UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior
Departamento de Ingeniería Informática

Multi-feature Construction based on Genetic Algorithms and Non-algebraic Feature Representation to Facilitate Learning Concepts with Complex Interactions

Doctoral dissertation by **Leila Shila Shafti**
under the supervision of **Eduardo Pérez**

June 17, 2008

Contents

List of Figures

List of Tables

Abstract

Attribute selection and feature construction have been used individually or together to change the primitive representation of data into a new representation in order to facilitate learning. Most machine learning algorithms assume attribute independence. Thus, they achieve high accuracy on domains when available domain knowledge provides a data representation based on highly informative attributes. Real-world data are often prepared for purposes other than data mining and machine learning, and therefore, are represented by primitive attributes. Primitive data representation facilitates the existence of *attribute interactions* whose complexity makes the relevant information opaque to most learners. Attribute selection is crucial for highlighting the importance of the interacting attributes to the learner. However, identifying relevant attributes by a feature selection method is not sufficient for learning in presence of complex attribute interactions because regularities are still opaque and difficult to learn. Constructive Induction (CI) has been introduced to ease learning by restructuring the primitive representation. When CI is applied to concepts with complex interactions, feature construction in CI aims to capture and encapsulate interactions into new features to highlight regularities to the learner. Recently, many progresses have been achieved in CI; nevertheless, learners still face serious difficulties to succeed when confronted with complex attribute interactions.

This research aims to ease learning concepts with complex interactions when the only available knowledge about the concept is the primitive training data. The thesis describes the major functional requirements that CI methods must satisfy in order to address this problem. The dissertation proposes a new framework which decomposes the difficult task of CI into smaller and easier tasks. Two methods, DCI and MFE3/GA, are designed based on the above framework and requirements. Empirical evaluations on synthetic and real-world problems show the effectiveness of these methods and the proposed framework in improving learning accuracy. The proposed framework can be used as a model for designing a tool to be integrated in a machine learning toolbox in order to ease learning hard concepts with primitive data.

Resumen

Las técnicas de selección de atributos y construcción de nuevas características han sido utilizadas en forma individual o conjunta para transformar la representación primitiva de los datos en una nueva representación para facilitar el aprendizaje. La mayoría de los algoritmos de aprendizaje logran una precisión alta en problemas donde el conocimiento disponible permite una representación de datos basada en atributos muy informativos. Los datos de problemas reales normalmente se han preparado para fines distintos a la minería de datos y el aprendizaje automático, y por lo tanto, se han representado mediante atributos primitivos. La representación primitiva de los datos facilita la existencia de *interacción entre atributos* cuya complejidad hace que la información relevante quede oculta para la mayoría de los sistemas de aprendizaje. La selección de atributos es crucial para subrayar la importancia de los atributos relevantes. Sin embargo, la identificación de los atributos a través de un método de selección de atributos no es suficiente para el aprendizaje en presencia de interacciones complejas entre atributos porque las regularidades son todavía opacas y difíciles de aprender. La Inducción Constructiva (IC) se ha introducido para facilitar el aprendizaje mediante la reestructuración de la representación primitiva. Cuando IC se aplica a los conceptos con interacciones, la construcción de características en IC intenta capturar y encapsular las interacciones para subrayar las regularidades al sistema de aprendizaje. Recientemente, se han obtenido muchos progresos en IC; sin embargo, los métodos de aprendizaje todavía se enfrentan a graves dificultades cuando se aplican a conceptos con interacciones complejas.

Esta investigación tiene como objetivo facilitar el aprendizaje de conceptos con interacciones complejas cuando el único conocimiento disponible sobre el concepto sea la representación primitiva de datos de entrenamiento. La tesis describe los principales requisitos funcionales que un método de IC deberá cumplir con el fin de afrontar este tipo de problemas y propone un nuevo marco de trabajo que descompone la tarea compleja de IC en tareas pequeñas y más fáciles. A parte de este marco y los requisitos funcionales se ha diseñado dos métodos, DCI y MFE3/GA. Las evaluaciones empíricas sobre problemas sintéticos y problemas reales muestran la eficacia de estos métodos para mejorar la precisión del aprendizaje. El marco propuesto se puede usar como un modelo para diseñar una herramienta que pueda integrarse en una caja de herramientas de aprendizaje automático con el fin de facilitar el aprendizaje de conceptos difíciles con datos primitivos.

Acknowledgements

This work would not have been possible without the support of many people. I wish to express my sincere gratitude to all those who gave me the possibility to complete this thesis.

I would like to give the main credit for this research to my supervisor Eduardo Pérez for his encouragement, support and constructive criticism, which were the triggers for much of the work presented in this dissertation. I am deeply indebted to him who was always there when I needed it. I like to give him special thanks not only for his professional support through out my research study, but also for all the guidance he provided me about my personal life. I have learned a lot from him. Hope I continue receiving his guidance.

I would like to appreciate Alberto Suarez for his helpful comments and technical discussions. Many thanks are offered to Dr. Alex Freitas for generously providing me the benefit of his time and knowledge during GECCO'07 conference. I am grateful also to the Data Mining Research Group of Imperial College of London, UK, with whom I initiated this research. I also like to thank Prof. Caro Lucas from University of Tehran, Iran, who helped me make my decision to move to Spain in order to start my PhD.

Many thanks go to all the members of the department who have made EPS-UAM a special place for me over all these years. I wish to express my gratitude to Juana Calle, the secretary of the department, for her efficiency and hard working. My special appreciation goes to my colleagues and officemates who have been my Spanish language, culture, and history teachers, my emotional supporters, my defenders, and most importantly, my friends and my family substitute in Spain. They immensely offered me any help, encouragement, and friendship I needed. I was lucky to have such good colleagues as Miguel, Germán, Pablo, Estefanía, Pedro, Manu F., Abraham, Fran, Javi, Enrique, María, Rosa, Alvaro, Manu H., Ruth, Carlos, Diana, Ismael, Abdel, and José. Life in Spain would have been difficult without them.

Above all, I am very thankful to Farhad, a real big brother, for his advice and tips, to Azin for being a sister to me and listening to me whenever I needed, and to Ali, the little brother who helped me to switch off and relax sometimes during the preparation of this dissertation.

No words can express my gratitude toward my parents.

I was only five years old when a pattern hidden under the apparently random design of the window glass of my aunt's house amazed me. I now recognize that what I did in my PhD was in fact inspired from that childhood experience. I therefore have to be thankful to that window glass as well.

This research has been partially supported by an FPI scholarship from Universidad Autónoma de Madrid and by the Spanish Ministry of Science and Technology under grant numbers TIC98-0247-C02-02, TIC2002-1948, and TSI2005-08225-C07-06.

To my father for his determination,
and my mother for her patience;
with these attitudes they taught me to live

... it is important to gain knowledge.
Grasp of the intelligibles determines
the fate of the rational soul in the hereafter,
and therefore is crucial to human activity.
Avicenna, 980–1037, Iran

Chapter 1

Introduction

Most real-word data are prepared for purposes other than knowledge induction and machine learning; and therefore, their representation is primitive. The low-level primitive representation sometimes makes relevant information difficult to discover. Analyzing data for inducing knowledge is nowadays a vital mankind requirement. Learning a concept represented by primitive data requires a technique that eases the task of knowledge induction for the learner.

The field of machine learning was formed about four decades ago with the aim of providing computational methods that would be able to induce knowledge from experience [Samuel, 1959; Griffith, 1966; Minsky and Papert, 1968]. This field now plays a central role in computer science. Knowledge induction is required specially for problems that do not have algorithmic solutions such as medical diagnosis, visual concept recognition, and more generally, detection of interesting regularities in large data sets. Researchers have made extensive efforts to find ways and means for computers to learn from experience, which resulted significant achievements in introducing various algorithms for certain kind of learning tasks [Langley and Simon, 1995].

Nevertheless, existing learning systems have difficulty in learning some concepts. There are various factors that make a concept difficult to learn, such as appearance of noise in data, absence of a relevant attribute, missing attribute values, etc. However, sometimes even if all relevant information for learning is included in data, concept is still difficult to learn. Rendell and Seshu [1990] refer to such concepts as hard concepts, defined as follows:

> *"A concept is hard if its attributes have high intrinsic accuracy but the concept cannot be learned with existing inductive strategies."*

In other words, a concept is hard for a learner if its attributes contain the relevant information for inducing an appropriate abstract description of the concept but this information cannot be discovered by the learner. This occurs when the data representation is primitive and the relationship between attributes and the concept is obscure. The primitive representation facilitates the existence of attribute interaction, which makes regularities opaque for the learner (see Section 1.1). Hence, without help of a domain expert, such data is hard to learn. The growing number of primitive data demands more research to ease learning concepts that are represented by such data.

Constructive Induction (CI) methods [Matheus and Rendell, 1989; Pagallo and Haussler, 1990; Ragavan and Rendell, 1993; Hu and Kibler, 1996] have been used to facilitate learning concepts when interactions exist among attributes. These methods aim to abstract and encapsulate interactions into new features. This goal is usually achieved by selecting attributes and defining a function over them to highlight the interaction. Despite many progresses in CI, these methods still have difficulties when applied to concepts with complex interactions [Perez, 1997; Freitas, 2001], as briefly referred to in Section 1.2 and fully discussed in Chapter 2. The main problem of most CI methods is the use of a greedy local search to find interactions [Hu, 1998b; Dhar *et al.*, 2000]. When several interactions exist among attributes the search space is large and with high variation. It is, therefore, more likely that a CI method based on a greedy local search finds one of local optimal solutions. For such search space, a global search strategy should be applied to find the global optimal solution [Bensusan and Kuscu, 1996; Vafaie and DeJong, 1998; Hu, 1998b; Bhanu and Krawiec, 2002; Freitas, 2003].

This dissertation concentrates on the problem of attribute interaction and proposes a new method that can deal with concepts with complex interactions. It highlights weaknesses of some CI methods with the aim of advancing in this field and designing a new method that can successfully outline interactions to the learner. In order to focus on interaction problem, other learning difficulties such as dealing with noisy data [Brodley and Friedl, 1999], discretizing continuous attributes [Liu *et al.*, 2002], and predicting unknown attribute values [Grzymala-Busse and Hu, 2001] are excluded from the scope of this investigation. Section 1.1 explains the problem that learning systems usually encounter when data are described by a primitive representation. Section 1.2 introduces CI as a solution to this problem and defines some terms in this field. Then, the common problems of CI methods are mentioned. Finally a summary of the research and an overview of the thesis structure are given in Sections 1.3 and 1.4.

1.1 Learners and the Problem of Attribute Interaction

Significant improvements have been achieved in certain kind of learning tasks, yet some concepts represented by primitive data are still difficult to learn. Since most real-world data are not particularly prepared for machine learning purposes, their representations are not often appropriate for learning; as it is the case for protein sequences [Qian and Sejnowski, 1988], image data [Antonie *et al.*, 2001] and raw text [Gelfand *et al.*, 1998]. The primitive representation of real-word data facilitates the existence of attribute in-

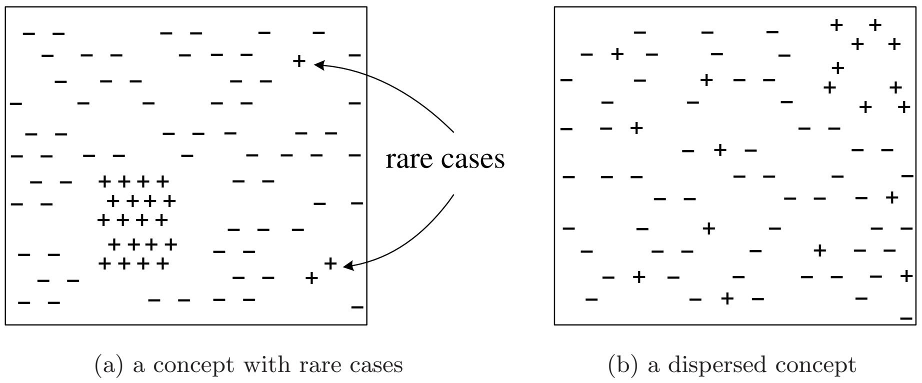


Figure 1.1: The instance space of two binary concepts with rare cases: the + and − points in the instance space represent examples belonging to positive and negative classes.

teractions, whose complexity makes the relevant information opaque to most learners, as it will be illustrated in this section.

Interaction exists among attributes when the relation between one attribute and the target concept is not constant for all values of the other attributes [Rendell and Seshu, 1990; Perez, 1997; Freitas, 2001; Jakulin and Bratko, 2003]. The first-order interaction among two attributes is defined as follows:

> Consider Y is the goal to predict using two independent variables or attributes X_1 and X_2. There is an interaction between X_1 and X_2 if the relationship between X_1 and Y depends on the value of X_2.

Similarly, the higher-order interaction among more than two attributes is defined. Interactions become *complex* when changing the value of one attribute does not only change the relation between another attribute and the target concept, but it yields an opposite relation. When complex interaction exists among n attributes, the value of all n attributes are needed to be considered simultaneously to predict the goal or the target concept.

One form of complex interaction is referred to as rare case or small disjunct [Holte *et al.*, 1989; Weiss, 2003]. A rare case is a small region in the instance space that covers relatively few examples of the same class (as shown in Figure 1.1(a)). When a large number of rare cases exist, concept is *dispersed*; that is, similar class labels are scattered through the instance space and surrounded by other class labels (see Figure 1.1(b)). Rare cases are difficult to identify because they contain few data and are usually misinterpreted as exceptions or noise. Rare cases form small disjuncts (rules that cover small number of the training examples) in the classifier. Many learning methods are in favor of discovering large disjuncts (rules covering a large number of examples). Hence, they do not include small disjuncts in the classifier. When a concept is dispersed a large number of small disjuncts are needed to cover rare cases. Since the number of samples covered by the set of small disjuncts is large, the classification

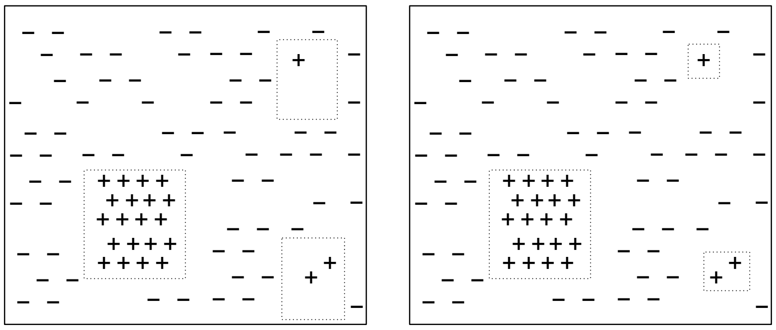


Figure 1.2: Two different generalizations of the rare cases of Figure 1.1(a): the dashed boxes show areas in the instance space covered by positive rules

accuracy of such learners is significantly degraded. Schaffer [1993] shows that the performance of learning methods with overfitting avoidance techniques such as pruning degrades if they are applied to concepts with rare cases. Moreover, when few training data are available, even if rare cases are identified, they are difficult to generalize [Weiss, 2005] (see Figure 1.2).

An extreme case of complex interaction is when different class labels are maximally dispersed across the data space (see Figure 1.3). In Boolean domains, parity concepts are the maximally dispersed concepts. The simplest parity function is the parity of two Boolean attributes defined as an exclusive-or (XOR) target concept $Y = X_1 \oplus X_2$. Here, if X_2 is false, then X_1 being true implies that Y be true (i.e., $Y = X_1$). Changing the value of X_2 to be true leads to an opposite relation between X_1 and Y; that is, X_1 being true implies that Y be false ($Y = \overline{X_1}$). Besides, X_1 has an analogous influence on the relation between X_2 and Y. The opposite relations cancel each other if only X_1 (or only X_2) is taken into account. Hence, there appears to be no relation between either individual attribute and the target. The interaction complexity augments with an increasing number of interacting attributes.

To see better the mutually canceling effects of attributes in such concepts, consider the Simplified Health Prediction problem in Figure 1.4, where the two independent

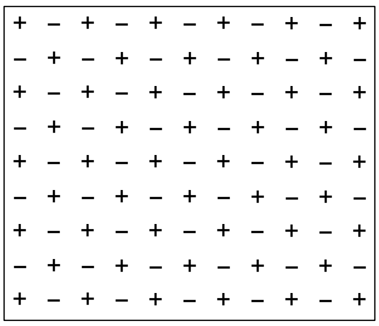


Figure 1.3: Maximally dispersed binary concept

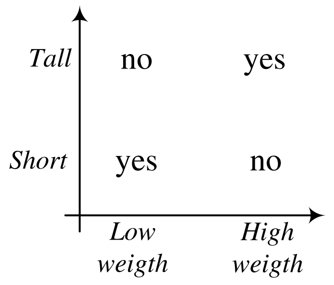


Figure 1.4: The Simplified Health Prediction: the weight of a person seems unrelated to the health conditions

attributes, weight and height, predict the health of a person. A person is healthy if he is short and low weight, or tall and high weight. If only the weight of people are taken into account, then among those whose weight is *high*, 50% are healthy (those who are also *tall*) and 50% are unhealthy (those who are *short*). In an opposite way, 50% of those whose weight is *low* are unhealthy (those who are *tall*) and 50% are healthy (those who are *short*). The opposite relations between each value of the attribute weight and the target cancel each other. Therefore, there appears to be no relation between the weight and the healthiness. The same is true for the attribute height. The two attributes interact with one another in the context of health prediction. Hence, each attribute by itself does not give enough information to determine the health situation. The values of both attributes are needed to predict the health of a person.

The mutually canceling effect is more obvious and extreme in maximally dispersed concepts, but is also appears in different degrees in other complex interactions. As illustrated with the above example, because of the mutually canceling effects of interacting attributes considering attributes individually does not help to uncover the underlying complex patterns that define the target concept.

