An Improved Genetic Algorithm Using Intelligent Symbolic Regression

Mohammed Shahbazuddin

Mechanical Engineering Department,
University of Louisiana at Lafayette

Dr. Terrence. L Chambers

Mechanical Engineering Department,
University of Louisiana at Lafayette

Abstract

In this paper, an optimization technique based on intelligent symbolic regression is presented. Intelligent symbolic regression methodology seeks to replace implicit functions of the original design optimization problem with an approximation model less expensive to evaluate. In order to address the issue of the high-computational cost of the implicit function evaluation, genetic programming methodology has been investigated to build approximation models of the best possible quality. The goal of genetic programming is to evolve mathematical expressions with no assumption about the structure, which is given as part of the solution. Once the symbolic regression model is available, the Genetic Algorithm, on subsequent runs, searches the inexpensive regression model for the global optimum, only rarely sampling the high-fidelity model. The efficiency of the proposed method has been tested on a set of standard optimization test problems and it was found to reduce the computational overhead by a significant amount without much loss of accuracy. Test results of optimization problems revealed that real–valued representation technique outperforms the binary representation technique. Such a representation is of particular concern for engineering applications, which motivated the present work.

Introduction

Optimization may be thought of in a simplistic fashion as the process of finding the best answer. The pool of potential solutions may be large, complex and computationally expensive. Optimization attempts to find the best answer for the information available. Many techniques exist from simple trial-and-error to more complex calculus-based methods. Other approaches

mimic natural phenomena such as the annealing of metals or biological evolution. A particular algorithm of this type, called the Genetic Algorithm, is the focus of this work. However, the algorithm is very slow to converge, often requiring many thousands of function calls to converge to a global optimum. This work proposes a new version of the Genetic Algorithm (GA) that improves computational efficiency by using Genetic Programming (GP) techniques to build an inexpensive symbolic regression model before it searches for the global optimum.

2.1 Genetic Algorithms

The Genetic Algorithm was first proposed as a way to model Darwin's theory of evolution or "natural selection" in nature. The Genetic Algorithm was developed into a stochastic optimization procedure by Holland [14] in the 1970s. The tendency of calculus-based methods to get stuck in local optima and the high computation cost of enumerative procedures increased the popularity of stochastic search algorithms. In recent years, the GA has found many applications in engineering design optimization [11]. In the standard algorithm, the objective function to be maximized is called the "fitness function." The input variables to the fitness function, or design variables, are usually coded as fixed-length binary strings and concatenated together into one long binary string called a "genotype," which is representative of chromosomal material in a living creature.

The Genetic Algorithm is modeled on the basis of Darwin's theory of natural selection. It uses a semi-random, or heuristic, search method to explore the design space. The initial runs of the Genetic Algorithm are used for exploring the design space and the later runs are used for exploitation or convergence. The ability to maintain a balance between exploration and exploitation makes the Genetic Algorithm less susceptible to the problems of local convergence than many other stochastic algorithms.

The Genetic Algorithm involves:

1. Creation of an initial population of individuals representing the design space.
2. Reproduction of successive generations on the basis of a fitness function.

The population size is the most important factor in the Genetic Algorithm. The population size affects efficiency as well as the computation time. A small population will lead to poor results when the number of variables is large, because the population provides an insufficient sample size for most hyperplanes. A large population is more likely to have representatives from a large number of hyperplanes. As a result, premature convergence to local minima is avoided. On the other hand, a large population requires more computation per generation, possibly resulting in a slow rate of convergence.

The initial population so created is usually encoded in a binary format called a chromosome. This chromosome then carries all the essential information about that particular individual. If a parameter (X) can vary between a minimum value *MIN* and a maximum value *MAX*, the following formula is used to determine a random value for the parameter X for each member of

the initial population:

$$X_{initial} = MIN + RandomNumber * (MAX - MIN) \quad (1)$$

To convert these real number values to binary strings, first the real numbers are converted to base 10 integer values, using the formula:

$$X_{int10} = round\left(\frac{(Xreal - MIN)J}{MAX - MIN}\right) \quad (2)$$

where,

X_{real} = real number value;

X_{int10} = base 10 integer;

$J = 2^P - 1$;

P = binary string length (say 8 or 12 bits)

For example, if $X_{real} = 3.567$, with a string of length 8, a maximum value of 10 and a minimum of 0, the base 10 integer value is:

$$X_{int10} = round\left(\frac{(3.567 - 0)255}{(10 - 0)}\right) = 91 \quad (3)$$

In binary, this value is 01011011.

Combining the encoded strings of all parameters together creates a chromosome. If there were two parameters, which in binary form were 11001010 and 10011101, the chromosome would be formed by concatenating the two binary strings together to get 1100101010011101.

The selection of the mating parents is often carried out by “roulette wheel” selection. The individuals are mapped to adjacent sectors of a circle, such that each individual's sector is equal in size to its relative fitness much like the slots on a roulette wheel. A random number is generated and the individual whose sector spans the random number is selected. This is equivalent to spinning the roulette wheel. The “wheel” is spun twice to select two individuals who will mate. Because individuals who are more fit have larger slots on the wheel, they are more likely to be chosen to mate. Mates are chosen in this fashion until the desired number of mating pairs are selected. The fitness of the function is obtained by substituting the values of variable X in the function f.

