# Learning Weighted Prototypes using Genetic Algorithms

**Jianping Zhang and Qiu Fan**
Department of Computer Science
Utah State University
Logan UT 84322-4205
jianping@zhang.cs.usu.edu

**Abstract**

Smith and Osherson proposed the prototype view for concept representation and category classification. In the prototype view, concepts are represented as prototypes. A prototype is a collection of salient properties of a concept. Under the prototype view, an instance is classified as a member of a concept if it is sufficiently similar to the prototype of this concept. Although the prototype view has been extensively researched in cognitive science, it has not been widely adopted in machine learning. In this paper, we discuss some preliminary work on a genetic algorithms approach to learning weighted prototypes. In this approach, a concept is represented as one or more weighted prototypes, each of which is a conjunction of weighted attribute values. In this approach, every prototype maintains its own attribute weights. A genetic algorithm is applied to generate prototypes and their attribute weights. This approach has been implemented in GABWPL (Genetic Algorithm Based Weighted Prototype Learning) and empirically evaluated on several artificial datasets.

## 1. Introduction

Smith and Osherson (1984) proposed the prototype view for concept representation and category classification. In the prototype view, concepts are represented as prototypes. A prototype is a collection of salient properties of a concept. Under the prototype view, an instance is classified as a member of a concept if it is sufficiently similar to the prototype of this concept. The prototype representation strongly supports human concept formation. People tend to remember those most often encountered instances and forget those rarely encountered instances. Concepts involved in real world applications usually possess graded structures (Barsalou 1985). Instead of being equivalent, instances of a concept may be characterized by a degree of typicality in representing the concept. Prototypes represent the central tendencies of such graded structures, so concepts described by prototypes are more human understandable than those described by regular instances and also easier for human to capture the basic principles underlying these concepts.

Although the prototype view has been extensively researched in cognitive science, it has not been widely adopted in machine learning. In recent years, instance-based learning (IBL) becomes popular for several reasons. First, it is strongly motivated by similar psychologically plausible algorithms that perform concept formation (Smith & Medin 1981; Aha & Goldstone 1992; Zhang 1992). Second, polymorphy and imprecision of natural concepts prevent many inductive learning approaches from inducing general concept descriptions, while instance-based (exemplar-based) approaches perform well in domains involving polymorphous and imprecise concepts (Bareiss, Porter, & Wier 1990). Third, instance-based learning algorithms are simple and easy to understand and implement. A number of instance-based learning methods, e.g. Protos (Bareiss, Porter, & Wier 1990), IBn (Aha, Kibler, & Albert 1991), and Each (Salzberg 1991), were developed and applied to many practical problems. These problems include clinical audiology (Bareiss, Porter, & Wier 1990), diagnosis of heart diseases, classification of congressional voting records (Aha & Kibler 1989), prediction of protein secondary structures, word pronunciation (Stanfill & Waltz 1986), and prediction of DNA promoter sequences (Cost & Salzberg 1991). The results obtained by these IBL methods were comparable to those obtained from other learning methods.

Unfortunately, IBL has its disadvantages. Because concept descriptions are represented by many individual instances, they are not human understandable. As we mentioned above, a prototype represents a central tendency of a concept, thus prototypes are easy for human to understand the concepts represented. Each classification of a novel instance involves measuring the distance between the novel instance and each of the stored instances, so it becomes very inefficient to make classifications when the number of stored instances becomes large. Therefore, it is important to develop effective storage reduction algorithms for IBL. Skalak (1993) showed that significant reduction in storage of instances can be achieved in some datasets using a few prototypes without decreasing IBL's classification accuracy.

