Genetic Approach to Constructive Induction Based on Non-algebraic Feature Representation[1]

Leila S. Shafti and Eduardo Pérez

Universidad Autónoma de Madrid, E–28049 Madrid, Spain
leila.shafti@ii.uam.es, eduardo.perez@ii.uam.es
WWW home page: http://www.ii.uam.es/

Abstract. The aim of constructive induction (CI) is to transform the original data representation of hard concepts with complex interaction into one that outlines the relation among attributes. CI methods based on greedy search suffer from the local optima problem because of high variation in the search space of hard learning problems. To reduce the local optima problem, we propose a CI method based on genetic (evolutionary) algorithms. The method comprises two integrated genetic algorithms to construct functions over subsets of attributes in order to highlight regularities for the learner. Using non-algebraic representation for constructed functions assigns an equal degree of complexity to functions. This reduces the difficulty of constructing complex features. Experiments show that our method is comparable with and in some cases superior to existing CI methods.

1 Introduction

Most machine learning algorithms based on similarity attain high accuracy on many artificial and real-world domains such as those provided in Irvine databases [1, 2]. The reason for high accuracy is that the data representation of these domains is good enough to distribute instances in space in a way that cases belonging to the same class are located close to each other. But for hard concepts due to low-level representation, complex interactions exist among attributes. The *Interaction* means the relationship between one attribute and the concept depends on other attributes [3]. Due to attribute interaction of hard concepts, each class is scattered through the space and therefore a similarity based learning algorithm fails to learn these concepts. Such problems have been seen in real-world domains such as protein secondary structure [4]. Attribute interaction causes problems for almost all classical learning algorithms [5].

Constructive induction (CI) methods have been introduced to ease the attribute interaction problem. Their goal is to automatically transform the original representation space of hard concepts into a new one where the regularity is more

[1] This work has been partially supported by the Spanish Interdepartmental Commission for Science and Technology (CICYT), under Grants numbers TIC98-0247-C02-02 and TIC2002-1948.

apparent [6, 7]. This goal is achieved by constructing features from the given set of attributes to abstract the relation among several attributes to a single new attribute.

Most existing CI methods, such as Fringe, Grove, Greedy3 [8], LFC [9], and MRP [10] are based on greedy search. These methods have addressed some problems. However, they still have an important limitation; they suffer from the local optima problem. When the concept is complex because of the interaction among attributes, the search space for constructing new features has more variation. Because of high variation, the CI method needs a more global search strategy such as Genetic Algorithms [11] to be able to skip local optima and find the global optima solutions. Genetic Algorithms (GA) are a kind of multi-directional parallel search, and more likely to be successful in searching through the intractable and complicated search space [12].

There are only a few CI methods that use genetic search strategy for constructing new features. Among these methods are GCI [13], GPCI [14], and Gabret [15]. Their partial success in constructing useful features indicates the effectiveness of genetic-based search for CI. However, these methods still have some limitations and deficiencies as will be seen later. The most important ones are their consideration of the search space and the language they apply for representing constructed features.

This paper presents a new CI method based on GA. We begin by analyzing the search space and search method, which are crucial for converging to an optimal solution. Considering these we design GABCI, Genetic Algorithm Based CI and present and discuss the results of our initial experiments.

2 Decomposition of Search Space

A CI method aims to construct features that highlight the interactions. For achieving this aim, it has to look for subsets of interacting attributes and functions over these subsets that capture the interaction. Existing genetic-based CI methods, GCI, and GPCI, search the space (of size 2^{2^N} for N input attributes in Boolean domain) of all functions that can be defined over all subsets of attributes. This enormous space has lots of local optima and is more difficult to be explored.

To ease the searching task, we propose to decompose the search space into two spaces: S the space of all subsets of attributes, and F_{S_i} the space of all functions defined over a given subset of attributes S_i (Fig. 1). The decomposition of the search space into two spaces allows a specific method to be adjusted for each search space. This strategy divides the main goal of CI into two easier sub-goals: finding the subset of interacting attributes (*S-Search*) and looking for a function that represents the interaction among attributes in a given subset (F_{S_i}-*Search*).