After computing all the fitness values for the population, the sum of the fitnesses is computed. This sum is denoted by "Sum" at the bottom of the Fitness column, as shown in the Table 1 below. The average fitness is obtained by dividing the sum of fitnesses by the number of parents.
This is denoted by "Average" in the table.
For example: Suppose we wish to maximize:

$f = x_1^2 + x_2^2$ such that $-2 \leq (x_1,x_2) \leq 5$, with the starting information as follows:

Variable	**Rand No**	**Real value**	**Base10**	**Binary**	**Fitness**	**Fit/Sum**	**CumProb**
$X_{1\text{-}1}$	0.503	1.521	64	1000000	2.9924	0.0506	0.0506
$X_{2\text{-}1}$	0.168	-0.824	21	0010101			
$X_{1\text{-}2}$	0.846	3.922	107	1101011	16.394	0.2773	0.3279
$X_{2\text{-}2}$	0.142	-1.006	18	0010010			
$X_{1\text{-}3}$	0.597	2.179	76	1001100	4.7491	0.0803	0.4083
$X_{2\text{-}3}$	0.281	-0.033	36	0100100			
$X_{1\text{-}4}$	0.244	-0.292	31	0011111	19.489	0.3297	0.7379
$X_{2\text{-}4}$	0.915	4.405	116	1110100			
$X_{1\text{-}5}$	0.211	-0.523	27	0011011	2.95	0.0499	0.7878
$X_{2\text{-}5}$	0.052	-1.636	7	0000111			
$X_{1\text{-}6}$	0.708	2.956	90	1011010	12.544	0.2122	1
$X_{2\text{-}6}$	0.007	-1.951	1	0000001			
				Sum	59.119		
				Average	9.8532		

Table1 Genetic Algorithm for the Function $f = x_1^2 + x_2^2$

The ratio of fitness over sum is obtained by dividing every value of fitness, in the fitness column by the sum. This value denotes the share the present fitness value has in the whole population. The cumulative probability is calculated by successively adding the fitness over sum ratio values. The cumulative probability value corresponds to the area that the present variable will have on the roulette wheel.

The mating pool is determined by drawing out six random numbers such as the following: 0.219, 0.480, 0.902,0.764, 0.540, 0.297. The range of cumulative probability within which these random numbers lie is determined, and the parent which corresponds to this range is chosen to mate.

For example: The first random number drawn is 0.219. This value lies between cumulative probability range 0.0506 and 0.3279 and this corresponds to parent 2. Hence parent 2 is chosen to mate.

Therefore, the parents chosen to mate in this example are 2 ,4, 6, 5, 4, 2. The parents are mated in the order they were drawn. Thus, 2 mates with 4, 6 with 5, and 4 with 2. Notice that parents 2 and 4 with high fitness values were chosen to mate twice, whereas parents 1 and 3 with lower fitness values were never chosen to mate.

In order to search other points in the search space, some variation is introduced into the new population by means of genetic recombination operators. The most important recombinational operator is crossover. The crossover operator exchanges the portions of two parents in the population to produce new individuals for the next generation. For example, if the crossover point is picked randomly (say 9), all the bits after the 9^{th} position in Parent 1 are replaced by all the bits after the 9^{th} position in Parent 2, and vice versa:

Before Crossover		After Crossover	
Parent 1: 11001011	00010110	Child l: 11001011	10100011
Parent 2: 01011000	10100011	Child 2: 01011000	00010110

Crossover serves two complementary search functions. First, it provides new points for further testing within the hyperplanes already represented in the population. Second, crossover introduces representatives of new hyperplanes into the population.

After producing the new generation, the encoded parameters are decoded and are substituted into the fitness function. The decoding is done with the following equation:

$$X_{real} = MIN + \left(\frac{(MAX - MIN)X_{10}}{J} \right) \qquad (4)$$

Example:

Given a chromosome for two parameters (X_1, X_2) each of string length 10:

00101101111011011100

Partitioned into:
X_1 : 0010110111 X_2:1011011100

Minimum and maximum values:
$5 \leq X_1 \leq 10$ $1 \leq X_2 \leq 25$

Base 10 integer values:

$X_{1,int\ 10}$ = 183 $X_{2,int\ 10}$ = 732

Continuous real values:
$X_{1,real}$ = 5.894 $X_{2,real}$ = 18.17

The real values obtained after crossover are then used in the fitness equation and new fitness values are computed. The sum of the fitnesses is obtained by adding all the fitness in a generation. The average is computed by dividing the sum by the number of children in the generation. The fitness over sum ratio is calculated by dividing the fitness of each child by the sum of fitnesses. The cumulative probability is calculated by adding the values in the fitness over sum ratio.

The other operator which causes variation in the population is "mutation." Mutation is the random change of a gene from 0 to 1, or 1 to 0. A random number is generated for every gene, and if the random number is greater than the mutation probability, then the gene is flipped from either 0 to 1 or 1 to 0. The mutation operator offers the opportunity for new genetic material to be introduced into the population.

When creating a new population by crossover and mutation, the best chromosome might be lost. Hence, "elitism", the procedure by which the weakest individual of the current population is replaced by the fittest individual of the previous population, is often employed. This ensures that the best design ever encountered will survive to the final generation.

