CONSTRUCTIVE INDUCTION FROM DATA IN AQ17-DCI: Further Experiments

Eric Bloedorn and Ryszard S. Michalski

Artificial Intelligence Center

George Mason University

Fairfax, VA 22030

MLI 91-12

December 1991

CONSTRUCTIVE INDUCTION FROM DATA IN AQ17-DCI: Further Experiments

Abstract

This paper presents a method for data-driven constructive induction, which generates new problem-oriented attributes by combining the original attributes according to a variety of heuristic rules. The combination of attributes are defined by different logical and/or mathematical operators, thus producing a potentially very large space of features. This space is reduced by applying an "attribute quality" evaluation function which selects the "best" set of features. The data, enhanced with the new attributes, is used to generate rules which are then evaluated by a "rule quality" function. Attribute construction and rule generation is repeated until a termination condition is satisfied. Attributes produced by the method often represent meaningful and useful concepts. The program, AQ17-DCI, implementing the method has been experimentally applied to a number of problems and produces very satisfactory results. These results are comparable to the best existing machine learning methods.

key words: concept learning, empirical methods, empirical evaluation

Acknowledgements

The authors wish to thank Michael Hieb, Ken Kaufman and Janusz Wnek for their ideas and critical review of this paper. We also wish to thank the UCI repository of machine learning databases for the 1984 congressional voting data.

This research was conducted in the Center for Artificial Intelligence at George Mason University. Research of the Center for Artificial Intelligence is supported in part by the Defense Advanced Research Projects Agency under grant, administered by the Office of Naval Research, No. N00014-87-K-0874, and No. N00014-91-J-1854, in part by the Office of Naval Research under grant No. N00014-88-K-0397, No. N00014-88-K-0226, and No. N00014-91-J-1351, and in part by the National Science Foundation under grant No. IRI-9020266.

1. INTRODUCTION

Most inductive learning programs perform a "selective" induction, that is, they generate descriptions (rules, decision trees, etc.) that involve only attributes selected among those provided in the examples. Thus, if the attributes used in the examples are weakly relevant, the learned descriptions may also be weak. It is possible, however, that although the original attributes may be of poor quality, there exist certain combinations or functions of these attributes that are highly relevant to the problem. This paper is concerned with the problem of discovering such relevant combinations of the original attributes. A method of data-driven constructive induction (DCI) is presented here, for the construction of new attributes through the application of various mathematical and logical operators to the initial attributes. The currently available operators include multiplication, integer division, addition, subtraction, relational operators, average, most-common, least-common, maximum, minimum, and number of attributes having some value. The method has been implemented in AQ17-DCI.

There are a number of other programs which perform constructive induction. One of the first programs to exhibit some constructive induction abilities was INDUCE (Michalksi, 1980) which generated new attributes or predicates by applying "constructive generalization rules". BACON.1 (Langley, Bradshaw and Simon, 1983), and ABACUS.2 (Greene, 1988) are quantitative discovery programs which search for mathematical relationships that summarize all of the data. Other programs include the LEX system for acquiring and refining problem-solving heuristics (Mitchell, Utgoff and Banerji, 1983), Schlimmer's STAGGER (Schlimmer, 1986), which uses three learning modules, Muggleton's Duce (Muggleton, 1987) which is an oracle based approach and Pagallo and Haussler's FRINGE, GREEDY3 and GROVE (Pagallo and Haussler, 1990). Other similar efforts include CITRE (Matheus and Rendell, 1989) which constructs new terms using the repeated application of boolean operators to nodes on the positively labelled branches. An approach which uses boolean and arithmetic combinations of the original attributes to extend the initial attribute set is proposed by Utgoff (Utgoff, 1986). A method complementary to data-driven constructive induction for producing new features involves an analysis of the resulting hypotheses, rather than the training data. In this method learning on a subset of the training data results in rules ofter characterized by patterns of features and values. These features are then proposed as new attributes (AQ17-HCI, Wnek and Michalski, 1991).

