The Use of Genetic Programming to Evolve Passive Filter Circuits

Ogri J. Ushie[1,*], Maysam F. Abbod[2], and Julie C. Ogbulezie[3]

[1,2]Department of Electronic and Computer Engineering, College of Engineering, Design and Physical Sciences, Brunel University London, Uxbridge, UK.

[1,3]Department of Physics, University of Calabar, Calabar, Nigeria.

Received 15 January 2016; received in revised form 20 March 2017; accepted 27 July 2017

Abstract

This paper introduces the use of Genetic Programming (GP), Genetic Folding and symbolic circuit analysis in Matlab for the evolution of passive filter circuits. Instead of combining MATLAB and PSPICE in electronic circuit simulation, in this work, only MATLAB is used. It helps to reduce elapsed time for transferring the simulation between the two software packages. The circuit evolved from GP using the Matlab program and is automatically converted into a symbolic netlist also by using a Matlab code. The netlist is fed into symbolic circuit analysis in Matlab (SCAM); the SCAM is used to generate matrices that are used for simulation. In this case, it is used to analyse frequency response of passive low-pass, high-pass and band-pass filter circuits. The algorithm is tested with four different examples and the results presented have proved that the algorithm is efficient concerning the design wise. The work has provided an alternative way of using GP for the evolution of passive filter circuits.

Keywords: genetic folding, genetic programming, netlist, passive filter circuits, symbolic circuit analysis in Matlab

1. Introduction

The physical interpretation of the world is analogue in nature which makes analogue circuits very important in circuit design. Although the amount of digital circuit design is more than that of analogue design, most digital designs require analogue modules for interfacing to the external world. Instead of combining Matlab and PSpice in electronic circuit simulation, in this work, only Matlab is used which helps to reduce elapsed time for transferring the simulation between the two software packages. The circuit evolved from genetic programming (GP) by using Matlab program and is converted into a symbolic netlist automatically by using genetic folding (GF) coded in Matlab. The netlist is then fed into symbolic circuit analysis in Matlab (SCAM); the SCAM is used to generate symbolic matrices that are used for the simulation. In this case, it is used to analyse frequency response of passive filter.

In this paper, GP is used to evolve electronic circuits based on the initial specification set by the user; examples are given for the design of passive filters where the cut-off frequency and the attenuation sharpness are used as the objectives for designing the filter.

Evolvable hardware (EHW) is a field in the evolutionary algorithm used in electronic circuit design without manual engineering design. It combines fault tolerance, autonomous system, artificial intelligence and reconfigurable hardware. Some of EHW applications in electronic circuit design are discussed by different researchers in [1-6] and related filter design is presented in [7, 8]. In addition, some automatic syntheses of analogue circuits are described in [9-11].

GP has been used widely in a number of applications: GP is used to optimise non-functional properties of programs such as speed, power consumption, throughput, bandwidth and size by constructing the Pareto program surface [12]. Also, GP application in automated software repair has been shown by Forrest et al. and Weimer et al. [13, 14] and showing its usage to

* Corresponding author. E-mail address: ogri.ushie@unical.edu.ng

automatically finding patches [15]. In addition, Jason and Silvano have used evolutionary search technique for circuit design automation. The main limitation in the presentation is the inherent restriction on circuit topologies [16].

The concept of GP was introduced by Koza in 1992 for automatic analogue circuit evolution which solve complex circuits compared to GA. Also, Koza illustrated 76 examples of results using GP to be competitive with human-produced methods [17-20]. Besides, a basic introduction to genetic programming is documented in [21]. Also, the use of current – flow analysis and GP for the invention of CMOS amplifier is introduced in [22]; the paper explains how current-flow analysis corrects and screens circuits using topology-independent design rules. The technique is designed to handle connections between transistors. Furthermore, a genetic programming toolbox for MATLAB and GP algorithm is presented in [23, 24]. In addition, Hou et al. [25] presented GP based on the tree representation for passive filter synthesis and the results presented show that the method can generate both economical and compliant passive filter circuits. The paper also specifies how the authors intended to add more design objectives such as component value sensitivity and group delay variation to be considered in their future work. Chang et al. [26] used the same approach as Hou et al., but claim that his method is better regarding efficiency compared to traditional method and faster than previous work. Also, similar tree representation approach in circuit design is illustrated by Senn et al. the authors combined GP and two-port theory for analogue circuit synthesis. The model of a circuit as the two-port network makes it straight forward for evaluation and encoding of their circuit's structure and is used for passive and active (transistor) linear circuits [27]. An alternative approach for data modelling using genetic folding as an evolutionary algorithm for support vector regression is in [28]. Hou et al. presented GP based on the tree representation for passive filter synthesis [25], the approach in this paper use GP, GF and SCAM that make all the simulations possible in Matlab software.

2. Methodology

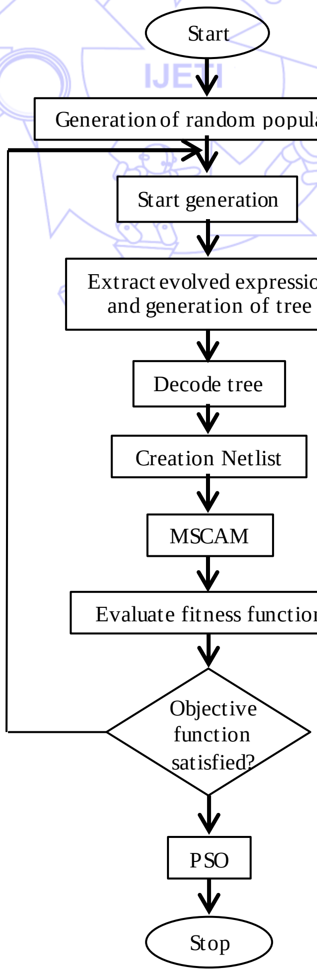
Start
Generation of random population
Start generation
Extract evolved expression and generation of tree
Decode tree
Creation Netlist
MSCAM
Evaluate fitness function
Objective function satisfied?
PSO
Stop

Fig. 1 The GP algorithm

The flowchart is shown in Fig. 1 summarises the method used in this work. The randomly generated population is evaluated to ascertain how well each individually evolved circuit performed regarding the objective function. If the evolved circuit satisfies the objective with zero error, the program ends otherwise generation starts. The evolved circuit from the generation is extracted and converted to a symbolic netlist by application of genetic folding coded in Matlab. The netlist is then fed into the SCAM program that generates the matrices used for the formulation of a fitness function. The processes continue until zero error is obtained or the objective function is satisfied. The detailed procedure involved is explained below the flowchart.

