Data mining based automated reverse engineering and defect discovery

James F. Smith III*, ThanhVu H. Nguyen

Naval Research Laboratory, Code 5741, Washington, D.C., 20375-5000

ABSTRACT

A data mining based procedure for automated reverse engineering and defect discovery has been developed. The data mining algorithm for reverse engineering uses a genetic program (GP) as a data mining function. A GP is an evolutionary algorithm that automatically evolves populations of computer programs or mathematical expressions, eventually selecting one that is optimal in the sense it maximizes a fitness function. The system to be reverse engineered is typically a sensor that may not be disassembled and for which there are no design documents. The sensor is used to create a database of input signals and output measurements. Rules about the likely design properties of the sensor are collected from experts. The rules are used to create a fitness function for the GP allowing GP based data mining. This procedure incorporates not only the experts' rules into the fitness function, but also the information in the database. The information extracted through this process is the internal design specifications of the sensor. These design properties can be used to create a fitness function for a genetic algorithm, which is in turn used to search for defects in the digital logic design. Significant theoretical and experimental results are provided.

Keywords: data mining, knowledge discovery, genetic programs, genetic algorithms, reverse engineering, defect discovery

1. INTRODUCTION

An engineer has a borrowed system and must determine what design flaws may be present. The owner of the system will not allow it to be disassembled, and the design specifications for the system are not available. The engineer however has access to good information about the period in which the system was designed and constructed, the cost of the system and as such the most likely parts the system used. The engineer also knows that the original engineers who designed the system were subject to significant cost constraints, so the design is very efficient. Finally the engineer has access to a historical database of input and output data associated with the system.

Couldn't the engineer approach the system in a trial an error fashion and eventually find the flaw? It might be possible to do this, but if the system is complex the engineer might not live long enough to find the problem. Finally, experimental time can be extremely costly, if the engineer can predetermine the most likely input prior to actual experimentation in the lab, then the savings can be enormous. So it would be useful if an automated procedure were available for discovering design flaws.

To solve this problem, a data mining[1] based procedure for automated reverse engineering and defect discovery has been developed. The data mining algorithm for reverse engineering uses a genetic program (GP) as a data mining function. A genetic program is an algorithm based on the theory of evolution that automatically evolves populations of computer programs or mathematical expressions, eventually selecting one that is optimal in the sense it maximizes a measure of effectiveness, referred to as a fitness function[2-5]. The system to be reverse engineered is typically a sensor. The sensor is used to create a database of input signals and output measurements. Rules about the likely design properties of the sensor are collected from experts. The rules are used to create a fitness function for the genetic program. Genetic program based data mining is then conducted[3-5]. This procedure incorporates not only the expert's rules into the fitness function, but also the information in the database. The information extracted through this process is the internal design specifications of the sensor. The design properties extracted through this process can be used to

* Correspondence: Email: jfsmith@drsews.nrl.navy.mil

Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2005, edited by Belur V. Dasarathy, Proceedings of SPIE Vol. 5812 (SPIE, Bellingham, WA, 2005) · 0277-786X/05/$15 · doi: 10.1117/12.602077

create a fitness function for a genetic algorithm[6-15] (GA), which is in turn used to search for defects in the sensor design. Unlike a GP, a GA manipulates strings of numbers, not computer programs, with the intent of producing optimal results by maximizing a fitness function. Significant theoretical and experimental results are provided.

Section 2 introduces the ideas of genetic algorithms and genetic programs. Section 3 discusses data mining and the use of a genetic program as a data mining function. Section 4 examines the digital logic design to be reverse engineered using genetic program based data mining. Section 5 explains the genetic program's terminal set, function set, fitness function, stopping criteria, parameters, and data mining results. Section 6 discusses how a genetic algorithm can be used to automatically discover defects in the design of digital logic. Section 7 gives an example of a defect discovered by the genetic algorithm in the design of the digital logic reverse engineered by the genetic program. Finally, section 8 provides conclusions.

2. GENETIC ALGORITHMS AND GENETIC PROGRAMS

The following subsections describe genetic algorithms and genetic programs. Differences between the algorithms' structure, applications as well as some characteristics specific to the GP used for data mining are discussed.

2.1 Genetic algorithm

A genetic algorithm is an optimization method that manipulates a string of numbers in a manner similar to how chromosomes are changed in biological evolution[6-15]. An initial population made up of strings of numbers is chosen at random or is specified by the user. Each string of numbers is called a "chromosome" and each numbered slot is called a "gene." A set of chromosomes forms a population. Each chromosome represents a given number of traits that are the actual parameters that are being varied to optimize the "fitness function". The fitness function is a performance index that is to be maximized.