AQ14 (Mozetic, 1985), the rule generation program used by DCI, learns decision rules by performing inductive inference on examples. Training examples are vectors of attribute values. These attributes may be provided explicitly by the user, or they may be constructed by the program itself as directed by the user. These constructed attributes are combinations of given attributes and involve relations selected by the user. Available operations currently include addition, subtraction, multiplication, and the equality, greater-than, or less-than relations. The addition of a newly constructed attribute to the available *attribute set* (AS) is determined by the attribute quality function (AQF). Training examples are expressed as conjunctions of attribute values, and initial or induced decision rules are logical expressions in disjunctive normal form. The program performs a heuristic search through the space of logical expressions, until it finds a decision rule that is satisfied by all positive examples and no negative examples. This search is guided by a rule preference criterion. The program is based on the AQ algorithm for solving the general covering problem (Michalski, 1969). A detailed description of the data-driven method is given in section 3, and a brief review of the AQ algorithm is given in section 4.

A drawback of programs that cannot construct new attributes is an inability to take advantage of some fairly simple relationships. For example, suppose there exist two classes of boxes, each box described by three attributes: Height, Length, and Width. Sample data are shown in table 1.

Class1			**Class2**		
Height	Length	Width	Height	Length	Width
2	12	2	12	4	2

6	4	2	4	12	2
3	8	2	8	6	2
4	4	3	4	8	3

Table 1. Example of two classes of boxes described by Height, Length and Width.

The rule which describes the characteristic of each class of box when found by a non-attribute-constructing program, such as AQ14, is fairly complex:

Class1 <:: [Height= 2 v 3] v [Height=4 v 6] & [Length=4]
Class2 <:: [Height=4 v 8] [Length=6 v 8 v 12] v [Height= 12]

Table 2. Rules describing examples in Table 1 using only original attributes.

This complexity is due to the limits of the representation language. By generating new attributes the representation language is enriched and constructed rules are simpler and more accurate. Various combinations of attributes using a variety of operations are calculated. Useful combinations, which in this case involve multiplication, are kept. In the "box" example the area of the front face of the box, Height*Length was calculated and found to be useful. This area was then retained and combined with the Width attribute to discover another useful attribute which we call volume. The system named this new attribute HLW. The rules produced using this constructed attribute are shown below.

Class1 <:: [HLW = 48]
Class2 <:: [HLW = 96]

Table 3. Rules describing examples in Table 1 using a constructed attribute.

2. GENERATION OF NEW ATTRIBUTES

The DCI method for constructing new attributes is primarily one of generate and test. First all candidate features are identified (all linear-type features). Then the operation to be performed on this pair is selected from the list supplied by the user. With the features, and operation selected the values for the new feature are calculated. The discriminatory power of these feature values is then tested using the Attribute Quality Function (AQF). The form of the AQF is shown below:

$$\frac{\#\text{Positive events}}{\text{Total Number of events in selector class}}$$

The AQF is calculated for each class, for each attribute. A positive event value is defined by the *ambig* parameter found in the parameters-table. The *ambig* parameter has three possible values: *pos*, *neg* and *empty*. A value of *pos* is meaningless for attribute quality because it results in all new features evaluating to unity. This value is, however, useful for handling ambiguous *events* when constructing rules so it is retained. The default value for the ambig parameter is *neg*. With *neg*, only values unique to a class will be considered positive for that class. Only unique values are allowed with the ambig parameter set to empty as well. The number of events in the selector class is the total number of events in the original class. A perfect discriminatory attribute, which alone discriminates one class from all the other classes, will have an AQF value of 1. Possible AQF values range from negative (total #of events in other classes-1) to 1.

A number of different operations are available to construct new features. These operations can be classified as either binary operators or multi-argument operators (functions). In the binary group are currently the relational operator, and a number of mathematical operators including: addition,

subtraction(absolute difference), multiplication, and integer division. Examples of each of these operations is shown below:

Operator	**feature 1**	**feature 2**	**result**
relation	x	y	1 if x = y; 2 if x < y; and 3 if x > y
addition	6	8	14
subtraction	6	8	2
multiplication	6	8	48
integer division	6	8	1

In the multi-argument class are the following functions: maximum, minimum, average, least_common, most_common, and #VarEQ(x). Except for the latter, these function are self-explanatory. #VarEQ(x) is a function which calculates the number of times the value x appears in a row. For a vector of binary attributes, #VarEQ(1) measures the number of variables (attributes) that take the value x in an example of a given class. Examples of each of these operations is shown below:

