Automatic Programming Methodologies for Electronic Hardware Fault Monitoring

Ajith Abraham
(IITA Professorship Program, School of Computer Science and Engineering,
Chung-Ang University, Seoul, South Korea
ajith.abraham@ieee.org)

Crina Grosan
(Department of Computer Science
Babes-Bolyai University, Cluj-Napoca, 3400, Romania
cgrosan@cs.ubbcluj.ro)

Abstract: This paper presents three variants of Genetic Programming (GP) approaches for intelligent online performance monitoring of electronic circuits and systems. Reliability modeling of electronic circuits can be best performed by the stressor – susceptibility interaction model. A circuit or a system is considered to be failed once the stressor has exceeded the susceptibility limits. For on-line prediction, validated stressor vectors may be obtained by direct measurements or sensors, which after pre-processing and standardization are fed into the GP models. Empirical results are compared with artificial neural networks trained using backpropagation algorithm and classification and regression trees. The performance of the proposed method is evaluated by comparing the experiment results with the actual failure model values. The developed model reveals that GP could play an important role for future fault monitoring systems.

Keywords: genetic programming, neural networks, decision trees, fault monitoring, computational intelligence, electronic hardware
Categories: I.2.2, I.2.6, B.8.1, B.8.2

1 Introduction

Real time monitoring of the healthiness of complex electronic systems/circuits/hardware is a difficult challenge to both human operators and expert systems. When the electronic circuit or system is controlling a critical task fault prediction will be very important. This paper proposes a stressor-susceptibility interaction model for analyzing the hardware and three variants of genetic programming methods for approximating the various complex functions to monitor the performance of the system.

In the literature several fault monitoring/analysis methods have been proposed [Abraham and Grosan, 2005]. Advances in integrated circuit technology have made failure site localization extremely challenging. Charge-induced voltage alteration (CIVA), low energy CIVA (LECIVA), light-induced voltage alteration (LIVA), Seebeck effect imaging (SEI) and thermally-induced voltage alteration (TIVA) are five recently developed failure analysis techniques which meet the challenge by rapidly and non-destructively localizing interconnection defects on ICs. Yamada and

Komoda proposed a failure analysis on a 0.18 μm CMOS device by combining several fault localization techniques [Yamada and Komoda, 2004]. Mohsena and El-Yazeed addresses the problem of fault diagnosis of analog circuits based on dictionary approach [Mohsena and El-Yazeed, 2004]. The proposed approach first identifies an adequate set of test frequencies to optimize the process of detection and isolation of simulated fault scenarios. The circuit under test is then excited by an input stimulus composed of a set of sinusoidal waveforms with the selected test frequencies. The circuit response, at different fault scenarios, is preprocessed by an autoregressive moving average model to yield a set of features formulating the fault dictionary. Collected features are utilized to train and test a neural network based classifier. Demonstrative results from soft fault simulation of two active circuit examples prove the excellent effectiveness of the proposed algorithm.

El-Gamal and Abdulghafour proposed a fuzzy inference system for single analog fault diagnosis [El-Gamal and Abdulghafour, 2004]. The ability of fuzzy logic to encode structured knowledge in a numerical framework is exploited in isolating faults in analog circuits. A training set that simulates the behaviour of the circuit due to a set of anticipated single faults as well as the fault-free situation is first constructed. For each anticipated fault, this set relates the circuit measurements to the corresponding deviation in the faulty circuit element from its nominal. These measurements and the deviations in circuit elements are both fuzzified into appropriate linguistic fuzzy values. A fuzzy rule base for each fault that characterizes the circuit response by linking symptoms to causes is built. The outputs of the fuzzy rule bases are then defuzzified to recover crisp values for the deviations in circuit elements.

Blyzniuk et al. proposed a new methodology of probabilistic analysis of CMOS physical defects in complex gates [Blyzniuk et al., 2001]. It is based on the developed approach for the identification and estimation of the probability of actual faulty functions resulting from shorts caused by spot defects in conductive layers of IC layout. The aim of this methodology is realistic representation of physical defects in fault models. The list of defects, identified faulty functions, defect coverage table, conditional defect probabilities, and effectiveness and optimal sequence of test patterns are the main output data of probabilistic-based faults characterization. The experimental data obtained during complex gates characterization are used for the estimation of the physical defects coverage by hierarchical defect simulation.

Dai and Xu proposed an analog circuit fault diagnosis method using a noise measurement and analysis approach [Dai and Xu, 1999]. Compared to the conventional circuit fault diagnosis methods, this method can discover hidden and early circuit fault caused by the device defects.

SRAM's are frequently used as monitor circuits for defect related yield, due to the ease of testing and the good correlation to the yield characteristics of logic circuitry. For the identification of the failure/fault type and the nature of the defect causing the failure, measured failbitmaps are mapped onto a failbitmap catalog obtained from defect-fault simulation. Often this mapping is not unique. A given failbitmap can be caused by several faults or defects. Schienle et al. demonstrated the application of current signature analysis for a stand-alone 16kx1 SRAM monitor circuit [Schienle et al., 1999]. It is found that the resolution of the failbitmap-fault-defect catalog can be improved considerably by additional current signature measurements.

Catelani and Giraldi proposed a method for fault detection and fault isolation of analog circuits by considering the response of the circuit under test which is obtained by measuring the output voltage at an accessible node when a stimulus constituted by a signal with a particular test frequency is applied at its input [Catelani and Giraldi, 1999]. In a fault condition, such a response represents the fault diagnosis equation for the analog circuit. The theoretical formulation is confirmed by the results achieved for an active low-pass filter.

Toczek et al. proposed a component fault isolation procedure for the robust fault diagnosis of analog circuits [Toczek et al., 1998]. The procedure is divided into two stages. The first stage is based on nonlinear analysis of circuit under test and verification is performed with circuit model linearized in the neighbourhood of the operating point, and the second stage is based on nonlinear analysis of circuit with only some nonlinear devices modeled by piecewise-linear function. This economical approach keeps the computation time within the acceptable limits in comparison with entire PWL model approach and diagnosis is accomplished at low measurement cost.

Stressor is a physical entity influencing the lifetime of a component or circuit. A stressor, indicating a physical entity x will be denoted as ψ_x. Stressors can be broadly classified into three main groups [Brombacher, 1995]. First group contains the electrical stressors, parameters related to the electrical behavior of the circuit. Second group of stressors is the mechanical stressors, which are related to the mechanical environment of the component. Third group of parameters influencing the lifetime of components is related to the thermal environment of the component. Susceptibility of a component to a certain failure mechanism is defined as the probability function indicating the probability that a component will not remain operational for a certain time under a given combination of stressors. The susceptibility related to the failure mechanism y is usually defined as $S_y\ (t,\ \psi_p,\ \psi_q, \psi_r)$.

The new technique of electronic system failure prediction using stressor-susceptibility interaction [Abraham, 2000],[Abraham and Nath, 1999] is briefly discussed in Section 2. This technique can be extended to simple electronic components and for complicated electronic circuits and equipment. Section 3 presents some of the common failure mechanisms in practical situations. The derivation of stressor sets using Monte Carlo Analysis is given in Section 4 followed by Section 5 where we had derived a stressor-susceptibility model for a circuit. Section 6 gives some theoretical background about the variants of genetic programming models used, artificial neural networks and decision trees. In Section 7 we have reported the experiment results and finally conclusions are provided in Section 8.

2 Stressor-Susceptibility Interaction

Failure probabilities require detailed analysis of both stressors and susceptibility. Most components tend to have more than one failure mechanism, resulting in more than one "failure probability". It can be shown that there is a strong correlation between the various failure mechanisms existing within a component.

