

Pattern Theoretic Knowledge Discovery

Jeffrey A. Goldman
Wright-Laboratory
WL/AART-2 Bldg 22
2690 C Street STE 1
Wright-Patterson AFB, Ohio 45433-7408 *

Abstract

Future research directions in Knowledge Discovery in Databases (KDD) include the ability to extract an overlying concept relating useful data. Current limitations involve the search complexity to find that concept and what it means to be "useful." The Pattern Theory research crosses over in a natural way to the aforementioned domain. The goal of this paper is threefold. First, we present a new approach to the problem of learning by Discovery and robust pattern finding. Second, we explore the current limitations of a Pattern Theoretic approach as applied to the general KDD problem. Third, we exhibit its performance with experimental results on binary functions, and we compare those results with C4.5. This new approach to learning demonstrates a powerful method for finding patterns in a robust manner.

1 The Pattern Theory Approach

Pattern Theory is a discipline that arose out of machine learning [3] [7] and switching theory [6]. The original goal was to develop formal methods of algorithm design from specifications. The approach is based on a technique called function decomposition and a measure called decomposed function cardinality (DFC). Since Pattern Theory is able to extrapolate on available information based on the inherent structure in the data, it is directly related to KDD.

Decomposing a function involves breaking it up into smaller subfunctions. These smaller functions are further broken down until all subfunctions will no longer decompose. For a given function, the number of ways to choose two sets of variables (the partition space) is exponential. The decomposition space is even larger, since there are several ways the subfunctions can be

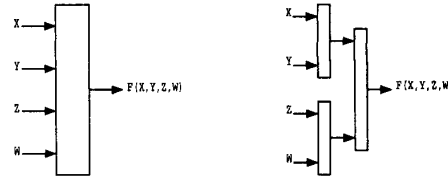


Figure 1: Lookup Table Figure 2: Decomposition

combined and there are several levels of subfunctions possible. The complexity measure that we use to determine the relative predictive power of different function decompositions is called DFC.

DFC is calculated by adding the cardinalities of each of the subfunctions in the decomposition. The cardinality of an n -variable binary function is 2^n . We illustrate the measure in the above figures. In Figure 1, we have a function on four variables with cardinality $2^4 = 16$. In Figure 2, we show the same function after it has been decomposed. The DFC of this representation for the original function is $2^2 + 2^2 + 2^2 = 12$. The DFC measures the relative complexity of a function. When we search through the possible decompositions for a function, we choose one with the smallest DFC. This decomposition is our learned concept.

The decomposed representation of the function is one that exhibits more information than the alternative. For example, Figure 1 is essentially a lookup table of inputs and outputs. Figure 2, on the other hand, is a function that is not simply a table. The decomposition, for example, could be two simple functions combined together.

Throughout the paper when we refer to a minimal function decomposition, we use "minimal" to mean a decomposition such that the DFC is the smallest possible for the entire set of decompositions. It is noted that a given minimal decomposition is not unique. For a more rigorous explanation of the inner workings of function decomposition or function extrapolation, the

*(513) 255-3215; Email: goldmanj@aa.wpafb.af.mil

reader is referred to [2] and [6].

An important point is that a function with a low DFC has been experimentally and theoretically determined to be learnable with a small number of samples [6]. Also, functions we are interested in learning, (i.e., functions that are highly “patterned,”) have a low DFC. Moreover, “useful” concepts in databases correspond to functions that have a low DFC. The Function Learning And Synthesis Hot-Bed (FLASH) was developed to explore function decomposition, and pattern finding. This paper will show that the FLASH program exhibits promising results for finding patterns robustly.

2 Pattern Theory and KDD

The decomposability of a concept gives a relative measure of “usefulness.” If we achieve a minimum decomposition, we have extracted a concept with the given information. If this minimum decomposition is small in function cardinality, we have a strong pattern and thus a “useful” concept. It is important to note that we need not have a minimum decomposition in order to exhibit a strong pattern. Moreover, with a small, nonoptimal decomposition, we can still extrapolate with a small number of errors.