Operator	**feature1**	**feature2**	**feature3**	**feature4**	**result**
maximum	4	8	6	6	8
minimum	4	8	6	6	4
average	4	8	6	6	6
most_common	4	8	6	6	6
least_common	4	8	6	6	4 (first seen)
#VarEQ(4)	4	8	6	6	1
#VarEQ(6)	4	8	6	6	2
#VarEQ(8)	4	8	6	6	1

The program has a default list of global functions, but allows the user to modify the list to fit the problem at hand. The default list of functions include maximum, minimum, average, most frequent, least frequent and #VarEQ(x).

3. DESCRIPTION OF THE DCI METHOD

The algorithm for data-driven construction of new attributes is shown below. The algorithm has two stages: the first stage uses linear-typed attributes as the basis for construction. Linear type attributes have a finite number of discrete ordered values. The second stage of the algorithm uses binary nominal typed attributes. (attributes with a finite number of discrete unordered values). This stage of the algorithm is constructs sums of binary attributes, which is equivalent to searching for *m of n* concepts. Stage II is partially based on ideas found in SYM-1 (Jensen, 1975).

Stage I

1. Identify in the data all attributes that are linear.
2. Repeat steps 3 through 5 for each possible attribute pair.
3. Repeat steps 4 and 5 for each binary mathematical, or relational operator. (operators available include addition, subtraction, multiplication, division, and relation)
4. Calculate the values of this attribute pair for the given operator.
5. Evaluate the quality of this newly constructed attribute using the Attribute Quality Function (AQF) described above. If the attribute is above some threshold then store it, else discard it.
6. If new attributes are constructed, repeat steps 3-5 for each pair of new and original attributes.
7. Repeat steps 4-5 for each user-selected function (available functions include: maximum, minimum, average, least-common, most-common, #VarEQ(x)).

Stage II

1. Identify in the data all attributes that are binary.
2. Search for pairwise symmetry among the attributes and then for larger symmetry or approximate symmetry groups, based on the ideas described in (Michalski, 1969a; Jensen, 1975).
3. For each candidate symmetry group, create a new attribute that is the arithmetic sum of the attributes in the group.
4. Determine the quality function of the newly created attributes, and select the best attribute.
5. Enhance the dataset with the values of this attribute, and induce new decision rules.

Figure 1. The DCI Method

After finding all linear-type attributes, the algorithm generates every binary combination of attribute and operation. After each new attribute's values are calculated, the attribute quality function, AQF, is used to judge its quality before adding it to the AS.

After all binary combinations are proposed and evaluated, the program then calculates the values for each of the user-selected functions. The values of these new attributes are evaluated just as in the earlier case. After all linear type constructions are attempted, the program searches for all binary nominal typed attributes. The sum of binary attributes, when those attributes signal the presence or non-presence of a feature, can be described as "if x of the following y features exist" (this type of concept is also known as an "m of n" concept"). Such an attribute could possibly be quite useful in a medical domain where these binary attributes signal the presence of a symptom or disease. The *m of n* concept is also captured in the attribute "#VarEQ(x)" which is the number of values in a row (feature vector) that have the value *x*. With a group of *z* boolean attributes, #VarEQ(1)=3 is equivalent to "3 of the *z* features are true". The differences between the #VarEQ(x) function and stage II of the algorithm lie in their method and domain. The function can take non-boolean valued attributes as its domain, and it checks only the quality of combining *all* available features. Part II of the algorithm uses only boolean attributes, and initially tries combining all available features. If unsuccessful, this method tries subsets of features as well. Figure 2 gives a functional description of the algorithm.

Because the newly generated attributes are combinations of original attributes they are more complicated than the original attributes. The cost of these attributes should reflect this added complexity. To do this each operation and relation has been assigned a cost. These costs were determined from an overall ranking by the authors of selector complexity. The actual values are not meaningful, but are only meant to reflect relative complexity. These values can be changed by the user if desired. The current default values for these costs is shown in Table 5.

