Knowledge Discovery using Genetic Programming with Rough Set Evaluation

David H. Foster, W. James Bishop, Scott A. King, Jack Park
ThinkAlong Software, Inc.
Brownsville, California 95919
jpark@aip.org

Abstract

An important area of KDD research involves development of techniques which transform raw data into forms more useful for prediction or explanation. We present an approach to automating the search for "indicator functions" which mediate such transformations. The fitness of a function is measured as its contribution to discerning different classes of data. Genetic programming techniques are applied to the search for and improvement of the programs which make up these functions. Rough set theory is used to evaluate the fitness of functions.

Rough set theory provides a unique evaluator in that it allows the fitness of each function to depend on the combined performance of a population of functions. This is desirable in applications which need a population of programs that perform well in concert and contrasts with traditional genetic programming applications which have as there goal to find a single program which performs well.

This approach has been applied to a small database of iris flowers with the goal of learning to predict the species of the flower given the values of four iris attributes and to a larger breast cancer database with the goal of predicting whether remission will occur within a five year period.

Introduction

An important area of KDD research involves development of techniques which transform raw data into forms more useful for prediction or explanation. We present an approach to automating the search for "indicator functions" which mediate such transformations. The fitness of a function is measured as its contribution to discerning different classes of data. Genetic programming techniques are applied to searching for and improving the programs which make up these functions. Rough set theory is used to evaluate the fitness of functions.

Rough set theory provides a unique evaluator in that it allows the fitness of each function to depend on the combined performance of a population of functions. This is desirable in applications which need a population of programs that perform well in concert and contrasts with traditional genetic programming applications which have as there goal to find a single program which performs well.

This approach has been applied to a small database of iris flowers with the goal of learning to predict the species of the flower given the values of four iris attributes (Fisher's iris data reproduced in Salzberg, 1990) and to a larger breast cancer database (breast cancer data reproduced in Salzberg, 1990) with the goal of predicting whether remission will occur within a five year period.

The process begins by applying a population of randomly-generated programs to elements of a database. Program results are placed in a matrix and evaluated to obtain a measure of each program's fitness. These fitness values are used to determine which programs will be kept and used in breeding the next generation. This process continues for a specified number of generations and is illustrated in Figure 1.

From: AAAI Technical Report WS-93-02. Compilation copyright © 1993, AAAI (www.aaai.org). All rights reserved.