Pattern Theory offers the ability to look for patterns for a given set of data in a robust way. “Robust” is used to mean that Pattern Theory has no inherent bias to a particular class of patterns except those of low complexity (Occam’s Razor). This makes Pattern Theory directly applicable to Knowledge Discovery. In general, the approach is to look for some decomposition in a number of binary variables. We can think of a database as a set of m records with n binary fields each. It is also possible to choose a field as the output and attempt to find a description for it in terms of the other fields as in data mining. Our approach will then attempt to find any pattern, if one exists, via function decomposition.

The only domain knowledge used in this approach is implicit in the representation of the inputs. In our discussion of relating this approach to KDD, we use a natural representation as mentioned above. Aside from this implicit representation, every attempt was made to omit any domain knowledge when developing the theory.

Since the Pattern Theory approach does not bias toward a specific function representation, the learned function can be unintuitive to the user. There is room, however, to restrict FLASH to a specific set of functions if the domain calls for it. For example, if there

is a database where the only important relationships include AND, OR, and NOT, then the concepts we find will only contain those operators while still constrained to find the minimum DFC.

2.1 FLASH’s Current Limitations

FLASH has no solution to handle missing values or contradictory information (noise) in the data for all fields. For noise, we can preprocess the data. However, preliminary results suggest that the correct approach is to have some internal method of handling noise.

FLASH is equipped to test decomposition of functions with no more than 24 variables. In general, finding the *minimal* function decomposition is an exponential problem because we must search the entire decomposition space. In order to perform in real time for database query, (i.e., a few minutes) we are limited to about eight variables. In practice, an eight field database is quite small and as we add more variables, the time growth is exponential.

Finally, it is possible that our minimum decomposition is less “useful” than a small decomposition. This happens when our minimum decomposition takes advantage of simple functions, whereas a small nonoptimal decomposition may give us a more complex function that we have a name for. For example, the function $F(X,Y,Z) = ((\bar{X} \text{ xor } Y) \text{ or } Z)$ is not a function we have a common name for, however, $G(X,Y,Z) = XYZ$ is an “and” function which is more intuitive to the user, even though both F and G have the same DFC. If a larger function decomposed into more smaller subfunctions like F instead of less subfunctions like G , we can have a minimal decomposition be less useful than a decomposition which is not minimal.

2.2 FLASH’s Strengths

Irrelevant [1] fields (vacuous variables) are found easily in the decomposition process [6]. If the field is not present in the learned decomposition, then it has nothing to do with the pattern. For example, suppose a field is “pregnancy” but we are looking for patterns among men only. Although it is true that the concept among men includes the tautology that they are not pregnant, it is irrelevant. The method also does not omit relevant fields even if they may appear on the surface to be irrelevant. For example, looking for knowledge about a particular disease in a database of patients, it would appear that a zip code field should be irrelevant. However, if there is a pattern based on symptoms as well as geographic region, FLASH is equipped to find the entire relationship.

In our tests, we are simplifying the problem by only allowing binary valued fields. However, since this approach has been proven to be valid for k valued fields, we will continue with this model without loss of generality [6]. In fact, our method has potential to generalize to continuous variables as well [5].

3 Experimental Results

The objective of this section is to present some functions indicative of concepts in a database and to show how the Pattern Theory approach is relevant to this domain. Specifically, we wish to demonstrate a correlation between DFC and the complexity of the pattern. Also, we compare learning ability in number of errors and in the concepts formed with the standard decision tree approach of C4.5 [4].

3.1 Experimental Setup

All of the learning curves were generated with the same decomposition strategy. The strategy was to search through all possible partitions from the top level, down to the evaluation of the cardinality of each function's subfunction, and in turn to its subfunction. This means for a given breakdown of eight variables into two functions, each was further broken down one additional time. The lowest DFC representations were chosen and recombined. The search was not completely exhaustive. Although the choice of a decomposition plan does affect the learning ability of a given function, the functions in this paper were not chosen to make the plan perform better. The choice of plans is equivalent to a choice of heuristics to search the space of decompositions. Our tests use essentially a 2-ply look-ahead search with the calculated DFC as the node evaluation. The important point is to show that decomposition is useful for learning.

