GENETIC ALGORITHM FOR FUNCTION APPROXIMATION: AN EXPERIMENTAL INVESTIGATION

Abdulrahman Baqais[1]
[1]Department of Information and Computer Science, KFUPM, Dhahran, Saudi Arabia

ABSTRACT

Function Approximation is a popular engineering problems used in system identification or Equation optimization. Due to the complex search space it requires, AI techniques has been used extensively to spot the best curves that match the real behavior of the system. Genetic algorithm is known for their fast convergence and their ability to find an optimal structure of the solution. We propose using a genetic algorithm as a function approximator. Our attempt will focus on using the polynomial form of the approximation. After implementing the algorithm, we are going to report our results and compare it with the real function output.

KEYWORDS

Genetic Algorithm, Function Approximation, Experimentation

1. INTRODUCTION

Function approximation is used in science and engineering field in order to find the relationship between two or more variables: the independent variable (x) and the dependent variable (y) [1-3]. In such situation, input-output pairs of any engineering or scientific problems are collected using a suitable experiment or instrument. However, a direct relation in the form of equation can't be inferred between these pairs due to the absence of strong correlations.

To address this issue, Artificial Intelligence (AI) algorithms have been utilized to produce an equation between input-output pairs with higher accuracy. Genetic algorithms have been implemented in various articles with promising results as in [4-6].

2. LITERATURE REVIEW

Genetic algorithm is a metaheuristic algorithm that is designed by Holland in 1975 [7]. The algorithm starts by providing a random population of points of the search space. At each iteration, the algorithm applies some operators such as: crossover and mutation on the current population and evaluate them using an objective (fitness) function. In Function approximation, the objective function of each problem is to come up with an equation with higher accuracy between input-output pairs. That is, the objective function is to reduce the errors of the equation. Different error measurement can be used such as: Least Absolute Error (LAE), Mean Squared Error (MSE) or residual sum of squares (RSS).

There are many studies that use different AI algorithms including GA to finding the function approximation as in [4, 6].

DOI: 10.5121/ijaia.2016.7301

In [8], a variant of neural network named Wavelet Neural Network was used to approximate the function of pollution at Texas in USA. In [9, 10], the authors were able to provide an equation of higher accuracy with few coefficients of real-world data problem concerning the seawater density. The equation was generated using GA. Fuzzy Systems were used in [11] to approximate nonlinear relations.

3. RESEARCH METHODOLOGY

In this paper, we are proposing using genetic algorithms for approximation of polynomial functions. Genetic algorithms are known for their fast search for solution in a large complex space. Basically, Genetic algorithms can effectively find an optimal approximated curve to any mathematical equations with high accuracy. It should be noted that genetic algorithms requires a deep understanding of the problem and a crafted design of the chromosome. The fitness function must be chosen carefully to reflect the most contributable feature to the final solution. By maximization of the fitness function in each generation, genetic algorithm will exhibit a schema that converges to an optimal robust solution.

In the next section, we are going to explain the details of our program and experiment. The rationale behind each decision in our experiment is explained in details with sound justifications. Each step in setting up our experimental framework is written clearly with supporting figures. It must be understood that the objective of this project is to design a specific genetic algorithm for approximating polynomial function. Since this is an Artificial Intelligence course, our aim is to give a deep analysis of the behavior of genetic algorithm and its parameters.

We used Matlab Toolbox to give us a quick access to various parameters of the genetic algorithm such as selection, number of generation and for plotting purposes. However, the design of the chromosome and the fitness function is coded to be adjusted to our problem.

In function approximation, the most feature of the solution is to have a high accuracy of the obtained curve to the original equation. This implies very low errors in approximating each point in relative to the actual point. So our fitness function will be minimization rather than maximization. To ensure, that our fitness function is coded correctly, the errors must be decreasing through the generations until a convergence or an optimal solution is found. Mean squared error (MSE) was used as a fitness function and it's defined:

$$\text{MSE} = \frac{1}{n} \sum (approximated - actual)^2$$

The population of our solutions is represented as binary bits. This gives us the flexibility we need to manipulate the chromosome in a variety of ways as we are going to explain shortly. Each chromosome (or individual) contains many terms where each term is composed of two parts:

- **Coefficient**
- **Power of the variable.**

Each term is composed 15 bits, where 10 bits is allocated to coefficient and the remaining bits are allocated to represent the power.

