Machine Learning, 13, 189-228 (1993)

A Knowledge-Intensive Genetic Algorithm for Supervised Learning

CEZARY Z. JANIKOW JANIKOW@RADOM.UMSL.EDU
UMSL, Department of Mathematics and Computer Science, St. Louis, MO 63121

Abstract. Supervised learning in attribute-based spaces is one of the most popular machine learning problems studied and, consequently, has attracted considerable attention of the genetic algorithm community. The full-memory approach developed here uses the same high-level descriptive language that is used in rule-based systems. This allows for an easy utilization of inference rules of the well-known inductive learning methodology, which replace the traditional domain-independent operators and make the search task-specific. Moreover, a closer relationship between the underlying task and the processing mechanisms provides a setting for an application of more powerful task-specific heuristics. Initial results obtained with a prototype implementation for the simplest case of single concepts indicate that genetic algorithms can be effectively used to process high-level concepts and incorporate task-specific knowledge. The method of abstracting the genetic algorithm to the problem level, described here for the supervised inductive learning, can be also extended to other domains and tasks, since it provides a framework for combining recently popular genetic algorithm methods with traditional problem-solving methodologies. Moreover, in this particular case, it provides a very powerful tool enabling study of the widely accepted but not so well understood inductive learning methodology.

Keywords. Genetic algorithms, machine learning, symbolic learning, supervised learning

1. Introduction

With the growing amount of available information, limited knowledge-extraction capabilities become a major bottleneck in understanding the world. Recognizing this, there has been an explosion of machine learning attempts to solve this problem. A big part of that research is devoted toward restricted attribute-based spaces. This can be attributed to the existence of many practical problems without a sufficiently understood body of knowledge, but with widely available data in the form of feature descriptions. Traditionally, all approaches have been classified as non-symbolic and symbolic, depending on the output language. Non-symbolic inductive learning systems, often called subsymbolic, usually do not acquire any explicit knowledge but rather gather other information necessary for the descriptive process. They include statistical models, where the only representation is by means of all stored examples or some statistics on them, and the connectionist models, where the knowledge is distributed among network connections and an activation method. Symbolic systems, on the other hand, produce explicit knowledge in a high-level descriptive language. However, an equally important distinction can be based on the level of inference. All non-symbolic approaches process low-level entities (usually numerical parameters). On the other hand, the level of inference of symbolic approaches widely varies, with those operating at higher level showing an advantage: mechanisms based on symbol manipulations, in addition to being closely related to their task objectives, allow for the use of the same input and output

language. This, in turn, creates the possibility of employing some more sophisticated learning paradigms, such as incremental and closed-loop learning, and defining the processing on the same level, which is one of the most important features of our approach described in this article.

With the development of the computer, there has been an increasing interest in simulating nature as a means for problem solving. One of the best-known frameworks was developed by Holland (1975) and is known as genetic algorithms (GAs). This approach models the natural processes of inheritance of coded knowledge and survival by fitness or degree of adaptation to the environment. The two most important characteristics of GAs are robustness and the domain independence of their search mechanism. Robustness, an ultimate goal of any system, is a natural by-product of the search strategy, which performs simultaneous exploration and exploitation. This strategy makes the mechanism quite independent of the characteristics that normally cause difficulty for most other approaches, such as non-smoothness or discontinuity of the evaluation function. Domain independence, on the other hand, is obtained by designing the search operators in the space of the lowest-level representation. However, such an approach has both advantages and disadvantages. On the positive side is the fact that a new application requires only a coding of the problem to this artificial space. On the negative side lies the fact that the quality of such a coding is crucial to the genetic algorithm's performance. Moreover, operating in this space means using problem-blind operators that often overlook some important information that could be utilized to guide the search.

Nevertheless, genetic algorithms have been quite successfully applied to a number of problems. The most outstanding results come from the field of parameter optimization (DeJong, 1988), where the coding is rather straightforward. Other successful applications include optimization problems like wire routing and scheduling, game-playing, cognitive modeling, transportation problems, the traveling salesman problem, and control problems (e.g., Davis, 1991; DeJong, 1985; Goldberg, 1989; Grefenstette, 1987). However, applications to machine learning, although partially successful (Koza, 1989; Rendell, 1985; Schaffer, 1985; Sedbrook, Wright, & Wright, 1991), generally are still too weak for more complex domains and face many related issues (DeJong, 1988, 1990). The genetic algorithm approach to supervised learning in an attribute-based space is normally referred to as symbolic. However, the processing has traditionally been done in symbols of the artificial, not the problem, language. This has begun to change (for example, Grefenstette, 1991; Koza, 1989), following recent changes in the representation utilized (Spears & DeJong, 1990; Goldberg, 1985; Grefenstette, 1991; Smith, 1980).

Following this trend, we propose a different approach: rather than trying to extend the set of operators, we start with a set of inference rules specific to the task, and incorporate them into the genetic algorithm framework. To achieve this, we propose to use a rule-based representation as the natural choice for a symbolic system operating in this space. Having that, we can directly use the inference rules defined in such a framework, namely, those of the inductive learning methodology (Michalski, 1983). By doing so, we utilize the task-specific problem-solving methodology and abstract the genetic algorithm's inference to the problem-specific symbol level. This can be viewed as a knowledge-intensive approach, one using a vast amount of task-specific information, which replaces the blind search of traditional domain-independent operators by a heuristic search. Implementing all the extra

knowledge in the operators leaves the remainder of the genetic algorithm intact and allows for a clear separation of specific knowledge from general mechanisms. Because of the richness of such new operators and their problem-specific behavior, the new algorithm does not enjoy the same theoretical foundations as the traditional GAs do. Nevertheless, we try to justify it intuitively, and the results of our initial experiments indicate its applicability.

Compared to the classical symbolic systems ID and AQ, this new approach, which combines the inductive inference rules with the quality of the GA search, leads to a potentially very robust design that does not assume any prior relationships among different attributes (as, e.g., ID does). The robustness is a result of the existence of the platform for both cooperation (by information exchange) and competition (by selection) among many different simultaneous solutions. This, in turn, can be seen as an extension of the AQ's ideas of processing competing directions (AQ does it by simultaneously retaining a number of partial covers). Here, we provide the cooperation, and we use more powerful heuristics: the inference rules and their adjusting applicabilities.

Successful results may find manifold applications. First of all, the system would provide a quality alternative to existing machine learning techniques. Moreover, applications of all competition and cooperation, generalization and specialization, and more powerful problem-specific heuristics should increase the system's potentials, especially in complex problems. Second, the system would provide a valuable tool to study inductive learning methodology and its inference methods, since the system's principles of operations are based on cooperation and competition among simultaneous applications of different inference rules of the methodology. Third, the approach explored here, which abstracts the genetic algorithm to provide a setting for an ease of accommodation of deep problem understanding to build a hybrid system, can easily be applied to other domains and tasks. Finally, these ideas provide a framework for modular separation of knowledge and performance components in such hybrid genetic algorithms. Such a separation proved to be very successful and desired in other AI architectures.

In this article we attempt to justify our approach, present the system's design, and conclude with some initial experimental results. More experimentation, aimed both at more comprehensive comparison of this system to others and at measuring its exact characteristics, is left to be completed in the near future and will be presented separately. This article is organized as follows. In section 2 we start with a brief description of the genetic algorithm framework used, and follow in section 3 with a specific problem attacked along with a discussion of related issues that reflect on the system's design. We follow with a brief overview of the previous approaches in section 4. Then, we describe our new ideas and their application to the design (section 5) in the simpler case of single concepts. We address some important implementation issues in section 6. In section 7 we attempt both to illustrate the system's behavior by tracing its applications and to present results of some initial experiments. Finally, in section 8, we draw some conclusions and outline work to be done in the future.

2. Genetic algorithms

Genetic algorithms are adaptive methods of searching a solution space by applying operators modeled after the natural genetic inheritance and simulating the Darwinian struggle for

survival. They belong to the class of probabilistic algorithms, yet are distinguished by their different search method and relative insensitivity to local traps. In general, a GA performs a multi-directional search, and it encourages information formation and exchange among such directions. It does so by maintaining a population of proposed solutions (chromosomes) for a given problem. Each solution is represented in a fixed alphabet (often binary) with an established meaning. The population undergoes a simulated evolution: relatively "good" solutions produce offspring, which subsequently replace the "worse" ones. The estimate of the quality of a solution is based on an evaluation function, which plays the role of an environment. The existence of such a population provides for the superiority of genetic algorithms over pure hill-climbing methods, for at any time the GA provides for both exploitation of the most promising solutions and exploration of the search space. The simulation cycle is performed in three basic steps. During the selection step a new population is formed from stochastically best samples (with replacement). Then, some of the members of the newly selected populations recombine. Finally, all new individuals are reevaluated.

The mating process (recombination) is based on the application of two operators: mutation and crossover. Mutation introduces random varability into the population, and crossover exchanges random pieces of two chromosomes in the hope of propagating partial solutions. Because both of these operators are often defined on syntactic pieces of the underlying representation (when each chromosome is viewed as a sequence of the symbols of the low-level alphabet), the search has domain-independent properties. However, the applicability of a GA to a particular problem depends on the representation emphasizing meaningful semantic pieces of information (called building blocks) to be used by crossover operators. Consequently, this applicability may be reduced by operating on the low-level syntactic structures.

Specifying a genetic algorithm for a particular problem involves describing a number of components. Among them, the most important are a genetic representation for potential solutions to the problem, which also defines the search space of the algorithm; a method of generating the initial population of potential solutions; an evaluation function that plays the role of the environment, rating solutions in terms of their "fitness" or "adaptation" to this environment; genetic operators that alter the composition of chromosomes during recombination; and values for various parameters that the GA uses.

The theoretical foundations of genetic algorithms rely on the notion of a schema (e.g., Holland, 1975)—a similarity template allowing an exploration of similarities among chromosomes. Using schemata, a growth equation may be derived that indicates the following hypotheses (e.g., Goldberg, 1989): a genetic algorithm seeks near optimal performance through juxtaposition of special kinds of schemata—the building blocks. Although some effort has been made to prove this hypotheses, for most nontrivial applications we rely on empirical results. Nevertheless, this hypothesis suggests that the coding problem for a genetic algorithm is critical for its performance, and that such a coding should emphasize meaningful building blocks. This, in turn, suggests the following intuitive approach to problem solving by genetic algorithms: the problem representation in a GA should be such that conceptually related alleles are close together in the resulting genotype, where the closeness is defined relative to the crossover operators.

3. Inductive learning from examples

Concept learning is a fundamental cognitive process that involves learning descriptions of some categories of objects. When the objects are described by features (attribute-value pairs) based on a number of multi-valued attributes, the learning is said to be in an attribute-based space. A priori knowledge consists of a set of events that are examples of the space. When each such event is preclassified as belonging to a category, the learning is said to be supervised. The task is to generalize the a priori knowledge in order to produce descriptions of the concepts. When a rule-based framework is used to express the descriptions, the acquired knowledge is often called decision rules. Such rules can subsequently be used both to infer properties of the corresponding classes and to classify other, previously unclassified, events from the space.

The idea of a concept itself may be defined in many different ways, depending on the assumed concept representation and on methods of instance classification. In addition to psychological evidence, this choice is often dictated by the underlying knowledge acquisition method, or rather its output language. For example, assuming a sufficient set of attributes for the events to be consistent and a crisp view, a decision tree can naturally produce complete and consistent partition of the search space. This is so because the decision tree mechanism starts with the whole space and recursively cuts it into disjoint subspaces. On the other hand, it is more difficult for a set of rules to cover the search space in the same manner. Therefore, an extra mechanism is needed to account for possible cases of no-match and multiple-match when recognizing new events. Such problems can be avoided while learning single concepts if the system learns only the concept description and assumes that the subspace not covered by this description represents the complement of the concept. This simplified case is used in this initial design.