2.1.1 Fitness Scaling

The fist step in the roulette wheel selection process is to scale all function values to be positive in order to assign a positive area on the roulette wheel. The following scaling scheme is used. The highest function value f_h and the lowest function value f_l are evaluated . The function values are converted to positive values by adding the quantity

$$C=0.1* f_h - 1.1* f_l \quad (5)$$

to each of the function values. Thus, the new highest value will be

$$1.1* (f_h - f_l) \quad (6)$$

and the lowest value

$$0.1* (f_h - f_l) \quad (7)$$

Each of the new values is then divided by

$$D = max(1, f_h + C) \quad (8)$$

2.1.2 Genetic Algorithm Steps

The basic Genetic Algorithm includes the following steps:

1. Start with a population of designs. These are often generated randomly. A design is coded in binary version to form a "Chromosome."

2. Determine the mating pool. This is done using a "Roulette Wheel Selection."
3. Perform "Crossover," by randomly selecting the crossover point. A new population is created.
4. Perform "Mutation" and "Elitism." Mutation is the process of introducing a random change in the generation once in a while. Elitism makes sure that the worst child in the present generation is replaced by the best parent in the previous generation.
5. Decode the population strings and substitute into the fitness function to get the new fitness values.
6. Repeat steps 2-5 until the algorithm converges to a solution. The algorithm has converged when the difference between the maximum and minimum scaled fitness is compared to a user defined convergence factor. If the Genetic Algorithm does not converge within a user-defined number of maximum generations, the elite fitness value attained to that point is returned.

2.1.3 Real Valued Crossover

A drawback of binary single-point crossover, that it involves loss of precision when converting back and forth between real and binary numbers, is eliminated by using real valued crossover. Since this conversion is eliminated in real-valued representation, it typically gives more accurate results for problems with continuous real-valued variables.
In real-valued crossover, a random number "*r*" is drawn. The offspring are obtained by the following equation.

$$X_{child1} = X_{parent1} + r^* (X_{parent2} - X_{parent1}) \quad (9)$$
$$X_{child2} = X_{parent1} + (1- r)^* (X_{parent2} - X_{parent1}) \quad (10)$$

2.2.1 Introduction to Genetic Programming:

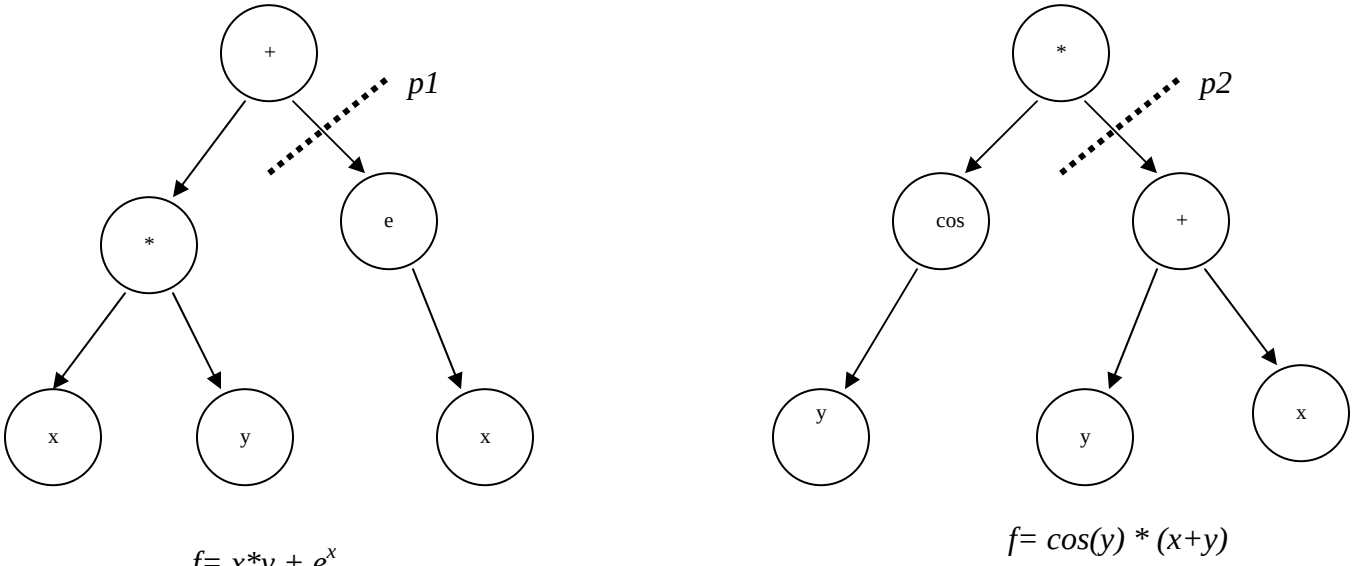
John Koza (1992) introduced the idea of Genetic Programming, which uses the Genetic Algorithm to "evolve" a computer program to accomplish a specific goal. To implement Genetic Programming, randomly generated computer programs are represented as parse trees, and tested to see how well they perform the desired goal. The best of the programs are selected to "mate" with each other and exchange genetic information, which amounts to swapping whole branches on the parse tree (from beyond a random crossover point) between mating programs. This evolutionary operator (crossover) and a few others (such as random mutation) produce changes in the population of programs. The new generation of programs is also tested to see how well each program satisfies the goal. Some of the changes will turn out to be beneficial while some will be damaging, but by imitating Darwin's principle of "survival of the fittest," the fitness of the overall population improves from one generation to the next until one of the programs achieves the desired goal. This is all accomplished without direct human intervention. One of the sample applications given in Koza (1992) is symbolic regression, which is called Automatic

Function Definition (AFD) in that book. The unique feature of AFD is that it can automatically find the best symbolic function for conventional regression. However, one limitation of AFD is that it is not well suited to find the coefficients of the regression model that are needed to fit the model to the data.

