工学ジャーナル　第3巻
2004年3月
一般投稿論文

MATLAB Implementation of a Genetic Algorithm for Linearly Constrained Optimization Problems

Alcigeimes Batista CELESTE*, Koichi SUZUKI* and Akihiro KADOTA*

A genetic algorithm (GA) is implemented in MATLAB and its features and applications are presented. The GA can minimize functions subject to linear inequality constraints and uses real-valued chromosomes, stochastic universal sampling, elitism, fitness ranking, and various genetic operators. Several experiments are carried out to show the performance of the genetic algorithm and compare it with calculus-based techniques. Applications to engineering problems are also illustrated. The results show that the program is efficient and robust for solving optimization problems.

Keywords: genetic algorithms, constrained optimization, MATLAB

1 Introduction

The availability of algorithms for solving optimization problems is very large and the development of new techniques is constantly increasing. Most of the methods are though appropriate only for certain types of problems in which several assumptions regarding the objective function, such as continuity and convexity, must be taken into consideration. Moreover, many good procedures can only find local rather than global solutions. In real-world problems, particularly in engineering applications, the functions are generally highly multimodal and discontinuous and the traditional approaches might fail sometimes.

Because of the limitations of the conventional calculus-based techniques there has been an increase of popularity of genetic algorithms, search procedures based on the phenomenon of natural selection. They have shown to be capable to deal successfully with optimization problems which are difficult to be solved by other methods (see [1-4]).

This paper deals with the development and implementation of a genetic algorithm for solving optimization problems with linear inequality constraints. The algorithm is constructed in MATLAB and its performance is compared with traditional techniques by means of various test cases. The program has been successfully applied to many engineering applications and showed to be a useful tool.

The paper consists of five sections including this introduction. Next section presents all the features of the genetic algorithm and shows how it deals with linearly constrained optimization problems. Section 3 concerns the implementation of the algorithm in MATLAB environment. Analyses from the utilization of the GA to solve a set of test problems are illustrated in section 4, which also exemplifies its use for solving some engineering applications. Section 5 concludes the work.

2 Description of the Genetic Algorithm

GAs mimic Darwin's evolution process by implementing a "survival of the fittest" strategy. In principle, the search starts with a set of initial random solutions called *population*. Individuals called *chromosomes* compose the population. Each chromosome represents a potential solution for the problem and is described by a string of symbols. Every solution is evaluated to provide some measure of *fitness*. A new population is then formed by selecting the more fit individuals. Some members of this new population are submitted to transformations by means of genetic operators (*crossover* and *mutation*) to form new solutions, called *offspring*. After successive iterations, or *generations*, the algorithm converges to the best chromosome, which hopefully represents the optimum or suboptimal solution to the problem. In general, the more fit individuals tend to reproduce and thus improve the successive generations. However, inferior individuals may happen to survive and also reproduce [2, 5, 6]. The general structure of a genetic algorithm is displayed below:

general genetic algorithm
begin
    Gen ← 1　　{*Gen*: generation number}
    initialize population *P*(*Gen*)
    evaluate *P*(*Gen*)
    while (**not** *termination-condition*) **do**

*Department of Civil and Environmental Engineering, Faculty of Engineering, Ehime University
原稿受理　平成15年10月30日

```
        Gen ← Gen + 1
        select P(Gen) from P(Gen-1)
        recombine P(Gen)
        evaluate P(Gen)
    endwhile
end
```

Goldberg [4] pointed out the following ways in which GAs differ from normal optimization and search procedures: GAs work with coding of the variable set, not the variables themselves; GAs search from a population of points, not an individual point; GAs use objective function information, not derivatives or other auxiliary knowledge; GAs use probabilistic rather than deterministic transition rules.

The genetic algorithm must have the following main components which are discussed in section 2.1 to section 2.4: chromosome representation; selection procedure; genetic operators; creation of an initial population; evaluation function; termination criterion. Section 2.5 explains how the constraints of the optimization problem are handled.

2.1 Chromosome Representation

Each chromosome in the population is represented (or *coded*) by a string of parameters called *genes*, from a certain alphabet. An alphabet can consist of binary digits, floating point numbers, integers, symbols, etc. Holland [7] first showed, and many still defend the idea that a binary alphabet should be used for the string [8, 9]. However, it has been shown that floating point coding is faster and provides higher precision than binary implementation (see [2]). The genetic algorithm developed in this work uses floating point representation for the chromosomes.

2.2 Selection Procedure

The selection procedure is used to choose individuals (*parents*) from the current population for undergoing *recombination* (crossover and mutation) and generate offspring to the next generation. Individuals are selected in a random way that favors the more fit chromosomes, i.e., the points that have the best values of fitness (objective function).

