Optimization with Machine Learning Aided Genetic

Fazel Arasteh*, Saeed Khodaygan*,

*Mechanical Engineering Department, Sharif University of Technology, Tehran, Iran

Abstract

The initial population of an evolutionary algorithm is the most important factor in speed and the probability of convergence of the algorithm. The initial population is often selected uniformly from the feasible space. In this article, we propose a method for selecting a promising initial population for the genetic algorithm. We call our approach as Learned Genetic Algorithm or LGA since it is based on machine learning. After sampling the feasible space with several local searches, we use a support vector machine (SVM) to learn the relation between the cartesian orientation of a point and its prospective final fitness value. The points with a good fitness estimation are the promising initial points that are passed to the downstream genetic algorithm. Empirical experiments on classical test functions and the problem of optimal design of a PID controller of an AC-servo-motor illustrate that our approach increases the speed and accuracy of the genetic algorithm.

Keywords

Genetic Algorithm, Support Vector Machine, PID controller

1. Introduction

One of the well-known methods for global optimization is the genetic algorithm (GA). Many problems have been solved using the genetic algorithm [1,2]. One of the important factors in the performance of the genetic algorithm is the selection of the initial population. The more the initial population is away from the global optima, the less probability to spot the global optima. Hence, a better initial population means less run time and more accuracy of the algorithm. In this paper, we are looking for a method to produce a promising initial population for the genetic algorithm. This method uses Machine Learning to drive an estimation for the location of the global optima. The initial population is chosen from the promising area that is created in the earlier step.

The use of machine learning for accelerating optimization problems has precedences in the literature of the optimization. In [3], classification methods were used for selecting promising initial points for the Multi Start algorithm. A random point from the feasible space is used as the starting point of the Multi Start Algorithm if a previously learned model accepts it as a promising point. In [4], a convex underestimation for the objective function is built based on a set of function evaluations. The global optima of this underestimated objective function are readily available with typical convex optimization tools. This global optimum is a good promising point to start the Multi Start algorithm and find the global optima of the original function. Many other methods also make use of former information to build a better model to estimate the objective function. For instance, in [5] a version of learning is embedded in evolutionary algorithms. Calculating the original fitness function in the process of an evolutionary algorithm can be computationally heavy. A good estimation of the fitness function that can be calculated with less computation effort accelerates the whole algorithm. In [6], a new version of the genetic algorithm is introduced. Based on the population of every generation, a classification model is learned. The population of the next generation is the ones that are accepted by the model learned in the current generation. However, to the best of the knowledge of the authors, there has been no previous effort to make a promising initial population for the genetic algorithm based on

machine learning. We call this new algorithm Learned Genetic Algorithm (LGA). Since the initial population of a genetic algorithm certainly affects its speed and accuracy this new method can be of prominent importance.

The rest of the paper is organized as the following: in 2, an introduction to the support vector machine is reviewed. In 3, the Learned Genetic Algorithm (LGA) is introduced. The definition of a typical Genetic Algorithm (UGA) is reviewed and a second algorithm namely the Sampled Genetic Algorithm (SGA) is introduced. In 4, the effectiveness of the proposed algorithm is tested against numerical experiments. In 5, LGA, UGA, and SGA are compared. Finally, in 6, the optimal design of a PID controller system for an AC Servo Motor is deployed using LGA and the results are investigated.

The main contributions of this work are:

- Introducing LGA, Learned Genetic Algorithm: a novel method to create a promising initial population for the genetic algorithm.
- Empirical testing of the LGA with well know test functions: Rastrigin, Shewfel, Michalewicz test function.
- Case Study: optimal design of a PID controller system to control an AC Servo Motor

2. Introduction to Support Vector Machines

In this section, we will review the Support Vector Machine (SVM) as a supervised learning method. Assume a set of labeled data with equation (1):

$$TS=\left\{(xi,yi),\ xi \in Rn,\ yi \in \{0,1\},\ i=1,\ldots,N\right\} \quad (1)$$

In classification problems, we are interested in a model that accepts a feature vector as input and predicts the appropriate label as output. Assume that X is the feature vector in R^n. The SVM model for binary classification is given with equation (2):

$$h_\theta(x)=\begin{cases}1 & if\ \theta^T x \geq 0\\ 0 & otherwise\end{cases} \quad (2)$$

In (2), θ is the weight vector, and the goal is to find the optimal which minimizes the prediction error of the model. Intuitively, equation (2) describes a hyperplane that divides the space into two parts. Every point that resides on the same side is labeled with the same label.

The prediction error of the model can be formulated with (3):

$$J(\theta)=\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\text{cost}_1\left(\theta^T x^{(i)}\right)+\left(1-y^{(i)}\right)\text{cost}_0\left(\theta^T x^{(i)}\right) \quad (3)$$

In (3), cost_0 and cost_1 are defined with equations (4-6):

$$\text{cost}_0(z)=\max(0,\text{k}(1+z)) \quad (4)$$

$$\text{cost}_1(z)=\max(0,\text{k}(1-z)) \quad (5)$$

$$z=\theta^T x \quad (6)$$

To find the optimal hyper-plane, we solve the optimization problem (7) that minimizes the error function given in (3):

$$\text{Min}\ J(\theta) \quad (7)$$

To overcome the overfitting problem, one should add the regularization parameter (C) to the cost function. Equation (8) gives the general form of the cost function with the regularization parameter:

$$J(\theta)=C\sum_{i=1}^{m} y^{(i)}\text{cost}_1\left(\theta^T x^{(i)}\right)+\left(1-y^{(i)}\right)\text{cost}_0\left(\theta^T x^{(i)}\right)+\frac{1}{2}\sum_{j=1}^{n}\theta_j^2 \quad (8)$$

Nonlinear Regions and Kernel Trick

In many practical problems, we need models that can classify nonlinear regions. The kernel methods enable us to build complicated nonlinear classifiers. To use the kernel method, one should define an inner product as the kernel. In this method, the first step is to choose the Land Marks (usually every point in the training set is considered a landmark). The second step is using the Kernel to calculate the proximity of every arbitrary point to every landmark. Based on the chosen landmarks and the predefined kernel, a new feature vector (which is denoted with f) is calculated with equation (10). The rest of the process is the same as the linear classifiers. The cost function for kernel method is given in (11).