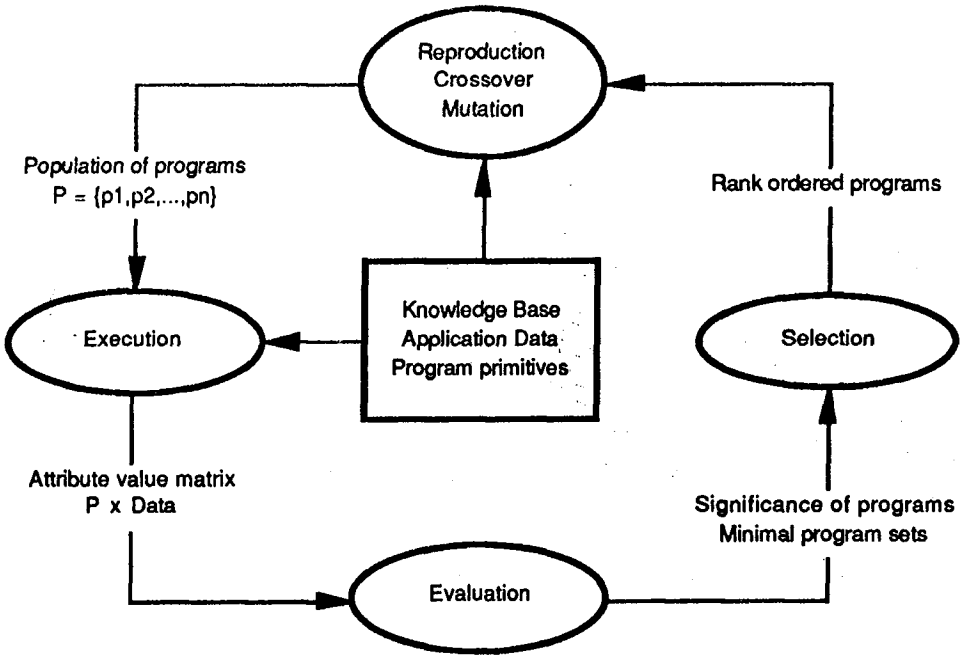

Figure 1: Genetic Programming

Executing Programs and Filling Evaluation Matrix

Programs are constructed of boolean operators connecting application-specific primitives. Our implementation constructs programs using scheme, a dialect of lisp. This representation allows programs to be both easily modified and directly executable.

An evaluation matrix is created by applying each program to each point in the training data. For each data point (i), and for each program (j), the program is executed and the result is stored at location (i,j) in the matrix. Thus each column of the matrix contains the values returned by a particular program for all the data points, and each row contains the values returned by all programs for a particular data point.

Rough set theory is applied to the evaluation matrix to determine each programs fitness.

Rough Set Fitness Evaluation

Our fitness evaluation is based on rough classification described by Pawlak (1984,1991) and Ziarko (1989). These methods allow the fitness of functions to be evaluated in a context with other functions. We expect this approach to promote population diversity by a natural tendency to assign lower fitness to redundant programs.

Definitions

S = (U,A,V,f) is an information system consisting of a set of data objects (U), a set of attributes (A), a set of possible attribute values (V), and a function (f) that maps data object-attributes pairs to attribute values.

U is the set of all data objects. In our irises application it is the set of all examples in our database.

A is the set of all attributes. In our implementation it is the set of concepts measured or tested by the attribute programs. The set A also corresponds to the set of all columns in the evaluation matrix.

V is the domain of attribute values. In our irises application it is {setosa, virginica, versicolor} in the prediction column and { true, false} elsewhere.

f is the description function mapping $U \times A \rightarrow V$. In our implementation it corresponds to the evaluation matrix.

N is the set of predicted attributes or columns of the evaluation matrix. In our application it is a single column containing the species name for the data objects.

P is the set of predictor attributes or columns of the evaluation matrix. These correspond to the set of programs to be determined.

M is a minimal subset of P which retains the full ability of P to discern elements of N'.

N' is the set of elementary sets or equivalence classes based on predicted attributes. It corresponds to the set of sets of rows which have matching values in all the N columns.

P' is the set of elementary sets or equivalence classes based on all predictor attributes. It corresponds to the set of sets of rows which have matching values in all the P columns.

M' is a set of elementary sets or equivalence classes based on a minimal set of predictor attributes. It corresponds to the set of sets of rows which have matching values in all the M columns.

$\underline{ind}(P)n'_i$ is the union of all elementary sets in P' which are subsets of the ith element in N'. These are the lower approximations of the sets in N'.

POS(P,N) is the union of $\underline{ind}(P)n'_i$ for all elements n'_i in N'. This is the subset of U for which P is sufficient for discerning membership in the equivalence classes of N'.

$k(P,N) \equiv card(POS(P,N)) / card(U)$.
This is the fraction of the set U for which P is sufficient for discerning membership in the equivalence classes of N'.

$SGF(P,N,p_i) \equiv (k(P,N) - k(P-p_i,N)) / k(P,N)$.

This is the relative change in k(P,N) resulting from deletion of p_i from P. SGF has the advantage that it rewards programs for their contribution to recall or prediction in the context of all other programs.

Evaluating Attributes

The sets and measures above are used to determine the significance of members of P for classifying members of U into the equivalence classes of N'. One goal is to find a minimal subset of attributes that retains the capability of all P for discerning elements of N'. Another goal is to assign a significance value to each attribute. This value will be used in calculating program fitness for the genetic programming process.

The following method calculates significance factors for each program while determining a minimal set. Programs with zero significance are not included in the minimal set, M.

