Combining Competition and Cooperation in Supervised Inductive Learning

Cezary Z. Janikow
Department of Mathematics and Computer Science
University of Missouri at St. Louis
St. Louis, MO 63121
email: janikow@radom.umsl.edu

Abstract

Tools for automatic generation, verification, and maintenance of knowledge bases become more and more important with the amount of widely available information growing. For supervised concept learning in attribute–based spaces, many approaches have been proposed including the symbolic AQ and ID based algorithms. These algorithms exhibit competitive characteristics since either partial covers or the attributes compete for consideration at any given moment. In this paper, we describe a new full memory approach which implements a very unique search mechanism combining the competition with cooperation. This approach uses the VL_1 language in a framework utilizing operators of inductive learning methodology and an inference engine modeled upon genetic algorithms. We also present some experiments indicating the applicability of this approach to both quantitative and qualitative learning.

1 Preliminaries

Concept learning from examples, in attribute–based description spaces, has been extensively studied. This popularity is due to wide availability of data, ease of compiling data from numerous databases, and relative simplicity of the language required. This simplicity, especially when further restricted to discrete domains (either naturally such or artificially discretized), allows for a number of methods to be applicable to the problem. Accordingly, there is a number of approaches ranging from statistical, connectionists and genetic algorithms to decision trees and decision rules.

The objective of supervised concept learning is to transform initial knowledge consisting of a set of preclassified event descriptions into output knowledge. The purpose of such acquired knowledge is two–fold: to predict classifications of new events and to extract such knowledge, possibly for processing by other entities of a hybrid intelligent system (or a human). However, very few researchers attempted to measure the latter properties of systems, mainly because very few systems possess them. Generally, a system can exhibit such properties only if the output knowledge is expressed in a high level language; decision trees and decision rules (ID and AQ) based systems fall into this category.

The theory and methodology of inductive learning has been proposed in (Michalski 1983). It provides a general framework for inductive learning end describes the operators needed for knowledge manipulation, including *rule drop, condition drop, reference extension, reference restriction, interval closing, etc.*, in a rule–based framework. For the attribute–based description spaces, the VL_1 language is used. Having that, two questions come to mind: what about directly implementing the methodology as means for supervised concept learning and empirically studying it.

The direct implementation is very difficult because of huge search spaces of attribute–based descriptions with relatively weak heuristics. For example, using ten descriptive attributes with three values per domain, we are faced with 7^{10} number of different rules. Then, the number of possible concept descriptions is $2^{7^{10}}$, or almost $10^{100000000}$. One solution to this problem would be to use hill-climbing techniques. However, the available heuristics, relying mostly on partial measures of completeness and consistency, could easily lead to local traps. Therefore, irrevocable strategies are quite inapplicable. Another solution would be to use some tentative techniques. However, the huge search spaces require any such method to be extremely well informed, or otherwise the explored database would grow unmanageably fast. Again, the available heuristics are not strong enough to provide such qualities. As to the empirical studies, we would need to have a direct implementation of the methodology in order to study the inductive learning. Having such capabilities would be very useful not only in justifying the methodology itself, but also in understanding the learning process.

A system implementing these ideas would have some additional desirable properties. It would express both the input and the output knowledge in the same language, VL_1 in this case. Such a scenario allows for both processing of hypothesis and incremental concept formation (as does AQ). Moreover, with the amount of automatically processed knowledge rapidly growing, there is an emphasis on building systems whose performance can be justified by understanding and validating not only the generated knowledge, but also the underlying mechanisms and processing principles (Michalski 1986). Because this system would apply the inductive learning mechanisms (operators), it would exhibit exactly these characteristics.

Between the two prominent symbolic approaches, ID uses processing mechanisms that are conceptually quite distant from the problem level: it uses information measure, and it iteratively specializes inconsistent subspaces. AQ provides mechanisms at the conceptual level of the problem (it processes descriptions), but it does so by utilizing logic–based unions and intersections of VL_1 formulas that actually implement only the *extension against* rule.

