and x^u are the lower and upper bounds on the gene in question, it is assumed that $x^{(1)}$ is less than $x^{(2)}$ and that n_c is the SBX strength parameter (usually on the order of 1 to 10):

1. Generate a random number u between 1 and 0
2. Calculate beta: $\beta = 1 + \frac{2}{x^{(2)} - x^{(1)}} \min\left[\left(x^{(1)} - x^l\right)\left(x^u - x^{(2)}\right)\right]$
3. Calculate alpha: $\alpha = 2 - \beta^{-(n_c+1)}$
4. Calculate beta bar: $\overline{\beta} = \begin{cases} (\alpha u)^{1/(n_c+1)} & \text{If } u \le 1/\alpha, \\ \left(\frac{1}{2-\alpha u}\right)^{1/(n_c+1)} & \text{otherwise} \end{cases}$
5. Calculate $y^{(1)}$, $y^{(2)}$: $y^{(1)} = 0.5\left[\left(x^{(1)} + x^{(2)}\right) - \overline{\beta}\left|x^{(2)} - x^{(1)}\right|\right]$, $y^{(2)} = 0.5\left[\left(x^{(1)} + x^{(2)}\right) + \overline{\beta}\left|x^{(2)} - x^{(1)}\right|\right]$

Once the children are complete, they may be placed back in the mating pool, or, the best of the parent/children group

may go back in the mating pool, depending on what the GA user has chosen. Once the overall crossover operation has been applied to the mating pool, the next step is to apply the mutation operator to create a new population of individuals for the next generation.

2.5. Mutation with Parameter Based Mutation

Once crossover has been applied to the mating pool, the last step to creating a new population for the next generation is the application of the mutation operator. The purpose of the mutation parameter, similar to the niching technique, is to maintain diversity in the population. Unlike the niching technique, however, the mutation operator also explores new directions in the search space that the crossover operator cannot. Usually the mutation operator is applied sparingly to the mating pool, usually with a probability of around 10%. In other words, each member of the mating pool is selected in turn and chances of mutation occurring are on the order of around 1 in 10 (this can vary depending on user input). If mutation is not to occur, then the individual is placed back into the mating pool, unaltered. If mutation is to occur, then the parameter based mutation (PBM) operator (Deb and Goyal, 1996) is applied to create a mutated individual which is put in mating pool in the stead of the original individual.

Similar to the SBX operator, the PBM operator functions directly on the genes within the chromosome of an individual. Therefore, if an individual has multiple genes, the PBM operator can be applied to each gene in the

chromosome. However, usually the PBM operator is applied to the genes with a probability of around 50%. This means that on average roughly half of the genes in an individual will become mutated, and the other half will remain unchanged. Once a gene is to be mutated, the PBM sequence of operations is applied to create a mutated gene y from the original gene x. Noting that n_m is the mutation strength parameter (usually on the order of 10 to 100) and that x^l and x^u are the respective lower and upper bounds on the gene being operated on, the following steps are taken to create the gene y:

1. Generate a random number u between 1 and 0
2. Calculate delta: $\delta = \frac{\min\left[\left(x - x^l\right), \left(x^u - x\right)\right]}{x^u - x^l}$
3. Calculate delta bar:

$$\bar{\delta} = \begin{cases} \left[2u + (1-2u)(1-\delta)^{n_m+1}\right]^{1/(n_m+1)} - 1 & \text{If } u \le 0.5, \\ 1 - \left[2(1-u) + 2(u-0.5)(1-\delta)^{n_m+1}\right]^{1/(n_m+1)} & \text{otherwise} \end{cases}$$

4. Calculate y: $y = x + \bar{\delta}\left(x^u - x^l\right)$

Once an individual singled out for mutation has had the PBM operator applied to selected genes, the mutated individual is complete and is placed back in the mating pool.