Operator types	**Default Cost**
relation(<,<=,=,=>,>)	1
addition (+)	5
subtraction (-)	5
multiplication (*)	9
integer division (div)	9
functions (max, min, ave)	10
functions (most-common, least-common)	11
functions (counting)	12

Table 5. Operator costs

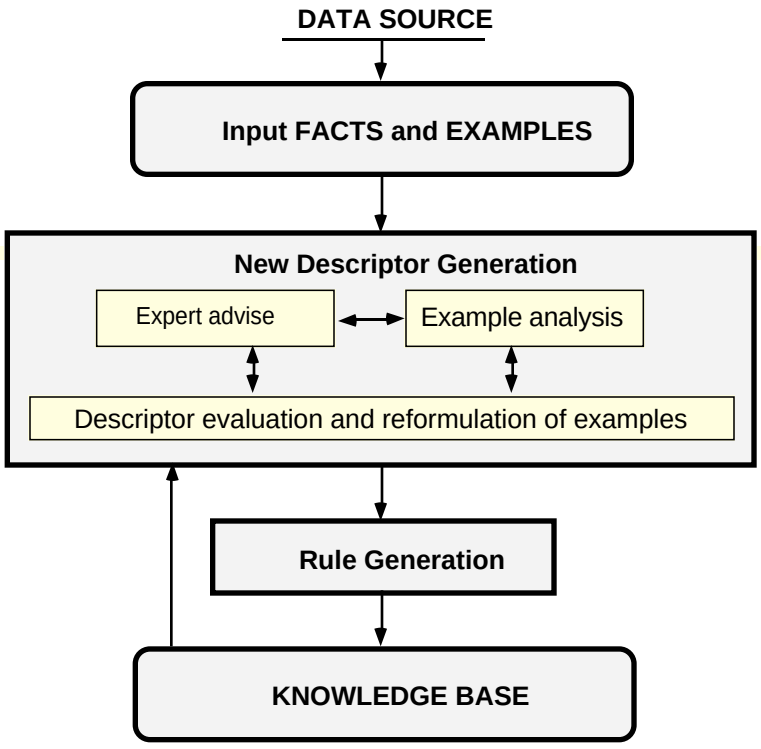

Figure 2. A functional diagram of the method.

4. A BRIEF DESCRIPTION OF THE AQ ALGORITHM

Because the AQ algorithm is used in the inductive module of this method, for completeness, we provide a brief description of it. The AQ algorithm generates the minimum or near minimum number of general decision rules characterizing a set of instances, as originally described in (Michalski, 1969; Michalski and McCormick, 1971).

1. A single positive example, called a seed, is selected and a set of most general conjunctive descriptions of this example is computed (such a set is called a star for the seed). Each of these descriptions must exclude all negative examples.
2. Using a description preference criterion a single description is selected from the star, called the 'best' description. If this description covers all positive examples, then the algorithm stops.
3. Otherwise a new seed is selected among the unexplained (uncovered) examples, and steps 1 and 2 are repeated until all examples are covered.

The disjunction of the descriptions selected in each step constitutes a complete, consistent and general description of all examples. The preference criterion used in selecting a description from a star is expressed as a list of elementary criteria that are applied lexiographically and with a certain tolerance. The criteria may be simplicity of description (measured by the number of variables used), cost (the sum of the given costs of the individual variables), or other criteria (Michalski and Larson, 1978).

The description of a class is expressed using the variable-valued logic system 1 (VL_1), which is a multiple-valued logic propositional calculus with typed variables (Michalski, 1974). A class description is called a cover which is a disjunction of complexes describing all positive examples and none of the

negative examples. A complex is a conjunction of selectors, and is the simplest statement in VL_1. A selector relates a variable to a value or a disjunction of values, for example [temperature = cold], or [x < 5]. The general form of a selector (condition) is:

[L # R]

where L, called the referee, is an attribute, and R, called the referent is a set of values in the domain of the attribute in L, # is a relational symbol which can be one of the following: =, <,>,>=,<=,<>.

