SOLVING BOUNDARY VALUE PROBLEMS BY GENETIC ALGORITHM

Ali Tavakoli[1], Daniel Plante[2]

In this paper, we describe the genetic algorithm method for solving large scale ill-posed problems. By using a suitable initial population generated by iterative methods, a genetic algorithm is developed that converges well. Moreover, in the crossover process we use a random Flexible GMRES method with (or without) an SSOR preconditioner.

Keywords: Genetic algorithm, Boundary value problem
MSC2000: 78M50, 92D15, 30E25.

1. Introduction

The idea of solving boundary value problems (BVPs) using genetic algorithm (GA) is not new. In [1], solving the inverse initial-boundary value problems via GA is presented. However, no method is presented for the selection of first population (that is very important in convergence of solution) and moreover, the presented method works only for small scale inverse boundary value problems. Furthermore, in [2] a large scale problem has been solved by GA. But, the proposed genetic algorithm is for optimal scheduling and resource allocation problems (that are formulated as integer linear programs) and are not boundary value problems.

In this paper, we propose an efficient GA for solving such large systems. By using a suitable initial population and a novel approach to real-valued recombination, the algorithm presented yields good results for convergence on systems that otherwise converge slowly or not at all. In addition, for discrete ill-posed problems, we use the FGMRES method for the crossover process with (or without) a preconditoner such that the FGMRES with GA converges well compared with the FGMRES method alone. By iterative methods converging slowly or diverging after some iterations, the idea of using these methods to initialize a population for a GA to then search for a solution is appealing.

[1]Mathematics department, Vali-e-Asr University of Rafsanajn, Iran, e-mail: tavakoli@mail.vru.ac.ir

[2]Department of Mathematics and Computer Science, Stetson University, USA

2. Operators of Genetic Algorithm

In order to obtain the solution of the large scale system

$$Au = f, \tag{1}$$

we can solve equivalently the following nonlinear programming problem:

$$\begin{array}{ll} \min \max_{1 \le i \le n} & \{|f_i - A^i u|\} \\ s.t. & Au \le f \end{array} \tag{2}$$

where A is an $n \times n$ full rank matrix, A^i and f_i for $i = 1, \cdots, n$ denote the rows of A and the columns of f respectively.

The optimization problem (2) forms a nonlinear system that is readily transformable to a linear system [3]. However, when the system is large in scale, it is not convenient to solve it by linear programming algorithms (e.g. simplex algorithm). Hence, we should use an alternative method more suitable for such large-scale problems. One method that we propose is a variation of a GA for solving (1). By an appropriate recombination (crossover) operator, we find that the algorithm works very well.

Remark 2.1. *Of course, one can use an alternative objective functions such as* $||Au - f||_2^2$. *However, in GA, we consider the fitness value based on the components of* $f - Au$. *Hence, the given objective function seems suitable.*

GAs provide a means for solving various optimization and search problems by simulating the process of natural selection and evolution [4, 5, 6]. An outline of the GA presented in this paper is provided in Algorithm 1.

Algorithm 1. Implementation of GA

(1) Input: p^c, p^m;
(2) Output: a single solution satisfying the termination criteria;
(3) initialize parent population;
(4) WHILE termination criteria not meet DO
(5) evaluate fitness of all individuals in population;
(6) WHILE child population smaller than parent population DO
(7) select two individuals from parent population;
(8) with probability p^c, apply recombination to obtain two children;
(9) with probability p^m, apply mutation operator to two children;
(10) add children to child population
(11) END
(12) apply elitism by directly transferring fittest n "parents" to child population;
(13) replace parent population with child population (new parent population);
(14) END

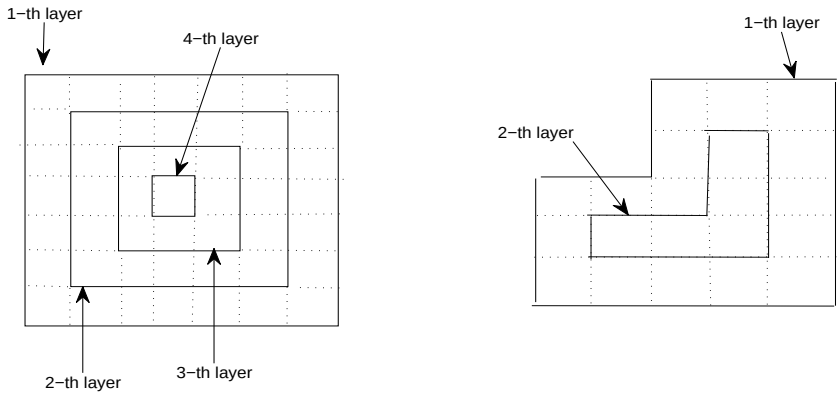

FIGURE 1. The layers of a subdivided domain.