Genetic algorithms are search techniques. A genetic algorithm represents a specialized search technique, which, although it is not guaranteed to arrive at a "best" solution, generally arrives at a good or near optimal solution. Furthermore, in terms of time, it is considerably more efficient than random or exhaustive searches when the search space is large. Learning a concept description is a search problem. There have been many applications of genetic algorithms to machine learning (e.g., Wilson 1987; De Jong 1988; De Jong, Spears, & Gordon 1993). Learning weighted prototypes is to find a set of prototypes in a given instance space and their attribute weights. A prototype is an instance, but may not be a training instance. The number of subsets of an instance space is huge. The search space becomes much larger after attribute weights are used. Therefore, we choose a genetic algorithms approach to the task of learning weighted prototypes.

In this paper, we discuss some preliminary work on a genetic algorithms approach to learning weighted prototypes. In this approach, a concept is represented as one or more weighted prototypes, each of which is a conjunction of weighted attribute values. In this approach, every prototype maintains its own attribute weights. A genetic algorithm is applied to generate prototypes and their attribute weights. This approach has been implemented in GABWPL (Genetic Algorithm Based Weighted Prototype Learning) and empirically evaluated on several artificial datasets.

Michalski (1994) defined a multistrategy learning system as a learning system that integrates two or more inferential and/or computational strategies. According this definition, GABWPL is a multistrategy learning system because it combines a genetic algorithm approach with a prototype learning approach.

## 2. Previous work

Recently, there have been some attempts at creating systems for learning prototypes in machine learning (de la Maza 1991; Zhang 1992; Datta & Kibler 1995). This section discusses the work in applying genetic algorithms to learning prototypes.

The work done by Sen & Knight (1995) is the most closely related to the work reported in this paper. Sen and Knight implemented a genetic algorithm based prototype learning system PLEASE. In PLEASE, a concept is represented by one or more prototypes and a prototype is a set of attribute values. Each population structure represents descriptions of all possible concepts and consists of one or more prototypes belonging to each of the possible concepts. PLEASE does not use the bit-string representation, instead it uses real attribute values. Three operators (*mutation*, *creep*, and *two point crossover*) are used to evolve population structures. PLEASE was run on a set of artificial datasets and achieved higher classification accuracy than C4.5 on these datasets.

The work reported in this paper improves PLEASE with several extensions. First, the attribute values of a prototype are weighted in our work: Each prototype maintains its own set of attribute weights. The attribute weights of a prototype are learned with the prototype. Second, we use a different set of operators: *crossover*, *mutation*, *addition*, and *deletion*. The first two operators are similar to those used in PLEASE and addition and deletion are new operators. Addition adds a new prototype to a population structure, while deletion deletes a prototype from a population structure. Third, we use a different fitness function which takes not only the classification accuracy on training set but also the number of prototypes into consideration. The population structure with fewer prototypes is preferred over the one with more prototypes. In PLEASE, the number of prototype is limited by a user input parameter.

Skalak (1993) describes an application of the random mutation hill climbing algorithm to prototype learning. In Skalak's method, the number of prototypes is fixed and determined by a parameter $m$. All prototypes are selected from training instances. In his method, Skalak applies random mutation hill climbing to select the best combination of $m$ prototypes, which maximizes the classification accuracy on the training set. Skalak also applies random mutation hill climbing to select attributes. Experiments conducted by Skalak show that significant reduction in storage of instances can be achieved by his simple search method without decreasing classification accuracy on the selected datasets.

Kelly and Davis (1991) apply genetic algorithm technique to learn real-valued attribute weights for a simple instance-based learning algorithm. In their method, attributes weights are the same in the entire instance space. They propose to maintain a different set of weights for each concept.

## 3. Learning and Classification with Weighted Prototypes

Given $a$ attributes, a weighted prototype $P$ is a vector of $a$ pairs ($v_i$ $w_i$) plus its class membership, where $v_i$ is $P$'s value of the ith attribute and $w_i$ is $P$'s weight of the ith attribute taking a value between 0 and 1. A different weighted prototype has a set of different attribute weights. Given a set of classified training instances, the task of learning weighted prototypes is to generate a minimum number of weighted prototypes, which maximizes the classification accuracy on the training instances. At least one weighted prototype is generated for each class.