Initialize: $M_1 = P$

For each p_k in P:

Calculate $SGF(M_k,N,p_k)$

If $SGF(M_k,N,p_k) = 0$

then $M_{k+1} = M_k - p_k$

else $M_{k+1} = M_k$

The minimal set is not unique. The order of selecting p_k during calculation of M will affect the final contents of M. We have chosen to select the p_k beginning with the previous generation and in order from lowest to highest SGF. This choice of old rules first encourages turnover in the population by allowing an older program to be replaced by an equivalent set of new programs.

Genetic Programming Process

Our implementation of genetic programming is based on Koza (1992). Genetic programming is used to modify a population of programs which perform tests on the data. Programs in this study are constructed of boolean expressions. The terminal values of these expressions are generated by application-specific primitives. These primitives are simple functions which return true and false answers to questions about the data. For example one primitive asks whether the ratio of petal length to petal width is greater than a randomly generated number. The following describes the generation cycle:

Create an initial (random) population of programs.

1. Execute and evaluate programs to determine fitness.
2. Rank order programs according to fitness.
3. Generate a new population of programs by applying reproduction, crossover, and mutation to the best of the old programs.

Go to step 1.

NOTE: This cycle terminates when a predefined number of generations have passed or when accuracy of recall and prediction has failed to improve in the previous n generations. Refer to "Testing Recall and Prediction".

Normally the result is chosen as the best program to appear in any generation. In our application the result is the final population of programs with non-zero fitness. These programs will be used in concert for recall and prediction.

Survival and Reproduction

While investigating survival and reproduction methods we examined the following:

Program Survival

We have experimented with three options for survival of programs from one generation to the next.

1. Retain a fixed number of the best programs.
2. Calculate the minimum SGF for all programs and retain all programs with an SGF greater than the minimum.

   NOTE: Typically several programs will posses the minimum SGF and will therefore be discarded.
3. Retain M; i.e. programs with SGF > 0. Note that M is a minimal set so it will by definition contain a diverse population.

Program Reproduction

We have experimented with two options for production of new programs from one generation to the next.

1. Replace all discarded programs thus maintaining a constant P.
2. Supplement M by a constant number of programs thus allowing P to increase or decrease according to the size of the minimal set M. This causes the program to adapt as the problem progresses toward solution.

Regardless of the options chosen the general sequence of events were as follows:

1. Select survivors.
2. Determine the number or programs to be reproduced.
3. According to a preset percentage reproduce a fraction of these via crossover.
4. Reproduce the remainder via mutation.

Crossover and Mutation

Crossover was implemented by selecting one sub-expression from each parent and swapping them to produce two new programs. Mutation was implemented by selecting a sub-expression within a single parent and replacing it with a randomly generated expression.

Program Crossover

The crossover operator first conducts a random selection of two members of the current population of programs. The selection is weighted by program fitness in the "roulette wheel" fashion. This involves assigning each program a fraction of an interval proportional to its fitness, randomly selecting an element of this interval, and choosing the program determined by this element. A crossover point is selected, again at random, within each program and the sub-expressions below these points are swapped to produce two new programs. Figure 2 shows two programs with sub-expressions selected.

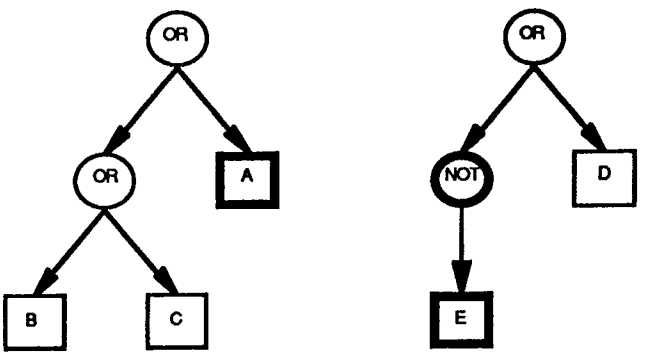

Figure 2: Sub-expressions selected for crossover in parent programs

Figure 3 shows the two new programs created by the crossover operation.