2.1. Genetic Programming

Genetic Programming (GP) is one of the concepts in the research area of Evolutionary Computation (EC). It originated from the genetic algorithm (GA) created by John Koza. The major difference between the GA and GP is that: GA is represented by a fixed length of numerical strings, whereas GP is represented by variable length structures containing whatever elements are needed to solve the problem. The Tree Structure (TS) in GP population is used to create neural networks, determine designs for analogue electric circuits and parallelise computer programmes. The TS is great because it can produce solutions of complexity and arbitrary size, as opposed to GA with fixed-length. Genetic Programming (GP) not the same as Geometric Programming (GP) which is the type of mathematical optimisation characterised by constrained and objective function that have a special form [29]. It has been used successfully in a different number of applications: biology and bio-information, arts and entertainment, medicine, control, time series prediction, image and signal processing, modelling and regression. In GP, a population is randomly created and each individual in the population is evaluated to ascertain its fitness that serves selection criteria. The best individual is picked and reproduced, crossover or mutated with other individuals to produce new individuals for the next generation. The GP algorithm; according to Koza [20], is based on the three steps:

1. Generate a random population composed of the original function and termination criteria for the problem.
2. Perform the following sub-steps iteratively until the termination criteria are reached:
   (a) Each program in the population is executed such that a fitness measure that specifies how well the problem is solved is clearly formulated.
   (b) New population is created by selecting individual(s) with probability based on fitness and then these operations are applied:
      (i) Reproduction: Copy existing individual to the new population.
      (ii) Crossover: Two individuals are created for the new population by randomly recombining chosen parts of two existing individuals.
3. The single best individual in the population produced during the run is taken as the result.

The GP algorithm discussed above is applied to evolve the low pass passive filter circuits with the combination of automatically generated netlist, SCAM and GF.

2.1.1. Initialisation of parameters

The following parameters are initialised: Length of parameters (total number of parameters in tree structure) = 8191, population size = 100, crossover rate = 0.90, mutation rate = 0.10, length of chromosome = Length of parameters by bit group (8191 $\times$ 2 = 16382. These are the initial values that give the best results. After initialisation of parameters, the population is randomly generated of size equal to population size multiplied by the length of the chromosome.

2.1.2. Decoding

The string is coded into a circuit. In this case, the chromosome is divided into a bit group of two, and each is converted to its equivalent decimal. The decimal equivalent is interpreted as:

'0' represents capacitor (X), '1' represents inductor (Y), '2' represents series part (+) and '3' represents parallel part (|)

The input voltage, input and output resistances are fixed since their values are not changed whereas the evolutionary process is carried out.

2.1.3. *Creation*

A tree is randomly generated using the operands (terminals, in this case C and L) and operators (operators, in this case + and |) defined in section 2.1.2 above. It is better to begin with many trees of different shapes and sizes. Trees are generated using the grow or the full technique:

- Grow – path lengths in the tree will vary up to the maximum length.
- Full – all branches in the tree must reach the maximum depth.
- Ramp half – and - half technique – trees of varying depths from the minimum to the maximum depth. Half of the trees are initialised with full and the other with grow. The ramp half - and – *half is used.*

2.1.4. *Mutation*

Pick a mutation point in one parent and swap its subtree with a randomly generated tree. In this work, the mutation rate of 0.1 is used.

2.1.5. *Crossover*

Pick crossover points in both parents and exchange the subtrees. The offspring will be different even if the parents are same. The crossover rate of 0.9 is used. Tournament strategy is used to select two individuals from the present population, and the ten randomly selected subtrees of the parents are swapped to create two offspring.

2.2. *Genetic Folding*

Genetic Folding (GF) is a class of evolutionary algorithm based on numbers of genes structurally organised in order of linear numbers separated by dots [30]. In this research, the GF was used to show how the chromosome is structurally linked from beginning to end so that the circuit can be extracted to generate the netlist. For better understanding of GF, Eq. (1) is used for illustration. Fig. 2 shows its GP representations whereas its GF representation is shown in Table1.

$$y^2 + y - 3 \tag{1}$$

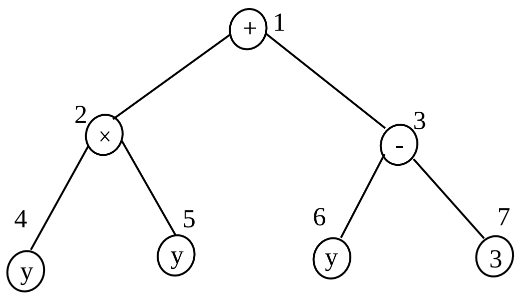

Fig. 2 Tree representations

The tree structure expression looks like (from top to down and from left to right). The GF equation is in Eq. (2).

$$+\times\text{-}yyy3 \tag{2}$$

The expression in Eq. (2) is read as follow: the plus operator is a two operands operator with two values (multiplication, minus). The multiplication operator is a two operands operator with values (y, y), the minus operator is a two operand operator with values (y, 3). The expression is represented using GF in Table 1. Each element is given a position number in order as in the first row, the elements are in the second row and the elements are folded over their complementary position genes in the third row.

Table 1 GF representation of Fig. 2

1	2	3	4	5	6	7
+	×	-	y	y	y	3
2.3	4.5	6.7	0.4	0.5	0.6	0.7

The GF starts with plus operator in position 1 and terminates with element ‘3’ in position 7. The plus operator has the multiplication operator in position 2 and minus operator in position 3. The multiplication operator has terminals (y, y). The minus operator has terminals (y, 3). The terminals are represented using their indices position. GF is summarised with the following points:

- The chromosome arrangement comprises of float string in the gene and the position of the gene.
- The gene arrangement is left child (LC) side separated by dot and right child (RC) side.
- The dot mean and.
- The operator with two operands has LC and RC.
- The operator with one operand has LC and 0 in the RC.
- The terminals have 0 in LC and value in the RC.

2.3. *Creation of Netlist*

To evaluate how well an individual (evolved circuit) has performed in the population regarding the objective function, the circuits are extracted and represented in the form of a symbolic netlist. Initial variable representations in the Matlab code are single variable defined in section 3.1.2 before they are encoded into individual component type. Furthermore, a particular component type is encoded with different subscripts to differentiate them if they are more than one in a circuit. All these encodings are done symbiotically. Using one component type for an illustration, all capacitors are represented by X. Assuming there are ten X (ten capacitors), they are being replaced by [‘a-j’] so that if an element is picked and is ‘a’ it is assigned as C_1, if another element is picked and is ‘b’ it is assigned as C_2 and so on. The evolved circuits are presented in the form of a tree structure. A branch is terminated with the operand (capacitor and inductor) whereas the tree continues with an operator (series or parallel part). It is read from the left to the right and from the top to the bottom. The branches after the operand branches are represented by ‘0’. Likewise, the branches after the ‘0’ branches are also represented by ‘0’ so that all the branches after the operands branches are represented by ‘0’ up to the maximum length. All the ‘0’ elements are then removed to leave only the evolved circuit.