5. EXPERIMENTAL RESULTS

5.1 Monk's Problems

The Monk's problems are three artificial problems proposed by the attendees of the 2nd European Summer School on Machine Learning (Thrun et.al. 1991). These three problems were intended to serve as benchmarks with which to test a wide variety of learning algorithms. Each problem is a binary classification task. From a subset of the total data, the task is to produce a generalized description that accurately predicts the membership of the remaining events in the learning space. The goal concept of the first problem is stated in standard disjunctive form and is given by:

[x1 = x2] or [x5 = 1]

From 432 possible examples 124 were randomly selected for the training set. There were no misclassifications.The second problem has a more distributed coding, similar to the parity problems and the goal concept is given as:

exactly two of the six attributes have their first value

From 432 possible examples 189 were randomly selected. There was no noise. The third problem has noise in the training set and the goal concept is given by:

[x5 = 3] and [x4 = 1] or [x5 <> 4] and [x2 <> 3]

From 432 examples, 122 were randomly selected. Among the training set were 6 misclassifications.

The performance of AQ17-DCI and a number of other machine-learning programs applied to the Monk's problems is shown in table 6 (table from Thrun, et. al. 1991). AQ17-DCI performed very well capturing perfectly the first and second goal concepts. For the third concept an approach designed to handle noisy data was employed with the data-driven method. The approach for the third problem and resulting testing value was only recently completed and is not shown in (Thrun et al. 1991). The standard data-driven method resulted in the 94.2 testing accuracy reported by Thrun.

The noise-tolerant method used by AQ17-DCI for the third Monk problem is derived from the one used by AQ17-NT (Thrun, et al, 1991). This method selects those examples in the data which were covered by light disjuncts in the generated rules. These examples are eliminated from the training set, and rule generation is repeated. In the AQ17-DCI/NT approach feature construction and rule generation is performed as normal, followed by example elimination and new feature generation and rule construction. A detailed description of the noise-tolerant method can be found in the AQ17-NT description (Thrun, et al, 1991).

The values shown in Table 6 and table 8 for the AQ programs were calculated using a testing tool called ATEST (Reinke, 1984). In ATEST rule performance is measured by the degree of agreement between a

Program	Percent Correct Classification		
	#1	#2	#3
AQ17-DCI	100%	100%	97.2%
AQ14	100%	77%	84%
AQ17-HCI	100%	93.1%	100%
AQ17-FCLS		92.6%	97.2%
AQ17-NT			100%
AQ17-GA			100%
Assistant Professional	100%	81.2%	100%
mFOIL	100%	69.2%	100%
ID5R	81.7%	61.8%	
IDL	97.2%	66.2%	
ID5R-hat	90.3%	65.7%	
TDIDT	75.7%	66.7%	
ID3	98.6%	67.9%	94.4%
ID3, no windowing	83.2%	69.1%	95.6%
ID5R	79.7%	69.2%	95.2%
AQR	95.9%	79.7%	87.0%
CN2	100%	69.0%	89.1%
PRISM	86.3%	72.7%	90.3%
Backpropagation	91.7%	100%	87.7%
Cascade Correlation	100%	100%	97.2%

Table 6. Summary of results for Monk's Problems

class description rule and a testing example from an assigned class. ATEST views rules as expressions when comparing them to a vector of attributes (e.g., a testing example). The result of this evaluation is a real number which is the degree of consonance between the conditional part of the rule and the event. For a set of rules and classified testing events, ATEST resturn three values: overall percent correct, overall percent first rank correct, and overall percent only choice. Overall correct is the percentage of events that matched the correct class with degree greater than 50%. The overall percent first rank value is the percentage of events which matched the correct description rule within a small degree of error (this is also known as a flexible match). Percent correct only is the percentage of events for which the correct rule was the one that matched the testing example to the highest degree. Results for AQ17-DCI for problems 1 and 2 are overall percent correct only choice, and for problem three is overall percent first rank.

Program	Number of disjuncts in Rule		
	#1	#2	#3
AQ17-DCI	3	3	3
AQ14	8	35	8

Table 7. Number of disjuncts:Monk's Problems

Not only did AQ17-DCI perfectly predict all of the events in the testing class, but it also produced rules that were very simple. In the first problem for example, the rule produced by AQ14 for the negative class was:

Pos-class
[x1=1][x2=2,3][x5=2..4] or
[x1=2,3][x2=1][x5=2..4] or
[x1=2][x2=3][x5=2..4] or
[x1=3][x2=2][x5=2..4]

This rule performed perfectly in the testing data, but it is much more complicated than the rule produced by AQ17-DCI:

Pos-class
[x5=2..4][x1<>x2]

AQ17-DCI produced rules of significantly higher accuracy for problems 2 and 3 than AQ14. In addition the rules produced by AQ17-DCI were usually simpler than those produced by AQ14. Table 7 shows the total number of disjuncts in the rules produced by AQ17-DCI and AQ14 for the three Monk problems

5.2 Congressional Voting

This data in these experiments is from voting records for members of the U.S. House of Representatives for 1981 and 1984. In both datasets, there are two classes: Republican and Democrat. The goal of this learning is to find discriminant descriptions for the voting records of the two parties. The 1981 training data consists of 51 events (31 democrat and 20 Republican) described by 19 attributes. These attributes are primarily records of a member's vote on a particular issue (MX missle, for example), but also includes attributes concerning the members home state location, and income. The testing set for the 1981 data consisted of 49 events (29 Democrat and 20 Republican). The 1981 data was taken from (Bergandano et al, 1990) where is was used to test a system called POSEIDON.

The 1984 data was randomly split into equal-sized groups of 116 events (62 Democrat and 54 Republican) described by 16 boolean attributes for testing and training. This data is from the house-voting-records data found in the UCI database with all of the events containing unknown values removed. All data was converted to integer values so that it may be operated upon by the data-driven operators.

For the 1981 data, the difference operator found two powerful new features: (draft_vote diff education_vote), and (draft_vote diff food_stamp_cap_vote). The difference operator applied to these

attributes is equivalent to a check for equality. If the difference between voting records is 0, then the votes were the same, otherwise the votes were different. AQ17-DCI found useful new features that increased predictive accuracy, and that were meaningful. For the 1984 data the relational operator was used, and it found three new useful features. These three new features were (physician-fee-freeze rel anti-satellite-test-ban), (adoption-of-the-budget-resolution rel el-salvador-aid) and (adoption-of-the-budget-resolution rel superfund-right-to-sue). Once again these features represent relationships between votes on different issues. The rule found by AQ17-DCI, for the Republican class is shown below:

[physician-fee-freeze = no] or
[adoption-of-the-budget-resolution = el-salvador-aid] and [physician-fee-freeze > anti-satellite-test-ban]

This rule can be translated into "A Republican votes against physician fee freezes, *or* he votes the same on the adoption of the budget bill as he does on el-salvador aid, and votes for physician-fee-freeze and against the anti-satellite-test-ban". Results from testing AQ17-DCI, AQ14, POSEIDON, ASSISTANT and a simple-exemplar based method on the 1981 data and AQ17-DCI and AQ14 on the 1984 data are shown in table 8. The results for POSEIDON, ASSISTANT, and the exemplar method are taken from (Bergadano et al, 1990). Results for AQ17-DCI, and AQ14 are overall percent correct first rank results from ATEST (see section 5.1 for description of ATEST).

Percentage Correct Classification		
Program	**1981**	**1984**
AQ17-DCI	**95.9 %**	**98.2 %**
AQ14	**85.7 %**	**93.1 %**
POSEIDON	**92 %**	**-**
ASSISTANT	**86 %**	**-**
Simple exemplar-based method	**86 %**	**-**

Table 8. Summary of results for AQ17-DCI, AQ14, POSEIDON, ASSISTANT and a simple exemplar-based method on Congress voting records

6. CONCLUSION

The method of data-driven constructive induction implemented in AQ17-DCI, performed well in the experiments producing new features that were useful as well as understandable. In the Monk's problems new features were constructed that represented the relationship between values, and the number of times a particular value was present (value cardinality). Features representing relationships *between* votes, Republican's vote the same on the budget and on aid to El Salvador for example, were found in the Congressional voting domain With these features rules which were meaningful, and had high predictive acccuracy were generated.

Despite these strong results, the method reported here can be improved. One improvement would be to check the original feature set for quality: if there already exists a single feature that discriminates between the given classes, then constructive induction is not needed. Another improvement involves the evaluation of new features. New features should be rewarded for covering previously uncovered

examples, rather than for those already covered. Uncovered examples are likely the ‘hard’ examples that cause complex rules and should be the inspiration for new features. However, these examples may also be noise and should be avoided. A method for satisfying both concerns is being investigated.