The operation of the genetic algorithm proceeds in steps. Beginning with the initial population, "selection" is used to choose which chromosomes should survive to form a "mating pool." Chromosomes are chosen based on how fit they are (as computed by the fitness function) relative to the other members of the population. More fit individuals end up with more copies of themselves in the mating pool so that they will more significantly effectthe formation of the next generation. Next, two operations are taken on the mating pool. First, "crossover" (which represents mating, the exchange of genetic material) occurs between parents.

In crossover, a random spot is picked in the chromosome, and the genes after this spot are switched with the corresponding genes of the other parent. Following this, "mutation" occurs. Mutation is a change of the value of a randomly selected gene. After the crossover and mutation operations occur, the resulting strings form the next generation and the process is repeated. Another process known as "elitism" is also employed. Elitism consists of copying a certain number of fittest individuals into the next generation to make sure they are not lost from the population. Finally, a termination criterion is used to specify when the algorithm should stop, e.g., when a preset maximum number of generations has been reached, the fitness has gained its maximum possible value or that the fitness has not changed significantly in a certain number of generations.

2.2 Genetic program

A genetic program[2-5] is a problem independent method for automatically creating graphs that represent computer programs, mathematical expressions, digital circuits, etc. Like a genetic algorithm it evolves a solution using Darwin's principle of survival of the fittest. Unlike the genetic algorithm, of which it can be considered an extension: its initial, intermediate, and final populations are computer programs.

Like a genetic algorithm, the individuals that make up the population are subject to selection, crossover, mutation and elitism. The crossover and mutation operations are constrained to produce structurally valid offspring, i.e., functional computer programs, digital circuits, etc. Two additional GP operations not used with GAs are architecture altering steps[2] (AAS) and symmetrical replication (SR).

AAS is a method introduced to improve the GP's convergence. This function randomly changes genes in elite chromosomes, i.e., typically the 50 chromosomes with the highest fitness. For the digital logic diagram of Figure 1, AAS might consist of changing H_1 to H_3 or SUM_sig123 to SUM_sig2. This notation is explained in greater detail in sections 4 and 5.

SR refers to replacing a branch entirely with another branch that already existed on the chromosome. SR is similar to mutation, it is not used often and its purpose is to break out of the local fitness landscape, if possible. Like mutation, this replication happens after a specific number of generations that have the same maximum fitness. SR is designed to help the GP escape the neighborhood of what may be a local maximum so that the GP can locate the global maximum. Finally, it is generally applied to only a small random set of the elite chromosomes, this allows the GP to better search the space while maintaining diversity in the population.

GPs require a terminal set and function set as inputs. The terminals are the actual variables of the problem. These can include a variable like "x" used as a symbol in building a polynomial and also real constants. The function set consists of a list of functions that can operate on the variables. For example, in a previous application of a GP as a data mining function to evolve fuzzy decision trees symbolically[3-5], the terminal set consists of fuzzy membership functions and the function set consists of logical connectives like *AND*, *OR* and logical modifiers like *NOT*. Since fuzzy logic allows more than one mathematical representation of *AND* and *OR*, the function set could be complicated. When the GP is used as a data mining function, a database of input and output information is required. In the case of fuzzy decision trees the database represented a collection of scenarios about which the fuzzy decision tree to be evolved would make decisions. The database also had entries created by experts representing decisions about the scenarios. The optimal fuzzy decision tree would be the one that could most closely reproduce the experts' decisions about the scenarios. When the GP is used as a data mining function for evolving digital logic (DL), the database contains inputs to the DL as well as measured outputs. The experts' opinions are manifested in the selection of the input and associated output to be included in the database. For the DL case an additional form of input consisting of "rules" about DL construction are included. Finally, the terminal set, function set, database to be data mined, and rules for DL construction are described in greater detail in sections 4 and 5.

3. EVOLUTIONARY ALGORITHM BASED DATA MINING

Data mining is the efficient extraction of valuable non-obvious information embedded in a large quantity of data[1]. Data mining consists of three steps: the construction of a database that represents truth; the calling of the data mining function to extract the valuable information, e.g., a clustering algorithm, neural net, genetic algorithm, genetic program, etc; and finally determining the value of the information extracted in the second step, this generally involves visualization.