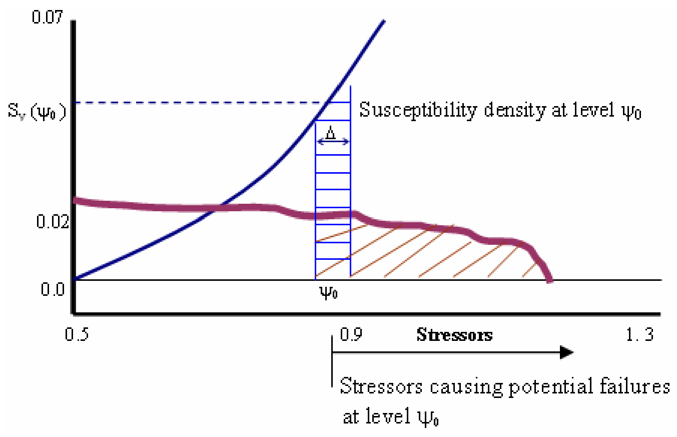
0.07
$S_y(\psi_0)$
0.02
0.0
Susceptibility density at level ψ_0
Δ
ψ_0
0.5
0.9
Stressors
1.3
Stressors causing potential failures at level ψ_0

Figure 1. Stressor-Susceptibility interaction for single failure mechanism.

Figure 1 illustrates the stressor - susceptibility interaction for a single failure mechanism. The main source of problem is the overlap between stressor and susceptibility density. The first step is to calculate the failure probability for this stressor distribution on a failure mechanism with a single, one variable, time independent catastrophic susceptibility model. This results in the probability function as given below:

$$f_{fail,y,\psi}(\psi_0) = \int_{\psi_0}^{\infty} f_y(\psi)d\psi \qquad (1)$$

To calculate the failure probability as a function of more complex susceptibility model, it will be necessary to calculate the failure probability of a part of the susceptibility model, for a certain stressor interval *Δ*, characterized by its mean value ψ_o and the corresponding susceptibility density function at that point $S_y(\psi_o)$. Considering the probability that a part has failed at a lower susceptibility level, results in the possibility to predict the failure probability per time interval of a certain failure at stressor level ψ_0 using (2).

$$f_{fail,y,\psi}(\psi_0) = \Delta(S_Y(\psi_0)\int_{\psi_0}^{\infty} f_y(\beta\psi)d\psi)\left(1 - \int_{0}^{\psi_0-\Delta} f_{fail,y}(\psi)d\psi\right) \qquad (2)$$

The last term is introduced to subtract failures caused by stressors at a lower susceptibility level. As, most often, failure probabilities are very small, in many cases the previous expression will simplify to (3).

$$f_{fail,y,\psi}(\psi_0) = \Delta(S_y(\psi_0)\int_{\psi_0}^{\infty} f_y(\beta\psi)d\psi) \qquad (3)$$

when $$(1 - \int_{0}^{\psi_0 - \Delta} f_{fail,y}(\psi) d\psi) = 1 \qquad (4)$$

Since the susceptibility is defined as the probability that a component will not remain operational during a certain time, it is therefore possible to calculate the failure probability during a certain observation time t_{obs}.

$$f_{fail,y,\psi}(\psi_0) = t_{obs} * \Delta * (S_y(\psi_0) \int_{\Psi_0}^{\infty} f_y(\psi) d\psi) \qquad (5)$$

The important requirement for using (5) is that the observation time t_{obs} must be larger than the total elapsed sampling time to obtain an ergodic description of the associated stressors $t_{total\ sample}$ ($t_{obs} > t_{total\ sample}$); $f_{fail,\ y,\psi,}(t,\ \psi)$ is assumed to be constant during the time interval t_{obs}. From (5) it is possible to calculate the failure probability of a part per fail mechanism per time interval using (6).

$$f_{fail,y} = \int_{0}^{\infty} f_{fail,y,\psi}(t, \psi) d\psi \qquad (6)$$

Equation (6) can now be used to calculate the part failure probability per time interval

$$f_{fail} = \sum_{i=1}^{n} f_{fail,i} \qquad (7)$$

Using the previous assumptions it is also possible to calculate the probability that a component survives from time t to $t+dt$. Equation (8) can be used to calculate the failure probability for one single failure mechanism within one single device.

$$R(t \ .... \ t+\Delta t) = \frac{\sum_{i=1}^{k} Devices\ \ operational\ \ at\ \ time\ \ (t+\Delta t)}{\sum_{i=1}^{n} Devices\ \ operational\ \ at\ \ (t)} = R(t)F(\Delta t) = R(t)\Delta t f(t) \qquad (8)$$

As for large series of components, the physical structures of the individual components will be different for every component; the survival probability of such a series of components will also show individual differences. The stress on a component may vary with time due to circuit behavior and circuit use. The circuit behavior will differ amongst a series of circuits due to physical differences in the individual circuit components, the physical structure of a circuit, the use of a circuit and the environment (electrical, thermal, etc.) of the circuit. To summarize the variety of effects it is useful to describe stressors as stochastic signals with properties depending on the influencing factors mentioned above. These assumptions make it possible to derive the failure probability and reliability of a component using a Markov approach. For Markov approach the following requirements should be fulfilled:

- Susceptibility of all failure mechanisms in a component is known and is constant in the time interval (t, $t+\Delta t$).
- All stressors ψ_a (t), ψ_b (t), … are known as stochastic signals for the time interval (t, $t+\Delta t$).
- The failure probability (or reliability) is known at a certain (initial) time t.

Using these properties it is possible to calculate the reliability and failure probability for components, derived from internal failure mechanisms for time $t+\Delta t$. For this purpose the following relationships are used

$$P(t+\Delta t) = P(t) ✪ (\Delta t)$$

$$= \vec{P}(t)\begin{bmatrix} P_{x\to x} & \cdots & \cdots & P_{x\to y} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ P_{y\to x} & \cdots & \cdots & P_{y\to y} \end{bmatrix} \qquad (9)$$

where $P(t)$ is the state probability vector of a component. This state probability vector is defined as

$\vec{P}(t) =$

$P_{operational}\,(t)$: probability that part is operational at time t
$P_a\,(t)$: probability that part fails due to failure mechanism a at time t
$P_b\,(t)$: probability that part fails due to failure mechanism b at time t
$P_n\,(t)$: probability that part fails due to failure mechanism n at time t

$$\sum_{j=1}^{n} P_j\,(y) = 1$$

$$\begin{aligned} P_1\,(t) &= P_{operational}\,(t) &&= R(t) \\ P_{2\ldots.n}\,(t) &= P_{fail,\,2\ldots\,n}\,(t) &&= F_{fail,\,2\ldots.\,n}\,(t) \\ P_{x\prod y} &= P_{(y(t+\Delta t)\,|\,x(t)\,)} &&= f_y(t)\,\Delta t\;P_x\,(t) \end{aligned}$$

It is possible to replicate this calculation process for a whole batch of circuits. In this case, for every circuit the individual stressor/ susceptibility interaction is calculated thus simulating batch behavior. Using this method, it is possible to derive the failure probability for many parts in many practical situations, also in cases where considerable differences (in stressors and susceptibility) exist within a batch.

3 From Failure Mechanisms to Stressor Sets and Susceptibility Models

There are two different categories of failure mechanisms applicable to electronic components [Chan, 1994], [Jenson, 1995]. First, the failure mechanisms that are related to the electrical stress in a circuit [Klion, 1992].

