Multi-Valued Functional Decomposition as a Machine Learning Method

Craig M. Files Marek A. Perkowski
Department of Electrical Engineering
Portland State University
Portland, OR 97207-0751, USA
email: cfiles@ee.pdx.edu

Abstract

In the past few years, several authors have presented methods of using functional decomposition as applied to machine learning. These authors explore the ideas of functional decomposition, but left the concepts of machine learning to the papers that they reference. In general, they never fully explain why a logic synthesis method should be applied to machine learning. This paper explores and presents the basic concepts of machine learning, and how some concepts match nicely with multi-valued logic synthesis, while others pose great difficulties. The main reason for using multi-valued synthesis is that many problems are naturally multi-valued (i.e., values taken from a discrete set). Thus, mapping the problem directly to a multi-valued set of inputs and outputs is much more natural than encoding the problem into a binary form. The paper also shows that any multi-valued logic synthesis method could be applied to the machine learning problem. But, this paper focuses on multi-valued functional decomposition because of its generality of minimizing a given data set.

1. Introduction

This paper explores functional decomposition as a machine learning method. Functional decomposition of binary functions was first proposed by Ashenhurst in the 1950s [3] as a method of Boolean multi-level logic minimization. While this process has been known for many years, up until recently it could not be utilized because of the large complexity computation procedures that are required. Thus, very little was written about the practical aspects of decomposition from the 1950s until the late 1980s. In the cases that something was written, the papers were structured around decomposing small functions to binary circuits in various technologies and normally didn't present many extensions to the original Ashenhurst decomposition method. Lately, though, decomposition has become quite popular because of its application to logic synthesis for Field Programmable Gate Arrays (FPGAs) [13]. Also, note the early work in the field of multi-valued functional decomposition [18, 21].

The relationship between machine learning and logic synthesis is based on the similarities between realizing circuits with low complexity (smaller size) and the natural assumption of Occam's Razor [5]. In logic synthesis, the result of synthesis is a circuit that is designed with respect to a minimal number of gates, levels, inputs, literals, or product terms. In the case of machine learning, the goal is to effectively generalize concepts from the data [24]. This is done by using Occam's Razor, which is a principle in machine learning that states that if several theories explain all the facts then the simplest theory is the best. Thus, logic synthesis and machine learning are not all that independent, in that they are both attempting to reduce the description complexity of a given system specified by its partial behavior samples.

The central idea in machine learning is to learn or gain information from a given data set. This can be a very difficult task since practical databases are usually very large. To help handle this difficulty, the assumptions of Occam's Razor and introducing explicit domain knowledge are used to reduce the search space. Machine learning is not an exact methodology, but is an attempt of *learning* based on heuristics and probabilities. This attempt of *learning* is becoming more and more important as computers provide relatively inexpensive means to collect and store data. Thus, traditional methods, such as manual data analysis, are insufficient to fully evaluate a given data set. Instead, a new method called Knowledge Discovery in Databases (KDD) is being used to analyze data using analytical tools from statistics, pattern recognition, and artificial intelligence.

It is unfortunate that concepts that are already known in the logic synthesis community had to be re-learned by the machine learning community. In fact, several papers were written on rule generation systems that are based on circuit design techniques [2, 10]. The bridge between the two communities, called *pattern theory* [26], is an approach to finding algorithms for both the logic synthesis community and the machine learning community. The idea is to analyze the behavior of these algorithms using Computational Learning Theory (COLT). COLT ascertains how much information is needed for the algorithm to be within some probabilistic measure of correctness. The findings so far show that *pattern theory* algorithms, based on logic synthesis, are very promising in finding solutions for machine learning. However, the computational complexity of these algorithms still remains an outstanding problem. This is where heuristics in multi-valued functional decomposition or other multi-valued logic synthesis techniques are needed.

This paper is structured as follows: in Section two, a background of machine learning is given. Section three presents some of the difficulties in mapping machine learning problems into functional decomposition. Section four explains how decomposition can be used for finding solutions in machine learning. This is done by exploring previous Boolean decomposition results that were compared to machine learning approaches and other logic synthesis results on machine learning problems. While noting that decomposition of multi-valued decomposition should do much better than the Boolean decomposition techniques. In Section five, conclusions are raised about the use of functional decomposition in machine learning.