In the rest of the paper, we present the ideas leading to the design and outline of the algorithm. Then, we present few experimental results aimed at evaluating the system's qualities. Finally, we draw some conclusions about future research.

2 DESIGN

In this section we first present our motivations and ideas for this approach, and then we detail the algorithm by describing the top level system components.

2.1 IDEAS

Our goal is to implement the inductive learning methodology for attribute–based spaces. We decide to use the VL_1 language as the choice for all the input, the output, and the processing mechanisms. This uniformity could provide for incremental learning, ability to process hypothesis (and, therefore, for applicability for knowledge maintenance), and ease of understanding of the knowledge reformulation process itself.

To deal with the huge search spaces, we must provide as much heuristics as possible. Therefore, in addition to the standard correctness measure of the proposed descriptions (measured by completeness and consistency) and some learning bias, we use the special operators of the inductive methodology and provide for adjustments to their applications in order to apply those that are more likely to improve the descriptions (for example, we make specializing operators more applicable to inconsistent descriptions, *etc.*).

Finally, there is the question of the architecture itself. Because we are dealing with descriptions that are being changed by means of operators, we propose to use a production system framework. In other words, we want to have three separate components: current description(s), the operators, and an inference mechanism. Such a top–level separation would allow us to easily apply similar ideas to other problems: only the module containing the domain–specific knowledge would have to be changed (see Figure 1).

Figure 1: The architecture.

Two new problems that arise in this context are: manageability of the database size and selection of appropriate operator–state pairs for firing. The latter problem does not exist in a standard production system, where the set of candidates is determined by matching operator conditions. In our case, there are hardly any conditions, the operators basically describe the actions to be performed on descriptions. Moreover, we actually want to avoid any operators that would require extensive pattern matching for being a candidate — we do not use the *inductive resolution* rule. For the others that require some matching, we devise special efficient techniques. For example, we have an operator that uses an uncovered positive event e^+ to build a new rule: e^+ ::> *decision*. However, this operator does not require any pattern matching since each rule knows, at any given moment, the set of incomplete (and inconsistent) events (see Janikow 1991). With these operators, we decide to use stochastic selection for firing, with heuristically adjusted probabilities.

The size manageability problem is of a different nature: due to the huge size of the search space, it is impossible to retain all previously explored descriptions, and therefore there is a possibility for the number of actually explored states, large to start with, to become infinite because of loops. Under such conditions, the performance of the system would depend on the control mechanism selecting both the appropriate operators and states for further exploration. Here, heuristics play an important role. Between potentially unmanageable database growth and a single state exploration, we decide to take an intermediate approach: retain only a fixed number of states for further exploration. The mechanism that decides the set of states to be retained is again stochastic, but employs evaluation based on problem specific heuristics.

With these ideas, a straightforward algorithm would be one similar to *bestFirstSearch*, with the additional mechanisms restricting the number of retained states. Such an approach would provide a framework for competition among different directions of exploration. However, we decide to provide cooperative mechanisms as well. They are to provide for information exchange among the different search directions. To implement these ideas, we decide to use control mechanisms of genetic algorithms, which exhibit the desired properties.

Genetic algorithms (GAs) are adaptive methods of searching solution spaces. They belong to the class of probabilistic algorithms, with a unique search method that is relatively insensitive to local traps — a result of multi–directional search with special mechanisms for information formation and exchange. This is achieved by maintaining a population of proposed solutions (*chromosomes*) for a given problem. Each solution is represented in a fixed alphabet (usually binary) with an established meaning. This population iteratively 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 the 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.

Each iteration, called a reproduction cycle, is performed in three 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 and are reevaluated. This mating process is based on the application of two operators: mutation and crossover. Mutation introduces random variability into the population, and crossover exchanges random pieces of two chromosomes.

Goal descriptions of supervised learning are of unpredictable length. To deal with this problem, there are two different GA approaches. Michigan approach, known as CS for classifier systems, uses populations consisting of fixed length elements, and the solution is represented by a set of chromosomes from the population. Pitt approach, known as LS for learning systems, represents an extension of the traditional fixed–length chromosome approaches. Here, variable length chromosomes are used to represent proposed solutions individually.