Encoding; In our algorithm, each chromosome (individual) encodes a real-valued string. Each gene represents the corresponding component of the solution. For the recombination and mutation steps, we define **1-th layer** to be the boundary of domain $\Omega \subset \mathbb{R}^2$ and **K-th layer** be the generated K-th boundary by regular subdivision scheme in each level (see Figure 1).

Initial Population: We consider three types of initial populations:
Type 1: Run the iterative methods ν times with initial guess u_0 where ν is an integer number satisfying $1 \leq \nu \leq 3$. However, here more than one iterative method can be applied. For example we can use the Jacobi, Gauss Seidel and SOR stationary relaxation methods or non-stationary conjugate gradient and GMRES methods with different initial guesses.

Remark 2.2. *Individuals of Type 1 are used to construct a suitable initial population, because the relaxation methods converge well in a few first iterations while the subsequent iterations converge slowly. This is why the multigrid algorithm runs a few iterations with the relaxation operators. However, the multigrid algorithm needs the prolongation and restriction operators, though their construction is hard most of the times. This is one of the advantages of using a GA rather than a mutigrid algorithm for solving a BVP.*

Type 2: Generation of this type of population depends upon the physical properties of the problem and may vary for different problems. For example, when we are solving a problem with a Dirichlet boundary condition $u = 0$, the value of u can increase as we move to higher layers or it may become oscillatory from one layer to the next one. For some BVPs there exist, not necessarily exact, additional information about the physical properties of the problem other than just the boundary information. For example, symmetry of solution for a rectangular domain with respect to its diagonal or the maximum values of the solution may be known (see [7] for instance).

Type 3: In this type, an individual is produced by an interpolation multivariate polynomial. For example, when the boundary is $u = 0$ and there is information about the physical properties of solution, we may construct an interpolation polynomial (see next section).

Fitness: The fitness of an individual in a GA is the value of an objective function for its phenotype. In general, the formula for fitness depends on the objective function. Here we propose two fitness measures. One possible fitness measure is defined to be

$$fitness\_value(u) = \tfrac{1}{n}\sharp\{i : |f_i - A_i u| \leq \epsilon\} \tag{3}$$

where $\sharp$ shows the cardinality of the set, n is the dimension of the matrix A and ϵ is an acceptable bound for $\|f - Au\|_\infty$ and depends on the problem. An alternative fitness measure is

$$\begin{aligned} fitness\_value(u) &= \tfrac{1}{nw_1}(w_1\sharp\{i : |f_i - A_i u| \leq \epsilon_1\} \\ &+ w_2\sharp\{i : \epsilon_1 < |f_i - A_i u| \leq \epsilon_2\} \\ &+ w_3\sharp\{i : \epsilon_2 < |f_i - A_i u| \leq \epsilon_3\}) \end{aligned} \tag{4}$$

where $\epsilon_1, \epsilon_2, \epsilon_3$ are acceptable bounds for $|f_i - A_i u|$ and w_1, w_2, w_3 are the weights of these bounds, respectively. Moreover, ϵ_3 is the minimum acceptable bound, namely

$$fitness\_value(\{u : |f_i - A_i u| > \epsilon_3\}) = 0$$

for $i = 1, \cdots, n$. Clearly w_1 and w_3 should be the largest and smallest weights of the bounds, respectively. Also, we note that in both formulas (3) and (4), the fitness value of every vector u is no larger than 1, with $0 \leq fitness\_value(u) \leq 1$.

Selection: For our implementation of GA, we use *truncation selection* to determine which individuals survive to the next generation. This can be implemented by allowing some percentage of the present generation to pass on to the next generation or by selecting some fixed number of individuals to do so, here, we choose the latter approach. Also, we implement *elitism* to preserve monotonicity of maximum fitness for the population from generation to generation, with the fittest three parents in the present generation being copied to the new population.

