The Use of Version Space Controlled Genetic Algorithms to Solve the Boole Problem

Dr. Robert G. Reynolds, Jonathan I. Maletic, Shan-Ping Chang

Department of Computer Science
Wayne State University

431 State Hall
Detroit, Michigan 48202

1. Introduction

As originally proposed by Holland [5] 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.

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 [4]. Davis [1], and Fourman [2] focused on the evaluation of chromosomes. Reynolds [9] 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. 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" [4].

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 resources, an increasing number of trials to equivalence classes of structures exhibiting above average performance [5]. These equivalence classes are called schemata [4] 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 [4]. 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.

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]:

\[
m(H, t+1) \geq m(H, t) \left(1 - p_c \frac{dlen(H)}{len} - p_m o(H) \right)
\]

For a given schema \( H \), the number of instances of it in the 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 [3] summary of Holland's [5] 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 problem results from the fact that the pool of chromosomes perform a dual function. They record not only the new candidates for search but also the systems past history. Increasing the frequency of the genetic operators increases the number of new chromosomes in the population at the cost of modifying older successful ones. In order to correct this problem the history component is separated from the component describing the current state. The past history of the search process is summarized in terms of a version space which keeps a record of above average hypotheses. These hypotheses in turn are used to constrain the generation of chromosomes by the GA system. The resultant system is capable of learning at a higher rate with a smaller population of chromosomes. This new system is termed the Version
Space Guided Genetic Algorithm (VGA). The VGA has been used to solve an inference problem posed by Nilsson [7].

In this paper it is demonstrated that the VGA is a particular instantiation of a more general class of systems, termed Autonomous Learning Elements (ALE). In section 2 the basic components of an ALE are discussed. In section 3 the Boole problem posed by Wilson [11] is introduced and its expression in terms of the VGA framework discussed. The details of the VGA system are given in section 4, followed by a discussion of results in section 5. In particular, in section 5, the performance of the VGA on two versions of the Boole problem are described and the performance compared with that for classifier systems [ll] and decision trees [8].

2. Autonomous Learning Elements

An autonomous learning element (ALE) is a hybrid learning system consisting of two components. The first component, the experience generator, (EG), is an empirical learning component that learns to find examples in an instance space. The second component, the knowledge base integrator, (IG), is an analytical learning component that learns to identify relationships that pertain to the high performance examples and not to those exhibiting low performance. These relations are stored in terms of a given knowledge representation. The two components are linked by a protocol composed of three basic phases or relations. This protocol is termed the VIP or Vote-Inherit-Promote cycle. It is through this protocol that the two learning systems interact.

This cycle is illustrated graphically in figure 1. There, the current set of active examples after t cycles in the experience generator of empirical component, EG, is given as EG(t). Likewise, the current set of active relationships in the knowledge integrator, KI, at time t is KI(t). A subset of examples in EG(t) is evaluated in terms of the problem environment. The performance of each example is said to "vote" for those active relations in the KI that it represents. These active relations "inherit" the performance of those empirical experiments that represent them. This involves updating the performance for each active relation to produce a modified knowledge base, (KI(t))'. The performance of the examples in the EG are modified based upon their current performance in conjunction with the performances of those active hypothesis in (KI(t))' that they represent. That is, the performance of an example can be enhanced if it is associated with above average relations. These successful relations are said to "promote" the individual instances supporting them just as a successful candidate for public office will reward their supporters with political appointment. (EG(t))' corresponds to the state of the experience generator once its examples have been updated and promoted.