References

F. Bergadano, S. Matwin, R.S. Michalksi, and J. Zhang, "Learning Two-Tiered Descriptions of Flexible Concepts: The POSEIDON System", *Reports of Machine Learning and Inference Laboratory*, MLI 90-10, Center for Artificial Intelligence, George Mason University, Fairfax, VA, 1990.

G.H. Greene, "Quantitative Discovery: Using Dependencies to Discover Non-Linear Terms", M.S. Thesis, University of Illinois at Urbana-Champaign, 1988.

G. Jensen, "SYM-1: A Program that Detects Symmetry of Variable-Valued Logic Functions", UIUCDCS-R-75-729, Department of Computer Science, University of Illinois at Urbana-Champaign, 1975.

C. J. Matheus, and L.A. Rendell, "Constructive Induction on Decision Trees", In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 645-650, 1989.

R.S. Michalski, "Recognition of Total or Partial Symmetry in a Completely or Incompletely Secefies Switching Function," Proceedings of the IV Congress of the International Federation on Automatic Control (IFAC), Vol. 27 (Finite Automata and Switching Systems), pp. 109-129, Warsaw, June 16-21, 1969.

R.S. Michalski, "On the Quasi-Minimal Solution of the Covering Problem" *Proceedings of the V International Symposium on Information Processing* (FCIP 69), Vol. A3 (Switching Circuits), Bled, Yugoslavia, pp. 125-128, 1969.

R.S. Michalski and B.H. McCormick, "Interval Generalization of Switching Theory." Report No. 442, Dept. of Computer Science, University of Illinois, Urbana. 1971.

R.S. Michalski, "Variable-Valued Logic: System VL_1, *Proceedings of the 1974 International Symposium on Multiple-Valued Logic*, pp. 323-346. West Virginia University, Morgantown, 1974.

R.S. Michalski and J.B. Larson, "Selection of Most Representative Training Examples and Incremental Generation of VL_1 Hypotheses: the underlying methodology and the description of programs ESEL and AQ11, " Report No. 867, Dept. of Computer Science, University of Illinois, Urbana, 1978.

R.S. Michalski, "Pattern Recognition as Rule-Guided Inductive Inference," IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI, vol 2, No. 4, pp.349-361, 1980.

T. Mitchell, Utgoff, P. and Banerji, R., "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics," in Machine Learning: An Artificial Intelligence Approach, R. Michalski, J. Carbonell, and T. Mitchell (eds.), Morgan Kaufman, Los Altos, CA, 1983.

I. Mozetic, "NEWGEM: Program for Learning from Examples - Program Documentation and User's Guide", Report No. UIUCDCS-F-85-949, Department of Computer Science, University of Illinois at Urbana-Champaign, 1986.

S. Muggleton, "Duce, an Oracle-Based Approach to Constructive Induction", Proceedings of IJCAI-87, pp. 287-292, Morgan Kaufman, Milan, Italy, 1987.

G. Pagallo and D. Haussler, "Boolean Feature Discovery in Empirical Learning", *Machine Learning*, vol. 5, pp. 71-99, 1990..

R.E. Reinke, “Knowledge Acquisition and Refinement Tools for the ADVISE Meta-expert System,” Master’s Thesis, University of Illinois, 1984.

J. Schlimmer, "Concept Acquisition Through Representational Adjustment," Machine Learning, vol. 1, pp. 81-106, 1986.

S. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S. Fahlman, R. Hammann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, H. Vafaie, W. Van de Velde, W. Wenzel, J. Wnek, and J. Zhang, “The Monk’s Problem’s: A Performance Comparison of Different Learning Methods”, Carnegie Mellon University, October, 1991.

P. Utgoff, "Shift of Bias for Inductive Learning,", in Machine Learning: An Artificial Intelligence Approach, Vol. II, R. Michalski, J. Carbonell, and T. Mitchell (eds.), Morgan Kaufman, Los Altos, CA, pp. 107-148, 1986.

J. Wnek, R.S. Michalksi, “Hypothesis-Driven Constructive Induction in AQ17: A Method and Experiments”, MLI Report 91-9, Center for Artificial Intelligence, George Mason University, Fairfax, Va. 1991.