2. Machine Learning Background

Within the structure of this paper, the term *machine learning* is used to describe the process of applying expert knowledge of a particular problem domain to solve problems within the domain. This is also known as an *expert system* or a computer program that automates the process of *learning* or gaining information from a given data set. What makes up an expert system, is a process called *inductive learning* [17] which uses empirical evidence, in the form of examples, to derive rules for the given data. Systems that use *automated inductive inference* (in the machine learning paradigm) use rules to classify data. Rules are operations on variables, for instance, the average function, addition, subtraction, etc. Finding rules on variables is a process of discovering patterns and relationships that exist between variable combinations [26]. In logic synthesis terminology this means that *learning* is done by evaluating all the known outputs (or *cares*) of a given function to help in the determination of all the *don't cares* (*don't knows* in the machine learning paradigm) in the output of the given function. Thus, by treating *examples* as *cares* and considering functions with many *don't cares*, machine learning problem can be directly converted to a logic synthesis problem. While several papers have been written on this topic [8, 15, 14, 23, 22, 26, 27], one of the first papers to deal with this topic was [16].

Given that induction is a method of extrapolating samples of a function, the problem becomes very complex without making some reasonable assumptions. The assumption that is generally made is the one of a low complexity, as in Occam's Razor. This assumption has been applied and theoretically analyzed in several papers, including [4, 5]. Plus, Occam-based learning theory has been extended to actual implementations. One such implementation is Abductory Inference Mechanism (AIM[1]) [19] which creates networks of polynomials. The resulting network is the network with the least complexity based on the size of a given polynomial network and how well the network maps to the given data. Another implementation, C4.5 [25], is based on decision trees, where the resulting network is determined by comparing the sizes of decisions trees and selecting the least complex, or the smallest decision tree. The decision tree implementation used in C4.5 can be thought of as a network of multi-valued multiplexors, where the least complex solution is the smallest network of these gates. Note, that both of these implementations can handle data with noise, continuous variables, and conflicting data. In Section 3, aspects of *learning* and their corresponding types of data are explained in more detail, but note that even though functional decomposition was initially created for discrete Boolean values, it has been used in solving problems with all of these types of data [28, 11, 27, 6].

One drawback for both C4.5 and AIM is their use of a narrow complexity measure to determine the effectiveness of a solution. For instance, neither program finds the parity function to have low complexity, even though parity is highly patterned and can easily be computed [26]. One solution for determining complexity was found by [26] to convert the problem to pattern theory, where the complexity is determined as the problem of finding a pattern. Given that the complexity is based on combinational methods, there are many possible measures of complexity that could be used. Each of these complexities is based on the number of basis functions (gates) necessary to represent that function. The complexity measure selected is a measure proposed by [1]. This method, called Decomposed Function Cardinality (DFC) is the sum of the cardinalities of the component functions in a combinational representation, when that sum has been minimized. Any function can be a component, but its cost goes up with its *cardinality*. The *cardinality* of an n-input, m-output binary function is $2^n * m$. In the Boolean world, an AND-gate will have the same *cardinality* as an XOR-gate.

So why use functional decomposition? Decomposition has the advantage that it is not based on a set of operators or gates (in contrast to all other logic synthesis methods). This is especially notable in the case of multi-valued logic,

[1] Trademark of AbTech Corporation

where the number of operators can increase as the value of the logic grows. Thus, decomposition is not constrained by a technology or a pre-selected single theory. This has wide applications in both logic synthesis and machine learning. After decomposition the sub-blocks are small, so highly sophisticated and/or specialized logic synthesis/rule derivation methods can be used [20]. Since [18], Epstein, Liu, Miller, Muzio [7] and others have found that functional decomposition for a function class can be used to discover optimal operations for the decomposition theory of functions in the class.

3. Machine Learning to Logic Synthesis

This section gives an overview of mapping a machine learning problem to a multi-valued functional decomposition problem. While there are many aspects that are the same between the two methodologies, this section will also point out the differences between the two. Also presented are some problems that even the machine learning community has not fully implemented.

3.1. Machine Learning Examples

In general, machine learning concepts or examples are used to find a generalization of the original data set. An example or concept is thought of as a *Care Term* by the logic synthesis but in machine learning it has a different meaning. For example, in the case of a cancer database, doctors may take data about a person's age, sex, height, weight, x-rays, etc., and if the person has cancer or not. Now, the number of input data values (variables) that are taken can be enormous while the number of patients that are evaluated in the database can be quite small. For instance, if a doctor asks 20 questions about sex, height, etc., but evaluates only 500 patients, then the number of examples is quite small compared to the number of possible outcomes given the number of questions. In this example, if each question was a yes or no question, then there could possibly be 2^{20} $\approx$1-million combinations given different values of the inputs, while only 500 patients were evaluated. Thus, this is only a small representation of the possible values that exist, which explains the name *example*. Each *example* is only an small piece or example of the actual function.