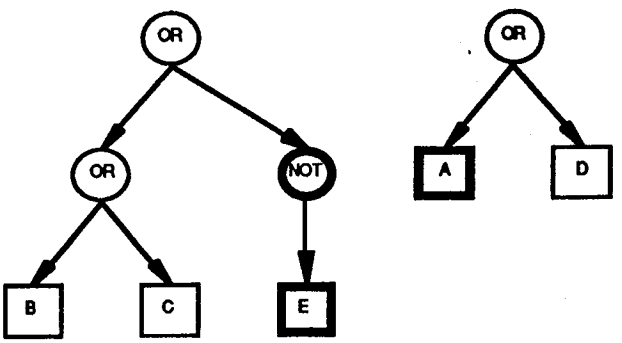

Figure 3: New programs created by crossover

Program Mutation

The mutation operator first makes a weighted random selection of one member of the current population of programs. A mutation point is randomly selected within the program and the sub-expression below this point is replaced with a randomly generated expression. Figure 4 shows an example of an original program and one possible result of applying the mutation operator.

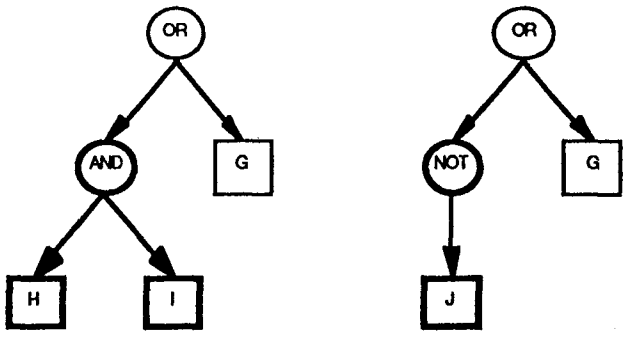

Figure 4: Original and new program created by mutation

Testing Recall and Prediction

We have tested our implementation with a relatively small database of iris flowers (Fisher's iris data reproduced in Salzberg, 1990) and with a larger database of breast cancer patients (Breast cancer data reproduced in Salzberg, 1990).

We tested for both recall and prediction. The data was partitioned into two disjoint sets for training and prediction testing. Recall was tested using a subset of the training data. Prediction was tested using data which was not used in training. Testing was accomplished as follows:

> Each program in the minimal set, M, is applied to the test data point.
>
> The resulting list of values is matched against the corresponding values in each row of the evaluation matrix. We use an analog of hamming distance to select a matching row for prediction. We calculate the distance as the sum of SGF(M,N,m_i) for columns m_i that do not match the corresponding test value. The row with the minimum distance from the list of test values is selected. If several rows have the same distance measure then the first of these is arbitrarily selected.
>
> The field to be recalled or predicted is retrieved from the N columns of the selected row. Success is measured as the fraction of the test data that is correctly recalled or predicted.

Our testing confirmed the ability of this approach to recall stored patterns and to predict from unseen patterns.

IRIS DATA

Each entry includes the name of the species and values for sepal length, sepal width, petal length, and petal width. The prediction field is species.

We achieved 100% accuracy on recall and 96% accuracy for prediction. This is consistent with the 93% to 100% accuracy reported by Salzberg (1990). Our results for the iris database are illustrated below.

Figure 4 shows the k(P,N) values obtained during iris training.