Most learners assume attribute independence and consider attributes one by one. They compute the evidence about the class for each attribute and sum up all these evidences to predict the target concept. Consequently, these algorithms achieve high accuracy when available domain knowledge provides a data representation based on highly informative attributes, as in many of the UC Irvine Databases used to benchmark machine learning algorithms [Blake and Merz, 1998; Holte, 1993]. Otherwise, their performance degrades when interaction exists among attributes.

Complex interaction becomes a stronger hindrance for these learners when data contains irrelevant attributes, since interacting attributes are easily confounded with irrelevant attributes [Ragavan and Rendell, 1993; Kohavi and John, 1997; Perez, 1997; Freitas, 2001]. To illustrate this, consider the previous example of Figure 1.4 with an extra attribute called leapling. This attribute that determines if a person is born on February 29, is irrelevant to the target concept. Table 1.1 shows training data of this example. Note that in Figure 1.4 all data samples were presented. Table 1.1

Table 1.1: The training data for the Simplified Health Prediction

Weight	Height	Leapling	Healthy?
low	*short*	*no*	no
low	*tall*	*no*	yes
high	*short*	*no*	yes
high	*tall*	*yes*	no

presents the same data samples with an additional attribute value; thus, it presents 50% of all data. Suppose the learner evaluates the attribute weight and the condition `if weight=low`. This condition does not provide any evidence about the target concept, since the two examples that match this condition have a class distribution equal to the whole data class distribution (50% yes and 50% no). Likewise, the other condition based on this attribute, `if weight=high`, and also other possible conditions based on the second attribute, `if height=short` and `if height=tall` provide no evidence about the target. However, the condition based on the leapling, `if leapling=yes`, perfectly classifies the only sample that matches this condition. Thus, the learner incorrectly believes that the attribute leapling is more informative than the other attributes and selects this attribute to generate the first rule, `if leapling=yes then healthy=no`. Recall that in this example 50% of data samples were available. In case that all data is provided, then, like the previous example of Figure 1.4, none of the attributes by itself give enough information about data classification.

Other learners which learn concepts by discovering similarities in the instance space, also find concepts with complex interactions difficult to learn. These learners assume that cases belonging to the same class are located close to each other. So, they fit a linear model to the data to classify regions. These algorithms achieve high accuracy when the data representation of the concept is good enough to maintain the closeness of instances of the same class. When complex interaction exists among attributes each class label is scattered through many small regions of the instance space, each covering relatively few examples of the same class. Thus, regularities are less prominent. This problem worsens when few training data are available.

As mentioned earlier, interaction prevails in domains where lack of knowledge imposes primitive representation. Interactions have been seen in several real-world problems. Financial data sets involve a considerable amount of attribute interactions [Dhar *et al.*, 2000]. Jakulin *et al.* [2003] show evidence of attribute interaction in medical data analysis for predicting the clinical outcome in hip arthroplasty domain. Danyluk and Provost [1993] detected interactions in telecommunications data, and found out that small disjuncts match a large percentage of training data.

Distinguishing interacting attributes from irrelevant ones has recently received attention, and few feature selection methods have been designed to tackle attribute interaction problem [Pappa *et al.*, 2002; Jakulin and Bratko, 2004; Zhao and Liu, 2007]. These methods consider several attributes together and apply heuristics to distinguish between subsets of interacting and redundant attributes. However, when interactions are complex, identifying relevant attributes may not be sufficient for learning the concept and improving accuracy. Because of concept dispersion, even if interacting attributes are identified correctly, regularities are still difficult to detect (see Section 3.2.1 for an empirical analysis). Moreover, when primitive attributes are provided for representing data, the concept description that is to be generated using primitive attributes tends to be complex [Bloedorn and Michalski, 1998]. Therefore, it is likely that the learner makes mistakes in constructing such description. Hence, its accuracy will be low.

Thus, in spite of correctly identifying interacting attributes, learners fail to learn concepts with complex interactions among attributes. Interactions need to be highlighted in order to learn these concepts properly. Next section explains how CI, by means of constructing new attributes, can encode attribute interactions in a new representation to highlight regularities to the learner.

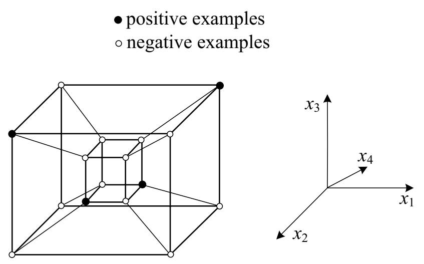
1.2 Constructive Induction as a Solution

CI methods have been designed to facilitate learning hard concepts when the representation space is of poor quality. The original idea of CI was to generate additional, more relevant and predictive attributes derived from the set of primitive attributes to improve the performance of a particular learning system [Michalski, 1978]. It is later extended to any kind of changing representation space such as removing primitive attributes, adding new attributes, and replacing original ones [Matheus and Rendell, 1989; Ragavan and Rendell, 1993; Liu and Motoda, 1998]. CI aims to automatically transform the original representation space into a new one where the regularity is more apparent, by eliminating irrelevant attributes as well as constructing more relevant features (*new attributes*) [Aha, 1991; Dietterich and Michalski, 1981].

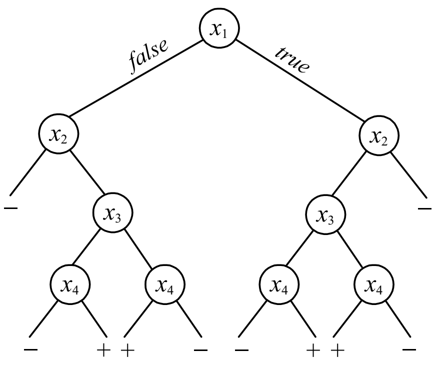
When the hard concept consists of complex interactions, Feature Construction (FC) plays the main role in a CI method to alleviate attribute interaction problem. FC aims to encapsulate the interaction among several attributes into a new one and outline it to the learner. Matheus and Rendell [1989] define FC as follows:

> *Feature construction is the application of a set of constructive operators* $\{o_1, o_2, \ldots, o_n\}$ *to a set of existing features* [given attributes] $\{f_1, f_2, \ldots, f_m\}$, *resulting in the construction of one or more new features* $\{f'_1, f'_2, \ldots, f'_N\}$ *intended for use in describing the target concept.*

FC intends to discover opaque information about the relations between subsets of attributes and the target concept. The discovered information is abstracted into a feature (new attribute) and added to the set of original attributes. The new feature groups samples of the same class, which could be scattered in the original data space. If



(a) the instance space in four dimension using primitive attributes



(b) the decision tree to be generated for the concept

Figure 1.5: The concept $y = (x_1 \oplus x_2) \wedge (x_3 \oplus x_4)$

FC finds the appropriate features, after changing the representation the instance space is less dispersed and highly regular; thus, the concept is easy to learn.

To illustrate how FC can ease the problem of attribute interaction, consider the concept $y = (x_1 \oplus x_2) \wedge (x_3 \oplus x_4)$ in Figure 1.5. The concept consists of conjunction of two complex interactions (two exclusive-ors). As shown in Figure 1.5(a), the concept is dispersed (i.e., positive examples are scattered and surrounded by negative examples) and, therefore, hard to learn. Figure 1.5(b) illustrates the classifier to be constructed by a learner to represent the concept. Due to the complex interactions, usually a learner fails to discover the exclusive-or relations that exist among attributes and construct the classifier. FC encapsulates these relations into two features, f_1 and f_2, as new attributes, each representing one interaction. Then, CI transforms the instance space into a new one represented by the new attributes (Figure 1.6). With the new representation of data, learning system can see regularities and learn the concept easier. After constructing features, the classifier is smaller and easier to generate.

There are several issues that determine the behavior of a CI algorithm [Hu, 1998a]. The relevant ones to this dissertation are *Hypothesis Driven* vs. *Data Driven* and *Interleaving* vs. *Preprocessing*, which are described bellow.

A hypothesis driven method uses a hypothesis generated by a learner for FC. These

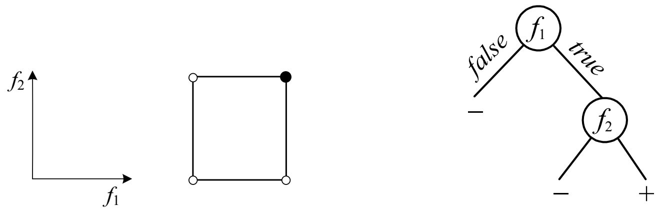


Figure 1.6: The new representation of the concept of Figure 1.5, $y = f_1 \wedge f_2$, and the decision tree to be generated

methods may use the hypothesis for constructing and/or evaluating new features. For instance, CITRE [Matheus and Rendell, 1989] and Fringe [Pagallo and Haussler, 1990] apply a learner to generate a hypothesis using original attributes, and examine the hypothesis to construct new features. The new features are then added to the set of original attributes and data are redescribed. The process of hypothesis generation and FC is repeated until the desired conditions are obtained. Vafaie and DeJong [1998] do not make use of a learner for constructing new features; they apply a learner for evaluating candidate functions as new features. Their method constructs functions and, then, assesses their effectiveness in classifying data using the hypothesis generated by the learner. Hypothesis-driven methods use hypotheses to guide CI. So, they can benefit from the previous knowledge obtained from the hypotheses. However, the FC in these methods is dependent on the quality of the hypotheses. If the learner is unable to generate a helpful hypothesis in the first pass, then the method fails to discover useful features [Hu and Kibler, 1996; Rendell and Ragavan, 1993]. Contrary to these methods, a data driven method (such as AQ17-DCI [Bloedorn and Michalski, 1998]) directly analyzes training data to extract relations and construct new features. The constructed features are then examined using an evaluation function on training data. If they are evaluated as good features, they are added to the attribute set and data are redescribed for further processing. These methods do not benefit from any hypothesis generated previously by a learner. Their strategy helps the method to work independently from any learner.

An interleaving method (e.g., Greedy[3] and Grove [Pagallo and Haussler, 1990]) integrates a learning process with FC; that is, one or more features are constructed, data are classified and partitioned using new features, and the process is repeated on each partition of data till the desired solution is achieved. When the process terminates a classifier has been generated using constructed features. These methods benefit from the classified data to construct features. However, the FC is affected by the bias of the data classification. Bias is the assumption of the method regarding the target concept and the strategy applied to generalize from training data to infer classification of new unobserved data [Mitchell, 1980; 1997]. If the learning bias does not match with hard concept, features that are constructed and used in the classifier may not classify new data correctly. On the contrary, in a preprocessing method (e.g., GALA [Hu and Kibler, 1996]), FC is performed before any learning and independently. When all features are constructed and data are redescribed by the new set of attributes, the new attribute set and data can be passed to any standard learner.

In spite of many progresses in CI, the problem of complex attribute interaction has not been completely solved yet. Current FC systems face difficulties to succeed when domain contains several complex interactions involving many combinations of attributes. Three main reasons that cause this failure are as follows (Chapter 2 explains more about these weaknesses). Firstly, most FC methods apply a local search such as hill climbing or beam search to find interacting attributes. When complex high-order interactions exist among attributes the search space is large and with high variations. Therefore, a local search may find a local optimum and construct a function over less relevant attributes [Freitas, 2001]. Secondly, FC methods usually apply a greedy algorithm to construct features; that is, the construction of each feature depends on the earlier constructed feature. When complex interactions exist among attributes, such greedy methods may mistakenly construct an incorrect feature in a primary step [Vafaie, 1998]. If features are constructed incorrectly in the primary steps, the successive features will be irrelevant because their construction depends on the previously constructed features. This problem is aggravated when several complex interactions exist in the concept and more features are needed to be constructed. Finally, FC methods often apply some algebraic operators such as arithmetic or Boolean operators to represent features. When concept consists of complex interactions, a complex algebraic expression is required to capture and encapsulate each interaction into a feature [Zheng, 1995]. In addition, if no prior domain knowledge is available, it is difficult to define appropriate operators.

This thesis is concerned with these three difficulties of CI methods. Few recently proposed methods tend to prevent some of these problems (see Chapter 2). However, there is no CI method that is not affected by at least one of these particular problems.

1.3 Summary of the Research

The main thesis of this research is the following:

> *When the main difficulty of a learning task is the presence of complex interactions among attributes, few training data are provided, and no prior domain knowledge is available, a CI method requires a mechanism that:*
>
> 1. *provides a global search to find subsets of interacting attributes*
> 2. *applies a feature representation language that reduces the difficulty of constructing complex features*
> 3. *allows simultaneous construction of several features.*

The above thesis defines a hypothesis consisting of three parts which determines three requirements for a CI method when applied to hard concepts with complex interactions. To prove this hypothesis two preprocessing data-driven CI methods are proposed. The first method, DCI [Shafti and Pérez, 2003a], aims to support the first two parts of the hypothesis. This method applies a Genetic Algorithm (GA) [Michalewicz, 1999] as a global search to find interacting attributes and construct a function over them. The notion of *non-algebraic (operator-free) representation* of constructed features is introduced as a utility to reduce the difficulty of constructing complex features when no prior knowledge is available about the concept. The research shows that a GA as a global search can ease the goal of finding interacting attributes and the relation among them. Also, the experiments support the claim that non-algebraic representation reduces the difficulty of FC. In addition, the experimental results confirm the need for simultaneous construction of several features, mentioned as the third requirement in the above hypothesis. The second method, MFE2/GA [Shafti and Pérez, 2005], while maintaining the advantages of DCI, allows simultaneous construction of more than one feature to prove the third part of the thesis. The GA in this method provides the ability to construct and evaluate several features at the same time, which is important for a CI when several complex interactions exist. Experiments illustrate that this characteristic along with the use of GA and non-algebraic representation make MFE2/GA to outperform other methods when the learning problem consists of several complex interactions.

1.4 Structure of the Dissertation

The dissertation is organized as follows:

- Chapter 2 studies the requirements for a CI method to handle the complex interaction problem. While briefly reviewing some related works, this chapter discusses the hypothesis that a global search is required for selecting subsets of interacting attributes, and a non-greedy method is necessary for simultaneously constructing and evaluating several features together. The application of evolutionary algorithm [Holland, 1975], a stochastic search method based on genetic inheritance, is proposed as a non-greedy global search technique, and some current genetic-based CI methods are reviewed. This chapter also explains the problem of representing features by the use of algebraic operators and suggests the application of a non-algebraic (operator-free) representation.

- Based on the requirements explained in Chapter 2, DCI is designed and described in Chapter 3. This CI method applies GA to select a set of interacting attributes and construct a non-algebraic function over it. The empirical evaluation and comparisons in this chapter confirm the need to apply a global search. Also, the advantage of the non-algebraic feature representation is illustrated. This chapter highlights the need for constructing more than one feature when concept consists of several interactions.

- Chapter 4 proposes the design of MFE2/GA. This method maintains all the advantages of DCI, while allows simultaneous construction of more features. MFE2/GA fulfils all the requirements discussed in Chapter 2. Experiments were conducted to compare this method with some traditional CI methods.

- Chapter 5 analytically and empirically compares the proposed method with a relevant CI method that applies non-algebraic feature representation. This chapter discusses the reasons that each method may outperform the other under different circumstances, supported by some experiments. The aim of this chapter is to illustrate the important characteristics of each method that can be integrated together to achieve better performance, which gives an open-line for future work.

- The experimental results in Chapter 4 lead to improve the method in Chapter 6. This chapter reviews different kinds of fitness functions and introduces a new fitness evaluation based on MDL Principle. The MDL-based fitness function is integrated into a new system called MFE3/GA. Experiments empirically illustrate that the new fitness function improves the performance of GA. Also experiments on some machine learning benchmarks illustrate that MFE3/GA outperforms other methods.

- Chapter 7 summarizes the thesis work, presents the conclusions, and outlines the open lines for future work.

Chapter 2

Constructive Induction: Difficulties and Solutions

As described in Chapter 1, the primitive representation of data facilitates the existence of complex interactions among attributes. Complex interactions hinders a learner from uncovering regularities. CI aims to ease learning hard concepts by improving data representation. When complex interaction prevails in data, this aim is achieved by constructing features as new attributes with greater predictive power.

This chapter describes difficulties of existing CI methods when they are applied to the attribute interaction problem, and suggests solutions that can be integrated into a system to provide a more promising method. Section 2.1 argues that most CI methods apply a greedy local search for finding interacting attributes, while a global search is more convenient for hard concepts in presence of attribute interactions. Then, it describes the greedy-based techniques often used by methods to construct features one by one, and suggests the simultaneous construction and evaluation of several features when various interactions exist. Section 2.1 also explains that CI methods often apply an algebraic language for representing interactions as new features, which sometimes makes construction of features difficult. Non-algebraic operator-free representation is compared with algebraic representation and preferred for complex concepts when no prior knowledge is available about the domain. Section 2.2 introduces evolutionary algorithms, and more particularly, Genetic Algorithms (GAs) and Genetic Programming (GP) as global search techniques that can be used by CI. These techniques also provide the ability to construct and evaluate several features simultaneously, which is necessary for a CI method in presence of several complex interactions. Section 2.3 reviews some recent CI methods that apply GA or GP, and highlights their weaknesses. Considering problems of CI methods, a new framework for CI is proposed in the last section.

Throughout this dissertation two terms are used to classify CI methods. The first term, *greedy method*, refers to a method that applies a local search to find interacting attributes and construct features (see Section 2.1). The second one, *genetic method* is a method which uses a global search such as GA or GP (see Section 2.3).