The options chosen for C4.5 were to assume no noise in the data and choose the best of 10 trees (unpruned, -m 0, -t 10). These options were not simply chosen at random. Instead, after a careful testing of more than 15 different settings, the options that yielded C4.5's best performance on our benchmark functions was used for comparison. It is noted that we can always increase the number of trees built in C4.5 to increase performance. However, when we tested 100 trees, the performance increase was not statistically significant. Furthermore, C4.5 with 10 trees and FLASH had roughly the same CPU times.

Table 1 is a list of all of the functions tested. Table 2 shows the functions with their corresponding min-

Original Function ¹	
F_1	$= \bar{x}_4$
F_2	$= (x_1 x_3) + \bar{x}_2$
F_3	$= (x_1 \bar{x}_2) \text{ XOR } (x_1 x_5)$
F_4	$= x_2 + x_4 + x_5 + x_8$
F_5	$= (\bar{x}_1 + x_2) + (\bar{x}_1 x_4 x_6)$
F_6	$= (x_1 \bar{x}_2 x_3)(x_4 + \bar{x}_6)$
F_7	$= (x_1 \Rightarrow x_4) \text{ XOR } (x_7 x_8 (x_2 + x_3))$
F_8	$= (x_2 \text{ XOR } x_4)(\bar{x}_1 \text{ XOR } (x_5 x_7 x_8))$
F_9	$= (x_1 x_2) + (x_3 x_4) + (x_5 x_6) + (x_7 x_8)$
F_{10}	$= (x_1 x_2 \bar{x}_4) + (x_3 x_5 x_7 x_8) + (x_1 x_2 x_5 x_6 x_8) + (\bar{x}_3 \bar{x}_5)$

Table 1: The Functions Tested

Original Function	Actual DFC	Average Error		# Samples	
		C4.5	Flash	C4.5	Flash
F_1	2	0	0	8	7
F_2	8	0.32	0	31	25
F_3	8	6.35	0	83 ²	25
F_4	12	2.48	3.72	74	67
F_5	12	1.28	2.72	61	76
F_6	16	2.76	2.4	97	126
F_7	20	17.52	8.18	200	60
F_8	20	13.79	6.55	224	104
F_9	28	20.69	10.53	256	126
F_{10}	36	10.52	11.11	249	251
Average		7.57	4.52	128.3 ³	86.7

Table 2: DFC, Mean Error, and # Samples to Learn

imal DFC. The functions were chosen as an attempt to represent concepts in a database. It is noted to the reader that many other kinds of "patterned" functions have been tested with FLASH besides Boolean Expressions. Some of them include symmetric functions, numerical functions, image functions, and string functions. For results with those types of functions, the reader is referred to Ross et al. [6].

The tests on the individual functions were as follows. First, each method was given a random set of data to train on ranging from 25 to 250 out of a total of 256 possible cases. Once the method was trained, the entire 256 cases were tested and the number of differences were recorded as errors. This procedure was repeated 10 times for a given sample training size in intervals of 25. Thus, the total number of runs for each function was 100 of varying sample size. None of the learning was incremental. All of the runs were independent.

For brevity, the comparison graphs were omitted in lieu of a condensed summary of each of the learning curves. For each function, the average number of er-

¹ Functions were relabeled from the original work for clarity

² C4.5 with pruning learned this function with 46 samples

³ The average with the better score for F_3 is 124.6

rors for the entire run was recorded in the table. It was then possible to compare FLASH and C4.5 with these averages for a given function (middle two columns). The far right two columns of Table 2 show the number of samples necessary before the learning method obtains a concept such that in all ten separate runs, the number of errors was 0. The value at the bottom of the table is the average over all of the functions. The smaller the number here, the better the performance.