$$f_i=\text{similarity}\left(x,l^{(i)}\right)=exp\left(-\frac{\left|x-l^{(i)}\right|^2}{2\sigma^2}\right) \quad (10)$$

$$h_\theta(x) = \sum_{i=1}^{\text{Number of land marks}} \theta_i f_i \qquad (11)$$

In (10), we used one of the most well-known kernels, Gaussian Kernel. To use this kernel, one must assign the hyper-parameter σ. Intuitively, this parameter shows how fast the effect of that landmark on the fitness function decreases by moving away from a landmark. Assigning the hyper-parameters σ and C is the most important step in building a good SVM model. Using a grid-search is very common to assign these two parameters. In this method, the data set is divided into two separate sets, the training set, and the cross-validation set. The training set is usually 75% of the whole data set, and 25% is left for the cross-validation set. Separate arrays of subsequent values are assumed for σ and C. For every combination of values for σ and C, we train a model based on the training set. Then the model error is calculated on the cross-validation set. Finally, we choose the hyper-parameters of the model with least cross-validation error.
We used LibSVM library which is available in MATLAB as an optimized SVM library. For further information regarding SVM we cite [7].

3. Proposed Algorithm

In this section we first review the steps of a normal Genetic Algorithm. Then we propose the main contribution of this work, LGA. Finally, we introduce SGA that we used for comparison purposes.

Genetic Algorithm with Uniform Distribution of the Initial Population

Assume the following functions:
G(size): it generates a uniformly distributed population with the assigned size from the feasible space.
CrossOver(ppul): it receives a population as input and generates a new population by performing the cross over action on the pairs of the old population. Then it returns the best-fitted individuals of the new population as output.
Mutate(ppul): it mutates a few individuals of the input population randomly and outputs the generated population.
TerminalCheck(ppul): it receives a population as input and if the algorithm meets a terminal condition, it returns true. The terminal condition can be the use-up of memory or time resources or the convergence of the algorithm.
The sudo-code for a typical genetic algorithm is given in (code_1).

```
UGA(Population_Size){
        ppul=G(Population_Size);
//Uniformly sample the  feasible space
        while(!TerminalCheck(ppul)){
//Continue until some terminal condition is meet
                ppul=CrossOver(ppul);
                ppul=Mutate(ppul);
        }
        return (best fitted individual of the
population);
}
```

Sudo Code1- conventional GA

We call this algorithm Uniform Genetic Algorithm or UGA.

Genetic Algorithm with Machine Learning Driven Initial Population

Suppose routine R(ppul) is a routine that receives an initial population as input and performs a local search on each individual and returns the local optima (refinedPpul) and the optimal values (Vals) for each local optimum. We must decide a threshold value (T). We will discuss it in future sections. Then we fill the index array (Indx). The sudo-code-2 shows the procedure:

```
for ( i= 1 to size(Vals)){
        if(Vals(i)<=T)
                Indx(i)=1; //accepted point
        else
                Index(i)=-1; //rejected point
        }
```

Sudo Code 2: Indexing

For each individual in the initial population that the value of the obtained local optima is less than the threshold, we put a “1” in index array. We put a “0” for each local optimum that is greater than threshold.

The hyper-parameter T is the border that divides the promising initial points from others. We suppose that values for the obtained global optima have a normal distribution. We expect that around 2.5% of the data are less than the average minus two times the standard deviation. Hence, we choose the value of the average minus two times of the standard deviation as the hyper-parameter T. However, since the distribution is not necessarily normal, it is possible that the obtained T is less than all the data. In this case, we choose the smallest value in the population as T so that we at least have one positive point in the index array.

We call our proposed algorithm Learned Genetic Algorithm, LGA. Sudo-code-3 illustrates this algorithm:

```
Create_Model (Sampling_Size) {
        uniformPpul=G(Sampling_Size);
//uniformly sample the feasible space
        [refinedPpul, Vals]=R(uniformPpul);
// run a local search from each point and store gained values in Vals

T=max (min(Vals) , mean(Vals) — 2*sqrt(var(Vals));
// use 2_sigma rule of normal distributions and check with
//the menimum of the population
        Indx=(Vals<=T);
//generate the appropriate indexes (MATLAB Style)
        [Cross_Validation_Set , Training_Set]=divide_population(refinedPpul , Indx , 25% , 75%);
[C, Sigma] = search_C_Sigma(Cross_Validation_Set , Training_Set);
SVM_model= SVM.Train(refinedPpul , Indx , C , Sigma);
// train the classifier with the labeled data
        if (Test_Model(model))
// store the model if it has a good performance
                STORE(SVM_model);
}

LGA(Population_Size ) {
        LOAD(SVM);
// load an appropriate model for the problem
        ppul={};
 //initially null population
        while(ppul.size()<Population_Size) //create a new population that  is accepted by the classifier
                X=generate a random point in feasible area:
```

Sudo Code 3: Learned Genetic Algorithm (LGA)

Genetic Algorithm with Sampled Initial Population

To illustrate the importance of the machine learning model in the performance of the algorithm, we propose another algorithm. We use the local optima that were driven in the first step of the LGA directly as the initial population of the genetic algorithm. We call this algorithm Sampled Genetic Algorithm, SGA:

```
SGA(Population_Size) {
uniformPpul=G(Population_Size); //uniformly sample the feasible space
        refinedPpul=R(uniformPpul);        // run a local search from each point and replace
                                                each point with the local optima that is found
        while(!TerminalCheck(refinedPpul))
                        ppul=CrossOver(ppul);
                        ppul=Mutate(ppul);
        end
        return (best fitted individual of the population);
}
```

Sudo Code4: Sampled Genetic Algorithm (SGA)

4. Empirical Experiments

In this section, we first introduce several well-known test functions. To get a grasp of how LGA works, we solve these test functions in their 2-D version with the LGA algorithm, and we illustrate the results. We investigate the effect of sampling size on the effectiveness of LGA in the next section.