Now, for simplicity we will assume that all terms are polynomial functions since polynomial functions are known to be a universal function approximator.

Another contribution of this paper lies in the range the power of each terms may have. In normal regression approximation, the researcher is constrained to use only integer values in the power part. However, using a resolution concept, we are allowing both the coefficient and the power to have real values. The range of values that the coefficient and the power may take is depicted in the Table 1.

Table 1: Range of Coefficient and Power Variables

Variable	**Resolution**
Coefficient	(Coefficient /10) – 50.4
Power	(Power *0.25) - 4

le. Another aspect that needs to be considered is the probability of crossover and mutation and the range of bits they are applied to. Crossover and mutation can be implemented in various ways: either on the term boundary, on the part unit boundary or within the individual bits. Each method has its merits and drawbacks based on the targeted problem. In this specific chromosome design, we opt to play with the crossover and mutation within the bits themselves. This is done for twofold: one all of our terms are in polynomial shape and hence targeting with the term boundary will not help much in converging the solution. If our chromosome design exhibits other function types such as trigonometric function, then targeting terms boundary may have better results. The second reason is that manipulating individual bits increase the chances of finding better solution. Even though, It may increases the complexity of the problem and the search space to explore in general large problems, the simplicity of our problem that allocates only 15 bits to the term mitigate these issues and the extra time added will be negligible or of little significance as we are going to show in the experiment sections.

We opt to represent our population of individuals as binary strings. That gives us the flexibility to specify the range of the values of the coefficient and the powers. The following figure shows the basic design of the chromosome:

The Fitness Function: $f(x) = |y - \hat{y}|$ where y refers to the real output and $\hat{y}$ is the approximated output.

The Final form of the polynomial equation will be on the form of:

$$A(x) = coefficent * X_1^{power} + coefficient * X_2^{power} \ldots + coefficent * X_n^{power}$$

4. EXPERIMENTAL SETUP:

In the literature review, we explored different functions proposed by many authors as the target equation. Some choose normal polynomial; others opt for Rastrigin functions due to its suitability since it poses many local minimum points and some may prefer to run their experiments approximating real equations from engineering domains. In this paper, we opt to target polynomial functions of the form:

$$F(x) = x^2$$

Where x ranges between 0.1 - 10.00 with an increasing step of 0.1.

Choosing the number of samples is very critical to be able to approximate any function. If the number of samples is too small, then approximated function may miss some critical points and thus the curve may not be correctly approximated. If the number is too large, then we will have

the issue of the over fitting, not to mention the added penalty in time and performance. So, in regarding to that, we might apply some sampling theorem that is applied extensively in statistics. Instead, we prefer to include only 100 of points. It's important to understand the features of the problem to be able to provide an efficient solution. Since we know from basic mathematics that a squared function usually takes a shape of a parabola, then it will be enough to take all the points on one side of the parabola and be able to approximate it, since the parabola is symmetric around their axis. Arbitrarily, we decided to take all the points in the positive side of the parabola, which has no more advantage or any increasing in contribution than the negative side. It's just a matter of an arbitrary decision.

A sharp observer may notice that the starting point of our samples is 0.1 not 0 and he or she might wonder what is the rationale behind this decision. Selecting 0 as a sample point will do more harm than good. Multiplying 0*0 in the actual calculation and then trying to approximate it will result in a returning value of **INF** in Matlab. Matlab toolbox is set to tolerate a predefined value of approximation to any actual number. By attempting to approximate ($0^{\wedge 2}$), the approximator will be largely deviated and the error values between actual and approximated is reaching a very large value that consequently leads to the returning value of infinity by Matlab.

5. RESULTS

After presenting to the reader the design of our solution and all the rationale behind our experiments setup, we are ready now to run our experiment and record all the values.

To provide a good framework on how our experiment is designed to improve the solution, we must decide on what parameters affect the obtained results. Next, we can validate our solution accuracy by varying these parameters. In this problem, we have three parameters they may affect the accuracy which are:

- Number of terms in the chromosome.
- The population of individuals.
- The number of generations.