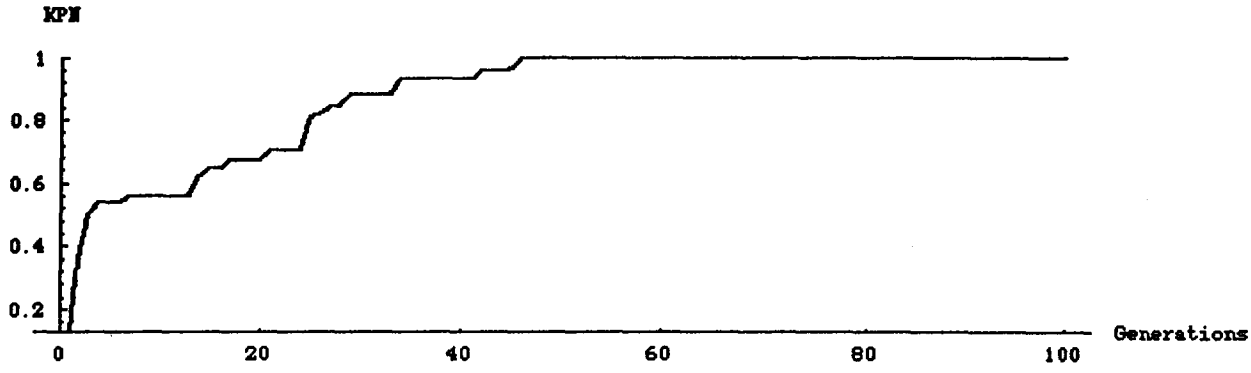

Figure 4: k(P.N) during iris training

Figure 5 shows results of testing for pattern recall during iris training. This data was obtained by testing on a subset of the data used for training.

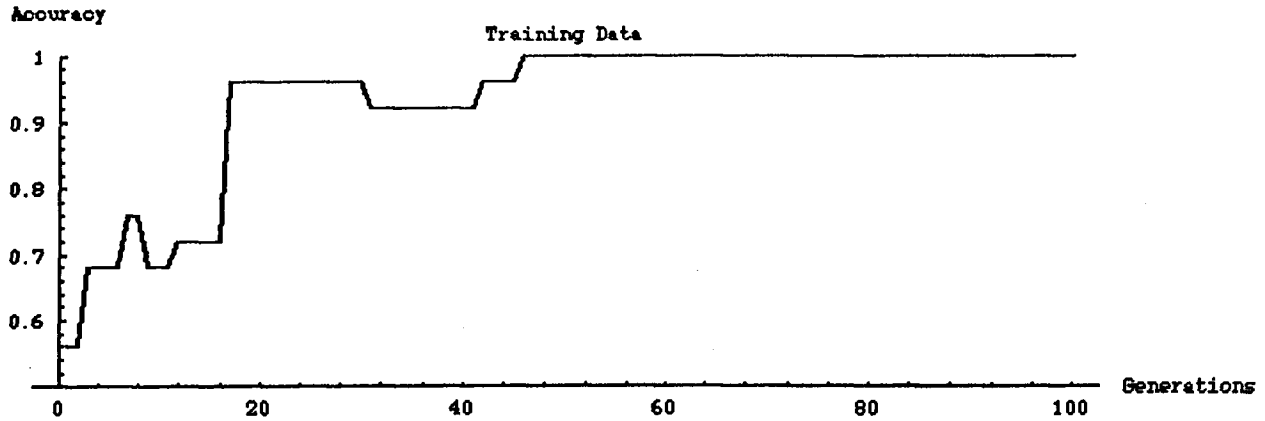

Figure 5: Recall accuracy during iris training

Figure 6 shows results of testing for predictive ability during iris training. This data was obtained by testing on a subset of the data disjoint from that used for training.

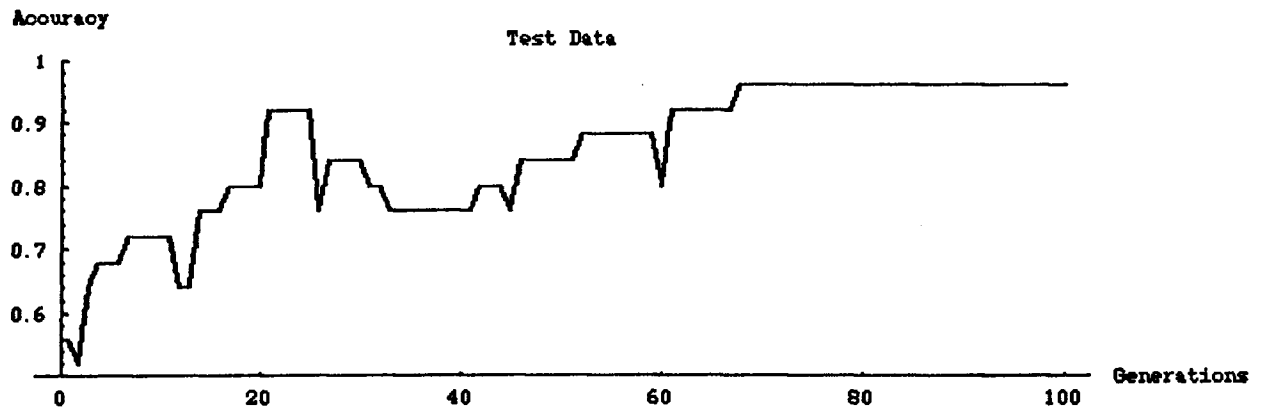

Figure 6: Prediction accuracy during iris training

Figure 7 shows changes in the size of the minimal set, M, during iris training.