An important issue is that of defining both the input and the output language. The input language serves as an interface between the environment (the teacher) and the system. Therefore, it should combine requirements of both of these entities. Moreover, it should minimize inconsistencies among data. The output language serves as an interface between the system and the application environment. Therefore, it should combine the requirements of the learning system with those of the environment. For example, for a pure classification application, there is no need to express the acquired knowledge on a comprehensible level. The output interface is only to provide classifications of some new events. On the other hand, a learning agent used as a part of an intelligent system must be able to communicate its knowledge to other parts of the whole system. If such a system contains elements operating in a high-level language (as an expert system, human expert, etc.), our learning agent should be able to express its knowledge at the same level. Another reason, pointed out by Michalski (1986), relates to the increasing dependence on any automatically generated knowledge:

> An important implication . . . is that any new knowledge generated by machines should be subjected to close *human scrutiny* before it is used. This suggests an important goal for research in machine learning: if people have to understand and validate machine-generated knowledge, then machine learning systems should be equipped with adequate *explanation capabilities*. Furthermore, knowledge created by machines should be

expressed in forms closely corresponding to human descriptions and mental models of this knowledge; that is, such knowledge should satisfy what the author calls the *comprehensibility principle.*

One widely used language, which is closely associated with rules, is VL_1 (Michalski, Mozetic, Hong, & Lavrac, 1986). Variables (attributes) are the basic units having multi-valued domains. According to the relationship among different domain values, such domains may be of different types: nominal, for example $\{Yes, No\}$ in boolean attributes; linear with linearly ordered values; or structured with partially ordered values.

Relations "$=, \neq, <, \leq, >, \geq$" associate variables with their values by means of selectors having the form [*variable relation value*], with the natural semantics. For example, $[Age > Young]$ is interepreted as the set of people of *Middle* or *Old* age, assuming that the three values $\langle Young, Middle, Old \rangle$ are in the domain of the linear attribute *Age*. The *value* in a selector is a single domain value. However, for the "=" relation, it may be a disjunction of such values (the so-called internal disjunction). The internal disjunction, along with the natural semantics, makes the equality relation alone sufficient to represent any formula of the language. This important property is used in the design of our operators. Conjunctions of selectors form complexes.

The significance of the language relies on the natural correspondence to a rule-based paradigm. Selectors can be used to express conditions on single attributes. Complexes can be used to express rules of the form *complex* ::> *decision*. Because of that, the semantics of the language's constructs is easily understood. For example, the following set of rules describes people with heart problems as those who are older and have high blood pressure, or those with high cholesterol levels:

$$[BloodPressure = High][Age \neq Young] ::> HeartRiskGroup$$
$$[CholesterolLevel = High] ::> HeartRiskGroup$$

However, when only single concepts are being learned, the *decision* is redundant and can be omitted. Again, this property is used in our design.

There are two different approaches to learning from examples. The first assumes the existence of working memory able to remember all previously seen events for a future reference. This approach is normally referred to as full memory learning, and the incremental processing is associated with both previously generated knowledge and previously seen examples, in addition to the newly presented events (one should also mention here the case of so-called batch-incremental systems, which process the incrementally available events in relation to only the previously seen events, disregarding the previously generated knowledge). The feasibility of such full-memory systems is restricted by the data set size. However, the other advantages of these systems, especially their relative conceptual simplicity, cause many learning systems to follow this direction—as does our system. In addition, two other important factors favor this approach in the domain of attribute-based spaces: the available data sets for many interesting concepts are appropriately small, and such systems can normally accommodate the large data sets by means of some preprocessing mechanisms. The other approach assumes that the only available memory is for the generated body of knowledge.

Incremental learning capabilities are important characteristics of a learning system. This is so mainly for two reasons: human learning shows incremental characteristics, and a natural setting for a learning system often displays dynamic properties—new evidence becomes available from time to time, as might be the case in a medical database environment where some of the diagnoses may be confirmed after a while, or when new patients are diagnosed. Conceptually, the incremental character is obtained by abilities to generalize and specialize existing knowledge, as necessary, upon new experience. Since our design is based on the inductive learning methodology, which is incremental in nature, the system should possess such capabilities.

Michalski (1983) provides a detailed description of various inductive operators that constitute the process of inductive inference. In the restricted language VL_1 (for induction in an attribute-based space), the most important are as follows: *condition dropping*—that is, dropping a selector from the concept description; *adding alternative rule* and *dropping a rule*—adding/removing one rule from the description; *extending a reference*—extending an internal disjunction; *closing an interval*—for *linear* domains filling up missing values between two present values in a selector; *climbing generalization*—for structured domains climbing the generalization tree; *turning a conjunction to a disjunction*; *inductive resolution*—analogous to the resolution principle. These operators are either generalizing or specializing existing knowledge. There is no provision for generating the initial set of rules. In section 5 we define our versions of these operators to be used in our genetic learning system. We also discuss the choice of the initial knowledge.

4. Previous approaches

Over the past few decades there have been many different approaches to the problem of supervised learning in attribute-based spaces. Some of these approaches came from the AI community, and others from fields such as statistics. One of the most recent ideas has been to use genetic algorithms, to which we pay special attention since our new approach tries to extend such ideas by those of the inductive methodology.

4.1. Traditional approaches

As mentioned earlier, non-symbolic systems rely mostly on quantitative information processing. Therefore, they are further away from mainstream AI devoted to symbol processing. On the other hand, the symbolic systems apply the qualitative approach to learning: the output is a high-level description, and the processing itself is often done at the symbol level.

Statistical approaches account for the vast majority of non-symbolic, or numerical, approaches. They usually operate in batch mode on the data set in order to obtain some statistical measures, which are later used as probabilistic approximations of appearances of different features. To achieve a suitable classification, the correlation of this information to a new example is accumulated using some inference methods. Among such methods, the Bayesian probabilistic model is the best known. The disadvantage of these approaches is that they rely on low-level processing for high-level learning. Furthermore, such treatment

makes it hard to process any available problem-specific knowledge. In addition, the measures used treat all features independently, and high processing complexity does not allow exploration of inter-feature dependencies, even though they could be incorporated (Rendell, Cho, & Seshu, 1989).

Another numerical approach comes from the neural network community. A neural network is a cognitive model of the brain and is composed of two kinds of elements: processing elements (nodes of the network) and connections. Viewed as a memory, such a network has its knowledge distributed among the connections—called weights. These weights determine the propagation of excitatory and inhibitory signals that, in turn, determine the excitation of certain nodes. Such a memory model is capable of learning. The backward propagation of a failure is the best-known method of setting the weights. A neural network method has been applied to simple cases of concept learning with some successes. However, these applications are usually quantitative as well, which makes it difficult to establish a platform for any higher-level knowledge utilization or understanding.

As described, the non-symbolic systems do not follow the methods of the inductive learning methodology, but rather perform numerical computations. This, however, leads to an apparent advantage of better applicability to processing noisy information (Rendel et al., 1989) and more gradual performance degradation.

The two prominent symbolic approaches to supervised feature-based learning, recognized as benchmarks, are Michalski's AQ (Michalski, 1983) and Quinlan's ID (Quinlan, 1986). They are both considerd symbolic systems, even though they have some numerical elements: ID uses an information measure function, while the two-tiered representation of AQ performs a partially probabilistic inference.

The AQ approach is based on inductive generalization and specialization of the VL_1 formulas using the idea of a cover of the positive against the negative events. The cover is constructed in an iterative manner, starting with only one positive and one negative event and continuing until the generated cover is complete and consistent. To prevent an apparent exponential growth in the number of generated descriptions, special heuristics, which accommodate user-defined learning criteria, are employed to reduce the size of partial covers. Retaining a number of such current covers provides for a competition among different solutions. This approach conceptually follows the ideas of inductive methodology, since the generated knowledge is either generalized or specialized, as appropriate. However, the algorithm itself uses only the logic-based operators of negation, union, and intersection to process the current descriptions.

In the ID approach, the training examples are represented by feature vectors similar to events in VL_1. The algorithm constructs a decision tree, where each leaf is associated with a single decision class and each internal node corresponds to an attribute, while each node's branches correspond to a value of that attribute. One of the features of such a tree is that no path from the root to a leaf has two nodes corresponding to the same attribute. The algorithm itself is an iterative application of the information content formula: $I = p * log(p)$, where p is a probability of given information (Quinlan, 1986). At each node of the tree the algorithm only treats events satisfied by the path to this node: the information content is calculated for all such remaining attributes and events, and the attribute giving the maximal information gain is selected as the label for this node. This approach, despite its apparent assumption of attribute independence, proved to be successful in terms of recognition

quality. However, the numerical formula used causes serious problems when one tries to incorporate some task-specific knowledge. Moreover, the algorithm is conceptually very distant from the inductive learning methodology. It iteratively applies specialization, starting with the whole event space. Only then generalization can be applied, by means of tree pruning or rule construction techniques.

Both of the above are full-memory systems, meaning that they assume the availability of all previously seen events at any time during incremental learning. However, they both allow for processing large quantities of data: the AQ uses a preprocessing mechanism selecting only the most representative events, and the ID uses the idea of data windowing.

4.2. Genetic algorithm approaches

Since the early 1980s, there has been an increasing interest in applying GA methods to machine learning—in particular to learning production rules, whose special case is the problem of supervised learning from examples of an attribute-based space. The main problem in such applications is to find a suitable representation, able both to capture the desired problem characteristics and to represent a potential solution. Using a rule-based concept representation brings a different kind of problem: the number of such rules (disjuncts) is not known a priori. Therefore, the traditional fixed-length representation is unsuitable. Two different approaches have been proposed:

- **Michigan** approach, where the population still consists of fixed-length elements, but the solution is represented by a set of chromosomes from the population. This methodology, known as CS for classifier systems, along with a special "bucket brigade" mechanism for credit assignment, was originally developed by Holland and colleagues (Holland, 1986). Here, each chromosome, called a classifier, represents a structure composed of conditions and messages lists. The environment, together with the activated rules, provides a set of active messages. These, in turn, activate other classifiers by satisfying their conditions. The chained actions of message–condition pairs cluster the rules together. Because of this chaining mechanism, this approach seems more suitable for planning than concept learning.
- **Pittsburgh** approach, which represents an extension of the traditional fixed-length chromosome approaches. Here, variable-length chromosomes are used to represent proposed solutions individually. This approach (LS for Learning System) was originally proposed by Smith (1980). This representation seems more naturally suited for the supervised learning, since each chromosome represents an independent solution.

Both of these approaches suffer from some drawbacks. A chromosome of a classifier system does not individually represent a solution. The variable-length approach constitutes a wide divergence from the traditional GA, and, therefore, requires special treatment. Nevertheless, some applications prove to be successful, although in quite limited applications. The two most noticeable genetic algorithm approaches to supervised concept learning in attribute-based spaces, in the LS framework, come from Koza and Spears with DeJong. Koza uses Lisp programs as a means of representing potential solutions, with some tree-

based operators that are closed in the space of such representation. Spears and DeJong use a binary representation for multi-valued domains, and implement only the traditional operators of mutation and crossover in their GABIL system (Spears & DeJong, 1990). Another important LS system is SAMUEL (Grefenstette, 1991), but it was designed and applied mostly to sequential decision problems. Classifier systems have also been used for concept learning (e.g., Wilson, 1987). However there are some additional problems associated with the indirect solution (DeJong, 1990). Moreover, as pointed out in Liepins and Wang (1991), the traditional CS architecture is not suitable for stationary concept learning. Nevertheless, some recent modifications relax these limitations (Booker, 1989).

With very few exceptions (e.g., Spears & DeJong, 1990; Grefenstette, 1991; Koza, 1989), all systems for rule-based learning use a three-symbol alphabet {0, 1, #}, where # stands for a wild-card character (e.g., Goldberg, 1985; Greene & Smith, 1987; Schaffer, 1985), to code each individual feature. Such an alphabet is the choice of many CS- and LS-based systems. However, it is not so well suited for non-binary domains, since it often increases the representation length and extends the search space to infeasible regions. Then an additional mechanism must be implemented to deal with these problems (Green & Smith, 1987). A more systematic approach would be to model closely the non-redundant and feasible-only representation of VL_1, as used in this work and independently in Spears and DeJong (1990). The approach of Grefenstette (1991) also follows a similar direction.

5. The modified genetic algorithm

In this section we first describe the major ideas used in the system's design, and then present the system itself by defining the necessary GA components. Some implementation issues, strongly associated with any such approach, are addressed in the next section.