Roulette Wheel (also called *Stochastic Sampling with Replacement*) was the first selection method and is apparently the most popular procedure. The selection probability p_k for each chromosome k is proportional to the fitness value f_k:

$$p_k = f_k \Big/ \sum_{j=1}^{PopSize} f_j$$

in which *PopSize* is the population size. The wheel has a form of a pie graph of which the area of each zone is proportional to the probability of its representative chromosome. The wheel is spun *PopSize* times, and at each time a single chromosome is selected for the *mating poll*, a population composed by the individuals that will go through recombination.

We have used another selection technique called *Stochastic Universal Sampling* (SUS). Proposed by Baker [10], SUS is a method in which a number of equally spaced pointers equal to the population size are put around the "pie" and after a *single* wheel spin the chromosome pointed by each marker is selected (Figure 1). This approach is better than roulette wheel because it keeps the diversity and prevents *super chromosomes* from dominating the population [5].

2.3 Genetic Operators

Genetic operators are used to recombine parents after they are selected from the population. Crossover and mutation are the two operators used to create the offspring. Crossover operators combine information from two parents to form two offspring such that the *children* possess characteristics from each parent. Mutation operators tend to make small random changes in one parent to form one child in attempt to explore all regions of the state space [6].

There are several kinds of crossover and mutation operators. Their application depends on the chromosome representation used. Michalewicz [2] developed a number of operators for use with chromosomes represented by floating point values. These operators were also used in our system and are explained below:

- ***Uniform mutation***: randomly selects one variable k and sets it equal to a uniform random number from the range (L_k, U_k), the lower and upper bound, respectively, for variable k:

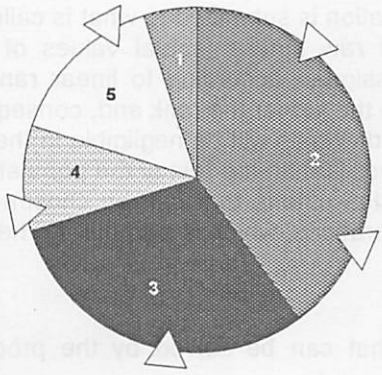


Figure 1. Stochastic Universal Sampling. The selected parents are: 2 2 3 4 5.

Parent: $\mathbf{x} = (x_1, \ldots, x_k, \ldots, x_n)$; *Offspring*: $\mathbf{x}' = (x_1, \ldots, x_k', \ldots, x_n)$; $x_k' \in (L_k, U_k)$

where n is the number of variables.

- ***Boundary mutation***: a variation of uniform mutation that selects a random variable and sets it equal to either L_k or U_k:

Offspring: $\mathbf{x}' = (x_1, \ldots, x_k', \ldots, x_n)$; $x_k' = \begin{cases} L_k & \text{if a random binary number is 0} \\ U_k & \text{if a random binary number is 1} \end{cases}$

- ***Non-uniform mutation***: randomly selects one variable and sets it equal to a non-uniform random number:

Offspring: $\mathbf{x}' = (x_1, \ldots, x_k', \ldots, x_n)$; $x_k' = \begin{cases} x_k + \psi(Gen, L_k - x_k) & \text{if a random binary number is 0} \\ x_k - \psi(Gen, x_k - U_k) & \text{if a random binary number is 1} \end{cases}$

where $\psi(Gen, y) = y \cdot r \cdot (1 - Gen / MaxGen)^p$, *Gen* is the generation number, r is a random number from (0,1), *MaxGen* is the maximum number of generations and p is a system parameter determining the degree of non-uniformity.

- ***Arithmetic crossover***: produces two complementary linear combinations of the parents:

$\mathbf{x}_1' = a \cdot \mathbf{x}_1 + (1-a) \cdot \mathbf{x}_2$; $\mathbf{x}_2' = (1-a) \cdot \mathbf{x}_1 + a \cdot \mathbf{x}_2$

where $a \in (0,1)$.

- ***Simple crossover***: The "head" (first sequence of values up to a certain position randomly chosen) of one individual is connected to the "tail" (remaining sequence after the crossover position) of the other and vice-versa to produce two new offspring:

Parent 1: $\mathbf{x}_1 = (x_1, \ldots, x_n)$; *Parent 2:* $\mathbf{x}_2 = (y_1, \ldots, y_n)$

Offspring 1: $\mathbf{x}_1' = (x_1, \ldots, x_k, y_{k+1}, \ldots, y_n)$; *Offspring 2:* $\mathbf{x}_2' = (y_1, \ldots, y_k, x_{k+1}, \ldots, x_n)$

- ***Heuristic crossover***: produces a linear extrapolation of two parents utilizing fitness information:

$\mathbf{x}_1' = \mathbf{x}_1 + r \cdot (\mathbf{x}_1 - \mathbf{x}_2)$; $\mathbf{x}_2' = \mathbf{x}_1$

where r is a random number between 0 and 1, and $\mathbf{x}_1$ is better than $\mathbf{x}_2$ in terms of fitness. If $\mathbf{x}_1'$ is unfeasible, another value r is generated and another offspring created. If after a given number of attempts unfeasibility still remains, the children equal the parents.