In Genetic programming randomly generated computer programs are represented as parse trees as shown above. The crossover points p1 and p2 are chosen randomly. The entire branch below crossover point p1 is replaced by the entire branch under crossover point p2. In this way new mathematical equations are generated.

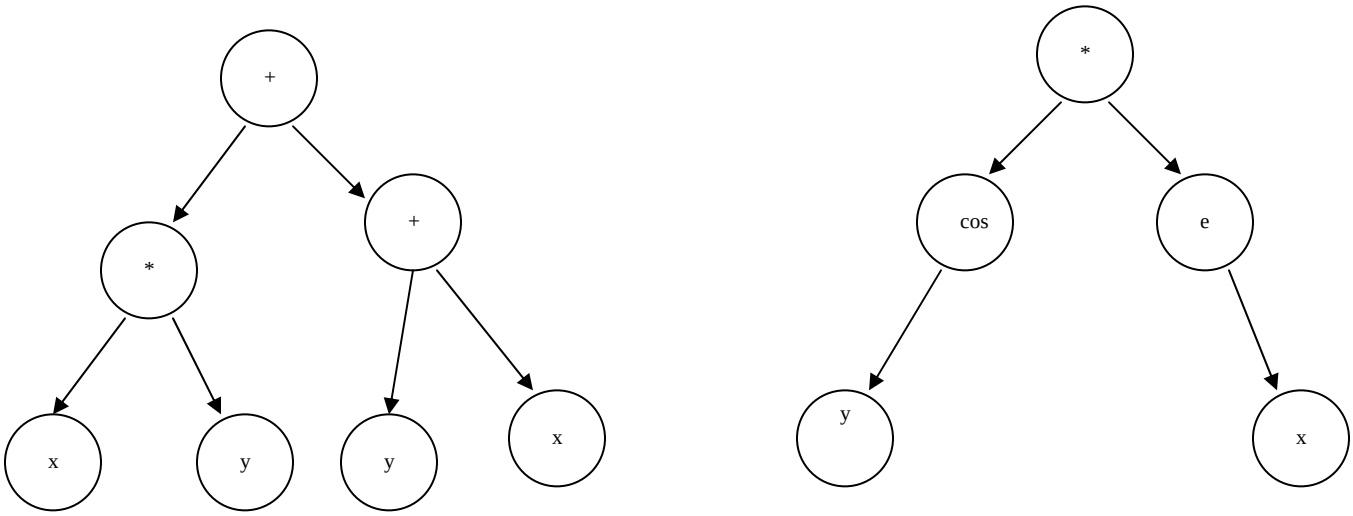
In this research, Genetic Programming is used to create multiple symbolic regression models are created. A grid size is chosen and the original function is sampled along this grid. The symbolic regression model that has the lowest least square error when compared to the real objective is selected for optimization.

An example of genetic programming is given below.



Before Crossover

Figure - 1 Genetic Programming Crossover



After Crossover

Figure - 2 Genetic Programming Crossover

Implementation Strategy

For the purpose of this research, the following strategy was used:

1. Using Genetic Programming multiple symbolic regression models are created.
2. Standard optimization techniques are used to adjust the coefficients of each symbolic regression model to minimize the least squared error between the model and the actual function.
3. The model that has the lowest error when compared to the real objective is selected for optimization .
4. The Genetic Algorithm searches for the global optimum using the inexpensive symbolic regression model, rather than calling the expensive analysis model.
5. Steps 3 will be repeated m times, or until the convergence criterion is satisfied.
6. The elite fitness and variable values will be obtained and the value of the function calculated. This value will be compared with the real objective function value to calculate the average percentage error.

3.2 Testing Strategy

The following testing strategy is proposed

1. Identify the problems to be solved by the proposed algorithm and order them in a sequence to be tested.
2. For the first two sample problems, test runs will be taken for the Genetic Algorithm using binary crossover and real valued crossover. The solution accuracy and the number of generations taken to converge will be compared. After comparison, the best representation scheme will be adopted for other sample problems.
3. Test the remaining problems based on this representation technique.
4. For every test problem run, analyze the variation in Grid Size on the average error.
5. Analyse the results using graphs and surface plots.

Discussion of Results

The developed methodology has been applied to two types of problems, the standard optimization test problems and engineering problems.

Problem Type	**Average Function Calls to Optimize Real Function**	**Average Function Calls to Create Regression Model**	**Grid Size**	**Percentage Saving in Computational Effort**	**Average Error**
DeJong's First Function	1321	100	10	1221%	2.5%
DeJong's Fourth Function	1841	81	9	2172%	1%
Rosenbrock's Function	2401	81	9	2864%	2%
2 Bar Truss Problem	1521	400	20	280.25%	5%
Spring Problem	4041	169	13	2291%	8%

Table 2 - Results of Test Problems

From the above table, it can be seen that a significant reduction in computational effort has been achieved using the proposed methodology. In the above table, only the time taken to compute the regression model is taken into account and the time taken to optimize the regression model is neglected. This is done because the time taken to optimize is fairly small as compared to time

taken to create the regression model.
Also, from the above table it can be concluded that fairly correct approximations are achieved using the proposed methodology. The error is less in case of standard optimization problems like DeJong's function and Rosenbrock's function. The error is higher in engineering problems because the constraints in these problems are converted to penalty functions which have an approximate weighting factor. When the penalty functions are less the average error is less, and vice- versa. The truss problem has four penalty functions and the average error is 5%, while the spring problem has nine penalty functions and the average error is 8%. Also, in the case of the spring problem the average error was 8% using the real objective optimization, requiring 4041 function calls. The proposed method required an average of 2480 function calls to achieve the same range of accuracy.