Rastrigin:

The n-dimensional Rastrigin test function is defined with (12) [9]. The domain is usually a box defined with (13). The global minimum of this function is in X=0 with value=0.

$$f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos\left(2\pi x_i\right) \right] \quad (12)$$

$$x_i \in \left[-5.12, 5.12\right], \text{ for all } i = 1,\ldots,n \quad (13)$$

The number of local optima for this function in higher dimensions is too many for the multi-start algorithm to solve it readily. It is also a challenging problem for other optimization methods. It is important to note that although this function has many local optima, the location of these optima has a normal distribution in the feasible space. Fig3 illustrates the performance of the LGA model with a hundred sample points. Yellow circles show negative points, while the black pluses are the positive points. Note that the one black plus is many of the positive points that are all in the same position. Note that the model has well found the location of the global optima because the problem is also easy to solve for the multi-start algorithm.

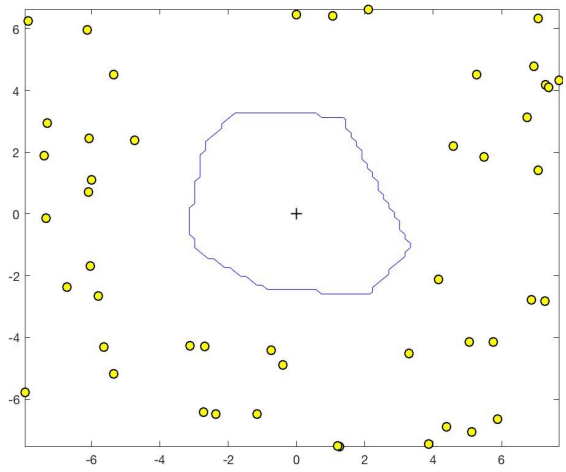

Fig 3-Model function on 2-D Rastrgian test function

Schewfel

The n-dimensional Schewfel function is defined with equation (14) [9]. It is usually defined on a box domain (15).

$$f(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{|x_i|}\right) \tag{14}$$

$$x_i \in [-500,500] \ , \ \text{for all } i = 1,\ldots,n \tag{15}$$

The global minimum of this function is placed in X_i=420.9687 with the optimal value -418.9829*n. From an optimization perspective and analytical methods, this is a trivial test function since one can decompose it to n=1 and solve it analytically or numerically. However, for the optimization algorithms that cannot decompose the problem, it turns out to be a very complicated test function more intensively in higher dimensions. Fig4 illustrates the performance of the LGA model with a hundred sample points.

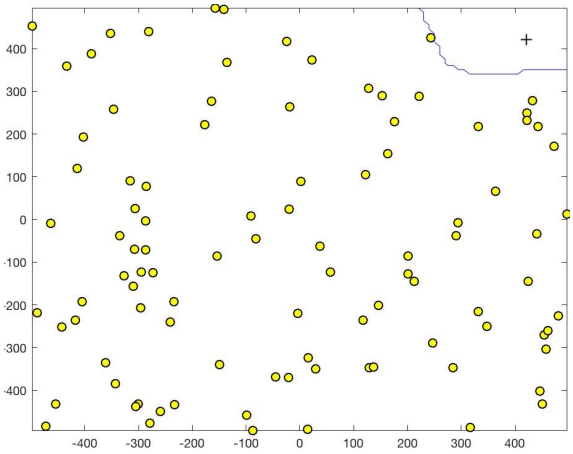

Fig 4-Model function on 2-D schwefel test function

Michalewicz

The n-dimensional Michalewicz test function is defined with equation (16) with a box domain (17). The global optima of this function with m=10 is given in (17-20).

$$f(x) = -\sum_{i=1}^{n} \sin(x_i)\sin^2\left(\frac{i x_i^2}{\pi}\right) \tag{16}$$

$$x_i \in [0,\pi] \ , \ \text{for all } i = 1,\ldots,n \tag{17}$$

$$\text{at } n = 2: f(x^*) = -1.8013, \ \text{at } x^* = (2.2, 1.57) \tag{18}$$

$$\text{at } n = 5: f(x^*) = -4.687658 \tag{19}$$

$$\text{at } n = 10: f(x^*) = -9.66015 \tag{20}$$

This function has *n!* local optima and has flat shoulders that make the optimization harder. The hyper-parameter *m* identifies the slope of the valleys and hills. A very large *m*, results in the failure of every optimization algorithm. A moderate setting is *m*=10.

Fig5 illustrates the performance of the LGA model with a hundred sample points. Fig6 shows the procedure of a genetic algorithm that its initial population is driven by the learned model. Due to the simplicity of the problem in low dimensions, the answer exists in the initial population, and hence, a perfect start has happened.