In the Pitt framework, which seems more suitable for this class of problems, the two most important approaches come from Koza, who uses Lisp programs as the underlying representation and provides operators on Lisp trees (Koza 1989), and Spears with DeJong, who use only the traditional operators of mutation and crossover (Spears & DeJong 1990). In the CS framework, which seems more suitable for action planning than for concept learning (Liepins & Wang 1991), the most important approaches come from Grefenstette (Grefenstette 1991) and Booker (Booker 1989).

Our ideas are different from those, which start with a genetic algorithm (either alone or embedded in a classifier system) and attempt to apply it to supervised concept learning. In our case, we start with the problem solving methodology (the inductive operators and heuristics), and only then use genetic algorithms to provide the necessary control mechanisms needed. Although the idea of using the GA control to guide problem–specific search is unique in nature, it should be noted that some of the recent advances in CS applications to concept learning lean in the same direction — use of high level problem–specific operators (Grefenstette 1991).

2.2 DATABASE

As we mentioned before, we attempt to directly implement the inductive learning methodology in the VL_1 language. Because we want to work with proposed descriptions as the entities on which to operate, we use the LS approach. Then, a single state of our database (a chromosome of a genetic algorithm) is a complete potential solution — a set of rules. For simplicity of presentation, we assume that we are dealing with single concepts and that we are to learn only the description of the concept — the space not covered by this description is assumed to represent the complement of the concept.

This assumption provides for both architectural simplicity and avoids the problem of multiple matches and no matches normally occurring with rule–based systems. The architectural simplicity is based on the fact that all rules of a single state (of the form *complex* ::> *decision*) are associated with the same decision, which does not have to be explicitly expressed. For a discussion on possible relaxations of these assumptions, refer to (Janikow 1991).

The database contains states, each of which is a potentially feasible solution and can now be expressed as a set of VL_1 complexes. The database size remains fixed (as a parameter of the system). Initially, it must be filled with some descriptions. Such an initialization might be totally random (as is normally the case in genetic algorithms), or it might incorporate some task–specific knowledge like initial hypotheses, background knowledge, or the positive events.

2.3 CONTROL

The control mechanism is similar to that of genetic algorithms. Its main task is to retain the database size fixed, determine the exploration directions, and match states with operators for firing to generate new states.

The first two objectives are accomplished by using the standard GA selection mechanism: all current states are evaluated using the correctness measures and some learning criteria (from the domain–specific knowledge module), and a new database is formed by stochastic selection proportional to such evaluations. In other words, the new database is likely to contain single or even multiple copies of above–average states and to exclude below–average states.

The last objective, matching the states with the operators, is performed under supervision of the control module but is actually done by the active operators of the knowledge module.

2.4 DOMAIN–SPECIFIC KNOWLEDGE

This module contains all the problem–specific knowledge about the problem, and it includes (see Figure 1) the inductive operators, heuristics used to evaluate individual descriptions and their substructures, and heuristics used to determine the application of these operators.

2.4.1 Inductive Operators

The operators transform the descriptions to new (possibly better) states in the search space. Since the system operates in the problem space, the operators are those of the inductive methodology operating in attribute–based spaces. These operators are those competitive, since they change the states one at a time in hope of producing better offspring. They are similar to mutation in classical genetic algorithms, yet they differ substantially since the original idea of mutation was just a random variation. We also provide some operators that are to take advantage of the population of descriptions and produce new states by combining information contained in more than one (we used two) states. These operators follow the idea of crossover in genetic algorithms.

A more important classification of the operators can be done with respect to the syntactic level of applicable structures. Accordingly, we have condition, rule, and rule set operators. Moreover, we also classify the operators according to the relationship between the original state and the offspring. Here, we distinguish generalizing, specializing, and independent operators, where the latter are those whose action can be either of generalization or specialization, or whose action changes only the description without affecting the coverage.