3.2 Analysis

Looking at the relationship between a function expression and its DFC, the simpler functions, or more intuitive patterned functions, have a lower DFC. For example, function 2 is less complicated than function 10 and it is a more intuitive relationship between the variables. Thus, its DFC is appreciably lower (note that the DFC range for functions on eight variables is from 0 to 256). Comparing these 10 functions to any eight variable function in general, our examples are relatively simple.

FLASH does a good job at learning all 10 functions. Also, there is a correlation between the number of samples needed to learn a function and its DFC ($\rho = 0.9$). In general, the higher the DFC is, the more samples required to learn the function. The functions with lower DFC on the average require fewer samples to be learned. With the exception of function 10, FLASH learns all of the functions quickly.

Now that we have shown FLASH's ability to learn and the connection between low DFC, patterned functions, and learnable functions, we wish to examine what concepts FLASH learned. In all cases but one, FLASH found an equivalent function with considerably less information than the entire specification of that function. Since FLASH is not restricted to any representation, it can find unintuitive functions with the possibility of discovery. For example, function 3 was listed in the table as $(X_1 \text{ and } \bar{X}_2) \text{ xor } (X_1 \text{ and } X_5)$. FLASH found the function: X_1 and $(X_2 \text{ xor } X_5)$. This new function is equivalent to the original representation but it is more simplified and in a sense, a discovery. C4.5, on the other hand, yielded the more complex function expression $X_1\bar{X}_2\bar{X}_5$ or $X_1X_2X_5$. FLASH found functions 1, 4, and 9 exactly as they were represented in Table 1. C4.5 found function 1 exactly as well. However, it found function 4 as: X_2 or \bar{X}_2X_4 or $\bar{X}_2\bar{X}_4X_8$ or $\bar{X}_2\bar{X}_4X_6\bar{X}_8$. C4.5's concept for function 9 is horrendously complicated! Some of the other functions FLASH found were less informative. However, those unusual representations exactly matched the outputs of the original function.

4 Summary

Although the problems of conflicting data in input or output variables and missing values in input variables still needs development, we have a strong basis for finding patterns in a robust way. Pattern Theory is particularly well suited for problems that arise dealing with irrelevant fields. The approach simply looks for the pattern within the variables and will not report "obvious" knowledge. We have shown a correlation between low DFC and highly "patterned" functions. We have also shown that only a small number of samples were needed to learn a highly patterned function. Moreover, FLASH can find unintuitive functions with the possibility of discovery.

With the theoretical results showing the generalization of function decomposition to n -state variables, the relationship between finding patterns in Pattern Theory and Knowledge Discovery in Databases is clear. The problems and goals are similar. One of the main thrusts for Pattern Theory is to be able to handle hundreds of variables. Analogously, databases are daunting in size and to extract the relevant information, we need to be able to handle large numbers of fields. In conclusion, we have shown that learning by function decomposition is a powerful method for finding patterns.

References

- [1] Hussein Almuallim and Thomas G. Dietterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 547-552, Anaheim, CA, 1990. AAAI Press.
- [2] Robert L. Ashenurst. The decomposition of switching functions. In *Proceedings of the International Symposium on the Theory of Switching*, April 1957.
- [3] David Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36(2):177-221, 1988.
- [4] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, Palo Alto, California, 1993.
- [5] Timothy D. Ross, Jeffrey A. Goldman, David A. Gadd, Michael J. Noviskey, and Mark L. Axtell. On the decomposition of real-valued functions. In *Third International Workshop on Post-Binary ULSI Systems in affiliation with the Twenty-Fourth International Symposium on Multiple-Valued Logic*, 1994.
- [6] Timothy D. Ross, Michael J. Noviskey, Timothy N. Taylor, and David A. Gadd. Pattern theory: An engineering paradigm for algorithm design. Final Technical Report WL-TR-91-1060, Wright Laboratory, USAF, WL/AART, WPAFB, OH 45433-6543, August 1991.
- [7] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134-1142, November 1984.