2.4 Initial Population, Evaluation Function and Termination Criterion

The initial population needed to start the GA is, in general, randomly generated but it can also be seeded with potentially good solutions. In our system, the user can instruct the program to create an initial random population or seed it with a single starting point or even a set of points.

The evaluation function used to determine the fitness of each chromosome in the population is, in principle, the objective function of the optimization problem. However, after the function is evaluated

for all the chromosomes, the population is submitted to what is called *fitness ranking*, a process where individuals are sorted in order of *raw* fitness (actual values of the objective function), and then *reproductive* fitness values are assigned according to linear ranking. In case of minimization, the smaller the objective function value the bigger the rank and, consequently, the fitness. Because of this, the effect of one or two extreme individuals will be negligible in the selection process, independent of how much greater or less their fitness is than the rest of the population (see [8, 9]).

There exist several termination criteria that can be chosen for GAs. The most used one is a specified maximum number of generations, which is also utilized here.

2.5 Handling the Constraints

The optimization problem that can be solved by the procedure built in this study has the following general form:

$$\textit{minimize } f(\mathbf{x}) \tag{1}$$

$$\mathbf{x} \in \Re^n$$

$$\textit{subject to } A\mathbf{x} + \mathbf{b} \le 0\,;\ \mathbf{l} \le \mathbf{x} \le \mathbf{u} \tag{2}$$

in which f is the objective function that returns a scalar value (f: $\Re^n \rightarrow \Re$); A (m x n) is a matrix and $\mathbf{b}$ (m x 1) is a vector, where m is the number of constraints; and $\mathbf{l}$ and $\mathbf{u}$ are the lower and upper boundary vectors of variable $\mathbf{x}$, respectively.

When the set of constraints is linear, the offspring generated by the operators described in section 2.3 are always feasible, considering (L_k,U_k) to be the feasible range of a variable k where other variables remain fixed (see details in [2]). L_k and U_k are hence not necessarily equal to l_k and u_k, respectively (see Figure 2). This feature, needed by the mutation operators, is sufficient to handle the constraints of the problem above.

3 MATLAB Implementation

The genetic algorithm detailed in the previous section has been implemented in MATLAB, a high-performance language for technical computing that integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation and the basic element is an array that does not need to be dimensioned. MATLAB has been selected because the program can be constructed in an easier and more elegant way than in other languages and, moreover, it contains several functions that help the formulation of the procedure and make the numerical computation efficient.

The program is composed by a set of twelve functions. The main function is named *fgenec* and founded on the general structure presented in section 2. *fgenec* is basically organized as follows:

fgenec
begin
    Gen ← 1 {*Gen*: generation number}

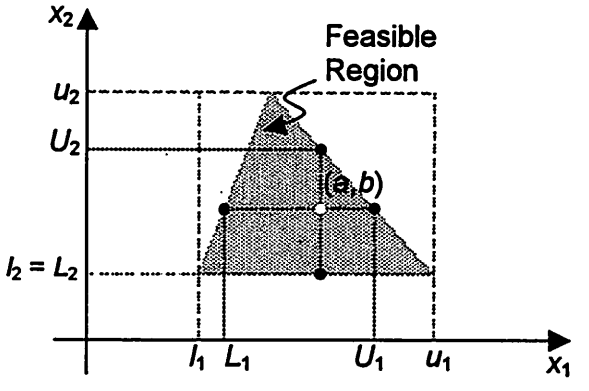


Figure 2. Example of determination of the feasible range for a given point $(x_1,x_2) = (a,b)$ considering a problem with two decision variables x_1 and x_2. (L_1,U_1) is the range inside where x_1 can vary from a setting b to be fixed. Similarly, (L_2,U_2) is the range inside where x_2 can vary from b fixing a.

StartPop ← **initialize** population using function *startp*
OldPop ← *StartPop*
evaluate *OldPop*
OldBest ← best point from *OldPop*
while (*Gen* < *MaxGen*) **do** {*MaxGen*: maximum number of generations}
Gen ← *Gen* + 1
NewPop ← **generate** new population from *OldPop* using function *generate*
evaluate *NewPop*
NewWorst ← worst point from *NewPop*
NewBest ← best point form *NewPop*
NewWorst ← *OldBest* {elitism}
OldPop ← *NewPop*
OldBest ← *NewBest*
endwhile
end

First, a starting population is created (in case the user does not provide one) by means of subfunction *startp*. Then the population is evaluated using the objective function and its best point is identified. While the number of generations is less than its maximum defined value, a new population (*NewPop*) is generated using subfunction *generate*. The best and worst points of the new population are detected and the worst one is replaced with the best chromosome from the old population (*OldPop*). This procedure is called *elitism*, which avoids the best chromosome to be lost from one generation to another by crossover or mutation (see [8, 9]). When the loop ends, the remaining population will be the optimal population of points and the optimal points of the decision variables are chosen as the points with best fitness or minimum value of objective function.