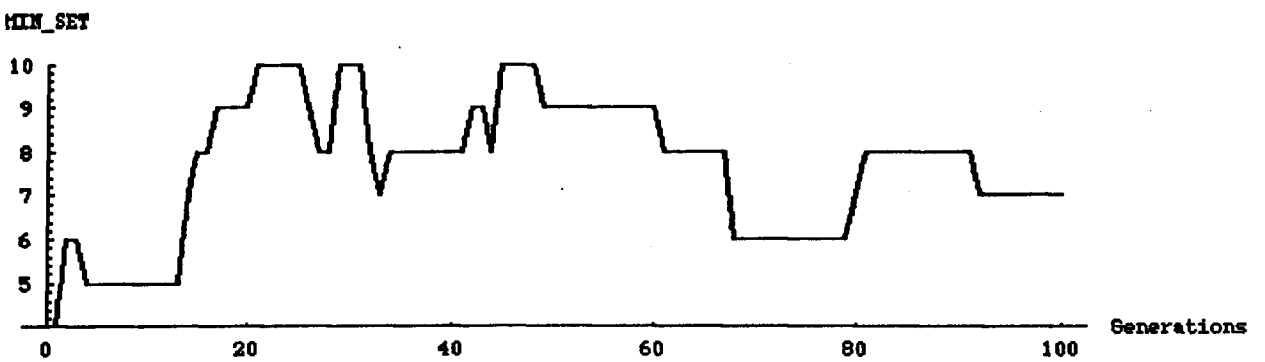

Figure 7: Number of programs in minimal set during iris training

CANCER DATA

Each entry includes the patient age, whether they had gone through menopause, which breast had the cancer, whether radiation was used, whether the patient suffered a recurrence during the five years following surgery, the tumor diameter and four other measures of the tumor itself. The prediction field is recurrence.

We achieved 100% accuracy on recall and 79% accuracy for prediction. Again this is comparable to the 78% accuracy reported by Salzberg (1990). These results are illustrated below.

Figure 8 shows the k(P,N) values obtained during cancer training.

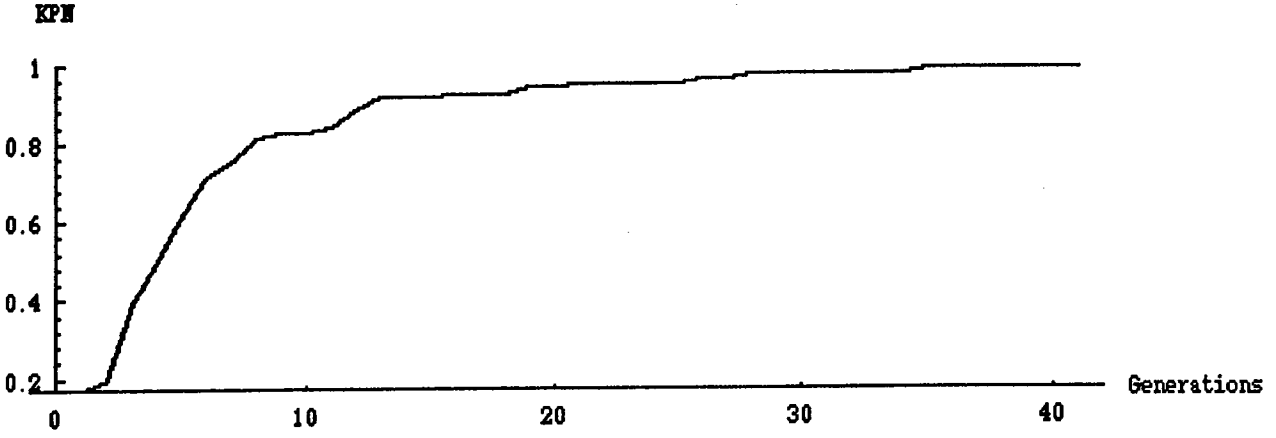

Figure 8: k(P.N) during cancer training

Figure 9 shows results of testing for pattern recall during cancer training.