Since our objective in this paper is to come up with an approximated curve of high accuracy, the number of terms is not of much significance. If our problem is targeting reducing the number of terms, then that parameter should be incorporated in the experiment. Hence, we fix the number of terms in any chromosome to be 135 bits. That is, any chromosome will have 9 terms. The population and the number of generations however will change and the obtained fitness function will be recorded in each run. Table 2 gives the variation of our experiments.

Table 2: Experiments parameters

Experiment Case No	No of terms	Population Size	No of Generation
1	9	1000	1000
2	9	1000	300
3	9	1000	50
4	9	200	1000
5	9	50	1000

Case 1: (Best Case)

Population Size: 1000
Generations : 1000
Fitness : 0.252990008807331

In this run instance, we set the population size and generation size to 1000, As you can see from the figure (1) , we reached the best value of all of our running experiments by obtaining a fitness value of 0.25299. Since the fitness Value represents the error, it implies that the approximated function returned by this instance of genetic algorithm will have an error as low as 0.25299. To have a clearer picture of the behavior of our fitness function, figure (2) shows the same figure but at log scale. Here, we can see that our fitness function is minimized from one generation to another until it reaches convergence by not evolving in hundreds of generations.

It is clear that Log Figure shows the minimization from one generation to another better than the linear figure. Thus, in the following experiments, we will show the Log figures only.

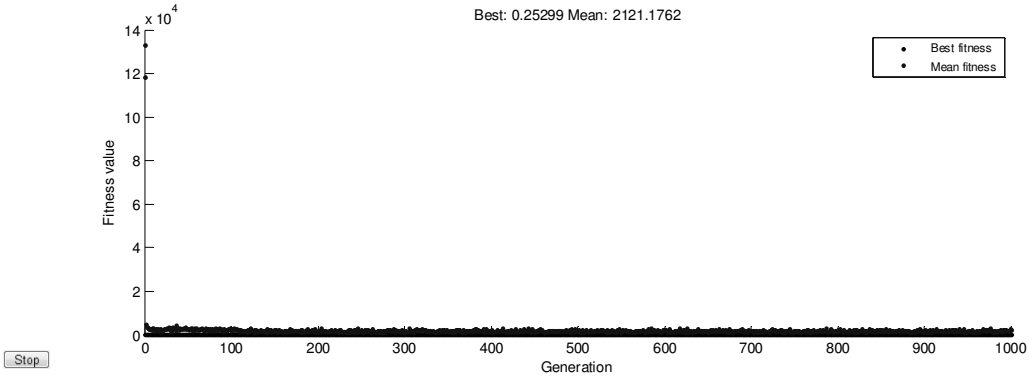

Figure 1: Case 1 Fitness value Linear Scale

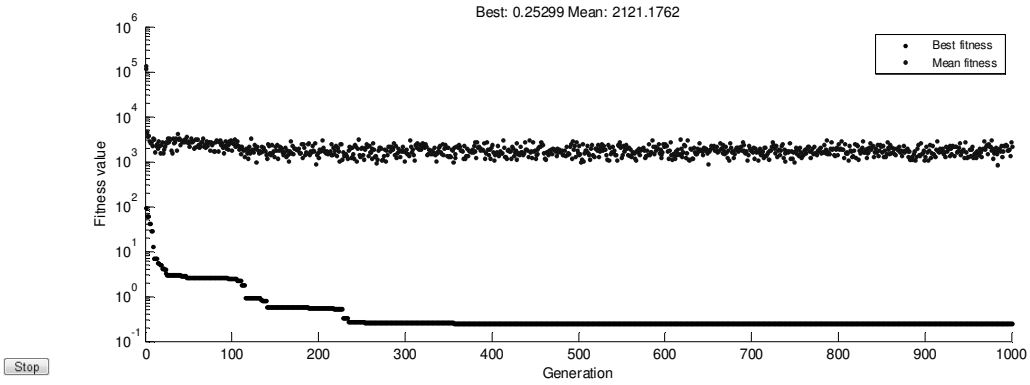

Figure 2: Case 1 Fitness Value Log Scale

Case 2:

Population Size: 1000
Generations : 300
Fitness : 0.57708

In this experiment, we reduced the number of generations to be maximum 300. Hence, the fitness value is still small relatively, but is not as optimal as the previous case. It assists in

implying that the number of generation's parameter has a small impact on the final result. The following figure (3) using Log scale shows the behavior of fitness function in finding the schema.

