Abstract

In this paper we demonstrate how the traditional Genetic Algorithm can be augmented by incorporating domain knowledge in the form of a Version Space into the structure. This hybrid inductive learning system is designed to handle problems in concept learning using the Version Space to control the search process that is performed by the Genetic Algorithm. In this hybrid system a new class of schemata is present called hyperschema. A theorem for hyperschema analogous to that for traditional schema is presented. This theorem demonstrates how the addition of domain knowledge in the form of a Version Space allow the hybrid system to exploit schemata of higher order and defining length via a hitchhiking effect. A prototype program that implements this approach is briefly described in pseudocode and the actual system is used to solve a problem in concept learning posed by Nilsson [13].

1. Introduction

As originally proposed by Holland [9] genetic algorithms make few assumptions about the problem solving domain. As such they can be classified as weak problem solving methods since their activities are weakly tied to the background knowledge. However, there are points within the basic structure where problem specific background knowledge can be used to improve the search. In figure 1, an outline of the traditional structure for the Genetic Algorithm is given.

Various researchers have taken on the task of integrating domain knowledge into the algorithm at particular points in the process. Among them, Grefenstette investigated the use of background knowledge to initialize the population [7]. Davis [2], and Fourman [4] focused on the evaluation of chromosomal structures. Reynolds [14] employed domain specific operators in a model of the cultural evolution process. Grefenstette also explored the use of domain specific recombination operators in terms of the traveling salesman problem [7]. In that paper he suggests a heuristic mutation operator to perform local search, once a high performance region has been identified. He observed that, "...perhaps the optimal arrangement is to use a Genetic Algorithm as a preprocessor for one of the many domain specific local search techniques for the Traveling Salesman Problem" [7].

In this paper the focus is on the incorporation of domain knowledge into procedure SELECT in figure 1. This domain specific approach essentially uses the traditional Genetic Algorithm as a preprocessor to set the stage for its more domain-specific inductive learning activities as suggested by Grefenstette. Therefore, it is assumed that the Genetic Algorithm has already been used as an exploratory tool to identify subspaces of high performing individuals. Within each subspace we will want to identify individuals exhibiting the highest performance and in the process produce a general description of those individuals in terms of the basic concepts relevant to the problem at hand. Such localized search can pose problems for traditional genetic algorithms. This is because genetic algorithms may lack the domain knowledge
required to search the local space efficiently. That is, the power of genetic algorithms lies in their ability to implicitly allocate an increasing number of trials to equivalence classes of structures exhibiting above average performance [9]. These equivalence classes are called schemata [7] and for structures over a binary alphabet they are expressed as a sequence of binary digits interspersed with don't care symbols, #. An example is 1101###1. Such a schema can be described in terms of its order and defining length [7]. The order, o, of a schema is the number of positions that are fixed by a 0 or a 1. In this case the order is five. The defining length of a schema, dlen, is calculated by subtracting the position indices for the left and rightmost fixed positions. Here the defining length is 8-1 or 7.

Procedure GA;
begin
initialize population POP(0);
evaluate POP(0);
t=1;
repeat
select POP(t) from POP(t-1);
recombine POP(t);
evaluate POP(t);
until (termination condition);
end.

Figure 1. Pseudocode Outline of the Genetic Algorithm

These properties can be used to describe the likelihood that a schema will contribute to the population in the next time step in terms of the Schema Theorem given below [5]:

\[ m(H, t+1) \geq m(H, 0)^f(H) \cdot \left( 1 - \frac{dlen(H)}{len} - p_m \right) \]

For a given schema H, the number of instances of it in the next population, m(H, t+1), will be a function of the current number of instances, its fitness relative to the average fitness of the current population, and the likelihood that its structure will be disrupted by the actions of the two genetic operators. This assumes a selection function that employs fitness proportional reproduction in conjunction with the operations of crossover and mutation.