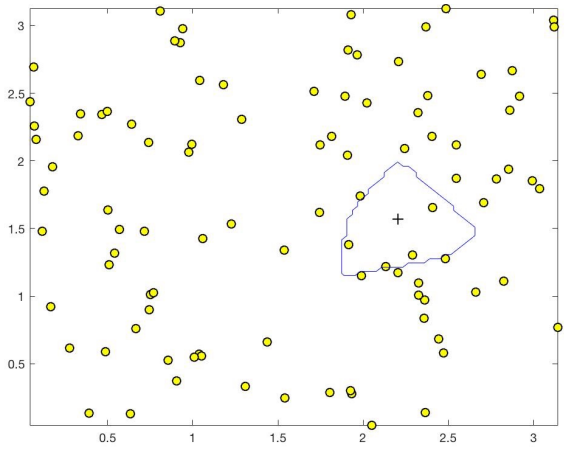

Fig 5-Model function on 2-D Michalewicz test function

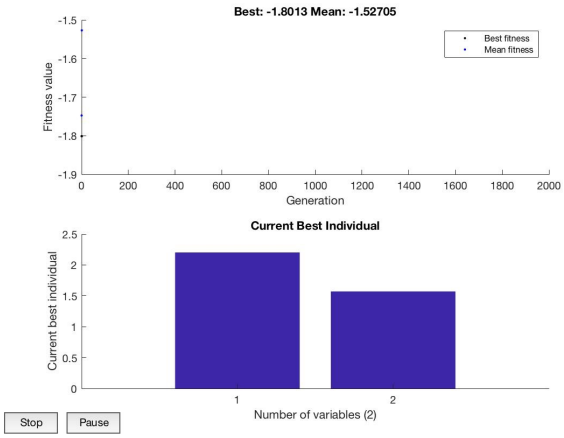

Fig 6-GA run with initial population formed by the model

Sampling Size Effect

The most important factor in the performance of every machine learning model is the number of training data. We designed an experiment to investigate the effect of the sampling size on the performance of the LGA. For different sampling sizes, we ran the LGA for 20 times, and we calculated the following metrics:

- CV_Error: this metric shows the cross-validation error.
- Perfect_Initiation_Ratio: this metric shows that what percent of the runs have halted in the first iteration. In a perfect run of the algorithm, the answer exists in the initial population.
- meanNG: this metric shows the average number of generations that the genetic algorithm took to complete.

Sampling Size Experiment on Rastrigian:

We set the following settings:

- Number of G.A. runs: 20
- Fitness Limit[1]: 2*10^-9
- Max Stall Generations[2]: 10,000
- Population Size:30

Figure 7 illustrates the results of this experiment.

Sampling Size Experiment on Schewfel:

We set the following settings:

- Dimension: 6
- Number of G.A. runs: 20
- Fitness Limit: -418.9* Dimension
- Max Stall Generations: 20,000
- Population Size:30

Figure 8 illustrates the results of this experiment.

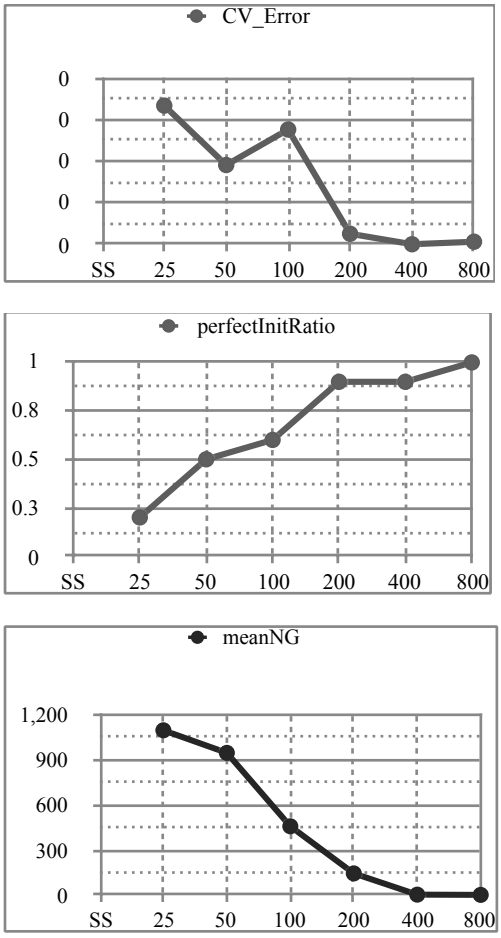

Fig 7- Sampling size effect on functionality of LGA (Rastrigian)

Note that increasing the sampling size, increases the Perfect_Initiation_Ratio and decreases the Cross-Validation Error and the average number of generations for algorithm to stop. Hence, with the increase of the sampling size, the accuracy of the model as a whole increase. However, with the increase of sampling size, the LGA diverges from a typical genetic algorithm and is further like a multi-start algorithm. With further increasing the sampling size, the computational cost to make the samples will dominate the cost to run a usual genetic algorithm. Hence, the sampling size must be rational. Based on the above experiments sampling_size=100 is a promising choice.

[1] When the fitness function in the G.A. reaches this limit, it halts meaning that it has found the answer .

[2] The number of generations to stall

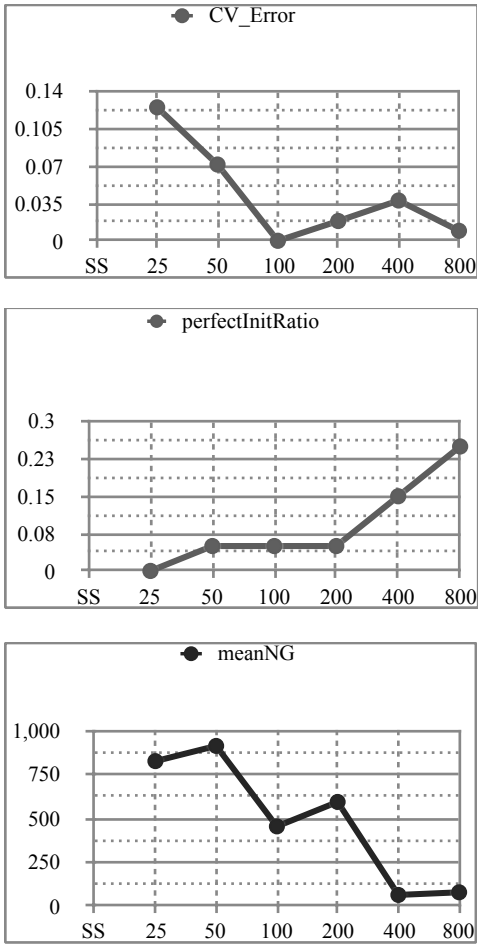

Fig 8- Sampling size effect on functionality of LGA (Schewefel)

made with 100 samples has readily found the place of the global optimum.