The genetic-based CI method Gabret also divides the search space into two spaces. It applies an attribute subset selection module before and independently from the feature construction module to filter out irrelevant attributes. However, when high interaction exists among attributes, attribute subset selection module

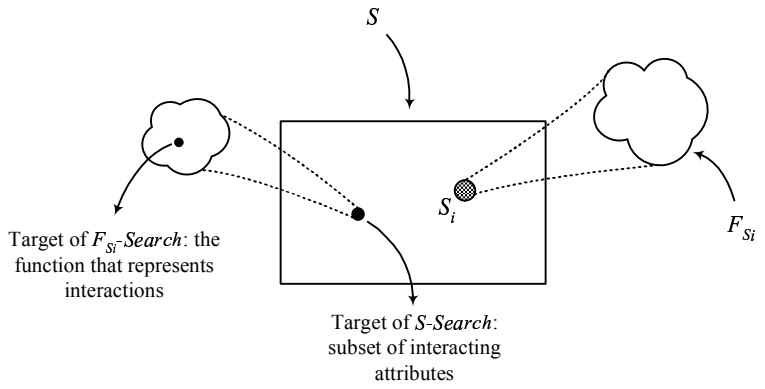

Fig. 1. The decomposition of space of all functions defined over all subsets of attributes. Let S to be the space of all subsets of attributes, $S_i \in S$, and F_{S_i}, the space of all functions defined over subset S_i.

cannot see the interaction among primitive attributes. Therefore, this module often excludes some relevant attributes from the search space that should be used later for constructing new features.

The two spaces of attribute subsets and functions over subsets are related and each one has a direct effect on the other. In order to find a good subset of attributes it is necessary to see the relation among attributes. An existing interaction among attributes in subset S_i is not apparent unless a function f_{S_i}, that outlines this interaction, is defined over the proper subset of attributes. Similarly, a function can highlight the interaction among attributes only when it is defined over the related attributes. If the chosen subset of attributes is not good enough, the best function found may not be helpful. Therefore, the two tasks, *S-Search* and F_{S_i}*-Search*, should be linked to transfer the effects of each search space into the other.

3 GABCI

Our CI method GABCI consists of two search tasks *S-Search* and F_{S_i}*-Search* as described in Sect. 2. Since both spaces S and F_{S_i} grow exponentially with the number of attributes and have many local optima, GABCI applies GA to each search task. Thus, GABCI is a composition of two GA with two populations: a population of attribute subsets (for *S-Search*) and a population of functions (for F_{S_i}*-Search*). The goal is to converge each population to an optimal solution by means of genetic operators.

Recalling the need to keep the connection between two tasks, we propose to have F_{S_i}*-Search* inside *S-Search*. That is, for each subset S_i in *S-Search*, in order to measure its goodness, GABCI finds the best function defined over this subset using F_{S_i}*-Search*. Then the fitness value of the best individual in F_{S_i}*-Search*

determines the fitness of the subset of attributes S_i in *S-Search* (Fig. 2). By this strategy we can maintain the relation between two tasks and their effects to each other, while improving each of them.

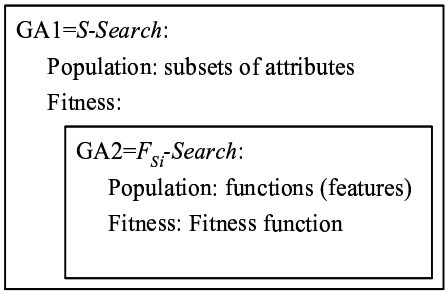

Fig. 2. The GABCI's framework: the fitness of each individual of *S-Search* is determined by performing F_{S_i}-*Search* and calculating the fitness of the best individual of GA2.

Among the several factors that need to be defined carefully when designing a GA, the two main ones are the representation language of individuals and the fitness function. The language has an impact on the way the search is conducted. An inadequate language can make the search fruitless by hiding solutions amongst many local optima. Fitness function has the role of guiding the method toward solution. If this function assigns an incorrect fitness value to individuals the method will be misguided to local optima. In the following, we explain the representation language and the fitness function used for each space in GABCI.