Hybrid Recombination (Crossover): Recombination is the process in which two parent solutions are mated to produce offspring. For our implementation, we begin by implementing recombination as explained in [8]. There, Mühlenbein and Schlierkamp-Voosen present their Breeder Genetic Algorithm (BGA), which is a recombination of evolutionary strategies (ES) [9, 10] and GAs [11]. Mühlenbein and Schlierkamp-Voosen describe three recombination algorithms whose operators work well especially when BGA is used for parameter optimization. However, when the genes of the individuals collectively represent a function, we have found that the above algorithms do not work.

In the present work, all solutions are found to diverge when these algorithms are implemented.

Algorithm 2. Crossover process

(1) Input: u_1, u_2 (two individuals);
(2) Output: Generation of children by crossover process;
(3) set $r = [rand \times n] + 1$ where n and $rand$ show the length of the matrix and a random number between 0 and 1, respectively. Also, $[x]$ denotes the largest correct number no less than x;
(4) set $ch_1(1:r) = u_1(1:r)$ and $ch_1(r+1:n) = u_2(r+1:n)$;
(5) set $ch_2(1:r) = u_2(1:r)$ and $ch_2(r+1:n) = u_1(r+1:n)$;
(6) run one iteration of selected method with initial guesses ch_1 and ch_2 to produce $child_1$ and $child_2$, respectively;
(7) modify ch_1 and ch_2.

Remark 2.3. *In Step 6 of Algorithm 2, only those individuals whose fitness value is more than the average of fitness values participate in the crossover process.*

Therefore, we present a novel approach which is found to work well. First, the reproduction operator selects at random a pair of two individual strings, u_1 and u_2, for mating. Each pair of selected parents generate two children using Algorithm 2. In Step 6 of the algorithm, we run one iteration of a stationary relaxation method with initial guesses ch_1 and ch_2 to generate two initial children. However, for discrete ill-posed problems, the crossover Algorithm 2 may not be effective and one can instead use the Krylov subspace method. In Algorithm 3, we provide a flexible GMRES method (FGMRES) as described by Simoncini and Szyld [12]. Using this algorithm, the method selected in Step 6 of Algorithm 2 is the FGMRES method with SOR variable preconditioning.

Algorithm 3. Random Flexible GMRES method (RFGMRES)

(1) Input: A, b, x_0;
(2) Output: Generation of a child;
(3) $m = \lfloor 30 \times rand + 1 \rfloor$;
(4) compute $r_0 = b - Ax_0, \beta = \|r_0\|_2$ and $w_1 = r_0/\beta$;
(5) FOR $j = 1, \ldots, m$ DO
(6) compute $z_j = GMRES(A, w_j)$;
(7) FOR $i = 1, \ldots, j$ DO
(8) $T_{i,j} = (Az_j, w_i)$;
(9) END
(10) compute $v = Az_j - \sum_{i=1}^{j} T_{i,j} w_i$;
(11) compute $T_{j+1,j} = \|v\|_2$;
(12) compute $w_{j+1} = v/T_{j+1,j}$;

(13) `END`
(14) `Compute` $y_m = argmin_y \|\beta e_1 - T_m y\|_2$ and $x_j = x_0 + z_j y_m$.

In the Algorithm 3, m is a random integer number in the interval $[1, 30]$. We note that when the inner system is preconditioned from the right, this can be viewed as a global preconditioning strategy (see for instance [12]). In other words, we consider preconditioning for the inner system (Step 6 of Algorithm 3) with a fixed matrix P, so that the inner system is transformed to $AP^{-1}\hat{z}_j = w_j$ with $z_j = P^{-1}\hat{z}_j$. Hence, we apply FGMRES for the system $AP^{-1}\hat{x} = b$ with $\hat{x} = Px$. One can consider P, for example, to be either a SSOR preconditioner, in which the relaxation parameter ω is a random number in the interval $(0, 2)$, or an incomplete LU factorization (ILU) in which the tolerance is a random number in the interval $(0, 1)$. Also, one can use other preconditoners, such as Lanczos bidiagonalization preconditioner, for discrete ill-posed problems [13].

Mutation: After recombination, the strings are subjected to mutation. Mutation prevents the algorithm from becoming trapped in local minima. We implement mutation by allowing each gene x_i of individual x to be mutated with probability p_m where $p_m = 1/n$ and

$$z_i = x_i \pm f \times r_{max} \times p.$$

Here, r_{max} is set to the maximum range of values for x, f is some fraction of the maximum range (typically some small values such as 0.1), and p is a randomly generated real number in the interval $(0, 1)$.