5.1. Ideas used

One of the most praised characteristics of a genetic algorithm is its domain-independent search (DeJong, 1988). This is the source both of the many successes of GAs, especially in parameter optimization, and, at the same time, of many limitations in other applications. In general, such arguments here are similar to those calling for more task-specific AI methods two decades ago. Recall that, at first, general problem solvers were devised, which were to play the role of general tools for problem solving. It soon turned out that, due mostly to the unmanageable complexity of such methods, it was necessary for the designers of intelligent systems to incorporate domain- and problem-specific knowledge, either by making it explicit or by hiding it in the implementation.

The need for problem-specific knowledge incorporation into GAs was recognized as a method for improvement in many different domains (e.g., Grefenstette, 1987). Davis (1991) calls for such approaches where he calls them "hybrid" genetic algorithms and argues for the exploration of combinations of GAs with existing methodologies in any possible domain. Similar ideas were called for in applications to machine learning. For example, Forrest (1985) proposed using high-level operators such as "concept specialization" and

Table 1. The spectrum of knowledge incorporation in a GA.

Level of task-knowledge incorporated	
None	Full
Only classical mutation and crossover operators	No domain-independent operators, only fully implemented problem-solving methodology

"value restriction"; Antonisse and Keller (1987) also called for similar incorporations. The above were restricted to classifier system architectures. In the LS approach, Grefenstette's SAMUEL system (Grefenstette, 1991) used similar problem-specific operators, including specialization, generalization, rule merge, and delete. However, that approach was aimed at more general sequential decision problems and non-full-memory learning.

Following the Pittsburgh approach, which allows direct representations of VL_1 solutions, we are faced with chromosomes of varying number of structures. Then there is a whole spectrum of possible GA designs along the dimension of task-specific knowledge utilization (see table 1). On one side of the spectrum lies a method that only uses the classicial operators of mutation and crossover. This approach is conceptually very easy. Moreover, it enjoys the same theoretical foundations as the fixed-length GAs: Smith (1980) showed it was true provided that such structures are positionally independent. The above determines the existing popularity of such approaches in any domain. The same is true in the particular case of supervised inductive learning. For example, the previously mentioned GABIL system follows this path.

On the other side of the spectrum lies a knowledge-intensive method that completely abandons the traditional domain-independent operators, and, instead, fully implements the specific problem-solving methodology. This approach is conceptually much more challenging, since it requires, in addition to the GA implementation, a clear understanding of the problem being solved, along with a well-described, complete solving methodology defined at the problem level. This fact, in addition to the lack of well-established theoretical foundations and the resulting diversion from models of nature, determines the low popularity of such approaches. Nevertheless, some machine learning approaches drift in this direction. For example, the new Lamarckian learning operators used in SAMUEL (Grefenstette, 1991) implement learning-specific mutation and crossover.

Even though this latter approach does not have the same theoretical support, it is backed by the task-specific knowledge used to guide and conduct the search. This property should provide for a faster convergence to a desired solution. Moreover, it may be easily shown that all the operators we subsequently define and use (section 5.5) are actually special cases of the traditional mutation and crossover. This provides an intuitive support for the same theoretical foundations. Also, because the operators are defined on the semantic pieces of the problem, one may easily argue that this design naturally satisfies the building-block hypothesis as well.

These are some of the reasons for our decision to pursue this path. But an even more appealing and important justification arises in the context of the learning process understanding and validation. Applications of the task-specific knowledge as the means for inference

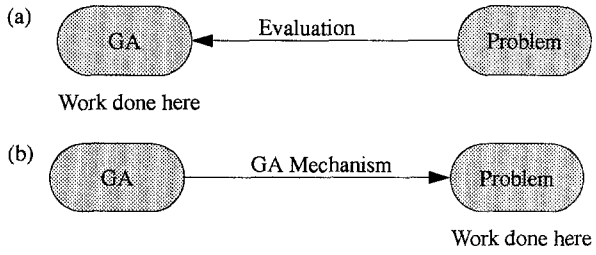


Figure 1. A GA with (a) traditional and (b) task-specific operators.

mechanisms provide for a better understanding of the underlying principles of the learning system. This becomes increasingly important while designing systems that are not only able to generate knowledge, but also able to explain and justify their behavior. For example, Michalski wrote (Michalski et al., 1986):

> ... one should strive to facilitate human understanding not only of the surface results but also of the underlying principles, assumptions, and theories that lead to these results.

This approach is also justified as an abstraction of the genetic algorithm approach. Following the previous discussion on GAs, and those intuitive results stating that the best representation should provide the chromosome structure reflecting syntactic and conceptual knowledge of the problem, we actually go to the extreme of using the problem space as the working search space. In other words, while applications of the traditional domain-independent operators provide for a domain-independent search conducted in the artificial representation space (figure 1a), we set the genetic algorithm to operate directly in the problem space by organizing the work there (figure 1b).

5.2. Representation and search space

We adopt the multiple-valued logic language VL_1 as the choice for chromosome's representation. Then the search space is the space of sets of rules, spanned by given features. This is the space of VL_1 concept descriptions. Because we do not employ any extra axioms, it is quite feasible and possible to have redundant descriptions, such as

$$[Age > Young] ::> HeartRiskGroup, [Age = Old] ::> HeartRiskGroup$$

For simplicity of presentation (but not lack of generality), from now on we only consider VL_1 formulas built using the sufficient "=" relation with internal disjunctions. Moreover, for the same reasons, we assume that we are dealing with single concepts, and that we are learning only a single description (as, e.g., in Spears & DeJong, 1990). This simplification allows us to assume a crisp rule-based conceptualization. We address possible generalizations of this approach to multiple concepts and non-crisp views in the part on future research directions (section 8).

Because of the assumption of learning only a single concept description, all rules are associated with the same single decision, which subsequently does not have to be stated explicitly. Accordingly, when no confusion can arise, we may refer to the same set of rules as just a logical disjunction of VL_1 complexes.

5.3. Initial population

The population contains individuals, each of which is a potentially feasible solution (a set of rules of the VL_1 language). Its size remains fixed (as a parameter of the system). Initially the population must be filled with potential solutions. This process might be totally random (as is normally the case in genetic algorithms), or it might incorporate some task-specific knowledge. There is an obvious trade-off between the level of knowledge used in such an intelligent initialization. On one side of the spectrum is the random choice, very cheap and simple. On the other side, we have an initialization that produces actual solutions to the problem, differing possibly by some applied criteria. This later initialization is actually as hard as the problem we wish to solve. Therefore, it is inapplicable.

We follow the idea of an initialization that is as simple as possible, yet intelligent. Accordingly, we allow for three different types of chromosomes to fill the population initially:

1. The first type is a random initialization. Each individual is a set of a random number of complexes, randomly generated on the search space.
2. The second type is initialization with data. Each individual is a random positive training event.
3. The third type is initialization with prior hypotheses, provided such are available. Each individual is just a single hypothesis given a priori. Having such capabilities, the system can be used as a knowledge refinement tool—possibly cooperating with an expert system of an intelligent hybrid framework.

Actual experiments show that the best average behavior is obtained while using a combination of these three (or the first two if initial hypotheses are not specified), even though the importance of the initialization with positive events seems to diminish with the use of an operator that adds such positive events to current descriptions.

5.4. Evaluation mechanism

The evaluation function must reflect the learning criteria. In supervised learning from examples, the criteria normally include completeness, consistency, and possibly complexity. In general, one may wish to accommodate some additional criteria, such as cost of attributes, length of descriptions, their generality, etc., but we did not consider them in the current implementation.

The completeness and consistency of a rule, or a rule set, measures its quality with respect to the set of training events. We use the formulas presented in table 2, where e^+/e^- is the number of positive/negative training events currently covered by a rule, ϵ^+/ϵ^- is the

Table 2. Completeness and consistency measures.

Structure Type	Completeness	Consistency
A rule set	ϵ^+/E^+	$1 - \epsilon^-/E^-$
A rule	e^+/ϵ^+	$1 - e^-/\epsilon^-$

number of such events covered by a rule set, and E^+/E^- is the total number of such events. These two measures are meaningful only to rule sets and individual rules. For conditions, the measures of the parent rule are used. These definitions assume the full-memory model.

Combining multiple criteria in a single evaluation measure is very dificult and critical for the convergence problem (Goldberg, 1989). In our case we need to combine three such values. We can ease this task by replacing the completeness and consistency measures with a single measure of correctness:

$$correctness = \frac{w_1 \cdot completeness + w_2 \cdot consistency}{w_1 + w_2}$$

This is only one of possible combinations, and in the future we plan to explore the choice's dependence on some problem-specific meta-level knowledge. Finally, we combine correctness and cost by

$$evaluation = correctness \cdot (1 + w_3 \cdot (1 - cost))^f$$

where w_3 determines the influence of *cost* (which itself is normalized on [0, 1]), and f grows very slowly on [0, 1] as the population ages (a dynamic approach). The description's cost is measured by its complexity: $complexity = 2 \cdot \#rules + \#conditions$, as in Wnek, Sarma, Wahab, and Michalski (1990). The above evaluation function is an experimental rather than a theoretical choice, and provides for a controlled bias with respect to complexity of descriptions. Moreover, the w_1 and w_2 coefficients bias the search to more complete or consistent descriptions.

For most practical tests we used a very low w_3 weight ($\sim$ 0.01). Too high a value may cause weak rules to be dropped from the descriptions; too low a value reduces the probability of simplifying the generated descriptions (e.g., dropping redundant rules). The primary reason for the cost accommodation is to force differentiation between the same or similarly covering rule sets that have different complexity. We use a dynamic approach to the use of cost (an approach that adjusts its effects as the population ages). We successfully applied similar dynamic ideas in other domains (e.g., Michalewicz & Janikow, 1991). The effect of the very slowly raising f is that initially the cost influence is very light in order to promote wider space exploration, and it only increases at later stages in order to minimize complexity. Moreover, initial experiments suggest that the system performs better when the f exponent somehow fluctuates, and that the final increase should start based upon an anticipated exhaustion of resources or when the currently learned description is already complete and consistent.

5.5. *Operators*

The operators transform chromosomes to new (possibly better) states in the search space. Since the system operates in the problem space, the operators directly follow the inductive learning methodology. Accordingly to the three syntactic levels of the rule-based framework (conditions, rules, rule sets), we divide the operators into three corresponding groups. In addition, each operator is classified as having either generalizing, specializing, or unspecified—or independent—behaviors. Note that the inductive methodology does not define independent operators. Their introduction is strongly associated with the use of a poulation.

For a better illustration, we exemplify some of the operators using the following search space:

Attribute	Values	Type
A	0, 1, 2, 3	Linear
B	0, 1	Nominal
C	0, 1, 2	Nominal
D	0, 1	Nominal

and the idea of diagramatic visualization (Wnek et al., 1990), which is a multi-valued extension of the well known "Karnaugh map" or "Veitch diagram." Following the correspondence of VL_1 complexes and rules in the single-concept scenario, we represent a rule set as a disjunction of VL_1 complexes. Each complex is a left-hand side of a rule corresponding to the same decision. Also, we try to define the operators as simply as possible, in order to reduce the computational overhead. For example, we do not use the inductive resolution rule, which requires an extensive pattern matching in order to find two complexes having selectors that are negations of each other.

5.5.1. *Rule set level*

This is the level of sets of VL_1 complexes. Here, the operators act on whole rule sets (one or two at a time):

- Independent:

  Rules exchange. This operator requires two parent rule sets, and it exchanges random rules between these two. It requires two parameters: probability of application to a rule set, and probability of rules selection for the exchange.
- Generalization:

  Rules copy. This operator requires two parent rule sets, and it copies a random rule from each of the sets to the other. It differs from the "rules exchange" operator, since it does not remove information being propagated from the rule set. It requires one parameter: probability of application to a rule set.

  New event. This operator acts on a single rule set: if there is a positive event not covered yet by the current rule set, this event's description is added to the set as a new rule:

$$chrom = \cup_i (cpx_i ::> dec) \text{ and } \exists_{e^+} \forall_i (e^+ \not\Rightarrow cpx_i) \lhd chrom \cup (e^+ ::> dec)$$

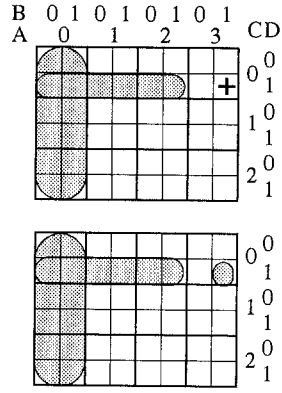


Figure 2. A visualization of the "new event" operator.