When used for reverse engineering, the GP, typically data mines a database to determine a graph-theoretic structure, e.g., a system's DL diagram or an algorithms flow chart or decision tree[3-5]. The GP mines the information from a database consisting of input and output values, e.g., a set of inputs to a sensor and its measured outputs. GP based data mining will be applied to the construction of the DL described in section 4.

To use the genetic program it is necessary to construct terminal and function sets relevant to the problem. Before the specific terminal and function sets for the reverse engineering problem are described, a more detailed description of the digital logic to be considered will be given in the section 4.

4. THE DIGITAL LOGIC TO BE REVERSE ENGINEERED

The DL to be reverse engineered is depicted in Figure 1. This DL is not known to the GP. The GP only has access to a database of input signals to the DL and measured output, as well as, a database of rules provided by experts for building the DL.

In Figure 1 the DL consists of three input channels each with a sensor attached. The sensors receive signals from sources one, two and three. Only measurements from sources two, the central source is of interest. Due to the geometry of the sources and properties of the sensors only sensor two can receive emissions from source two that are

significant. Unfortunately, sensor two's measurement may be corrupted by emissions from sources one and three. The digital logic is constructed so that if there were significant corruption of sensor two's measurements, then the final OR-gate returns unity, so the measurements can be ignored.

4.1 Description of the DL and its elements

In Figure 1 there are a number of DL elements depicted that are used repeatedly. DL components and signals will ultimately become elements of the GP's terminal and function sets. The sensors in Figure 1 will receive an analog signal and convert it to a digital form, i.e., they will map real-valued input to the set of integers. A sampling window of size N is used, i.e., the signal is sampled every Δt seconds for a total of N samples in that window. The sample is indicated by the vector $\vec{s}_j$ in (1) with sampling beginning at time t_o. The j-subscript implies the signal originate in the j^{th} source, where $j=1,2,3$,

$$\vec{s}_j = [s_j(t_o), s_j(t_o + \Delta t), \quad \ldots \quad , s_j(t_o + (N-1)\cdot \Delta t)]. \tag{1}$$

The DL function, *SUM*, given explicitly in (2), represents the logarithmic sum of the absolute value of the time components of the digitized input that has been received for a single window of length N.

$$SUM(\vec{s}_j) = \ln\left[\sum_{k=1}^{N} \left| s_j(t_o + (k-1)\cdot \Delta t) \right|\right] \tag{2}$$

The elements labeled H_i, for $i=1,2,3$, are Heaviside step functions as given in (3). If the input is greater than or equal to a threshold, τ_i, for $i=1,2,3$; then a value of unity is transmitted, otherwise a zero is transmitted.

$$H_i(s) = \begin{cases} 1, & if \quad s \geq \tau_i \\ 0, & if \quad s < \tau_i \end{cases} \tag{3}$$

The DL function, *MAX*, given in (4), returns the natural logarithm of the maximum absolute value of the time components of the input signal for a single window of length N. The element labeled *DIFF*, takes the difference between input to its first and second terminals as indicated in (5).

$$MAX(\vec{s}_j) = \ln\left[\bigvee_{k=1}^{N} \left| s_j(t_o + (k-1)\cdot \Delta t) \right|\right] \tag{4}$$

$$DIFF(I_1, I_2) = I_1 - I_2 \tag{5}$$

The DL function, *Ordelay3*, takes only Boolean inputs, i.e., it expects zero or one as an input. It waits until it has three consecutive inputs from three consecutive time windows, hence the "3" in its name. Once it receives three consecutive inputs, it yields as an output the maximum of its inputs. Also not depicted, but used in the GP's function set are *Anddelay3*, which takes three inputs of zero or one corresponding to three consecutive time windows and yields as output the minimum of its inputs. *Anddelay2* uses only two inputs and returns their maximum. Finally, the symbols labeled *AND3*, *OR3*, *AND2*, and *OR2* are the conventional logical connectives *AND* and *OR*, with the numerical designation indicating the number of inputs expected, e.g., *AND3* expects three Boolean inputs.

The signals are additive, at any given time sensor two may record a superposition of the three sources' transmissions, which is represented by $s_1(t) + s_2(t) + s_3(t)$. If the three sensors' signals are of sufficient magnitude then this is characteristic of corruption and the final *OR* in Figure 1 returns unity.