A novel instance is classified to the class of a weighted prototype which is the closest to the novel instance among all weighted prototypes. The distance metric used in our work is the weighted Euclidean distance metric, in which the distance between an instance $I = (x_1, ..., x_a)$ and a weighted prototype $P = ((v_1\ w_1), ..., (v_a\ w_a))$ is

$$\sqrt{\sum_{i=1}^{a} w_i\ (x_i - v_i)^2}\ ,$$

Zhang and Yang (1996) show that the decision boundary between two weighted prototypes is a quadratic hypersurface and that the decision boundary between two unweighted prototypes is a hyperplane. When all prototypes share the same set of weights, the decision boundary is still a hyperplane. Depending on the weights of the two weighted prototypes, the decision boundary between them could be an ellipse, a hyperbola, a parabola, or two line in a two dimensional instance space.

## 4. Genetic Algorithms for Learning Weighted Prototypes

In this section, we describe the genetic algorithms method which we use to learn weighted prototypes. We introduce population structures, fitness functions, and operators used in our method.

### 4.1 Population Structures

Given $n$ classes, a population structure consists of n class descriptions, each consisting of one or more weighted prototypes. In many genetic algorithms based learning systems (e.g. De Jong, Spears, & Gordon 1993), a rule is encoded as a string of binary bits. This binary bit representation is good for rule induction learning, but not for prototype or instance-based learning. In our method, a weighted prototype is represented as a string of $a$ pairs of real values, where $a$ is the number of attributes. The first value of a pair is the value of the corresponding attribute and the second value is the attribute weight and takes a value between 0 and 1.

Weighted prototypes of the same class are placed consecutively, and classes are separated by a comma. The number of classes is fixed, but the number of weighted prototypes in a class is variable. Therefore, the length of a population structure in our method is not fixed and may grow or shrink during evolution. The following is an example of a population structure in a domain with two attributes and two classes.

$v_{11}w_{11}v_{12}w_{12}v_{21}w_{21}v_{22}w_{22}$, $v_{31}w_{31}v_{32}w_{32}v_{41}w_{41}v_{42}w_{42}$

where $v_{ij}$ is the value of the jth attribute of the ith prototype and $w_{ij}$ is the weight of the jth attribute of the ith prototype. In this example, the population structure consists of four prototypes two for each class.

### 4.2 Fitness Function

A fitness function measures the goodness of a population structure during evolution. The population structure with a greater fitness value is stronger so as to get a greater survival chance. Fitness functions used in many genetic algorithms based learning systems (e.g., De Jong, Spears, & Gordon 1993; Kelly & Davis 1991) use only classification accuracy. In our fitness function, we take both classification accuracy and the number of prototypes into consideration. We prefer the population structure with high classification accuracy and a small number of prototypes. Too many prototypes may cause overfiting. Namely, they may perform well on training data but not on test data.

The fitness function used in our method is:

$$f(x,y) = \frac{x}{C+y},$$

where $x$ is the percentage of correctly classified instances, $y$ is the number of prototypes in population structure, and $C$ is a nonnegative real value which controls the tradeoff between the classification accuracy and the number of weighted prototypes. When $C$ is 0, every time the number of prototypes increases by 1, the classification accuracy must increase by more than the average percentage of classification accuracy per prototype in order to get a larger fitness value. A larger $C$ means that when a new prototype is added to a population structure, a less increase in classification accuracy is required in order to get a larger fitness value. $C$ is dynamically adjusted during evolution.

### 4.3 Operators

Four operators are used in our method to evolve population structures. They are *crossover*, *mutation*, *addition*, and *deletion*. The first two operators are often used in almost all genetic algorithms methods. *Addition* and *deletion* are specially designed operators to change the length of a population structure.

A two point crossover operator is applied in our method. The crossover operator is executed as follows. The two crossover points are first randomly selected on the parent with a shorter length. The two crossover points on the second parent are the same as those on the first parent. The two offsprings have the same lengths as the two parents respectively. The probability of crossover is determined by the parameter $P_{cross}$. Table 1 shows an example that demonstrates an application of the crossover operator in a two class and two attribute domain.