No.	Failure Mechanism	Influencing aspect or associated stressors
1	Thermal failure (general)	• Dissipated power • Environmental temperature • Thermal resistance • Thermal capacitance
2	Current breakdown (hot spot melting)	• Resistivity of the material • Impurities/ mechanical distortions in the material causing increase in current density. • Thermal resistivity coefficient.
3	Power breakdown (thermal cracks)	• Thermal expansion coefficient of the materials. • Thermal resistivity coefficients of the materials.
4	Impact ionization	• Electric field
5	Avalanche breakdown	• Electric field (positive temperature coefficient)
6	Zener breakdown	• Electric field (negative temperature coefficient)
7	Corrosion	• Environmental temperature (negative influence on susceptibility) • Dissipated power • D C Voltage
8	Electro-migration	• Current density • Environmental temperature
9	Secondary diffusion	• Temperature
10	Switch on pulse power dissipation (for bipolar junctions)	• Voltage slope *dV/dt* • Current slope *dI/dt*
11	Switch off pulse power dissipation (for bipolar junctions)	• Voltage slope *dV/dt* • Maximum reverse junction current • Applied reverse voltage • Storage charge *Q*'s in the diode at the moment of polarity reversal.
12	Forward bias second breakdown (for power transistors)	• Collector emitter voltage • Slope of the base current during switching on dI_b/dt • Slope of the collector current during switching on dI_c/dt • Environmental temperature
13	Reverse bias second breakdown (for power transistors)	• Collector emitter voltage • Discharge speed dI_b/dt (optimum value) • Stored charge at the moment of transistor switch off (closely related to collector current at the moment of switch off). • Environmental temperature

Table 1. Some common failure mechanisms with associated causes and stressors

Second there are failure mechanisms related to the intrinsic aspects of a component [Fuqua, 1987], [Arsenault and Roberts, 1980]. Table 1 shows some of the typical failure mechanisms and their causes with associated stressors. There are two possible ways to obtain stressor sets for practical circuits [Brombacher, 1995]. The first possibility involves usage of computer simulation models to derive all circuit signals using one single simulation. Second possibility is to derive stressor sets from practical measurements. In those cases where sufficient systems are available it is possible to do a statistical evaluation of the individual stressor functions existing in individual systems. As the stressor sets are dependent on the conditions of use and the operation modes of a system it is important that the measured stressor is based on all the possible operation modes of a circuit and all the possible transitions between the various operation modes. This can become a quite tedious job as the entire operation is to be repeated for a number of systems to obtain an accurate statistical mean stressor model. Accurate description of a stressor set needs a sampling frequency of at least twice the highest frequency in the stressor frequency spectrum. Accurate description of a stressor set will require a number of samples sufficient to cover all the different states of the system. As a signal has often more than one quasi-stationary states, each characterized by their stressor set, it is possible to derive the overall stressor set function from the individual state stressor sets using (10)

$$f_{str,y(x)} = \sum_{i=1}^{n} \frac{T_i}{T_{total}} f_{str,y,i(x)} \qquad (10)$$

$f_{str,y\,(x)}$ is the stressor probability density function of quasi-stationary state i. T_i / T_{total} is the fraction of time that the stressor is in quasi-stationary state i.

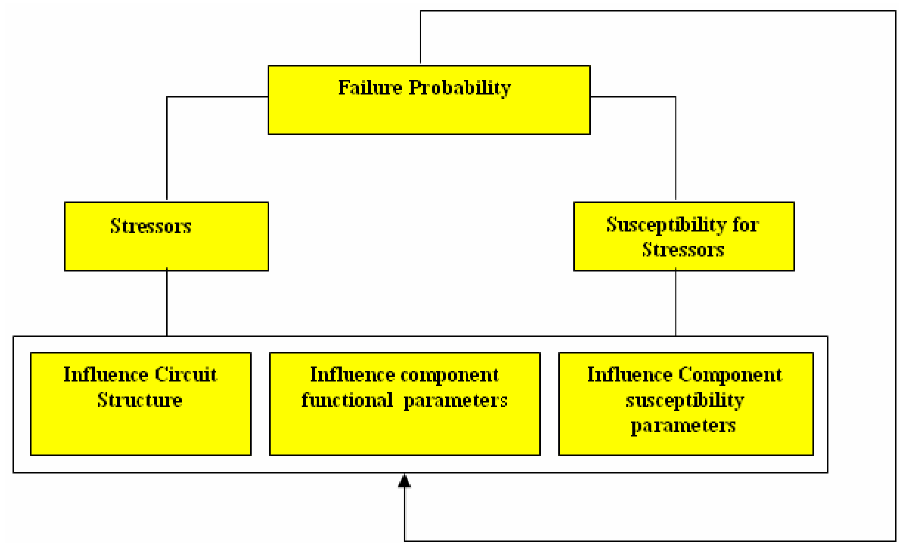
Failure Probability
Stressors
Susceptibility for Stressors
Influence Circuit Structure
Influence component functional parameters
Influence Component susceptibility parameters

Figure 2. Monte Carlo analysis

4 Monte Carlo Analysis for Stressor Sets

In a Monte Carlo Analysis (MCA), a logical model of the system being analyzed is repeatedly evaluated, each run using different values of the distributed parameters. The selection of parameter values is made randomly, but with probabilities governed by the relevant distribution functions. Statistical exploration covers the tolerance space by means of the generation of sets of random parameters within this tolerance space. Each set of random parameters represents one circuit. Multiple circuit simulations, each with a new set of random parameters, explore the tolerance space. Statistically the distribution of all random selections of one parameter represents the parameter distribution. Although the number of simulations required for MCA is quite large, this analysis method is useful, especially because the number of parameters in the failure prediction of circuits is often too large to allow the use of other techniques. Figure 2 illustrates the MCA. With MCA it is possible to simulate the behavior of a large batch of circuits and derive stressor sets. The next phase will be the combination of the derived stressor sets with the component susceptibilities in order to decide whether a component will fail or not. As for the failure prediction, the most important aspect is to prevent failures; susceptibility will be expressed using the susceptibility limit. To distinguish circuits where failures are possible any circuit in the MCA causing to exceed a susceptibility limit are marked as fail. Circuits where no stressors exceed susceptibility limit are marked as pass.

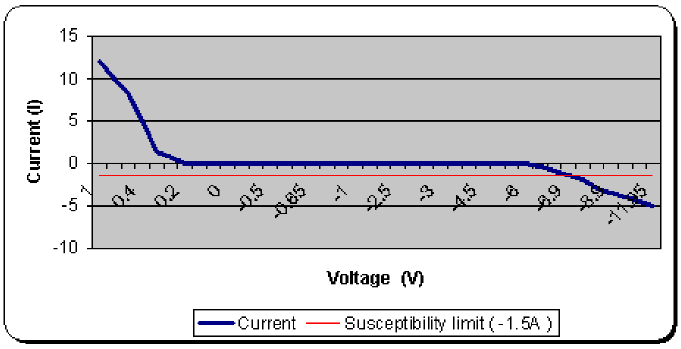

Figure 3. Stressor - susceptibility interaction model

5 Modelling Stressor Sets and Susceptibility

The analysis was carried out on a power circuit and the main cause of the failure of the circuit was a Schottky diode. The main failure mechanisms are leakage current and excess crystal temperature. Using the procedure described earlier, it was possible to derive a complete individual stressor set for the failure mechanism of this diode as follows.

Parameter	**Susceptibility limit**
T crystal	125o Celsius
dV/dt	109 V/s
dI/dt	0.51 x 109 A/s
I (reverse)	-1.5 A

Figure 3 illustrates the joint stressor – susceptibility interaction model in terms of voltage and current. The susceptibility limit for leakage current is set at -1.5A.

6 Computational Intelligence Paradigms Used

Following intelligent techniques are used in the experiments: Linear Genetic programming (LGP), Multi Expression Programming (MEP), Gene Expression Programming (GEP), Artificial Neural Networks (ANN) and Classification and Regression Trees (CART). Each of them is described in the following sub-sections.