3.1 Individuals Representation

For *S-Search*, we represent individuals (i.e., attribute subsets) by bit-strings; each bit representing presence or absence of an attribute in subset. F_{S_i}-*Search* is more complicated than *S-Search*. Recalling that the aim of F_{S_i}-*Search* is to construct a function over S_i, the given subset of attributes, to represent interactions; the representation language should provide the capability of representing all complex functions or features of interests. This representation has an important role in convergence to optimal solution.

There are two alternative groups of languages for representing features: algebraic form and non-algebraic form. By algebraic form, we mean that features are shown by means of some algebraic operators such as arithmetic or Boolean operators. Most genetic-based CI methods like GCI, GPCI, and Gabret apply this form of representation using parse trees [16]. GPCI uses a fix set of simple operators, AND and NOT, applicable to all Boolean domains. The use of simple operators makes the method applicable to a wide range of problems. However, a complex feature is required to capture and encapsulate the interaction using

simple operators. Conversely, GCI and Gabret apply domain-specific operators to reduce complexity; though, specifying these operators properly cannot be performed without any prior information about the target concept.

In addition to the problem of defining operators, an algebraic form of representation produces an unlimited search space since any feature can be appeared in infinite forms. As an example see the following conceptually equivalent functions that are represented differently:

$$X_1 \wedge X_2$$

$$((X_1 \wedge X_2) \vee \overline{X}_1) \wedge X_1$$

$$((((X_1 \wedge X_2) \vee \overline{X}_1) \wedge X_1) \vee \overline{X}_2) \wedge X_2 \ .$$

Therefore, a CI method needs a restriction to limit the growth of constructed functions.

Features can also be represented in a non-algebraic form, which means no operator is used for representing the feature. For example, for a Boolean attribute set such as $\{X_1, X_2\}$ an algebraic feature like $(X_1 \wedge \overline{X}_2) \vee (\overline{X}_1 \wedge X_2)$ can be represented by a non-algebraic feature such as $\langle 0110 \rangle$, where the j^{th} element in $\langle 0110 \rangle$ represents the outcome of the function for j^{th} combination of attributes X_1 and X_2 according to the following table:

X_1	X_2	f
0	0	0
0	1	1
1	0	1
1	1	0

A non-algebraic form is simpler to be applied in a CI method since there is no need to specify any operator. Besides, all features have equal degree of complexity when they are represented in a non-algebraic form. For example two algebraic features like $f(X_1, X_2) = (X_1 \wedge X_2) \vee (\overline{X}_1 \wedge \overline{X}_2)$ and $f'(X_1, X_2) = X_1$ are represented in non-algebraic form as $R(X_1, X_2) = \langle 1001 \rangle$ and $R'(X_1, X_2) = \langle 0011 \rangle$, which are equivalent in terms of complexity. Since for problems with high interaction, CI needs to construct features more like the first feature in example, which are complex in algebraic form, a non-algebraic representation is preferable for these methods. Such kind of representation is applied in MRP [10], a learning method that represents features by sets of tuples using multidimensional relational projection. However, MRP applies a greedy method for constructing features. As the search space for complex problems has local optima, MRP fails in some cases.

Non-algebraic representation also eases the traveling through the search space by GA. Changing one bit in an individual as non-algebraic function by genetic operators can produce a large change in the function. For example if the function is $X_1 \vee X_2$, represented as $\langle 0111 \rangle$, changing the last bit gives $\langle 0110 \rangle$, that is $(X_1 \wedge \overline{X}_2) \vee (\overline{X}_1 \wedge X_2)$, which is very different from its parent in terms of algebraic

form of representation. So this form of representation results in more variation after performing genetic operators and can provide more useful features in hard problems.

Thus we apply a non-algebraic form of representation for individuals in F_{S_i}-*Search*. Without losing generality, suppose that attributes have Boolean domain. For each subset S_i of size k of the population in *S-Search*, a population of bit-strings of length 2^k is generated in F_{S_i}-*Search*. The j^{th} bit of the string represents the outcome of the function for j^{th} combination of attributes in the subset S_i.