It requires only one parameter: probability of application to a rule set. Contrary to expectations, there is no pattern matching involved in this operator, and its actions have very little computational overhead due to data compilation—see section 6.3. For an illustration, see figure 2.

Rules generalization. This operator acts on a single rule set. It selects two random rules and replaces them by their most specific generalization:

$$cpx_1 ::> dec,\ \mathrm{cpx}_2 ::> dec \lhd (cpx' ::> dec)$$

where *cpx'* is the most specific generalization (not necessarily consistent with respect to previously excluded negative events) of the two complexes. It requires one parameter: probability of application to a rule set. For an illustration, see figure 3.

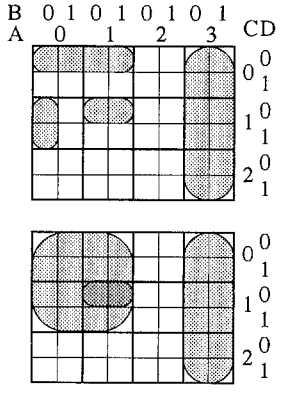


Figure 3. A visualization of the "rules generalization" operator.

- Specialization:
  Rules drop. This operator acts on a single rule set, and it drops a random rule from that set. It requires one parameter: probability of application to a rule.
  Rules specialization. This operator acts on a single rule set, and it replaces two random rules by their most general specialization:

$$cpx_1 ::> dec,\ cpx_2 ::> dec \rhd (cpx' ::> dec)$$

  where cpx' is the most general specialization (not necessarily complete) of the two complexes. It requires one parameter: probability of application to a rule set.

5.5.2. *Rule level*

This is the level of VL_1 complexes. Here, the operators act on one rule at a time.

- Independent:
  Rule split. This operator acts on a single rule, and it splits it into a number of rules, according to a partition of the values of a condition (an absent condition can be selected as well, using all domain values). The set of domain values present in the selected condition can be split according to each value individually or according to two disjoint subsets of values. For the linear data types, the latter split is more desired. In this case, the present domain values are split by cutting the ordered set of values in a single random place. For the nominal data type, the former split is more desired. A structured type requires a slightly more sophisticated approach, similar to that of the linear type, but with differently defined values and orderings. This operator requires three parameters: probability of application to a rule, and probabilities of a subset vs. all values split, separately for the linear and nominal data types.
- Generalization:
  Condition drop. This operator acts on a single rule, and it removes a present condition from that rule:

$$((cpx = \wedge_i sel_i) ::> dec) \lhd (cpx' ::> dec)$$

  where cpx' has all but one selectors of cpx. In other words, one of the selectors of cpx is extended to cover the whole domain of the associated attribute. It requires a single parameter: probability of application to a rule.
  Turning conjunction into disjunction. This operator acts on a single rule, and it splits the complex into a disjunction:

$$((\wedge_i sel_i \wedge \wedge_j sel_j) ::> dec) \lhd (\wedge_i sel_i ::> dec) \vee (\wedge_j sel_j ::> dec)$$

  where the complex's separation into n and m selectors is random and position independent. It requires a single parameter: probability of application to a rule.

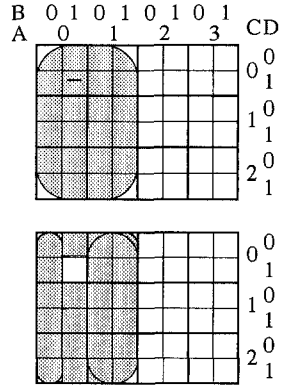


Figure 4. A visualization of the "rule directed split" operator.

- Specialization:

  Condition introduce. This operator acts on a single rule, and it introduces a random condition associated with an unconditioned attribute:

$$((cpx = \wedge_i sel_i) ::> dec) \rhd (cpx' ::> dec)$$

  where cpx' has, in addition to all the selectors of cpx, a new selector associated with an attribute not present in cpx. The new selector is a random choice from among all of its possible internal disjunctions. It requires a single parameter: probability of application to a rule.

  Rule directed split. This operator acts on a single rule. If this rule covers a negative event, it is split into a set of maximally general rules that are yet consistent with that event, in the following way:

$$(cpx ::> dec) \text{ and } \exists_e\text{-}(e^- \Rightarrow cpx) \rhd \cup_i (cpx_i ::> dec)$$

  where the new set has cpx's such that $(\vee_i cpx_i) = (cpx \wedge \neg e^-)$. This operator resembles the action in the heart of the covered procedure of AQ15. It requires a single parameter: probability of application to a rule. Again, there is very little computational overhead involved due to data compilation. For an illustration, see figure 4.

5.5.3. Condition level

This is the level of VL_1 selectors. The operators act on one condition at a time.

- Independent

  Reference change. This operator acts on a single condition, and it randomly removes or adds a single domain value to this condition. It requires a single parameter: probability of application to a condition.

- Generalization:

  Reference extension. This operator acts on a single condition, and it extends the domain by allowing a number of additional values. For the nominal type, some random values are selected for extension. For the linear type, a single value may be selected or, with a higher chance, a range may be closed (between two present values, using the domain ordering). Moreover, such shorter open ranges have a higher chance of being selected over longer ones. This operator requires quite a few parameters, including a probability of application to a condition, and a number of selection probabilities determining the choice of an action. For the structured type, we replace some of the present values by their parent in the generalization tree, giving preference to those offsprings that prevail in number (generalization climbing).
- Specialization:

  Reference restriction. This operator acts on a single condition, and it removes some domain values from this condition. Its actions and parameters are analogous to those of the "reference extension," but have opposite effects.

5.5.4. Operator selection probability

As seen in the definitions, each operator is given some initial probabilities from two separate groups: selection and application probabilities.

The selection probabilities serve as means of selecting one of a number of possible actions or substructures to participate in the operations—these probabilities are static. The application probabilities serve as a means of firing operators for structures of their type (based on the definition level). These probabilities have a dynamic character with respect to the current context, that is, to both the current coverage and the current, problem-dependent, size of the average chromosome. First, probabilities of generalizing operators are increased for applications to structures (rule sets, rules, conditions) that are incomplete, and are decreased for those that are inconsistent. On the other hand, the probabilities of specializing operators are increased for applications to structures that are inconsistent, and decreased for those that are incomplete. Moreover, the levels of probability increase/decrease are based on the levels of inconsistency/incompleteness. In other words, these two measures serve as additional heuristics (in addition to fitness), which guide the selection of appropriate operators. Second, all application probabilities are adjusted to achieve a constant chromosome update rate. For example, more complex problems, which cause the intermediate chrmosomes to be longer (both in terms of the number of complexes and their sizes), decrease all such probabilities by the same fraction.

While selecting the appropriate bias for completeness and consistency, we must be careful not to decrease the probabilities so far as to prevent certain operations from performing. Since we want the changes to be proportional to those measures, the following choices seem natural (but still experimental):

$$\text{Generalizing operators: } p' = p \cdot \left(\frac{3}{2} - \textit{completeness}\right) \cdot \left(\frac{1}{2} + \textit{consistency}\right)$$

$$\text{Specializing operators: } p' = p \cdot \left[\frac{1}{2} + \textit{completeness}\right] \cdot \left[\frac{3}{2} - \textit{consistency}\right]$$

The new value p' is the adjusted probability, and p is the actual probability. It is important to mention that since p' is computed differently for each rule and rule set, it does not replace the a priori p. The simplicity of this formulas guarantees a low computational overhead.

To accommodate the changes in problem-specific characteristics, namely, the average size of a complex and the average length of chromosomes in the current population, we use the following approximation. We observe the number of chromosomes undergoing recombination in a given population. If this number represents too large a portion of the population, we decrease all the a priori application probabilities by a fraction for the next reproductive iteration. If this number is too small, we do the opposite. Since the adjustments are computed for the whole population, they actually always replace the previous values. Experiments show that with an appropriately small such adjustment fraction, the changes converge and then the probabilities remain relatively steady. This method provides for a partial independence of such probabilities from some characteristics of the problems.

5.6. Algorithm

The algorithm uses the above components and the control of genetic algorithms. At each iteration, all rule sets of the population are evaluated and a new population is formed by drawing members from the original one in such a way that more fit individuals have a higher chance of being selected (sections 2 and 5.4). Following that, the operators are applied to population members in order to move these partial solutions, hopefully, closer to the desired state. Each structure stochastically invokes some operators defined for its level: the selection depends on the operators' initial strength, the consistency/completeness of the structure, and the average size of the current chromosomes. Then the cycle repeats until a desired description is found or computational resources are exhausted. For a better illustration, refer to the experimental tracing of section 7.2.

6. Some implementation issues

We implemented a simple **C** prototype of the proposed approach in order to be able to test our ideas. The choice of the language, aside from efficiency, was based on availability of bitwise logical operators used to speed up the evaluation mechanism (section 6.3). We call this implementation **GIL** (for **G**enetic-Based **I**nductive **L**earning). In this section we present the most important issues facing any such implementation, along with approaches used in GIL.

6.1. Sampling mechanism

The selection algorithm is to choose some chromosomes from the current population to form a new population, with possible omissions or repetitions. There are many standard

ways of performing this step. Baker (1987) provides an excellent discussion. We use the stochastic universal sampling mechanism. This method builds a roulette wheel for the chromosomes, with each chromosome having allocated a portion of the wheel proportional to its evaluation fitness. A second wheel is constructed with equally spaced marks. The number of such marks is the same as the number of samples to be drawn (the size of the population). The wheels are placed on a single axis and the one with marks is randomly spun against the other. The positions of the marks, in relation to the space allocated for each chromosome on the other wheel, are then observed. A chromosome is selected once for each mark landing in its allocated space.

6.2. Internal representation

In section 5.2 we described the architecture of the chromosome, in terms of the language used. Now we discuss some important issues associated with the internal representation.

In section 4.2 we mentioned that the widely used three-symbol alphabet is not suitable for the multi-valued domains of feature-based spaces. A more appropriate solution was suggested originally by Greene and Smith (1987) and was recently used by Spears and DeJong in their GABIL system (Spears & DeJong, 1990). This approach uses binary digits to represent domain values. For example, assuming that an attribute has five domain values, the binary vector 11001 represents the condition saying that the attribute must have the first, second, or the last value (assuming some positional enumeration of values from the left, and a use of the internal disjunction). We use exactly these ideas to implement conditions (selectors of VL_1). A chromosome's length is actually unrestricted—a rule set may contain any number of rules, organized as linked lists. An important issue associated with the rule set is that of treating rules that are invalid, that is, conditions that are totally restricted (exclude all domain values). Spears and DeJong (1990) suggested keeping such rules as possible sources of valid conditions. We performed some experiments to study the trade-off between anticipated increases in the computational cost of retaining these rules vs. improvements in predictive accuracy of the system. Our conclusions are far from final. Nevertheless, they suggest there is a clear conflict between these two factors. A similar issue arises in the context of empty rules—those not covering any positive events. Such rules, again, may be removed or retained. Also, there seems to be a similar kind of complexity vs. quality trade-off. Currently, GIL removes both invalid and empty rules.

Each complex is a conjunction of a number of conditions. The number of such possible conditions, in a given complex, is bounded by the total number of attributes. We use that bound as a way of simplifying the internal representation: a complex is represented by a vector of conditions. Furthermore, for simplicity and efficiency, we associate a fixed positional correspondence between the attributes of the vector. This does not introduce any problems, since no operators acting at the condition level are positionally dependent.

Such an implementation introduces a new dilemma: how to treat unrestricted conditions, that is, those that include all domain values in the selector. It is a question of elegance, or possible efficiency, rather than power: an unrestricted selector can be dropped from its complex without any semantic change to the rule associated with this complex. We tried both approaches: one with all selectors present in each complex, and the other with

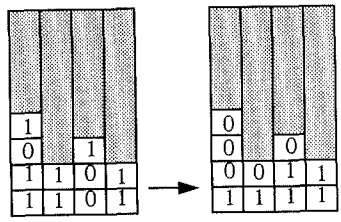


Figure 5. An internal representation of a sample chromosome.

unrestricted selectors invisible to all operators by means of a special flag. We found no significant difference in the system's performance under both conditions; we used the latter approach in GIL.