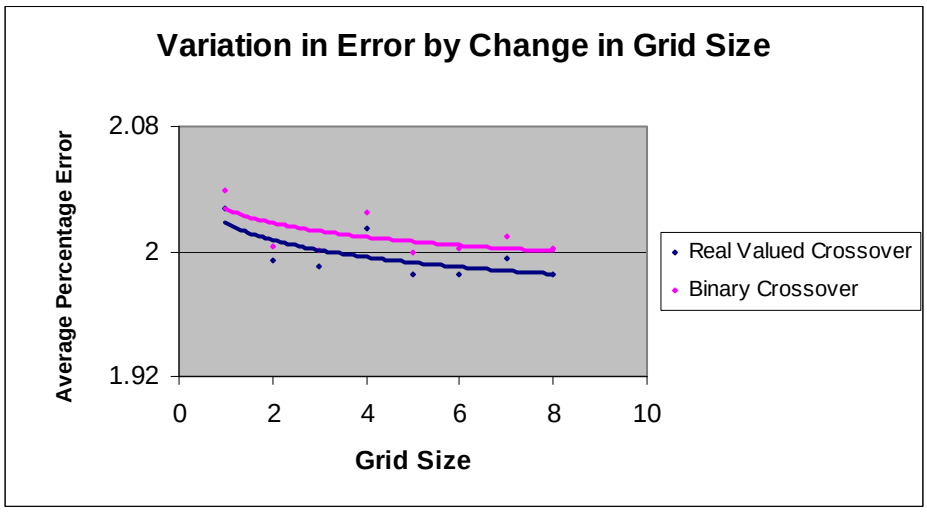


Figure 3 - Average Error Using Binary and Real Representation for Rosenbrock's Function

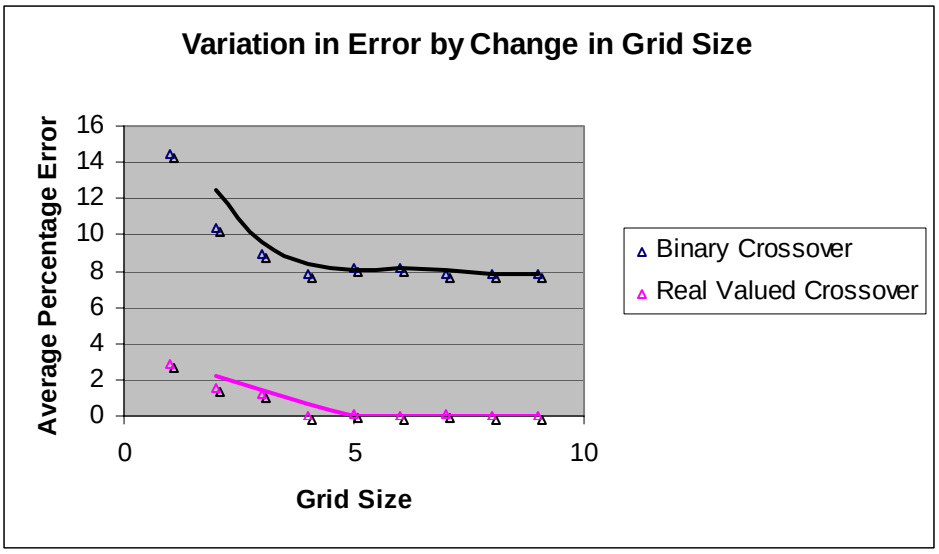


Figure 4 - Average Error Using Binary and Real Representation for DeJong's Fourth Function

The graphs in Figure 3 and Figure 4 show the advantage of using real-valued crossover over binary crossover. In binary single-point crossover, one of the design variables remains the same after crossover. Hence, it searches along straight lines. However, in real point crossover because of the random number, both the design variables are altered also, the real valued representation does not suffer from the loss of precision associated with converting back and forth between binary and real numbers. As a result real-valued crossover has far lower percentage error than binary crossover. In case of Rosenbrock's function, the average percentage error using binary representation is more that 2% while the average error using the real representation is less than 2%. For Dejong's Fourth function, the average percentage error using the binary representation is approximately 8% while the average error using the real representation is less than 1%. Hence, real- valued crossover is used in this research for solving the next three optimization problems.

Generally, a large number of points in the plan of experiments are desirable in order to provide more information to the Genetic Programming algorithm.
Grid size tests were performed ranging from a grid size of two to ten. Standard optimization problems and engineering problems were included in this test. The results in Figures 5-7 show that the higher the grid size, the better the approximation.