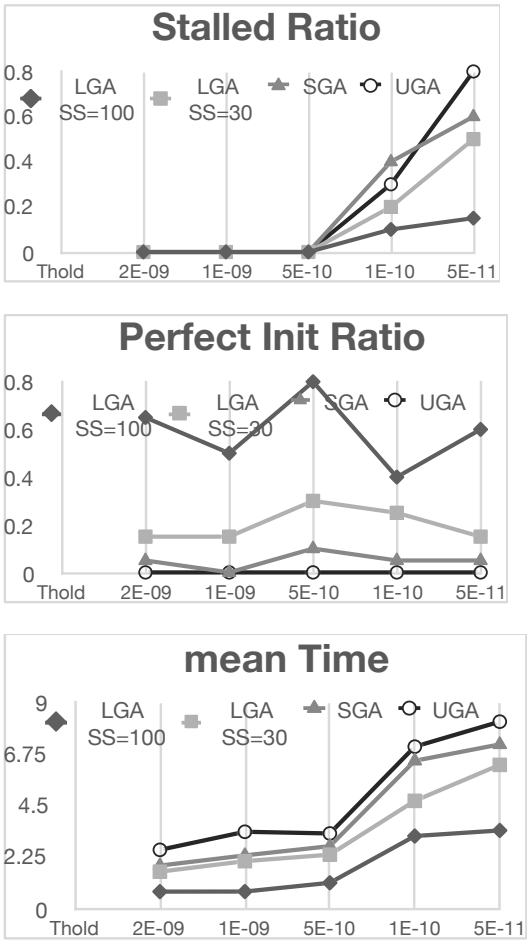

Fig 9- Functionality of LGA with the threshold getting more extreme (Rastrigian)

5. Comparison between LGA, SGA, and UGA

In this section, we compare the LGA, SGA, and UGA on the metrics of the number of generations, Perfect Initiation Ratio, and Stall Ratio. In each step, we chose a more accurate fitness limit. The setting for the genetic algorithm is the same as the previous section. Fig9 illustrates the results of Rastrigian_10D test function. Note that with increasing the accuracy of the fitness limit, the Stall Ratio for all three algorithms increases. However, LGA performs better than UGA and SGA. Also, note that LGA_100 performs better than LGA_30. The Perfect Initiation Ratio for LGA_100 is very high, meaning that the model

It is important to note that although LGA_30 and SGA have used the same computational effort, LGA_30 is more successful. Hence, using the machine learning model rather than the local optima driven from local searches is accelerates the genetic algorithm.

For the Schewfel_6D test function, the same trends can be observed in Fig10. In this case, the domain is wider which makes the problem much harder for UGA. Here, we can see that both LGA_30 and LGA_100 perform better than UGA and SGA by a large margin.

In the case of Michalewicz_5D, the trends in the Fig11 are the same. Note that in this case LGA_30 and LGA_100 are performing

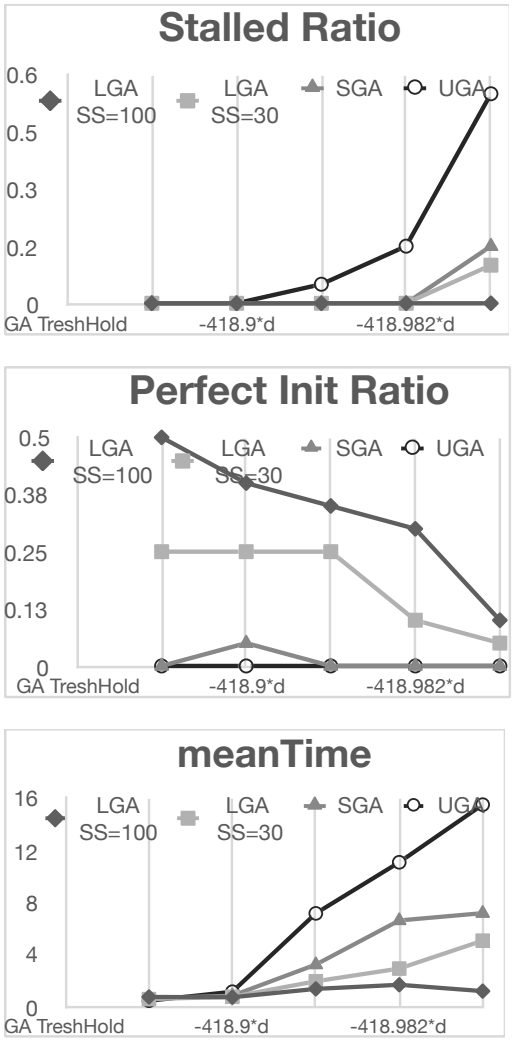

Fig 10- Functionality of LGA with the threshold getting more extreme (Schewefel)

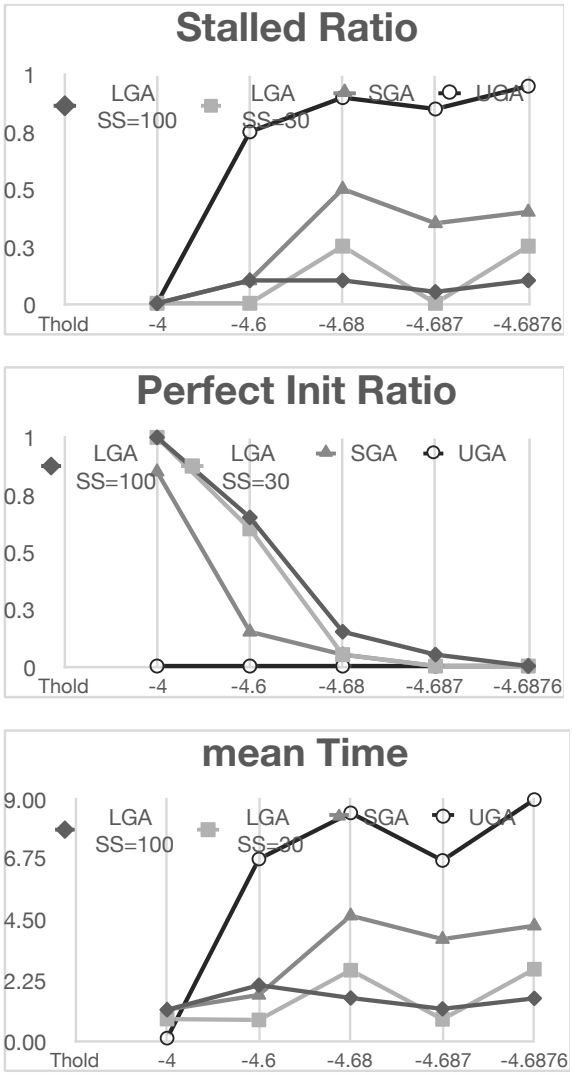

Fig 11- Functionality of LGA with the threshold getting more extreme (Michalewicz)

at the same level. This means that the problem was easy enough that 30 sample points sufficed to learn the location of the global optimum. Again, in this case, LGA_30 performs better than SGA by a large margin.