6.1 Linear Genetic Programming (LGP)

Linear genetic programming is a variant of the GP technique that acts on linear genomes [Banzhaf et al., 1998]. Its main characteristics in comparison to tree-based GP lies in that the evolvable units are not the expressions of a functional programming language (like LISP), but the programs of an imperative language (like C/C ++). The basic unit of evolution here is a native machine code instruction that runs on the floating-point processor unit (FPU). Since different instructions may have different sizes, here instructions are clubbed up together to form instruction blocks of 32 bits each. The instruction blocks hold one or more native machine code instructions, depending on the sizes of the instructions. A crossover point can occur only between instructions and is prohibited from occurring within an instruction. However the mutation operation does not have any such restriction. LGP uses a specific linear representation of computer programs. Instead of the tree-based GP expressions of a functional programming language (like *LISP*) programs of an imperative language (like *C*) are evolved. A LGP individual is represented by a variable-length sequence of simple *C* language instructions. Instructions operate on one or two indexed variables (registers) *r*, or on constants *c* from predefined sets. The result is assigned to a destination register, for example, *ri* = *rj** *c*. A sample LGP program is given below:

```
void LGP(double v[8]) {
[0] = v[5] + 73;
v[7] = v[3] – 59;
if (v[1] > 0)
if (v[5] > 21)
v[4] = v[2] . v[1];
v[2] = v[5] + v[4];
v[6] = v[7] . 25;
v[6] = v[4] – 4;
v[1] = sin(v[6]);
if (v[0] > v[1])
v[3] = v[5] . v[5];
v[7] = v[6] . 2;
v[5] = v[7] + 115;
if (v[1] <= v[6])
v[1] = sin(v[7]);
}
```

A LGP can be turned into a functional representation by successive replacements of variables starting with the last effective instruction. The maximum number of symbols in a LGP chromosome is 4 * Number of instructions.

Evolving programs in a low-level language allows us to run those programs directly on the computer processor, thus avoiding the need of an interpreter. In this way the computer program can be evolved very quickly. An important LGP parameter is the number of registers used by a chromosome. The number of registers is usually equal to the number of attributes of the problem. If the problem has only one attribute, it is impossible to obtain a complex expression such as the quartic polynomial. In that case, we have to use several supplementary registers. The number of supplementary registers depends on the complexity of the expression being discovered. An inappropriate choice can have disastrous effects on the program being evolved. LGP uses a modified steady-state algorithm. The initial population is randomly generated. The following steps are repeated until a termination criterion is reached: Four individuals are randomly selected from the current population. The best two of them are considered the winners of the tournament and will act as parents. The parents are recombined and the offspring are mutated and then replace the losers of the tournament.

We used a LGP technique that manipulates and evolves a program at the machine code level. The settings of various linear genetic programming system parameters are of utmost importance for successful performance of the system. The population space has been subdivided into multiple subpopulation or demes. Migration of individuals among the subpopulations causes evolution of the entire population. It helps to maintain diversity in the population, as migration is restricted among the demes. Moreover, the tendency towards a bad local minimum in one deme can be countered by other demes with better search directions. The various LGP search parameters are the mutation frequency, crossover frequency and the reproduction frequency. The crossover operator acts by exchanging sequences of instructions between two tournament winners. Steady state genetic programming approach was used to manage the memory more effectively.

6.2 Multi Expression Programming (MEP)

MEP genes are (represented by) substrings of a variable length [Oltean and Grosan, 2003]. The number of genes per chromosome is constant. This number defines the length of the chromosome. Each gene encodes a terminal or a function symbol. A gene that encodes a function includes pointers towards the function arguments.

Function arguments always have indices of lower values than the position of the function itself in the chromosome. The proposed representation ensures that no cycle arises while the chromosome is decoded (phenotypically transcripted). According to the proposed representation scheme, the first symbol of the chromosome must be a terminal symbol. In this way, only syntactically correct programs (MEP individuals) are obtained. An example of chromosome using the sets $F = \{+, *\}$ and $T = \{a, b, c, d\}$ is given below:

1: a
2: b
3: + 1, 2
4: c
5: d
6: + 4, 5
7: * 3, 6

The maximum number of symbols in MEP chromosome is given by

$$\textit{Number of symbols} = (n + 1) * (\textit{number of genes} - 1) + 1$$

where n is the number of arguments of the function with the greatest number of arguments. The maximum number of effective symbols is achieved when each gene (excepting the first one) encodes a function symbol with the highest number of arguments. The minimum number of effective symbols is equal to the number of genes and it is achieved when all genes encode terminal symbols only.

The translation of a MEP chromosome into a computer program represents the phenotypic transcription of the MEP chromosomes. Phenotypic translation is obtained by parsing the chromosome top-down. A terminal symbol specifies a simple expression. A function symbol specifies a complex expression obtained by connecting the operands specified by the argument positions with the current function symbol. For instance, genes 1, 2, 4 and 5 in the previous example encode simple expressions formed by a single terminal symbol. These expressions are:

$E_1 = a$,
$E_2 = b$,
$E_4 = c$,
$E_5 = d$,

Gene 3 indicates the operation + on the operands located at positions 1 and 2 of the chromosome. Therefore gene 3 encodes the expression: $E_3 = a + b$. Gene 6 indicates the operation + on the operands located at positions 4 and 5. Therefore gene 6 encodes the expression: $E_6 = c + d$. Gene 7 indicates the operation * on the operands located at position 3 and 6. Therefore gene 7 encodes the expression: $E_7 = (a + b) * (c + d)$. E_7 is the expression encoded by the whole chromosome.

There is neither practical nor theoretical evidence that one of these expressions is better than the others. This is why each MEP chromosome is allowed to encode a number of expressions equal to the chromosome length (number of genes). The chromosome described above encodes the following expressions:

$E_1 = a$,
$E_2 = b$,
$E_3 = a + b$,
$E_4 = c$

$E_5 = d$,
$E_6 = c + d$,
$E_7 = (a + b) * (c + d)$.

The value of these expressions may be computed by reading the chromosome top down. Partial results are computed by dynamic programming and are stored in a conventional manner. Due to its multi expression representation, each MEP chromosome may be viewed as a forest of trees rather than as a single tree, which is the case of GP.

6.3 Gene Expression Programming (GEP)

The individuals of gene expression programming [Ferreira, 2001] are encoded in linear chromosomes which are expressed or translated into expression trees (branched entities). Thus, in GEP, the genotype (the linear chromosomes) and the phenotype (the expression trees) are different entities (both structurally and functionally) that, nevertheless, work together forming an indivisible whole. In contrast to its analogous cellular gene expression, GEP is rather simple. The main players in GEP are only two: the chromosomes and the Expression Trees (ETs), being the latter the expression of the genetic information encoded in the chromosomes. As in nature, the process of information decoding is called translation. And this translation implies obviously a kind of code and a set of rules. The genetic code is very simple: a one-to-one relationship between the symbols of the chromosome and the functions or terminals they represent. The rules are also very simple: they determine the spatial organization of the functions and terminals in the ETs and the type of interaction between sub-ETs. GEP uses linear chromosomes that store expressions in breadth-first form. A GEP gene is a string of terminal and function symbols. GEP genes are composed of a *head* and a *tail*. The head contains both function and terminal symbols. The tail may contain terminal symbols only. For each problem the head length (denoted h) is chosen by the user. The tail length (denoted by t) is evaluated by: $t = (n - 1)h + 1$, where n is the number of arguments of the function with more arguments. Let us consider a gene made up of symbols in the set S:

$S = \{\times, /, +, -, a, b\}$

In this case $n = 2$. If we choose $h = 10$, then we get $t = 11$, and the length of the gene is 10 + 11 = 21. Such a gene is given below:

$C_{GEP} = + \times$ ab - +aab + ababbbababb

The *expression* encoded by the gene C_{GEP} is:

E = a + b × ((a + b) - a)

GEP genes may be linked by a function symbol in order to obtain a fully functional chromosome. In the current version of GEP the linking functions for algebraic expressions are addition and multiplication. A single type of function is used for linking multiple genes. GEP uses mutation, recombination and transposition. GEP uses a generational algorithm. The initial population is randomly generated. The following steps are repeated until a termination criterion is reached: A fixed number of the best individuals enter the next generation (elitism). The mating pool is filled by using binary tournament selection. The individuals from the mating pool are randomly paired and recombined. Two offsprings are obtained by recombining two parents. The offspring are mutated and they enter the next generation. There are some problems regarding multigenic chromosomes. Generally, it is not a good idea to assume that the genes may be linked either by addition or by multiplication. Providing a particular linking operator means providing partial information to the expression which is discovered. But, if all the operators {+, -, ×, /} are used as linking operators, then the complexity of the problem substantially grows (since the problem of determining how to mix these operators with the genes is as difficult as the initial problem). Furthermore, the number of genes in the GEP multigenic chromosome raises a problem. As can be seen in [Ferreira, 2001], the success rate of GEP increases with the number of genes in the chromosome. But, after a certain value, the success rate decreases if the number of genes in the chromosome is increased. This happens because we cannot force a complex chromosome to encode a less complex expression. A large part of the chromosome is unused if the target expression is short and the head length is large. Note that this problem arises usually in systems that employ chromosomes with a fixed length.