Goldberg's [6] summary of Holland's [9] schema theorem says that above average, short, low-order schemata are the ones given increased trials in successive generations. However, during the localization phase of the search process the number of relevant schemata can be drastically reduced, and it is likely that the most relevant schemata are ones of high order and long defining length. These will certainly be the ones that have not been well sampled up to this point according to the schema theorem. This will result in a noticeable degradation of the power of genetic algorithms in this situation since, in order to keep these schemata around, the mutation and crossover probabilities may have to be reduced.

This paper poses a modification to the genetic algorithm in order to upgrade the search performance when only a few bits of a schema remain to be determined. The approach taken is to: 1) identify schema of above average performance, 2) group these schema into equivalence classes based on knowledge present in the version space (these equivalence classes are called hyperschemata), 3) the Vote Inherit Promote (V.I.P.) cycle is employed to incorporate knowledge of hyperschemata performance into the GA's behavior.

In section 2 the basic approach to integrating the two inductive learning schemes is described. The capabilities of the approach are demonstrated using a prototype Lisp system on a problem in Concept Learning suggested by Nilsson [13]. In section 3 a theorem for hyperschemata, analogous to Holland's schema theorem is described. This is used to suggest that additional domain knowledge provided by version spaces is sufficient to provide a local search that retains the power of the original Genetic Algorithm.

2. The Use of Hyperschema to Control the Genetic Algorithm Search Process

The goal of this section is to introduce into the Genetic Algorithm framework explicit information about schemata that will be relevant to the local search process. Specifically we wish to produce a maximal generalization in terms of those domain concepts that affect a schema's performance. Since version spaces have been
demonstrated to be a useful tool in other data
-driven generalization programs (e.g., LEX,
META-DENDRAL) [11,12] it was decided to
employ them here.

Let us assume a two-dimensional grid
decomposed into cells of uniform size.
Associated with each cell is a function
(unknown) from a given domain into real
numbers. The goal facing the Genetic Algorithm
system is two-fold. First, to isolate the cells
associated with those functions whose average
payoff is above a certain level. Second, to
produce a conceptual description of those domain
objects associated with above average
performance in a given cell. Since the first task
is more in keeping with traditional uses of
Genetic Algorithms, we will focus on the second
one here. The Genetic Algorithm is then given
knowledge about the function in terms of a
version space or spaces, as well as a history of
previous experiments performed on the cell.
Adjustments can be made to the Genetic
Algorithm in order to access this information.
The nature of these adjustments is best motivated
with an example.

In our example, it is be assumed that the function
to be investigated is over a standard bridge deck
of 52 playing cards. An experiment consists of
picking a card by specifying its suit and face
value. Its associated real value is returned by an
oracle. This problem is described in more detail
in [13].

The task of describing cards of above average
performance for the current function can be
factored into two subtasks. This is because the
face value of a card is descriptively independent
of the card's suit and visa versa. Thus, each
subtask is described using its own version space.
The version graphs for the two factors are given
in figure 2. Both graphs are considered
taxonomies since the following properties hold:
1) each child is a special case of its parent, 2)
there are no instances in common between nodes
at a given level, and 3) the child nodes exhaust
all of the possibilities associated with the parent
node [10].

The objective here is to isolate the most general
concept in each graph associated with above
average cards. A given concept cannot be
associated with above average performance
unless at least one of its child exhibits above
average performance. Thus, the hypothesis that a
given concept, for example ALLRED, is above
average corresponds to the hypothesis that there
exists at least one path from a leaf node to
ALLRED where all of the nodes on that path are
all above average. Therefore, the most general
hypothesis for a given version graph will be
associated with path(s) of consecutive above
average nodes beginning with a leaf node and
ending with a node(s) of highest generality.
Although this problem pertains to the production
of a maximal generalization, the system can be
easily extended to deal with other categories of
generalization problems such as the production
of a maximally specific generalization.