5. TERMINAL SET, FUNCTION SET, FITNESS FUNCTION, PARAMETERS, AND DATA MINING RESULTS

This section describes the terminal set, function set, data base construction, fitness function, the GP's essential parameters and the results of data mining for the DL represented by Figure 1. The description is given in terms of DL elements and properties, but the genetic program based reverse engineering technique is very general and can be applied to any system that can be described in a graph theoretic language, e.g., decision processes described in terms of decision trees[3-5].

5.1 Terminal and function sets

The terminal set consists of the following elements:

$$T=\{SUM\_sig123,\ MAX\_sig123,\ SUM\_sig1,\ MAX\_sig1,\ SUM\_sig3,\ MAX\_sig3\} \tag{6}$$

where

$$SUM\_sig123 = SUM(\vec{s}_1 + \vec{s}_2 + \vec{s}_3) \tag{7}$$

$$MAX\_sig123 = MAX(\vec{s}_1 + \vec{s}_2 + \vec{s}_3) \tag{8}$$

$$SUM\_sig1 = SUM(\vec{s}_1) \tag{9}$$

$$MAX\_sig1 = MAX(\vec{s}_1) \tag{10}$$

$$SUM\_sig3 = SUM(\vec{s}_3) \tag{11}$$

$$MAX\_sig3 = MAX(\vec{s}_3) \tag{12}$$

All sensor measurements begin at time, t_o.

The function set consists of the following elements:

$$F=\{AND3,\ OR3,\ AND2,\ OR2,\ ANDDELAY3,\ ORDELAY3,\ H1,\ H2,\ H3,\ DIFF\} \tag{13}$$

The function *ANDDELAY3* is not used in Figure 1. By including it, the GP's ability to discriminate against extraneous functions is emphasized.

The GP's goal is to evolve a solution that represents Figure 1. The GP's ability to do this will be determined largely by the fitness function and the underlying databases to be discussed. The chromosome to be evolved by the GP in prefix notation that represents Figure 1 is given in (14).

$$\begin{aligned} &OR2\ ORDELAY3\ AND3\ H_3\ SUM\_SIG3\ H_2\ DIFF\ SUM\_SIG3\ SUM\_SIG123\ H_1\ MAX\_SIG123 \\ &ORDELAY3\ AND3\ H_3\ SUM\_SIG1\ H_2\ DIFF\ SUM\_SIG1\ SUM\_SIG123\ H_1\ MAX\_SIG123 \end{aligned} \tag{14}$$

5.2 Creating the database

The database used for constructing the fitness function consists of inputs to the system to be reverse engineered

and measured output. For the digital logic diagram of Figure 1 the input is a collection of signals from sources one through three and the associated measured output. The output of the digital logic represented by Figure 1 will consists entirely of zeros and ones. The fact that the system maps real valued input into the doublet set {0,1} complicates the reverse engineering process. For a case like this the rule set used to construct the fitness function is of paramount importance if a unique solution is to be found. These rules correspond to conditions provided by human experts regarding what elements the logic may have, how they are interconnected, and their abundance.

5.3 Calculating the fitness based on rules and parsimony pressure

This subsection discusses a few of the rules used to construct the fitness function. Typically, each rule represents a desirable feature that the final solution must have or might have. Toward this goal, candidate solutions, i.e., chromosomes within the GP's evolving population are given a higher fitness for showing characteristics consistent with the rules.

5.3.1 Rule examples and their rewards

For the digital logic depicted in Figure 1, a number of rules can be developed without knowing the specific design. For example, when actually using the DL or upon close analysis of input -output relations it is observed that the logic waits for a period corresponding to three consecutive time windows before making a declaration. Based on the logic's function the following rules are formulated

Rule 1: There must be a three time window time delay built into the DL.

Other rules provided by experts are

Rule 2: The digital logic must have a delay near the output level.

Since data and expert intuition seem to indicate that there is an *AND* or *OR* at the end, the rule follows

Rule 3: The last operation is a Boolean.

It follows from the properties of the hardware implementation of the Heaviside step function and the DIFF operator that

Rule 4: The Heaviside step function must take an integer input

Rule 5: The DIFF operator must have a two signal input

Likewise, Rule 6 represents commonsense

Rule 6: There is no point in having a functional composition of Heaviside step functions.

In each case when the GP determines a candidate solution satisfies the rule, the fitness for that solution is incremented by unity. Other rules were used, but the above set captures the spirit of how rules are obtained and rewards are awarded.

So it is observed some of the rules, like Rule 1 arise from an observation, but others like Rule 3 arise from properties of the logic elements or in the case of Rule 6 from common sense.