The operators currently used are listed in Table 1, where "I", G", and "S" stand for independent, generalizing, and specializing. The newly introduced cooperative operators are "rules exchange" and "rules copy", which exchange and copy rules between two descriptions. The newly introduced "rule split" does not change the description's coverage, but rather change the expression to possibly enable exploration in new directions. The "new event" adds a new rule with the complex being a previously uncovered positive event. The "rules generalization" and "specialization" replace selected rules of a description with their most specific generalization and most general specification. The "star" operator implements the star mechanism used in AQ, which replaces an inconsistent rule with

Table 1: The currently used operators.

Syntactical Level	Type	Name
Rule set level	I	rules exchange
	G	rules copy
	G	new event
	G	rules generalization
	S	rules drop
	S	rules specialization
Rule level	I	rule split
	G	condition drop
	G	turning conj. into disj.
	S	condition introduce
	S	star
Condition level	G	reference extension
	S	reference restriction

the minimal set of rules that are consistent with respect to a negative event. Finally, the "reference extension" and "restriction" change conditions (VL_1 selectors) differently for different types of domains. For example, the former acts as *interval closing* when applied to linear domains.

2.4.2 Evaluation Heuristics

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 as cost of attributes, length of descriptions, their generality, and so on. Moreover, in a more general setting the current knowledge could be evaluated by its average performance as while monitoring an on–line system or guiding an autonomous object, but we leave these consideration for the future.

Combining multiple criteria in a single evaluation measure is very difficult and critical for the convergence problem. In our case, we need to combine three such values. We can ease this task by replacing the completeness and consistency measures with a standard single measure of correctness. Then, we can combine correctness and cost by the experimental formula:

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

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. The cost of a description is measured by its complexity, which combines the number of rules and conditions as follows: $complexity = 2 \cdot \#rules + \#conditions$.

The above evaluation measure provides for a controlled bias with respect to descriptions' complexity. Correctness is defined as the average of completeness and consistency, whose definitions are presented in Table 2. e^+ and e^- are the number of positive/negative training events currently covered by a rule; ε^+ and ε^- are the number of such events covered by a rule set; E^+ and E^- are the total number of such events.

Table 2: Completeness and consistency measures.

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

The primary reason for the cost accommodation is to force differentiation between the same or similarly covering rule sets but of different complexity. The effect of the very slowly raising f is that initially cost's influence is small to promote deeper space exploration and only increases at later stages in order to minimize complexity.

2.4.3 Operator Application Heuristics

Each operator is given some initial probabilities from two separate groups: application and selection probabilities. The latter are fixed while the former adjust appropriately to different VL_1 structures. For example, on the rule set level, the generalizing operators increase their applicability to incomplete descriptions and the specializing operators increase their applicability to descriptions that are highly inconsistent. On the rule level, the same happens based on the completeness and consistency of the individual rules. Finally, we do not apply these heuristics on the condition level since it is difficult to decide which of the conditions contribute to the incompleteness and inconsistency.

2.5 ALGORITHM

The algorithm uses the described components in a framework that resembles the production system. A single run is a series of iterations, each of each is accomplished by evaluating the current database, selecting a new one, and generating new states by means of operators with dynamically adjusted application probabilities. This iterative process continues until some termination condition is met. Such a condition may include time constraints, generation of a description satisfying some goals, and so on.

3 EXPERIMENTS

In this section we present results of a few experiments aimed at evaluating the system's quantitative and qualitative capabilities. These results should not be treated in absolute terms since they were obtained with a prototype implementation without any extensive parameter tuning.

For the experiments, we decided to use two standard data sets: DNF descriptions and multiplexers. Because of that, the results can be compared to those of other systems. Learning random DNFs has become a standard way of evaluating learning algorithms. For our experiment, we used random one– and two–disjunct descriptions, with each such a disjunct being a conjunction of up to three selectors. All descriptions were spanned by six three–valued attributes, giving $3^6 = 729$ possible events. The number of possible rules here was $7^6 = 117649$ (with the internal disjunction), which gives a total of $2^{117649} \simeq 10^{39216}$ competing concept descriptions. As big number as it seems for the system that nearly explicitly searches the concept space, results show that the task was not so difficult after all. An average learning time was about ten seconds on a SPARC2 station.