3. Numerical results

In this section, we present some numerical experiments and compare the exact solutions with the approximated solutions obtained by our implementation of the GA.

Example 1. The first example is defined on a unit square. To define the grid, we start by dividing the square into nine smaller squares of side length 1/3 and then dividing each smaller triangle into two triangles (see Figure 2). Thus, a triangulation τ_1 is determined for the coarsest grid. Suppose τ_k with $k \geq 2$ is obtained by τ_{k-1} via a regular subdivision; the edge midpoints in τ_{k-1} are then connected by new edges to form τ_k.

We consider the following Poisson's problem:

$$\begin{aligned} -\Delta u &= f(x, y), \qquad \text{in } \Omega = [0, 1] \times [0, 1], \\ u &= 0, \qquad\qquad \text{on } \partial\Omega. \end{aligned} \tag{5}$$

Suppose that we know the maximum value of exact solution u occurs at the midpoint $(1/2, 1/2)$ of Ω, but its value is unknown. In order to construct an individual of Type 3 of the initial population, we make a bivariate interpolation polynomial $P(x, y)$ by setting $P(0, 0) = P(1/2, 0) = P(1, 0) = P(0, 1) = P(1/2, 1) = P(1, 1) = 0$ and $P(1/2, 1/2) = \alpha$ where α is a scalar. Therefore,

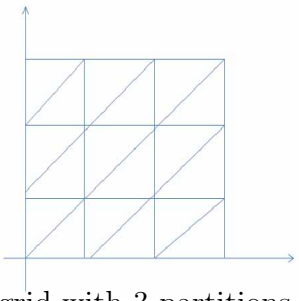

FIGURE 2. Coarsest grid with 3 partitions in x and y directions.

TABLE 1. GA for Poisson's problem with different right hand sides f.

u	ft_{ip}	GA_{itr}	J_{itr}	GS_{itr}	SOR_{itr}
$10x(1-x)y(1-y)$	1	1	130	∞	∞
$10x(1-x)y(1-y)sin(\pi x)sin(\pi y)$	0.27	22	120	∞	∞
$10(x-x^2)^3(y-y^2)^2$	0.96	3	10	6	5
$sin(\pi x)sin(\pi y)$	0.18	24	142	∞	∞
$10x(1-x)y(1-y)sin(2\pi xy)$	0.18	21	120	∞	∞

TABLE 2. GA for Poisson's problem.

h_k	ft_{ip}	GA_{itr}	J_{itr}	rate
1/12	0.18	24	142	5.92
1/24	0.34	87	130	1.49
1/48	0.62	249	1000	4.02
1/96	0.81	45	1416	31.47

we can form the individual u_α in terms of α. In order to find α, it is enough to solve the linear equation $b(i) - A^i u_\alpha = 0$ for some i.

For Step (7) of the recombination algorithm in GA mentioned in the previous section, we first modify all points located in the neighbourhood of midpoint $(1/2, 1/2)$ as

$$child(i) = (b(i) - \sum_{j \neq i} A(i,j) \times child(j))/A(i,i).$$

We consider the fitness value as defined in Equation (4).

In Tables 1 and 2, ft_{ip} shows the maximum fitness of the initial population, GA_{itr} denotes the fewest number of iterations need to reach the fitness value 1 in GA. Moreover, J_{itr}, GS_{itr} and SOR_{itr} show the number of iterations needed to converge the Jacobi, Gauss-Seidel and SOR algorithms, respectively. In Table 1, we consider different right hand sides f with corresponding exact solutions u of Poisson's problem given by Equation (5) for $h_k = 1/24$. We used $\omega = 1.5$ for the relaxation parameter in the SOR algorithm. GA_{itr} calculates the total number of iterations carried out for all generations. For the

generation of the initial population, we used $\nu = 2$ iterations for the Jacobi, Gauss-Seidel and SOR methods with initial guesses $x_0 = 0, \pm 1$. There are 13 individuals in the initial population and they are constructed as follows:

- 9 individuals are of Type 1 comprised of 3 each for Jacobi, Gauss-Seidel, and SOR
- 2 individuals of Type 2
- 1 individual of Type 3
- 1 individual produced from the initial layer

The stopping criteria is considered as $\|b - Av\|_\infty < \epsilon$ for the fittest individual in the population, where $\epsilon = .001$.