5.3.2 Controlling bloat

One of the fundamental problems associated with genetic programming is controlling bloat. Bloat[16] refers to excessive growth of candidate solutions. It is found empirically that every 50 generations the length of candidate solutions, i.e., chromosomes within a GP's evolving population increases by a factor of three. Various procedures have

been invented for controlling bloat. Two of the most popular are the Koza depth limit[2] and parsimony pressure[3-5, 16]. Another technique that involves the use of computer algebra to control bloat when using a GP to evolve mathematical expressions has been invented recently[3]. This algebraic approach was not used for the study at hand but may be applied in a future extension of this work.

The procedure applied for evolving the digital logic of Figure 1 was parsimony pressure. The basic fitness is the fitness determined by the input-output and rule database. The parsimony pressure for a given chromosome is calculated by multiplying the parsimony coefficient, α, by the length of the chromosome. The overall fitness is determined by subtracting the parsimony pressure from the basic fitness. Thus given two candidate solutions with the same basic fitness, the shorter of the two will have the higher overall fitness. The parsimony pressure can be regarded as a numerical representation of Occam's razor[17].

5.4 Setting the GP's parameters

The GP requires the specification of many parameters for efficient operation. Modification to these parameters can have a significant impact on the convergence time and solution found by the GP. Some of the most important parameters are population size, database size, probability of mutation, probability of crossover, and parsimony pressure.

The size of the population of evolving chromosomes is very important, a large population can represent a significant degree of diversity, potentially resulting in the determination of a better solution. Increased population size can also have a significant effect on run time. The default population size used for the DL depicted in Figure 1 is 1000.

Increasing the size of the database can have a significant effect. A larger database may contribute to the determination ultimately of a unique solution or a smaller final class of high fitness potential solutions. Unfortunately, given that the digital logic in Figure 1 maps from real-valued inputs to the doublet set {0,1}, the benefits of a larger database are limited. It becomes important to embed high quality rules in the fitness function, rewarding good candidate solutions, i.e., those consistent with the rules and penalizing poor quality chromosomes.

The probabilities of crossover and mutation can also be very important. Low probabilities of crossover and mutation can result in only limited regions of the fitness landscape being explored, i.e., the GP will get locked into a local neighborhood and not escape.

Two other parameter dependent operations are AAS and SR which are applied on to the elite chromosomes, i.e., the top 50 with the highest fitness. When applying AAS, the GP selects about 63% of the elite chromosomes in each generation. Likewise, SR is applied to seven percent of the elite chromosomes that have been through 12 generations where the maximum fitness has not changed.

5.5 Data mining results

The GP successfully discovered the DL of Figure 1 through data mining after 60 generations using the parameters given in the previous subsection. The SR operation proved to be very effective and helped accelerate convergence significantly.

6. AUTOMATED DEFECT DISCOVERY USING A GENETIC ALGORITHM

The second phase of this work, i.e., automatic defect detection involves use of a genetic algorithm instead of a genetic program. Automatic defect detection as presented here is an optimization problem as opposed to a data mining problem and as such it will be treated in less detail than reverse engineering.

The function of the DL in Figure 1 is to determine if emissions from sources one and three are corrupting the measured emission from source two. If only data from source two is measured then the DL is expected to return a zero, but if sensor two's measurements are corrupted by contributions from sources one and three then the DL returns unity. If the output of the DL is unity then measurement is discarded. The intention of the design engineers was that the DL would allow relatively uncorrupted measurements of the output of source two, to be recorded.

Fortunately, sources one and three do not always emit in the same time window as source two, allowing the output of source two to be measured. Source two is a sufficiently weak emitter that any energy from it that goes into sensors one or three can be neglected. In addition, it is assumed that the magnitude and phase of the signal from source one measured by sensor two is the same as the magnitude and phase of the signal measured by sensor one. The same assumption is made about measurements of the signal from source three by sensor three and sensor two.

Given the design indicated by Figure 1, if the original design engineers were not careful, it might be possible for sensors one and three to receive signals $\vec{s}_1$ and $\vec{s}_3$ of sufficiently low magnitude that the DL does not recognize them as corruption. At the same time they obscure the characteristics of the desired signal, $\vec{s}_2$ broadcasts by the source two and received by sensor two. Such an event would represent a significant defect. It is this defect that the GA actually found. The defect found by the GA reflects the fitness function used. It is possible for the GA to find many other defects given other fitness functions.