Information about this path can be accumulated
by the Genetic Algorithm in the following way.
It was mentioned previously that the power of
Genetic Algorithms corresponds to their ability to
allocate increasing trials to high performance
schemata. In our example, testing the eight of
diamonds is equivalent to acquiring information
about the following schema, ["8" "] and ["D"
"D"]. However, in the context of the domain
knowledge present in the version space, the card
can be considered as an instance of a number of
paths through each of the factor spaces. The
paths in this case are "DIAMOND" "DIAMOND-
ALLRED" and "DIAMOND-ALLRED-
ALLSUITS". In fact, each of these can be
expressed in schema-like terms as follows:
"DIAMOND-#" "DIAMOND-ALLRED-
ALLSUITS". These schema are called hyperschema
because each of the nodes that is part of the path is indeed a
schema in the traditional sense. A hyperschema
is a structured collection of nodes (schema)
connected by links which is analogous to the
definition of hypertext in database systems [15].

What must be done now is to modify the
selection algorithm for the Genetic Algorithm
in order to allow the system to search for high
performing hyperschema. In order to do this we
must investigate the structure of the traditional
selection phase for Genetic Algorithms. This
phase consists of two distinct algorithms, the
selection algorithm and the sampling algorithm
[9]. The task of the selection algorithm is to
assign to each individual \( x \) in the population, at time \( t \), a real number that represents its target sampling rate, \( \text{tsr}(x, t) \). This number corresponds to the expected number of offspring to be generated. The sampling algorithm maps this real value into an integral number of copies. Since this latter algorithm will not be affected by our suggested changes we will not discuss it further here. We will assume that an optimal sampling algorithm such as that proposed by Baker [11] will be used.

The number of trials that are allocated to schemata in future populations is directly determined by the selection algorithm which operates on the set of individuals in the current population. The most widely employed technique is based upon the principle of proportional selection. This was the approach taken originally by Holland and was predicated on the Darwinian notion of survival of the fittest. Here, the \( \text{tsr}(x, t) \) is determined by the ratio of the payoff for the individual, \( u(x) \), to the average for the population, \( u(t) \), as follows:

\[
\text{tsr}(x, t) = \frac{u(x)}{u(t)}
\]

This induces a target sampling rate for a hyperplane, \( H \), associated with a given schema which is also proportional to the average instances of that schema in the population at time \( t \) as given below [9]:

\[
\text{tsr}(x, t) = \frac{u(H)}{u(t)}
\]

Now we want to establish a similar relationship for hyperschemata. This is done by incorporating into the payoff function performance information about the existence of paths of consecutive above average nodes from leaf nodes to nodes of higher generality. This relationship is expressed in the following equation:

\[
\text{tsr}(x, t) = \frac{u(x, VS)}{u(t)}
\]

Simply stated, the sampling rate for \( x \) is a function of both its structure and that of the domain knowledge. In this case, the domain knowledge is the version space, \( VS(t) \). \( VS(t) \) corresponds to the current state of the version space annotated with relevant performance information at each node.

The payoff function can be described as a composition of two functions, an objective function, \( f(x, VS) \), and a transform function, \( g \), as given below [3]:

\[
u(x, t) = g(f(x, VS))
\]

The objective function computes the raw score for the individual and \( g \) adjusts that score to fit within a given range of positive fitness values. The function \( g \) is necessary when the goal is to minimize the objective function or when the objective function can produce negative values. The transformation associated with \( g \) is frequently linear with the form:

\[
u(x, t) = (a \ast f(x, VS)) + c
\]

Let us decompose \( f \) into two functions \( f' \) and \( f'' \). \( f' \) assesses the performance of an individual. \( f'' \) assesses the performance of paths in the version space related to and individual.

\[
u(x, t) = (a \ast f(x)) \ast (b \ast f'(VS, x)) + c
\]