2.1 Constructive Induction Problems

When CI is applied to learn concepts with complex interactions, it has to perform three tasks: searching for interacting attributes, discovering the relations among them and the target concept, and representing each relation as a function (defined over interacting attributes) or new feature. Considering these tasks, CI methods are evaluated in the following sections from three points of view:

1. The strategy applied to identify interacting attributes

2. The strategy applied to discover interactions among identified attributes

3. The language used to define functions that represent interactions.

Note that the first two aspects are related and each one has a direct effect on the other. In order to identify a subset of interacting attributes, it is necessary to see the relation among attributes and the target concept. But this relation is not apparent unless a proper feature outlines it. At the same time, a feature can highlight the interaction among attributes only when it is discovered using the relevant attributes. If the chosen subset of attributes is not good enough, the best feature found may not be helpful. Thus, the two tasks are not independent and should be linked to transfer the effect of the search space of each task into the other. However, for analyzing weaknesses of CI methods, these two aspects are studied separately in Sections 2.1.1 and 2.1.2. The third one, the representation language, has an impact on the way the search is conducted to achieve the goal. This importance is described in Section 2.1.3

2.1.1 Search for Interacting Attributes

Section 1.1 described that due to the mutually canceling effect of interacting attributes, the importance of each single attribute is masked. Thus, a major difficulty of a CI method is to find interacting attributes for constructing features. Researchers have been experimenting various techniques to automatically identify interacting attributes for FC when there is no prior knowledge about the domain.

One technique is the use of a learner to guide the search. The learner generates a hypothesis using original set of attributes. Then, the CI method applies some criteria to select the most relevant attributes in the hypothesis as interacting attributes, and generates new features by defining functions over these attributes. The new features are added to the original attribute set, and the process continues until no new useful feature can be constructed. Fringe-like algorithms [Pagallo and Haussler, 1990; Pagallo, 1990; Yang *et al.*, 1991] apply this technique. They generate a decision tree using C4.5 [Quinlan, 1993], and then, consider the attributes in the fringe of the tree as candidates for constructing new features. CITRE [Matheus and Rendell, 1989] also builds a decision tree and selects attributes in branches that lead from the root to the positively labeled nodes to construct features. These hypothesis-driven methods are limited by the quality of the hypothesis generated by the learner. When the concept is complex the primitive representation of data is not good enough for the learner to generate a high-quality hypothesis. If the hypothesis is improper, irrelevant attributes are selected, and an irrelevant feature is constructed. Thus, the subsequent features that are constructed in next iterations will be meaningless.

Other methods, such as MRP [Pérez and Rendell, 1995] and X-of-N [Zheng, 1995], apply a greedy local search to select interacting attributes one by one. They use a heuristic function to guide the search algorithm. These methods usually apply one of the following approaches originally used in statistics [Draper and Smith, 1981; Neter *et al.*, 1996] to find subset of interacting attributes [John *et al.*, 1994]:

- Forward Selection: starts from an empty subset and selects and evaluates attributes one by one as candidates for inclusion in the subset.
- Backward Elimination: starts from the entire set of attributes and selects and evaluates attributes one by one as candidates for elimination from the set.
- Forward Stepwise Selection: starts from an empty subset and selects and evaluates attributes one by one as candidates for inclusion in the subset or elimination from the subset.
- Backward Stepwise Elimination: starts from the entire set of attributes and selects and evaluates attributes one by one as candidates for inclusion in the subset or elimination from the set.

These greedy methods intend to locally search for the best subset of interacting attributes by adding or removing one attribute at a time. When high interaction exists among attributes, each interacting attribute by itself does not give enough information about the concept (see Section 1.1). Thus, the heuristic function cannot see the goodness of these attributes. Therefore, it may mistakenly remove relevant attributes or add irrelevant attributes. Since the selection of each attribute depends on attributes previously selected, the local search may find a locally optimal subset (i.e., an attribute subset that is better than all its neighbors in the search space but not globally the best). This occurs since the search space has high variation and lots of local optima due to the interaction among attribute (see Section 2.2.3 for more explanation about the local search problem).

To illustrate the problem of greedy-based local search, consider data are represented by six Boolean attributes $\{x_1, \ldots, x_6\}$ and the target concept is $x_1 \oplus x_2 \wedge x_3 \oplus x_4$, where $\oplus$ is exclusive-or (XOR). A CI method needs to identify the subset $\{x_1, x_2\}$ or $\{x_3, x_4\}$ to construct a function over it. A forward selection method uses a heuristic function to evaluate attributes and select the best one for for inclusion in the subset. But since there are complex interactions among attributes the goodness of a relevant attribute is not apparent on its own. Only when interacting attributes x_1 and x_2, or x_3 and x_4 are considered together the goodness of these attributes is seen. Hence, the heuristic function may mistakenly evaluate an irrelevant attribute (x_5 or x_6) as the best attribute to be included in the subset of interacting attributes. This problem worsens when more attributes are involved in the complex interaction because a larger number of attributes are needed to be considered together to identify interacting attributes. Backward elimination has a similar problem. In the first two steps, elimination of x_5 and x_6 results in a better subset. But then removing any of relevant attributes x_1 to x_4 does not improve the goodness of the subset. Thus, the method selects $\{x_1, x_2, x_3, x_4\}$ as the subset of interacting attributes. Though all attributes in the subset are relevant, the function that is constructed over this subset represents the relation among four attributes, that is, $x_1 \oplus x_2 \wedge x_3 \oplus x_4$. This function is larger than a function defined over two interacting attributes (for example, $x_1 \oplus x_2$). Therefore, it is more complex to construct. The method, instead of a greedy local search, needs to jump in the search space from $\{x_1, x_2, x_3, x_4\}$ to $\{x_1, x_2\}$; that is, remove two attributes at the same time (two steps further in the search space). Then, it could construct a simpler and efficient function over the subset $\{x_1, x_2\}$ and in the next step another simple function over $\{x_3, x_4\}$ to achieve a better result. This problem aggravates when the target is composed of several interactions among attributes. Same happens with forward stepwise selection and backward stepwise elimination.

Alternatively, some methods like LFC [Ragavan and Rendell, 1993] use a lookahead search to reduce problem of local optima. LFC looks ahead through a number of original attributes and previously constructed features using a beam search. The method keeps a number of attributes with high information gain [Quinlan, 1983] for the beam. Then beam attributes and attributes within a specified distance (based on the information gain value) from the beam attributes are selected as operands for constructing new features. This technique achieves high accuracy on some real-world problems. However, it is sensitive to the parameter that specifies the lookahead depth. When high interaction exists among attributes, many possibly long terms should be constructed. This method may not be able to form these terms if the lookahead depth is too small. Increasing this parameter may help. Nevertheless, without having previous knowledge about the concept the value of the parameter cannot be predicted. Blindly increasing this value is also computationally inefficient. Another problem of such a method is that the search is still constrained by a one-by-one evaluation as a heuristic to select an attribute, and may exclude some promising attributes in earlier steps.

Such kind of one-by-one attribute selection or elimination for constructing features is not appropriate for hard concepts in presence of complex interactions. CI needs to consider several attributes at a time to notice their relevancy and avoid the locally optimal solution.

For this reason some methods search the space of all possible subsets of attributes to choose the best one. At each step they evaluate a subset of attributes instead of one attribute. The space of subsets of attributes has a high variation. Therefore, a local search may not find the global optimum. Some methods [Dzeroski and Lavrac, 1993; Srinivasan and King, 1999] consider all combinations of attributes and perform an exhaustive search procedure. As the search space of subsets of attributes grows exponentially with the number of attributes (2^N subsets for N attributes in Boolean domain), an exhaustive search is computationally expensive. Alternatively, some methods impose restrictions in exploring the search space and forming subsets to reduce complexity. HINT [Zupan *et al.*, 2001] performs an exhaustive search over the space of all subsets of maximum size b ($b = 3$ by default). Since each function is constructed over a small subset of attributes, this method has to construct more features to represent an interaction over a larger subset of attributes (Chapter 5 studies more about this method). Furthermore, such methods limit the size of search space, which may result in ignoring some relevant subsets of attributes.

In order to effectively search the space of all subsets of attributes, CI needs a search strategy that eventually finds the globally optimal solution in such a complex and enormous search space in a reasonable computation time. A greedy local search approach may be suitable only when the variation of this excessively large search space is not high; otherwise, it will find a local optimum.

As an alternative to a local search, a global search such as evolutionary algorithms can be used. Evolutionary algorithms, which are based on genetic inheritance, are theoretically and empirically proven to be more successful in searching through complicated search spaces [Holland, 1975; Goldberg, 1989; Michalewicz, 1999]. Section 2.2 explains more about these global searches, and Section 2.3 reviews some recently proposed genetic CI methods.

2.1.2 Search for Interactions

FC in CI aims to capture and encapsulate interaction into a new feature. It generates a function defined over a subset of candidate attributes and estimates the predictivity of the function as a new feature. When complex interactions exist among attributes, constructing more than one feature becomes necessary. The strategy followed to construct several features is important in a CI method. Most CI methods are greedy in the sense that they generate and evaluate features one by one during an iterative process. The construction of each new feature depends on the earlier constructed features.

One group of such methods uses a previously constructed feature as a new attribute to incorporate in construction of the new functions. For instance, LFC applies a beam of selected original attributes and uses these attributes to construct a feature. Then the constructed feature is added to the original set of attributes, the beam is reset using the new attribute set, and the process is repeated. HINT [Zupan *et al.*, 2001] analyzes data to induce a function. When the function is generated, the attributes used for constructing the function are removed and the function as a new feature is added to the original attribute set. Then data are redescribed using the new attribute set and the process is continued. In such methods the quality of new functions directly depends on the quality of the previously constructed functions. Since there is a high variation in the search space of candidate functions, the system may construct a locally best function in primary steps. Consequently, the successive features will be irrelevant and the whole process will converge to a local optimum.

Some other greedy methods, such as GALA [Hu and Kibler, 1996], are recursive splitters. They generate a decision tree while constructing features. At each iteration, they construct a feature that best splits data and use it as a condition to partition data. Then the process is repeated for each part of data to generate more features. This approach is called *divide and conquer*. These methods use a data-driven measure such as entropy [Quinlan, 1986] to guide the FC to select the best feature. If the earlier constructed feature is not good enough, data are split improperly. Therefore, the data-driven measure misguides the FC in the next step and an improper feature is selected. Moreover, some of these methods, such as MRP [Pérez and Rendell, 1995], induce features directly from data. Then, after an improper split of data, there might not be enough and proper training data for inducing a function with a high predictive power. Thus, the new constructed function overfits data, that is, fits well with this small part of data but does not fit well with unseen test data. These methods confront more difficulties when few training data are available.

Above problems degrade if the number of interacting attributes is high. When more attributes interact, the feature that encapsulates the high-order interaction is complex and difficult to construct. A CI method must break down the complex high-order interaction into several smaller features. Such set of features works as a theory of intermediate concepts that bridge from the primitive data representation to the hard complex target concept. When more functions are to be constructed, such greedy methods need more iterations. Each feature that partially shows the interaction, by itself, may not provide much information about the concept, and may be evaluated as an irrelevant feature. The greedy method may construct a feature that mistakenly appears to be relevant. Thus, the iterative process is misguided and subsequent features are constructed incorrectly. For this reason such methods fail when number of interacting attributes grows and the interactions are complex.

In order to avoid these problems, CI methods need to construct several features at the same time and evaluate a set of features together as related parts of the theory. Since, evaluating a set of features is essential, a CI method has to consider all possible functions and search the space of subsets of candidate features. The major problem is that the search space is huge. CI needs to find out which functions over which attribute subsets (among 2^{2^N} functions in case of N attributes in a Boolean domain) are more predictive. Then, the challenge is to design a cost-effective CI method that can construct and evaluate several features together.

In Section 2.1.1, genetic-based search is referred to as a global search to apply over the space of subsets of attributes. Genetic-based search also provides the ability to evaluate several constructed features as one single individual as it will be seen later in Sections 2.2 and 2.3.

2.1.3 Representing Interactions

In addition to the problems described in Sections 2.1.1 and 2.1.2, a CI method may also confront difficulties when representing interactions as functions or features. The representation language has an impact on the way the search space is conducted. It should reduce the difficulty of constructing functions that represent complex interactions. Also, the language should provide the capability of representing all complex functions or features of interests. It has an important role in convergence of a CI method to the optimal solution in less time.

There are two alternative groups of languages for representing features: algebraic form and non-algebraic form.

Algebraic form is the common form used by most CI methods. In algebraic form, an interaction is represented by a function defined by means of algebraic operators such as arithmetic or Boolean operators. When using this form of representation, the operators should be defined properly. If prior knowledge is available about the functions that represent interactions, methods can use this information to define operators before applying FC to the problem. For instance, FICUS [Markovitch and Rosenstein, 2002] and LAGRAMGE [Todorovski and Dzeroski, 1997] enable the user to define any kind of operators using domain knowledge.

Some other methods use a fix set of complex operators designed for a specific kind of problems. ID2-of-3 [Murphy and Pazzani, 1991] uses M-of-N operator (which returns true if at least M of N conditions are true; otherwise returns false), and Zheng [1995] uses the operator X-of-N (which returns the number of conditions that are true among N conditions). LMDT [Utgoff and Brodley, 1991] and Swap-1 [Indurkhya and Weiss, 1991] use hyperplane representation. These complex operators simplify construction of some features, yet they are designed for specific problems and do not cover all kind of relations among attributes. They can be applied only to problems which are known to contain interactions that match with this form of representation. For example, if the problem contains interactions in form of complex DNF (Disjunctive Normal Form) with long terms, X-of-N or M-of-N are not sufficient to represent interactions [Zheng, 2000].

In order to make FC applicable to a wide range of problems, some methods use a fix set of simple and general operators. For instance, Fringe, CITRE and GALA use the set of Boolean operators conjunction and negation ($\wedge$ and $\neg$). These two simple operators are sufficient to represent all Boolean relations among attributes; though, they are not efficient. The disjunction ($\vee$) is represented by the negation of conjunction. Thus, if the feature must represent several disjunctions, it is excessively complex and harder to construct. Some methods such as LFC include disjunction in the set of operators in order to construct features easier. However, still construction of complex interactions is difficult when using simple operators since these operators have a limited expressiveness power.

To better illustrate this point, assume that the interaction to be represented is the parity of four Boolean attributes. This interaction can be represented as bellow using the Boolean operator $\oplus$ (i.e., exclusive-or):

$$(x_1 \oplus x_2) \oplus (x_3 \oplus x_4) \ .$$

But if the set of predefined operators are conjunction, disjunction, and negation, the function that represents the interaction may have the following form:

$$(x_1\overline{x}_2 + \overline{x}_1x_2)\overline{(x_3\overline{x}_4 + \overline{x}_3x_4)} + \overline{(x_1\overline{x}_2 + \overline{x}_1x_2)}(x_3\overline{x}_4 + \overline{x}_3x_4) \ .$$

Note that, in this representation, $a + b$, ab, and $\overline{a}$ are equivalent to $a \vee b$, $a \wedge b$, and $\neg a$, respectively. It can be seen that this representation is more complex; therefore, the function is more difficult to construct. A complex feature is required to capture and encapsulate the interaction using these simple operators.

If there is prior information about the kind of operators needed for representing interactions then algebraic representation can ease the task of constructing functions. Otherwise, defining operators without any domain knowledge may limit the search space or make it more difficult to explore. Thus, if there is no prior domain knowledge available, methods with algebraic representation may fail to construct promising functions. In this case an operator-free representation is preferable.

In addition to the problem of defining operators, an algebraic form of representation may introduce *redundancy* in the representation; that is, each function is represented in several forms. An example of redundancy can be seen in the following, where the three functions are conceptually equivalent but represented differently:

$$x_1x_2$$

$$((x_1x_2) + \overline{x}_1)x_1$$

$$((((x_1x_2) + \overline{x}_1)x_1) + \overline{x}_2)x_2 \ .$$

Redundancy in representation increases the size of search space and may produce an unlimited search space if any feature can be represented in infinite forms. When redundant representation is applied a CI method needs a restriction to limit the search space and avoid redundant solutions. Therefore, such representation is less efficient.

These problems of algebraic representation cause CI methods to fail when complex features are needed to represent a high-order complex interaction among attributes. Note that the genetic CI methods also apply algebraic representation typically using parse trees [Koza, 1992](see Section 2.3).

Alternatively, interactions can be represented in a non-algebraic form, which means no operator is used for the representation. For instance, for a Boolean attribute set such as $\{x_1, x_2\}$ the exclusive-or interaction $x_1\overline{x}_2 + \overline{x}_1x_2$ can be represented by a non-algebraic feature such as $R = \langle 0110 \rangle$, where the j^{th} element in R 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

Note that the representation in this example is not the only one in non-algebraic form. Also, for simplicity, a Boolean concept is chosen for the example but the non-algebraic representation can be used for any domain after discretizing continuous attributes.

A non-algebraic form is simpler to apply in a CI method since there is no need to specify any operator and it does not introduce any redundancy in representation. Besides, all functions defined over a subset of attributes have equal degree of complexity when they are represented in a non-algebraic form. For instance, using the non-algebraic representation of the above example, two algebraic features like $f(x_1, x_2) = x_1\overline{x}_2 + \overline{x}_1x_2$ and $f'(x_1, x_2) = x_1$ are represented as $R(x_1, x_2) = \langle 0110 \rangle$ and $R'(x_1, x_2) = \langle 0011 \rangle$, which are equivalent in terms of complexity. Note that for problems with high interaction, CI needs to construct features more like the first feature in this example, which are complex in algebraic form, but not more complex than others when represented non-algebraically. Hence, if there is no prior knowledge about the kind of functions we are looking for, a non-algebraic operator-free representation is preferable.