6.4 Artificial Neural Networks (ANN)

Artificial Neural Networks have been developed as generalizations of mathematical models of biological nervous systems. A neural network is characterised by the network architecture, the connection strength between pairs of neurons (weights), node properties, and updating rules. The updating or learning rules control weights and/or states of the processing elements (neurons). Normally, an objective function is defined that represents the complete status of the network, and its set of minima corresponds to different stable states of the network. It can learn by adapting its weights to changes in the surrounding environment, can handle imprecise information, and generalise from known tasks to unknown ones. Each neuron is an elementary processor with primitive operations, like summing the weighted inputs coming to it and then amplifying or thresholding the sum. Learning typically occurs by example through training, where the training algorithm iteratively adjusts the connection weights (synapses). Backpropagation (BP) is one of the most famous training algorithms for multilayer perceptrons. BP is a gradient descent technique to minimize the error E for a particular training pattern. For adjusting the weight (w_{ij}) from the i^{th} input unit to the j^{th} output, in the batched mode variant the descent is based on the gradient ∇E ($\frac{\delta E}{\delta w_{ij}}$) for the total training set:

$$\Delta w_{ij}(n) = -\varepsilon * \frac{\delta E}{\delta w_{ij}} + \alpha * \Delta w_{ij}(n-1) \qquad (11)$$

The gradient gives the direction of error E. The parameters ε and α are the learning rate and momentum respectively [Abraham, 2005].

6.5. Classification and Regression Trees (CART)

Tree-based models are useful for both classification and regression problems [Breiman et al., 1984]. In these problems, there is a set of classification or predictor variables (X_i) and a dependent variable (Y). The X_i variables may be a mixture of nominal and / or ordinal scales (or code intervals of equal-interval scale) and Y a quantitative or a qualitative (i.e., nominal or categorical) variable.

The CART methodology is technically known as binary recursive partitioning. The process is binary because parent nodes are always split into exactly two child nodes and recursive because the process can be repeated by treating each child node as a parent. The key elements of a CART analysis are a set of rules for:

- splitting each node in a tree;
- deciding when a tree is complete; and
- assigning each terminal node to a class outcome (or predicted value for regression)

CART's binary decision trees are more sparing with data and detect more structure before further splitting is impossible or stopped. Splitting is impossible if only one case remains in a particular node or if all the cases in that node are exact copies of each other (on predictor variables). CART also allows splitting to be stopped for several other reasons, including that a node has too few cases. Once a terminal node is found we must decide how to classify all cases falling within it. One simple criterion is the plurality rule: the group with the greatest representation determines the class assignment. CART goes a step further: because each node has the potential for being a terminal node, a class assignment is made for every node whether it is terminal or not. The rules of class assignment can be modified from simple plurality to account for the costs of making a mistake in classification and to adjust for over- or under-sampling from certain classes. A common technique among the first generation of tree classifiers was to continue splitting nodes (growing the tree) until some goodness-of-split criterion failed to be met. When the quality of a particular split fell below a certain threshold, the tree was not grown further along that branch. When all branches from the root reached terminal nodes, the tree was considered complete. Once a maximal tree is generated, it examines smaller trees obtained by pruning away branches of the maximal tree. Once the maximal tree is grown and a set of sub-trees is derived from it, CART determines the best tree by testing for error rates or costs. With sufficient data, the simplest method is to divide the sample into learning and test sub-samples. The learning sample is used to grow an overly large tree. The test sample is then used to estimate the rate at which cases are misclassified (possibly adjusted by misclassification costs). The misclassification error rate is calculated for the largest tree and also for every sub-tree. The best sub-tree is the one with the lowest or near-lowest cost, which may be a relatively small tree. Cross validation is used if data are insufficient for a separate test sample.

In the search for patterns in databases it is essential to avoid the trap of over fitting or finding patterns that apply only to the training data. CART's embedded test disciplines ensure that the patterns found will hold up when applied to new data. Further, the testing and selection of the optimal tree are an integral part of the CART algorithm. CART handles missing values in the database by substituting surrogate splitters, which are back-up rules that closely mimic the action of primary splitting rules. The surrogate splitter contains information that is typically similar to what would be found in the primary splitter.

7 Experiment Results and Performance Analysis

The experiment system consists of two stages: model construction (training) and performance evaluation. The stressor – susceptibility interaction model was analyzed in detail (as illustrated in Figure 3) and the main causes of failures were identified. Analysis showed that the main cause of the failure was excess junction temperature and leakage current. A mathematical model was built relating the failure probability, leakage current and junction temperature. A failure simulation was carried out and the data set was generated. We attempted to predict the component temperature and leakage current for a given voltage and current. Data was generated by simulating circuit failure. 80% of the randomly selected data was used for training and remaining for testing and validation purposes. All the training data were standardized before training. The input parameters considered are the Voltage (V) and Current (I). Predicted outputs are the junction temperature and leakage current.

Parameter		**Value**
Population size		500
Mutation frequency		90%
Crossover frequency		60%
Number of demes		10
Program size	Initial	80
	maximum	256

Table 2. Parameters used by LGP

Parameter	**Value**
Population size	500
Number of mutations per chromosome	4
Crossover probability	0.9
Code length	40
Number of generations	300

Table 3. Parameters used by MEP

Parameter	Value
Population size	500
Mutation probability	0.05
Crossover probability (one point crossover)	0.3
Number of genes	3, 4, 5 and 6
Genes recombination	0.1
Genes transposition	0.1
Inversion	0.1

Table 4. Parameters used by GEP.

- **ANN Training**

We used a feedforward neural network with 2 hidden layers in parallel, 2 input neurons corresponding to the input variables and 2 output neurons. Initial weights were randomized between +0.3/-0.3 and learning rate and momentum used were 0.1 and 0.1, respectively. The training was terminated after 3500 epochs.

7.1 Performance and Results Achieved

Parameters used by the three GP variants are depicted in Tables 2-4. Figures 4 and 6 illustrate the fitness values of the evolved models using LGP for junction temperature and leakage current prediction. Figures 5 and 7 depict the average code length and best code length (program size) for the two prediction models using LGP. As illustrated in Figure 8, the MEP models converged after 300 generations. Following are the evolved functions using MEP. Table 5 summarizes the comparative performance of LGP, MEP, GEP, ANN and CART. For the test data, LGP and neural networks performed well for junction temperature and leakage current approximation respectively.

Leakage current model

```
(((cos(x[1] - (x[0] * x[0]) - (x[1] > (x[0] * x[0]) ? x[1] : (x[0] * x[0]) - (x[1] - (x[0] *
x[0]))))) > x[1] ? (cos(x[1] - (x[0] * x[0]) - (x[1] > (x[0] * x[0]) ? x[1] : (x[0] * x[0]) -
(x[1] - (x[0] * x[0]))))) : x[1]) / (sin(x[0] * x[0] + x[0]))) < ((fabs(0.060518)) / 0.0605182)
? (((cos(x[1] - (x[0] * x[0]) - (x[1] > (x[0] * x[0]) ? x[1] : (x[0] * x[0]) - (x[1] - (x[0] *
x[0]))))) > x[1] ? (cos(x[1] - (x[0] * x[0]) - (x[1] > (x[0] * x[0]) ? x[1] : (x[0] * x[0]) -
(x[1] - (x[0] * x[0]))))) : x[1])/(sin(x[0]*x[0]+ x[0]))) : ((fabs(0.0605182))/ 0.060518)
```

Temperature model

```
0.80123294778283 > (fabs((x[0] + Log2(x[0])) / ((Lg(x[0])) > x[0] ? (Lg(x[0])) : x[0]))) ?
0.80123294778283 : (fabs((x[0] + Log2(x[0])) / ((Lg(x[0])) > x[0] ? (Lg(x[0])) : x[0])))
```

	Root Mean Squared Error (RMSE)				
	LGP	MEP	GEP	ANN	DT
Training data					
Junction Temperature	0.00948	**0.00593**	0.0237	0.0069	0.0180
Leakage Current	**0.00493**	0.00829	0.0194	0.00589	0.0221
Test data					
Junction Temperature	**0.00911**	0.01034	0.0200	0.01278	0.028
Leakage Current	0.00493	0.010032	0.0236	**0.00359**	0.034

Table 5. Performance comparison among the different paradigms

Figures 9-16 illustrate the performance (RMSE and correlation coefficient) of GEP for leakage current and junction temperature approximation. The best temperature approximation (lowest RMSE) and correlation coefficient was obtained using a chromosome size of 75 (Figures 9 and 10) and using a gene size of 5 (Figures 11 and 12). The best leakage current approximation (lowest RMSE) and correlation coefficient was obtained using a chromosome size of 75 (Figures 13 and 14) and using a gene size of 5 (Figures 15 and 16).