One of the problems with the Genetic Algorithms is the possibility of premature convergence due to the predominance of locally above average structures. Dynamic scaling of the performance for each structure is frequently used to reduce the relative performance in order to force the continued modification of the current population of structures. With this approach, the constant, \( c \), is replaced by a dynamic function of the population at time \( t \).

\[
u(x, t) = ((a \ast f(x)) + (b \ast f'(VS, x))) + c(POP, VS)
\]

When the version space can be factored into \( n \) components, \( f' \) can be expressed as follows:

\[
f'(VS, x) = \sum_{i=1}^{n} f'(VS(i), x)
\]

In our example, \( f'(x) \) is the unknown card mapping function while \( f'' \) is expressed as:
\( f'(\text{SUIT},x) + f'(\text{RANK},x) \)

\( f' \) can be implemented in a variety of ways. The approach selected here is called the V.I.P. method based upon its three phases: Promote, Inherit, and Vote. The actions performed in each phase are given below.

1. The Promote Phase. In this phase the value associated with \( f(x) \) is used to update the performance of all concepts in the version space that contain \( x \). This is equivalent to updating all nodes on a path originating from \( x \) in each of the factors in the version space. In our example, the performance of interest at each node is the average of those experiments (cards) that are members of the conceptual relation represented by the node. For the eight of diamonds, its performance contributes to the single path, \( \text{DIAMOND.ALLRED.ALLSUITS} \) in the SUIT taxonomy.

2. The Inherit Phase. The inherit process is used to propagate performance information from a parent node to any child node that has yet to have associated with it any performance information. This information can be used to calculate the scaling function, \( c \), at time \( t \). Traditionally scaling is done by adjusting performance downward (in maximization problems). One common approach is to make \( c(t) \) equal to \(- \[ \min[I(x) \mid x \in \text{POP}(t)] \]\) in order to adjust \( f' \) for the GA population. However, here we must also be concerned with early convergence relative to paths in the version space of a higher performance. Therefore, some adjustment relative to \( f' (\text{VS},x) \) must be made as well. This can be done on the basis of the following analogy. The traditional reduction of performance can be viewed as reflecting the non-monotonic inference that no current individual represents a solution to the problem. This is called the open-world assumption. Thus, the solution is the complement of the current set, thus setting up a continuation of the search.

A similar non-monotonic reasoning activity is used to provide scaling information for paths in the version graph. While there are many possibilities, the approach taken is to select the minimum default value inherited by some child in the version graph, \( \min[\text{INHER}(x) \mid x \in \text{VS}] \).

3. The Vote Process. This portion of the cycle takes the votes associated with nodes along the path from that structure in the version space. Any node whose performance satisfies the performance criterion is given a positive vote. The vote can be weighted by a value reflecting a relationship between the voting node and the chromosome \( x \) for which it is voting. Possibilities for this weight therefore include the depth (maximum path length to chromosome \( x \)) and breadth (number of nodes) of the subgraph for which the voting node is the root. For maximization problems such as the card problem, the weight for a node can reflect its maximum distance from a leaf node.

Votes are taken only from active nodes in the version space. The approach taken to compute the active region is predicated on the assumption that every path has both a maximal element (most general node) and a minimal element (most specific). This will be true for any version space over a finite universe [13]. For a well-structured version space we specify the active region as the set of nodes or relations with above average performances. In a maximization problem, boundary adjustments can be done in the following fashion. For any given path from a leaf to a root there may exist a non-null subsequence of above average nodes beginning with the leaf node. The set of nodes in that subsequence constitutes the active region for that path, with the most general node constituting the boundary of that path. The union of all active regions associated with paths in the version space gives the overall active region for the version graph. The corresponding set of boundary nodes is the current set of maximally general elements in each path. Candidate nodes are eliminated from the boundary set in two ways. First, a node is excluded when its cumulative average is lower than the average by at least a certain amount. Second, a node is excluded if the
average for all experiments of any of its immediate children falls below the current average by at least a certain amount. For example, the average performance for the root will always equal the global average but the performance of one of its children may be far below it.

```
Procedure VGA
begin
  t=0;
  Read in Number.of.Factors;
  Initialize POP(t);
  Initialize Boundary.set(t);
  For i=1 to Number.of.Factors do
    Initialize Graph(Factor(i));
  Evaluate Structures in POP(t);
  For i=1 to Number.of.Factors do
    VIP(POP(t),Graph(Factor(i)));
  While Vote.change.any.factor do
    begin
      t=t+1;
      Select POP(t) from POP(t-1);
      Recombine structures in POP(t);
      Evaluate structures in POP(t);
      For i=1 to Number.of.Factors do
        VIP(POP(t),Graph(factor(i)),); Update Boundary
    end
end.
```