Multiplexers represent another widely used experimental data. Each multiplexer is a specific case of the more general DNFs. For each integer $k = 1, 2, \ldots$ there is a multiplexer boolean function defined in the following way: the function's 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 is specified by the address bits. This function can be expressed in VL_1 (Wilson 1987). The f_6 multiplexer has relatively small event space (64) and search space ($\simeq 10^{243}$), and the cost of the solution (using the definition of section 2.4.2) is twenty. The f_{11} multiplexer has quite larger event space (2048) and search space ($\simeq 10^{59049}$), but a relatively similar solution complexity (forty eight). The average learning time for f_6 and f_{11} was about ten seconds and twenty minutes, respectively.

For all of our experiments we assumed the crisp concept representation. In other words, a description recognizes an event only if the intersection of the description and the event is not empty. This follows our assumption of learning only the concept description, and treating its negation as the description of the negated concept. All numerical results are averages of ten independent runs.

3.1 QUANTITATIVE RESULTS

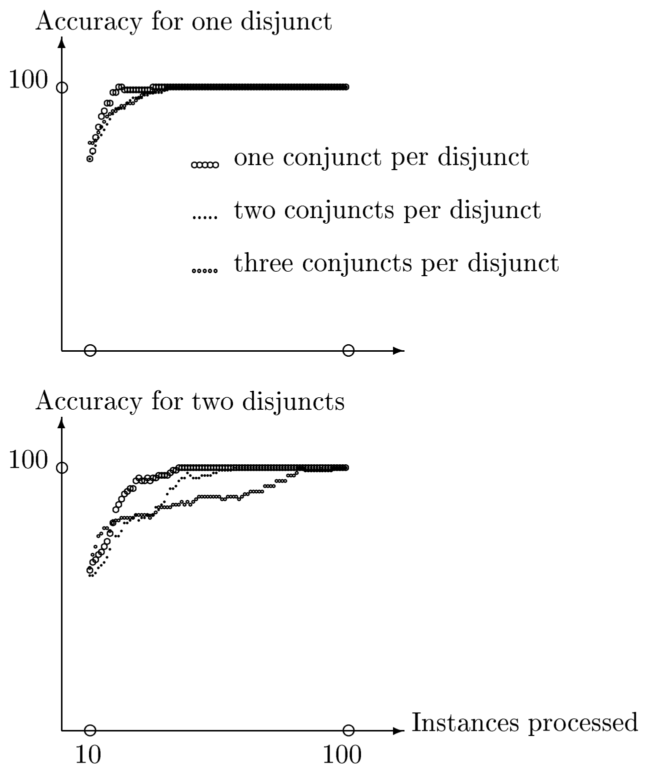

Figure 2: Batch–incremental results on DNF data.

The reported DNF results (Figure 2) represent batch–incremental learning curves using the same experiment as that reported in (Spears & DeJong 1990) for ID5R and GABIL: the 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 $n < 10$. The results indicate that the learning accuracy decreases with the increasing concept complexity, or that the system requires more training data to achieve similar recognition rate for more complex goal descriptions. These results and observations are similar to those for ID5R and GABIL.

The multiplexers were tested in a batch mode: the value of the learning curve at $n\%$ was computed as the recognition rate on the unseen events after training with $n\%$ of the available events. The resulting curves are shown in Figure 3a. When the number of processed descriptions is considered, the f_6 learning is similar to that reported in (Koza 1989). When the accuracy is considered, the f_{11} multiplexer results are similar to those of C4 (Quinlan 1988). The results indicate good generalization properties of our system, and they support the previous observation that the system's learning accuracy grows faster (with respect to the percentage of training events) for simpler goal concept descriptions — f_{11} has much lower complexity with respect to its event and search spaces). In (Janikow 1992) we show that these results can be improved by controlling the learning bias.

3.2 QUALITATIVE RESULTS

Because our goal was to be able to both produce high recognition rate and highly comprehensive output, we performed few experiments to evaluate the latter qualities as well. The only results we present here refer to the multiplexer data.