Figures 17-18 depicts the decision trees developed using CART algorithm for junction temperature and leakage current.

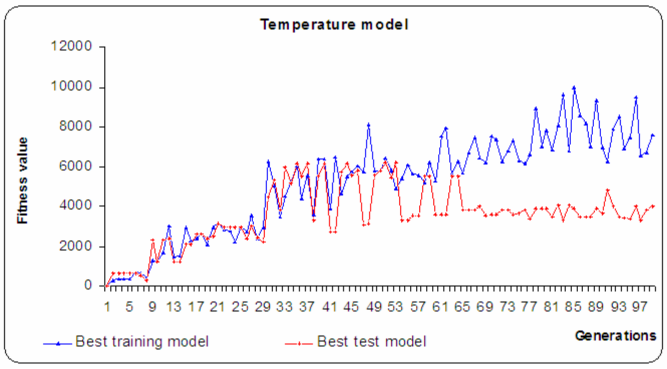

Figure 4. LGP evolved models for temperature

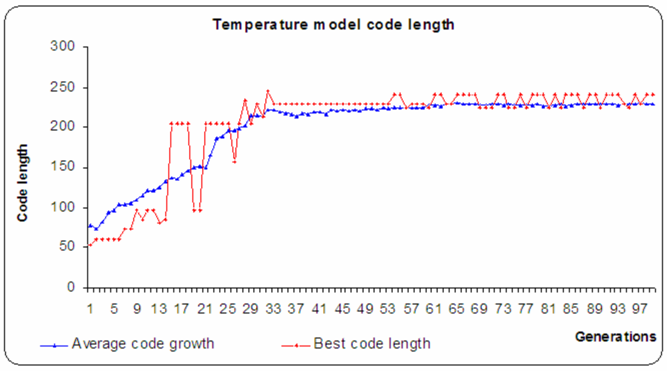

Figure 5. LGP program size growth for the temperature model

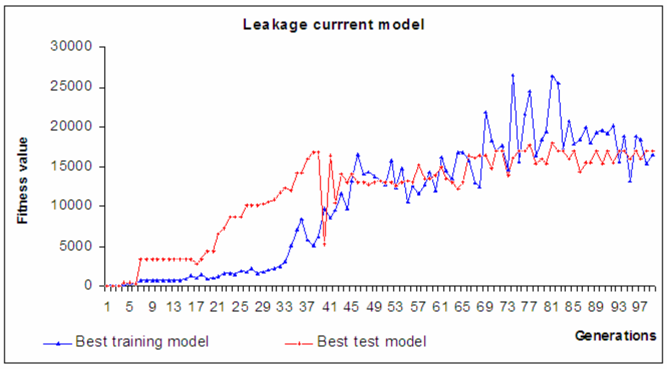

Figure 6. LGP evolved models for leakage current

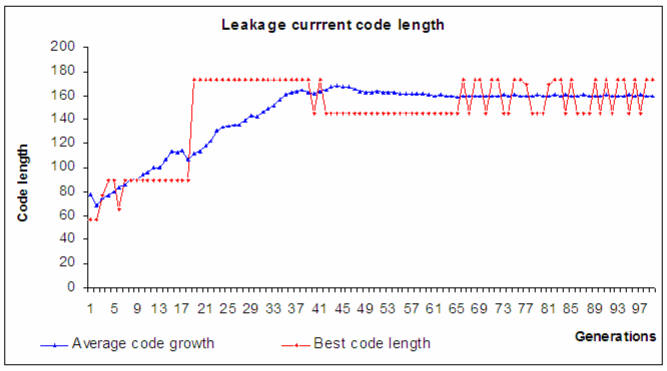

Figure 7. LGP program size growth for leakage current

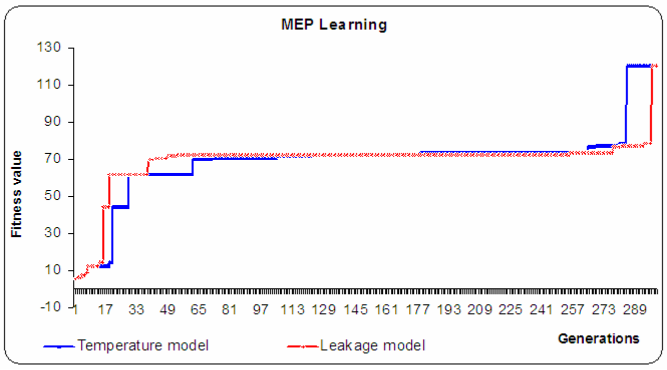

Figure 8. MEP training for the two models

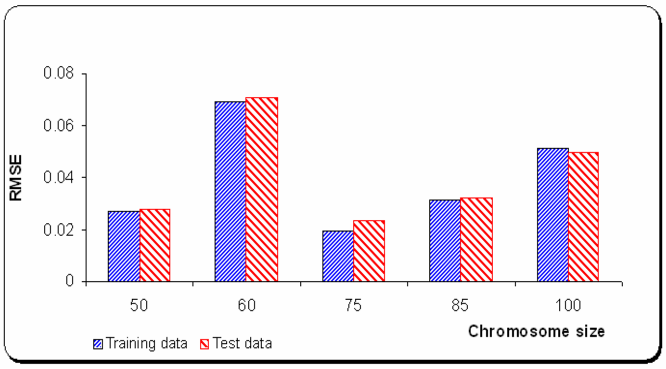

Figure 9. GEP temperature approximation error for different chromosome sizes

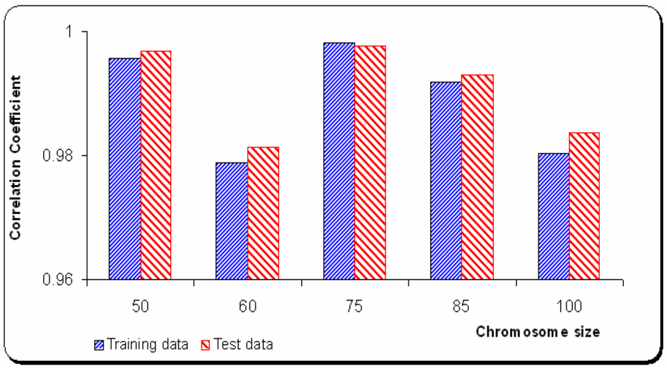

Figure 10. GEP correlation coefficient for temperature approximation for different chromosome sizes