At this point, the learning process for each component is initiated. For the empirical component this involves adjusting the set of active examples based upon their performance to produce a new set, EG(t+1), for the next cycle. The learning process for the analytical components corresponds to updating the active set of relations as a result of the new empirical evidence. This produces a new set of active relations, KI(t+1), for use in the next cycle.

\[ \text{KI}(t) \xrightarrow{\text{Inherit}} (\text{KI}(t))' \xrightarrow{\text{Learning (KI)}} \text{KI}(t+1) \]

\[ \text{EG}(t) \xrightarrow{\text{Vote}} (\text{EG}(t))' \xrightarrow{\text{Promote}} \text{EG}(t+1) \]

![Figure 1. The ALE Framework.](image)

The process continues until the set of active relations in the KI are no longer effected by new evidence generated by the experience generator over a given number of cycles. At this point the contents of the KI can be examined in order to extract the conclusion produced by the empirical investigation.

A particular implementation of this learning system, the VGA, has been used to solve a card inference problem posed by Nilsson [7], a multiplexer inference problem posed by Wilson [11], and a problem in the acquisition of software engineering knowledge posed by Reynolds [lo]. In the VGA, the set of possible relationships is represented as a lattice of hypotheses. The learning activity involves systematically isolating the most general set of hypothesized relationships that are consistent with the observed data.

3. Representing Background Knowledge for the Boole Problem.

This section illustrates the mapping of background knowledge needed for the multiplexer problem into the framework of the VGA. A multiplexer is a logic circuit that accepts several binary inputs from the data lines and selects one of them as output. The selection of the
appropriate bit is a function of the address lines. Usually, a multiplexer is specified by the total number of data and address lines. A block diagram of a six input multiplexer is shown below. It has two address inputs, \( A_0 \) and \( A_1 \), and four data inputs, \( D_0 \) through \( D_3 \). In addition, there is one output line \( F_6 \) for transmitting the input signal. Each data and address input takes a binary signal. The signals on the address input are decoded as integers. The integer is used to index which data line is selected so that its signal will be sent out as output. For instance, assuming that the address inputs on the figure are \( A_0 = 0 \) and \( A_1 = 1 \), (which is an integer 2 in base 10) then the data line labeled \( D_2 \) is selected and the value of the output is equal to that of \( D_2 \).

![Block Diagram of a Six Input Multiplexer](image)

A multiplexer can be represented by a bit string. One possible representation of the six input multiplexer is shown below. The address inputs are placed at the left side of the string, the output is the rightmost bit and the data inputs are in between.

<table>
<thead>
<tr>
<th>( A_1 )</th>
<th>( A_0 )</th>
<th>( D_0 )</th>
<th>( D_1 )</th>
<th>( D_2 )</th>
<th>( D_3 )</th>
<th>( F_6 )</th>
</tr>
</thead>
</table>
| 1 0 1 0 1 0 1

The boolean function performed by a six multiplexer can be described as follows:

\[
F_6 = A_0 A_1 D_0 + A_0 A_1 D_1 + A_0 A_1 D_2 + A_0 A_1 D_3
\]

\( F_6 \) represents the output of the function for a six input multiplexer. This boolean function defines the relationship between the inputs and outputs in terms of their boolean values. That is, the function is true if and only if the selected input is true. Otherwise, the function is false. For example, the multiplexer represented by the string ‘1000101’ is true and ‘1000000’ is false.

The problem to be solved here, is to learn the boolean function performed by a multiplexer in terms of its functional characteristics. The functional characteristics of a multiplexer can be defined as follows. A logical circuit is a multiplexer if and only if the logical value of its output is equal to that of its selected input which is determined by the states of the address inputs. For example, both strings used previously satisfy the definition of the boolean function performed by a six line multiplexer. Therefore, they are considered as positive instances of the boolean function to be solved.

Suppose that a pool of binary string are presented to the learning system as input. Each string represents a potential instance of the boolean functions performed by some logical circuit. An objective function is applied to each of the strings. If the string satisfies the criteria defined by that function, a reward is given. Based on the bit values of the strings and their payoffs, the learning system is asked to describe the overall characteristics of the objective function.

Figure 2. Partial Version Graph

Figure 2 shows a portion of the version graph that represents background knowledge for the six input multiplexer problem. The nodes are the patterns and the arcs are MST (more specific than) relations. At the top level of the graph, there is a single pattern, ‘#######’. This pattern matches all of the possible instances in the application space. The children of that node at the next level define all of the possible values for the output bit of the circuit.