6. Case Study, Optimal Design of the PID Controller of an AC Servo Motor

In this section, we aim to illustrate the potential of LGA in improving the accuracy of the optimization. We used LGA for the optimal design of a dynamic system and compared the results with the design of a typical genetic algorithm.

Problem Definition

In industry, one typical way to control an AC servo motor is to use two PID controller. One of the controllers is for the speed feedback and the other is for the placement feedback.

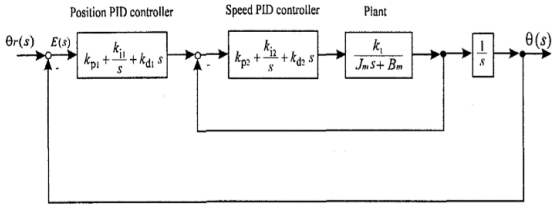

Fig 12- Schematic of the control system of an AC servo motor

The schematic of this system is illustrated in Fig12. It is assumed that the plant is a single-input-single-output (SISO), Linear, and time-invariant system. The plant is modeled with the transform function G given in (21):

$$G(s) = \frac{k_t}{J_m s + B_m} \tag{21}$$

The PID controllers are modeled with transform function C given in (22):

$$C(s) = k_p + \frac{k_i}{s} + k_d s \tag{22}$$

The parameters of the placement PID controller are indexed with 1 which are Kd1, Ki1, Kp1 and the parameters of the speed PID controller are indexed with 2 which are Kd2, Ki2, Kp2.
Error signal, E(s), for the system shown in Fig12 when the input is a unit step is given in (23). The parameter of (23) are given in (24-32):

$$E(s) = \frac{d_0 s^3 + d_1 s^2 + d_2 s + d_3}{a_0 s^4 + a_1 s^3 + a_2 s^2 + a_3 s + a_4} \tag{23}$$

$$d_0 = J_m + k_{d2} \bullet k_t \tag{24}$$

$$d_1 = B_m + k_{p2} \bullet k_t \tag{25}$$

$$d_2 = k_{i2} \bullet k_t \tag{26}$$

$$d_3 = 0 \tag{27}$$

$$a_0 = J_m + k_{d2} \bullet k_t + k_{d1} \bullet k_{d2} \bullet k_t \tag{28}$$

$$a_1 = B_m + k_{d2} \bullet k_{p1} \bullet k_t + k_{p2} \bullet k_t + k_{d1} \bullet k_{p2} \bullet k_t \tag{29}$$

$$a_2 = k_{d2} \bullet k_{i1} \bullet k_t + k_{i2} \bullet k_t + k_{d1} \bullet k_{i2} \bullet k_t + k_{p1} \bullet k_{p2} \bullet k_t \tag{30}$$

$$a_3 = k_{i2} \bullet k_{p1} \bullet k_t + k_{i1} \bullet k_{p2} \bullet k_t \tag{31}$$

$$a_4 = k_{i1} \bullet k_{i2} \bullet k_t \tag{32}$$

A typical and well-known method for designing PID controllers is the Integral of Squared Error, ISE because it is easy to calculate its integral in the frequency space. One drawback for this method is the long settling time that is due to the fact that in this method all the errors have the same weight independent from their time. An alternative method is Integral of time-weighted squared error (ITSE) [10]. Suppose W is the value of ITSE criteria given in (33):

$$W = \int_0^\infty t e^2(t) dt \tag{33}$$

It can be transformed in to Laplacian space in the form of (34):

$$= -\frac{1}{2\pi j} \int_{-j\infty}^{j\infty} \frac{d}{ds}\big(E(s)\big) E(-s) ds \tag{34}$$

Assuming $E(s) = D(s)/A(s)$, $D(s) = \sum_{h=0}^{n-1} d_h s^{n-h-1}$, $A(s) = \sum_{h=0}^{n} a_h s^{n-h}$ we can rewrite equation (34) in the form of equation (35):

$$= -\frac{1}{2\pi j} \int_{-j\infty}^{j\infty} \frac{d}{ds}\left(\frac{\sum_{h=0}^{n-1} d_h s^{n-h-1}}{\sum_{h=0}^{n} a_h s^{n-h}}\right) \frac{\sum_{h=0}^{n-1} d_h (-s)^{n-h-1}}{\sum_{h=0}^{n} a_h (-s)^{n-h}} ds \tag{35}$$

This integral is solved using the residual theorem. The solution for Wn [10] with respect to the parameters of the transform function E(s) which is itself a function of the parameters of the controllers is given in (36). The parameters of (36) are given in (37-43):

$$W_4\left(k_{p1}, k_{p2}, k_{i1}, k_{i2}, k_{d1}, k_{d2}\right) = A - B + C \tag{36}$$

$$A = \frac{d_3^2}{4a_4^2} \tag{37}$$

$$B = \frac{Z_1}{N_1} \tag{38}$$

$$Z_1 = 2a_4 d_0^2 + 2a_0\left(d_2^2 - 2d_1 d_3\right) + a_2\left(d_1^2 - 2d_0 d_2\right) + a_1\left(d_1 d_2 - 3d_0 d_3\right) + a_3 d_1 d_1 + \frac{\left(a_1 a_2 - a_0 a_3\right)}{a_4} d_2 d_3 + \frac{d_3^2}{a_4}\left(a_0 a_2 + a_1^2\right) \tag{39}$$