Separation evaluation using the stack is applied to arrange the circuit as it is connected. The series (‘+’) is numbered from 0 to the highest number series part while the parallel part are all numbered 0 as all are connected to the ground. The components are also numbered with subscript from 1 to the last e. g. a capacitor of 10 in a circuit is numbered as C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9 C_{10}. The netlist is formed, thus, if a component is picked; it is between the first node and the second node. It is important to note that, a series path is always connected to the next number that is not zero. For instance, if the extract from evolved circuit is in Eq. (3):

$$+V+R_S+L_1|C_1+C_2+R_L \tag{3}$$

Substituting the values of series and parallel form Eq. (4)

$$0V1R_S2L_10C_13C_24R_L \tag{4}$$

The netlist is formed symbolically as follow:

It begins with the element name, followed by node1, node2 and then the component value as illustrated in Table 2.

Table 2 Netlist formation interpretation with respect to Eq. (4)

V 0 1 component value	R_S 1 2 component value	R_L 4 0 component value
L_1 2 3 component value	C_1 0 3 component value	C_2 3 4 component value

The tree interpretation (Fig. 3) whereas its equivalent circuit representation is in Fig. 4.

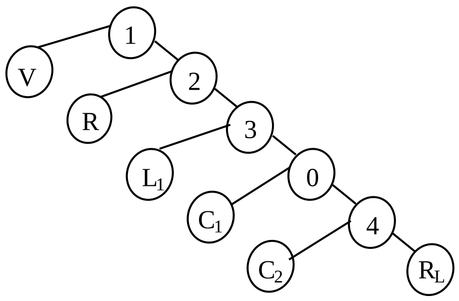

Fig. 3 Equivalent tree representations

The circuit representation is as:

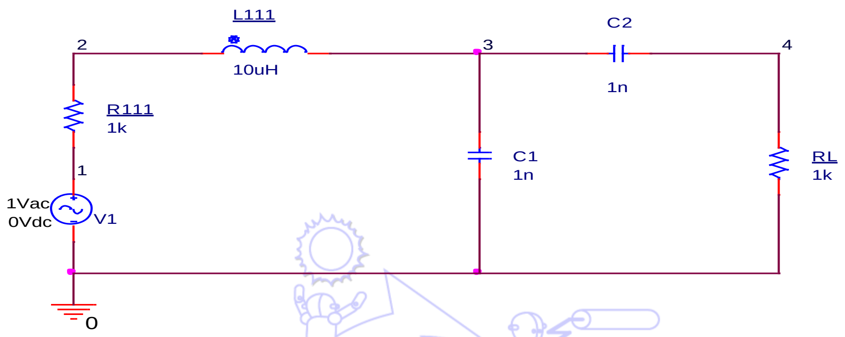

Fig. 4 Equivalent circuit representations

The symbolically generated netlist is then fed into symbolic circuit analysis that helps to generate symbolic matrices. The matrix generated is then used to formulate the objective functions to compute the fitness of an individual.

2.4. Symbolic Circuit Analysis in Matlab

According to Gielen and Sansen, symbolic simulation can aid in automatically creating a large part of analytical prototype of a circuit required to size that circuit in design system automation [30]. Symbolic circuit analysis in Matlab (SCAM) is a tool that makes use of a symbolic netlist formed above to generate matrices. These matrices can be used for optimization or to analyse circuit parameters. The tool can handle passive and active components such as capacitors, resistors, inductors, transistors, and operational amplifier. The transistors are transformed into small signal analysis, and operational amplifiers applied the small signal analysis that is easy to be implemented in a programme as explained in [31, 32]. In the research, the SCAM is employed to take in automatically generated a netlist from the simulation that is symbolic, applies it to generate matrices that are also symbolic before substituting their real values to get frequency response and then compares it to the set frequency response. The process continues until the set frequency is achieved.

2.5. Specifications of the objective function

Frequency response (voltage gain) is used to calculate the RLC circuit.

(a) The frequency range between 1 Hz to 1 GHz is specified as follow: 1 Hz - 1 GHz is specified for example 1 circuit with a cut-off frequency of 1MHz1 whereas Hz - 1 MHz with a cut-off frequency of 1.5KHz1 is specified for example 2 and so on. The impedance of each operand node is calculated using three parameters: the component value, component type and the given frequency.

(b) Each operand node returns impedance upward. The operator node performs the corresponding arithmetic (i.e., series and parallel) to obtain its impedance after receiving impedance from its children and the process continues until the circuit impedance is computed.

(c) The source voltage is divided by the circuit impedance to obtain the current flowing in the tree. Starting from the source, the current flowing in the series node is equal to the current flowing in from the source. While in the parallel node, the current flowing in is divided inversely proportional to the children's impedance and continues till the node end.

(d) The node voltage is obtained by multiplying the impedance by the current, and the voltage gain is obtained by dividing the potential difference drop across the specified output node by source voltage. The process is illustrated mathematically below.

Taking the circuit from Example 2 for illustration, the symbolic matrices A in Eq. (1) and B in Eq. (2), generated from SCAM can be used to formulate the fitness function as:

$$A = [(V_S \div R_{111});\ 0;\ 0;\ 0;\ 0;\ 0] \tag{5}$$

where A is a column matrix with fist element having a voltage source (V_S) divided by input resistance (R_{111}).

$$B = \begin{bmatrix} b_{11} & b_{12} & 0 & 0 & 0 & 0 \\ b_{21} & b_{22} & b_{23} & 0 & 0 & 0 \\ 0 & b_{32} & b_{33} & b_{34} & 0 & 0 \\ 0 & 0 & b_{43} & b_{44} & b_{45} & 0 \\ 0 & 0 & 0 & b_{54} & b_{55} & b_{56} \\ 0 & 0 & 0 & 0 & b_{65} & b_{66} \end{bmatrix} \tag{6}$$

where B is a square matrix with individual elements defined as

$$\begin{gathered} b_{11} = 1 \div R_{111}, b_{12} = -1 \div R_{111},\ b_{21} = -1 \div R_{111},\ \omega = 2 \times \pi \times freq,\ s = j \times \omega,\ b_{22} = 1 \div R_{111} + 1 \div L_{111} \times s, \\ b_{23} = -1 \div L_{111} \times s, b_{32} = -1 \div L_{111} \times s,\ b_{33} = C_1 \times s + 1 \div L_2 \times s + 1 \div L_{111} \times s,\ b_{34} = -1 \div L_2 \times s,\ b_{43} = -1 \div L_2 \times s, \\ b_{44} = C_2 \times s + 1 \div L_2 \times s + 1 \div L_3 \times s, b_{45} = -1 \div L_3 \times s, b_{54} = -1 \div L_3 \times s, b_{55} = C_3 \times s + 1 \div L_3 \times s + 1 \div L_4 \times s, \\ b_{56} = -1 \div L_4 \times s, b_{65} = -1 \div L_4 \times s, b_{66} = C_4 \times s + 1 \div R_L \times s + 1 \div L_4 \times s \end{gathered} \tag{7}$$

where the variables in Eq. (7) are defining individual corresponding variables of the matrix in Eq. (6)

$$C = B^{-1} \times A \tag{8}$$

where C in Eq. (8) contains unknown voltages in all the nodes to be determined. C_6 is the voltage across the load resistor being analysed to get its frequency response within a certain range of frequency specified above. To substitute the values of variables in the automatically generated matrices (symbolic matrices A and B) the eval command in Matlab is used. The logspace command in Matlab is used to specify a frequency range of 50. The specification is done for both evolved circuits using GP and the targeted circuit. The difference between the root mean square (rms) value of the targeted frequency response and the GP evolved circuit frequency response is the error that controls the GP toward the desired circuit specifications. The relationship is in Eq. (9), as follow:

$$W = rms(f_1 - f_2) \tag{9}$$

where W is the error, f_1 is the targeted or set frequency response, and f_2 is the GP evolved circuit frequency response.