The defect signal described above, that will escape detection by the DL takes the form

$$\vec{S}_D = \vec{s}_1 + \vec{s}_2 + \vec{s}_3, \tag{15}$$

such that there is so little variation in the magnitude of $\vec{S}_D$, that the signature form of $\vec{s}_2$ is no longer observed. One way of potentially satisfying the constraints is to look for a superposition signal, $\vec{S}_D$, such that it is flat, i.e., has no variation and the signals $\vec{s}_1$ and $\vec{s}_3$ do not result in a declaration of unity by the DL.

The DL reverse engineered by the GP yields unity, if the signals $\vec{s}_1$ and $\vec{s}_3$ are of sufficient magnitude during any three consecutive time windows. In this section, it is assumed that each time window has a length five, thus three intervals give a time window of length 15. The selection of a time window of length five was made to provide a simple example. It is assumed that the form of the signal, $\vec{s}_2$, broadcast by source two is known, whereas the signals broadcast by sources one and three are unknown and may vary in time.

The GA will attempt to find signals $\vec{s}_1$ and $\vec{s}_3$ such that $\vec{S}_D$ shows little variation over three consecutive time windows while at the same time the DL declares zero. The fitness function includes rules to facilitate finding this solution. It is important to observe that the DL is built into the fitness function and used for numerical calculation.

The chromosomes for this application represent the values of candidate solutions for $\vec{s}_1$ and $\vec{s}_3$. It is not necessary to solve for $\vec{s}_2$, since it is assumed to be known. Also, unlike GP based data mining, there is no database required for automatic defect discovery.

The chromosome size is 2 x length(time window) x 3, or 30. The factor of two arises because the chromosome represents both $\vec{s}_1$ and $\vec{s}_3$; the factor of three, because measurements from three consecutive time windows are included for both signals. The three time window measurements of $\vec{s}_1$ are encoded in the first half of the chromosome, positions one to 15. The second half of the chromosome, positions 16 to 30, represents $\vec{s}_3$. Chromosomes were selected to represent both signals to allow genetic information to be exchanged between the signals through crossover.

The GA uses the following parameter values: the population size is 1000, chromosome size 30, the probability of crossover is 90 percent, and probability of mutations 10 percent. The elitism size, i.e., the number of highest fitness individuals automatically retained from the previous generation is 50.

The GA uses as stopping criteria that the fitness must exceed a threshold, .99 or that 1000 generations have passed. These parameters were determined based on experience with the DL problem and GAs.

7. AN EXAMPLE OF AUTOMATIC DEFECT DISCOVERY

The program converges after about 300 generations. The following assumption is made: over three consecutive time windows sensor two measures the following data [-1, 2, -3, 20, -4, -2, 18, -1, 1, 2 , -1, -2, 17, -1, 1]. For the assumed valued of sensor two's measurements the genetic program found the following solution for $\vec{s}_1$ and $\vec{s}_3$ over the same three consecutive time windows

$$\text{sensor one measurements} = [\text{-10, -22, 24, 31, 58, 41, -51, -17, -46, -30, -25, -10, 13, -29, 9 }] \quad (16)$$

$$\text{sensor three measurements} = [51, \text{-20, 19, -11, -14, 1, -7, -22, 5, -12, -14, -28, 10, -10, 30 }] \quad (17)$$

It is observed that for the chromosome found that the DL returns zero and the 15 element of $\vec{S}_D$ are all –40, i.e., $\vec{S}_D$ shows no variation over the three time windows. Thus the GA has found a significant design flaw in the DL. It is readily observed that this example is very simple. It was included to illustrate the potential of the GA based procedure for automatically finding flaws in designs. Detection flaws that are more complex and harder to discover will be the subject of a future publication.

8. CONCLUSIONS

Given a system that may not be disassembled, subjected to inspection and for which the design specification are unavailable, determining the design of its underlying digital logic can be difficult. A genetic program has been used as a data mining function to reverse engineer digital logic. The database that was subjected to data mining consisted of known input to the digital logic, the associated measured output and a set of rules provided by experts relating to their assumptions about the digital logic.

The genetic program has been proven to be capable of reverse engineering complex digital logic designs. It is found that having a set of expert rules in the database is essential; the measured output of the digital logic is rarely sufficient to uniquely reverse engineer the design.

Frequently, part of the motivation for reverse engineering digital logic is to understand potential errors the design can introduce into measurement systems. To automate the discovery of design defects a genetic algorithm based procedure has been developed. This procedure uses the digital logic design discovered with the genetic program as part of the fitness function for the genetic algorithm. In addition to digital logic making up part of the fitness function, rules about the design's expected output during use are incorporated into the genetic algorithm's fitness. For the digital logic example the genetic program reverse engineered, the genetic algorithm is able to find a significant defect within 300 generations.