The above ideas are illustrated in figure 5, assuming the set of features used in section 5.5 while defining the operators on an eight-bit machine. This figure shows an internal representation of a chromosome, which can be viewed as a disjunction of complexes or a set of rules using an implicit decision. Moreover, depending on the treatment of unrestricted domains, the same chromosome can be described in different ways as well. The resulting four possible semantically equivalent views are as follows:

1. $[A = 0, 1, 3][B = 0, 1][C = 2][D = 0, 1] ::> dec$, $[A = 0][B = 0][C = 0][D = 0, 1] ::> dec$
2. $[A = 0, 1, 3][C = 2] ::> dec$, $[A = 0][B = 0][C = 0, 1] ::> dec$
3. $[A = 0, 1, 3][B = 0, 1][C = 2][D = 0, 1] \vee [A = 0][B = 0][C = 0, 1][D = 0, 1]$
4. $[A = 0, 1, 3][C = 2] \vee [A = 0][B = 0][C = 0, 1]$

6.3. Data compilation

A very commonly cited disadvantage of genetic approaches to problem solving is their time complexity (e.g., Quinlan, 1988). This problem becomes especially visible when the evaluation requires an extensive computation. This is the case in supervised learning when evaluating rule sets in a full-memory system, since this process involves extensive pattern matching. Concerned with such problems, we designed a special method of data compilation aimed at improving the time complexity of the system.

The idea is as follows: rather than storing data in terms of features, store features in terms of data coverage. In other words, for each possible feature, retain information about the events covered by this feature. This must be done separately for each concept. Moreover, it must be done even for concepts not being explicitly learned. For example, this means that GIL has to remember such coverage separately for both the concept and its negation. We achieve this by enumerating all learning events and constructing binary coverage vectors. This approach is feasible only for problems with small to medium size of data set. For those rare cases of larger sets (e.g., those larger than 10^4), special data-reduction mechanisms would have to be employed.

Table 3. Examples of binary coverage vectors of a feature.

Positive coverage vector:	10000001000000001010000000
Negative coverage vector:	000000000001010

The idea behind these vectors is analogous to that of representing conditions. A coverage vector is constructed for both E^+ and E^- separately. In this vector, a binary one at position n indicates that the structure that owns this coverage vector covers event #n. For example, the vectors in table 3 indicate that the given feature covers positive events #1, 8, 17, 19 (out of 25), and negative events #12 and 14 (out of 15).

Prior to learning, as mentioned, all data are precompiled into such vectors, for all possible features. During the actual run of the system, similar vectors are constructed for all structures of the database: from the features upwards. For example, having the feature coverages, we can easily construct both positive and negative coverage of the condition $[A = 0, 2]$, by means of a simple bitwise OR on appropriate coverage vectors of features $(A = 0)$ and $(A = 2)$. Subsequently, condition coverages are propagated to rules by means of a simple bitwise AND. Finally, rule coverages are propagated to rule sets again by means of the bitwise OR.

Perhaps the most important effect of such an approach is that we can incrementally upgrade such coverages using a minimal amount of work after the initial database is fully covered. For example, consider a copy of the "rules copy" operator applied to the following two rule sets:

$$R_1 = r_1^1, r_1^2$$

$$R_2 = r_2^1, r_2^2, r_2^3$$

and suppose the operator copies r_2^2 to R_1. The coverage of the second rule set does not change. To compute the coverage of the first rule set, it is sufficient to perform bitwise OR between the coverage of the rule r_2^2 (which did not change during this operation) with the coverage of the original R_1. In other words, we compute this coverage using two bitwise OR operations (one for the positive and one for the negative coverage). In general, the number of such required operations increases (very slowly linearly) with the number of training events.

As another example, consider the case of the "reference change" operation, with a single change from 0 to 1, on position #, in a condition's binary vector. All that needs to be done to update the coverage of this condition is to perform bitwise OR on the coverages (positive and negative) of the corresponding feature number # associated with the given attribute with those of the original condition. Then this change must be propagated to the appropriate rule and rule set's coverages, using similar simple computations.

7. Experimental studies

In this section we attempt both to illustrate the system's behavior by tracing its applications and to experimentally compare its behavior to that of other learning methods. A more comprehensive testing to determine the system's characteristics is left for the future.

7.1. Experimental methodology

In the literature, learning systems are often evaluated and compared using a standard set of well-recognized artificial and real data. Examples widely used are random DNFs, multiplexers, soybean disease, breast cancer, etc. To evaluate GIL, we use some of these standard data sets. This also allows us to use published results of experiments with other systems, under the assumption of repeating exactly the same experimental sessions.

In inductive learning, the acquired knowledge should meet two criteria: high predictive accuracy of unseen events and comprehensibility at some high cognitive/conceptual level. The quality of the former measure estimates the generalization power of the system; we call it a quantitative property. On the other hand, we call the latter a qualitative property.

The most common experimental methodology is to split the available events into training and testing groups (usually 70% and 30%). Subsequently, the experiment calls for a learning session using the training group, followed by testing using the other group (containing events unseen during the training). Different measures are then used to determine the qualities of a system. For example, for the quantitative properties, Michalski and Chilausky (1980) define a set of conflicting measures under the assumption that in some cases it makes no sense to distinguish between two close diagnoses. However, most researchers use a single measure of accuracy, defined as the ratio of correctly classified events to all testing events: this is the measure we use unless otherwise stated. To measure the qualitative properties, we either list separately the number of rules and conditions used or combine them according to the well-accepted formula shown in section 5.4.

Another important issue is the estimate of statistical measures. We followed those used in the reference publications for each experiment. However, the default approach was to take an average of five independent resampled runs.

7.2. Emerald's robot world

Recently, a report of the AI laboratory at George Mason University (Wnek et al., 1990) evaluated a number of different learning systems using the world of robots from the Emerald system (Kaufman, Michalski, & Schultz, 1989). Since this report provides a detailed description of the experiment, it was relatively easy to repeat exactly the same tests with our system and compare the results directly. In this experiment, we attempted to achieve two goals: to illustrate the system's algorithms and exemplify its comprehensive processing mechanism by tracing one of the experiments; and to gather both qualitative and quantitative results in order to compare them with those published.

This robot world is very suitable for this kind of experiment, since it is moderately complex to allow comparative study, yet simple enough to be illustrated by the diagramatic visualization method. It is described by the following six attributes (we boldface the abbreviations subsequently used in the trace):

Attribute	Values
HeadShape	*Round, Square, Octagon*
Body	*Round, Square, Octagon*
Smiling	*Yes, No*
Holding	*Sword, Balloon, Flag*
JacketColor	*Red, Yellow, Green, Blue*
Tie	*Yes, No*

The robots were classified into the following five categories created by a human:

Concept	Description
C_1	*Head* is *Round* and *JacketColor* is *Red* or *Head* is *Square* and *Holding* a *Balloon*
C_2	Smiling and *Holding* a *Balloon* or *Head* is *Round*
C_3	*Smiling* and not *Holding* a *Sword*
C_4	*Jacket* is *Red* and no *Tie* or *Head* is *Round* and is *Smiling*
C_5	*Smiling* and *Holding* either a *Balloon* or a *Sword*

The task was to learn a description of each concept while seeing only a varying percentage of the positive and negative examples. There were a total of 432 different robots present in this world. The error rate reported is the average error in recognizing all the 432 (both seen and unseen) events. This measure explicitly estimates the predictive accuracy, while at the same time implicitly judges the generalization and specialization power.

7.2.1. The trace

For the behavior trace we used concept C_1, which can be represented by the following set of rules:

$$[H = R][J = R] ::> C_1,\ [H = S][Ho = B] ::> C_1$$

or, assuming an implicit decision as used in GIL, by the following formula:

$$[H = R][J = R] \vee [H = S][Ho = B]$$

Of the 432 events, 84 satisfy the concept. The training was done using a random 20% of both positive and negative examples: 17 and 70, respectively (see figure 6 for a visualization of the target concept and the training events). The population size was set to 40, initialized equally by both random descriptions and positive training events. The system was set to run 100 iterations. Other implementation parameters were set as follows: $w_1 = w_2 = 0.5$; $w_3 = 0.02$; and the cost was normalized with respect to the highest cost in the current population. The initial application probabilities, along with actual adjusted values (adjustment for the currently average size of the chromosomes; see section 5.4) at the end of this experiment, are presented in table 4. This scaling was computed assuming a desired rate

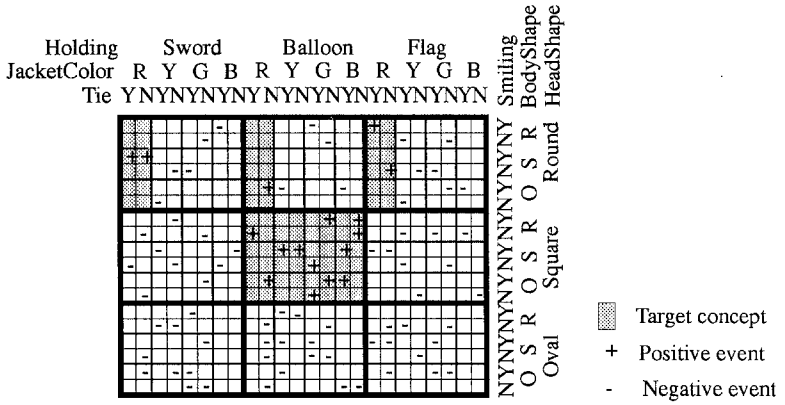


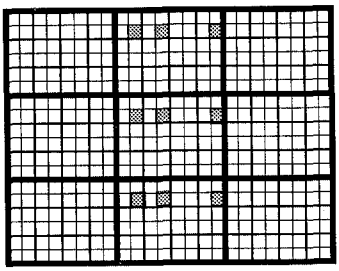
Figure 6. The goal concept and the training events.

Table 4. Application probabilities.

Level	Operator	Initial Values	Final Values
Rule set	Rules exchange	0.20	0.115
	Rules copy	0.10	0.058
	New event	0.40	0.230
	Rules generalization	0.50	0.288
	Rules drop	0.50	0.288
	Rules specialization	0.50	0.288
Rule	Rule split	0.02	0.011
	Condition drop	0.10	0.058
	Turning conj. into disjunc.	0.02	0.011
	Condition introduction	0.10	0.058
	Rule directed split	0.12	0.069
Condition	Reference change	0.02	0.012
	Reference extension	0.03	0.017
	Reference restriction	0.03	0.017

of 80% chromosomes to be updated by the recombination step. The selection probabilities were as follows: 0.2 for a rule selection in "rules exchange"; 0.1 for "splitting a rule" according to two subsets, as opposed to all domain values, for the nominal type, and 0.7 for the linear type; and 0.5 for all probabilities on the condition level.

Before the algorithm could start iterating, it had to perform three initial steps: data compilation, initialization, and initial evaluation. Data compilation involved building the binary coverage vectors for all possible features of the space. Initialization involved setting up the population. Finally, evaluation involved finding the completeness, consistency, and complexity of the rules and accumulating them according to our formula of section 5.4. The best such initial chromosome is illustrated in figure 7, which happened to be a randomly generated description.



Iteration 0: (from initialization)

Cost = 7 : 1 rule, 5 conditions
Positive coverage = 1
Negative coverage = 0

Rule set:
[B=R][S=N][Ho=B][J=R,Y,B][T=N]

Figure 7. The best initial chromosome.

Each iteration of the genetic algorithm consists of three basic steps: selection, reproduction, and evaluation. However, the special data-compilation method, along with the use of the binary coverage vectors, allows for combining the last two steps as follows. Each operator is followed by a proper update to all the affected vectors and a recalculation of the completeness and consistency measures. Then, the only task of the evaluation step is to calculate the total fitness: there is no pattern matching involved. In the case of an incremental learning (this particular experiment was conducted in the batch mode), every time a new example is presented to the system, all appropriate features (those present in that example) are incrementally updated and propagated to other structures present in the population.