Table 1. Example of an application of the crossover operator

| | |
|---|---|
| Parent #1: | $v_{11}^1 w_{11}^1 v_{12}^1 \mid w_{12}^1 v_{21}^1 w_{21}^1 v_{22}^1 w_{22}^1, v_{31}^1 w_{31}^1 v_{32}^1 w_{32}^1 v_{41}^1 w_{41}^1 \mid v_{42}^1 w_{42}^1$ |
| Parent #2: | $v_{11}^2 w_{11}^2 v_{12}^2 \mid w_{12}^2 v_{21}^2 w_{21}^2 v_{22}^2 w_{22}^2 v_{31}^2 w_{31}^2 v_{32}^2 w_{32}^2, v_{41}^2 w_{41}^2 \mid v_{42}^2 w_{42}^2 v_{51}^2 w_{51}^2 v_{52}^2 w_{52}^2$ |
| Offspring #1: | $v_{11}^1 w_{11}^1 v_{12}^1 \mid w_{12}^2 v_{21}^2 w_{21}^2 v_{22}^1 w_{22}^2 v_{31}^2 w_{31}^2 v_{32}^2 w_{32}^2, v_{41}^2 w_{41}^2 \mid v_{42}^1 w_{42}^1$ |
| Offspring #2: | $v_{11}^2 w_{11}^2 v_{12}^2 \mid w_{12}^1 v_{21}^1 w_{21}^1 v_{22}^1 w_{22}^1 v_{31}^1 w_{31}^1 v_{32}^1 w_{32}^1, v_{41}^1 w_{41}^1 \mid v_{42}^2 w_{42}^2 v_{51}^2 w_{51}^2 v_{52}^2 w_{52}^2$ |

The first parent has four prototypes two for each class, and the second parent has five prototypes three for the first class and two for the second class. The first offspring has four prototypes two for each class, and the second offspring has five prototypes three for the first class and two for the second class.

Mutation changes the value or weight of an attribute of a prototype, which is randomly selected. The amount changed is dependent on the ratio of the number of correctly classified instances to the number of incorrectly classified instances. The higher the ratio is, the smaller the amount changed is. The change may be increase or decrease determined randomly. The probability of mutation is determined by the parameter $P_{mut}$.

Addition adds a training instance to a class of a population structure as a new prototype. In a population structure, the new prototype is added to the class with the most error omission (the number of training instances belonging to the class, but misclassified to other classes). The training instance with the largest distance to the class is added as the new prototype of the class and all weights are initialized to 0.5. The probability of addition is determined by the parameter $P_{add}$.

Deletion deletes a prototype from a class of a population structure. In a population structure, the new prototype from the class with the most error commission (the number of training instances belonging to other classes, but misclassified to this class). The prototype to be deleted is randomly selected. The probability of addition is determined by the parameter $P_{del}$.

## 5. Experiments

The method discussed in Section 4 was implemented in a system called in GABWPL (Genetic Algorithm Based Weighted Prototype Learning). In this section, we report the experimental results on eight artificial datasets.

### 5.1 Test Problems And Experiments

All eight problems are defined in a two dimensional space. The two attributes are numeric attributes with values ranged from 0 to 1. All problems contain two classes: class 0 and class 1. The first four problems are the same as those used in (Sen & Knight 1995). In these four problems, decision regions are separated by one or more lines. They are called 1/1 line, 1/2 line, 1/3 line, and 1/4 line respectively. Figure 1 shows these four problems.

Next four problems are similar to the first four problems, but their regions are separated by arcs not lines. These four problems are more complex than the first four problems. The are called 1/1 arc, 1/2 arc, 1/3 arc, and 1/4 arc respectively. Figure 2 shows the four arc problems.