3. Using UMGAtoolbox1.0

This section of the User's Guide should prove to be the most useful to the user. While sections 1. and 2. provide a brief introduction to GAs as well as shed light on the inner workings of a GA, this section will outline the steps required to utilize this particular GA optimization package. To use this package effectively, it will be

useful to have at least a rudimentary grasp of how to program in Matlab®. Also, the user must have a function that is to be minimized. If not, manipulate the output of the function so that by minimizing the output, the function is maximized.

The first step to using UMGAtoolbox1.0 is to locate the directory where all the UMGAtoolbox1.0 m-files are. All of the m-files must be in the folder, and all but the GA.m m-file should be left alone. Other files the user creates and places in this directory are free to be manipulated in any way one pleases. With the directory located, open up Matlab® and make this the working directory. Once this has been accomplished, one is ready to select the inputs for GA as well as construct the objective and constraint functions.

3.1. GA.m m-file; the Input Page

To access the GA.m m-file, from Matlab® select File, then Open…, then click on the GA.m m-file. This will open the m-file in a new window. If one wishes, one can read through the commentary shown in green. However, to get at the inputs, scroll down to the section labeled 'Select GA parameters:'. Below this line are listed all the input parameters for the UMGAtoolbox1.0. The following section of this guide, section 3.2., explains what aspects of the GA these parameters control and also gives recommended values for the parameters. When altering this page, please be careful not to eliminate any of the parameters or the toolbox will not function properly.

3.2. GA Parameter Descriptions and Recommended Values

This section of the user's guide will outline the influence of the input parameters on the GA.m m-file page. This section will also give allowable ranges on these input variables where applicable as well as give recommended values which work well on a wide variety of problems.

3.2.1. max_gen and n_pop parameters

max_gen – The max_gen parameter is the maximum allowable number of generations that the GA will run. The GA may terminate in fewer generations outlined in this parameter, but the GA will not continue past the number of generations prescribed in max_gen.
Range: 1 and up.
Recommended Values: At least 100 or so for moderately complex problems.

n_pop – The n_pop parameter prescribes the size of the GA population. Please note that this number must be even!
Range: Any multiple of 2.
Recommended Values: The population size should be at least 10 times the number of design variables, preferably more.

3.2.2 n_genes, ub_1, lb_1, ub_2 and lb_2 parameters

n_genes – The n_genes parameter is set to the number of design parameters in your problem. For example, if the minimum of a function with respect to three variables is to be found, the n_genes is set to 3.
Range: 1 and up.

ub_1 – The ub_1 parameter is a row vector which contains the upper bounds for each design variable in each entry. This row vector must be of size 1 by n_genes. The ub_1 vector applies to only the initial population of individuals. For example, if the function to be minimized is of three variables, ub_1 may look like the following:

$$ub\_1 = \begin{bmatrix} 1.0 & 2.1 & 1.456 \end{bmatrix}$$

This would imply that the upper bound on the first, second and third variables of the initial population are 1.0, 2.1 and 1.456 respectively.

Range: Any row vector with n_genes entries.

lb_1 – The lb_1 parameter is a row vector which contains the lower bounds for each design variable in each entry. This row vector must be of size 1 by n_genes. The lb_1 vector applies to only the initial population of individuals. For example, if the function to be minimized is of three variables, lb_1 may look like the following:

$$lb\_1 = \begin{bmatrix} 0.45 & 0.9 & 1.456 \end{bmatrix}$$

This would imply that the lower bound on the first, second and third variables of the initial population are 0.45, 0.9 and 1.456 respectively.

NOTE: If desired for some reason, the lower and upper bounds on a particular design variable can be the same value.

Range: Any row vector with n_genes entries.

ub_2 – The ub_2 parameter is structured just like the ub_1 vector. However, the ub_2 vector outlines the upper bounds on the design variables for all generations after the initial generation. This is done so that the initial

population can be isolated in a certain portion of the search space if desired.

lb_2 – The lb_2 parameter is structured just like the lb_1 vector. However, the lb_2 vector outlines the lower bounds on the design variables for all generations after the initial generation. This is done so that the initial population can be isolated in a certain portion of the search space if desired.