3. Existing Mechanism

Genetic programming has been in existence but it normally combines software packages (for example MATLAB and PSPICE) for electronic circuit evolution. In this work, only MATLAB is used in the simulation with the introduction of SCAM, automatically generated netlist and genetic folding. It helps to reduce elapsed time for transferring the simulation between the two software packages. The circuit evolved from GP using the Matlab program and is automatically converted into a symbolic netlist also by using a Matlab code. The netlist is fed into symbolic circuit analysis in Matlab (SCAM); the SCAM is used to generate matrices that are used for simulation. The results are then compared to that simulated from PSpice.

4. Results and Discussion

4.1. Algorithm Benchmark Testing on Mathematical Functions

To test for reliability, validation and efficiency of optimisation algorithm are often performed using a benchmark. Benchmark function is very vital to validate and compare the working of optimisation algorithms, specifically for newly developed ones [33]. For the new GP algorithm developed, it is vital to validate its performance by using existing set of test function. The requirements of a benchmark according to Feldt et al. [34] are:

- Validity: mistakes that invalidate the required output would be avoided,
- Comparability: findings should be contrasted to other studies' findings.
- Reproducibility: experiment and problem should be well documented to enable other researchers to reproduce the same solutions for a given problem.

4.1.1. Benchmark Testing Expression

$$Z' = X^3Y^2 + 3X^2Y + Y^2 - 4X + 7 \tag{10}$$

Benchmark testing expression in Eq. (10) was used to test the newly developed algorithm. Both *X* and *Y* is given a range of values from -50 to 50 with an interval of 1 and its three-dimensional plot that is the same as that of original expression is represented in Figure. The GP algorithm evolved the expression with optimal solution in the 52nd iteration with zero errors and the GP TS is in Fig. 10 and the plot of errors against generations is shown in Fig. 11. The error plot shows how the errors decreased from first iteration to the iteration that gives the required solution and further run for additional 20 iterations with a constant error.

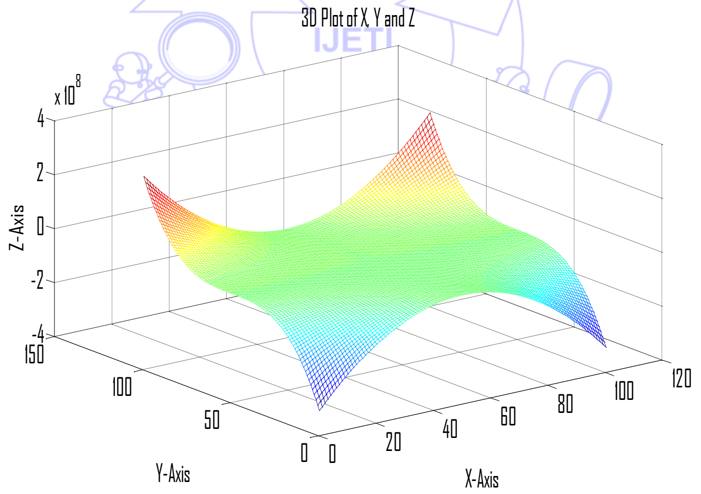

Fig. 5 Three-dimensional plots for expression in Eq. 10 for the 52nd iteration with zero error and the same as original expression

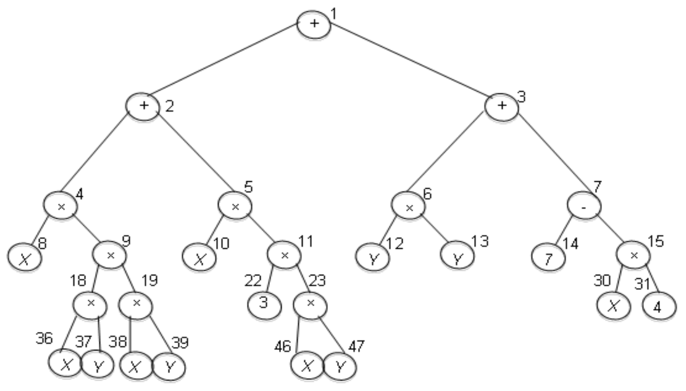

Fig. 6 52nd iteration GP evolved TS for expression in Eq. 10 with zero error

Careful evaluation of the evolved TS of Fig. 6 gives the expression simplified as:

$$Z = X(X \times Y) \times (X \times Y) + X(3) \times (X \times Y) + (Y + Y) + 7 - (X \times 4)$$

$$Z = X(X^2Y^2) + 3X(XY) + Y^2 + 7 - 4X$$

$$Z = X^3Y^2 + 3X^2Y + Y^2 - 4X + 7$$

From the analysis of the TS we can infer that the algorithm is efficient as it has successfully evolved the expression.

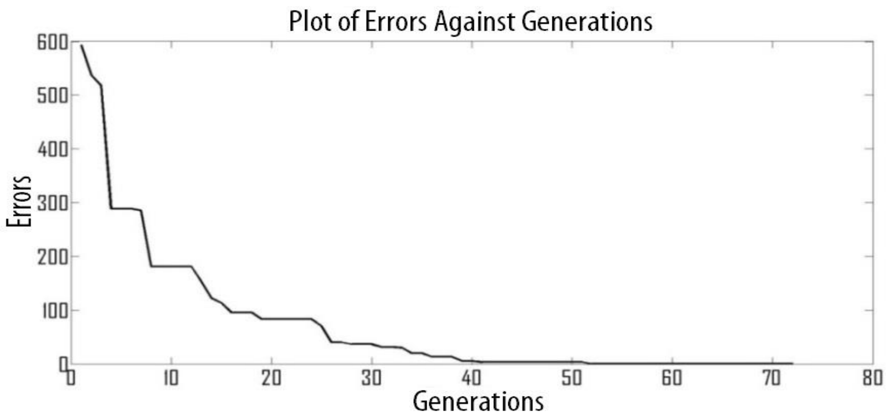

Fig. 7 Plot of errors against generations for expression in Eq. 9

Two different low-pass, one high-pass and one band-pass passive filter circuits are used to demonstrate the efficiency of the algorithm. The algorithm has proved to be efficient as it successfully evolves the four circuits with zero error. The results are the same as the set frequency response specified in the objective function. To get different sets of circuit's components values for the same circuit, the PSO algorithm is used to optimise the component values. The component values are given a range of values while PSO picked value within a given range as illustrated in respective tables of the circuits.