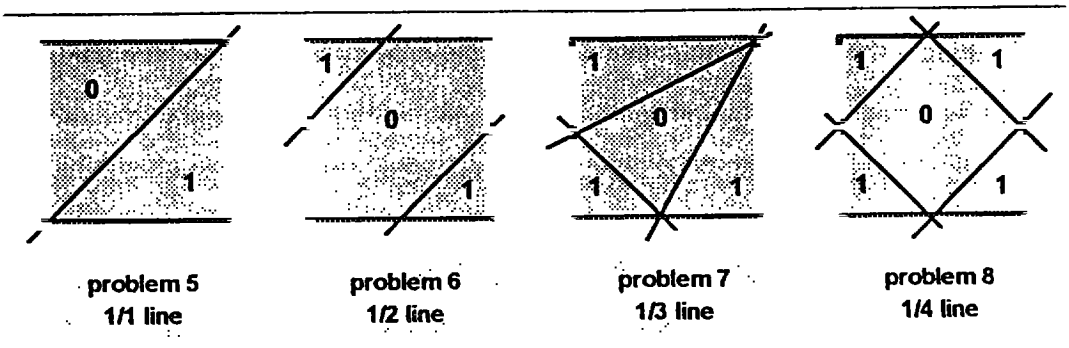


Figure 1. Four line problems

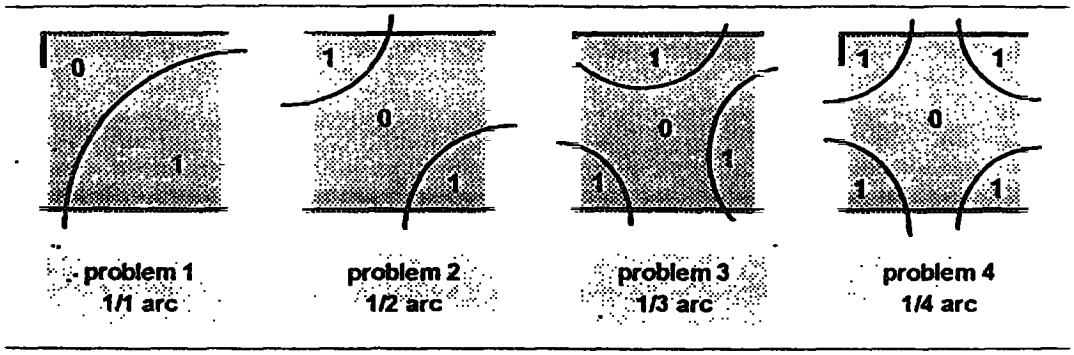


Figure 2. Four arc problems

In all experiments, 800 training instances and 200 test instances were used. All instances were randomly generated and the training and test datasets are disjoint. All experiments were repeated ten times on different set of training and test datasets. We ran two versions of GABWPL, one with no attribute weights and one with attribute weights. Our hypotheses are that GABWPL with weights should attain higher classification accuracy than GABWPL with no weight on the four arc problems while GABWPL with no weight may attain higher classification than GABWPL with weights on the four line problems. The decision boundary between two weighted prototypes is an arc while the decision boundary between two unweighted prototypes is a line (Zhang & Yang 1996). With some weight values, the decision boundary between two weighted prototypes becomes a line.

In all experiments, $P_{cross}$ was set to 0.6, $P_{mut}$ was set to 0.05, $P_{add}$ was set to 0.2, and $P_{del}$ was set to 0.15. The population size was 40 and the number of generations was 1200. Actually, experiments on all problems except for 1/3 arc and 1/4 arc converged before generation 200 and experiments on 1/3 arc and 1/4 arc converged around generation 400.

## 5.2 Experimental Results

All results reported in this section are the average of ten runs. Table 2 reports the classification accuracy on both training and test datasets. As we expect, GABWPL with weights attains about 2% higher classification accuracy than GABWPL with no weight on the four arc problems. GABWPL with weights achieves about the same classification accuracy as GABWPL with no weight on 1/1 line and 1/2 line and about 2% lower classification accuracy than GABWPL with no weight on 1/3 line and 1/4 line.