In Table 2, the values of ft_{ip}, J_{itr} and GS_{itr} are calculate for level 3 ($h_k = 1/12$) through level 6 ($h_k = 1/96$) and f has been chosen such that $u = sin(\pi x)sin(\pi y)$ is the exact solution to the Poisson equation (5). As can be seen from the table, as the level increases, the fitness of the initial population increases. In the last column, we indicate the rate between J_{itr} and GA_{itr} for each level. This rate is increasing saliently from level 4 to 6.

Example 2. We consider the computation of the second derivative **DERIV2** example from Hansen [15]. This is a mildly ill-posed problem that is a discretization of a Fredholm integral equation of the first kind,

$$\int_0^1 K(s,t)x(t)dt = g(s), \qquad 0 \le s \le 1,$$

whose kernel K is the Green's function for the second derivative:

$$K(s,t) = \begin{cases} s(t-1), & s < t \\ t(s-1), & s \ge t. \end{cases}$$

Both integration intervals are $[0, 1]$, and as the right-hand side, g, and corresponding solution, x, one chooses the following:

$$RHS\ 1 : g(s) = (s^3 - s)/6, x(t) = t,$$

$$RHS\ 2 : g(s) = \begin{cases} \dfrac{4s^3 - 3s}{24}, & s < \frac{1}{2}, \\ \dfrac{-4s^3 + 12s^2 - 9s + 1}{24}, & s \ge \dfrac{1}{2}, \end{cases} \quad x(t) = \begin{cases} t, & t < \frac{1}{2}, \\ 1 - t, & t \ge \frac{1}{2}. \end{cases}$$

We implement the GA for **DERIV2** with RHS 1 and RHS 2 for $n = 1000$ that yields $A \in \mathbb{R}^{1000\times 1000}$. For the fitness value, we consider $\epsilon_1 = 10^{-8}, \epsilon_2 = 10^{-4}$ and $\epsilon_3 = 10^{-2}$. We use an elitism process such that in each iteration, the 3 individuals with the highest fitness value directly survive to the next generation. Moreover, mutation is run on the 3 elite individuals as $z_i = x_i \pm 0.01p$ where p is a randomly generated real number in the interval $(0, 1)$. In the first iteration, only 12 individuals exist as initial population. In addition, we consider at most 25 individuals for the crossover process. The stopping criteria is chosen as $MAXF = 1$ where $MAXF$ denotes the maximum value

TABLE 3. Convergence trend of GA for **DERIV2** with RHS 1.

Iteration	IL	MAXF	$\|b - Ax_k\|_2$
1	12	0.5360	1.0114e-005
2	29	0.5360	1.0114e-005
3	29	0.5935	5.7405e-006
⋮	⋮	⋮	⋮
12	29	0.9985	3.8319e-008
13	29	1.0000	2.1456e-008

of fitness in each iteration. Also, in Tables 3 and 4, IL shows the number of individuals in each iteration.

Table 3 shows the convergence trend of GA for **DERIV2** with RHS 1. A good initial population is selected, since the maximum fitness value is 0.5360 in the first iteration (i.e. more than 50% fitness). Also, by including elitism in our GA, the maximum fitness value includes increasing trend for successive iterations. As is expected, the residual $\|b - Ax_k\|_2$ changes a few between any two iterations. However, from Table 3, we observe that in only 13 iterations, the residual significantly decreases. Of course, one may change the value of ϵ_1, ϵ_2 and ϵ_3 in the definition of fitness value to find a better residual of error with the cost of more iterations and CPU time.

TABLE 4. Comparison of GA and FGMRES(m) for **DERIV2** with RHS 2.

method	Iteration	IL	MAXF	$\|b - Ax_k\|_2$	CPU(sec.)
DERIV2-RHS 2	1	12	1	1.7e-17	15
FGMRES(10)	3	–	–	1.8e-16	1.6
FGMRES(15)	3	–	–	1.0e-17	2.0
FGMRES(20)	2	–	–	1.4e-16	1.4
FGMRES(25)	2	–	–	5.8e-17	1.6
FGMRES(30)	2	–	–	6.7e-17	1.8

Table 4 shows a comparison between GA and FGMRES(m) with different values of m for **DERIV2** with RHS 2, where m denotes the step number of FGMRES method. Both methods use the SSOR preconditioner. The FGMRES(m) method utilizes the relaxation parameter $\omega = 0.5$ for the SSOR preconditioner, whereas GA uses a random relaxation parameter $0 < \omega < 2$. Although GA requires more CPU time compared with FGMRES(m), GA converges with only one iteration. However, using RHS 2, results are less favourable.