The next level defines a set of patterns which are MST of their parents in terms of the number of fixed values in the string. For a parent node with \( N \) number of ‘#’s, there are \( 2^N \) number of child nodes needed to cover \( N \) possible cases in which one of the ‘#’ positions is set to be either 1 or 0. This is to ensure that the taxonomical structure is not biased. By the same token, the subsequent levels of the version graph can be defined. The leaf nodes of the version graph are a set of nodes that have four fixed values in their pattern. This is because the minimum number of fixed positions required to classify a boolean function performed by a six input multiplexer is four. Notice that the space need not be extended past this point since only two address lines, one data line, and one output are needed to specify the behavior.

4. Structure of the VGA system

Given the representation of the Boolean problem as defined in the previous section, a corresponding concept
formation problem can be specified. According to Nilsson [7], a concept formation problem is defined in terms of four components \((P,N,C,L)\). \(P\) is the set of positive instances and \(N\) is the set of negative instances for the concept. \(C\) is the set of primitive concepts that can be combined to describe the required concept. \(L\) corresponds to the acceptable syntax used to combine the concepts from \(C\).

Here the basic set of primitive concepts are the address lines and the data lines. the language with which they can be combined is a subset of boolean algebra using disjunction and negation. The resultant concept is a boolean characteristic function for the multiplexer circuit.

The version space [6] for a concept formation problem is the set of all admissible relations between concepts in \(C\). A relation is admissible if it is capable of describing the set of positive examples and excluding the set of negative ones. A version graph is a directed acyclic graph in which each node is a possible admissible relation. Each arc in the graph extends from node \(x\) to node \(y\) if (1) \(x\) is less general than \(y\) and (2) there is no node \(z\) that is more general than \(x\) and less general than \(y\).

The version graph for the Boole problem given in section 3 is in fact a taxonomy. The size of the graph precludes its complete enumeration. The number of nodes in the graph for a multiplexer with two address lines is 1970, and for three address lines is 253,444. However, one need only pay attention to the set of maximal (most general) relations that do not describe any negative examples, and to the set of minimal elements that describe only positive examples. the former set of relation is called the \(G\) set, and the latter the \(S\) set. The contents of these sets are modified as new examples are presented to the system. Here we will use the version space to cache generalizations about the performance of individual chromosomes from the GA population that serve to restrict the generation of new chromosomes. Thus, this version of the VGA will be concerned with updating the \(G\) set only. Only chromosomes that are consistent with admissible relations present in the \(G\) set will be generated. Each node in the graph is labeled by a performance value that summarizes the past performance of all generated examples (chromosomes) that represent it. A relation remains in the \(G\) set as long as its performance value exceeds some given lower bound.

The chromosomes that comprise the GA population each correspond to leaf nodes in the version graph. Their individual performances are inherited by the element in the \(G\) set that is specifically more general than it is. The advantage to this approach is that the history component is stored in the \(G\) set while the current set of new examples is store in the GA. As a result it is advantageous to modify each chromosome every timestep. Thus, in the VGA it is advantageous to keep the application probabilities for the genetic operators at high levels throughout the search process.

In figure 3 the basic psuedocode for the Version Space Controlled Genetic Algorithm is given. This procedure represents a generalization of the process described above. Often it is convenient to view the version graph as the product of a set of smaller independent factors. However, in this case only a single factor is used.