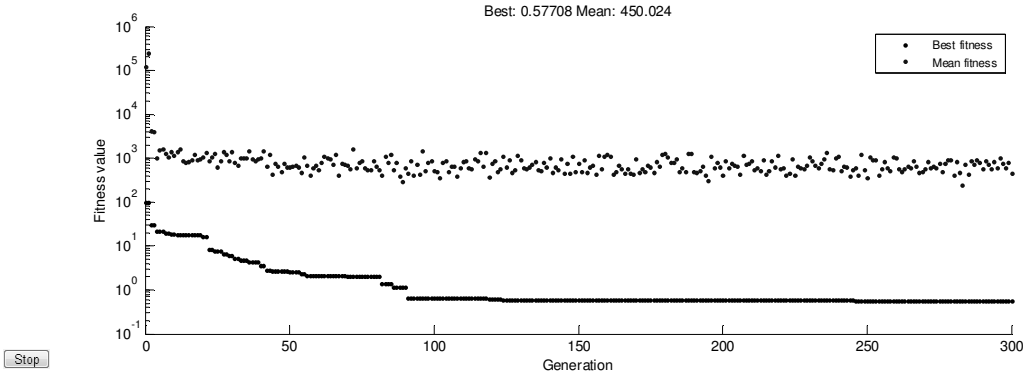

Figure 3: Case 2 Fitness Value Log Scale

Case 3:

Population Size: 1000
Generations : 50
Fitness : 0.84211

This experiment supports our claim about the small effect generation's parameter has on the final result. By reducing the number of generations to a very small number such as 50, the fitness value is higher than all of the previous two cases. It's clear from this figure, that if we extend the number of generation a little bit more, then a finer fitness value will be obtained.

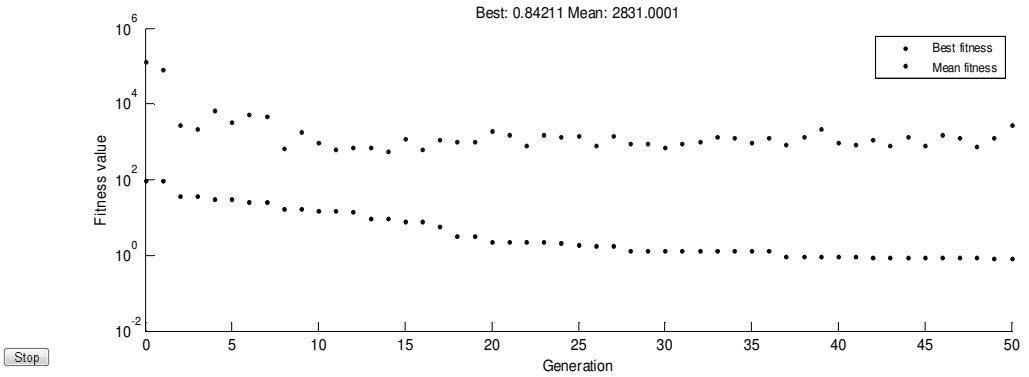

Figure 4: Case 3 Fitness Value Log Scale

Case 4:

Population Size: 200
Generations : 1000
Fitness : 2.0594

Case 4 & 5 targets the impact of the population size parameter on the solution. By having the population size reduced from 1000 to 200, the error reaches a higher value and a convergence point reaches quickly. This case and the following case prove the significant impact of Population size on the final result.