Mapping from machine learning to logic synthesis can then be done by mapping the examples to care terms. But, the number of *don't cares* in the functions that will be evaluated with logic synthesis is quite large. In fact, it is thought that most machine learning problems have more than 99.9% *don't cares*. Thus, any logic synthesis method that could possibly be used must be able to handle a large number of *don't cares*.

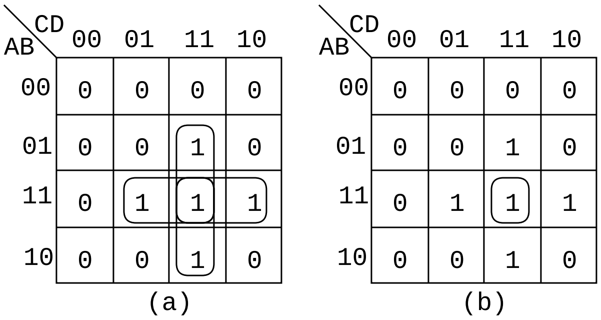

Figure 1. *Don't knows* **in the input variables**

3.2. *Don't Cares* in the Inputs Variables

The possibility exists that data collected could have a *don't care* in the input. This is different than a *don't care* in the input for a logic synthesis problem where the *don't care* is treated as covering all possible values for that variable. In the machine learning world, this means that you have a *don't know*. The *don't know* cannot be handled the same as in logic synthesis since it only represents one data value not all possible values. For instance, say one of the questions asked to patients was their age. But, the person did not want to state their age, and instead of guessing at an approximate age, a *don't know* was placed in the category. In terms of logic synthesis, this would mean that the person was all the ages between 0 and 150. While in machine learning this just means that person has only one age between 0 and 150.

Another example of this is shown if Figure 1, using 2-valued Karnaugh maps with *don't cares*. Both Karnaugh maps are given the same set of input cubes (a dash indicates a *don't know*, or a *don't care*): $-BCD$, $A-CD$, $AB-D$, and $ABC-$. In Figure 1a, the logic synthesis reduction method is shown where all the minterms are placed in essential prime implicants. The result is a minimum circuit with 4 product-terms and 1 sum-term. In Figure 1b, the machine learning result is a little bit different. For example, the cube: $-BCD$, is thought to have a *don't know* in it that specifies that either the minterm $ABCD$ exists or the minterm $\overline{A}BCD$ exists. The minimum solution to this problem is $ABCD$.

This problem of input *don't knows* can also be applied using probabilities. Figures 2a through 2d show the possible combinations when given the cubes: $A-$ and $-B$. Since there are four combinations, then the probability of finding any of the Karnaugh maps in Figures 2a-2d, is $\frac{1}{4}$. Thus, Figure 2e shows the probabilities of finding any given minterm actually being a one. By assuming any probability value in the interval[0,1], it can be calculated for every minterm to be true or false. For example, if the probability value is

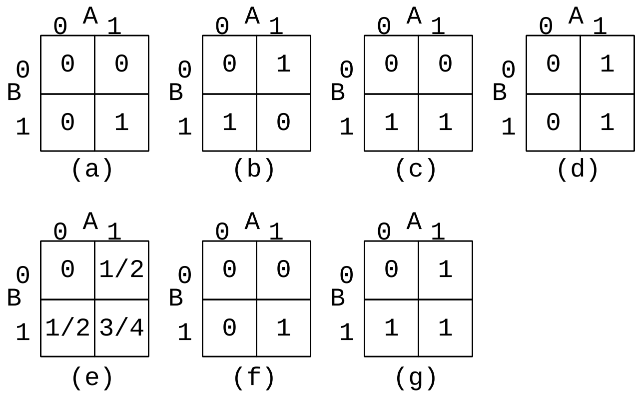

Figure 2. Example of using probabilities to calculate a function from cubes $A-$ and $-B$

greater than 0.5 then only the minterm AB is true, resulting in the Figure 2f. In the case of a probability value greater than 0.4, the function becomes as shown in Figure 2g.