4.2. *Example 1:10th Order Low-Pass Passive Filter Circuit*

In example 1, the expected or desired circuit GP representation is shown in Fig. 8 whereas the equivalent circuit representation is Fig. 9. It takes fifty-six minutes to evolve the circuit after twenty-eight (28) iterations. The SCAM frequency response of the evolved circuit simulation is shown with the blue colour (solid) whereas the equivalent circuit PSpice simulation is shown with the black colour (dash) in Fig. 10. The desired circuit specifications are achieved as regard design and frequency response curve. The MSCAM and original circuit specifications are with cut-off frequencies at 1.07 MHz with zero error and a gain of 1.

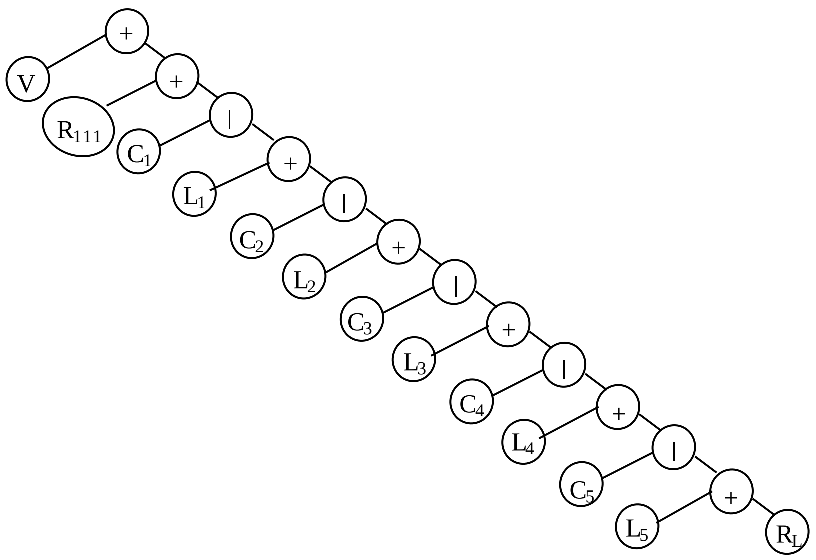

Fig. 8 Example 1 evolved circuit tree representations

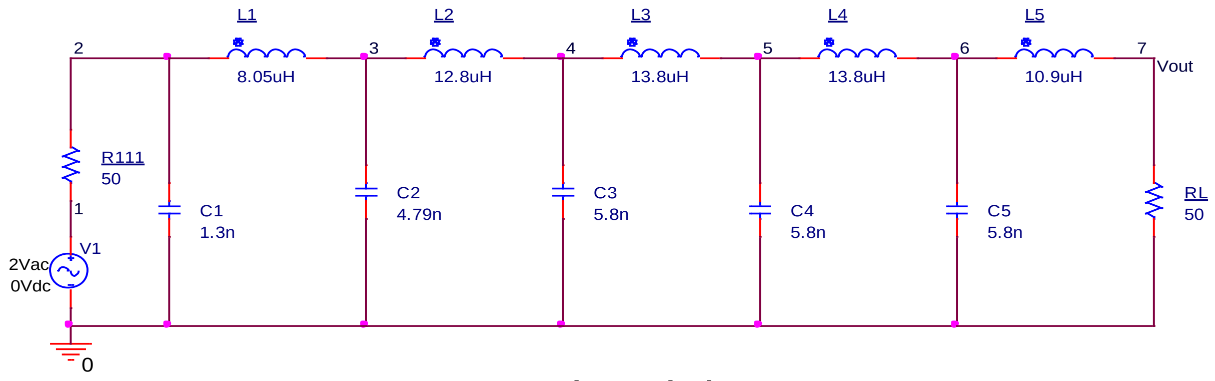

Fig. 9 Example 1 evolved circuit

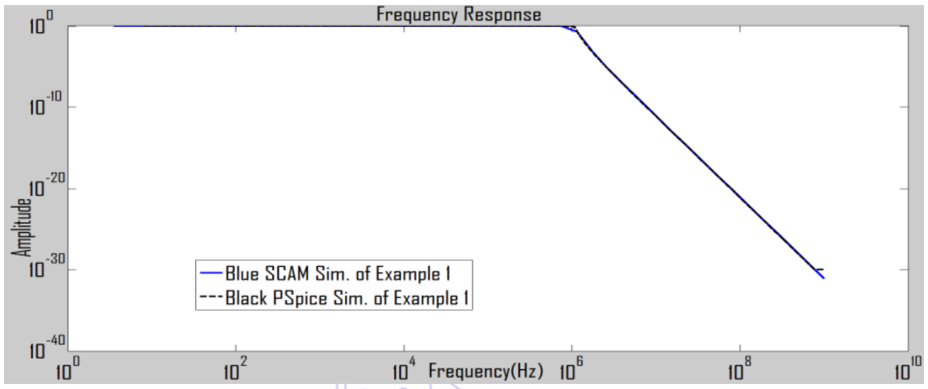

Fig. 10 Example 1 frequency response curve for SCAM (blue) and PSpice (black)