Figure 3. The Version Space Controlled Genetic Algorithm (VGA)

At this point the pseudocode for the augmented Genetic Algorithm can be presented. In figure 3 the pseudocode is given for a version graph that can be factored into a set of prime subgraphs. Note that the main difference between this and the original Genetic Algorithm is the code to initialize the respective version graphs and compute the \( f' \) function for each graph with the V.I.P. method. The system terminates when successive experiments over a certain time period (observation window) have not produced a change in votes in any of the factor graphs. This is equivalent to the lack of change in the set of maximal boundary elements for a specified period of time.

```
Initial Population Characteristics:
  population: 6
  chromosome length: 2
  max # of generations: 100
  crossover prob: 0.2
  mutation prob: 1.0
```

**Generation 2** population report:
There were 4 taxonomy generalization changes.
The maximal generalization(s) in suit
  taxonomy: (REDS)
The maximal generalization(s) in face value
  taxonomy: (ODD-NUMS)
68% of nodes have been visited thus far (13
  out of a total of 19)

**Generation 3** population report:
There were 1 taxonomy generalization changes.
The maximal generalization(s) in suit
  taxonomy: (REDS)
The maximal generalization(s) in face value
  taxonomy: (ODD-NUMS)
87% of nodes have been visited thus far (17
  out of a total of 19)

**Generation 4** population report:
There were 1 taxonomy generalization changes.
The maximal generalization(s) in suit
  taxonomy: (REDS)
The maximal generalization(s) in face value
  taxonomy: (PRIMES)
100% of nodes have been visited thus far (19
  out of a total of 19)

**Generation 5** population report:
There were 0 taxonomy generalization changes.
The maximal generalization(s) in suit
  taxonomy: (REDS)
The maximal generalization(s) in face value
  taxonomy: (PRIMES)
100% of nodes have been visited thus far (19
  out of a total of 19)

Figure 4. Input and Output Description for the VGA System.

In figure 4 the results of applying this approach to characterize an unknown Lisp function, \( f \), over the card domain is presented. Two recombination operators, mutation and crossover were used with a probability of application of 0.1 and 0.2 respectively. The initial population was a random collection of six individuals. The system went five generations until there were no
changes in the boundary set for either factor. The maximal generalization for the SUIT taxonomy was "RED", and for the FACEVALUE taxonomy it was "PRIME". Note that it took about twice as long for the system to produce the FACEVALUE generalization as it did for the SUIT. This is to be expected since the former graph is more complex than the latter. Also note the effectiveness of the combined scaling approach. After just 5 generations it had produced enough different structures to represent each of the high level concepts (percent of nodes visited).

The question now is how well do these generalizations characterize the unknown function? The function is given as: if the structure is hearts or diamonds then add 50 points to the total and if the structure is a 3, 5, or 7 then add 50 points to the total. The VGA system has characterized this function precisely in terms of those cards that are of high payoff here since only cards with "RED" SUITS and "PRIME" face values are rewarded.