The most important step to explain is reproduction. Normally, reproductive operators are selected based on some static probabilities. However, in our case there are two dynamic factors that affect such probabilities: the rate of chromosome update and the completeness and consistency measures of proper structures. First, the rate of chromosome update (from the previous generation) is compared to that desired (80% in this case, or 32 chromosomes); if it differs by more than some allowable margin, all the application probabilities are accordingly adjusted by a small fraction and the new values replace the old ones. Then each structure of the population is allowed to nondeterministically select operators to be applied to it. Competing operators are those defined for the level of the structure. Moreover, both the generalizing and the specializing operators have their application probabilities adjusted by each structure, before a uniform probability generator decides their actual application. For example, consider a rule set with the following measures: *completeness* = 0.2, *consistency* = 0.9. All operators defined for this level are tried in a random order. Each one actually found applicable updates this complex. The independent operators have probabilities of application exactly as the initial values (possibly adjusted with respect to the desired update rate). The generalizing operators have probabilities of application additionally adjusted by

$$\left(\frac{3}{2} - \textit{completeness}\right) \cdot \left(\frac{1}{2} + \textit{consistency}\right) = 1.82$$

and the specializing operators have the probabilities adjusted by

$$\left(\frac{1}{2} + completeness\right) \cdot \left(\frac{3}{2} - consistency\right) = 0.42$$

In other words, this particular rule set would face an increasing pressure for generalization.

Each operator actually selected for application updates the structure, and then it immediately incrementally updates binary coverage vectors from the structure up to the chromosome level. Also, the appropriate completeness and consistency measures are immediately reevaluated. Such an incremental approach not only reduces the time complexity of evaluations by taking into account some specific information about properties of the operators, but also leaves all the coverages and the measures consistent for the other competing operators. This is very important, since some of them rely on such information for further efficiency improvements. For example, the "directed rule split" operator needs to find a negative event inconsistent with the current complex. If the operator can rely on the coverage vectors, finding such an event is reduced to selecting a random binary one from the negative coverage vector of this complex. Otherwise, the complex would have to be sequentially matched against all possible negative training examples.

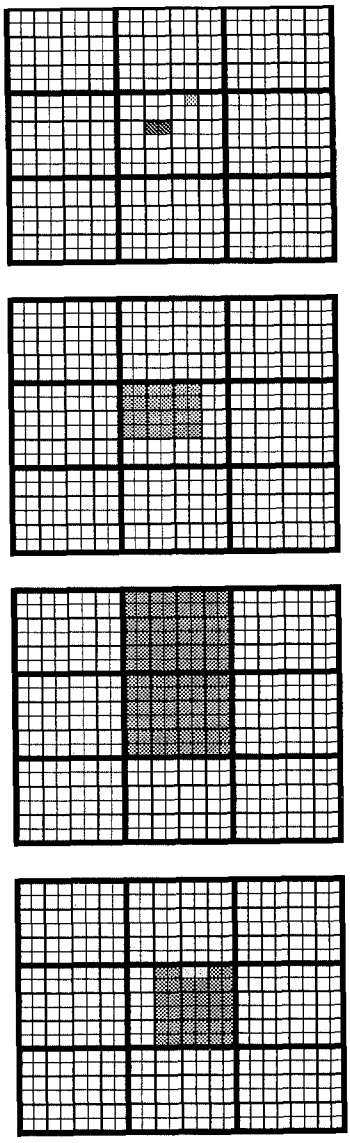
Following the initialization and the initial evaluation, the system was run for the specified number of iterations. Not every iteration produced a better chrmosome. There were 11 improvements in 27 iterations, after which the best description exactly matched that of the concept. The first four improvements are illustrated in figure 8.

This problem proved to be extremely simple for our algorithm. After only 27 iterations, or about 3.1 CPU seconds on a DEC3100 station, a complete and consistent description was found. Moreover, this description exactly matched the desired solution—there were no redundancies. This experiment illustrates both the quanitative and qualitative properties of the system.

7.2.2. Comparative experiments

The other systems used in the experiment (reported in Wnek et al., 1990) were rule-based AQ15, neural network BpNet, decision tree with rules generator C4.5, and genetic classifier system CFS. Table 5 reports the average error rate for the five experimental concepts for all five systems while learning one concept at a time (the results of the other four obtained from those published experiments). Surprisingly, GIL produced the highest recognition rate, especially when seeing only a small percentage of the events. This result can be attributed to the simplicity-biased evaluation formula that was used.

Table 6 reports the average acquired knowledge complexity by listing both the average number of rules and the average number of conditions, as learned by all five systems for different learning scenarios in the same experiment. The NR entry indicates that the complexity was large and was not reported in the reference paper. The reason for the higher complexity of the connectionist approach is that this is a non-symbolic system operating on numerical weights rather than on the problem symbols. On the other hand, the high complexity of the CFS approach can be attributed to the fact that the symbolic processing was being done in the representation rather than in the problem space and to the lack of a similar bias for simplicity. This result is rather common for classifier system approaches.



Iteration 1

Cost = 15 : 2 rules, 11 conditions
Positive coverage = 3
Negative coverage = 0

Rule set:
[H=S][B=S][S=Y][Ho=B][J=Y]
∨ [H=S][B=R][S=Y][Ho=B][J=G][T=N]

Iteration 2

Cost = 6 : 1 rule, 4 conditions
Positive coverage = 6
Negative coverage = 0

Rule set:
[H=S][B=S,O][S=Y][Ho=B][J=Y,G,B]

Iteration 3 (an overgeneralization)

Cost = 4 : 1 rule, 2 conditions]
Positive coverage = 12
Negative coverage = 4

Rule set:
[H=R,S][Ho=B]

Iteration 7 (one redundant rule)

Cost = 12 : 2 rules, 8 conditions
Positive coverage = 10
Negative coverage = 0

Rule set:
[H=S][Ho=B][J=Y,G,B]
∨ [H=S][B=R][S=Y][Ho=B][J=G]

Figure 8. The first four improvements.

Therefore, it was a pleasant surprise to find that GIL's knowledge was at the same complexity level as that of AQ15. This demonstrates one of the system's characteristics: the ability to generate easily comprehensible knowledge (measured here by complexity of its VL_1 output). The 1.4/2.6 result in the first column is clearly an oversimplification of the knowledge due to insufficient number of training events (here only about ten negative events were available).

Table 5. Error rate summary in the robot world.

	Learning Scenario (positive%/negative%)				
System	6%/3%	10%/10%	15%/10%	25%/10%	100%/10%
AQ15	22.8%	5.0%	4.8%	1.2%	0.0%
BpNet	9.7%	6.3%	4.7%	7.8%	4.8%
C4.5	9.7%	8.3%	11.3%	2.5%	1.6%
CFS	21.3%	20.3%	21.5%	19.7%	23.0%
GIL	4.3%	1.1%	0.0%	0.0%	0.0%

Table 6. Complexity's summary in the robot world (#rules/#conditions).

	Learning Scenario (positive%/negative%)				
System	6%/3%	25%/10%	50%/10%	75%/10%	100%/10%
AQ15	2.6/4	1.6/3	1.6/3	1.6/3	1.6/3
BpNet	NR	18/29	NR	NR	32/54
C4.5	6.8/12.2	4.4/9.2	4.8/9.2	4.8/9.2	3.8/7.3
CFS	NR	NR	NR	NR	NR
GIL	1.4/2.6	1.6/3	1.6/3	1.6/3	1.6/3

7.3. DNF concepts

Learning DNF descriptions has become a standard way of evaluating different systems. An interesting experiment was reported by Spears and DeJong (1990), in which the authors compared a decision-tree-based ID5R with their own genetic algorithm for supervised concept learning, GABIL (a newer version is described in DeJong and Spears (1991)). The test data for that experiment were a set of random DNF descriptions of a varying complexity. The results represent batch-incremental learning curves: a system's quality measure after seeing n examples is defined as an average recognition of a single unknown random event over the last ten experiments (from $n - 9$ to n). Accordingly, the learning curves are undefined for $n < 10$.

There was a total of six attributes, each having three possible values. Six sets of experiments were conducted, for six randomly constructed DNF concepts $x\mathrm{D}y\mathrm{C}$, where $x = 1, 2$ is the number of rules, and $y = 1, 2, 3$ is the number of conditions per rule.

For each experiment, a total of 100 events was chosen randomly. Then, using an increasing number of learning events and just one testing event, the learning curves were constructed using average results over ten independent runs with resampling. For the incremental ID5R, the knowledge was updated incrementally upon a new inconsistent event, while it was generated from scratch in the GABIL system (which does not possess such incremental properties).

We repeated the same experiments in exactly the same environment, using the same batch-incremental mode as GABIL. Because the DNFs actually used were not reported in that paper, we repeated the experiments not only with resampling, but also with randomly

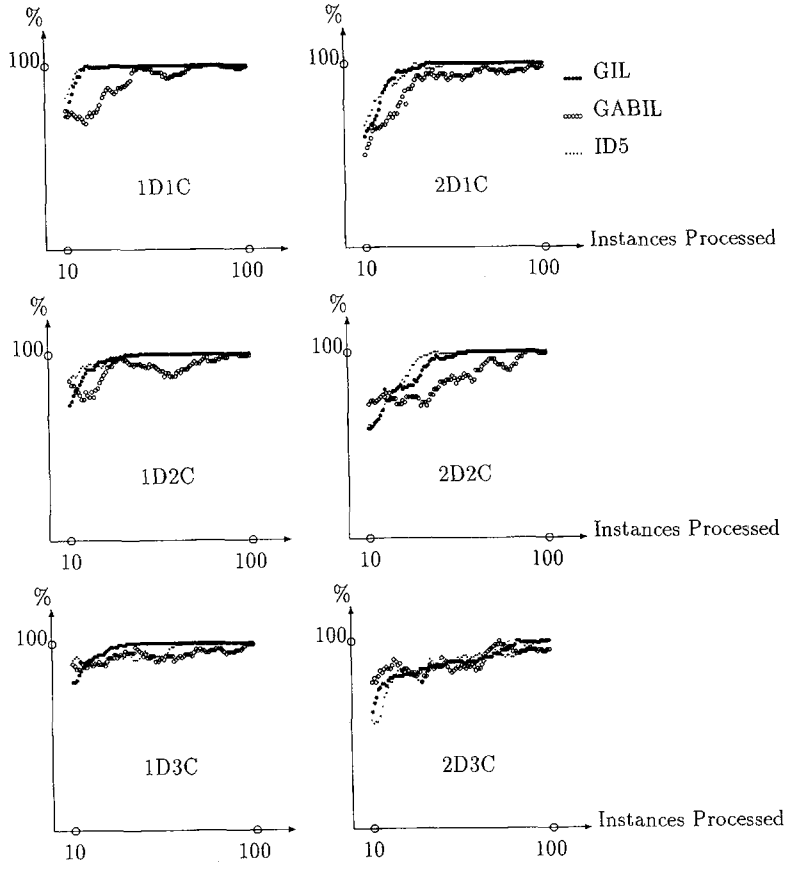


Figure 9. Batch-incremental results on CNF data.

regenerated target descriptions (each run of 100 iterations). Our results, along with the original ones, are presented in figure 9. The original claim of the GABIL system was that it could not learn as well as ID5R on simple concepts, but achieved about the same levels of performance (in some cases, slightly better) as the concepts' complexity increased. Our results show that GIL can do both at the same time: it achieves very high performance for all kinds of problems. It also clearly outperforms GABIL in terms of learning variability. Its smooth learning curve indicates low variability in performance, while the broken curve of GABIL indicates bigger differences from run to run. Moreover, GIL produced descriptions of much lower complexity (not reported for GABIL—private correspondence).

7.4. Multiplexers

The family of multiplexers is another widely used set of data. Each multiplexer is actually a specific case of the more general DNF. For each integer $k = 1, 2, \ldots$ there is a multiplexer

Boolean function defined in the following way: the inputs are the k bits (called addresses), and there are exactly 2^k outputs (called data bits). Accordingly, we have multiplexer f_3 for $k = 1$, f_6 for $k = 2$, f_{11} for $k = 3$, etc. The function of a multiplexer is to activate the data bit whose address (in binary, assuming some ordering of the data bits starting at 0) is specified by the address bits. For example,

$$[A_0 =][A_1 = 0][A_2 = 1] \vee [A_0 = 0][A_1 = 1][A_3 = 1]$$

$$\vee[A_0 = 1][A_1 = 0][A_4 = 1] \vee [A_0 = 1][A_1 = 1][A_5 = 1]$$

defines the f_6 multiplexer in the VL_1 language, assuming that attributes A_0 and A_1 are the two address bits and A_2 to A_5 are the four data bits.