4.3. Example 2: Low-Pass Passive Filter Circuit

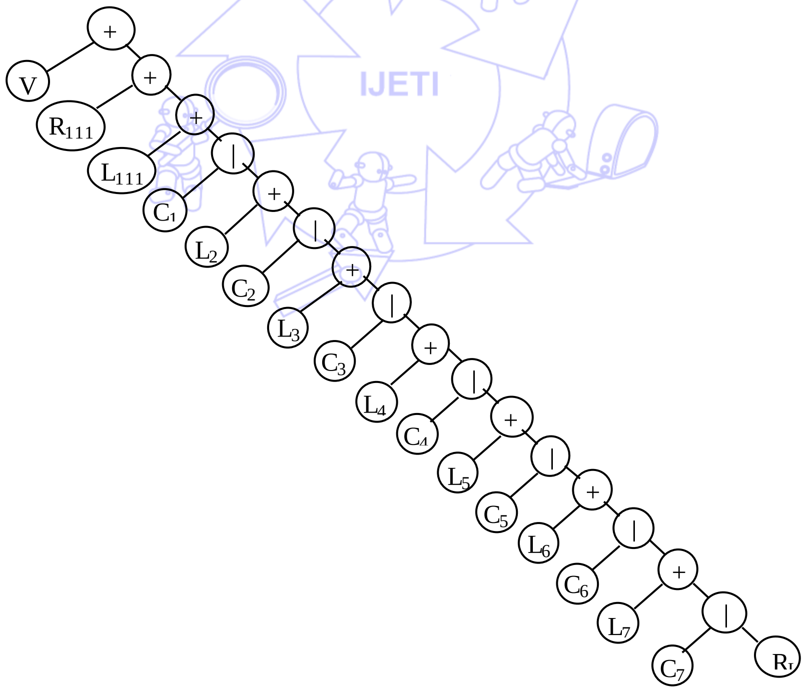

Fig. 11 Example 2 evolved circuit tree representations

In example 2, the expected or desired circuit GP representation is shown in Fig. 11 whereas the equivalent circuit representation is in Fig. 12. It takes three hours and twenty-four minutes to evolve the circuit after one hundred and two (102) iterations. The SCAM frequency response of the evolved circuit simulation is shown with the blue colour (solid) whereas the equivalent circuit PSpice simulation is shown with the black colour (dash) in Fig. 13. The desired circuit's specifications are achieved. The MSCAM and original circuit specifications are with cut-off frequencies at 1.489 kHz with zero error and a gain of 1.

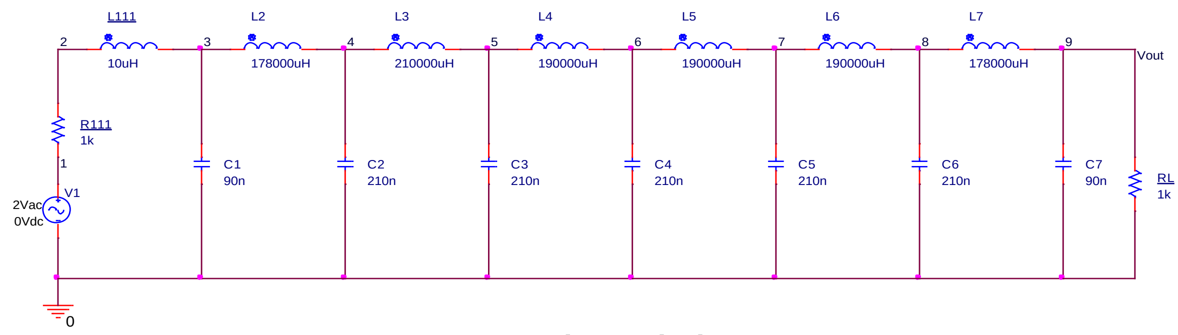

Fig. 12 Example 2 evolved circuit

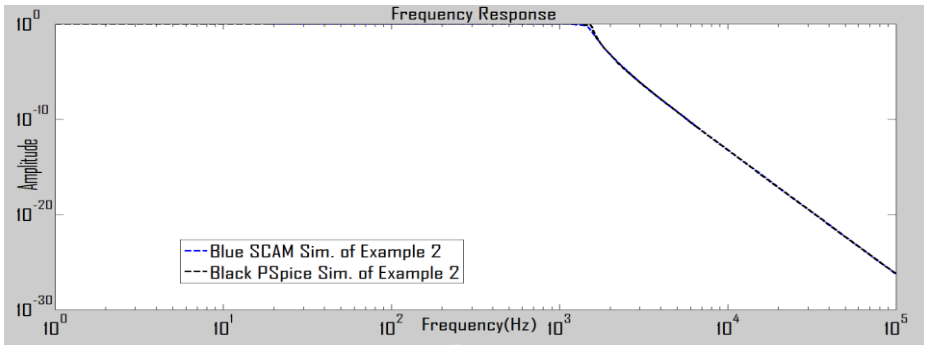

Fig. 13 Example 2 frequency response curve for SCAM (blue) and PSpice (black)

4.4. *Example 3: High-Pass Passive Filter Circuit*

In example 3, the expected or desired circuit GP representation is shown in Fig. 14 whereas the equivalent circuit representation is in Fig. 15. It takes thirty-eight minutes to evolve the circuit after one hundred and two iterations. The SCAM frequency response of the evolved circuit simulation is shown with the red colour (dash) whereas the equivalent circuit PSpice simulation is shown with the blue colour (dash) in Fig. 16. The desired circuit specifications are achieved regarding design wise and frequency response curve. The MSCAM, original circuit specifications, have the same cut-off frequencies at 8.35 MHz with zero error and a gain of 0.0625.

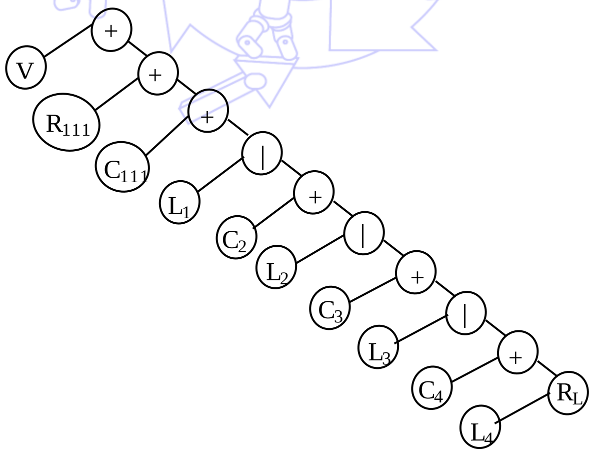

Fig. 14 High-pass evolved circuit tree representations

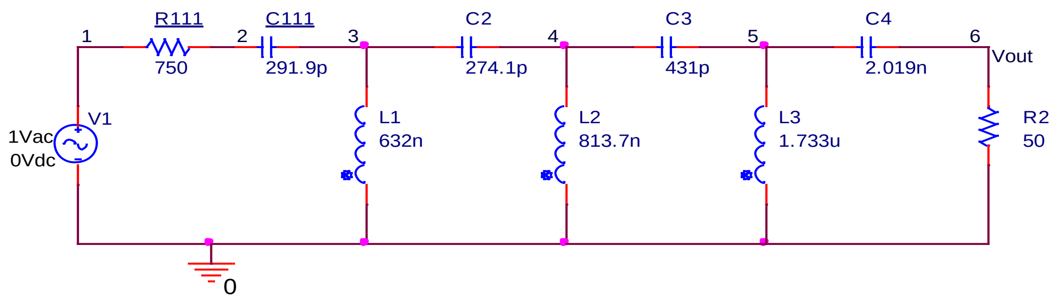

Fig. 15 High-pass evolved circuit

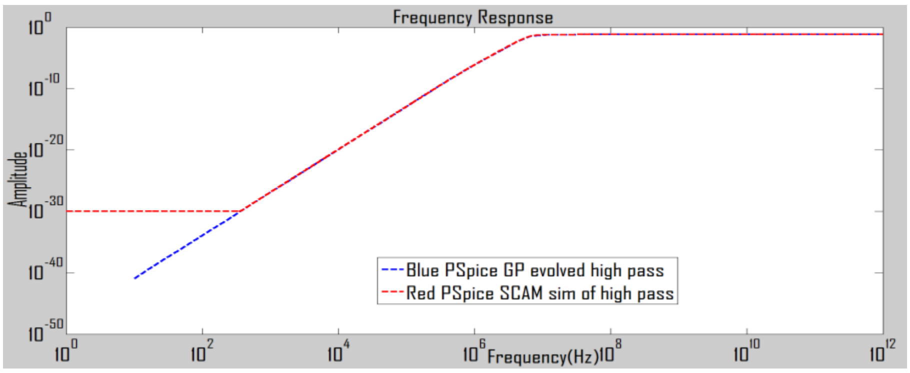

Fig. 16 High-pass frequency response curve for SCAM (blue) and PSpice (Red)