Figure 4 gives a schematic description of the VGA.
process with relation to the ALE framework. The basic sequence performed therein is now briefly discussed:

1. [Initialize] The population of individual chromosomes is represented as a collection of rectangular structures.
2. [Test] In a given time step a subset of individuals in the population are selected to interact with a given environment. Each of these selected individuals is "unfolded" in order to represent its interaction with the environment.
3. [Update] The performance of an individual is then "fed back" to the population in order to update its performance.
4. [Inherit] The inherit Process involves the actual updating of the performance information associated with the current set of active hypothesis based upon the performance of the individuals associated with each. In other words a subset of the G set inherits a performance or utility adjustment based upon the performance of those individuals who represent it.
5. [Promote/Demote] The updated performance figures for each active hypothesis are then sent back to the population of individuals in order to "promote" those individual chromosomes associated with relationships and to "demote" those who are not.
6. [Empirical Learning Using Genetic Operators] Individuals are then subject to modification using genetic operators based upon their individual performance in the environment as well as the performance of those hypothesis with which they are associated. A new population of chromosomes is produced for use in the next time step.
7. [Analytical Learning in the Version Space] This phase involves the updating of the active hypothesis in the G set based upon the new performance information provided by representative chromosomes in the GA population.

This hybrid system demonstrates certain symbiotic relationships between its components:

1. Generation of examples for the version graph is done automatically by the GA.
2. Presence of the version space makes the termination of learning easier to identify and the results easier to interpret. Learning ceases when experience produces no new generalizations. The contents of the G set can be used to describe the results.
3. Resultant system allows the schema theorem to apply to learning of schemata of higher order and greater defining length [10].

In rest of 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. A hyperschema associated with a given chromosome is the active hypothesis in the G and S sets that are found along a path from the root to the leaf node associated with the chromosome. 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.

**Hyperschema Theorem:**

\[
\begin{align*}
  m(H_{t+1} | HS \in PATHS(H_{t+1})) &\geq m(H_t | HS \in PATHS(H_t)) \times \\
  \frac{\text{avg}(f(H_t) | HS \in PATHS(H_t))}{\text{avg}(f(H_t) | HS \in PATHS(H_t))} \times \\
  \frac{\text{len} - 1}{1 - \left[ p_m \times \text{avg}(o(H) | HS \in PATHS(H_t)) \right] - \\
  p_c \times \frac{\text{avg}(\text{dlen}(H) | HS \in PATHS(H_t))}{\text{len} - 1}}.
\end{align*}
\]

This theorem basically redefines Holland's [5] 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.

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, $f'$.

$$\text{avg}(f'(H,t) \mid HS \in \text{PATHS}(H,t))$$

reflects the performance of the set of schema associated with our given hyperschemata, HS, for the objective function $f'$ relative to the performance of all schemata in the current population.

$$\text{avg}(f^*(H,t) \mid HS \in \text{PATHS}(H,t))$$

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 schema. In that case, each HS corresponds to a unique schema.

5. Results

This section illustrates the experiments used to test the VGA system. The performance of the system is described and than compared to that of Wilson's Boole and Quinlan's C4. Finally, a study is conducted to investigate the correlation between the initial population sizes and the chromosome sizes for the test problems. The VGA system is implemented in C on both a PC and a DEC 3100 work station.

The experiments consisted of two different multiplexers, namely, $F_6$ and $F_{11}$. For convenience, the names used here were the same as those used by Wilson's Boole classifier system. The subscript represents the total number of input lines of the multiplexer. Note that when the problem scales up from $F_6$ to $F_{11}$, the number of inputs of the multiplexer is merely doubled.