3.2.3. elite and best parameters

elite – The elite parameter is essentially a switch that turns on and off an elitist mechanism. If the elite parameter is set equal to 1, then the best individual in the current generation automatically gets two copies of itself in the next generation. However, if the elite parameter is set to 0, the elitist mechanism is turned off and the previous scenario does not occur.
Range: 1 (on) or 0 (off).
Recommended Value: 1

best – The best parameter is another switch that alters the characteristic of the GA search. In this GA, two parents give rise to two children through crossover, and the children are also mutated at this time if required. If the best parameter is set to 1, then the two best individuals of the parent/children group are placed in the next generation. If the best parameter is set to 2, then the children are automatically chosen and are placed into the next generation.
Range: 1 (best of group) or 2 (children).

Recommended Value: 1 if a high selection pressure is required, 2 if preservation of population diversity is important.

3.2.4 pc, pcg and nc parameters

pc – The pc parameter dictates the probability with which a pair of parents is going to undergo crossover. For example, if pc is set to 0.9, then there is a 90% chance that a pair of parents will undergo crossover.
Range: 0.0 to 1.0
Recommended Value: 0.9

pcg – The pcg parameter dictates the probability with which a pair of parent genes is going to undergo the SBX crossover operation (assuming that the parents have been selected to undergo the overall crossover operation). For example, if a pair of parents each with three genes is to undergo crossover, and the pcg parameter is set to 0.5, then each of the three genes will undergo the SBX operation with a 50% probability. This is used so that if one wishes, not all the genes of the parents are altered. This allows the preservation of some information which may be needed to move towards the global minimum.
Range: 0.0 to 1.0
Recommended Value: 0.5

nc – The nc parameter is essentially a crossover strength parameter. The nc parameter dictates how far children genes may stray from the parent genes during crossover operations. If nc is a large value, then the children genes resulting from crossover will be close to the parent genes. If the nc parameter is a small value, then the

children genes resulting from crossover may stray far from the parent genes.
Range: Any positive value.
Recommended Value: On the order of 2.0 to 10.0

3.2.5. pm, pmg and nm parameters

pm – The pm parameter dictates the probability with which an individual will undergo mutation. For example, if pm is set to 0.2, then there is a 20% chance that an individual will undergo the mutation process.
Range: 0.0 to 1.0
Recommended Value: 0.1 to 0.3

pmg – The pmg parameter dictates the probability with which a particular gene of an individual is to undergo the PBM operations to create a new, mutated gene. For example, if an individual with three genes is to be mutated and pmg is set to 0.5, then there is a 50% chance for each of the three genes that they will undergo the PBM mutation operation. This is done so that if it is desired, some of an individual's genes are preserved for the purpose of potentially holding on to information that may lead to a global minimum later on.
Range: 0.0 to 1.0
Recommended Value: 0.5

nm – The nm parameter is effectively a mutation strength parameter. The nm parameter dictates how far away a mutated gene can be created from a non-mutated gene. If the nm parameter is a large value, then the mutated gene will remain close to the original gene. However, if the nm

value is small, then the mutated gene is allowed to stray far from the original gene value.
Range: Any positive value.
Recommended value: 10.0 to 100.0

3.2.6. d_nich and nf_f parameters

d_nich – The d_nich parameter is used in the niching process. The d_nich value is the maximum allowable normalized Euclidian distance allowed between two individuals before they can be singled out to compete in a tournament for a spot in the mating pool. If one does not wish niching to have an effect, set the d_nich value to 1.0 (this is the upper limit on any calculated normalized Euclidian distance).
Range: 0.0 to 1.0
Recommended value: 0.1

nf_f – The nf_f parameter sets the maximum number of individuals to be checked in the niching process. Once the first individual to compete in a tournament has been selected, a certain percentage of the population will be searched for a second individual to compete in the tournament for which the normalized Euclidian distance between the two is smaller than the value set in d_nich (if no such individual is found, the first individual is declared the winner). This percentage is set in nf_f. Therefore, if nf_f is 0.25, up to 25% of the population will be searched in hopes of finding a suitable individual to compete in the tournament.
Range: 0.0 to 1.0
Recommended value: 0.25