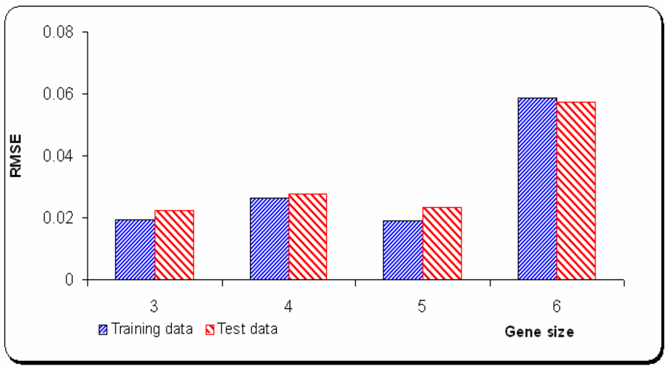

Figure 11. GEP temperature approximation error for different gene sizes

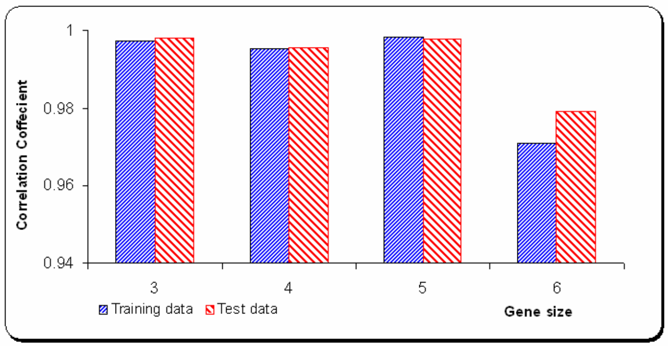

Figure 12. GEP temperature approximation correlation coefficient for different gene sizes

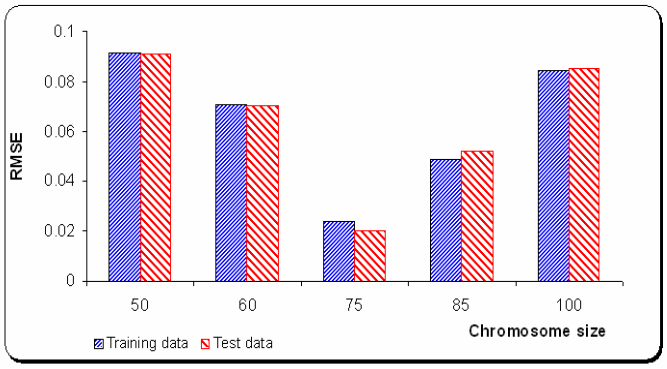

Figure 13. GEP leakage current approximation error for different chromosome sizes

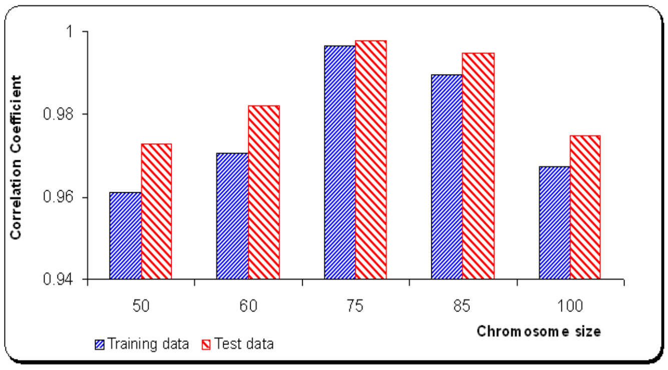

Figure 14. GEP correlation coefficient for leakage current approximation for different chromosome sizes

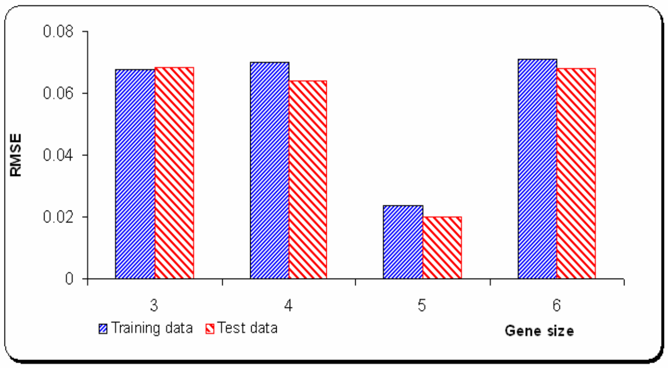

Figure 15. GEP leakage current approximation error for different gene sizes

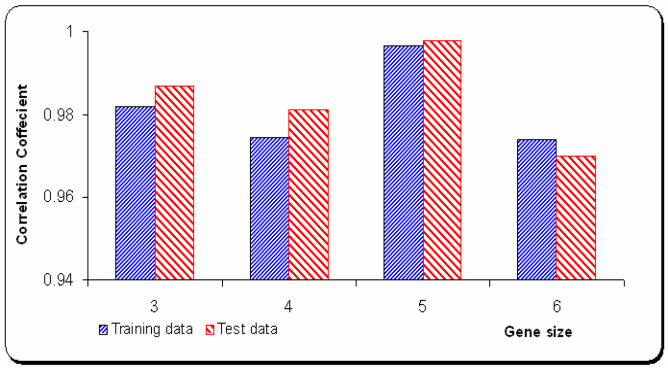

Figure 16. GEP leakage current approximation correlation coefficient for different gene sizes

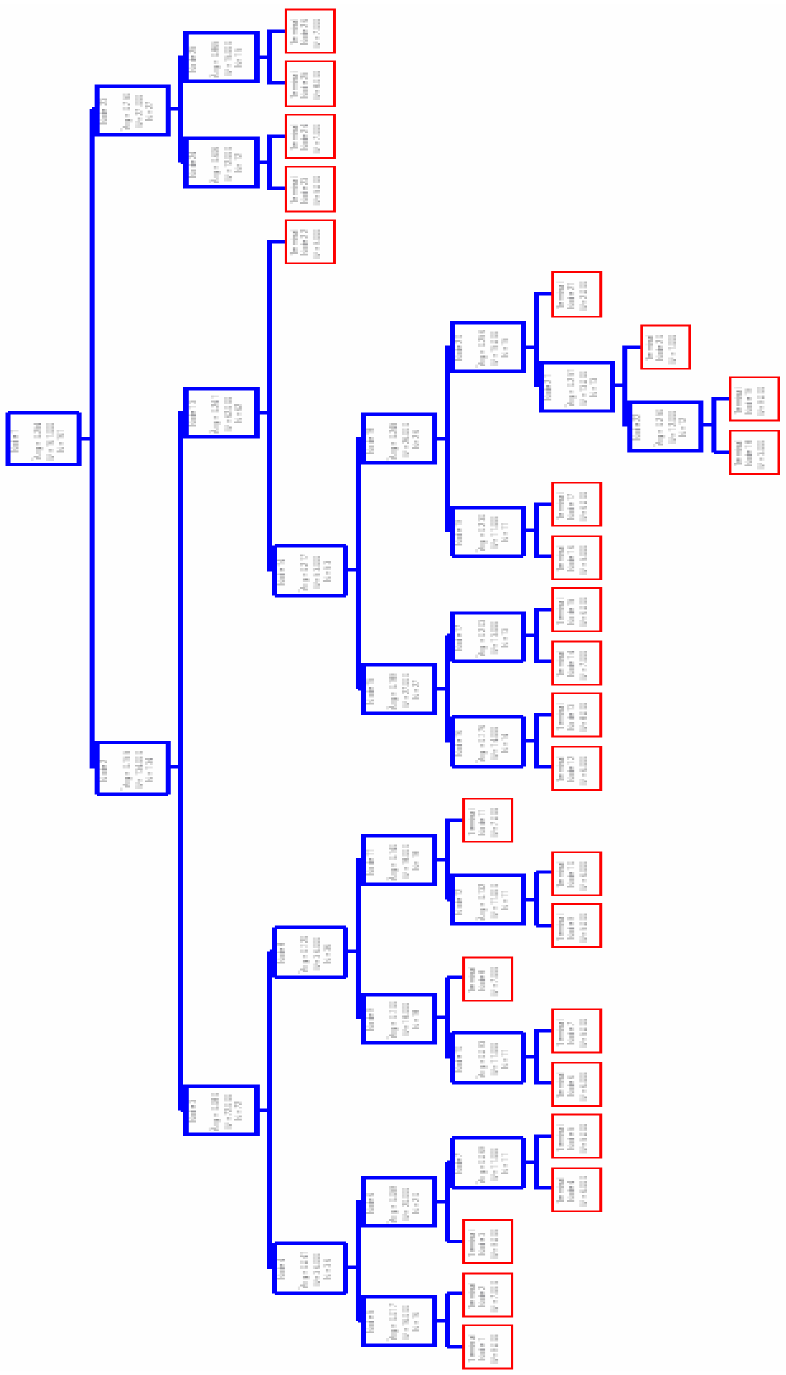
Figure 17. Constructed decision tree for temperature approximation