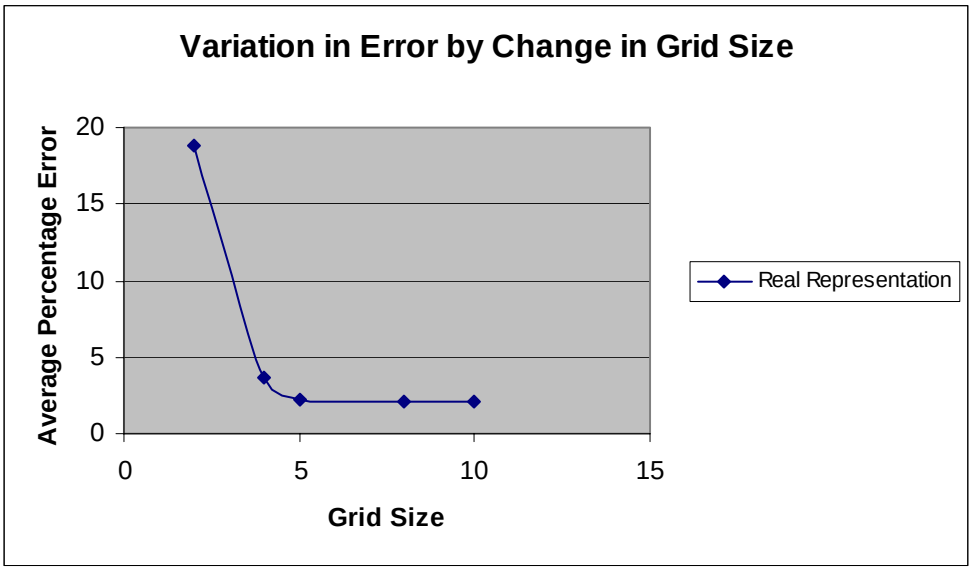


Figure 5 - Average Error Using Real Representation for DeJong's First Function

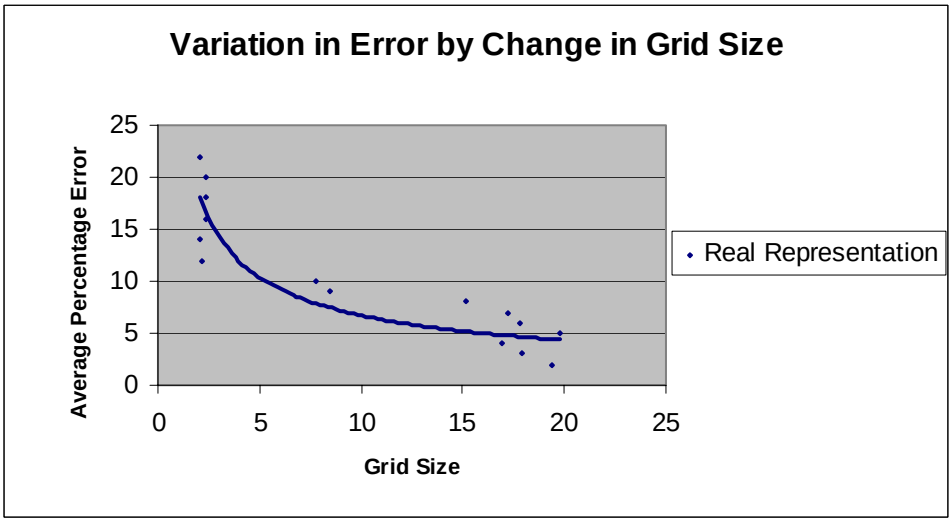


Figure 6 - Average Error Using Real Representation for 2 Bar Truss

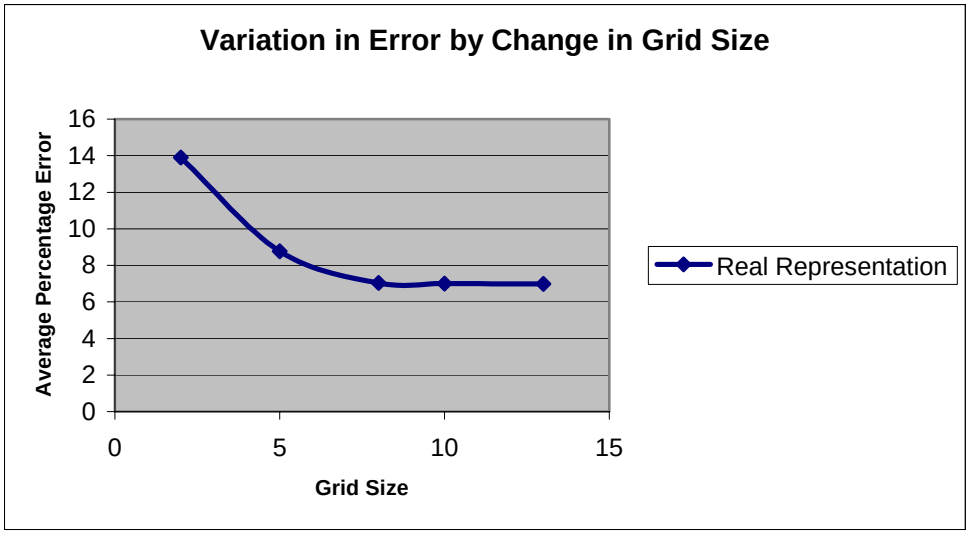


Figure 7 - Average Error Using Real Representation for Spring Problem

A history of runs is presented in the surface plots in Figure 8- Figure15. It can be clearly seen that the surface of the function becomes smoother as the size of the grid is increased. Achieving a good balance between the accuracy and computational overhead was one of the main objectives of this research work. A grid size of two does not give enough information to form a regression model. As a result, the average error is high; 22% in the case of the truss problem. Also, increasing the grid size beyond a certain extent does not further reduce the average error. A grid size of 10 gives good results with an average error of less than 1% in the case of DeJong's function. From the results, it can be fairly concluded that, in order to reduce the computational overhead without loss of accuracy, we can start with a grid size of four. If sufficient accuracy has not been achieved, the grid size can be incremented in steps of two, up to a maximum grid size of 10.