There are few methods that use such kind of representation. One of them is MRP, which represents features by sets of tuples using multidimensional relational projection [Pérez and Rendell, 1995]. Pazzani's method [1998] generates features constructed

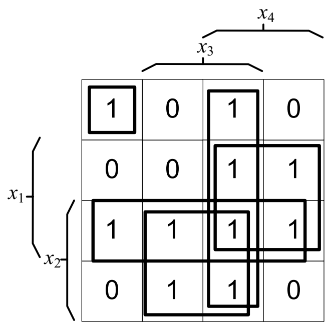


Figure 2.1: The Karnaugh map for function $R(x_1, x_2, x_3, x_4) = \langle 1001001101011111 \rangle$

by Cartesian product of two interacting attributes. HINT considers all tuples in the Cartesian product of interacting attributes and assigns a label to compatible tuples [Zupan *et al.*, 2001] (see Chapter 5 for more details about this method). The good performance of these methods over complex problems proves the advantages of non-algebraic features. However, these methods are greedy. As illustrated in Sections 2.1.1 and 2.1.2 greedy methods have difficulties in presence of several complex interactions.

It may be argued that non-algebraic form of representation is more difficult to comprehend by experts comparing to algebraic form. However, some existing tools and simple algorithms, such as Karnaugh map or Quine-McCluskey algorithm [Chan and Mourad, 1994], can be applied as a post-procedure to interpret constructed features in algebraic form. As an example in a Boolean domain, consider a non-algebraic function defined over four Boolean attributes as $R(x_1, x_2, x_3, x_4) = \langle 1001001101011111 \rangle$. This function can easily be interpreted by Karnaugh map (see Figure 2.1) as the following DNF function:

$$f(x_1, x_2, x_3, x_4) = x_1x_2 + x_2x_3 + x_3x_4 + x_4x_1 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4$$

which is easier to comprehend. C4.5-Rules [Quinlan, 1993] can also be used to receive a data set representing the non-algebraic function and generate rules expressing the function.

Thus, non-algebraic form of representation in addition to be sufficient and efficient to represent complex high-order interactions, is easily interpretable.

2.2 Genetic-based Search as a Global Search

Section 2.1 described that the search space for finding interacting attributes and constructing functions is large and has high variation; therefore, a global search such as a genetic-based search is more likely to be successful. Evolutionary algorithms are stochastic search algorithms based on genetic inheritance inspired by Darwinian theories of fighting for survival. They are theoretically and empirically proven to provide robust search in complex spaces [Holland, 1975; Goldberg, 1989; Michalewicz, 1999]. It was also explained in Section 2.1 that CI requires constructing and evaluating several features together, since each feature by itself may not be evaluated correctly due to the complex interaction in data. An evolutionary algorithm allows simultaneously constructing and evaluating several features represented as a single individual, which turns out to be essential for concepts with high complex interactions.

Genetic Algorithm (GA), introduced initially by Holland [1975], and Genetic Programming (GP) [Koza, 1992] are specific types of evolutionary algorithms, which have been applied successfully to a variety of learning problems [Freitas, 2003] as well as to other tasks such as solving optimization problems. These genetic methods are computationally simple but powerful. They are not limited to a specific search space. They are, like other search methods, looking for the optimal solution through the search space; however, they search simultaneously from several points in the space. The reason for their success is that these methods search in intractable search spaces by retaining a balance between the exploitation and exploration of the search space [Michalewicz, 1999].

Genetic-based algorithms work on *individuals* where each individual usually represents a hypothesis in the space of possible solutions. Individuals consist of *genes*, where each gene specifies one or several characteristics of the individual. Like in natural life, genes control the inheritance of individual's characters (properties or characteristics). The aim of a genetic-based algorithm is to search for the optimal solution by performing an evolution process on the *population* of individuals. By means of *genetic operations*, relatively good solutions survive and relatively bad solutions die after some generations. The goodness or badness of each individual is measured by a function called *fitness*.

Before analyzing genetic CI methods (Section 2.3), a description of GA and GP are given and some terms are defined. Section 2.2.1 describes a classical GA, which is the simplest genetic-based search. Section 2.2.2 describes GP, the global search often used by genetic CI methods. Section 2.2.3 explains why these algorithms are expected to perform better than other methods.

2.2.1 A Classical Genetic Algorithm

GA deals with a population of possible solutions and intends to converge this population to the optimal solution. Figure 2.2 shows the main structure of a GA. Each individual of the population $P(t)$ in iteration t represents a potential solution to the problem. Each iteration is referred to as a *generation*. As shown in this figure, first $P(0)$ is initialized and evaluated. Then, while the stopping conditions are not fulfilled, iterations are repeated. At each iteration, a new population is generated from the previous one and the fitness of each individual of the population is evaluated. If the desired stopping condition

```
begin
   $t = 0$
   initialize $P(t)$
   evaluate $P(t)$
   do while not desired condition achieved
      $t = t + 1$
      generate $P(t)$ from $P(t - 1)$
      evaluate $P(t)$
   end do
end
```

Figure 2.2: Main structure of a GA

is not achieved, the next population is generated and the procedure is repeated. When the procedure is terminated, the best individual of the population represents an optimal solution.

As Figure 2.2 shows, the main procedure of GA consists of three basic parts: initializing the first population, evaluating individuals, and generating next populations, which are described next.

Individuals and the Initial Population

Each individual in GA has a *genotype* and a *phenotype* representation. A genotype is the coded form of the individual in GA. In classical GA, it is a fixed-length bit-string such as $< a_1, a_2, \ldots, a_n >$, where each bit a_i is called a *gene* and determines a characteristic of the individual. The phenotype is the decoded form (the meaning) of the individual, which represents a potential solution to a particular problem. For example, if the problem is to search for an optimal subset of a set of five attributes, each potential solution is an attribute subset. Then, each individual can be coded in a string of five bits, where the i^{th} bit (gene) of the string indicates the presence or absence of attribute x_i in the subset. The bit-string is the genotype representation of the individual, and its meaning, that is a subset, is the phenotype representation. For instance, the genotype $< 1, 0, 1, 0, 0 >$ is interpreted as the phenotype $\{x_1, x_3\}$. Note that genotypes are used for the process of generating new population, while phenotypes are used for the process of evaluating individuals, as illustrated next.

In a classical GA, the population size p, that is the number of individuals in the population, does not change during the generations. The initial population $P(0)$ is generated by producing p random bit-strings of fixed length n. If some knowledge is available about potential solutions, it can be considered in generating the initial population.

The design of individuals has an important influence on convergence of a GA. The representation of individuals can change the complexity of searching task and may cause a GA to converge to a local solution [Rothlauf, 2006].

Fitness Function and Evaluation

When a population $P(t)$ is initialized or generated from the previous one (i.e., $P(t-1)$), individuals in $P(t)$ are evaluated by the fitness function and rated with a numeric fitness value. The fitness function evaluates the goodness of each phenotype. This function estimates how close the proposed solution is to the optimal solution. For example, if the problem is to approximate an unknown function given training examples of its input and output, the fitness function could be defined as the accuracy of the individuals over this training data. Fitness function guides the search process toward an optimal solution. Genetic operators will then use the fitness values for selecting individuals of the population $P(t)$ and producing new ones to generate $P(t+1)$, as described next.

The purpose of GA is to optimize the fitness value of individuals in the population using genetic operators. Without loss of generality, it can be supposed that the aim of GA is to maximize the fitness. This is because, if the fitness function is $f(x)$ and the optimal solution is the minimal value, then we can substitute $f(x)$ by $g(x) = -f(x)$. Therefore, the minimization problem converts to maximization problem. It can be supposed also that $f(x)$ always takes positive values. If it is not, then we can add a positive constant C to the fitness to transfer it to be positive, assuming that $-C \leq \min(f(x))$ which is typical in practical cases. Although GA might not find an optimal solution, it often succeeds in finding a solution with high fitness [Mitchell, 1997].

Generating a New Population

When the fitness of each individual in $P(t)$ is determined, genotypes are ready for producing the next generation by means of genetic operators. A simple GA applies three operators: reproduction, crossover and mutation.

Reproduction: this operator selects individuals according to their fitness value and copies them to a *mating pool* for further genetic operations. The mating pool has the same size as the population. Reproduction tends to choose individuals in the population that have a higher fitness value. Thus, some individuals are chosen more than once for the mating pool. Individuals in the pool will be selected later, and their genes will be altered using crossover and mutation to produce new individuals.

In classical GA, reproduction applies *proportional selection* for selecting individuals, as follows. A biased roulette wheel is used for selection whose p slots are sized according to the fitness values of individuals. The roulette wheel is spined p times and each time an individual is selected and copied to the mating pool. So, the probability that an individual I_i with fitness $fitness(I_i)$ in a population of size p is selected is:

$$Pr(I_i) = \frac{fitness(I_i)}{\sum_{j=1}^{p} fitness(I_j)} ,$$

that is, proportional to individual's own fitness and the inverse of the total fitness of the population.

The biased roulette wheel is simulated as follows. For each individual I_i, the value q_i, which is the value of cumulative distribution function for I_i (i.e., the probability that an individual I_j where $1 \leq j \leq i$ is selected), is calculated as:

$$q_i = \sum_{j=1}^{i} Pr(I_j) .$$

Then, for selecting each individual, a random number r between 0 and 1 is generated according to a uniform distribution, and the individual is selected using the following function:

$$Select(r) = \begin{cases} I_1 & \text{if } r \leq q_1 \\ I_i & \text{if } q_{i-1} < r \leq q_i \end{cases}$$

The selected individual is then copied to the mating pool and the selection is repeated until p individuals are copied to the pool.

Note that, highly fit individuals are more likely to be selected and copied to the mating pool and, therefore, will tend to produce more offspring, while worse individuals are more likely to die.

The next two genetic operators are, then, applied to the mating pool to generate new individuals (offspring).

Crossover Operator: when mating pool is prepared, individuals are selected for crossover operation. The probability that each individual in the pool is selected for crossover is p_c, that is called crossover rate. This predefined parameter determines the expected number of individuals that undergo crossover operation (i.e., $p_c \times p$). This operator swaps segments (genes) of two selected individuals as parents to form two new offspring as new individuals. It is performed as follows. First, for each individual in the mating pool, a uniform random number r between 0 and 1 is generated. If $r < p_c$, the individual is selected for crossover. Then, selected individuals are randomly chosen to form pairs of parents. Finally, segments of parents are exchanged, as described next, to produce pairs of offspring, and parents in the pool are replaced by new pairs.

In classical GA, *one-point crossover* is used. For each pair of parents, this operator randomly generates a number between 1 and $n - 1$ (where n is the length of individuals) as the crossover point. Then, parts of parents, separated by crossover point, are exchanged to form two new individuals.

To illustrate how one-point crossover forms new individuals, suppose two parents are:

$$I_1 =< a_1, \ldots, a_n >$$

and

$$I_2 =< b_1, \ldots, b_n > \quad .$$

Let the crossover point be i, where $1 \leq i < n$. The offspring would then be:

$$I'_1 =< a_1, \ldots, a_i, b_{i+1}, ..., b_n >$$

and

$$I'_2 =< b_1, \ldots, b_i, a_{i+1}, ..., a_n > \quad .$$

Then I_1 and I_2 in the pool are replaced by I'_1 and I'_2.

By crossover operator, new individuals inherit some characteristics of parents. The intention is to exchange information between different potential solutions and achieve construction of better individuals (i.e., *exploitation* of the search space).

Mutation Operator: after crossover, *mutation* is applied to the mating pool. This operator generates a new individual by making a small change in a selected individual of the mating pool. The probability that each individual in the pool is selected for mutation is p_m, that is called *mutation rate*. Similarly to crossover, for each individual in the pool a random number between 0 and 1 is generated. If this number is less than p_m, the individual is selected for mutation operation. Thus, the expected number of individuals that undergo mutation operation is $p_m \times p$. The operator changes the value of a randomly selected gene (according to a uniform distribution) of the individual from 1 to 0 or vice versa. It aims to introduce some extra variability into the population and provide more diversity (i.e., *exploration* of the search space).

When both operators have been applied to the mating pool, the resulting mating pool is used as the new population $P(t + 1)$.

2.2.2 Genetic Programming

It is not always convenient to handle problems with binary representations and classical crossover and mutation [Davis, 1989]. GP algorithms have been developed to deal with these problems. GP, like GA follows the idea of genetic-based search, but with more complex representation of individuals and operations. Most genetic CI methods use GP algorithms, as described in Section 2.3.

Individuals in GP are computer programs. Any abstract task can be thought of as requiring discovery of a program or a function that produces some desired output for a

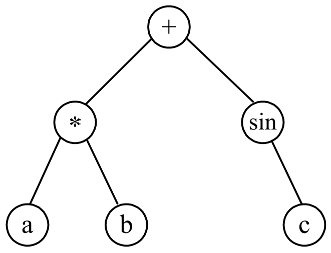


Figure 2.3: Parse tree representation of "$a * b + sin(c)$"

particular input. Therefore, solving these tasks can be reformulated as a search in the space of all possible computer programs to find the fittest program.

Programs are composed of functions and their arguments or terminals. Depending on the problem, functions can be arithmetic operations, programming operations, logical functions, and so on. Terminals are variables or constants. Koza [1991] represents individuals in GP as parse trees. Operators form the internal nodes of the tree and terminals are in the leaves of the tree. For example, an expression like "$a * b + sin(c)$" is represented as shown in Figure 2.3.

To apply GP algorithm to a particular problem, we need to define the set of functions and terminals. The GP algorithm then uses a genetic-based search to explore the space of programs that can be described using this set.

The fitness of each individual is measured in terms of how well it performs in the particular problem environment. For many problems, it is measured by the error produced by the computer program. For example, when the aim is to classify examples, the fitness can be the number of examples that the program classifies incorrectly.

Reproduction in GP is the same as in GA. But genetic crossover and mutation are different as individual's representation is not binary. Many approaches have been introduced for implementing these operators. The most common approach, introduced by Koza [1992], is as follows.

Crossover operator replaces a randomly chosen sub-tree of one parent by a sub-tree from the other parent. Parental crossover points are selected randomly. Figure 2.4 shows crossover operator applied to two parents. The mutation operator randomly selects a node of the parent and replaces it by a new sub-tree as Figure 2.5 shows.

The performance of GP strongly depends on the representation of individuals, fitness function and genetic operators. Research has been done to improve these factors, yielding variations in GP design. Researchers tried to explain different designs of GP by schema theory [Koza, 1992; O'Reilly and Oppacher, 1994; Poli and Langdon, 1998; Langdon and Poli, 2002]. Though, empirical analysis shows that in spite of variations in designing GP algorithms, they have potential to find the solution.

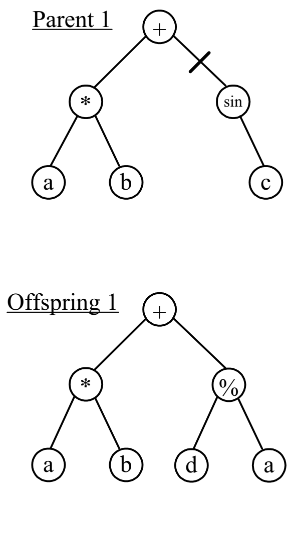


Figure 2.4: Crossover in GP

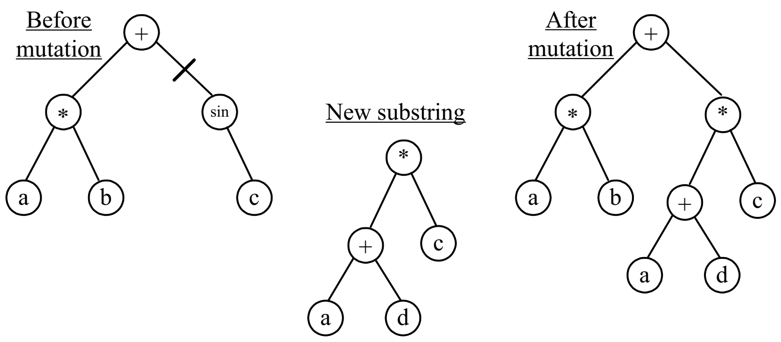


Figure 2.5: Mutation in GP

2.2.3 Why Genetic Search: Genetic-based Algorithms vs. Heuristic Algorithms

A genetic-based algorithm may be viewed as a stochastic algorithm with a strange performance. As explained in Sections 2.2.1 and 2.2.2, it starts from a random population, applies genetic operators to copy individuals, swap and mutate fragments of individuals, and eventually it is expected to find the solution. The schema theory [Holland, 1968; 1975; Michalewicz, 1999] explains how different components of a GA or GP guide these methods to converge to optimal solution (see [Langdon and Poli, 2002] for a survey of schema theories introduced for different designs of GA and GP). This section explains why genetic-based search is preferred to other search methods.

A heuristic local search such as hill-climbing starts from a (sometimes randomly) selected state of the search space. It evaluates the state by the fitness function (i.e., heuristic function), and looks at all neighbors of the state to select the neighbor with the best value of the fitness function. Neighbors are next states from the current state in the search path defined by the search strategy. If the best neighbor is better than the current state, it replaces the current state. Then, the procedure is repeated until no better state is found. Such algorithm might fail to find a solution by getting to a state from which no better neighbors can be selected. In this case, the program may be trapped by a local optimum. *Local optimum* is a state that is better than all its neighbors, but is not the best. This happens when there are varieties in search space, and many local optima exist. Finding a local optimum depends on selection of the starting state, because the best state to select depends on the previous selections.

Beam search tries to solve this problem by keeping m best members of the search space for future consideration. However, the possibility of missing the global solution still exists if the beam size, m, is not large enough. A small beam size can reduce the computational overhead, but it may rule out some promising solutions in the early stages. Moreover, this method still depends on the selection of a single starting point.

The genetic-based search differs from these methods in moving through the search space from one state to another. It starts from several states and jumps from one state to another that is not necessarily in neighborhood. It initiates from a population of randomly selected states instead of one state, and applies genetic operators to each state to jump to another possibly distant state, stay in the same state, or die. Genetic-based algorithms are also different from random algorithms. They are directed by a stochastic strategy, while random algorithms are blindly looking for the solution. In other words, genetic-based algorithms are a kind of multi-directional search.

Heuristic algorithms, therefore, are not effective when search space is complicated. Genetic-based algorithms will have more chance to be successful in searching through an intractable search space. As explained in Section 2.1.1, the search space of complex learning problems with interactions has a lot of variety and many local optima. Thus, a genetic method can be more effective for these problems to find the interacting attributes and construct a function that represents the interaction.