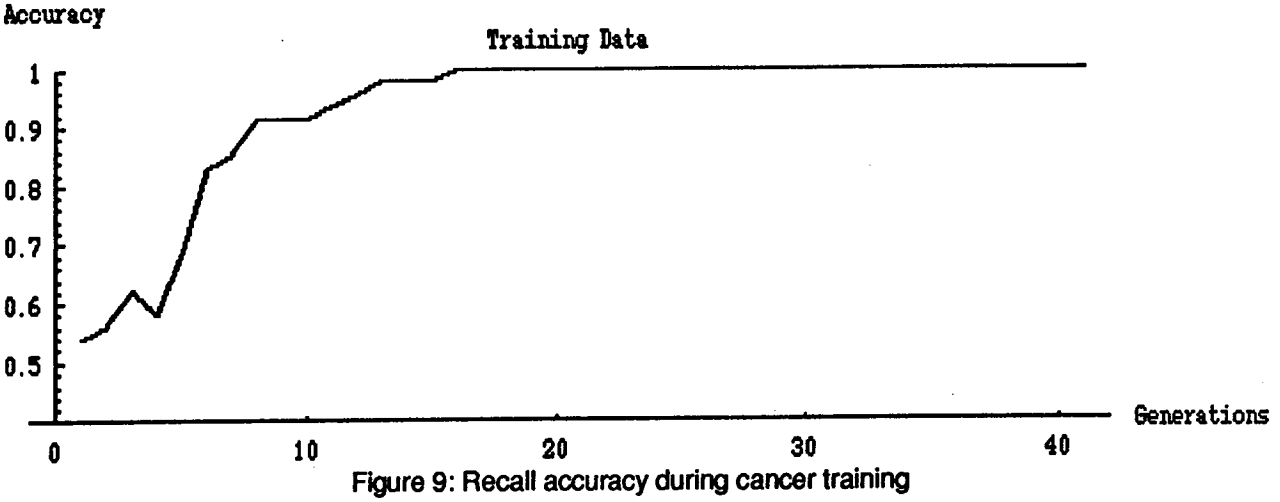

Figure 9: Recall accuracy during cancer training

Figure 10 shows results of testing for predictive ability during cancer training.