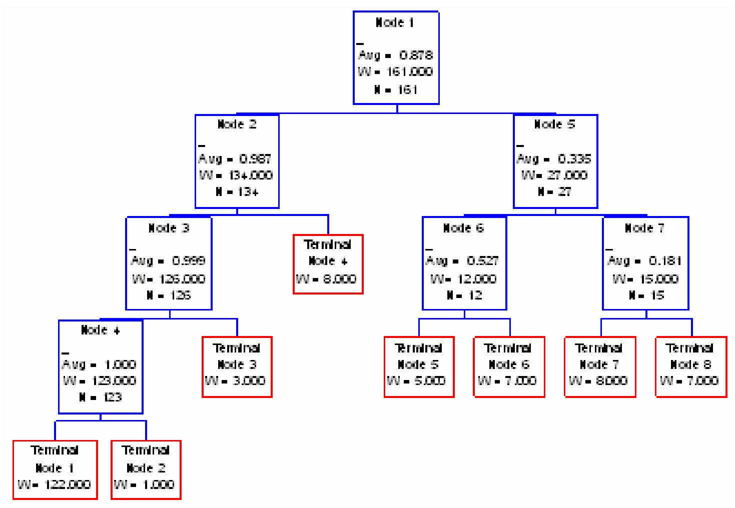

Figure 18. Constructed decision tree for leakage current approximation

8 Conclusions

In this paper, we attempted to predict the failures of electronic circuits and systems using three variants of genetic programming and the performance were compared using artificial neural networks and decision trees. The proposed GP models seems to work very well with LGP giving the optimal performance for modelling leakage current and junction temperature. Compared to neural network and decision trees, an important advantage of the GP models is its simplicity in implementing directly in the hardware itself. As depicted in Section 7 (MEP evolved functions), the massive neural network could be replaced by simple functions using hardware or light software.

The developed models should be also reliable during worst conditions. Our future research will be targeted in evaluating the developed GP models for robustness and handling of noisy and approximate data that are typical in circuits. The problem modeling using stressor– susceptibility interaction method can be widely applied to a wide range of electronic circuits or systems. However, it requires intense knowledge on the circuit behavior to model the various dependent input parameters to predict the results accurately.

Acknowledgements

This research was supported by the International Joint Research Grant of the IITA (Institute of Information Technology Assessment) foreign professor invitation program of the MIC (Ministry of Information and Communication), Korea.

Authors would like to thank the anonymous referees for the technical suggestions and remarks which helped to improve the contents and the quality of presentation.

References

[Abraham and Grosan, 2005] Abraham, A, and Groşan, C., Genetic Programming Approach for Fault Modeling of Electronic Hardware, 2005 IEEE Congress on Evolutionary Computation (CEC'05), Edinburgh, UK, IEEE Press, ISBN 0-7803-9364-3, pp. 1563-1569, 2005.

[Abraham, 2005] Abraham, A., Artificial Neural Networks, Handbook for Measurement Systems Design, Peter Sydenham and Richard Thorn (Eds.), John Wiley and Sons Ltd., London, ISBN 0-470-02143-8, pp. 901-908, 2005.

[Abraham, 2000] Abraham A., A Soft Computing Approach for Fault Prediction of Electronic Systems, In Proceedings of The Second International Conference on Computers in Industry, Published by The Bahrain Society of Engineers, Majeed A Karim et al (Eds.), pp. 83-91, 2000.

[Abraham and Nath, 1999] Abraham A. and Nath B., Failure Prediction of Critical Electronic Systems in Power Plants Using Artificial Neural Networks, First International Power and Energy Conference, INTPEC99, Australia, November 1999.

[Chan, 1994] Chan A.H., A formulation of environmental stress testing and screening, Proceedings of the Annual Reliability and Maintainability Symposium (IEEE Reliability Society), pp 99-104, January 1994.

[Arsenault and Roberts, 1980] Arsenault J.E. and Roberts J.A., Reliability and maintainability of electronic systems, Computer Science press, Maryland, 1980.

[Banzhaf et al., 1998] Banzhaf. W., Nordin. P., Keller. E. R., Francone F. D., Genetic Programming: An Introduction on The Automatic Evolution of Computer Programs and its Applications, Morgan Kaufmann Publishers, Inc., 1998.

[Brombacher, 1995] Brombacher A.C, Reliability by Design, Wiley, 1995.

[Jenson, 1995] Jenson F., Electronic Component Reliability, Wiley, 1995.

[Klion, 1992] Klion J., Practical Electronic Reliability Engineering, VNR Edition, 1992.

[Fuqua, 1987] Fuqua N.B., Reliability Engineering for Electronic Design, Marcel Dekker, 1987.

[Oltean and Grosan, 2003] Oltean M. and Grosan C., Evolving Evolutionary Algorithms using Multi Expression Programming. Proceedings of the 7th European Conference on Artificial Life, Dortmund, Germany, pp. 651-658, 2003.

[Breiman et al., 1984] Breiman, L., Friedman, J., Olshen, R., and Stone, C. J, Classification and Regression Trees, Chapman and Hall, New York, 1984.

[Ferreira, 2001] Ferreira C., Gene Expression Programming: A new adaptive algorithm for solving problems — Complex Systems, Vol. 13, No. 2, pp. 87–129, 2001.

[Yamada and Komada, 2004] Yamada Y. , and Komoda H., An example of fault site localization on a 0.18 µm CMOS device with combination of front and backside techniques, Microelectronics Reliability, Volume 44, No. 5 , pp. 771-778, 2004.

[Adel Mohsena and El-Yazeedb, 2004] A.K. Adel Mohsena and M.F. Abu El-Yazeedb, Selection of Input Stimulus for Fault Diagnosis of Analog Circuits Using ARMA Model, AEU - International Journal of Electronics and Communications, Volume 58, No. 3 , pp. 212-217, 2004.

[El-Gamal and Abdulghafour, 2003]M. A. El-Gamal and M. Abdulghafour, Fault isolation in analog circuits using a fuzzy inference system, Computers & Electrical Engineering, Volume 29, No. 1 , pp. 213-229, 2003.

[Blyzniuk et al., 2001] M. Blyzniuk, I. Kazymyra, W. Kuzmicz, W. A. Pleskacz, J. Raik and R. Ubar, Probabilistic analysis of CMOS physical defects in VLSI circuits for test coverage improvement, Microelectronics Reliability, Volume 41, No. 12, pp. 2023-2040, 2001.

[Dai and Xu, 1999] Yisong Dai and Jiansheng Xu, Analog circuit fault diagnosis based on noise measurement, Microelectronics and Reliability, Volume 39, No. 8, pp. 1293-1298, 1999.

[Schienle et al., 1999] M. Schienle, Th. Zanon and D. Schmitt-Landsiedel, Improved SRAM failure diagnosis for process monitoring via current signature analysis, Microelectronics Reliability, Volume 39, Nos. 6-7, pp. 1009-1014, 1999.

[Catelani and Giraldi, 1999] M. Catelani and S. Giraldi, A measurement system for fault detection and fault isolation of analog circuits, Measurement, Volume 25, No. 2, pp. 115-122, 1999.

[Toczek et al., 1998] W. Toczek, R. Zielonko and A. Adamczyk, A method for fault diagnosis of nonlinear electronic circuits, Measurement, Volume 24, No. 2, pp. 79-86, 1998.