3.2.7. drop and dyn parameters

drop, dyn – The drop and dyn parameters work together to dynamically alter the following aforementioned GA parameters: pc, pcg, nc, pm, pmg and nm. This is done with the intention of allowing the GA to search freely in the early generations and then gradually move to a more concentrated and focus search in the later generations. The pc, pcg, pm and pmg are altered as the generations progress with the following relation:

$$current\_parameter = initial\_parameter\left[1 - drop\left(1 - e^{-dyn \times generation\#}\right)\right]$$

Therefore, if drop and dyn are positive quantities (which they usually are), the values of pc, pcg, pm and pmg gradually decrease with time and eventually approach an approximate value of (1 – drop) times the initial parameter value. The parameters nc and nm, however, are altered as the generations progress according the following relation:

$$current\_parameter = initial\_parameter\left[1 + drop\left(1 - e^{-dyn \times generation\#}\right)\right]$$

This indicates that with positive values for drop and dyn, the parameters nc and nm increase to an approximate value of (1 + drop) times the initial parameter value as the generations progress. If one does not wish to dynamically alter these parameters, set either drop equal to 0 or dyn equal to 0.

drop range: 0.0 to 1.0

drop recommended value: 0.0 to 0.5

dyn range: Any positive value.

dyn recommended value: 0.02 and lower.

3.2.8. tolerance and span parameters

tolerance, span – The tolerance and span parameters are parameters utilized in a slightly more sophisticated method

of GA termination. In short, if the maximum number of generations has not been reached, and the fitness of the best individual has not changed by more than the value allotted in tolerance over a number of generations as specified in the parameter span, the GA will terminate.
tolerance range: Any positive value.
tolerance recommended value: 0.00001 times a typical fitness value or less
span range: 1 and up.
span recommended value: 10

3.2.9. grad_switch parameter

grad_switch – The grad_switch parameter enables one to transition into a gradient based optimization if so desired. If grad_switch is set to 1, then this option is turned on. What this means is that once the GA terminates (either by reaching the maximum number of generations or meeting the criteria set by tolerance and span), the best solution to date is used as an initial guess in a gradient based search. The gradient based search is the fmincon routine found in the Matlab® optimization toolbox (the optimization toolbox must be installed on your computer for this function to work). The hope is that a gradient based search will be better suited to honing in on a good solution once the GA has located the general vicinity of the global minimum. For more information on the fmincon routine, plase see Matlab®'s Optimization Toolbox User's Guide (The Mathworks, 2000). To disable this function, set grad_switch equal to 0.
range: 1 (on) or 0 (off).

3.2.10. plot_switch parameter

plot_switch – The plot_switch parameter turns on and off a plotting function in the GA. If plot_switch is set to 1, then the plotting function in UMGAtoolbox1.0 is turned on. This means, that at the end of the GA, plots of the best individual fitness and the population average fitness are plotted as a function of generation. Also, plots of the dynamically altered parameters pm, pmg, nm, pc, pcg and nc are also given as a function of generation. If plot_switch is set to 0, then the plotting function is turned off and no plots are given once the GA terminates.
range: 1 (on) or 0 (off).

3.3. GA Objective and Constraint Function Inputs

This section of the user's guide will briefly discuss how to input ones objective and constraint functions into the GA.m m-file input page as well as later on, how to construct these objective and constraint function m-files. First, the input requirements on the GA.m m-file will be discussed. With the GA.m m-file page open, scroll down to the portion of the file with the green heading '%Provide objective and constraint function names'. This section is right below the GA parameter section. Below this heading are two variables, one called objective and the other called constraint. Set the first variable, objective, equal to the name of the objective function m-file you have constructed (more on how to do this later). Omit the .m file extension. The provided name should be surrounded by quotes. If done correctly, the name and surrounding quotes will be a reddish maroon. The second variable, constraint, should be set equal to the name of the constraint function

m-file you have constructed (more on how to do this later). Once again, omit the .m file extension. The name of the constraint function should be surrounded by quotes and should automatically turn reddish maroon if done correctly. Now, the construction of these two functions will be elaborated upon.