One can see this topic is of great importance to the machine learning community. And, to the knowledge of the authors, there are no published papers on the subject that fully explore this topic. The paper [22] uses relations as an attempt at this problem. But the relations are only used to identify overlapping cubes. For example, given the cubes: $A-C$ and $AB-$, the relation recognizes that the two cubes overlap at ABC. Thus, if $A-C$ has the output value 5, and $AB-$ has the output value 3 then the cube ABC has the output value of 5 or 3.

3.3. Retaining MVL synthesis techniques

Since Boolean logic synthesis techniques have been in practice since the 1950s, they are well known, and fairly well understood, compared to multi-valued logic synthesis. That does not mean that they are the future of computing. It also does not mean that machine learning problems should be encoded to Boolean expressions before analysis. The encoding to Boolean expressions may destroy the data either by adding or removing components inherent in the multi-valued-ness of the function or data.

One important aspect of machine learning and expert systems that is not highly publicized is that an expert system will usually perform best by the way the data is given to it by a user. This implies that if the user has some understanding of the data being inputed into the expert system, the user may control the way the expert system handles the data. When the data is encoded to Boolean expressions, the user may loose all intuition of what the data means and how variables can interact with each other.

This is why it is important to leave machine learning problems in their original form, and why it is important to use multi-valued logic synthesis techniques, instead of Boolean techniques.

3.4. Noise in the Data

Another concept that comes up is the idea of noise. Noise in machine learning could be caused by many different things. An example of noise could be when the doctors wrote down the data, they could have written the data down in the wrong place, written down the wrong information, etc. This can cause problems in both the machine learning community and the logic synthesis community, in that, how do you detect that the example is actually noisy, and not the correct value. While [28] talks about this concept of stripping out noise, this is a theoretical point of view and may not be the best way of handling noisy data.

There are two types of noise: noise that is in the input, and noise that is generated on the output. Both types of noise cause different problems in determining that noise is actually in the data set. The removal of noise can be done when a certain threshold is hit (parameter set by the user). For instance given two sets with 1000 elements in each, if both sets are equal except for one pairing, then should this pairing be thought of as noise? A threshold could then be used to state that if two sets are, for example, 90% equal, then the two sets are equal. The use of a threshold should be variable since there may not be any noise on some data sets, while other data sets may have a large amount of error. Of course, setting a low threshold may allow sets to be combined that should not be combined.

Another problem that exists is what happens when the algorithm finds noisy data and tries to combine the data. What should the noisy values become; one of the values, an average of the two values, a *don't know*, etc. Handling noise is notably a very deep theoretical issue in machine learning, and needs to be looked into further.

3.5. Continuous Data

Machine learning problems also contain continuous data or data that is in the form of real numbers, not natural numbers. This type of data, so far, has shown itself to be very difficult for multi-valued functional decomposition. There have been several attempts at this, though. In [11, 28], the authors map real-valued variables onto natural numbers by discretization. In [6, 27], the data is actually left in the form of real values which causes problems in the decomposition process when trying to combine two values. A way around this is to combine values that are some δ distance apart from another. For example, 2.46 and 2.55 could be combined if δ=0.1. Now the problem is determining the value of the two combined elements: possibly average, one of the two values, a *don't know*, etc. This seems to be a topic that needs further research. But, the initial approach of the authors would be to use another machine learning tool (AIM, C4.5, neural networks) to take care of data in this form as a preprocessing for decomposition, because these tools are based on experience gained in years of experimentation.

This section has shown how data from the machine learning community can be mapped into multi-valued logic synthesis. While some of the data poses certain difficulties for the logic minimization techniques, they are also difficult for the applications in machine learning. Thus, there are a lot of areas that need to be explored on the boundary between multi-valued logic synthesis and machine learning.

4. Comparison

Given that there are already Occam-based software programs available in the machine learning community, why would the multi-valued logic synthesis community want to get involved? This section may not fully answer this, but it does show a comparison between the two methods. Among these existing approaches, the logic synthesis approach does much better than the machine learning approach. In our opinion, this is just a beginning, considering that multi-valued logic synthesis should perform much better than their Boolean counterparts. That is, machine learning problems often have multi-valued outputs and inputs. This is the reason for choosing multi-valued functional decomposition; it does not have to put constraints on the operations available in multi-valued systems or can give appropriate operations to achieve optimality in a decomposition theory.