The MATLAB command for calling *fgenec* is given below:

[*Xopt*, *FinalPop*] = *fgenec*(*FUN*,*Bounds*,*G*,*Options*,*StartPop*,*P1*,*P2*,*P3*,*P4*,*P5*,*P6*,*P7*,*P8*,*P9*,*P10*)

where:

- *Xopt* is the best solution found;
- *FinalPop* is the final population;
- *FUN* is the evaluation function. The function should return a scalar value;
- *Bounds* is the matrix of boundaries for the variables (*Bounds* = **[l u]**);
- *G* is the matrix of constraints (linear inequalities). *G* is formed by joining matrix *A* and vector **b** from the set of constraints (2): $G = [A \quad \mathbf{b}]$. If the problem is unconstrained, the user can set *G* equal to the null matrix;
- *Options* is the vector of options equal to [*PopSize MaxGen Display Trace*], in which: *PopSize* is the population size (default: 100); *MaxGen* is the maximum number of generations (default: 100); *Display* if 1 shows progress (value of objective function for each generation), 0 otherwise (default: 0); *Trace* if 1 shows performance (graph of generation versus objective function); 0 otherwise (default: 0);
- *StartPop* is an optional starting population that can be seeded to the algorithm. If not defined one will be chosen randomly;
- *P1*, *P2*,..., *P10* are optional arguments which are passed to the objective function, *FUN*(*X*,*P1*,*P2*,...,*P10*), if necessary.

The subfunction *generate* creates a new population from the old one and consists of the functions for selection and recombination:

generate
begin
OldPop ← *OldPop* **ranked** by using function *ranking*
MatingPoll ← population of individuals for recombination **selected** from *OldPop* by using function *select*
form single *G* by combining *G* and *Bounds*
NewPop ← *MatingPoll* after **recombination** by subfunctions *unifmut*, *bounmut*, *nonunmut*, *arithx*, *simplex* and *heurix*
end

Fitness ranking is done by subfunction *ranking*. Function *select* applies stochastic universal sampling (section 2.2) to select individuals for the mating poll. Recombination is carried out by the genetic operators described in section 2.3 using functions *unifmut*, *bounmut*, *nonunmut*, *arithx*,

simplex and *heurix*. Functions *unifmut*, *bounmut* and *nonunmut* use an auxiliary function, *feasib*, that returns the feasible range formed by the values (L_k, U_k) of each variable k where other variables remain fixed, which are required by the mutation operators (see section 2.3).

4 Test Cases and Results

The performance of the genetic algorithm was verified by applying it to a set of three test cases. These test problems were also solved using the MATLAB *Optimization Toolbox*, a collection of functions specially devised for mathematical optimization, and comparisons were carried out. Functions *fminunc* and *fmincon*, which find the minimum of unconstrained and constrained functions, respectively, were selected according to the test case. *fminunc* uses the BFGS Quasi-Newton method with a mixed quadratic and cubic line search procedure. *fmincon* uses a *Sequential Quadratic Programming* (SQP) method. In this procedure, the Hessian of the Lagrangian function is approximated at each iteration using a quasi-Newton updating method. A *Quadratic Programming* (QP) subproblem is then generated and its solution is used to form a search direction for a line search procedure. A full description of these functions can be found in the toolbox manual [11].

The first test case is based on the general problem configuration of section 2.5 and composed by a nonlinear function subject to linear inequality constraints. The second problem is unconstrained and contains a highly multimodal function which is minimized for four different situations depending on its dimension. The last case has a set of several linear *equality* constraints which are used by two objective functions (one unimodal and the other multimodal).

4.1 Test Case #1

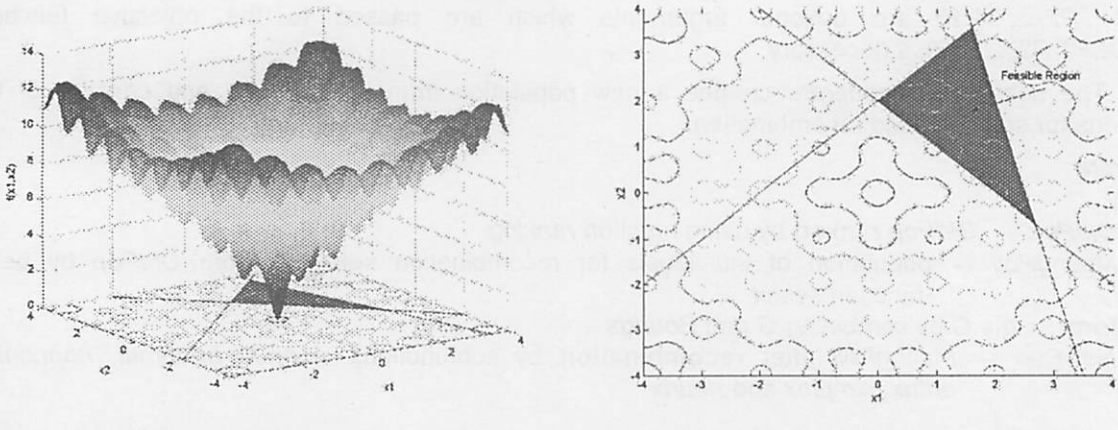
The first problem is to find the minimum value for the two-dimensional Ackley's Path [12], a highly multimodal function subject to a set of three linear inequalities and domain constraints:

$$\textit{minimize}\ f(x_1,x_2)=\left\{\begin{array}{l}-20\exp\left[-0.2\left(\dfrac{1}{2(x_1^2+x_2^2)}\right)^{1/2}\right]\\ -\exp\left\{\dfrac{1}{2[\cos(2\pi x_1)+\cos(2\pi x_2)]}\right\}+20+\exp(1)\end{array}\right\}$$

subject to

$$-x_1-x_2+2\le 0$$
$$-x_1+x_2-2\le 0$$
$$4x_1+x_2-10\le 0$$
$$-4\le x_i\le 4;\quad i=1,2$$

The minimum for the unconstrained Ackley's Path is located at position $(x_1^*, x_2^*) = (0, 0)$ where $f(x_1^*, x_2^*) = 0$. However, the constraints above form a triangular region inside where the optimal point has to be placed (Figure 3). Thus, the optimal solution for the constrained problem is $(x_1^*, x_2^*) = (1, 1)$



(a) (b)

Figure 3. (a) Two-dimensional Ackley's Path and (b) representation of the constraints and feasible region.

and $f(x_1^*, x_2^*) = 3.63$.

Starting from point (1.9, 1.9) *fmincon* found the "optimum" at $(x_1^*, x_2^*) = (1.97, 1.97)$ with $f(x_1^*, x_2^*) = 6.56$. *fgenec* was run ten times with a starting population of 100 points equal to (1.9, 1.9) and, for all runs, found the optimum at position $(x_1^*, x_2^*) = (1, 1)$. Figure 4 shows the performance of the genetic algorithm for a typical run of 200 generations.

4.2 Test Case #2

This is an unconstrained problem that uses the general test function from [13]:

$$\textit{minimize } f(\mathbf{x}) = s\sum_{i-1}^{n}(x_i - x_i^*)^2 + \sum_{k=1}^{k_{max}} a_k \sin^2[f_k P_k(\mathbf{x} - \mathbf{x}^*)]$$

in which $\mathbf{x}^* = (x_1^*, \ldots, x_n^*)$ is the global solution; $s > 0$ and $a_k > 0$ are scalars and f_k are integers (which can be changed in order to increase or decrease the level of difficulty); and $P_k(\cdot)$ are polynomials that vanish at the zero vector. We have solved the problem for n varying from 1 to 4, $s = 0.025n$, $a_k = f_k = 1$ (for all k, and $k_{max} = 2$), and lower and upper bounds equal to -5 and 5, respectively. The polynomials used were the following:

$$P_1(\mathbf{x} - \mathbf{x}^*) = \sum_{i=1}^{n}(x_i - x_i^*) + \sum_{i=1}^{n}(x_i - x_i^*)^2 \; ; \; P_2(\mathbf{x} - \mathbf{x}^*) = \sum_{i=1}^{n}(x_i - x_i^*)$$

Figure 5 shows the graph of the objective function for the two-dimensional case for an optimal point of (2.5, 2.5). The value 2.5 was assumed for all elements of vector $\mathbf{x}^*$ for $n = 1, \ldots, 4$.

For test case #2 the MATLAB function *fminunc* was used. *fgenec* was performed setting matrix G to be null (see section 3). *fminunc* needs an initial starting point $\mathbf{x}_0$ which was randomly determined

Table I. Results for test case #2.

n	$\mathbf{x}_0$	Theoretical $\mathbf{x}^*$	*fminunc*		*fgenec*		
			$\mathbf{x}^*$	$f(\mathbf{x}^*)$	*PopSize*	$\mathbf{x}^*$	$f(\mathbf{x}^*)$
1	-2.6886	2.5	-3.8978	1.0369	100	2.5000	0.0000
2	-0.4353 -4.8150	2.5 2.5	5.3105 -0.2855	0.7841	500	2.5004 2.4996	1.3713e-08
3	-3.0128 1.0379 -2.2781	2.5 2.5 2.5	-2.7409 -0.0372 -2.2507	4.2383	2000	2.5009 2.4920 2.5069	8.5194e-06
4	-3.0119 -4.8473 2.4679 -0.5490	2.5 2.5 2.5 2.5	-3.0086 -4.9250 2.7128 -0.4371	9.4198	2000	2.4965 2.5040 2.4958 2.5036	5.8647e-06

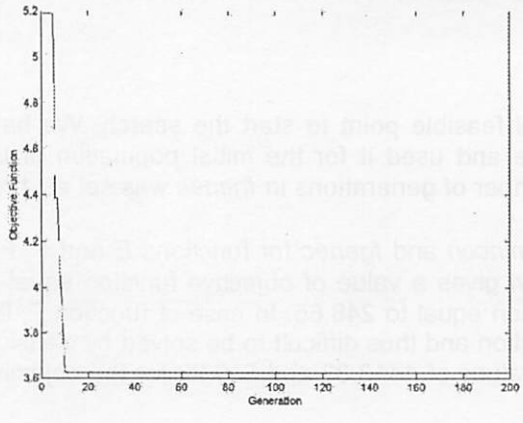


Figure 4. Performance of *fgenec* in a typical run for solving test case #1.