Many experiments have been documented, mostly using f_6 and f_{11} functions. For example, Koza (1989) describes learning the first of these two, with his LISP-influenced hierarchical genetic approach. He reports a case of learning the actual function after processing 4500 potential solutions, while seeing all 64 (2^6) possible instances. Our own experiments indicate an average learning after seeing only about 2000 individual (40 iterations, database size 50). However, these two results should not be compared directly—Koza apparently used a much larger population size of 300, and the solution was found after 15 iterations. Moreover, there is no indication whether this was the only experiment or the best of a number of experiments. Many classifier systems reported experimentations with a different multiplexer (e.g., Wilson, 1987). However, since CS are in general not full-memory systems, it is again difficult to compare the results directly. Instead, we again decided to illustrate GIL's behavior by tracing its run on f_{11}—this time by observing not the intermediate rule sets but rather the complexity, consistency, and complexity of the best rule set in each generation.

Figure 10 traces a sample run while training with 20% of the available f_{11} events. During this learning session, the exact concept was learned after 1700 iterations. A consistent and complete description of the training events was found shortly after 1000 iterations, and the remaining 700 cycles were required to simplify the generated description. The first of these two graphs traces completeness and consistency of the currently best database individual. The other graph traces the complexity of the best individuals. It is interesting to note that the complexity rises during the learning, as a result of not finding simple enough complete and consistent descriptions, and then decreases—forced down by an increasing cost influence and formation of such better descriptions.

Some quantitative results with f_{11} are reported in table 7. These results are similar to those from other systems (e.g., Quinlan, 1988), but a different experimental methodology does not allow for a direct comparison. They are also similar to reported results of AQ (Wnek & Michalski, 1991), where the authors report 74% accuracy after training with 6% of the events. Some more recent experiments (Janikow, in press) seem to suggest that GIL's results may be greatly improved by adjusting the learning bias. For example, while learning with an increased pressure on consistency, a 100% average accuracy can be achieved for the 20% learning case.

A few interesting differences were observed between f_6 and f_{11} runs. The time necessary for the learning rose from about 10 CPU seconds to about 20 minutes, or from about 100

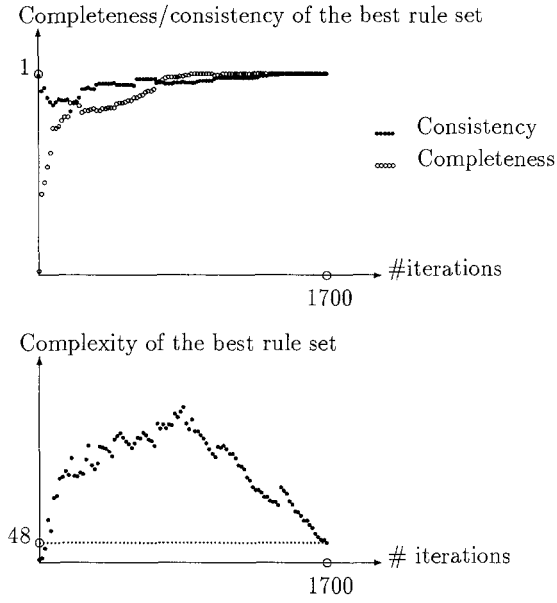


Figure 10. A sample behavior on multiplexer f_{11}.

Table 7. GIL's accuracy on multiplexer f_{11}.

% Training Events	Accuracy
5%	77%
10%	88%
20%	97%

to about 2000 iterations. This increase can be attributed to the increased event space (64 vs. 2048) and the increased search space ($\sim 10^{200}$ vs. $10^{60,000}$). More directly, this caused an increased number of iterations and a noticeable increase in the average complexity of chromosomes. A further investigation is needed to determine the scaling characteristics of the system.

Yet another interesting difference between the two multiplexers can be observed by comparing the accuracy while learning with a small percentage of the available events: f_{11} achieves high rates much more quickly. The reason for such a behavior seems to be the ratio of the concept complexity to the size of event space, which is about 0.313 for f_6 and 0.023 for f_{11}. This raises an interesting hypothesis about estimating the difficulty of generating descriptions: the difficulty is proportional, or at least highly correlated, to the ratio of the concept's complexity and the size of the event space. We hope to further investigate this assertion in the figure.

Table 8. Summary of the breast cancer experiment.

System/Method	Complexity	Accuracy
Human experts	Not reported	64%
AQ15/full rule set	41 rules/160 conditions	66%
AQ15/best rule only	2 rules/7 conditions	68%
Assistant/without tree pruning	63 leaves/120 nodes	67%
Assistant/with tree pruning	9 leaves/16 nodes	72%
GIL	36 rules/128 conditions	65%
GIL/with emphasized cost	10 rules/27 conditions	67%

7.5. Breast cancer

One of the most popular natural domains used in experiments with inductive learning systems is the breast cancer data. It contains 286 descriptions of female patients, classified as either developing or not developing a recurrence of breast cancer after a five-year period following the first surgery. The descriptions are generated using nine attributes, with an average of 5.8 values per domain. This descriptive language was found to be inconsistent, meaning that some patients having exactly the same description were classified differently. Such a situation puts an extra burden on the learning system.

An excellent paper by Michalski and colleagues (1986) lists both quantitative and qualitative results on this data set, while using the AQ15 system with a rule truncation mechanism and a decision tree system ASSISTANT (this version generates low-complexity binary trees and employs a tree truncation technique). In addition, this paper also reports the accuracy of human experts. We repeated these experiments (for 1000 iterations) and report all such results in table 8. Our second run was performed with an increased cost influence on the evaluation (increased w_3 parameter). Both results indicate the applicability of our approach in the case of natural domains as well. For compatability with the complexity results reported for the other systems, we ran GIL twice, each time learning one of the two possible concepts and summarizing the #conditions and #rules.

7.6. Incremental learning

Incremental learning capabilities are important attributes of a learning system. We expected our approach to possess such properties naturally, since it is based on generalization and specialization of the current hypotheses and does not explicitly favor either of these two classes of actions at any time. To evaluate this assertion, we performed both batch and incremental experiments to produce the learning curves of f_6. Both learnings were performed at 5% increments of the available training events: in the batch mode, each new experiment started with newly generated population; and in the incremental mode, each new experiment started with the population generated at the end of the previous learning session with fewer training events. All experiments were repeated five times with resampling, with population size set to 40 and run for 100 interations. Figure 11 presents these curves. It is interesting to note that there was no significant difference in the quantitative

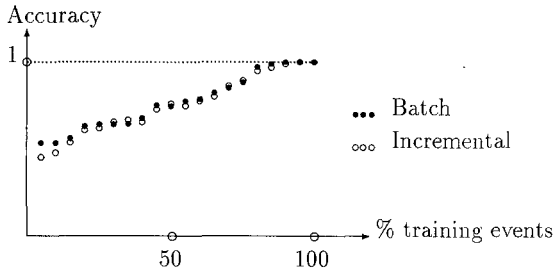


Figure 11. Comparison of the batch and incremental learning.

performance, suggesting GIL can be used in incremental environments. Moreover, observed faster convergence in the incremental mode suggests that the reused knowledge in forms of the initial population was processed very usefully, indicating the system's ability to process initial hypotheses and maintain knowledge in a dynamic environment. Again, more experimentation is needed to further investigate these properties.

8. Conclusions and further research

We have described a novel approach to full-memory, supervised inductive learning in attribute-based spaces that uses a knowledge-intensive genetic algorithm. Such an algorithm follows the ideas of traditional GAs, but replaces the domain-independent search by a search specific to the inductive learning methodology. This approach represents an abstraction of the traditional genetic algorithm to the symbolic level. Initial results show that GAs can be successfully applied to more complex, non-numerical tasks by defining the algorithm at the conceptual level of the problem. This allows for processing high-level structures using the problem-specific methodology and rich heuristics. Moreover, such an abstract view provides for the same clear separation of different systems' components as found in AI production systems. This modularity, in turn, allows for transparent applications of similar designs to other domains. Some of the most interesting properties of this approach are the clearly understood processing mechanisms, defined on the same language as that of input/output, and the low complexity of the generated knowledge. This complexity, measured here by number of rules and conditions, contributes to the higher comprehensibility of the output.

When pursuing this challenge, we did not attempt to produce a system that could compete with the existing symbolic systems (AQ and ID based), especially in terms of learning time complexity. Our goal was rather to investigate the potential of such a method of abstracting genetic algorithms, which may be carried to other domains. Nevetheless, by designing efficient data-compilation methods aimed at reducing the system's complexity, and by using more "intelligent" operators than those in the traditional GAs, we were able to tackle a number of interesting problems in a reasonable time. Moreover, since genetic algorithms are naturally suited for parallel architectures, we may hope that such reimplementations, along with new technological advances, may be faster without any additional efforts.

The system also shows significant potential from the machine learning point of view. First, it does not assume attribute independence, as the ID-based systems do. Second, it provides for a better match between problem-specific heuristics and actions taken than the AQ-based systems do. Finally, it extends the AQ's ideas of exploring a number of simultaneous directions to a more powerful platform, allowing for both competition and cooperation (by information exchange).

The current complexity, as well as the overall quality, seems to be unstable and to vary among different problems, different parameters used, and even different runs of the same experiment. Making the algorithm more stable requires an extensive study of its characteristics. Other important issues to be addressed include learning multiple concepts and dealing with noisy information. These issues are further discussed below along with some proposed solutions. Yet other, more challenging, future research involves using more powerful languages than VL_1 and adapting the system in a non-full-memory environment.

One should point out that most of the testing data were not suited to explore the full potential of this system: most experiments were conducted with two- or three-valued domains. Such domains are much more suitable to the other learning systems, while this approach can fully explore spaces with larger, typed domains. Actually, among the other known learning systems, only AQ and SAMUEL try to accommodate knowledge of domain types, which can be quite valuable when the domains grow. However, AQ does so only after learning the initial complete and consistent descriptions (a recent development in AQ systems attempts to use this information in a second stage of its two-tiered learning).

8.1. Multiple domains

The current system assumes single concepts only, and proceeds by learning only rules associated with the single decision. This lets us treat the decisions implicitly and then simplify the definitions and the implementation by processing VL_1 complexes in place of rules. For the sake of simplicity and continuity, we will try to preserve this property while generalizing the approach to learning multiple decisions. If we assume independence of different class descriptions, we may apply subsequent learning sessions of the same algorithm, one session per class. Then, during a learning sesson for class n, the examples of category n are considered as positive events, while the examples of all other categories are considered as the current negative events. The same idea may be used differently: we may conduct the learning sessions simultaneously in different populations, with one population assigned to one class being learned.

When discussing such generalizations, we must consider a related issue: treating descriptions that are incomplete and inconsistent with respect to the problem space. As we mentioned before, a rule-based framework is not well suited for learning descriptions with those properties, which are often relaxed. However, in such a case, a previously unknown event may be recognized by none or multiple descriptions. Then, a special arbitration protocol must be employed. The simplest such protocol simply returns such events as unrecognizable. This improves correct recognition rate, but also increases overall indecision. Moreover, this approach can treat descriptions of different classes independently. A more sophisticated protocol employs some flexible (e.g., probabilistic) measures, and thus changes

the conceptualization view to non-crisp. However, this also suggests that the descriptions should not be generated independently, but in a common context. Then each rule set must be evaluated in the context of the descriptions for the other classes. This suggests learning all descriptions in a single population, with a chromosome being a set of descriptions now. We can still preserve the implicitness of decisions if we introduce an additional syntactic level: the decision set set level. Then, while learning descriptions of n categories, each chromosome becomes a set of descriptions, with each description associated with exactly one of the categories. In other words, a chromosome becomes a set of the previously defined chromosomes. Then, all operators that were defined as acting on two chromosomes will act on two rules sets associated with the same decision. All other operators stay exactly as defined previously. However, we need new operators to act on the new level. The only proposed operator has an independent character, and it exchanges one or more whole-category descriptions between two chromosomes. In other words, this operator works at the level of granularity of a whole rule set associated with one implicit decision.