$$N_1 = 2(a_1 a_2 a_3 - a_0 a_3^2 - a_1^2 a_4) \tag{40}$$

$$C = \frac{Z_2}{N_2} \tag{41}$$

$$Z_2 = \left(a_3^2 d_0^2 + a_1^2\left(d_2^2 - 2d_1 d_3\right) + a_1 a_3\left(d_1^2 - 2d_0 d_2\right) + \frac{a_1}{a_4}\left(a_1 a_2 - a_0 a_3\right) d_3^2\right)\left(a_1 a_3 - 4a_0 a_4 + a_2^2\right) \tag{42}$$

$$N_2 = 2(a_1 a_2 a_3 - a_0 a_3^2 - a_1^2 a_4)^2 \tag{43}$$

To obtain the optimal controller, it is sufficient to optimize the equation (36) for W_n. In fact, it is enough to solve the (44) optimization problem. Since the controller gains cannot be negative and very large gains are impractical, the constraint (45) should be considered:

$$\min_{\text{K stabilizing}} W_4(K) \tag{44}$$

$$0 < K \leq 10 \tag{45}$$

Also, the answers that will lead the poles of the system to locate on the right side of the imaginary coordinate are not acceptable. To prevent these answers one might use the Routh Array [11]. For the performance reasons and the simplicity in the implementation, we use numerical methods to locate the poles of the system. The distance of the poles from the imaginary coordinate is considered as a constraint in the genetic algorithm. The answers that do not satisfy this constraint will get penalized and eliminated from the population.
This problem has many local optima, hence, is a hard problem for gradient based optimization methods. Since it is also a challenge for a typical genetic algorithm not to stuck in the local optima, this problem can showcase the potential of the LGA in finding the global optimum.

Implementation and Results

In this section the above problem is solved with UGA and LGA and the performance of the designed controllers is compared with each other. For the data of this section given in (46) we cite [2]:

$$k_t = 4.42\ 10^{-2}\text{Nm/A}\ ,\ J_m=2.5\ 10^{-4}\ \text{kg m}^2, \tag{46}$$

$B_m=8.59\ 10^{-5}$ Nm/(rad/s)

The settings for both of the algorithms are the following:
Population Size: 30
Max Generations: 200
Max Stall Generation: 200
The Sampling Size for LGA is set to 100 so that we make sure that the feasible space is well investigated and a promising model is created.
Fig13a and Fig13b illustrate the procedure of the G.A. for UGA and LGA respectively. The value of W function in the initial population for LGA is relatively low which means that the created model has well performed and has result in a promising initial population.

Table1 contains the results of UGA, LGA_100, and the results that Krohling et al. obtained in [2]. As expected, UGA results and Krohling et al results are very similar. However, LGA has out-performed both

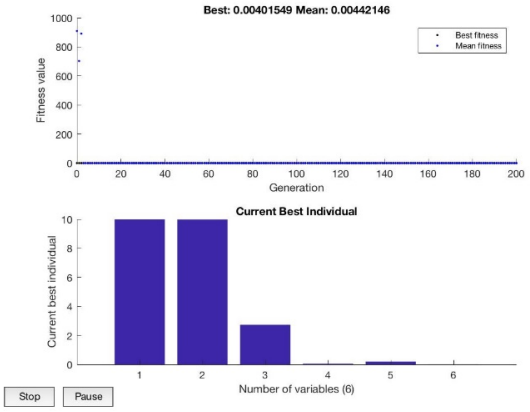

Fig 13a- Controller design: UGA

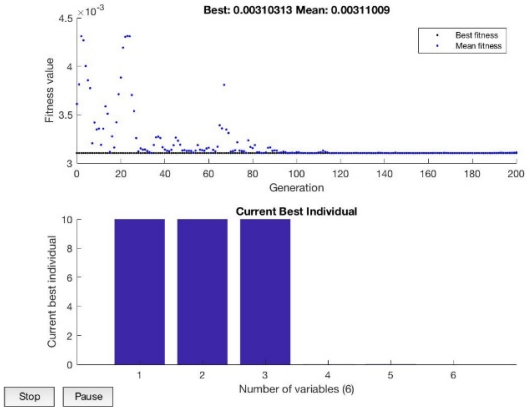

Fig 13b- Controller design: LGA

UGA and Krohling et al results with a 25% reduction in the optimal W. Fig14 illustrates the unit step response of both designs. Accordingly, the LGA design resulted in an 80% reduction in maximum overshoot, 92.5% reduction in settling time, and 45% reduction in rising time. In Fig15 one can see the location of the poles and zeros of the system designed by LGA and UGA. It is important to note that since the poles and zeros of LGA design are so intimately located the system is very stable.