3.3.1. Objective Function Construction

First, it would be in the interest of the user to have a decent grasp of how to write functions in Matlab®. That aside, it should also be stressed that any objective function to be used with this toolbox must be one that is to be minimized. If one wishes to maximize a certain function, alter the way the function is constructed so that by finding a minimum, the answer one seeks is found. Following is a template for constructing the objective function:

```
function [output] = function_name(x)

…Additional user's code for calculating scalar variable
'output' from the input vector (x)…

output = …
```

The previous little blip is all that is required for constructing an objective function. It should be noted that the function_name can be any name one wishes. This name is also the name that the file should be saved under (in this case, this function would be saved as function_name.m). This is also the name used on the GA.m m-file input page. Please note that the variable output

(which can also assume any name the user wishes) must be a scalar. Also note that input variable x (which once again, can take on any name one wishes), is a row vector with the number of entries outlined by the input parameter n_genes.

3.3.2. Constraint Function Construction

Construction of the constraint function is similar to the construction of the objective function. Once again, familiarity with Matlab® programming is helpful. The constraint function, similar to the objective function, takes in as input a vector and places as output a scalar. The constraint function should return a positive value proportional to the level of constraint violation if a constraint(s) is violated, and should return the value zero if the constraint(s) is met. The following is a template for constructing the constraint function:

```
function [c,ceq] = function_name(x)

ceq = [];

…Additional user's code for calculating the scalar
constraint violation value c from the input vector x
(Remember, c should be a positive value proportional to the
level of constraint violation, and should be zero if the
constraints are met)…

c = …
```

The previous template is all that is really required to create a constraint function. Some simple programming logic will likely need to be used. Please leave the names

of the scalar outputs, c and ceq, as they are. Please also always set ceq = []; (This is used in the event of a gradient based search). The name of the function, function_name, may be whatever the user wishes. This name must also be the name by which the file is saved (in this instance, function_name.m) and must also be the name provided in the GA.m m-file.

3.4. Running the UMGAtoolbox1.0 from Matlab®

Once the directory has been located and made the working directory, and once the GA.m m-file parameters are set with appropriately constructed objective and constraint functions in the same directory, then running the toolbox is quite simple. In the Matlab® command window, type the following:

GA

And then hit enter. The program will commence and will provide a generation by generation report in the command window containing the following: generation no., average fitness, and best individual fitness, constraint violation and genes. If one wishes to terminate the program early, just hit the following keys:

Ctrl + Alt + Delete

The Ctrl Alt Delete sequence will stop the program.

4. References

Deb, K. (1998). *Genetic Algorithms in Search and Optimization: The Technique and Applications*. Proceedings of International Workshop on Softy computing and Intelligent Systems, Calcutta, India: Machine Intelligence Unit, Indian Statistical Institute. (pp 58–87).

Deb, K. (2000). *An Efficient Constraint Handling Method for Genetic Algorithms*. Computer Methods in Applied Mechanics and Engineering 186(2-4), 311-338.

Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. Chichester, West Sussex: John Wiley & Sons, LTD.

Deb, K. and Agrawal, R. B. (1995). *Simulated Binary Crossover for Continuous Search Space*. Complex Systems 9(2), 115-148.

Deb, K. and Goyal, M. (1996). *A Combined Genetic Adaptive Search (GeneAS) for Engineering Design*. Computer Science and Informatics 26(4), 30-45.

Goldberg, D. E. (1989). *Genetic Algorithms for Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley

Goldberg, D. E. and Deb, K. (1991). *A Comparison of Selection Schemes used in Genetic Algorithms*. In Foundations of Genetic Algorithms 1 (FOGA-1). (pp 69-93).

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: MIT Press.

The Mathworks (2000). *Optimization Toolbox For Use with Matlab®*. Natick, MA: The Mathworks.