The first example of how well Boolean logic synthesis compares to machine learning concepts is found in [9]. This paper compares C4.5 (the top program in the machine learning community), functional decomposition (using DFC as the measure of complexity) and Espresso, a logic minimization tool distributed by the University of California at Berkeley [12] (an industry standard tool of accepted quality). The data found in [9] is shown in Table 1. Each of the 2-valued functions in Table 1, using operations AND, OR, EXOR, and COMPLEMENT, are assumed to have eight input variables. Thus, for each of the functions tested, there are several variables that are *vacuous*, i.e., variables that have no effect on the output of function. If these functions were completely specified, determining if the variables are *vacuous* is a trivial issue. But, when the functions are highly unspecified it becomes very difficult to determine which variables are *vacuous* and which variables actually have an effect on the output of the function. In Table 1, each method was given a random set of data to train on, ranging from 25 to 250 out of the 256 (2^8) possible cases. Once the method was trained, the entire 256 cases were tested and the number of differences were recorded as errors. This is shown in

Original Function	Known DFC	Average Error			Number of samples		
		C4.5	Decomp.	Espresso	C4.5	Decomp.	Espresso
kdd1	2	0	0	0	8	7	9
kdd2	8	0.32	0	0.96	31	25	40
kdd3	8	6.35	0	5.64	83	25	51
kdd4	12	2.48	3.72	2.64	74	67	76
kdd5	12	1.28	2.72	3.52	61	76	54
kdd6	16	2.76	2.4	12.86	97	126	113
kdd7	20	17.52	8.18	17.16	200	60	181
kdd8	20	13.79	6.55	16.54	224	104	205
kdd9	28	20.69	10.53	5.69	256	126	51
kdd10	36	10.52	11.11	8.44	249	251	229
Average		7.57	4.52	7.35	128.3	86.7	100.9

Table 1. Comparing a machine learning approach with logic synthesis approaches [9]

the section Average Error, where the values can range from 0 to 100, with 0 error being the best. The data found under the heading Number of samples, shows the number of random samples necessary before the learning method obtains a concepts, such that, the error in the result is 0.0. By looking at the averages at the bottom of the tables, notice that decomposition outperforms C4.5 and Espresso in both Average Error and Number of Samples.

In [23], the concept of machine learning was mapped to a Boolean logic synthesis tool that minimizes EXOR of products expressions. This paper showed that this type of synthesis performs as well as C4.5 on functions with a small and medium percentage of *don't cares* but did not do very well with functions with a large number of *don't cares*. Their conclusion is that EXOR synthesis modified for strongly unspecified functions is a viable method that could complement existing machine learning techniques.

Another result of decomposition being applied to machine learning is shown in [28]. This thesis shows many results of their multi-valued decomposition tool, HINT, compared to C4.5 and other machine learning techniques, such as neural networks. In most cases, HINT learned significantly better than C4.5. The interesting achievement here is that HINT was compared with other machine learning programs on data with continuous data, and data that contains noise. The continuous data were discretized in the method introduced by Kononenko et al. [11].

The results from previous researchers show that decomposition and logic synthesis methods can be used in the area of machine learning. In fact, this section showed results that logic synthesis tools outperformed the famous machine learning program C4.5.

5. Conclusion

This paper showed the basis for using multi-valued logic synthesis in the field of machine learning. While the problems inherent in machine learning are not exactly the same as the problems found in multi-valued logic synthesis there is still a correlation between the two areas. The important aspect of this is that bridging the gap between the two areas could prove beneficial to both communities. There are many theories to explore, especially in the context of manipulating machine learning data.

The reason that multi-valued functional decomposition was the main focus of this paper is because of its generalized behavior. That is, functional decomposition based on heuristics can be used to find a set of rules or used to reduce complexity issues within some data set. Again, because of the complexities inherent in functional decomposition, it is possible that other multi-valued logic synthesis techniques may demonstrate better efficiency. Since there has been little done in the comparison of multi-valued logic synthesis tools, multi-valued decomposition, and machine learning approaches, there is a lot of space for new research.

While there are many issues to work on in this new field of functional decomposition, the results are good enough to compete with the existing "top" machine learning programs. In fact, the data obtained during the research for this paper shows that logic synthesis techniques obtained better results than C4.5 which is a standard machine learning tool.