3.2 Fitness Function

The fitness of each individual S_i in *S-Search* depends on performance of F_{S_i}-*Search* and the fitness value of the best constructed feature. If the fitness value of the best feature F in F_{S_i}-*Search* is $Fitness_{S_i}(F)$, then the fitness of S_i is calculated as:

$$Fitness(S_i) = Fitness_{S_i}(F) + \epsilon \frac{W(S_i)}{|S_i|} \tag{1}$$

where $W(S_i)$ is the number of ones in bit string S_i, and $|S_i|$ is the length of the string. The first term of the formula has the major influence on fitness value of the individual S_i. The last term roughly estimates the complexity of the new feature, to follow Occam's razor bias, by measuring the fraction of attributes participating in the feature. It is multiplied by $\epsilon = 0.01$ to have effect only when two subsets S_j and S_k perform equally well in F_{S_i}-*Search*; that is, $Fitness_{S_j}(F') = Fitness_{S_k}(F'')$. Therefore, between two equally good subsets, this formula assigns a better fitness value to the smaller.

The fitness of individuals in F_{S_i}-*Search* can be determined by a hypothesis-driven or a data-driven evaluation. A hypothesis-driven evaluation can be done by adding the new function to the list of attributes and updating data, to then apply a learner like C4.5 [17] and measure its accuracy. GCI and Gabret apply this kind of evaluation. A hypothesis-driven evaluation relies on the performance of the learner. Therefore, if the learner assigns inaccurately a low or high fitness value to individuals, it will guide the algorithm to a local solution. Another possibility is to apply a data-driven evaluation formula like conditional entropy [18] to measure the amount of information needed to identify the class of a sample in data knowing the value of the new feature. A data-driven approach only depends on data, and therefore, is more reliable than a hypothesis-driven approach for guiding GA. Moreover, in GA computation time of fitness function is very important since this function is called for every individual during each GA generation. The current version of GABCI applies the latter evaluation function as explained below. These fitness functions will be compared empirically in Sect. 4.2.

To prevent the system from constructing features that fits well with training data but does not fit with unseen data (overfitting), we first shuffle and divide the training data into three sets T_1, T_2, and T_3. Then fitness of the new feature,

F, is measured by the following formula:

$$Fitness(F) = \frac{1}{3}\sum_{i=1}^{3} Entropy(T_i|F) \quad (2)$$

where $Entropy(T|F)$ is the conditional Entropy [18]. Figure 3 summarizes GABCI's algorithm.

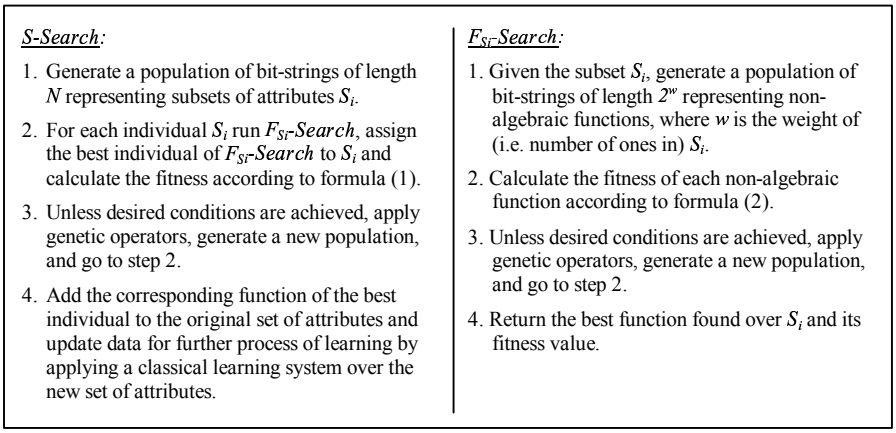

Fig. 3. The GABCI's algorithm

4 Experimental Results and Analysis

We compare GABCI with other CI methods based on greedy search by conducting experiments on artificial problems known as difficult concepts with high interactions. These problems are used as prototypes to exemplify real-world hard problems. Therefore, similar results are expected for real-world problems. We also analyze the behavior of GABCI with data-driven and hypothesis-driven fitness functions.