Surface Plot Showing the Effect of Variation in Grid Size for DeJong's Fourth Function

$$f = \sum_{1}^{n} ix_i^4 + Gauss(0,1)$$

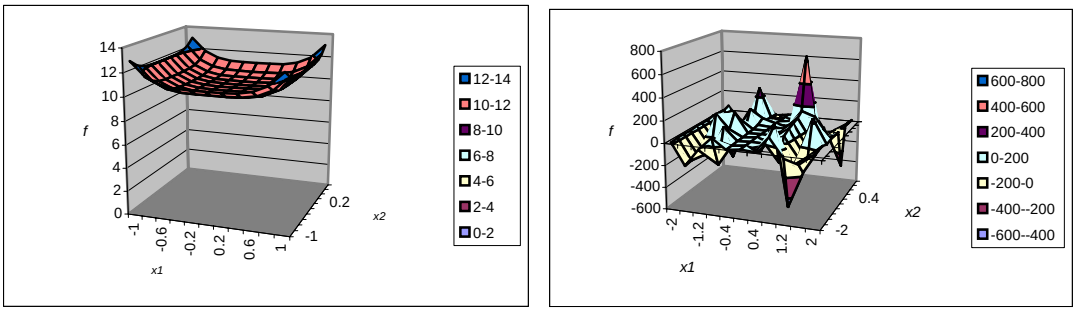


Figure 8 - Real Objective Optimization Using DeJong's Fourth Function

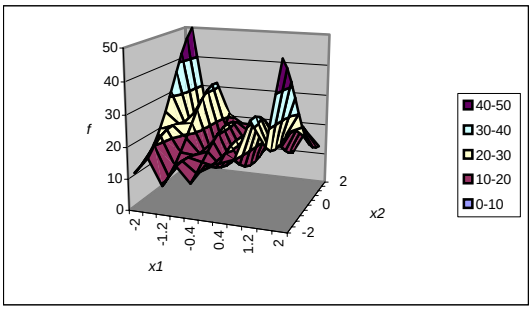


Figure 9 - DeJong's Fourth Function Grid Size = 4

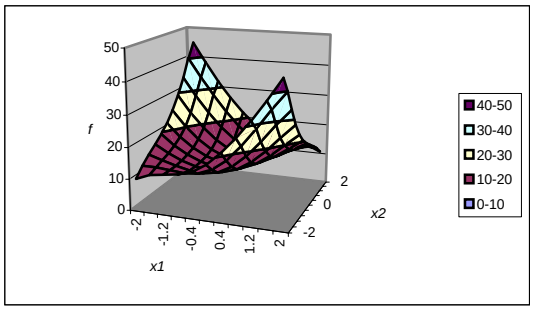


Figure 10 - DeJong's Fourth Function Grid Size = 16

Figure 11 - DeJong's Fourth Function Grid Size = 100

Surface Plot Showing the Effect of Variation in Grid Size for 2 Bar Truss

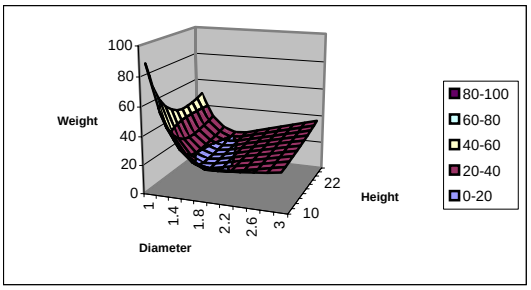


Figure 12 - Real Objective Optimization of 2 Bar Truss

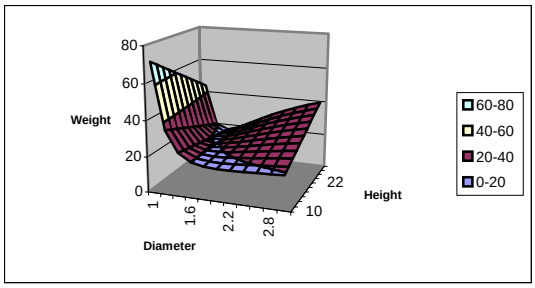


Figure 13 - Optimization of 2 Bar Truss for Grid Size = 25

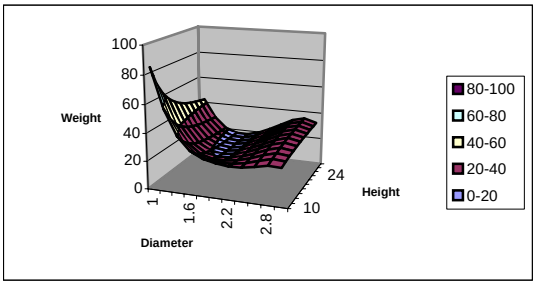


Figure 14 - Optimization of 2 Bar Truss for Grid Size = 64

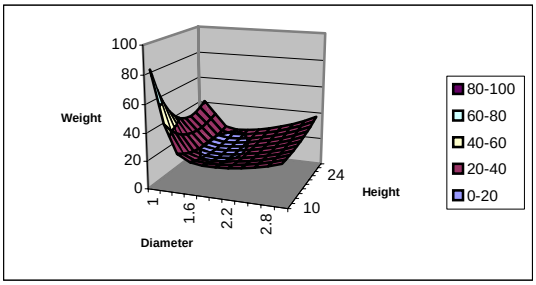


Figure 15 - Optimization of 2 Bar Truss for Grid Size = 100

5.1 Summary and Conclusions

In this thesis, an optimization technique based on symbolic regression has been presented. Intelligent symbolic regression methodology seeks to replace computationally expensive functions of the original design optimization problem with an approximation model less expensive to evaluate. In order to address the issues of the high-computational cost of the original function evaluation, genetic programming methodology has been investigated to build approximation models of the best possible quality. The goal of genetic programming is to evolve

mathematical expressions with no assumption about the structure, which is given as part of the solution. Once the symbolic regression model is available, the Genetic Algorithm, on subsequent runs, searches the inexpensive regression model for the global optimum. The efficiency of the proposed method has been tested on a set of standard optimization test problems and it was found to reduce the computational overhead by a significant amount without much loss of accuracy. Test results of optimization problems revealed that the real representation technique outperforms the binary representation technique. Such a representation is of particular concern for engineering applications, which motivated the present work.

5.2.1 Recommendations for Future Work

1. Test the symbolic regression methodology on problems with larger number of design variables.
2. Incrementally build up the grid to minimize function calls.
3. Programmatically determine the optimum grid size beyond which the average error does not fall.
4. Apply the techniques to engineering problems of extreme complexity where more traditional techniques are not applicable because of the prohibitive computational cost.
5. Improve the computational efficiency of the genetic algorithm, particularly, the evaluation of tuning parameters like the size of the population, maximum generations, crossover and mutation probabilities and the convergence factor.
6. Use other Artificial Intelligence techniques like Neural Networks for optimizing the regression model and comparing it with the Genetic Algorithm.