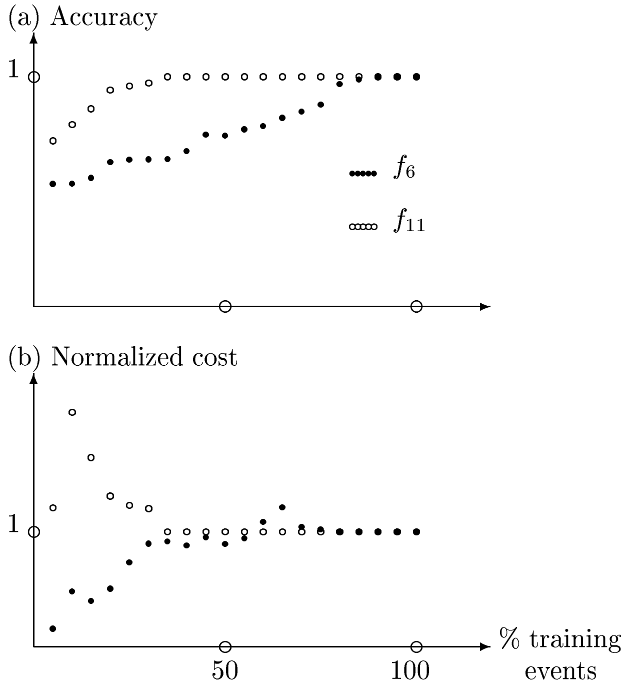

Figure 3: Batch learning curves on the multiplexer.

Figure 3b represents the average size of the batch generated descriptions for both multiplexers and are normalized with respect to the complexity of the sought descriptions. Two interesting observations can be stated. Firstly, for too few training events, the solutions were far different from those sought, but for a sufficient number of training events the system was generating descriptions very similar to the goals in terms of complexity. For trainings producing a perfect accuracy (see Figure 3a), the generated knowledge was actually precisely the same as the goal concept — no redundant descriptions were retained. Secondly, it is interesting to observe that for too few training events, the generated f_6 descriptions were too general and f_{11} descriptions were too specific. This behavior is a result of the learning bias introduced by the currently used evaluation (section 2.4.2). These differences are also visible in Figures 4 and 5, and are further studied in (Janikow, 1992).

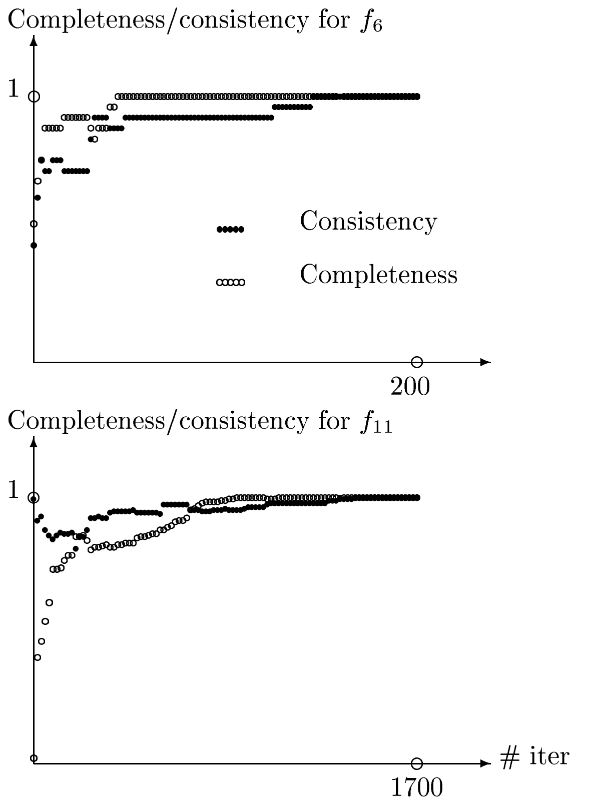

Figure 4: Tracing the consistency and competeness of the best database descriptions.