As the size of databases grows, it is obvious that the need for expert systems and machine learning is going to be necessary to process all the data. This is a tough problem, but by using multi-valued functional decomposition hopefully the problem will be reduced. It is difficult to predict the future, but the possibility exists that machine learning based tools may be used in circuit design as well.

References

[1] Y. Abu-Mostafa. *Complexity in Information Theory*. Springer-Verlag, New York, 1988.

[2] C. Apte, S. J. Hong, S. Prasad, and B. Rosen. Ramp: Rules abstraction for modeling prediction. *Technical report*, 1995.

[3] R. L. Ashenhurst. The decomposition of switching functions. *International Symposium on Theory Switching Functions*, pages 74–116, 1959.

[4] A. R. Barron and R. L. Barron. Statistical learning networks: A unifying view. *Symposium on the Interface: Statistics and Computing Science*, page 12, 1988.

[5] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, pages 377–80, 1987.

[6] J. Demsar, B. Zupan, M. Bohanec, and I. Bratko. Constructive induction by decomposition of real functions. In G. W. M. Someren, editor, *Proceedings of ECML-97*, pages 94–107, 1997.

[7] G. Epstein. *Multiple-valued Logic Design: An Introduction*. Institute of Physics Publishing, Bristol, 1993.

[8] C. Files, R. Drechsler, and M. Perkowski. Functional decomposition of MVL functions using multi-valued decision diagram. *International Symposium on Multi-Valued Logic*, pages 27–32, May 1997.

[9] J. A. Goldman and M. L. Axtell. On using logic synthesis for supervised classification learning. *Tools with AI conference*, 1995.

[10] S. J. Hong. A heuristic algorithm for generating minimal rules from examples. *Pacific Rim International Conference on Systematic Methods of Scientific Discovery*, 1994.

[11] I. Kononenko, E. Simec, and M. R. Sikonja. Experiments in automatic learning of medical diagnostic rules. Technical report, Jozef Stefan Institute, Ljubljana, 1996.

[12] E. R. Laboratory. *OCTOOLS-5.2 Users guide*. University of California at Berkeley, May 1993.

[13] Y. T. Lai, M. Pedram, and S. B. K. Vrudhula. BDD based decomposition of logic functions with application to FPGA synthesis. *Design Automation Conference*, pages 642–647, 1993.

[14] T. Luba. Decomposition of multiple-valued functions. *International Symposium on Multi-Valued Logic*, pages 256–261, 1995.

[15] T. Luba and R. Lasocki. Decomposition of multiple-valued boolean functions. *Applied Math and Computer Science*, 4(1):125–138, 1994.

[16] T. R. Martinez and D. M. Cambell. A self organizing binary decision tree for incrementally defined rule based systems. *IEEE Trans. on Systems, Man and Cybernetics*, 21, 1991.

[17] R. Michalski. A theory and methodology for inductive learning. In R. Michalski, J. Cabonnel, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134, 1983.

[18] D. M. Miller. *Decomposition in many valued design*. PhD thesis, University of Manatoba, May 1976.

[19] G. J. Montgomery and K. C. Drake. Abductive networks. *SPIE, Applications of Neural Networks Conference*, 1990.

[20] S. Muggleton and L. D. Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 16(20):629–679, 1989.

[21] J. C. Muzio and T. C. Wesselkamper. *Multiple-valued switching theory*. Adam Kilger, Boston, MA, 1986.

[22] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J. S. Zhang. Decomposition of multiple-valued relations. *International Symposium on Multi-Valued Logic*, May 1997.

[23] M. Perkowski, T. Ross, D. Gadd, J. Goldman, and N. Song. Application of esop minimization in machine learning and knowledge discovery. *International Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, 1995.

[24] G. Piatetsky-Shapiro, C. Matheus, P. Smyth, and R. Uthurusamy. Kdd-93:progress and challenges in knowledge discovery in databases. *Artificial Intelligence Magazine*, 15(3):77–82, 1994.

[25] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, Palo Alto, California, 1993.

[26] T. Ross, M. Axtell, M. Noviskey, and D. Gadd. Pattern theory paradigm for system design. *Midwest Symposium on Circuits and Systems*, 1993.

[27] T. Ross, J. Goldman, D. Gadd, M. Noviskey, and M. Axtell. On the decomposition of real-valued functions. *International Symposium on Multi-Valued Logic*, 1994.

[28] B. Zupan. *Machine Learning Based on Functional Decomposition*. PhD thesis, University of Ljubljana, Slovenia, 1997.