3. The Hyperschema Theorem

In this section the performance of the VGA system is characterized in terms of its ability to allocate an exponentially increasing number of trials to collections of hyperschema of above average payoff in the version space. The basic principle behind this performance is the presence of a specific association between schema defined over the structures manipulated by the genetic algorithm and hyperschema defined over paths in the version space. Each structure in the population is associated with one path from a leaf node to a root in a version graph, assuming a taxonomic hierarchy for the version graph. Associated with the structure in the population is a collection of schemata. Likewise, a collection of hyperschema is associated with the path in the graph. Thus, the relationship between the structure in the population and the path in the version graph induces a relationship between their associated schema types. This relationship can be made explicit in terms of the following hyperschema theorem. This theorem basically redefines Holland's [9] original theorem in terms of these collections of hyperschema. Therefore, proportional selection, a payoff function that equals the objective function, and the use of the mutation and crossover operators are assumed for this theorem. In addition, just a single taxonomic hierarchy is assumed for the present.

**Hyperschema Theorem**

\[ m(H, t+1 | HS \in PATHS(H, t+1)) \geq \\
\frac{m(H, t | HS \in PATHS(H, t)) \times avg(f''(H, t) | HS \in PATHS(H, t))}{f''} \times \\
\left[ 1 - [p_m \times avg(o(H) | HS \in PATHS(H, t)) - \\
p_c \times \frac{avg(dlen(H) | HS \in PATHS(H, t))}{kn - 1}] \right] \]

The theorem expresses the relationship between a given hyperschema, HS, and the number of schema in the current GA population. \( m(H, t | HS \in PATHS(H, t)) \) corresponds to the number of schema, \( H \), in the current population that contribute performance information to a path in the version graph that has HS as a hyperschema. The number of schema associated with HS in the next time step is proportional to the relative payoff of those schemata. The relative performance is expressed in terms of two functions, the objective function \( f \), and the voting function, \( f'' \).

\[ \frac{avg(f'(H, t) | HS \in PATHS(H, t))}{f'} \]

reflects the performance of the set of schema associated with our given hyperschema, HS, for the objective function \( f \) relative to the performance of all schemata in the current population.

\[ \frac{avg(f''(H, t) | HS \in PATHS(H, t))}{f''} \]

reflects the performance of the set of schema associated with hyperschema HS in terms of the voting function \( f'' \) relative to the average
weighted vote received by all schema in the current population.

The actual number of schema corresponding to a collection of hyperschema will be reduced by the action of mutation and crossover as in Holland's original theorem. However, note that since we are dealing with sets of schema our concern is with the average defining length and order for those schema associated with HS. Therefore, schema of higher order and longer defining length can be carried along with a collection of schema that are of low order and defining length on the average when that collection corresponds to a HS exhibiting above average performance. Thus, a high performance hyperschemata in the version graph is able to carry along with it a population of schema, some of which can be of high order and long defining length as long as the average order and defining length for the collection is low. This is called the hitchhiking effect. The power of the version space lies in its ability to retain more detailed schemata in the population as a basis for guiding the search process. In fact, the original schema theorem can be viewed as a special case of our hyperschema theorem where the version graph is simply a set of unconnected nodes, each node corresponding to a simple schemata. In that case, each HS corresponds to a unique schemata.

4. Conclusion

The success of a hybrid learning system can be measured in terms of the symbiotic nature of the interaction between its components. That is, in terms of the benefits which accrue to each participant in the combined system. In this paper, the nature of the symbiosis was highlighted. In terms of a version spaces, the presence of a GA component allowed the system to perform the concept learning activity in an exploratory and autonomous fashion without the need for a predefined set of examples provided by a teacher. From the perspective of a GA system, the presence of domain knowledge in the form of a version space allowed each trial of a schema in the GA population to represent more general classes of structures, hyperschema. The result of this relationship, as expressed in the hyperschema theorem, is to allow the system to retain schema of higher order and longer defining length. Thus, the symbiotic nature of the interaction produces a hybrid system that enhances the learning capabilities of its components.

5. References


Facevalue Taxonomy

Suits Taxonomy

Figure 2. Combined Taxonomies for the Card Problem.