Another important property of genetic-based algorithms, which is of interest for a CI system, is the ability to design individuals in the population in a way to adapt best to the problem. It was explained in Section 2.1.2 that CI needs to construct more than one feature at the same time and evaluate several features together. Individuals in a genetic CI can be designed to represent sets of attributes and constructed features. Then, the fitness function evaluates various features and attributes together as a single individual. This property is necessary for FC when several interactions need to be discovered.

Nevertheless, the performance of genetic-based algorithms depend on a number of factors including: the choice of genetic representation and operators, the fitness function, user-defined parameters such as population size and crossover and mutation rates; yet, they are preferred to heuristic local search methods when the search space is intractable and complicated. If a proper representation language, genetic operators and fitness function are provided, a genetic CI method has the potential to generate useful features.

It should be noted that genetic CI methods usually take longer to run than greedy methods, which could be considered as a disadvantage when applied to large training data. However, the running time can be reduced by parallel processing. Most part of the genetic-based algorithms such as fitness evaluation, crossover or mutation, can be implemented as parallel procedures to be executed for several individuals simultaneously [Neri and Giordana, 1995; Anglano *et al.*, 1997; Araujo *et al.*, 1999; Freitas and Lavington, 1997].

2.3 Genetic CI

Previous sections argued that when the search space is complex a global search such as GA is more promising in convergence to optimal solution. Genetic methods have been applied successfully to a variety of learning problems [DeJong *et al.*, 1993; Freitas, 2002]. Their success in achieving higher accuracy proves the advantage of genetic search over greedy search. Several genetic CI methods have been designed for changing the data representation by selecting the best subset of attributes and removing irrelevant attributes [Yang and Honavar, 1998; Hsu *et al.*, 2002; Sierra and Corbacho, 2002; Pappa *et al.*, 2002]. However, when complex interactions exist, identifying interacting attributes is not sufficient to ease learning; interactions still need to be highlighted by constructing features (see Sections 1.1 and 3.2.1). There are few CI methods that use genetic search strategy for FC. Their partial success in constructing useful features proves the effectiveness of genetic-based search. This section reviews some of these methods and then highlights their weaknesses.

Most genetic CI methods apply GP. They usually represent functions as parse trees with algebraic operators in internal nodes and attributes in leaves (see Section 2.2.2) and apply some restrictions to control the unnecessary growth of the tree. However, these methods differ in aspects such as the definition of algebraic operators, the number of constructed features in each individual, fitness function, and other GP parameters.

GCI is a CI method proposed by Bensusan and Kuscu [1996]. Each individual in GCI represents a single function with domain specific operators. It is a preprocessing CI; at the end, the best pair of individuals is added to the original set of attributes as new features, and data samples are updated to proceed learning. This method is limited to construct only two features. So, it cannot break down a complex high-order interaction into several smaller features. GP in GCI uses a hypothesis-driven fitness function. The accuracy of the hypothesis generated by a learner is used for evaluating the fitness of each function. Thus, the convergence to optimal solution directly depends on the performance of the learner.

Otero *et al.* [2003] apply a similar representation of GP individuals; however, uses the fix set of arithmetic operators and two relational comparison "$\leq$" and "$\geq$" for constructing features. The fitness function is a data-driven function using information gain measures [Quinlan, 1983]. Thus, this method is faster than GCI, which uses a learner as fitness.

GPCI [Hu, 1998b] is the genetic version of GALA (see Section 2.1). Instead of a greedy search, it uses GP to find interacting attributes and construct features. Before starting GP, original attributes are Booleanized, that is transformed to Boolean attributes (for details see [Hu and Kibler, 1996]). Each individual in GP represents a single function. Operands are Booleanized attributes and operators are `AND` and `NOT`. Information gain is used for evaluating the fitness of each function. GPCI does not apply mutation operator and, therefore, suffers from lack of variation. Mutation is one of the essential genetic operators, which provides random diversity in the population, while crossover is used for construction and survival [Spears, 1992]. These two operators together keep the balance between exploration (diversity) and exploitation (construction). Therefore, GPCI is expected to fail when the concept is complicated. Like GALA, this method applies a divide and conquer strategy to produce more than one new feature. After each performance of GP, the best individual, as a new feature, is used for splitting data. Then a new and independent GP is performed for each division of data. At the end of the procedure, one feature for each splitting of data exists which is added to the original set of attributes. In fact GPCI performs GP several times to construct several features. Yet, similarly to GALA, the whole process of constructing a set of features is greedy. The construction of each feature depends on the feature previously constructed. If the previous feature is not good enough it may misguide the whole process toward a locally optimal solution.

To solve the problem of constructing several features at a time, Bhanu and Krawiec [2002] designed two GP-based methods. In the first method, each individual represents a set of functions instead of a single one. Thus, a combination of features are constructed and evaluated together as an individual. The second method uses a co-evolutionary system where several GPs are interacting with each other [Michalewicz, 1999]. The method maintains m populations each aiming to generate one feature. The fitness of individuals in each population affects the behavior of other populations. The whole system constructs m features, each generated by a GP in co-evolution. For both methods C4.5 [Quinlan, 1993] is used as a hypothesis-driven fitness function. Data are redescribed using new features and the accuracy of C4.5 is measured to evaluate features.

Gabret [Vafaie and DeJong, 1998] also constructs and evaluates several features simultaneously. It consists of two genetic-based modules: feature selection and feature construction. Each module is performed separately and independently from the other. The feature selection module uses GA to search the space of all possible subsets of the given attribute set. Individuals are bit-strings representing subsets of attributes in current set. The feature construction module, similar to previous methods, uses GP with parse trees. Each individual in this module is a set of attributes and features. Genetic operators aim to generate the best set of attributes and functions. Thus, the classical GP operators are modified to be applied in two levels: feature-set level and feature-construction level. For both modules, the fitness is the accuracy obtained by C4.5 [Quinlan, 1993] on transformed training data using features and attributes in the individual. The two modules are applied sequentially by starting from feature selection. The best individual generated by each module is used as the given attribute set for the other module. The sequence of running modules is terminated when no new set is produced. When high interaction exists, the feature selection module cannot see the interaction among primitive attributes due to using a hypothesis-driven fitness function. Therefore, this module often excludes some relevant attributes from the search space that should be used later for constructing new features. GAP [Smith and Bull, 2003] applies a similar strategy.

The hybrid method of Ritthoff *et al.* [2002] applies GA for selecting attributes. Each individual is a set of attributes and constructed functions. A new genetic operator is defined. This operator arbitrarily constructs a feature from current sets of attributes and a set of predefined constructive operators. Though features are constructed during the GA search of good attributes the method does not apply any genetic operator or search strategy in feature-construction level. Therefore, it explores the space of features without having a strategy to guide the method toward constructing better features. This method also uses a hypothesis-driven fitness function.

Larsen *et al.* [2002] apply GA to the X-of-N system of Zheng (see Section 2.1). They did not use parse trees. Each individual is a set of attribute-value pairs representing N conditions in X-of-N function (which returns the number of conditions that are true among N conditions). So, this method is convenient for problems that match this

Table 2.1: Genetic CI methods

Method/ Authors	Fitness function	Num. of features constructed	Feature evaluation
GCI	Hypothesis-driven	Two	one-by-one
Bhanu and Krawiec	Hypothesis-driven	Several	simultaneously
GABRET	Hypothesis-driven	Several	simultaneously
GAP	Hypothesis-driven	Several	simultaneously
Ritthoff *et al.*	Hypothesis-driven	Several	simultaneously
Otero *et al.*	Data-driven	One	one-by-one
Larsen *et al.*	Data-driven	One	one-by-one
GPCI	Data-driven	Several	one-by-one

bias. Information gain ratio is applied as fitness function. Genetic operators aim to change both attributes and values in attribute-value pairs of X-of-N to generate different combination of pairs. When GA is finished the best individual is selected as new feature.

Table 2.1 summarizes the important characteristics of the reviewed genetic CI methods: the fitness function applied, and the number of features constructed and evaluated by GA or GP. Most of these methods use a hypothesis generated by a learner to evaluate constructed features in combination with other attributes. These methods add the new feature to the set of attributes and redescribe data. Then, they apply a learner like C4.5 to measure the accuracy. This kind of hypothesis-driven evaluation relies on the performance of the learner. When complex interaction exists in concept, a hypothesis-driven fitness function may incorrectly assign a low or high fitness value to individuals. Hence, it will mislead the genetic search to a local solution. Moreover, performing a learning system for evaluating each individual increases the execution time of the genetic search. The computation time of fitness function is very important for the system's performance, since this function is called for every individual during many generations.

Alternatively a data-driven evaluation formula like entropy can be used. This approach, contrary to hypothesis-driven approach, only depends on data. In addition, its computation time is less than a hypothesis-driven function. Muharram and Smith [2005] performed experiments to evaluate the effect of different data-driven fitness functions on performance of a genetic CI method. They used GP with parse trees as individuals, using a fix set of arithmetic operators. They showed that the improvement in performance is not significantly dependent on which data-driven fitness measure is used. This illustrates the genetic-search robustness to this kind of fitness function.

Another problem of some genetic methods is the strategy applied to construct features. Section 2.1.2 argued that when complex interaction exists among attributes, CI needs to break down the interaction into several smaller features. Then, evaluating a combination of features together is essential for a CI, as each attribute or constructed feature alone may not be evaluated correctly. A genetic method provides the ability to evaluate a set of constructed features as an individual. However, many CI methods do not exploit this capability of genetic search. Four methods reviewed in this section evaluate several features at the same time (see Table 2.1); though, Ritthoff *et al.*'s method does not apply genetic operators to evolve features. Individuals in these methods usually represent a set of features. Bhanu and Krawiec have also used a co-evolutionary system to construct and evaluate several features at the same time.

In addition to these problems, all genetic CI methods reviewed have the deficiency of using an algebraic representation of features. As explained in Section 2.1.3, the use of simple predefined algebraic 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. Using domain-specific operators may ease construction of complex function; though, specifying these operators properly cannot be performed without any prior information about the target concept. In addition, algebraic representations may produce redundant solutions and, therefore, increase the size of search space. Conversely, non-algebraic representation does not need any operator and is not redundant. Moreover, it reduces the difficulty of constructing complex features. This form of representation is more convenient when no knowledge is available about the concept.

Considering the above problems a CI framework is proposed in the next section. Chapter 3 introduces a new GA CI method using this framework.

2.4 Conclusion: a CI proposal

So far, this chapter has explained that a CI method aims to find subsets of interacting attributes and functions defined over these subsets that encapsulate interactions. The search space for finding interacting attributes and constructing functions is large and has high variation when complex interactions exist in concept. Thus, a global search is preferred to search this deceptive space. GA and GP are stochastic search methods based on genetic inheritance, which can be used as global search techniques for CI methods.

It was also explained that CI requires constructing and evaluating several features together, since each feature by itself may not be evaluated correctly due to the complex interaction in data. A genetic method allows simultaneously constructing and evaluating several features represented as a single individual, which turns out to be necessary for concepts with high complex interaction.

Two types of feature representation were made concrete, which are: algebraic and non-algebraic. It was described that when no prior domain knowledge is available, it is difficult to define appropriate operators; therefore, non-algebraic operator-free representation is preferred to algebraic representation. Also algebraic representations may produce redundant features and increase the size of search space and, therefore, be less efficient. Moreover, a complex algebraic expression is required to capture and encapsulate complex interaction into a feature, while non-algebraic representation reduces the difficulty of constructing complex features.

Some deficiencies of existing genetic CI methods were reviewed. First, most of these methods apply a hypothesis-driven fitness function, which depends on the performance of the learner applied. Moreover, using a learner as fitness slows down the performance of the genetic CI. Thus, a data-driven fitness function is preferable. Second, it is necessary to construct and evaluate several features simultaneously when complex interactions exist among attributes. Some genetic methods reviewed in this chapter do not exploit GA or GP to construct and evaluate several features as an individual. Third, genetic CI methods often apply algebraic form of representation.

Considering the problems of genetic and non-genetic CI methods, a new framework is designed. This framework applies non-algebraic feature representation and uses a genetic search with data-driven fitness evaluation to construct features that represent interactions. The space of functions defined over all subsets needs to be explored to construct features. This space grows exponentially with the number of attributes (2^{2^N} functions for N input attributes in Boolean domain) and is difficult to explore. Searching this space is computationally prohibited. To ease the searching task, the search space can be decomposed into two spaces: $\mathcal{S}_S$ the space of all subsets of attributes, and $\mathcal{S}_{F_i}$ the space of all functions defined over a given subset of attributes S_i (Figure 2.6). 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 subgoals: 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).

As explained in Section 2.3, Gabret [Vafaie and DeJong, 1998] also divides the search space into two spaces. It applies a feature selection module before and independently from the feature construction module to filter out irrelevant attributes. However, when high interaction exists among attributes, the feature selection module 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 tasks, detecting interacting attributes and discovering a function that represents the interaction are related. The importance of attributes in subset S_i is not apparent unless a function f_i, that outlines the interaction, is defined over the proper subset of attributes. Hence, to guide S-Search to find a relevant subset S_i, it is necessary to find and highlight the relation among attributes by F_{S_i}-Search.

Therefore, the two tasks, S-Search and F_{S_i}-Search, should be integrated together to reflect the effect of each search space on the other. Based on this idea a framework is illustrated in Figure 2.7. Genetic search is used for S-Search to find interacting attributes. Each subset S_i generated in S-Search is evaluated by a function that is constructed by F_{S_i}-Search over the subset. Non-algebraic form is used for representing

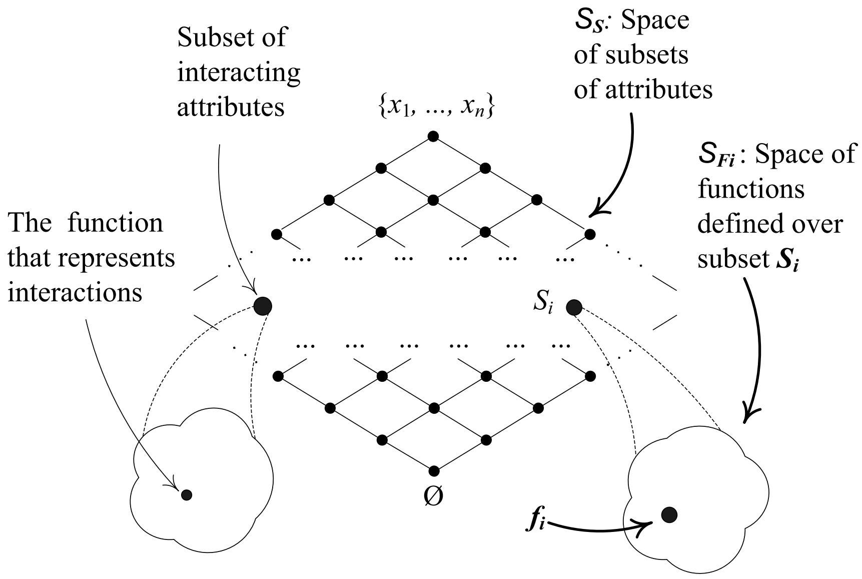

Subset of interacting attributes
$\{x_1, ..., x_n\}$
S_S: Space of subsets of attributes
S_{Fi}: Space of functions defined over subset S_i
The function that represents interactions
S_i
Ø
f_i

Figure 2.6: The decomposed search space

constructed functions. This form of representation permits *inducing* functions directly from data instead of searching for them (see Sections 3.1.1 and Section 4.2.1). F_{S_i}-Search analyzes training data to construct a function over the given attribute subset. The constructed functions are then evaluated to measure the goodness of the generated subsets of attributes in S-Search. Thus, each subset is mapped into a function by F_{S_i}-Search and the goodness of a subset is determined by the function defined over it. If F_{S_i}-Search finds a promising relation among given attributes represented by a function, the subset of attributes is considered as a good subset. By this strategy the link between two tasks, S-Search and F_{S_i}-Search, and their effects to each other are maintained, while improving each of them.

This framework is a major contribution of this thesis. Next chapter introduces DCI, an implementation of a method based on the proposed framework. This method constructs and evaluates one feature at a time. Later, Chapter 4 proposes a more

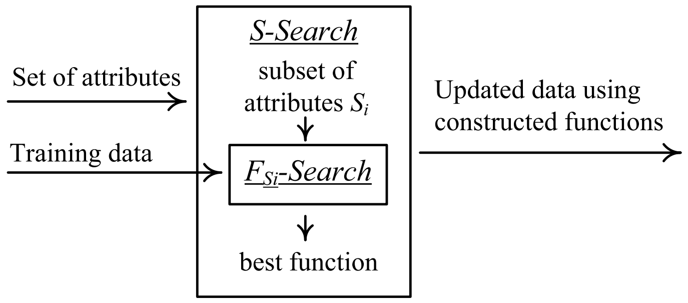

S-Search
subset of attributes S_i
Set of attributes
Training data
F_{Si}-Search
best function
Updated data using constructed functions

Figure 2.7: The decomposition framework for genetic CI

complete method, MFE2/GA, that allows simultaneous construction of more features. The new method fulfils all the requirements discussed in this chapter. MFE2/GA is theoretically and empirically compared with a relevant CI method in Chapter 5 and improved in Chapter 6.

Chapter 3

DCI: Decomposed Constructive Induction

Chapter 2 reviewed some CI methods and highlighted their deficiencies when they are applied to concepts with complex interactions. Most CI methods apply a greedy search to change the representation of hard concepts. Therefore, they suffer from the local optima problem when the search space is large and with high variation. When concept is complex because of the interaction among attributes, the search space for constructing new features has more variation. The CI method needs a global search strategy such as GA to skip local optima and find global optimal solutions. Some recently introduced CI methods based on genetic search were reviewed. 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: namely, the use of a hypothesis-driven fitness function, applying algebraic feature representation, and constructing and evaluating one feature at a time (see Section 2.3).