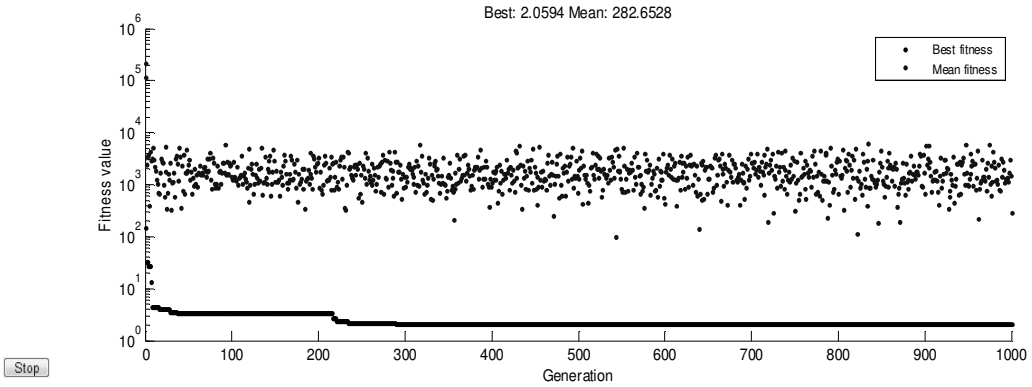

Figure 5: Case 4 Fitness Value Log Scale

Case 5:

Population Size: 50
Generations : 1000
Fitness : 4.5354

The population size is reduced significantly to only 50 random individuals. Even though the number of generations is high (1000), the algorithm reaches convergence very early at 800. That implies it has no improvement for 500 consecutive generations and as such it was stopped abruptly by the toolbox. The error is the highest than all of the previous cases. It's evident from the graph, that genetic algorithms can't gain much improvement when the population size is very small. Since the population size is very small, genetic algorithm is very restricted in finding an optimal solution and no matter how many generations are set or how other parameters behaves; the base population size contributes little to the final result.

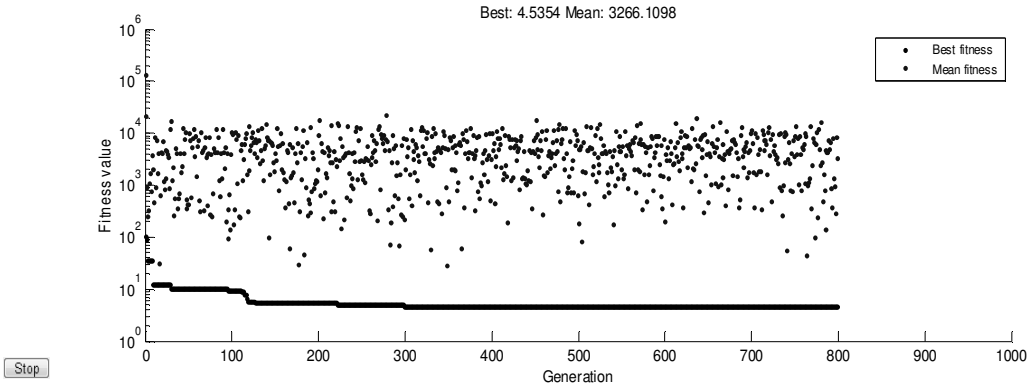

Figure 6: Case 5 Fitness Value Log Scale

Validation of the result can be done by comparing the results of the approximated function to the actual function. This type of validation is error prone and time consuming. Instead, the figures above show that the fitness function is decreasing across all the generation and this pattern is consistent in all experiments. Thus, in all the experiments above, our fitness function is trying to reach the global minimum before it reaches convergence. Hence, we can conclude that our program is running correctly and our results are valid and optimized given the parameters supplied. A summary of the result of the best case is given below where the initial population is given in Appendix A. An interesting researcher who would like to replicate the experiment

would definitely obtain the same results provided that he or she supplemented the exact same parameters values that we set.

Table 3: Results Summary

Number of Terms	9
Population Size	1000
No of Generations	1000
Fitness Values	0.2599
Individual populations :	Appendix A
Output Population (Binary):	1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0,0,1,1,0,0,1,0,0,1,1,1,0,0,1,1,1,0,1,0,1,1,1,0,1,1,1,0, 1,0,0,1,1,1,1,1,0,0,0,1,1,0,1,0,0,0,0,1,1,1,1,1,0,0,1,0, 1,1,1,0,0,1,1,1,1,1,1,1,0,0,1,0,1
Output approximated function:	$3.8 X^{3.25} - 5.4 X^{0.5} - 39.9 X^{-3.25} + 25.2 X^{2.25} + 45.4X^{-0.5} - 3.3 X^{-0.5} + 13.2 X^{-0.75} -44.2 X^{-1.25} + 13.5 X^{-2.75}$

6. CONCLUSION AND FUTURE WORK

Our attempt here of providing a robust yet fast method of finding an approximating function works correctly. Genetic algorithm and its stochastic search of optimal solution in a large complex space can save the effort and time to predict the approximated curve of any unknown engineering system by supplying the input -output pairs only.