Example 3. For final example, we consider the **PHILLIPS** test problem known as Phillips problem [14]. This problem involves the discretization

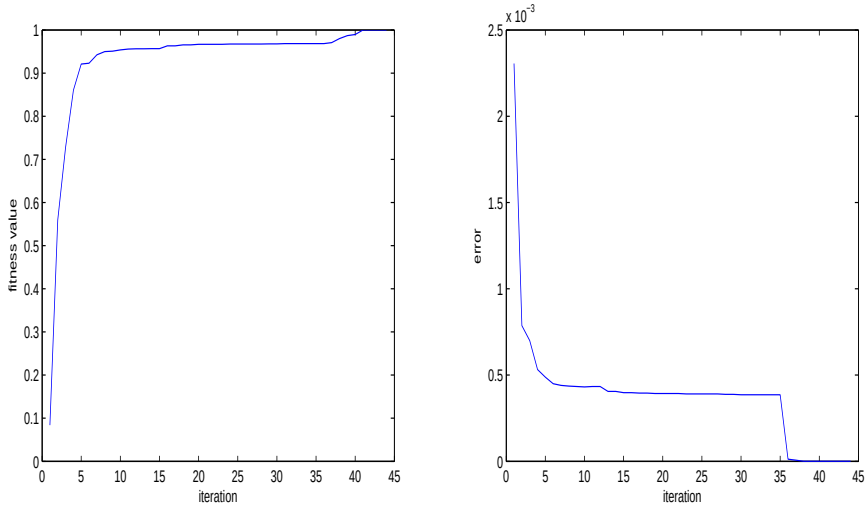

FIGURE 3. (Left) Illustration of fitness value, (Right) Illustration of error in each iteration for **PHILLIPS** test problem.

of the Fredholm integral equation of the first kind derived by D. L. Phillips. The function is defined by

$$\phi(x) = \begin{cases} 1 + \cos(x\pi/3), & |x| < 3, \\ 0, & |x| \geq 3. \end{cases}$$

with the kernel K, solution x, and right-hand side g given by

$$\begin{aligned} K(s,t) &= \phi(s-t), \\ x(t) &= \phi(t), \\ g(s) &= (6-|s|)(1+0.5\cos(s\pi/3)) + 9/(2\pi)\sin(|s|\pi/3). \end{aligned}$$

Both integration intervals are $[-6, 6]$ and also the order of discretization, n, must be a multiple of 4.

We implement GA for the **PHILLIPS** example with $n = 1000$ that yields $A \in \mathbb{R}^{1000\times 1000}$. For the fitness value, we consider $\epsilon_1 = 10^{-8}, \epsilon_2 = 10^{-4}$ and $\epsilon_3 = 10^{-2}$. We again use elitism to propagate the 3 individuals with highest fitness values to the next generation. Moreover, mutation is run on the elite individuals as $z_i = x_i \pm 0.01p$ where p is a randomly generated real number in the interval $(0, 1)$. In the first iteration, only 12 individuals exist as initial population. In addition, we require that no more than 25 individuals participate in the crossover process. The stopping criteria is chosen as $MAXF = 1$, where $MAXF$ denotes the maximum value of fitness in each iteration.

Figure 3(Left) shows the relation between iteration and fitness value. As is seen, although the maximum fitness of the initial population is 0.0839, the GA converges to maximum fitness 1 after only 44 iterations. Moreover,

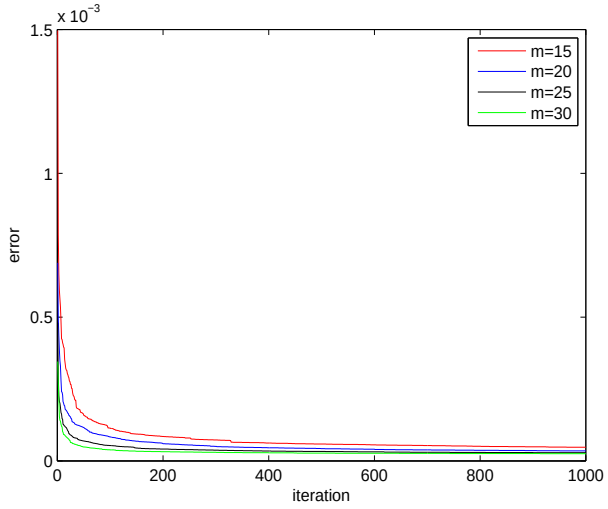

FIGURE 4. Convergence trend of FGMRES(m) for **PHILLIPS** test problem.