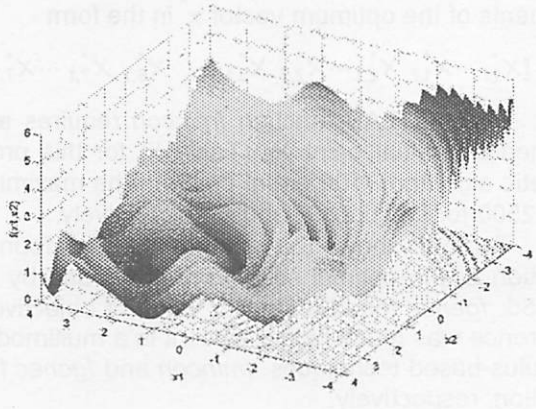


Figure 5. Function of test case #2 for $n = 2$ and $\mathbf{x}^* = (2.5, 2.5)$.

and also used for the initial population (100 equal points) in *fgenec*. For each *n*, *fgenec* was run ten times and found better values than *fminunc* (the best ones are shown in Table I). *fgenec* was executed with different population sizes depending on the value of *n*.

4.3 Test Case #3

The problem is to solve the *nonlinear transportation problem* (described in [14]) of forty nine variables:

$$\textit{minimize } f(x_1,\ldots,x_{49}) = \sum_{I=1}^{7}\sum_{J=1}^{7} g(x_{I,J})$$

subject to

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 = 27$$
$$x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} = 28$$
$$x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} = 25$$
$$x_{22} + x_{23} + x_{24} + x_{25} + x_{26} + x_{27} + x_{28} = 20$$
$$x_{29} + x_{30} + x_{31} + x_{32} + x_{33} + x_{34} + x_{35} = 20$$
$$x_{36} + x_{37} + x_{38} + x_{39} + x_{40} + x_{41} + x_{42} = 20$$
$$x_{43} + x_{44} + x_{45} + x_{46} + x_{47} + x_{48} + x_{49} = 20$$

$$x_1 + x_8 + x_{15} + x_{22} + x_{29} + x_{36} + x_{43} = 20$$
$$x_2 + x_9 + x_{16} + x_{23} + x_{30} + x_{37} + x_{44} = 20$$
$$x_3 + x_{10} + x_{17} + x_{24} + x_{31} + x_{38} + x_{45} = 20$$
$$x_4 + x_{11} + x_{18} + x_{25} + x_{32} + x_{39} + x_{46} = 23$$
$$x_5 + x_{12} + x_{19} + x_{26} + x_{33} + x_{40} + x_{47} = 26$$
$$x_6 + x_{13} + x_{20} + x_{27} + x_{34} + x_{41} + x_{48} = 25$$
$$x_7 + x_{14} + x_{21} + x_{28} + x_{35} + x_{42} + x_{49} = 26$$

This problem is composed by fourteen linear equality constraints. These kinds of constraints are not explicitly handled by *fgenec*. However, we can deal with them by eliminating the equalities and introducing them in other inequality or domain constraints. In this particular case, thirteen equations are dependent and one is independent. We can thus discard variables $x_1,\ldots, x_8, x_{15}, x_{29}, x_{36}, x_{43}$ and rename the remaining ones as $y_1,\ldots, y_{36}$. The variables y_i (i = 1,..., 36) have to satisfy two-sided inequalities resulting from the domain constraints and the transformations.

We have selected two types of functions for $g(x)$ taken from [14]:

- Function *E*:

$$E(x) = c_{I,J}\left[\frac{1}{1+(x-10)^2} + \frac{1}{1+(x-45/4)^2} + \frac{1}{1+(x-35/4)^2}\right]$$

where c_{ij} is a constant.

- Function *F*:

$$F(x) = c_{I,J}\, x[\sin(x\pi/4)+1]$$

where c_{ij} is a constant. Matrix *C*, composed by the elements c_{ij}, was taken also from [14] (Table II).

Figures 6(a) and 6(b) show the graphs of functions *E* and *F*, respectively, for the one-dimensional case. As can be noticed, *E* is unimodal while multimodality is the characteristic of *F*.