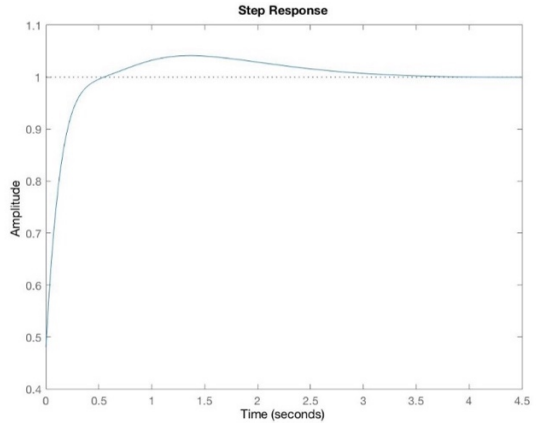

Fig 14a- Schematic of the control system of an AC servo motor

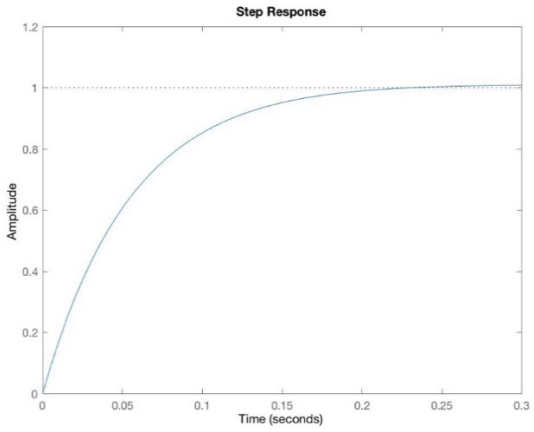

Fig 14b- Schematic of the control system of an AC servo motor

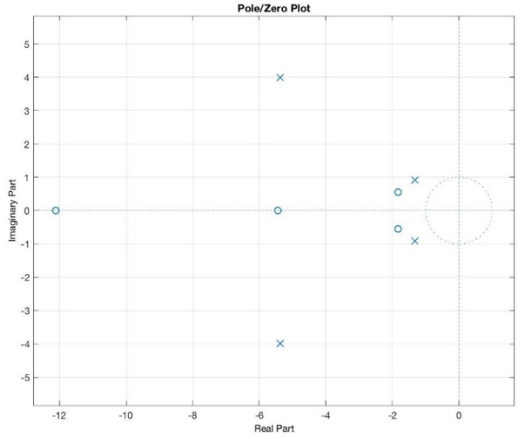

Fig 15a- Schematic of the control system of an AC servo motor

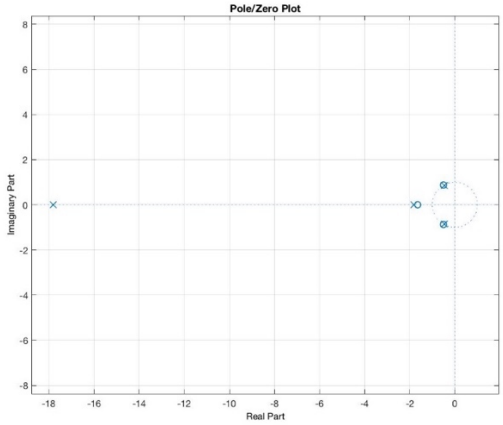

Fig 15b- Schematic of the control system of an AC servo motor

	Kp1	Ki1	kd1	Kp2	Ki2	kd2	W
Krohlong	10	9.98	1.675	0.071	0.362	0	0.004
UGA	9.995	9.988	2.734	0.050	0.191	0.003	0.004
LGA_100	10.00	10.00	10.00	0.010	0.017	0.00	0.003

Table 1: Comparison of the LGA, UGA and Krohlong et al. results

7. Conclusion and Further Development

In this paper we introduced a method to create a promising initial population for the genetic algorithm. We called our method LGA. Based on empirical studies, we showed that LGA performs better than typical GA especially when more accuracy is needed. We also introduced SGA. SGA uses the data from a series of local searches as the initial population of GA. SGA and LGA use the same processing power for their preprocess. We showed that LGA performs better than SGA in the empirical experiments. The reason for this ability of LGA can be traced back to the ability of machine learning models to be stable against noises. Since LGA performance is directly dependent on the performance of its background ML model it is very important to train the best possible ML model. ML methods are varied and SVM is only one classic method. For further studies one can consider using Neural Networks

instead of SVM to learn a more accurate model in less amount of time. Another important factor in the performance of the ML model is the feature vector of inputs. In this paper we used the simples possible feature vector
which was the coordinates of the points in the N-dimensional space. However, one can consider more complicated feature vectors. One suggestion is to combine the first and second derivatives of the objective function to the coordinates of the points and use the whole combination as the feature vector.
In this paper we used gradient descent methods to create the training set. However, one can use a series of GAs with limited maximum generation to accelerate the process of creating the training set.
Although LGA in its current version may not be able to substitute the typical GA, with further developments it can be outperform the typical GA in the cases that we need higher accuracies.

8. Bibliography

[1] P. K. Singh, S. C. Jain, and P. K. Jain, "Advanced optimal tolerance design of mechanical assemblies with interrelated dimension chains and process precision limits," *Computers in Industry,* .pp ,2 .no ,56 .vol .2005 /2005/02/01 ,194-179

[2] R. A. Krohling, H. Jaschek, and J. P. Rey, "Designing PI/PID controllers for a motion control system based on genetic algorithms," in *Proceedings of 12th IEEE International Symposium on Intelligent Control*.pp ,1997 , .130-125

[3] A. Cassioli, D. Di Lorenzo, M. Locatelli, F. Schoen, and M. Sciandrone, "Machine learning for global optimization," *Computational Optimization and Applications,* journal article vol. 51, no. 1, pp. 279-303, January 01 2012.

[4] O. L. Mangasarian, J. B. Rosen, and M. E. Thompson, "Global Minimization via Piecewise-Linear Underestimation," *Journal of Global Optimization,* journal article vol. 32, no. 1, pp. 1-9, May 01 2005.

[5] J. Yaochu, M. Olhofer, and B. Sendhoff, "A framework for evolutionary optimization with approximate fitness functions," *IEEE Transactions on Evolutionary Computation,* .vol .2002 ,494-481 .pp ,5 .no ,6

[6] R. S. Michalski, "LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning," *Machine Learning,* journal article vol. 38, no. 1, pp. 9-40, January 01 2000.

[7] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," (in English), *Data Mining and Knowledge Discovery Data Mining and Knowledge Discovery,* ,2 .vol .1998 ,167-121 .pp ,2 .no

[8] J. K. Patel and C. B. Read, *Handbook of the normal distribution*Marcel :New York . .1996 ,.Inc ,Dekker

[9] .(27/6/2018 ,2017) .Bingham .D *Virtual Library of Simulation Experiments*. Available: http://www.sfu.ca/~ssurjano/optimization.html

[10] J. H. Westcott, "The minimum-moment-of-error-squared criterion: a new performance criterion for servo mechanisms," *Proceedings of the IEE - Part II: Power Engineering,* ,101 .vol .1954 ,480-471 .pp ,83 .no

[11] E. J. Routh and e. Adams prize, "A treatise on the stability of a given state of motion : particularly steady motion," (in English), 2005.