References

[1] Boning, D., Mozumder, P. K., 1994, "DOE/Opt: A System for Design of Experiments, Response Surface Modeling, and Optimization Using Process and Device Simulation," IEEE Transactions on Semiconductor and Manufacturing, **4**(2), pp. 233 - 243.

[2] Wang, G., Zaloom, V., Chambers, T. L., 2001, "Application of Genetic Programming and Artificial Neural Networks to Improve Engineering Optimization," *Proceedings of the International Conference on Computers and Industrial Engineering (28th ICC&IE), and the International Conference on Industry, Engineering and Management Systems (IEMS)*, Cocoa Beach, Florida, March 5-7, 2001, pp 36 - 40.

[3] Resit, R. A. Lepsch and McMillin, M.L, 1998, "Response Surface Model Building and Multidisciplinary Optimization Using D-Optimal Designs." AIAA, 98-4759. (http://techreports.larc.nasa.gov/ltrs/PDF/1998/aiaa/NASA-aiaa-98-4759.pdf)

[4] Paul, F., Zheng, W., Chun, H., Moore, M., Hsuing P. and Thomas. D, *"Optimal Pump Operation of Water Distribution Systems using Genetic Algorithms."* (http://www.rbfconsulting.com/papers/genetic_algo.pdf)

[5] Alvarez, L. F. 2000, *Approximation Model Building for Design Optimization Using the Response Surface Methodology and Genetic Programming*. Submitted for the Degree of Doctor of Philosophy, Department of Civil and Environmental Engineering University of Bradford, UK.

[6] Parkinson, A., Sorenson, C., Pourhassan, N., 1993, "A General Approach for Robust Optimal Design," Journal of Mechanical Design, **115**, pp. 74 – 80.

[7] Lewis, L., Parkinson, A., 1994, "Robust Optimal Design Using a Second-Order Tolerance Model," Research in Engineering Design, **6**, pp. 25 – 37.

[8] Box, G.E.P., Draper, N.R., 1987, *Empirical Model Building and Response Surfaces*, John Wiley, New York.

[9] Koza, John, 1992, *Genetic Programming*, MIT Press, Cambridge, MA.

[10] Dhingra, A.H. and Lee, B.H., 1994, "A Genetic Algorithm Approach to Single and Multiobjective Structural Optimization with Discrete-Continuous Variables," International Journal for Numerical Methods in Engineering, **37**, pp. 4059- 4080.

MOHAMMED SHAHBAZUDDIN

Mohammed Shahbazuddin is scheduled to receive his Master of Science in Mechanical Engineering in the Spring 2004 from the University of Louisiana at Lafayette. His research interests include Artificial Intelligence, Programming, and Engineering Optimization. He received his Bachelor of Science in Mechanical Engineering from Nagpur University, India in 1999 and Master of Science in Computer Science from the University of Louisiana at Lafayette in 2002. He is a student member of the ASME, ASEE, ISTE and MESA.

TERRENCE CHAMBERS

Dr. Terrence Chambers currently serves as an Assistant Professor of Mechanical Engineering at the University of Louisiana at Lafayette. His research interests include engineering design and optimization, artificial intelligence, genetic algorithms and genetic programming, engineering software development, and numeric and symbolic solutions to engineering problems. Dr. Chambers is a registered Professional Engineer in Texas and Louisiana.