4.1 Comparison with Other Methods

The concepts used for these experiments are defined over 12 Boolean attributes $a_1, \ldots, a_{12}$. These concepts are parity of various degrees, multiplexor-6, n-of-m (exactly n of m attributes are one), Maj-n-m (among attributes in $\{a_n, \ldots, a_m\}$, at least half of them are one), and nm-4-5 (among attributes in $\{a_4, \ldots, a_{10}\}$, 4 or 5 of them are one). A high interaction exists among relevant attributes of these concepts.

Each experiment was run 10 times independently. For each experiment, we used 10% of shuffled data for training and kept the rest (90% of data) unseen for evaluating the final result. When GABCI finished, its performance is evaluated by the accuracy of C4.5 [17] on modified data after adding the constructed feature; using 10% data as training and 90% unseen data as test data.

For implementing GA, we used PGAPACK library [19] with default parameters except those indicated in Table 1.

Table 1. GA's modified parameters

GA Parameter	***S-Search***	F_{S_i}***-Search***
Population Size	20	20
Max Iteration	100	300
Max no Change Iter.	25	100
Max Similarity Value	80	80
No. of Strings to be Replaced	18	18
Selection Type	Proportional	Proportional
Crossover Type	Uniform	Uniform
Mutation and/or Crossover	And	And

Table 2 presents a summary of GABCI's accuracy and its comparison to the other systems' accuracy. The last column indicates the average accuracy after testing the result of GABCI by C4.5 using unseen data. The results are compared with C4.5 and C4.5-Rules [17], which are similarity-based learners, Fringe, Grove and Greedy3 [8], and LFC [9], which are greedy-based CI methods that apply algebraic form for representing features and MRP [10] which is a greedy CI method with a non-algebraic form of representation. In the third column of the table, we show the best results among C4.5-Rules, Fringe, Grove, Greedy3 and LFC, as reported in [10]. Numbers between parentheses indicate standard deviations.

Table 2. The accuracy's result average

Concept	**C4.5**	**The best result**	**MRP**	**GABCI + C4.5**
Parity-4	71.4 (17.7)	**100 (fringe)**	**100 (0.0)**	**100 (0.0)**
Parity-6	49.2 (2.8)	78.7 (fringe)	**99.8 (0.5)**	**99.8 (0.5)**
Parity-8	47.2 (0.2)	49.4 (C4.5-Rules)	88.6 (1.3)	**90.8 (1.9)**
MX-6	98.2 (2.3)	**100 (fringe)**	99.9 (0.4)	99.8 (0.5)
5-of-7	83.2 (1.8)	93.8 (grove)	**98.6 (2.6)**	98.4 (0.8)
6-of-9	78.5 (1.8)	**88.7 (grove)**	87.4 (2.1)	79.4 (2.7)
Maj-4-8	**100 (0.0)**	**100 (grove)**	**100 (0.0)**	**100 (0.0)**
Maj-3-9	88.3 (1.4)	94.0 (grove)	97.3 (1.1)	**98.3 (1.3)**
***nm*-4-5**	80.0 (2.7)	90.5 (fringe)	**98.4 (1.1)**	98.1 (2.3)

GABCI's performance is comparable with other CI methods, and sometimes better. Although its accuracy is not always the highest, it is among the best ones. The results show that GABCI has the capacity to be improved to give better accuracy (for instance, by carefully adjusting parameters). The large improvement in accuracy obtained for parity concepts shows that F_{S_i}-*Search* correctly guides the *S-Search* toward a proper subset, and once this subset is found, then F_{S_i}-*Search* constructs a feature defined over this subset that can help the learner to achieve a higher accuracy. In fact, in all experiments GABCI successfully found the subset of interacting attributes, except in one of the 10 experiments with concept *nm*-4-5.

The low performance of GABCI in 6-of-9 concept is due to overfitting. GABCI finds the relevant attributes but generates a function over these attributes, which fits well with training data only, resulting in low accuracy on unseen data; though it is still slightly better than the similarity-based learner C4.5.

4.2 Analysis of Fitness Functions

We also performed experiments to compare the effect of hypothesis-driven and data-driven fitness functions (see Sect. 3.2) on the performance of F_{S_i}-*Search*.