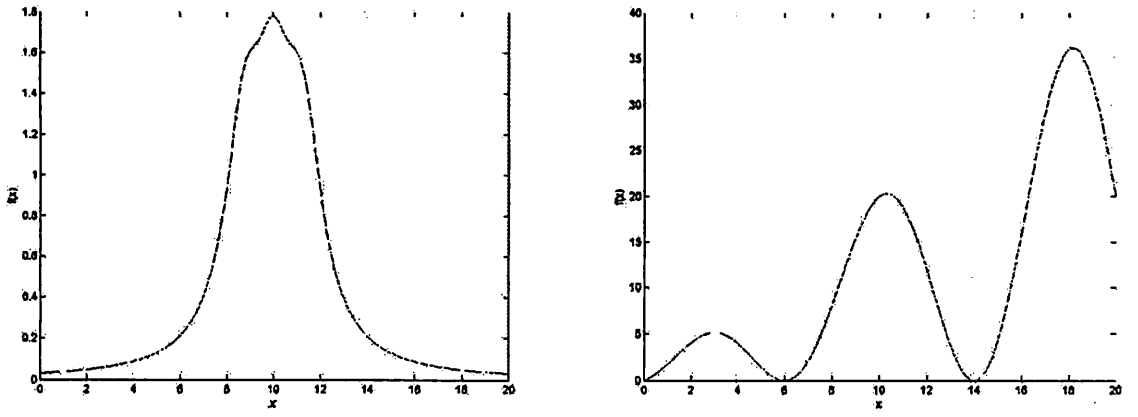
The solution of the problem is represented by a matrix X^* (7 x 7). This matrix contains the elements of the optimum vector $\mathbf{x}^*$ in the form

$$\mathbf{x}^* = [X^*_{1,1}\cdots X^*_{1,7}\ X^*_{2,1}\cdots X^*_{2,7}\ X^*_{3,1}\cdots\cdots X^*_{6,7}\ X^*_{7,1}\cdots X^*_{7,7}]^T$$

The MATLAB function *fmincon* requires an initial feasible point to start the search. We have defined the initial point X_0 (Table II) for this procedure and used it for the initial population of the genetic algorithm (100 equal points). The maximum number of generations in *fgenec* was set as 1000 and 2500 for functions *E* and *F*, respectively.

Table III shows the solution matrices found by *fmincon* and *fgenec* for functions *E* and *F*. For function *E*, the optimal solution matrix found by *fmincon* gives a value of objective function equal to 252.56. *fgenec* found a smaller value of objective function equal to 248.65. In case of function *F*, the difference was much bigger since it is a multimodal function and thus difficult to be solved by classical calculus-based techniques. *fmincon* and *fgenec* found values of 1413.33 and 510.22 for the objective function, respectively.

4.4 Other Applications



(a) (b)

Figure 6. Equivalents of functions (a) *E* and (b) *F* for the one-dimensional case in test case #3.

fgenec has been successfully applied to civil engineering problems related to the water resource field.

The use of the algorithm for parameter calibration of a rainfall-runoff model was treated in [15]. The optimization problem was formulated in terms of minimizing an objective function equal to the sum of squares of the difference between observed and calculated runoff. The calculated runoff was a function of fourteen parameters whose optimal values were appropriately determined.

Celeste [16] used *fgenec* for optimal operation of water resource systems in real time. An optimization problem was constructed for the operation of a system responsible for the water supply of Matsuyama City, in Japan. The system was composed by a multipurpose reservoir and a set of wells. The objectives were to determine the allocation of water from the reservoir and the set of wells that would best meet the demands for water supply and irrigation, and to maintain the reservoir storage as

Table II. Matrix *C* and initial point (X_0) for *fmincon*.

C							X_0						
0	21	50	62	93	77	1000	20	0	0	1	2	2	2
21	0	17	54	67	1000	48	0	20	0	2	2	2	2
50	17	0	60	98	67	25	0	0	20	0	1	2	2
62	54	60	0	27	1000	38	0	0	0	20	0	0	0
93	67	98	27	0	47	42	0	0	0	0	20	0	0
77	1000	67	1000	47	0	35	0	0	0	0	1	19	0
1000	48	25	38	42	35	0	0	0	0	0	0	0	20

Table III. Solution matrices found by *fmincon* and *fgenec* for functions *E* and *F*

	fmincon							*fgenec*						
Function *E*	20.00	0.42	0.21	2.04	1.77	2.56	0.00	0.49	0.00	0.00	0.10	1.41	25.00	0.00
	0.00	19.58	1.62	1.84	2.22	0.00	2.74	1.40	20.00	0.00	1.74	2.75	0.00	2.11
	0.00	0.00	18.17	0.59	0.55	2.10	3.60	0.00	0.00	20.00	1.17	1.83	0.00	2.00
	0.00	0.00	0.00	18.53	1.47	0.00	0.00	0.00	0.00	0.00	20.00	0.00	0.00	0.00
	0.00	0.00	0.00	0.00	20.00	0.00	0.00	0.00	0.00	0.00	0.00	20.00	0.00	0.00
	0.00	0.00	0.00	0.00	0.00	20.00	0.00	18.11	0.00	0.00	0.00	0.00	0.00	1.90
	0.00	0.00	0.00	0.00	0.00	0.34	19.66	0.00	0.00	0.00	0.00	0.00	0.00	20.00
Function *F*	20.00	0.08	0.05	0.94	0.00	5.92	0.00	14.54	0.00	0.46	0.00	0.00	6.00	6.00
	0.00	14.50	0.53	0.82	6.00	0.00	6.15	0.00	20.00	0.00	3.00	5.00	0.00	0.00
	0.00	5.41	19.33	0.25	0.00	0.00	0.00	5.46	0.00	19.54	0.00	0.00	0.00	0.00
	0.00	0.00	0.00	0.00	20.00	0.00	0.00	0.00	0.00	0.00	20.00	0.00	0.00	0.00
	0.00	0.00	0.00	20.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	20.00	0.00	0.00
	0.00	0.00	0.09	0.00	0.00	19.08	0.83	0.00	0.00	0.00	0.00	1.00	19.00	0.00
	0.00	0.00	0.00	0.99	0.00	0.00	19.01	0.00	0.00	0.00	0.00	0.00	0.00	20.00