To overcome deficiencies of current CI methods, DCI (Decomposed Constructive Induction) was proposed [Shafti and Pérez, 2003a]. This primary approach aims to evaluate the utility of genetic search, data-driven fitness function, and non-algebraic feature representation when a complex interaction exists among attributes. This method constructs only one feature. Therefore, it fails to improve learning when several interactions exist and more than one feature is needed. Later, this approach will be extended by a multi-feature construction method introduced in Chapter 4.

There are important factors that should be defined carefully when designing a genetic-based search, including individual's representation, genetic operators, and fitness function. These determine the behavior of the genetic search in converging to optimal solution. Section 3.1 explains the design of DCI and details these factors. Section 3.2 describes experiments performed to evaluate features constructed by DCI. This section also empirically compares this method with relevant greedy CI methods. Last section sums up the results and highlights problems of the proposed method, which will be tackled in Chapter 4 by introducing a new CI method called MFE/GA.

3.1 DCI's Design

DCI is a preprocessing CI method. It receives original attributes and training data set and constructs a function that represents the interaction among attributes. This function is then added to the original attribute set, and data samples are redescribed to proceed learning.

DCI needs to search for a function that highlights interactions. The space of all functions defined over all subsets of attributes grows exponentially with the number of attributes. As shown at the end of previous chapter, this space can be decomposed into two spaces: $\mathcal{S}_S$ the space of all subsets of attributes, and $\mathcal{S}_{F_i}$ the space of all functions defined over a particular subset (see Figure 2.6). Similarly, the goal of FC is broken down into two subgoals: searching subset of interacting attributes and finding a function that defines the interaction, S-Search and F_{S_i}-Search respectively. These two tasks are, however, not independent and should be integrated together to reflect the effect of each search space on the other, as illustrated in Figure 2.7. Since the space of subsets of attributes is large and has several local optima, S-Search in DCI applies genetic search, more precisely, GA. Each attribute subset generated in S-Search is given to F_{S_i}-Search for further proceeding. F_{S_i}-Search finds the best function over a given subset through analyzing the training data and inducing the relation among given attributes. The induced function is then evaluated to determine the fitness of the subset in S-Search.

Details about DCI are given in the next sections. Section 3.1.1 describes individual's representation in S-Search. This section also explains how F_{S_i}-Search is performed for each generated individual. Sections 3.1.2 and 3.1.3 describe fitness function and genetic operators respectively. Section 3.1.4 details about GA parameters and explains the DCI algorithm as an integration of S-Search and F_{S_i}-Search.

3.1.1 Individual's Representation

S-Search uses GA to find the subset of interacting attributes. Each individual in the GA is an attribute subset represented by a bit-string of length N, where N is the number of original attributes. Each bit represents presence or absence of an attribute in the subset. For instance, if the original attribute set is $S = \{x_1, x_2, \ldots, x_{12}\}$, then an individual such as $Ind_i = \langle 001000100100 \rangle$ indicates that only the third, seventh, and tenth attributes are included in the subset; that is, $S_i = \{x_3, x_7, x_{10}\}$. This kind of representation allows the use of a classical GA. The first population is initialized by generating p (i.e., the population size) random bit-strings of length N.

Each individual as a subset is associated with a function. F_{S_i}-Search, the FC module

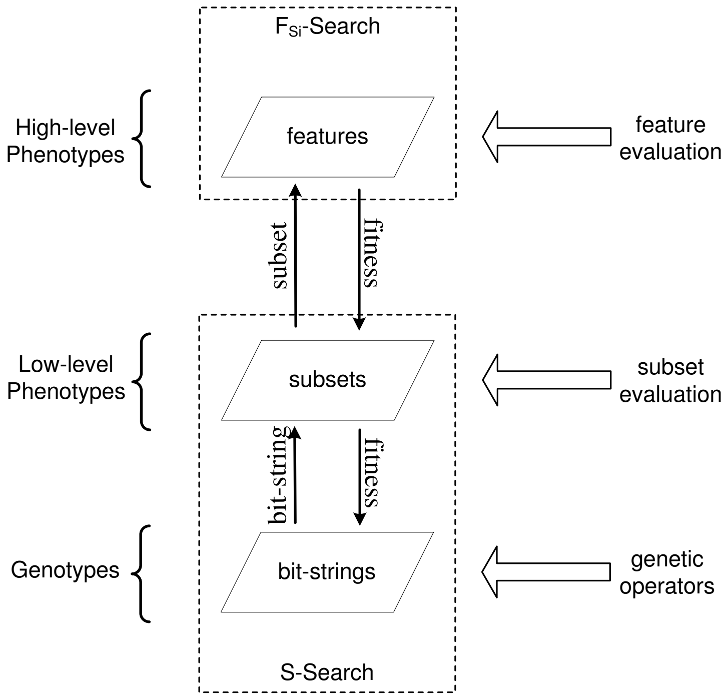


Figure 3.1: Low-level and high-level phenotypes in DCI

in DCI, aims to capture and encapsulate the relation among given attributes in a feature f_i defined as a function over S_i, as explained next. Thus, each individual S_i actually represents a function f_i. Bearing in mind that the constructed function is then used for determining the fitness of the subset as a GA individual, F_{S_i}-Search has the main role in guiding the method toward optimal solution.

Note that F_{S_i}-Search can be viewed as a function $\mathcal{F} : \mathcal{S}_S \rightarrow \mathcal{S}_{F_i}$ that maps each subset onto a feature; that is, for each subset there is one and only one feature. Each bit-string as a genotype represents a subset S_i, which in turn represents a feature f_i. In fact, GA in DCI has two levels (Figure 3.1): a lower level, where genotypes are mapped onto attribute subsets as *low-level phenotypes*; and a higher level, where subsets are mapped onto features as *high-level phenotypes*. The fitness of each genotype is determined by evaluating corresponding phenotypes in both levels (for details about fitness function see Section 3.1.2). During mutation and crossover operations over genotypes, subsets (low-level phenotypes) are evolved to generate better ones. Meanwhile, the integration of F_{S_i}-Search into S-Search provides the ability to link search spaces so that if a subset is changed, the associated function (high-level phenotype) is also changed. Therefore, GA not only is used for attribute selection in the lower level, but also indirectly evolves functions in the higher level to construct better ones. The mapping of genotypes onto low-level phenotypes, and then low-level phenotypes onto high-level phenotypes allows the decomposition of the FC task into two easier tasks: finding interacting attributes and discovering interactions.

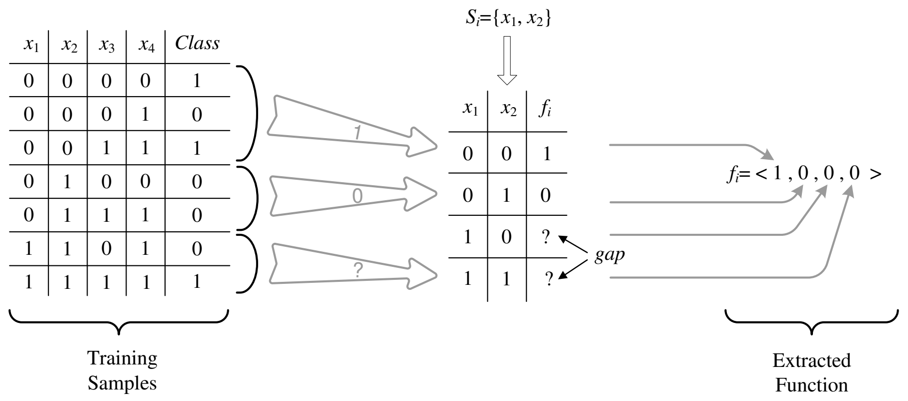


Figure 3.2: Extracting function f_i, defined over S_i, from training data in DCI: for $(x_1, x_2) = (0, 0)$ and $(x_1, x_2) = (0, 1)$ the majority labels are '1' and '0' respectively. No sample in data matches with $(x_1, x_2) = (1, 0)$ and there is no majority label for samples that match with $(x_1, x_2) = (1, 1)$. The outcomes of f_i for these two tuples are determined stochastically with the probability of 0.6 for '0' and 0.4 for '1'.

The proposed representation of individual as genotype and two levels of phenotypes also permits the use of GA instead of GP, because genetic operators are performed over genotypes which are bit-strings (while they indirectly evolve functions). As described in Section 2.3, most genetic CI methods use GP with algebraic feature representation, often parse trees, which adds difficulties to FC. The proposed representation is expected to reduce problems related to FC.

F_{S_i}-Search constructs functions by extracting the relations among attributes from data. For any given subset $S_i = \{x_{i1}, \ldots, x_{im}\}$, it defines a function by assigning a class label (as outcome of the function) to each tuple in Cartesian product $x_{i1} \times \ldots \times x_{im}$ (as input of the function). F_{S_i}-Search analyzes data to induce the tuples' labels. For each tuple t_j, that represents the j^{th} combination of attribute values in the Cartesian product, the label is determined by the majority label in set of examples in training data that match with this combination of attribute values. If there is a *gap*, that is, there is not such a majority or no example in data matches with this combination of attributes, then F_{S_i}-Search generalizes the label from all training data. For each gap a label is selected stochastically according to label's probability ratio in training data. When all tuples' labels are determined, then the function f_i is represented by a vector of values that shows the outcome of the function for each combination of attribute values in subset S_i. As an example, see Figure 3.2, which shows a training data set and a function defined over subset $S_i = \{x_1, x_2\}$ and induced from data. Once labels are stochastically selected for gaps, f_i can be defined as $f_i = \langle 1, 0, 0, 0 \rangle$.

Note that the proposed method with non-algebraic representation does not search for a function, but directly constructs the function by analyzing data, inducing a relation among given attributes, and generalizing the relation with its bias (labeling gaps).

3.1.2 Fitness Function

An attribute subset, as an individual of the GA population, is assigned a fitness value estimated by means of the fitness of the function defined over attributes and the number of attributes participating in constructing the function (attributes in the subset). If the fitness of the function f_i defined over S_i is $\mathit{Fitness}(f_i)$, then the fitness of individual Ind_i is calculated as:

$$\mathit{Fitness}(\mathit{Ind}_i) = \mathit{Fitness}(f_i) + \epsilon \times \frac{|S_i|}{|S|} \ , \tag{3.1}$$

where $|S_i|$ is the size of attribute subset S_i, and $|S|$ is the size of the original set of attributes. GA aims to minimize this formula. The first term of the formula uses the entropy concept [Quinlan, 1993; Shannon, 1948] as explained next. This term has the major influence on fitness value of the individual. The last term estimates the complexity of the new feature, by measuring the amount of attributes participating in the feature construction. 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, $\mathit{Fitness}(f_j) = \mathit{Fitness}(f_k)$. Therefore, between two equally good subsets, this formula assigns a better fitness value to the smaller.

The fitness of the constructed function can be determined by a hypothesis-driven or a data-driven evaluation. Recalling, from Section 2.3, the problem of hypothesis-driven fitness evaluation, a data-driven evaluation is preferred. The fitness of the new feature f_i, for the training data T, is measured by the following formula:

$$\mathit{Fitness}(f_i) = \mathit{Entropy}(T|f_i) \ , \tag{3.2}$$

where $\mathit{Entropy}(T|f_i)$ is the conditional entropy [Quinlan, 1986] defined as follows:

$$\mathit{Entropy}(T|f_i) = \sum_{v \in \mathit{Values}(f_i)} \frac{|T_v|}{|T|} \mathit{Entropy}(T_v) \ . \tag{3.3}$$

T_v in this formula is the subset of data for which the outcome of the feature f_i is v. $|T|$ and $|T_v|$ are the number of samples in T and T_v. $\mathit{Entropy}(T_v)$ is the average amount of information needed to identify the class of a case in T_v, calculated as:

$$\mathit{Entropy}(T_v) = -\sum_{j=1}^{k} \frac{\mathit{freq}(C_j, T_v)}{|T_v|} \times \log_2 \left(\frac{\mathit{freq}(C_j, T_v)}{|T_v|} \right) \ , \tag{3.4}$$

where $\mathit{freq}(C_j, T_v)$ is the number of cases in T_v belonging to class C_j, and k is the number of classes.

Conditional entropy (Formula 3.3) measures amount of uncertainty produced by the new feature f_i. GA tries to minimize this amount. This measure does not consider the complexity of the new feature. If only this measure is taken into account in Formula 3.1, features with small entropies are constructed. These features classify training data perfectly. But when they are evaluated on test data, they may produce errors. This is because GA does not consider the complexity of functions in fitness evaluation; and therefore, it constructs large functions that overfit training data and produce errors on test data. The second term in 3.1 is added to avoid overfitting and force GA to prefer small features to large features.

When fitness of individuals is measured, genetic operators are applied to generate a new population of attribute subsets.

3.1.3 Genetic Operators

DCI uses the following genetic operators: reproduction, crossover, and mutation. These operators are slightly different from those used in the classical GA of Section 2.2.1.

Reproduction

Section 2.2.1 explained that in a classical GA, genetic operators replace the whole population by a new one. This approach is called *non-overlapping* population model, which is useful for optimization problems. However, for machine learning and FC, it is more appropriate to apply *overlapping* population model; that is, to maintain good solutions while better solutions are constructed during each generation [Goldberg, 1989]. This approach, contrary to the classical approach, only replaces a few individuals in the population. If the population size is p, the new population contains m most fit individuals of the previous population (where $m \neq p$) and $p-m$ individuals are replaced using crossover and mutation operations.

Overlapping approach is used in DCI. First, reproduction operator copies m best individuals to the next population. Then, it selects $p - m$ individuals and copies them to mating pool for mutation and crossover operations. Proportional selection is used for choosing individuals (see Section 2.2.1).

Crossover and Mutation

When individuals are copied to the mating pool, crossover and mutation is applied. The crossover operator, explained in Section 2.2.1, is the simplest crossover called one-point crossover. It exchanges substrings (segments) of parents. DCI uses *uniform crossover* [Spears and DeJong, 1991], which is a generalization of one-point crossover. The uniform crossover exchanges bits of parents rather than substrings. For each bit in the first offspring, this operator decides with a probability p_u whether this bit is inherited from the first parent or the second parent. The second offspring inherits from the other parent for that bit. As an example consider two parents are:

$$parent_1 =< a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8 >$$

and

$$parent_2 =< b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8 > \quad .$$

Then, two offspring may be generated as follows, where first, third, forth, and sixth bits of the first child are from the first parent, and the rest are from the second parent; and vice versa for the second child:

$$child_1 =< a_1, b_2, a_3, a_4, b_5, a_6, b_7, b_8 >$$

and

$$child_2 =< b_1, a_2, b_3, b_4, a_5, b_6, a_7, a_8 > \quad .$$

This form of crossover provides the ability to explore the search space in an unbiased manner, which turns out to be more useful for large search spaces [Spears and DeJong, 1991; Spears and Anand, 1991]. To illustrate this point, assume that two parents are:

$$parent_1 =< 0, 0, 0, 0, 0, 0, 0, 0 >$$

and

$$parent_2 =< 1, 1, 1, 1, 1, 1, 1, 1 > \quad .$$

One-point crossover can produce seven different children for seven crossover points. Uniform crossover can produce any child anywhere in the search space. So, the uniform crossover produces more diversity. However, for problems where the relative location of bits is important, the uniform crossover may destroy the good characteristics of parents by exchanging bits. In DCI, the relative location in the string is not important. Therefore, the uniform crossover will not have disruptive effect. So, it provides both exploitation (constructing offspring with characteristics similar to parents) and exploration (diversity) in the search space. The p_u parameter controls the balance between exploration and exploitation. $p_u \gg 0.5$ or $p_u \ll 0.5$ results in a child more similar to the first or second parent respectively (a small jump in search space). $p_u \simeq 0.5$ produces a child that inherits almost half of one parent and half of another (a bigger jump in search space). For DCI this parameter is set to 0.6, which is the default value in the GA library [Levine, 1996] used for implementation.

Mutation in DCI is similar to the one in the classical GA (see Section 2.2.1). The only difference is that more than one bit may be changed by this operator. The probability that a bit is negated by mutation is equal to the reciprocal of the bit-string length.

Table 3.1: Modified parameters of PGAPack for DCI

GA Parameter	New Value
`Population Size` (p)	50
`Max Iteration`	100
`Max no Change Iter.`	25
`Max Similarity Value`	80%
`No. of Strings to be Replaced` ($p - m$)	46
`Selection Type`	Proportional
`Crossover Type`	Uniform
`Mutation and/or Crossover`	And

3.1.4 DCI's Algorithm

Figure 3.3 summarizes DCI's algorithm. PGAPack library [Levine, 1996] is used for implementing GA, with default parameters except those indicated in Table 3.1 (see Appendix B for default parameters of PGAPack). In this table "`Max Iteration`", "`Max no Change Iter`", and "`Max Similarity Value`" determine three stopping conditions as termination criteria of GA. The first condition is when a maximum number of iterations, "`Max Iteration`", is reached. The second one is when no change in the best solution is found in a given number of iterations, specified by "`Max no Change Iter`". The last one is when a percentage of individuals in the population, defined by "`Max Similarity Value`", have the same fitness value. "`No of Strings to be Replaced`" determines the mating pool size. "`Selection Type`" specifies how individuals are selected for reproduction, which is set to proportional selection that is the most common form in classical GA (Section 2.2.1). The last parameter is used to decide whether individuals in mating pool undergo both crossover and mutation (`And`) or undergo either crossover or mutation and not both of them (`Or`). It is set to "`And`" that is similar to what in classical GA is traditionally used; after applying crossover operator, the pair of offspring undergoes mutation. For reproducibility purposes the PGAPack random seed is initialized to one.