We used C4.5 [17] for hypothesis-driven fitness function. The training data were shuffled three times, each time divided into two parts of equal size: one for training and one for testing. For evaluating each individual, the new feature was added to the original set of attributes and training and test data were updated. Then C4.5 was performed on training data and the accuracy was estimated by test data. The average error rate of three independent runs of C4.5 over three pairs of training and test data determines the fitness of the constructed feature. As a data-driven fitness function, we used (2) of Sect. 3.2.

Since the aim was to observe the effect of fitness function on feature construction, we ran only F_{S_i}-*Search*, once with data-driven fitness and once with hypothesis-driven fitness. A subset of attributes was given to F_{S_i}-*Search* as input to generate new features. We performed these experiments on parity-8 concept. As before, each experiment was performed 10 times, using 10% of data for training and 90% for final evaluation of F_{S_i}-*Search* by C4.5.

Figure 4 shows the average results of 10 independent runs of F_{S_i}-*Search* for each experiment over different subsets of attributes for parity-8. In this figure the horizontal axis shows the number of attributes in the given subset S_i. For each subset S_i, let $S_i = \{x_1, \ldots, x_i\}$ and $|S_i| = i$. The relevant attributes are $x_1, x_2, \ldots, x_8$.

The vertical axis in Fig. 4.a shows the average error rate after running the learner C4.5 on updated data using constructed feature and in Fig. 4.b, represents the size (number of leaves) of the tree generated after pruning. The horizontal lines in these figures show the base-line performance on original set of attributes.

We see that for $i < 8$, C4.5 as fitness function yields lower error than Entropy, but for $i \geq 8$; that is when subset includes all interesting attributes needed for constructing feature, Entropy performs better. The tree size in Fig. 4.b indicates

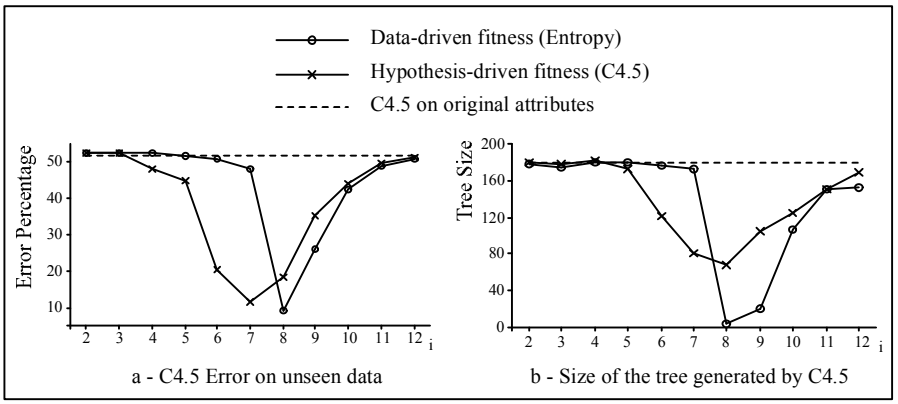

Fig. 4. F_{S_i}-*Search* on Parity-8

that for $i \geq 8$, Entropy generates a feature that abstracts interaction among attributes better than C4.5 fitness function; thus, produces a smaller tree. Focusing on $i = 8$, that is when the best subset (subset of all relevant attributes) is given to F_{S_i}-*Search*, reveals that C4.5 misleads GA to a local optimum and Entropy behaves better.

The reason for these behaviors of Entropy and C4.5 as fitness is that Entropy measures the fitness of the constructed function without involving other attributes, while C4.5 measures the fitness of the constructed feature in relation with original attributes. When the new function is defined over some relevant attributes, C4.5 can combine this function with other relevant attributes (not in the given subset) to incorporate missing attributes. Hence, C4.5 assigns a better fitness to constructed function and produces a lower error rate on unseen data. However, when the function includes all relevant attributes ($i \geq 8$), combining it with relevant attributes cannot help C4.5 and sometimes causes overfitting. By overfitting we mean the program constructs a function with a high accuracy on training data, as it fits well with this data, but when it is tested by unseen data the error rate is increased. These confirm that a hypothesis-driven function may mislead GA to a local optimum.

Furthermore, due to overfitting produced by C4.5 fitness function, the lowest error rate obtained when using this function is higher than the one obtained using entropy. Though the difference is not high, taking into account that computing the fitness using Entropy takes less time than using C4.5, the former is preferable.