Table 3 shows the average number of prototypes generated. On all arc problems, fewer weighted prototypes were generated than unweighted prototypes, because an arc needs to be approximated by more than one line segments. Figure 3 and 4 show where generated prototypes were located on the line and arc problems, respectively. The two numbers around a weighted prototype are the two weights of the weighted prototype.

Table 2. Average Classification accuracy

| Test Problems | Training Set | | Test Set | |
|---|---|---|---|---|
| | No weight | With Weight | No weight | With Weight |
| 1/1 line | 100% | 100% | 99.5% | 99.5% |
| 1/2 line | 99.5% | 99.5% | 99.7% | 99.1% |
| 1/3 line | 99.4% | 97.1% | 98.5% | 96.3% |
| 1/4 line | 98.3% | 97.1% | 97.8% | 95.1% |
| 1/1 arc | 97.9% | 100% | 96.9% | 99.8% |
| 1/2 arc | 97.3% | 98.9% | 96.9% | 98.5% |
| 1/3 arc | 93.8% | 96.0% | 90.8% | 92.9% |
| 1/4 arc | 97.5% | 98.2% | 95.7% | 97.3% |

Table 3. Average number of prototypes generated

| Test Problems | No weight | With weight |
|---|---|---|
| 1/1 line | 2 | 2 |
| 1/2 line | 3 | 3 |
| 1/3 line | 4 | 3.6 |
| 1/4 line | 5 | 4.2 |
| 1/1 arc | 3 | 2 |
| 1/2 arc | 3.3 | 3.1 |
| 1/3 arc | 5.4 | 4.5 |
| 1/4 arc | 7.6 | 5.5 |