close as possible to a given target storage in order to not let it decrease considerably. Besides, the distribution of water could not compromise the operation of the system violating its constraints and leading it to a collapse. Sequential Quadratic Programming and another evolutionary method were also used to solve the operation problem but *fgenec* was found to outperform them under several analyses.

fgenec was used in [17] to develop a computer model for simulating the performance of a water main system that transports water pumped from a well to a reservoir. The genetic algorithm was used to define the relation between the volumes in the well and reservoir and the procedure for controlling the pump, in order to provide a satisfactory working of the system under several objectives such as meeting of demands, maintenance of volumes and electric power saving.

5 Conclusions

This paper showed the characteristics and applications of a genetic algorithm written in MATLAB. MATLAB was chosen due to its user-friendly environment and the possibility of using several built-in functions that facilitate the numerical computation and the construction of the algorithm. The GA contains many features such as floating point representation of the chromosomes, selection procedure based on stochastic universal sampling, elitism, fitness ranking and a set of crossover and mutation operators specially devised to maintain the feasibility of the offspring for the case of linear constraints.

The algorithm was tested for a number of cases and examples of its application to water resources problems were displayed. The GA showed to work efficiently for the problems in contrast with some other techniques based on calculus.

References

[1] Z. Michalewicz and D. B. Fogel (2000). *How To Solve It: Modern Heuristics*. Springer-Verlag, New York.

[2] Z. Michalewicz (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd edition, Springer-Verlag, New York.

[3] D. Whitley (1993). A genetic algorithm tutorial. Technical Report CS-93-103, Department of Computer Science, Colorado State University.

[4] D. E. Goldberg (1989). *Genetic Algorithm in Search, Optimization and Machine Learning*. Addison Wesley, Reading, Massachusetts.

[5] M. Gen and R. Cheng (1997). *Genetic Algorithms & Engineering Design*. Wiley-Interscience, New York.

[6] C. Houck, J. Joines and M. Kay (1996). Comparison of genetic algorithms, random restart, and two-opt switching for solving large location-allocation problems. *Computers & Operations Research*, **23**(6), 587-596.

[7] J. H. Holland (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

[8] D. Beasley, D. R. Bull and R. R. Martin (1993a). An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, **15**(2), 58-69.

[9] D. Beasley, D. R. Bull and R. R. Martin (1993b). An overview of genetic algorithms: Part 2, research topics. *University Computing*, **15**(4), 170-181.

[10] J. E. Baker (1987). Reducing bias and inefficiency in the selection algorithm. *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, 14-21.

[11] *Optimization Toolbox User's Guide - Version 5* (1996). The MathWorks Inc., Natick, Massachusetts.

[12] D. H. Ackley (1987). *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Publishers, Boston.

[13] J. D. Pintér (2002). Global optimization: software, test problems, and applications. In: P. M. Pardalos and H. E. Romeijn (Eds.), *Handbook of Global Optimization Volume 2*. pp. 515-568. Kluwer Academic Publishers, Dordrecht.

[14] Z. Michalewicz, G. A. Vignaux and M. Hobbs (1991). A nonstandard genetic algorithm for the nonlinear transportation problem. *ORSA Journal on Computing*, **3**(4), 307-316.

[15] A. B. Celeste, K. Suzuki, M. Watanabe and A. Kadota (May 2001). Genetic algorithms for automatic calibration of Tank Model. *7th JSCE Congress of Civil Engineering - Shikoku Division*, pp. 150-151, Matsuyama, Japan.

[16] A. B. Celeste (March 2002). Optimal real-time operation of multipurpose water resource systems using genetic algorithms. *M.Sc. Thesis*. Ehime University, Department of Civil and Environmental Engineering, Matsuyama, Ehime, Japan, p. 147.

[17] C. W. S. Santana, C. O. Galvão, J. M. S. G. Barbosa and A. B. Celeste (December 2001). Computer simulation of water mains. *9th Scientific Initiation Meeting of Federal University of Paraíba*. CD-ROM, João Pessoa, Brazil (in Portuguese).