When any of the three stopping conditions is fulfilled GA is terminated. Then, DCI adds the constructed function associated with the best GA's individual (attribute subset) to the original set of attributes and updates data using the new set of attributes. The updated data are then given to a standard learner for learning.

3.2 Experiments

The following sections describe experiments conducted to evaluate DCI and compare it with other methods. Section 3.2.1 explains primary experiments performed to empirically evaluate the performance of DCI in terms of accuracy. The purpose of these experiments is to examine whether the feature that is constructed by DCI improves the

```
DCI
Receives: training and test data and the set of $N$ original attributes
Returns: a new representation of training and test data

  1. call $S$-Search.
  2. add the output of $S$-Search as a new feature to the set of original attributes,
     and redescribe training and test data.

  3. return new attribute set and data.

$S$-Search
Receives: training data and a set of $N$ original attributes
Returns: a new feature

  1. Generate a population of bit-strings of length $N$ each representing an at-
     tribute subset $S_i$.
  2. For each individual $S_i$, call $F_{S_i}$-Search, assign the function generated by
     $F_{S_i}$-Search to $S_i$, and calculate the fitness according to Formula 3.1.
  3. Unless predefined stopping conditions are achieved, apply genetic operators,
     generate a new population, and go to step 2.
  4. Return the function defined over the best individual.

$F_{S_i}$-Search
Receives: training data and subset $S_i$
Returns: $f_i$

  1. Given the subset $S_i$, consider Cartesian product of attributes in $S_i$ and for
     each tuple $t$ in the product do:
       (a) if there is a majority label in samples that match with $t$, assign this
           label to $f_i(t)$.
       (b) if there is no such a majority or there are no training samples that
           match $t$, assign a class label to $f_i(t)$ stochastically, according to the
           class distribution in the training data.
  2. return $f_i$
```

Figure 3.3: DCI, S-Search, and F_{S_i}-Search algorithms

Table 3.2: Comparing average accuracies of DCI and standard learners

Concept	Relev. atts	Irrel. atts	Maj %	C4.5 Org. Atts	C4.5-Rules Org. Atts	C4.5 Rel. Atts	C4.5-Rules Rel. Atts	DCI + C4.5
$\wedge$(P$_{1,4}$, P$_{3,6}$)	6	6	75.0	72.1(3.6)	70.9(2.9)	87.9(5.5)	94.4(4.8)	97.7(2.2)
$\wedge$(P$_{1,6}$, P$_{3,8}$)	8	4	75.0	72.2(3.3)	69.0(3.5)	73.2(2.4)	69.7(2.4)	81.7(1.3)
$\wedge$(P$_{1,3}$, P$_{3,5}$, P$_{4,6}$)	6	6	87.5	87.4(1.5)	85.1(3.2)	91.7(3.6)	96.4(2.8)	99.2(1.4)
$\wedge$(P$_{1,4}$, P$_{2,5}$, P$_{3,6}$)	6	6	87.5	87.5(0.1)	84.2(2.0)	90.6(2.5)	95.8(3.9)	99.4(0.8)
$\wedge$(P$_{1,4}$, P$_{3,6}$, P$_{5,8}$)	8	4	87.5	87.5(0.1)	84.2(3.0)	87.5(0.1)	85.4(1.8)	89.2(1.2)
$\wedge$(P$_{1,6}$, P$_{2,7}$, P$_{3,8}$)	8	4	87.5	86.5(2.2)	83.9(2.7)	87.0(1.6)	86.2(1.9)	89.0(1.1)
Average	7	5	83.3	82.2(1.8)	79.6(2.9)	86.3(2.6)	89.0(2.9)	92.7(1.3)

performance of a standard leaner such as C4.5 [Quinlan, 1993] in presence of complex attribute interaction. Section 3.2.2 empirically compares the method with other CI methods to highlight advantages and deficiencies of DCI.

Synthetic concepts with complex interactions are used for experiments. These concepts represent sources of difficulty that appear in real-world domains when primitive attributes are used for representing data. The use of synthetic concepts allows analyzing system's behavior deeply before moving on to try to solve real-world problems with difficulties similar to those exemplified by these synthetic concepts.

3.2.1 Empirical Evaluation

For empirical analysis of DCI, synthetic concepts are defined over a set of twelve Boolean attributes $\{x_1, x_2, \ldots, x_{12}\}$. Appendix A gives the definition of each concept. These concepts are conjunctions of two or three parity functions. Parity is an extreme case of complex attribute interaction where each class label is scattered across the data space (Section 1.1).

Columns one to four of Table 3.2 give a summary of these concepts. The numbers of relevant and irrelevant attributes for each concept are shown in columns 2 and 3, respectively. All relevant attributes participate in interactions. The main characteristic of these concepts is that there are *shared attributes* in concepts; that is, there are some attributes that participate in more than one complex interaction, which makes the concept more difficult to learn by greedy methods. For example, x_3 and x_4 in $\wedge$(P$_{1,4}$, P$_{3,6}$) are shared by P$_{1,4}$ and P$_{3,6}$. In all concepts some irrelevant attributes exist. The existence of irrelevant attributes makes concept difficult since relevant attributes are easily confounded with irrelevant attributes. The forth column shows the majority class percentage as a trivial baseline performance to compare accuracy with.

The accuracy of standard learners are compared before and after adding constructed features to the set of original attributes. C4.5 and C4.5-Rules [Quinlan, 1993] are used as standard learners, with default parameter's values. Each experiment is run 10 times over 10 sets of shuffled data and the average accuracy is calculated. For each trial, 5% of all 2^{12} data are used for training and the rest (95%) are kept unseen as test data for evaluating predictive accuracy.

Three sets of experiments are performed over each concept. For the first set of experiments C4.5 and C4.5-Rules are performed over original training data set and the accuracies are measured using the 95% unseen data as test data. The average accuracies of 10 runs are shown in the fifth and sixth columns of Table 3.2. The second set of experiments intends to ease the learning task for the learners. C4.5 and C4.5-Rules are forced to only use attributes that are manually marked as relevant. The purpose is to evaluate the utility of a perfect preprocessing feature selection for learning these concepts. The results are shown in columns 7 and 8. The last set of experiments is performed using DCI to construct a feature. Then, the new feature is added to the set of original attributes, data are redescribed, and the predictive accuracy of C4.5 [Quinlan, 1993] on modified 95% test data is measured. The result is presented in the last column of Table 3.2. For all experiments results for C4.5 are obtained after tree pruning since for all concepts the average accuracy with pruning was better than those without pruning. Numbers between parentheses in the table indicate standard deviations.

As it can be seen from the table, the accuracy of C4.5 and C4.5-Rules using original attribute set is lower than or equal to the majority class percentage for all concepts. Thus, if data are classified using the label of majority class, a better result is obtained, which means the learners do not learn the concepts. This shows the complexity of these concepts for standard learners.

Forcing C4.5 and C4.5-Rules to only consider relevant attributes helps them to achieve higher accuracy compared to the results in columns 5 and 6. However, the average accuracies for three of six concepts is still not better than the majority class percentage. This implies that guiding the learner to only consider interacting attributes does not help it to learn these complex concepts.

Comparing results in the last column with those in columns 7 and 8 shows that features constructed by DCI result in higher accuracies. This means FC improves accuracy over the standard learners preceded by a perfect feature selection phase. DCI outperforms other accuracies for all concepts with a significant level of 0.01, except for concept $\wedge(P_{1,4}, P_{3,6})$ that it outperforms C4.5-Rules preceded by manual feature selection with a significant level of 0.1 (using t-distribution test).

Two importance are concluded from these primary experiments. First, even if a feature selection method successfully detects all interacting attributes, still the underlying relation among attributes and the target concept is opaque to the learner. FC is needed to abstract the interactions into new features and highlight them to the learner. Second, the results show that the new genetic method considerably facilitates learning these concepts by constructing new features. However, it is necessary to compare this method with other CI methods.

3.2.2 Empirical Comparison

This section describes experiments performed to empirically compare DCI with other CI methods. The purpose is to evaluate the utility of two main characteristics of DCI which are genetic-based search and non-algebraic representation of features. DCI is compared with methods that do not have at least one of these two characteristics. These methods are Fringe, Grove and Greedy3 [Pagallo and Haussler, 1990], LFC [Ragavan and Rendell, 1993], and MRP [Pérez and Rendell, 1995]. All these methods are greedy CI methods. Fringe, LFC and MRP are briefly studied in Chapter 2. Grove and Greedy3 evaluate attributes individually to select the best attribute to be incorporated into the current feature under construction. Grove constructs features to generate a decision list, while Greedy3 aims to construct a DNF expression. All the methods apply an algebraic form for representing features except MRP. This method applies backward elimination (Section 2.1.2) to find subset of interacting attributes. For each subset S_i it constructs a feature that is represented by all positive or negative tuples obtained by projecting data onto attributes in S_i. Similar to DCI, this method extracts the label of each tuple from data. But different criteria are used by each method. Results are also compared with C4.5 and C4.5-Rules using original attribute set as the baseline performance.

Since the implementation of the greedy CI methods were not available, same experiments as those reported in [Perez, 1997] are performed over same synthetic concepts and the accuracy values are compared. All concepts are defined over 12 Boolean attributes and consist of complex interactions (see Appendix A for definition of each concept). The main point about these concepts is that their underlying interactions are too complex to be represented by an algebraic form. However, they can be represented by non-algebraic representation easily regardless of how complex they are. A summary of these concepts is given in Table 3.3. Number of relevant and irrelevant attributes and the majority class percentage for each concept are shown in columns 2, 3, and 4, respectively. Concepts are grouped according to the number of complex interactions in the concept and ordered by the number of relevant attributes. The top part of table includes concepts with one complex interaction and the bottom part includes those with two or more complex interactions.

Experiments are similar to those in Section 3.2.1, but this time, each trial is run 20 times instead of 10 times, over 20 sets of shuffled data to make the results comparable to those in [Perez, 1997]. For each experiment 5% of all 2^{12} data are used for training and the rest are kept unseen as test data for final evaluation.

The fifth column of Table 3.3 shows the accuracy of C4.5 using original attribute set. Column 6 represents the best results among C4.5-Rules, Fringe, Grove, Greedy3 and LFC as reported in [Perez, 1997]. The letters cr, fr, gr, $g3$, and lf indicate that the best competitor is C4.5-Rules, Fringe, Grove, Greedy3 or LFC respectively. Since MRP applies non-algebraic representation, comparing its accuracy with DCI is challenging.

Table 3.3: Comparing average accuracies of DCI with other CI methods on 5% training data

Concept	Relev. atts	Irrel. atts	Maj %	C4.5 Org. Atts	Prior best result	MRP	DCI + C4.5
$P_{1,4}$	4	8	50.0	60.7(9.3)	99.6 fr	100(0.0)$\triangleleft$	100(0.0)$\triangleleft$
$\mathrm{gw}_{5,8}$	4	8	68.8	100(0.0)$\triangleleft$	100 $g3$$\triangleleft$	100(0.0)$\triangleleft$	100(0.0)$\triangleleft$
$\mathrm{sw}_{5,8}$	4	8	62.5	71.6(9.0)	100 $fr$$\triangleleft$	100(0.0)$\triangleleft$	100(0.0)$\triangleleft$
$\mathrm{mx}_6\mathrm{c}_{6,7}$	4	8	50.0	87.1(5.8)	98.7 fr	100(0.0)$\triangleleft$	100(0.0)$\triangleleft$
$\mathrm{mj}_{4,8}$	5	7	50.0	96.0(2.8)	99.0 gr	99.8(0.7)$\triangleleft$	99.7(1.5)
$P_{1,6}$	6	6	50.0	48.8(1.2)	51.1 fr	97.0(1.6)	98.0(1.8)$\triangleleft$
$\mathrm{mx}_6\mathrm{c}_{5,8}$	6	6	50.0	71.1(2.3)	76.4 fr	96.4(3.7)	97.8(1.8)$\triangleleft$
$\mathrm{mj}_{3,9}$	7	5	50.0	80.4(1.4)	86.6 gr	85.2(5.1)	**89.1(2.7)**$\triangleleft$
$\mathrm{nm}_{4,5,7}$	7	5	56.3	70.6(2.8)	74.1 fr	86.7(7.6)	89.9(1.6)$\triangleleft$
$\mathrm{rk}_{5,7}$	7	5	83.6	81.3(1.6)	89.9 gr	91.4(4.7)	**95.1(1.7)**$\triangleleft$
$\mathrm{rk}_{6,7}$	7	5	94.5	94.1(0.7)	95.6 gr	96.8(2.3)	**98.3(1.3)**$\triangleleft$
$P_{1,8}$	8	4	50.0	48.5(0.2)	49.4 cr	75.1(1.4)	76.7(2.7)$\triangleleft$
Average	6	6	59.6	75.9(3.1)	85.0	94.0(2.3)	95.4(1.3)$\triangleleft$
$\mathrm{cp}_{5,8}$	4	8	75.0	78.4(7.5)	100 $fr$$\triangleleft$	100(0.0)$\triangleleft$	100(0.0)$\triangleleft$
mx_6	6	6	50.0	94.9(3.1)	100 $fr$$\triangleleft$	96.1(7.3)	97.8(2.1)
$\mathrm{cdp}_{3,11}$	6	6	62.5	81.3(8.2)	98.4 $fr$$\triangleleft$	97.3(3.9)	97.6(1.6)
$\mathrm{cp}_{4,9}$	6	6	75.0	73.1(2.7)	86.1 fr	**99.0(1.6)**$\triangleleft$	97.2(1.8)
$\mathrm{cp}_{3,10}$	8	4	75.0	73.1(2.6)	73.4 cr	**88.9(1.2)**$\triangleleft$	81.6(1.6)
$\mathrm{cdp}_{2,10}$	9	3	62.5	66.0(9.8)	78.1 fr	**92.0(6.5)**$\triangleleft$	67.7(2.0)
$P_{3,6} \vee (2)$	10	2	61.7	59.4(5.0)	70.5 fr	**97.1(2.9)**$\triangleleft$	57.9(2.2)
$P_{3,6} \vee (3or2)$	10	2	77.3	75.5(2.2)	75.1 fr	**80.4(6.0)**$\triangleleft$	69.5(2.2)
$P_{3,6} \vee (3)$	10	2	65.6	60.9(2.9)	68.0 fr	**95.9(4.4)**$\triangleleft$	59.3(1.7)
$\mathrm{cdp}_{1,9}$	12	0	62.5	56.6(4.9)	65.5 cr	**81.3(3.0)**$\triangleleft$	56.4(2.5)
Average	9	3	70.5	73.8(4.0)	80.8	**91.2(3.9)**$\triangleleft$	76.8(1.8)

For this reason the accuracy values of MRP are separated from other greedy CI methods and shown in column 7. Features constructed by DCI are evaluated by running C4.5 and obtaining its predictive accuracy on modified test data. This result is shown in the last column of table. Numbers between parentheses indicate standard deviations. The highest average accuracy is marked by $\triangleleft$ for each concept. Bold means that with a significant level of 0.02 this accuracy is the best between MRP and DCI, using t-distribution test.

It can be seen from Table 3.3 that for both groups of concepts when few attributes are involved in interactions most CI methods, including DCI, improve accuracy. When the ratio of relevant attributes to total number of attributes is low (less than $\frac{1}{2}$), concept is easier to learn by a CI method because there is enough data to see the whole structure of the interaction in concept. Although only 5% of all 2^{12} data, that is 205 samples, are available for training, there are enough replications of the relation among attributes in data. For instance, if the number of interacting attributes is 4, a sample set of $2^4 = 16$ cases can be enough to represent all the interaction. Since 205 samples are provided for learning, there is a high probability that the relation is repeated in data several times. This replication makes the interaction more apparent for a CI method. Hence, CI easily discovers the structure of the relation and learns the concept. The important point here is that MRP and DCI obtain 100% accuracy for all concepts with four relevant attributes. This is because these methods apply non-algebraic representation and extract functions from data. They easily see the replication of the interaction in data and induce a non-algebraic function that represents the interaction. Other CI methods which apply algebraic representation sometimes cannot construct functions correctly since the algebraic representation of these interactions is complex and more difficult to construct.

When the number of interacting attributes is increased, the accuracy of most CI methods is scaled down. MRP slightly gives better accuracy on concepts with high number of interacting attributes comparing to CI methods of column 6 because of its non-algebraic representation of features. DCI outperforms MRP for almost all concepts of the top part of Table 3.3. This shows that when one complex interaction exists among attributes, DCI successfully finds interacting attributes and constructs a function over them to encapsulate interaction into a new feature and improve accuracy. The higher accuracy of DCI comparing to MRP in almost all such concepts shows the advantage of using GA for finding subset of interacting attributes. MRP applies backward elimination search. As explained in Section 2.1.1, this form of one-by-one attribute elimination is not appropriate when complex interaction exists among several attributes since the method may lead to local optima for the high variation in the search space. As it can be seen from table, in cases that MRP achieves lower accuracy than DCI, the standard deviation is usually high, which indicates that in some trials of 20 experiments, this method could not find the proper subset of interacting attributes due to its local search. The global search of DCI reduces the local optima problem. For this reason, when the number of interacting attributes is high and the search space has high variation, the accuracy of DCI is better than MRP and other methods. In fact, in all experiments in this group of concepts, DCI successfully finds the subset of interacting attributes except for one of 20 experiments over concept $\mathrm{rk}_{6,7}$.