<table>
<thead>
<tr>
<th>Initial Pop. size</th>
<th>Ave. Num. of Repro.</th>
<th>Ave. Num. of Patterns in the following Sets</th>
<th>CPU time in Sec.</th>
<th>Marg Accu. of the Test Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>35.5</td>
<td>8</td>
<td>8</td>
<td>20.4</td>
</tr>
<tr>
<td>24</td>
<td>25.0</td>
<td>8</td>
<td>8</td>
<td>5.8</td>
</tr>
<tr>
<td>36</td>
<td>22.2</td>
<td>8</td>
<td>8</td>
<td>2.6</td>
</tr>
<tr>
<td>48</td>
<td>19.0</td>
<td>8</td>
<td>8</td>
<td>1.6</td>
</tr>
<tr>
<td>60</td>
<td>17.2</td>
<td>8</td>
<td>8</td>
<td>0.4</td>
</tr>
<tr>
<td>72</td>
<td>15.0</td>
<td>8</td>
<td>8</td>
<td>0.4</td>
</tr>
<tr>
<td>84</td>
<td>13.0</td>
<td>8</td>
<td>8</td>
<td>0.4</td>
</tr>
<tr>
<td>96</td>
<td>13.0</td>
<td>8</td>
<td>8</td>
<td>0.4</td>
</tr>
<tr>
<td>108</td>
<td>13.0</td>
<td>8</td>
<td>8</td>
<td>0.4</td>
</tr>
<tr>
<td>120</td>
<td>12.6</td>
<td>8</td>
<td>8</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 1. Experimental Results for $F_6$ as a function of population size.

<table>
<thead>
<tr>
<th>Initial Pop. size</th>
<th>Ave. Num. of Repro.</th>
<th>Ave. Num. of Patterns in the following Sets</th>
<th>CPU time in Sec.</th>
<th>Marg Accu. of the Test Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>80.0</td>
<td>16</td>
<td>24</td>
<td>9.4</td>
</tr>
<tr>
<td>44</td>
<td>48.6</td>
<td>15.8</td>
<td>24</td>
<td>9.5</td>
</tr>
<tr>
<td>66</td>
<td>37.0</td>
<td>15.8</td>
<td>23.8</td>
<td>12.6</td>
</tr>
<tr>
<td>88</td>
<td>31.6</td>
<td>16</td>
<td>24</td>
<td>1.2</td>
</tr>
<tr>
<td>110</td>
<td>26.0</td>
<td>16</td>
<td>24</td>
<td>0.0</td>
</tr>
<tr>
<td>132</td>
<td>23.8</td>
<td>16</td>
<td>24</td>
<td>1.0</td>
</tr>
<tr>
<td>154</td>
<td>22.2</td>
<td>16</td>
<td>24</td>
<td>0.2</td>
</tr>
<tr>
<td>176</td>
<td>20.2</td>
<td>16</td>
<td>23.8</td>
<td>0.2</td>
</tr>
<tr>
<td>198</td>
<td>19.8</td>
<td>16</td>
<td>24</td>
<td>0.0</td>
</tr>
<tr>
<td>220</td>
<td>19.0</td>
<td>16</td>
<td>24</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 2. Experimental Results of $F_{11}$ as a function of population size.

However, the size of the solution set is increased 32 times from $2^6$ to $2^{11}$. This implies the fact that adding a few additional inputs in the problem description...
will increase the complexity of the Boole learning tasks for the VGA exponentially.

For each test case, ten experiments were performed with different population sizes. Each experiment consisted of five trials in order to obtain the averaged performance. The initial population size was proportional to the chromosome length of the given problem. The reason for such an arrangement was to provide a constant ratio between the initial population size and the scale of the problem. In order to observe the learning behavior of the VGA system in a wide range of settings, the initial population size for this particular set of experiments was set between two and twenty times the chromosome size.

An example of the output produced by VGA is given figure 5. It gives the parameters of the test case, such as initial population size and the chromosome length. The number of reproduction timesteps performed by the GA is displayed. All the nodes in the G set of the version graph are listed next as the conclusion of the learning process. They represent a set of maximally general patterns that match instances of high performance in the application domain. The performance of the VGA can be measured by the marginal accuracy, the ratio of the number of correct patterns in the list versus the total number of patterns in the list. For this particular example the conclusion generated by the VGA includes the entire solution set for F6.

### Tables 1 and 2 summarize the averaged test results for F6 and F11 respectively. All of the patterns in the output list produced by the learning system are counted in the table under three different categories: 1) the number of solution patterns, 2) the number of overlapping or redundant patterns, 3) the number of incorrect patterns (the patterns that match negative instances). In addition, the average number of population reproductions and the initial population size are listed as well as the CPU time used. The total number of instances evaluated by the system can be computed by multiplying the number of reproduction steps by the initial population size. The marginal accuracy of the test result, which measures the performance of the learning system, is computed and listed.