The design of chromosome and building parts plays a significant factor in reaching an optimal solution quickly. The number of bits allocated to each building part and the decision of range values must be crafted carefully to direct the genetic algorithm to the right region where optimal solution may lay. Genetic algorithms even perceived as a successful implementation of random search, a guided direction at the initial stage of the algorithm will ensure a better results and that's can be proven in our results above. We tested our algorithm on one function only which is a squared function. Even though this might be considered as a drawback, it's very important to notice that our objective of this project not to design a universal function approximator using genetic algorithm. Instead, our main objective is to show how genetic algorithm can play a vital role in finding an optimal solution quickly and how adjusting the parameters is contributing to the results significantly. Testing the algorithm on many different polynomial and non-polynomial functions is set to be a future work plan and requires a deep understanding of mathematics to craft an excellent chromosome that absorbs all the contributable features of all mathematical equations.

7. THREATS TO VALIDITY

There are two main threats that may have impact on the results of this study. The first threat is that we used the data of one system, however, we plan to use the data of more systems in future studies.

Another threat is in data collection process. The process of collecting and analysing the data was semi-automated. This may impact the results as human error may occur.

ACKNOWLEDGEMENTS

The authors would like to thank KFUPM for all support.

REFERENCES

[1] Pati, Y.C., R. Rezaiifar, and P.S. Krishnaprasad. *Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition*. in *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on. 1993.*

[2] Shiqian, W. and E. Meng Joo, *Dynamic fuzzy neural networks-a novel approach to function approximation.* Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 2000. **30**(2): p. 358-364.

[3] Farrell, J., M. Sharma, and M. Polycarpou, *Backstepping-Based Flight Control with Adaptive Function Approximation.* Journal of Guidance, Control, and Dynamics, 2005. **28**(6): p. 1089-1102.

[4] Deep, K. and K.N. Das, *Quadratic approximation based hybrid genetic algorithm for function optimization.* Applied Mathematics and Computation, 2008. **203**(1): p. 86-98.

[5] Torrecilla-Pinero, F., et al., *Parametric Approximation of Functions Using Genetic Algorithms: An Example with a Logistic Curve*, in *Numerical Methods and Applications: 7th International Conference, NMA 2010, Borovets, Bulgaria, August 20-24, 2010. Revised Papers*, I. Dimov, S. Dimova, and N. Kolkovska, Editors. 2011, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 313-320.

[6] Hauser, J.W. and C.N. Purdy. *Designing a Genetic Algorithm for Function Approximation for Embedded and ASIC Applications*. in *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on*. 2006.

[7] Holland, J.H., *Adaptation in natural and artificial systems*. 1992: MIT Press. 211.

[8] Zainuddin, Z. and O. Pauline, *Modified wavelet neural network in function approximation and its application in prediction of time-series pollution data.* Applied Soft Computing, 2011. **11**(8): p. 4866-4874.

[9] Baqais, A.A.B., M. Ahmed, and M.H. Sharqawy. *Function Approximation of Seawater Density Using Genetic Algorithm*. in *Proceedings of the World Congress on Engineering*. 2013.

[10] Baqais, A., M. Ahmed, and M. Sharqawi. *Applying Binary & Real Genetic Algorithms for Seawater Desalination*. in *Draft*. 2016.

[11] Cao, Y.-Y. and P.M. Frank, *Analysis and synthesis of nonlinear time-delay systems via fuzzy control approach.* Fuzzy Systems, IEEE Transactions on, 2000. **8**(2): p. 200-211.

Authors

Abdulrahman Baqais is a PhD candidate at Computer Science & Engineering College, King Fahd University of Petroleum & Minerals (KFUPM), Saudi Arabia. He has obtained his Bachelor (Hons) in Software Engineering from Multimedia University, Malaysia in 2007 and his MSc from Staffordshire University, UK in 2010. His research interests including: software engineering, metaheuristics and search-based techniques and has published several articles in referred journal and conferences in these areas. He served as the session chair in IAENG conference in UK, 2013.