in the first few iterations, we observe that the fitness value grows rapidly, thereafter converging more slowly. Figure 3(Right) shows a relation between iteration and error $\|x_{exact} - x_k\|_2$. The error value is 0.0023 for the best initial individual (i.e. a individual with highest fitness value) and after 44 iterations, the error converges to 7.5701×10^{-9}. The error decreases rapidly in the first few iterations, levels off for a number of iterations, indicating temporary trapping in a local minimum, decreases rapidly after iteration 35. Figure 4 shows that the FGMRES(m) method with different steps m never decreases to an error below 10^{-5}. Also, while the initial error is approximately 10^{-4}, even after 1000 iterations, the error reaches a value only about 0.1 times of initial error. While not displayed here, we also ran FGMRES(20) for 10000 iterations, but again, the error never dropped below a value of 10^{-5}. We have also presented the elapsed CPU time for GA and FGMRES(m) in Table 5 that show the efficiency of GA compared with FGMRES(m).

TABLE 5. Comparison of GA and FGMRES(m) for **PHILLIPS** problem.

method	Iteration(k)	$\|x_{exact} - x_k\|_2$	CPU(sec.)
GA	44	7.6e-009	642
FGMRES(15)	1000	4.7e-005	684
FGMRES(20)	1000	3.46e-005	825
FGMRES(25)	1000	2.8e-005	965
FGMRES(30)	1000	2.5e-005	1104

4. Conclusion

A hybrid genetic algorithm was presented for solving boundary value problem. Several types of initial population are given for running the genetic algorithm. Moreover, in crossover process, we used Flexible GMRES method. Some test problems are given to confirm the importance of our method over the usual relaxation methods.

REFERENCES

[1] *C. L. Karr, I. Yakushin and K. Nicolosi*, Solving inverse initial-value, boundary value problems via genetic algorithm, Eng. Appl. Art. Intel., **13**(2000), 625-633.

[2] *K. Deb and K. Pal*, Efficiently solving: A large-scale integer linear program using a customized genetic algorithm, Genetic and Evolutionary Computation-GECCO 2004, 1054-1065.

[3] *D. Luenberger and Y. Ye*, Linear and nonlinear programming, third edition, Springer, 2008.

[4] *M. Gen and R. Cheng*, Genetic algorithms and engineering optimization, J. Wiley & Sons, 2000.

[5] *J. H. Holland*, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[6] *X. Yu and M. Gen*, Introduction to evolutionary algorithms, Springer, 2010.

[7] *Y. Kanno, M. Ohsaki, K. Murota and N. Katoh*, Group symmetry in interior point methods for semidefinite program, Opt. Eng., **2**(2001), 293-320.

[8] *H. Mühlenbein and D. Schlierkamp-Voosen*, Predictive models for the breeder genetic algorithm: I. continuous parameter optimization. Evolutionary Computation, **1**(1993), 25-49.

[9] *H.-P. Schwefel*, Numerical optimization of computer models, Wiley, 1981.

[10] *Th. Bäck, F. Hoffmeister, and H.-P. Schwefel*, A survey of evolution strategies. Proceedings of the Fourth International Conference on Genetic Algorithms, 1991.

[11] *D.E Goldberg*, Genetic algorithms in search, optimization and machine learning, Addison-Wesley, 1989.

[12] *V. Simoncini and D. B. Szyld*, Flexible inner-outer Krylov subspace methods, SIAM J. Numer. Anal., **40**(2003), 2219-2239.

[13] *Sh. Erfani, A. Tavakoli, and D.K. Salkuyeh*, An efficient method to set up a Lanczos based preconditioner for discrete ill-posed problems, Appl. Math. Model., **37**(2013), 8742-8756.

[14] *M. E. Kilmer, D. P. Oleary*, Choosing regularization parameter in iterative methods for ill-posed problems, SIAM J. Matrix Anal. Appl., **22**(2001), 1204-1221.

[15] *P. C. Hansen*, Regularization tools: a MATLAB package for analysis and solution of discrete ill-posed problems, Numer. Algorithms, **6**(1994), 1-35.