For both the F6 and F11 it is clear that increasing the population size of the empirical component has two basic effects:

1. The marginal accuracy, in general, increases as the population of chromosomes increases until 100% accuracy is reached. This is because it is more likely with a small population size for active hypothesis to be inadequately tested. That is, some false hypothesis might be carried along in the G set. As the population size increase from 12 to 120 for F6, and from 22 to 198 for F11 the marginal accuracy tends to increase until an accuracy of 100% is reached. Thus, a population around 120 chromosomes is sufficient to provide testing for all active hypothesis in F6. For F11, the initial population needed to produce the same affect increase to around 198.

2. In addition the rate of learning also increases with population size up to a point, at which time it then begins to decrease. Here the rate of learning is assumed to be proportional to the CPU time used. In F6, doubling the initial population form 12 to 24 reduces the CPU time required to solve the problem. Likewise for F11, increasing population size gradually reduces the CPU time used until a population of around 150 is reached. At this point the CPU used is approximately 25% of what it was initially for a population size of 12. This improvement in performance reflects the reduced effort spent in evaluating false hypothesis due to early isolation and pruning of false leads. The larger population allows a more adequate testing of active hypothesis so that poor performers can be isolated early.

Comparing the results with those obtained by other learning systems, the VGA is advantageous in several aspects. First, it successfully overcomes the difficulties of local search in which the traditional GA has problems dealing with. Table 3 lists the learning effort between VGA and the Boole system in terms of the number of instances seen and the percentage accuracy of the test results. In the F6 example, the VGA reaches 100% correctness after evaluating 1500 instances with the initial population size of 120. In contrast, the Boole system [11] needed 15,000 instances with initial population of 400 to reach an average score of 97.3%.
This reduction in effort is due to the fact that the VGA uses version spaces to keep the performance history of the hyperschemata so that the search process can be guided toward the plausible subspaces as predicted by the hyperschema theorem.

<table>
<thead>
<tr>
<th>Learning Task</th>
<th>Number of Instances Seen</th>
<th>Accuracy of Test Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Boole</td>
<td>VGA</td>
</tr>
<tr>
<td>F6</td>
<td>15,000</td>
<td>1500</td>
</tr>
<tr>
<td>F11</td>
<td>30,000</td>
<td>3920</td>
</tr>
</tbody>
</table>


Secondly, it performs as well as Quinlan's C4 without the overhead of manual production of the examples. Table 4 compares the results of the VGA with those for Quinlan's C4 system F11. The C4 system is an inductive learning scheme that generates decision trees from a set of positive and negative examples [8]. The performance of the two systems were compared using initial population for the VGA that was approximately equal to the size of the training sets used by Quinlan in each case,. The VGA system produced marginal accuracies for each problem that were slightly better than those for C4. While the overall performances were similar, the C4 approach required the user to generate 200 examples for F11 in order to achieve this performance while the VGA system randomly generated all the needed examples. Thus, it was easier to provide domain knowledge to produce this level of performance with the VGA than to produce the examples by hand.

Table 4. Quinlan's C4 System (1988).

<table>
<thead>
<tr>
<th>Learning Task</th>
<th>C4 Training Set</th>
<th>VGA Initial Pop.</th>
<th>Accuracy of Test Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C4</td>
<td>VGA</td>
<td>C4</td>
</tr>
<tr>
<td>F6</td>
<td>50</td>
<td>48</td>
<td>85.1%</td>
</tr>
<tr>
<td>F11</td>
<td>200</td>
<td>220</td>
<td>98.3%</td>
</tr>
</tbody>
</table>

Table 4. Quinlan's C4 System (1988).

6. 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 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 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.

7. References