5 Conclusion

This paper has presented a new CI method based on genetic (evolutionary) algorithms whose goal is to ease learning problems with complex attribute interac-

tion. The proposed method GABCI intends to outline the attribute interaction by constructing new functions over subsets of related attributes. Our genetic approach makes GABCI more promising than other methods in constructing adequate features when the search space is intractable and the required features are too complex to be found by greedy approaches.

We showed how CI search space could be decomposed into two different search spaces, which can be treated by two different but integrated genetic algorithms. Other CI methods based on genetic algorithms either consider these two spaces as a whole or treat them separately and independently. The former methods often fall in a local optimum by searching over a huge space with many local optima, that is more difficult to be explored; and the latter loses the relation and effects of two search spaces on each other.

Considering such integrated treatment of the decomposed search space, we designed GABCI, using non-algebraic representation of constructed features. The non-algebraic form of representation provides a finite search space of features. Besides, this form of representation assigns an equal degree of complexity to features regardless of the size or complexity of their symbolic representation. This reduces the difficulty of constructing complex features. Empirical results proved that our method is comparable with and in some cases better than other CI methods in terms of accuracy.

References

1. Blake, C.L., Merz, C.J.: UCI Repository of Machine Learning Databases, [http://www.ics.uci.edu/ mlearn/MLRepository.html]. University of California, Department of Information and Computer Science, Irvine, California (1998)
2. Holte, R.C.: Very simple classification rules perform well on most commonly used datasets. Machine Learning, **11**:63–91i (1993)
3. Freitas, A.: Understanding the crucial role of attribute interaction in data mining. Artificial Intelligence Review, **16(3)**:177–199 (November, 2001)
4. Qian, N., Sejnowski, T.J.: Predicting the secondary structure of globular proteins using neural network models. Journal of Molecular Biology, **202**:865–884 (August, 1988)
5. Rendell, L.A., Seshu, R.: Learning hard concepts through constructive induction: framework and rationale. Computational Intelligence, **6**:247–270 (November, 1990)
6. Dietterich, T.G., Michalski, R.S.: Inductive learning of structural description: evaluation criteria and comparative review of selected methods. Artificial Intelligence, **16(3)**:257–294 (July, 1981)
7. Aha, D.W.: Incremental constructive induction: an instance-based approach. In Proceedings of the Eighth International Workshop on Machine Learning, pages 117–121, Evanston, Illinois (1991) Morgan Kaufmann
8. Pagallo G., Haussler, D.: Boolean feature discovery in empirical learning. Machine Learning, **5**:71–99 (1990)
9. Ragavan, H., Rendell, L.A.: Lookahead feature construction for learning hard concepts. In Proceedings of the Tenth International Conference on Machine Learning, pages 252–259, Amherst, Massachusetts (June, 1993) Morgan Kaufmann

10. Pérez, E. Rendell, L.A.: Using multidimensional projection to find relations. In Proceedings of the Twelfth International Conference on Machine Learning, pages 447–455, Tahoe City, California (July 1995) Morgan Kaufmann
11. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, Michigan (1975)
12. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin, Heidelberg, New York (1999)
13. Bensusan, H. Kuscu I.: Constructive nduction using genetic programming. In Proceedings of the International Conference on Machine Learning, Workshop on Evolutionary Computing and Machine Learning, Bari, Italy (July, 1996) T. G. Fogarty and G. Venturini (Eds.)
14. Hu, Y.: A genetic programming approach to constructive induction. In Proceedings of the Third Annual Genetic Programming Conference, pages 146-157, Madison, Wisconsin, (July, 1998) Morgan Kauffman
15. Vafaie, H., De Jong, K.: Feature space transformation using genetic algorithms. IEEE Intelligent Systems and Their Applications, **13(2)**:57–65 (March–April, 1998)
16. Koza, J. R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, Massachusetts (1992)
17. Quinlan, R.J.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, California (1993)
18. Quinlan, R.J.: Induction of decision trees. Machine Learning, **1(1)**:81–106 (1986)
19. Levine, D.: Users guide to the PGAPack parallel genetic algorithm library. Technical Report ANL-95/18, Argonne National Laboratory (January 1996)