ACKNOWLEDGEMENTS

This work was sponsored by the Office of Naval Research. The authors would also like to acknowledge Dr. Jeffrey Heyer for useful conversations about and support for an unrelated application of genetic algorithms.

REFERENCES

1. J.P. Bigus, *Data Mining with Neural Nets*, Chapter 1, McGraw-Hill, New York, 1996.
2. J.R., Koza, F.H. Bennett III, D. Andre, and M.A. Keane,*Genetic Programming III: Darwinian Invention and Problem Solving*. Chapter 2, Morgan Kaufmann Publishers, San Francisco, 1999.
3. James F. Smith, III, "Fuzzy logic resource manager: real-time adaptation and self-organization", *Signal Processing, Sensor Fusion, and Target Recognition XIII*, I. Kadar, Vol. 5429, pp. 77-88, SPIE Proceedings, Orlando, 2004.
4. James F. Smith, III, "Fuzzy logic resource manager: decision tree topology, combined admissible regions and the self-

morphing property", *Signal Processing, Sensor Fusion, and Target Recognition XII*, I. Kadar, pp. 104-114, SPIE Proceedings, Orlando, 2003.
5. James F. Smith, III, "Fuzzy Logic Resource Manager: Evolving Fuzzy Decision Tree Structure that Adapts in Real-Time," *Proceedings of the International Society of Information Fusion 2003*, X. Wang, pp. 838-845, International Society of Information Fusion Press, Cairns, Australia, 2003,
6. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, 1989.
7. J.F. Smith, III and R. Rhyne, II, "A Resource Manager for Distributed Resources: Fuzzy Decision Trees and Genetic Optimization," *Proceeding of the International Conference on Artificial Intelligence, IC-AI'99*, H. Arabnia, Vol. II, pp. 669-675, CSREA Press, Las Vegas, 1999.
8. James F. Smith, III and Andrew Blank; "Co-evolutionary data mining for fuzzy rules: automatic fitness function creation, phase space, and experiments", *Data Mining and Knowledge Discovery: Theory, Tools, and Technology V*, B. Dasarathy, pp. 59-70, SPIE Proceedings, Orlando, 2003.
9. James F. Smith, III, "Co-evolutionary Data Mining to Deal with Adaptive Adversaries," *Proceeding of the International Conference on Machine Learning and Applications*, H. Arabnia, pp. 3-9, CSREA Press, Las Vegas, 2003.
10. James F. Smith, III; Robert D. Rhyne II and Kristin Fisher; "Data mining for multi-agent rules, strategies and fuzzy decision tree structure," *Data Mining and Knowledge Discovery: Theory, Tools, and Technology IV*, B. Dasarathy, pp. 386-397, SPIE Proceedings, Orlando, 2002.
11. James F. Smith, III; "Fuzzy Logic Resource Management in a Multi-agent Adversarial Environment"; *Proceeding of the International Conference on Information and Knowledge Engineering*, IKE 2002, H. Arabnia, pp. 234-240, CSREA Press, Las Vegas, 2002.
12. James F. Smith, III; "Genetic Program Based Data Mining to Discover Fuzzy Rules," *Proceeding of the International Conference on Artificial Intelligence*, H. Arabnia, pp. 502-508, CSREA Press, Las Vegas, 2002.
13. J.F. Smith, III and R.D. Rhyne, II, "Genetic Algorithm Based Optimization of a Fuzzy Logic Resource Manager: Data Mining and Co-evolution," *Proceeding of the International Conference on Artificial Intelligence, IC-AI'2000*, H. Arabnia, Vol. I, pp 421-428, CSREA Press, Las Vegas, 2000.
14. J.F. Smith, III and R. D. Rhyne II, "Fuzzy logic resource manager: tree structure and optimization", *Signal Processing, Sensor Fusion, and Target Recognition X*, I. Kadar, Vol. 4380, pp.312-323, SPIE Proceedings, Orlando, 2001.
15. James F. Smith, III, "Fuzzy logic resource manager: multi-agent techniques, fuzzy rules, strategies and fuzzy decision tree structure", *Signal Processing, Sensor Fusion, and Target Recognition XI*, I. Kadar, Vol. 4729, pp. 78-89, SPIE Proceedings, Orlando, 2002.
16. S, Luke and L. Panait, "Fighting Bloat With Nonparametric Parsimony Pressure", *Parallel Problem Solving from Nature - PPSN VII. 7th International Conference Proceedings*, J.J.M. Guervos, LNCS Vol.2439, pp. 411-421, Springer-Verlag, Berlin, 2002.
17. J. Gribbin, *Companion to the Cosmos*, p. 299, The Orion Publishing Group Ltd, London, 1996.