As to the non-crisp arbitration protocol, a very nice solution is used in the AQ family—the two-tiered conceptualization view. We suggest using the same approach in such an extended architecture, which could provide an additional advantage. The two-tiered view could be used during the learning process, while it is used only in a follow-up step in the AQ system. Because a given problem specifies a priori the number of categories to be conceptualized, we can easily extend the chromosome implementation of section 6.2 to a vector of such, where a vector position is associated with the decision number.

8.2. Other issues

One of the major disadvantages of the current implementation is its high parameterization: there are about 40 input parameters that must be specified, with most of them being continuous probability values. Under such conditions, it seems highly unlikely that the proper combination for a given run will be selected. This is the reason for the poorer performance noticed on more complex problems (e.g., multiplexer f_{11}). Moreover, on some occasions we observed an improved performance after slightly changing the initial probabilities. However, a much more extensive and systematic experimentation is necessary to determine the actual source of such variations: randomness of the runs or some problem-specific characteristics.

To deal with the problem of the size of the parameter space, it is necessary to explore inter- and intra-dependencies between such parameters. For example, all rule-set level application probabilities should be specified with respect to each other and, as a group, with respect to those of other groups. The dependence of the selection probabilities on the problem size should be explored. However, establishing such relations requires an extensive testing over a wide variety of different problems. To deal with the second problem (dependency on characteristics of the problem), the choice of such abstracted parameters should be further associated (in addition to the dynamic method of section 5.5) with the problem complexity (determined dynamically in the process of learning) and some learning criteria (specified by the user or some other requirements). Then all these parameters could be replaced by few conceptual ones, for example, desired type of descriptions (such as specific

vs. general, or low attribute cost vs. low descriptive cost). Such an abstraction would provide for both easier use and more efficient performance. To deal with other non-probabilistic parameters, most of which control the learning bias, it is necessary to study such bias extensively. Some recent results show that adjusting the bias appropriately can improve both the speed and the quality of learning (Janikow, in press).

While it might be conceptually advantageous to assume the existence of noise-free data, it is often unrealistic under natural conditions. Therefore, an artificial system aimed at working in a natural domain should deal with these issues. Some researchers addressed the problem of noisy features. For example, Clark and Niblett (1987) discuss the effect of noise on induction and present their probabilistic way of dealing with the problem. Quinlan (1990), showed how to deal with the effect of noisy data in decision trees. The two-tiered representation of AQ family of systems applies to noise effect reduction as well. Looking at these attempts, it is fair to say that noise accommodation generally employs some kind of numerical approach. Our simplified approach assumes a crisp concept representation with rule-based conceptualization and is not well suited for dealing with noise: the most appealing approach would be to combine rule-coverage information with rule complexity in such a way that rules with very low positive coverage become more costly and more likely to be removed. The same strategy applies also to all of the proposed generalizations to deal with multiple concepts. However, in the case of the extended chromosome architecture, the two-tiered description may itself be more valuable as a noise-accommodation agent, since this approach was shown to improve noisy recognition in the AQ15 system.

We mentioned in section 5.5 that pursuing atomicity and efficiency over complexity made us neglect the inductive resolution rule. It would be an interesting study to compare the behavior of the current system with another one implementing this operator, along with other possible new operators. Such a study should concentrate on both the quantitative and qualitative properties, as well as possible quality vs. time trade-off.

In addition, some of the rule set level operators described in section 5.5 could be differently defined, depending on the selection of applicable rules. For example, the "rules exchange" operator could overlook the selection probability and always select a single rule. On the other hand, the "rules copy," "rules generalization," "rules drop," and "rules specialization" could be enhanced by such probabilities, instead of always choosing a fixed number of rules (one or two, depending on the nature of the operator). Again, an extensive study is required to establish values of such new operators in relation to those presently used.

The ideas used here can be seen as a very specific case of more general ideas presented by Davis (1991), where he calls for exploring combinations of the traditional domain-independent genetic algorithms with some known problem-specific methods (the hybrid approach). The only difference between this system and those ideas lies in the initialization. Davis argues that the first population should be filled by chromosomes representing properly represented results of some known fast systems. The argument for such an enhancement is that then the genetic algorithm can only improve such ad hoc solutions. Therefore, it is guaranteed to perform at least as well as such other systems. In our domain, there are such programs. For example, most decision tree systems are very efficient and produce assertions easily convertible to a rule-based format. Since our initial experiments indicate the system's ability to process some initial hypotheses, there is a potential for a significant

performance and time improvement. Moreover, another interesting idea is to use decision-tree-based operators, which partition the current event subspace, instead of the less sophisticated "rule split" operator.

Acknowledgments

The author wishes to thank all of the reviewers for their valuable comments. This work was partially supported by a computational grant from the Microelectronic Center of North Carolina.

References

Antonisse, H.J., & Keller, K.S. (1987). Genetic operators for high level knowledge representation. *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 69–76). Hillsdale, NJ: Lawrence Erlbaum Associates.

Baker, J.E. (1987). Reducing bias and inefficiency in the selection algorithm. *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 14–21). Hillsdale, NJ: Lawrence Erlbaum Associates.

Booker, L.B. (1989). Triggered rule discovery in classifier system. *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 265–174). San Mateo, CA: Morgan Kaufmann.

Clark, P., & Niblett, T. (1987). Induction in noisy domains. In I. Bratko & N. Lavrac (Eds.), *Progress in machine learning*. Sigma Press.

Davis, L. (Ed). (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.

DeJong, K.A. (1985). Genetic algorithms: A 10 year perspective. *Proceedings of the First International Conference on Genetic Algorithms* (pp. 169–177). Hillsdale, NJ: Lawrence Erlbaum Associates.

DeJong, K.A. (1988). Learning with genetic algorithm: An overview. *Machine Learning, 3 (2/3)*, 121–138.

DeJong, K.A. (1990). Genetic-algorithm-based learning. In Y. Kondratoff & R.S. Michalski (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 3). San Mateo, CA: Morgan Kaufmann.

DeJong, K.A., & Spears, W.M. (1991). Learning concept classification rules using genetic algorithms. *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 651–656). San Mateo, CA: Morgan Kaufmann.

Forrest, S. (1985). Implementing semantic networks structures using the classifier system. *Proceedings of the First International Conference on Genetic Algorithms* (pp. 24–44). Hillsdale, NJ: Lawrence Erlbaum Associates.

Goldberg, D.E. (1985). Genetic algorithm and rule learning in dynamic system control. *Proceedings of the First International Conference on Genetic Algorithms* (pp. 8–15). Hillsdale, NJ: Lawrence Erlbaum Associates.

Goldberg, D.E. (1989). *Genetic algorithms in search, optimization & machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D.E., & Holland, J.H. (1988). Genetic algorithms and machine learning. *Machine Learning 3 (2/3)*, 95–99.

Grefenstette, J. (1987). Incorporating problem specific knowledge into genetic algorithms. In L. Davis (Ed.), *Genetic algorithms and simulated annealing*. London: Pitman.

Grefenstette, J. (1991). Lamarckian learning in multi-agent environments. *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 303–310). San Mateo, CA: Morgan Kaufmann.

Green, D.P., & Smith, S.F. (1985). A genetic system for learning models of consumer choice. *Proceedings of the Second International Conference on Genetic Algorithms* (pp. 217–223). Hillsdale, NJ: Lawrence Erlbaum Associates.

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Holland, J.H. (1986). Escaping brittleness. In R.S. Michalski, J. Carbonell & T. Mitchel (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). San Mateo, CA: Morgan Kaufmann.

Holland, J.H., Holyoak, K.J., Nisbett, R.E., & Thagard, P.R. (1986). *Induction*. Cambridge, MA: MIT Press.

Janikow, C.Z. (1991). *Inductive learning from attribute-based examples: A knowledge-intensive genetic algorithm approach*. Doctoral dissertation, University of North Carolina at Chapel Hill, Department of Computer Science.

Janikow, C.Z. (in press). Some experiments with a stochastic production system for supervised inductive learning. In *Artificial intelligence: Methodology, systems, applications* (Vol. 5). Amsterdam: North-Holland

Kaufman, K.A., Michalski, R.S., & Schultz, A.C. (1989). *EMERALD 1: An integrated system of machine learning and discovery programs for education and research.* (Technical Report MLI-89-12). Fairfax, VA: Center for AI, George Mason University.

Koza, J.R. (1989). Hierarchical genetic algorithms operating on populations of computer programs. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 768–774). San Mateo, CA: Morgan Kaufmann.

Liepins, G.E., & Wang, L.A. (1991). Classifier system learning of Boolean concepts. *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 318–323). San Mateo, CA: Morgan Kaufmann.

Michalewicz, Z., & Janikow, C.Z. (1991). Genetic algorithms for numerical optimization. *Statistics and Computing, 1 (1)*, 75–89.

Michalski, R.S. (1983). Theory and methodology of inductive learning. In R.S. Michalski, J. Carbonell, & T. Mitchel (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 1). San Mateo, CA: Morgan Kaufmann.

Michalski, R.S. (1986). Understanding the nature of learning. In R.S. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). San Mateo, CA: Morgan Kaufmann.

Michalski, R.S., & Chilausky, R.L. (1980). Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems, 4 (2)*, 125–161.

Michalski, R.S., Mozetic, I., Hong, J., & Lavrac, N. (1986). *The AQ15 inductive learning system: An overview and experiments* (Technical Report UIUCDCS-R-86-1260). Urbana, IL: Department of Computer Science, University of Illinois at Urbana-Champaign.

Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning, 1 (1)*, 81–106.

Quinlan, J.R. (1988). An empiricial comparison of genetic and decision-tree classifiers. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 135–141). San Mateo, CA: Morgan Kaufmann.

Quinlan, J.R. (1990). Probabilistic decision trees. In Y. Kondratoff & R.S. Michalski (Eds.), *Machine learning: An artificial intelligence approach (Vol. 3)*. San Mateo, CA: Morgan Kaufmann.

Rendell, L.A. (1985). Genetic plans and the probabilistic learning system: Synthesis and results. *Proceedings of the First International Conference on Genetic Algorithms* (pp. 60–73). Hillsdale, NJ: Lawrence Erlbaum Associates.

Rendell, L., Cho, H., & Seshu, R. (1989). Improving the design of similarity-based rule-learning systems. *International Journal of Expert Systems, 2 (1)*, 97–133.

Schaffer, J.D. (1985). Learning multiclass pattern discrimination. *Proceedings of the First International Conference of Genetic Algorithms* (pp. 74–79). Hillsdale, NJ: Lawrence Erlbaum Associates.

Sedbrook, T.A., Wright, H., & Wright, R. (1991). Application of genetic classifier for patient triage. *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 334–338). San Mateo, CA: Morgan Kaufmann.

Smith, S.F. (1980). *A learning system based on genetic algorithms.* Doctoral dissertation, University of Pittsburgh, Department of Computer Science.

Spears, W.M., & DeJong, K.A. (1990). Using genetic algorithms for supervised concept learning. *Proceedings of the Second International Conference on Tools for AI* (pp. 335–341). Washington, DC: IEEE Computer Society Press.

Weiss, S.M., & Kapouleas, I. (1989). An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 781–787). San Mateo, CA: Morgan Kaufmann.

Weiss, S.M., & Kulikowski, C.A. (1990). *Computer systems that learn.* San Mateo, CA: Morgan Kaufmann.

Wilson, S. (1987). Classifier systems and the animat problem. *Machine Learning, 2 (3)*, 199–228.

Wnek, J., & Michalski, R. (1991). *Hypothesis-driven constructive induction in AQ17: A method and experiments* (Technical Report MLI-91-4). Fairfax, VA: Center for AI, George Mason University.

Wnek, J., Sarma, J., Wahab, A., & Michalski, R.S. (1990). Comparing learning paradigms via diagramatic visualization. In M. Emrich, Z. Ras, & M. Zemankowa (Eds.), *Methodologies for intelligent systems 5.* Amsterdam: North-Holland.

Received October 7, 1991
Accepted March 27, 1992
Final Manuscript July 23, 1992