4.5. *Example 4: Band-Pass Passive Filter Circuit*

In example 4, the expected or desired circuit GP representation is shown in Fig. 17 whereas the equivalent circuit representation is in Fig. 18. It takes two hours to evolve the circuit after one hundred and two iterations. The SCAM frequency response of the evolved circuit simulation is shown with the red colour (dash) whereas the equivalent circuit PSpice simulation is shown with the blue colour (dash) in Fig. 19. The same algorithm has been used for all these circuits evolution with little modifications in the cut-off frequency, length of chromosome, bit groups, which are values to play with, and it eliminates mathematical computations involving human methods. The GP evolved MSCAM simulation and original circuit specifications are with the lower and upper cut-off frequencies at 8.242 kHz and 48.59 kHz respectively with zero error and a gain of 0.0625.

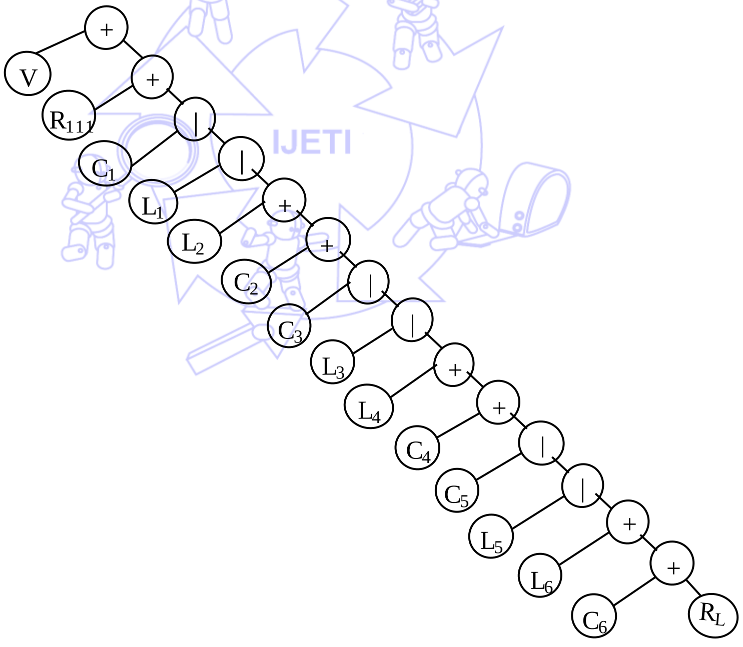

Fig. 17 Band-pass evolved circuit tree representations

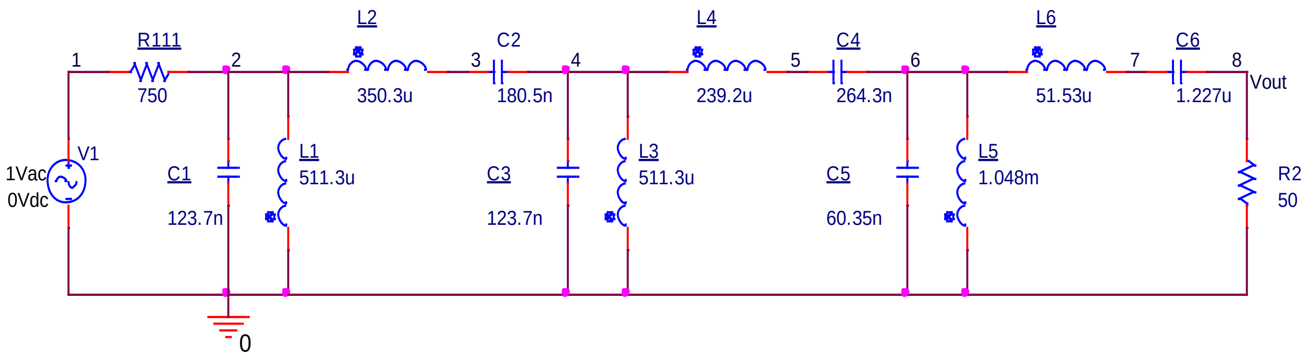

Fig. 18 Band-pass evolved circuit tree representations

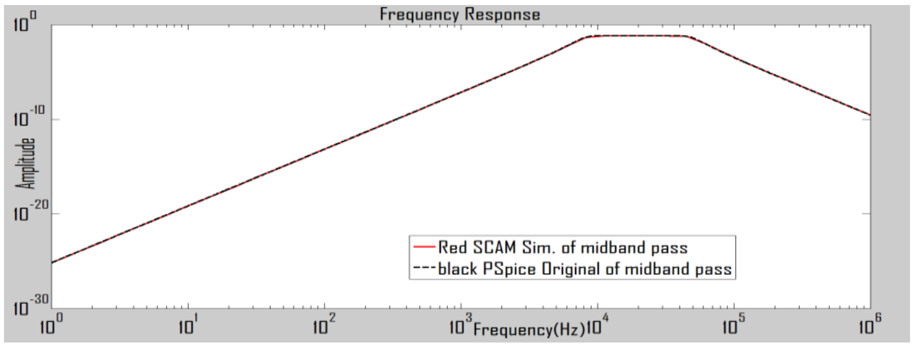

Fig. 19 Band-pass frequency response curve for SCAM (red) and PSpice (black)

5. Conclusions

This paper introduced the use of GP, GF and SCAM for the first time use for the evolution of passive filter circuits. The work has provided an alternative way of using GP in Matlab only for the evolution of passive filter circuits compared the existing GP approach that combine two software packages (Matlab and PSpice). It also reduced the elapse time used while transferring simulation between the two software packages during simulations. Results presented show that the algorithm is efficient in terms of design wise.

References

[1] K. K. Anumandla, R. Peesapati, S. L. Sabat, S. K. Udgata, and A. Abraham, “Field programmable gate arrays-based differential evolution coprocessor: a case study of spectrum allocation in cognitive radio network,” IET Computers & Digital Techniques, vol. 7, no. 5, pp. 221-234, August 2013.