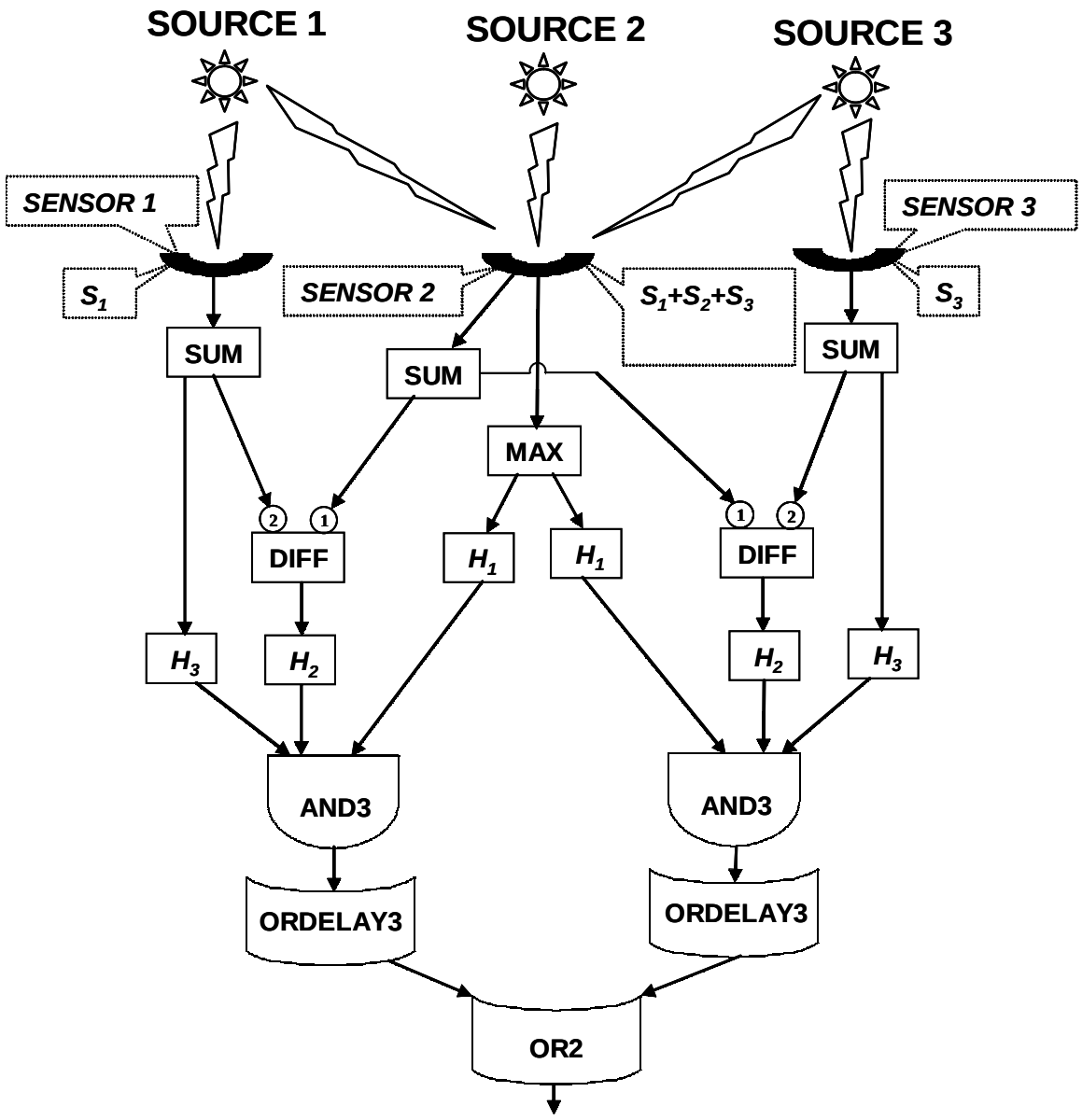

Figure 1: A system of three sensors designed to measure signals from source two while minimizing the corrupting influences of signals from sources one and three.