Figure 4 traces two individual runs, one for f_6 and one for f_{11}, both of which resulted in finding the exact goal descriptions. These traces illustrate the differences in the system's performance on these two problems. The f_6 descriptions (this training with 80%) were being built by producing first complete solutions, and then specializing them for consistency. This observation supports the previous one that these descriptions were initially built too general. The f_{11} descriptions (this training with 30%) were being built by rather producing consistent solutions first and then generalizing them. In (Janikow 1992) we show that this property can be changed by adjusting the bias, which can often lead to improving both learning time and accuracy. Finally, the reasons for the higher here time complexity for the latter data can be explained in Figure 5, where we trace the complexity of the same currently best descriptions. Clearly, the amount of work necessary for the f_{11} multiplexer was quite larger because of the larger number of rules being processed simultaneously.

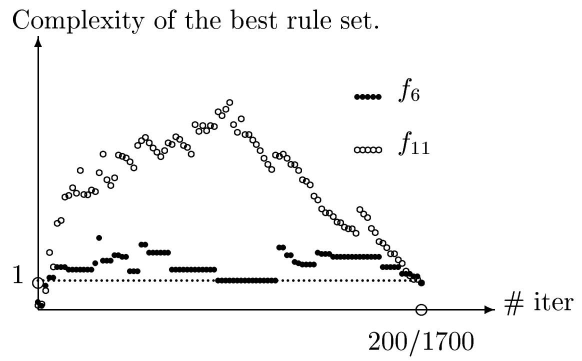

Figure 5: Tracing the cost of the best database description.

4 SUMMARY

We described here a full–memory system for learning concept descriptions from examples of attribute–based spaces. This system is designed following the ideas of production systems. It uses the operators of the inductive learning methodology. To control the search process, we used the ideas of genetic algorithms: the size of the database remains fixed by applying selection that favors better intermediate descriptions, and the operator applications are stochastic. We also included few additional operators providing cooperation among different search directions. The system operates in the VL_1 rule–based language.

Application of the inductive operators and the high level language allow for utilization of powerful problem–specific heuristics. Application of the same input and output language should allow for incremental learning and for ability to process initial hypotheses. These qualities will be tested in the future. Processing mechanisms defined at the same level make it easier to justify the process of knowledge reformulation. The system achieves not only high recognition rates, but it also produces knowledge of low complexity.

Much work remains to be done here. The most important are methods of dealing with the huge parameter space, controlling the learning bias, and relaxing the current restrictions and assumptions.

References

Booker, L.B. (1989). "Triggered Rule Discovery in Classifier System". In J.D. Schaffer, ed., *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann.

Spears, W.M. & DeJong, K.A. (1990). "Using Genetic Algorithms for Supervised Concept Learning". In J.J. Grefenstette, ed., *Proceedings of the Second International Conference on Tools for AI*, IEEE Computer Society Press.

Grefenstette, J. (1991). "Lamarkian Learning in Multi-agent Environments". In R.K. Belew and L.B. Booker,eds., *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann.

Liepins, G.E. & Wang, L.A. (1991). "Classifier System Learning of Boolean Concepts". In R.K. Belew and L.B. Booker,eds., *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann,

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, 1991.

Janikow, C.Z. (1992). "Some Experiments with a Stochastic Production System". In *ARTIFICIAL INTELLIGENCE: Methodologies, Systems, Applications*, Vol. 5, North–Holland, to appear.

Koza, J.R. (1989). "Hierarchical Genetic Algorithms Operating on Populations of Computer Programs". In *Proceedings of the International Joint Conference on Artificial Intelligence.*

Michalski, R.S. (1983). "Theory and Methodology of Inductive Learning". In R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds., *Machine Learning I*, Morgan Kaufmann.

Michalski, R.S. (1986). "Understanding the Nature of Learning". In R.S. Michalski, J.G. Carbonell and T.M. Mitchell, eds., *Machine Learning II*, Morgan Kaufmann.

Quinlan, J.R. (1988). "An Empirical Comparison of Genetic and Decision–tree Classifiers". In *Proceedings of the Fifth International Conference on Machine Learning*, Morgan Kaufmann.

Wilson, S. (1987). "Classifier Systems and the Animat Problem". In Machine Learning, 3:2.