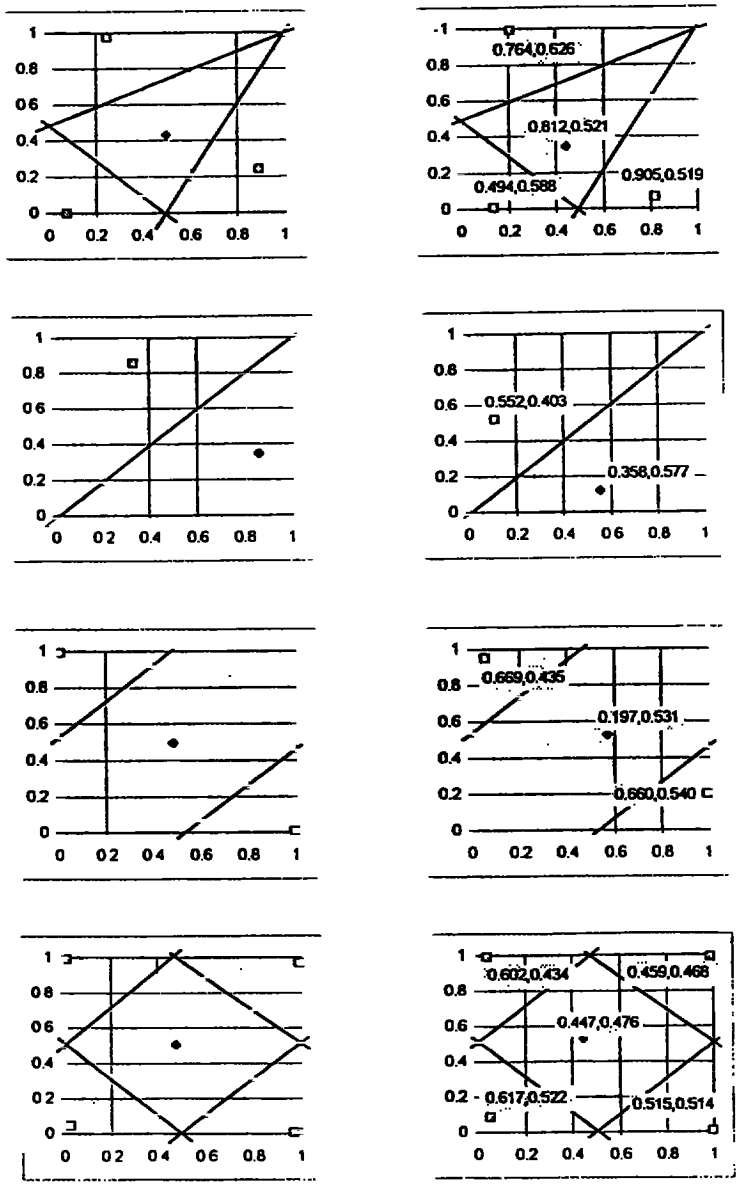


Figure 3. Prototypes generated for the line problems

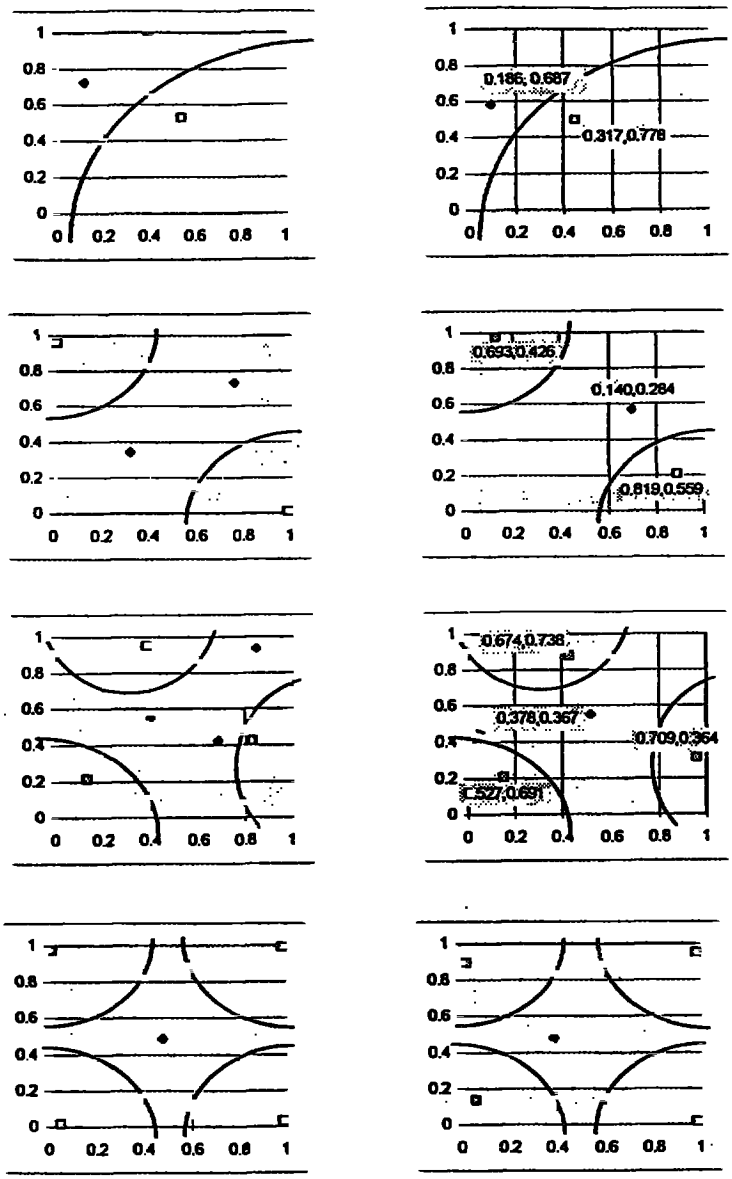


Figure 4. Prototypes generated for the arc problems

## 6. Conclusion and Future Work

In this paper, we have described a multistrategy learning system that combines a genetic algorithm approach with a prototype learning approach. The major novelty of this approach is to apply genetic algorithm to learn attribute weights and prototypes together. Our experimental results show that this approach works well on some simple artificial problems. Other contributions include the use of operators *addition* and *deletion* and a fitness function which takes the number of prototypes into consideration.