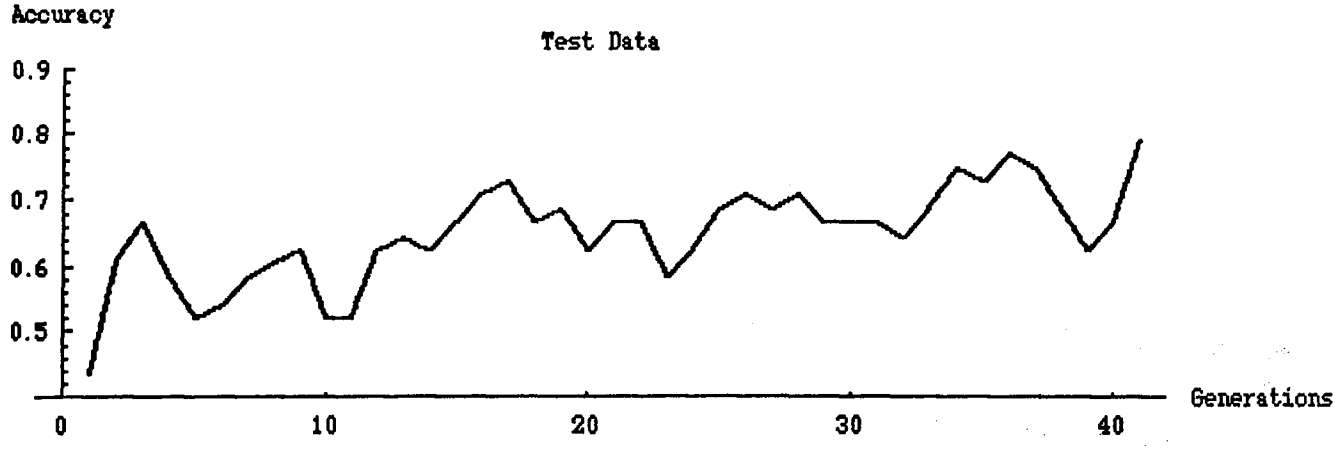

Figure 10: Prediction accuracy during cancer training

Figure 11 shows changes in the size of the minimal set, M, during cancer training.

Figure 11: Number of programs in minimal set during cancer training

Conclusions

Our test results suggest there may be two distinct aspects to learning in this approach. The first is the development of a population of programs sufficient to recall members of the training set. We see this in Figures 4 and 5 and Figures 8 and 9 where k(P,N) and the recall accuracy proceed to the maximum value of 1.0. At this point we might expect learning to stop. However, in a number of trials we found that prediction accuracy continued to improve, albeit sometimes erratically.

Looking at Figure 6 we see that prediction accuracy continued to increase for some time after k(P,N) reached its maximum. We suggest an explanation may be found in the size of the minimal set shown in Figure 7. At about the same time as prediction accuracy began its rise to a final maximum of 96% the size of the minimal set began to decrease from a high of 10 programs to a range of 6-8 programs. This is not inconsistent with other machine-learning techniques in which smaller representations tend to have a greater ability to generalize.

We believe the approach described here will have important advantages for some difficult applications of KDD. The biggest drawback to this approach is the computational cost. Our implementation performed well on iris and cancer data bases but the computational burden was a significant problem in preliminary tests on the much larger task of discovering regularities between the primary and secondary structure of proteins.

We have experimented with reducing the effective size of the data set through random sampling. This made the evaluation process much faster, but accuracy tended to stop improving and begin fluctuating about slightly lower values.

Future directions

There are several aspects of this approach that warrant further research. The most important of these are related to improving computational efficiency. It would be useful to explore the relationships between the complexity of individual programs, the size of the program population, and the time to solution.

Reducing the search space for the genetic programming process is a worthwhile goal. The use of more intelligent methods for creating and modifying primitive functions is one possibility. Another possibility would be to reduce the dimensionality of the data using application-specific knowledge for preprocessing. Alternatives to strict boolean expressions should also be explored.

Stopping criteria would be useful for less goal-oriented applications of KDD such as database mining. For example, in a blind search for regularities, the data attributes might be partitioned into different N (predicted) and P (predictor) sets. The methods presented here would be applied for each partition in a search for interdependencies.

REFERENCES

Koza, John R. *Genetic Programming: on the programming of computers by means of natural selection.* 1992, Cambridge MA, MIT Press.

Orlowska, Ewa and Pawlak, Zdzislaw. "Expressive power of knowledge representation systems," 1984, *Int. J. Man-Machine Studies,,* vol. 20 pp. 485-500, London, Academic Press Inc.

Park, Jack. "On an Approach to Index Discovery." 1993, submitted to KDD 1993.

Park, Jack, and Dan Wood. *User's Manual—The Scholar's Companion.* 1993, Brownsville, CA, ThinkAlong Software.

Pawlak, Zdzislaw. "Rough Classification," 1984, *Int. J. Man-Machine Studies,,* vol. 20 pp. 469-483, London, Academic Press Inc.

Pawlak, Zdzislaw. *Rough Sets: Theoretical Aspects of Reasoning about Data,* 1991, Boston, Kluwer Academic Publishers.

Salzberg, Steven L., *Learning with nested generalized exemplars,* 1990, Boston, Kluwer Academic Publishers.

Ziarko, Wojciech. "A Technique for Discovering and Analysis of Cause-Effect Relationships in Empirical Data," 1989, *KDD Workshop 1989.*