[2] E. A. Coyle, L. P. Maguire, and T. M. McGinnity, “Design philosophy for self-repair of electronic systems using the UML,” IEE Proceedings-Software, vol. 149, no. 6, pp. 179-186, December 2002.

[3] S. Maheshwari, “Analogue signal processing applications using a new circuit topology,” IET Circuits, Devices & Systems, vol. 3, no. 3, pp. 106-115, June 2009.

[4] A. Tyrrell, R. Krohling, and Y. Zhou, “Evolutionary algorithm for the promotion of evolvable hardware,” IEE Proceedings-Computers and Digital Techniques, vol. 151, no. 4, pp. 267-275, July 2004.

[5] S. Vakili, S. M. Fakhraie, and S. Mohammadi, “Evolvable multi-processor: a novel MPSoC architecture with evolvable task decomposition and scheduling,” IET Computers & Digital Techniques, vol. 4, no. 2, pp. 143-156, March 2010.

[6] J. Wang, Q. S. Chen, and C. H. Lee, “Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware,” IET Computers & Digital Techniques, vol. 2, no. 5, pp. 386-400, September 2008.

[7] B. Singh, V. Verma, A. Chandra, and K. Al-Haddad, “Hybrid filters for power quality improvement,” IEE Proceedings-Generation, Transmission and Distribution, vol. 152, no. 3, pp. 365-378, May 2005.

[8] S. Maheshwari and B. Chaturvedi, “High-input low-output impedance all-pass filters using one active element,” IET Circuits, Devices & Systems, vol. 6, no. 2, pp. 103-110, March 2012.

[9] G. Alpaydin, S. Balkir, and G. Dündar, “An evolutionary approach to automatic synthesis of high-performance analog integrated circuits,” IEEE Transactions on Evolutionary Computation, vol. 7, no. 3, pp. 240-252, June 2003.

[10] E. Martens and G. Gielen, “Classification of analog synthesis tools based on their architecture selection mechanisms,” Integration, the VLSI Journal, vol. 41, no. 2, pp. 238-252, February 2008.

[11] O. Mitea, M. Meissner, L. Hedrich, and P. Jores, “Automated constraint-driven topology synthesis for analog circuits,” Design, Automation & Test in Europe Conference & Exhibition, IEEE press, May 2011, pp. 1-4.

[12] M. Harman, W. B. Langdon, Y. Jia, D. R. White, A. Arcuri, and J. A. Clark, “The GISMOE challenge: Constructing the pareto program surface using genetic programming to find better programs (keynote paper),” Proc. IEEE/ACM International Conf. Automated Software Engineering, IEEE press, April 2012, pp. 1-14.

[13] S. Forrest, T. Nguyen, W. Weimer, and C. Le Goues, “A genetic programming approach to automated software repair,” Proc. Annual Conf. Genetic and Evolutionary Computation, ACM press, July 2009, pp. 947-954.

[14] W. Weimer, S. Forrest, C. Le Goues, and T. Nguyen, “Automatic program repair with evolutionary computation,” Communications of the ACM, vol. 53, no. 5, pp. 109-116, May 2010.

[15] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest, "Automatically finding patches using genetic programming," Proc. International Conf. Software Engineering, ACM press, May 2009, pp. 364-374.

[16] J. D. Lohn and S. P. Colombano, "A circuit representation technique for automated circuit design," IEEE Transactions on Evolutionary Computation, vol. 3, no. 3, pp. 205-219, September 1999.

[17] J. R. Koza, "Human-competitive results produced by genetic programming," Genetic Programming and Evolvable Machines, vol. 11, no. 3-4, pp. 251-284, May 2010.

[18] J. R. Koza, S. H. Al-Sakran, and L. W. Jones, "Cross-domain features of runs of genetic programming used to evolve designs for analog circuits, optical lens systems, controllers, antennas, mechanical systems, and quantum computing circuits," Proc. 2005 NASA/DoD Conf. Evolvable Hardware, IEEE press, September 2005, pp. 205-212.

[19] J. R. Koza, F. H. Bennett III, D. Andre, M. A. Keane, and F. Dunlap, "Automated synthesis of analog electrical circuits by means of genetic programming," IEEE Transactions on Evolutionary Computation, vol. 1, no. 2, pp. 109-128, July 1997.

[20] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," Statistics and Computing, vol. 4, no. 2, pp. 87-112, June 1994.

[21] M. Walker, "Introduction to genetic programming," Tech.Np: University of Montana, 2001.

[22] T. Sripramong and C. Toumazou, "The invention of CMOS amplifiers using genetic programming and current-flow analysis," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 11, pp. 1237-1252, November 2002.

[23] S. Silva and J. Almeida, "GPLAB-a genetic programming toolbox for MATLAB," Proc. Nordic MATLAB Conference, 2005, pp. 273-278.

[24] K. Rodríguez and R. Mendoza, "A Matlab genetic programming approach to topographic mesh surface generation," Engineering Education and Research Using MATLAB, pp.427-442, 2011.

[25] H. Hou, S. Chang, and Y. Su, "Economical passive filter synthesis using genetic programming based on tree representation," IEEE International Symp. Circuits and Systems, IEEE press, July 2005, p. 3003.

[26] S. Chang and Y. Su, "Automated passive filter synthesis using a novel tree representation and genetic programming," IEEE Transactions on Evolutionary Computation, vol. 10, no. 1, pp. 93-100, February 2006.

[27] A. Senn, A. Peter, and J. G. Korvink, "Analog circuit synthesis using two-port theory and genetic programming," AFRICON, IEEE press, November 2011, pp. 1-8.

[28] M. Mezher and M. F. Abbod, "A new genetic folding algorithm for regression problems," UKSim 14th International Conf. Computer Modelling and Simulation (UKSim), IEEE press, May 2012, pp. 46-51.

[29] S. Boyd, S. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," Optimization and Engineering, vol. 8, no. 1, pp. 67-127, March 2007.

[30] G. Gielen and W. Sansen, Symbolic analysis for automated design of analog integrated circuits, Springer Science & Business Media, New York, 2012.

[31] O. J. Ushie, M. Abbod, and E. Ashigwuike, "Matlab symbolic circuit analysis and simulation tool using PSpice netlist for circuits optimization," International Journal of Engineering and Technology Innovation, vol. 5, no. 2, pp. 75-86, April 2015.

[32] E. Cheever, "Symbolic Circuit Analysis in MATLAB (SCAM)," Swarthmore College, http://www.swarthmore.edu/, November, 2005.

[33] M. Jamil and X. Yang, "A literature survey of benchmark functions for global optimisation problems," International Journal of Mathematical Modelling and Numerical Optimisation, vol. 4, no. 2, pp. 150-194, 2013.

[34] R. Feldt, M. O'Neill, C. Rayn, P. Nordin, and W. B. Langdon, "GP-beagle: A benchmarking problem repository for the genetic programming community," Late Breaking Papers at GECCO, UCL Discovery press, June 2000, pp. 1-8.