The work reported in this paper is preliminary. In the future, a number of problems need to be addressed. First, more experiments need to be conducted. We need to run GABWPL on more difficult problems: problems with more attributes and problems with irrelevant attributes. We need to apply GABWPL to some real world applications. And also we need to compare GABWPL with other learning methods, particularly instance-based learning methods.

Second, we will explore how domain knowledge (knowledge extracted from training instances) can be used in our method to speed up the search for the best descriptions. Use of domain knowledge may be very important for complex problems.

Third, the current version of GABWPL works only for numeric attributes and it should be extended to symbolic attributes. Fourth, the fitness function needs to adjusted. Finally, more operators should be investigated.

## References


Aha, D. and Kibler, D. 1989. "Noise-tolerant instance-based learning algorithms," Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit, MI..

Aha, D.W. & Goldstone, R.L. 1992. "Concept learning and flexible weighting," Proceedings of Fourteenth Annual Conference of the Cognitive Science Society.

Aha, D., Kibler, D, and Albert, M. 1991. "Instance-Based Learning Algorithm," *Machine Learning* 6.

Bareiss, E. R., Porter, B. W., and Wier, C. C. 1990. "Protos: An Exemplar-Based Learning Apprentice," in Kodratoff & Michalski (Eds), *Machine Learning: An Artificial Intelligence Approach V III*, San Mateo, CA: Morgan Kaufmann

Barsalou, L. 1985. "Ideals, central tendency, and frequency of instantiation as determinants of graded structure in categories," in Journal of Experimental Psychology: Learning, Memory and Cognition, 11.

Cost, S., and Salzberg, S. 1991. "Q weighted nearest neighbor algorithm for learning with symbolic features," Technique report, Department of Computer Science, The Johns Hopkins University

Datta, P., & Kibler, D. 1995. "Learning prototypical concept descriptions," Proceedings of the 12th International Conference on Machine Learning

Dejong, K.A. 1988. "Learning with genetic algorithms: an overview," *Machine Learning* 3.

Dejong, K.A., Spears, W.M., & Gordon, D.F. 1993. "Using genetic algorithms for concept learning," *Machine Learning* 13.

de la Maza, J. 1991. "A prototype based symbolic concept learning system," Proceedings of the 8th International Workshop on Machine Learning.

Kelly, J.D., & Davis, L. 1991. "A hybrid genetic algorithm for classification," Proceedings of the 12th International Joint Conference on Artificial Intelligence.

Michalski, R.S. 1994. "Inferential theory of learning," in Michalski & Tecuci (Eds), *Machine Learning: A Multistrategy Approach*, Vol. IV, San Mateo, CA: Morgan Kaufmann.

Salzberg, S. 1991. "A nearest hyperrectangle learning method," Machine Learning, 6:3.

Sen, S., & Knight, L. 1995 "A genetic prototype learner," Proceedings of the 14th International Joint Conference on Artificial Intelligence.

Skalak, D.B. 1994. "Prototype and feature selection by sampling and random mutation hill climbing algorithms," Proceedings of the 11th International Conference on Machine Learning.

Smith, E. E., & Medin, D. L. 1981. *Categories and Concepts*. Harvard University Press.

Smith, E.E., & Osherson, D.N. 1984. "Conceptual combination with prototype concepts. *Cognitive Science* 9.

Stanfill, C. and Waltz, D. 1986. "Toward memory-based reasoning," Communications of the ACM, 29:12.

Wilson, S.W. 1987. "Classifier systems and the animate problem," *Machine Learning* 2.

Zhang, J. 1992. "Selecting Typical Instances in Instance-Based Learning," Proceedings of the Ninth International Conference on Machine Learning.

Zhang, J., & Yang, J. 1996. "The role of attribute weights in instance-based learning," Unpublished manuscript.