However, in the bottom part of table the results are vice versa. For these concepts MRP obtains significantly higher accuracy than DCI when number of interacting attributes is increased. This is due to the structure of these concepts. They consist of two or more complex interactions. For instance, $\mathrm{cp}_{i,j}$ is composed of conjunction of two parities. The underlying high-order interaction, that is conjunction of parities, can be represented by two smaller interactions, each representing a parity function. MRP, in spite of using greedy search, sometimes successfully breaks down the difficult task of constructing one complex function that represents all interactions into easier tasks of constructing several smaller functions, each representing one interaction. In case of $\mathrm{cp}_{i,j}$, for example, MRP first constructs a function that represents one of two parities. Then, it uses this function to split data and construct the other function that represents the second parity. As explained in Section 2.1.2, sometimes constructing more than one function is necessary to ease learning concept with several interactions. For these concepts, DCI successfully detects all attributes involved in interactions in almost all experiments due to its global search. But then, it tries to construct a single function

Table 3.4: Comparing average accuracies of DCI with other CI methods on 1% training data

Concept	Relev. atts	Irrel. atts	Maj %	C4.5 Org. Atts	Prior best result	MRP	DCI + C4.5
$P_{1,4}$	4	8	50.0	50.9(4.0)	50.7	81.7(16.0)	**94.6(11.1)**◁
$gw_{5,8}$	4	8	68.8	83.6(6.1)	90.3	81.6(11.1)	93.2(7.7)◁
$sw_{5,8}$	4	8	62.5	59.1(7.2)	58.6	76.9(18.3)	**95.6(4.1)**◁
$mx_6c_{6,7}$	4	8	50.0	59.7(5.5)	61.8	81.2(17.4)	**96.5(3.8)**◁
$cp_{5,8}$	4	8	75.0	68.4(7.3)	74.4	88.6(12.1)	**95.6(4.1)**◁
Average	4	8	61.3	64.3(6.0)	67.2	82.0(15.0)	**95.1(6.2)**◁

that encapsulates several interactions. When few attributes participate in interaction and enough data are available, this method can discover the interaction and construct a function that abstract it, as seen for $cp_{5,8}$. But constructing a feature that represents interactions becomes difficult for this method when the number of interacting attributes grows and few training data are available. In spite of correctly detecting interacting attributes, DCI cannot construct a proper function that represents all complex interactions in the concept. For this reason, its performance degrades while the number of interacting attributes grows. However, it still improves the accuracy of C4.5 by constructing new features for some concepts in this group. In cases that the accuracy of C4.5 using original data set is better than the accuracy obtained by DCI, the results are lower than majority class percentages.

For the same reason, DCI's accuracy degrades rapidly for concepts in Table 3.2 of the previous section when the number of interacting attributes is increased. These concepts are also composed of several complex interactions.

Note that MRP outperforms DCI only for second group of concepts. For concepts in the first group such as $P_{i,j}$, MRP cannot break down interaction into parts. Therefore, similarly to DCI, MRP constructs only one feature to represent the interaction; and due to its local search obtains lower accuracy than DCI.

The good performance of all CI methods for concepts with small number of interacting attributes in Table 3.3 implies that the concepts are easy for these methods. In order to compare the methods over these concepts experiments are repeated but only 1% of data are used for training and 99% are kept unseen for final evaluation. The size of training data is reduced to make the concepts more difficult to learn. This is closer to the situation of real-world problems. The results are shown in Table 3.4 with the same format as Table 3.3.

Since the size of training data is very small (41 samples), the standard deviation is increased for all methods. The differences among accuracies of CI methods are now clearer. The results in Table 3.4 show that when a complex interaction exists among attributes, with a small number of training data, most CI methods with algebraic form of representation fail to construct a useful feature. These methods obtain an accuracy value almost equal to the majority class percentage for 3 of 5 concepts. DCI and MRP, due to their non-algebraic representation, are less sensitive to data size than methods with algebraic feature representation. DCI outperforms MRP because of its global search for selecting attributes. The small number of training data produces more variation in search space; therefore, the local search of MRP fails to find optimal subset of interacting attributes. DCI successfully finds the interacting attributes in all experiments except one of 20 experiments over concepts $gw_{5,8}$ and $P_{1,4}$. The high accuracy of DCI comparing to other CI methods indicates that the size of training data is less problematic for DCI than the others. As the bold accuracies in Table 3.4 show, the accuracy of DCI is significantly better than all methods in most cases with a significant level of $\alpha = 0.02$, using t-distribution test.

3.3 Conclusion

As the primary step to analyze the first two parts of the hypothesis of this dissertation (Section 1.3), this chapter proposed a new CI method, DCI. DCI is a method based on genetic-search and non-algebraic representation of features, whose goal is to ease learning problems with complex attribute interaction. The method decomposes the search space into two spaces: the space of subsets of attributes, and the space of features. The integration of two searching tasks in DCI maintains the effect of each search space to the other while improving each task. The design of DCI made the method capable to apply GA instead of GP. GA is used to search the space of subsets of attributes. Since each subset is associated with a function defined over it, GA indirectly evolves features to construct better ones.

Empirical results showed that DCI performs well on concepts with complex interaction and outperforms other CI methods in terms of accuracy when concept is composed of one complex interaction. The genetic approach of DCI makes this method more promising than other methods in finding interacting attributes and functions over them when the search space is large and with high variation.

The non-algebraic representation of features used by DCI 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. Furthermore, this form of representation provides the ability to extract features directly from training data. For this reason, when few data are available for learning, the non-algebraic representation of DCI and MRP makes these methods more promising than other CI methods. Though, since MRP applies a greedy search to find interacting attributes, its accuracy is lower than DCI with a global search. The non-algebraic representation along with GA make DCI work better than the other methods when concepts are composed of one complex interaction.

Experiments over synthetic concepts allowed a deep analysis of system's behavior that highlights an important requirement for CI methods (mentioned as the third part of the hypothesis of this dissertation in Section 1.3), which was not considered in DCI. As explained in Section 2.1.2, when concept consists of several complex interactions and the number of interacting attributes is high, the function that encapsulates the interaction is complex and difficult to construct. A CI method needs to break down the high-order interaction into smaller interactions each represented by a function. Then, each function works as an intermediate concept that partially represents the whole interaction.

DCI is incapable of constructing more than one feature. When the concept consists of several interactions, it tries to find the subset of interacting attributes and construct a single function over it to abstract interactions among attributes. Although it successfully detects interacting attributes due to its global search, if the size of training data is small and the number of interacting attributes is high, DCI constructs a function that is not good enough to outline all the interactions.

Next chapter presents a new CI method that constructs a set of new features. This method maintains all advantages of DCI that are non-algebraic representation of features and genetic-based search for finding interacting attributes. However, the GA aspects of this method differ from those in DCI. Also the algorithm for feature induction in this method is improved to construct more proper features. Since this method is capable of breaking down the difficult task of constructing one complex function into smaller tasks, its performance is expected to be better than DCI. Also, since the new method applies a global search too, it is anticipated to outperform greedy methods. Experiments empirically prove this theory.

Chapter 4

MFE2/GA: Multi-Feature Extraction Using Genetic Algorithms

The previous chapter presented DCI, a genetic CI method with non-algebraic representation of features. It was shown that when complex interaction exists in concept, few training data are available, and no prior knowledge is provided, an FC method with non-algebraic operator-free representation of features is more convenient. Also, it was observed that a global search such as GA is more appropriate for searching the intractable space of attribute subsets. DCI reduced the problem of learning in presence of complex interactions for some concepts. Because of its global search, DCI successfully recognizes interacting attributes from irrelevant ones. However, this method is incapable of constructing more than one feature. It was illustrated in Section 3.2.2 that when a large number of attributes participate in interactions and few training data are provided, DCI fails to construct a proper function over interacting attributes. The function that encapsulates all interactions among attributes is complex and difficult to construct. This difficulty augments when few training data are available. CI methods should evaluate several features together as related parts of the theory that represents interactions in concept.

Motivated by this deficiency of DCI, this chapter presents MFE2/GA [Shafti and Pérez, 2005]. This CI method intends to construct a set of features representing several interactions. The method is an improved version of MFE/GA (Multi-feature Extraction based on GA) presented in [Shafti and Pérez, 2004]. Since MFE2/GA gives better results than its predecessor, this chapter focuses on the later version. The dissimilarities between the two versions are explained in Section 4.2.1. As the differences are not significant, the empirical comparison between these two is moved to Appendix C.

Section 4.1 explains the need for constructing and evaluating several features together when few training data are available. Section 4.2 describes MFE2/GA, and details the representation of GA's individuals, genetic operators, and fitness evaluation. Experiments to evaluate genetic operators in MFE2/GA are described in Section 4.3. Section 4.4 empirically compares MFE2/GA with other CI methods. The experiments in this section result in improving the method in Chapter 6 to conform to MDL Principle [Rissanen, 1983; Grunwald, 2007]. Conclusions are summerized in Section 4.5.

4.1 Why Multi-feature Construction

It was illustrated in Section 3.2.2 that when several complex interactions exist among attributes and few data samples are provided, DCI fails to construct a function that properly represents interactions. Figure 4.1 presents an example of such concepts. The target in this figure is $\wedge(WL3_{1,4}, WL3_{3,6}, WL3_{5,8})$, that is the conjunction of three interactions of form $WL3_{i,j}$ (Weight Less than 3), each one involving the four Boolean attributes x_i through x_j. The three interactions in this example are analogous and correspond to the condition that less than three of the four involved attributes, x_i to x_j, are set to one. Each box in the figure shows the positive tuples for each interaction. The concept has eight relevant attributes (x_1 to x_8) and four irrelevant attributes (x_9 to x_{12}). There are some attributes shared by two interactions. For instance, attributes x_3 and x_4 participate in both $WL3_{1,4}$ and $WL3_{3,6}$. This concept is hard because of complex interactions among attributes and existence of irrelevant attributes. The relevant attributes are likely to be mistaken as redundant attributes, and irrelevant attributes may be misinterpreted as relevant. Also, the existence of shared attributes makes interactions more difficult to discover. But more importantly, the FC difficulty augments because the concept is composed of, not just one, but three complex interactions.

The underlying regularities in this concept are complex when represented by a single function using eight primitive relevant attributes. DCI successfully finds the subset of eight relevant attributes, and then, intends to construct a function defined over this subset and extracted from data. In order to extract this function correctly, DCI needs to see the structure of the function in data. The function is defined by $2^8 = 256$ tuples. The repetition of these tuples in data highlights the structure of the function. If few training data are available this structure may not be apparent; therefore, the constructed function may not properly represent regularities in the concept.

This function can be broken down into three functions, each defined over four attributes and abstracting one interaction (i.e., $WL3_{i,j}$). Each function is represented by $2^4 = 16$ tuples, which is considerably less than the number of tuples needed for representing a function over eight attributes. It is more likely that the training data set contains the replication of these 16 tuples. Then, an FC can successfully construct a function representing the interaction of four attributes. Hence, when few data are provided for this concept, construction of such functions each defined over four interacting

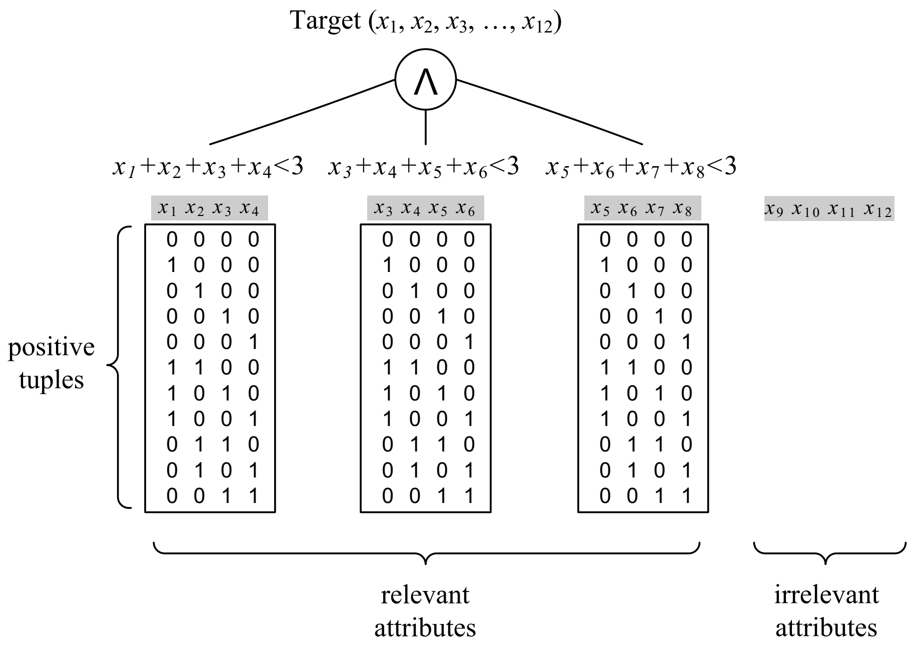


Figure 4.1: A concept composed of several complex interactions

attributes is preferred.

A CI method needs to break down the difficult task of constructing one complex feature into several easier tasks of constructing smaller features, each representing one interaction. Each constructed feature works as an intermediate concept, which forms part of the theory that highlights interactions in primitive data representation.

When several new features are required, each feature that partially shows interactions, by itself, may not give enough information about the concept. Hence, a CI method may consider the feature as an irrelevant one. Most CI methods construct features successively and evaluate them one by one; the construction of each feature depends on those previously constructed (see Section 2.1.2). If several complex interactions exist among attributes and few training data are available, it is more likely that such CI methods fail in constructing relevant features in early steps. Then, all successively constructed features will be irrelevant too. Constructing and evaluating several features together as a set of related parts of the theory is essential for a CI method in order to see the importance of each one.

Considering this requirement, MFE2/GA is designed. This method, like DCI, is a preprocessing CI method that uses GA search and non-algebraic form of representing features. However, it differs from DCI in other aspects. The main difference is in the number of functions constructed by each method. MFE2/GA aims to find a *set* of functions that best represent interactions, while DCI abstracts interactions into *one* single function. The GA in MFE2/GA provides the ability to generate and evaluate several features represented as one individual. This *multi-feature construction property* of MFE2/GA provides better experimental results than DCI and other CI methods when concept is composed of several interactions, as illustrated in Section 4.4. The construction of features is also improved in MFE2/GA. Next section describes the design of this method.

4.2 MFE2/GA's Design

MFE2/GA is a CI method that applies GA to construct a set of functions as new features. Due to following a different goal, GA design in MFE2/GA is different from DCI. The goal of DCI is to find a subset of interacting attributes, S_i, and a function f_i defined over subset S_i. So it searches the space of all functions defined over all attribute subsets. MFE2/GA's goal is to find a set of subsets of interacting attributes $\{S_1, \ldots, S_k\}$ and a set of functions $\{f_1, \ldots, f_k\}$, each function f_i defined over an attribute subset S_i. Hence, the search space for the new method is the space of different *sets* of functions defined over different sets of attribute subsets.

MFE2/GA follows the same framework presented in Section 2.4. However, the search space, here, is different; and therefore, the decomposition of the space is different too. Thus, the tasks S-Search and F_{S_i}-Search in the framework of Figure 2.7 are defined differently here. For this method, the search space of all sets of functions is decomposed into two spaces: $\mathcal{S}_{SS}$, the space of all sets of attribute subsets; and $\mathcal{S}_{SF_i}$, the space of all sets of functions defined over given attribute subsets. Thus, the task of constructing a set of functions is divided into two tasks: searching through $\mathcal{S}_{SS}$ to find the set of subsets of interacting attributes, S-search; and looking for a proper set of functions in $\mathcal{S}_{SF_i}$ defined over a given set of subsets, F_{S_i}-Search. As shown in Figure 2.7, the two tasks are integrated together. GA is used for S-Search. Each set of attribute subsets as an individual in S-Search is given to F_{S_i}-Search for constructing features. F_{S_i}-Search analyzes data and induces the best functions that can be defined over attribute subsets. The fitness of individuals in S-Search is measured by evaluating functions constructed in F_{S_i}-Search.

The rest of Section 4.2 explains the design of GA individuals, fitness function, and genetic operators in MFE2/GA. The current version of the method assumes that the class labels are binary and all continuous attributes have been converted to nominal attributes before running the system. Chapter 7 suggests techniques, as future work, for developing the system to be applicable to a wider range of domains.

4.2.1 Individual's Representation

S-Search in MFE2/GA uses GA to search the space of all sets of attribute subsets, $\mathcal{S}_{SS}$. Like DCI, two levels of phenotypes are maintained (see Section 3.1.1). But this time, if S is the set of original attributes, each low-level phenotype is a set of subsets of S,

$Ind_1 = \langle$101100101101:010101001011:000101110010:011010011001$\rangle$
$Ind_2 = \langle$111111000000:001111110000$\rangle$
$Ind_3 = \langle$111100000000:001111000000:000011110000$\rangle$

Figure 4.2: GA individuals for concept of Figure 4.1

$Ind = \{S_1, \ldots, S_k\}$, where $S_i \subset S$ and $|S_i| > 1$. Each high-level phenotype is a set of functions such as $\{f_1, \ldots, f_k\}$, where f_i is defined over S_i and extracted from data. Fitness evaluation is performed over high-level phenotypes, and genetic operators are performed over low-level phenotypes (for details about fitness evaluation and genetic operators see Sections 4.2.2 and 4.2.3 respectively).

Each subset S_i in an individual is represented by a bit-string of length N, where N is the number of original attributes; each bit showing the presence or absence of the attribute in the subset. Therefore, each genotype is a bit-string of length $k.N$ ($k > 0$) such as $Ind = \langle b_{11} \ldots b_{1N} \ldots b_{k1} \ldots b_{kN} \rangle$. Since each individual has different number of subsets, the length of individuals is variable. To avoid unnecessary growth of individuals, the number of subsets in individuals is made limited to the up bound $K = 5$ by default. K is a parameter that can be adjusted if it seems necessary.

For producing each individual in the first population, a random number k between 1 and K is generated (according to a uniform distribution) to determine the number of subsets in the individual. Then, a random bit-string of length $k.N$ is generated to represent the individual.

Note that if the genotype representation of an individual contains a subset S_i where $|S_i|$=1 or $S_i = S$, then S_i is ignored in the phenotype representation; and therefore, it does not participate in the fitness measurement (Section 4.2.2). However, it is considered in the genotype representation for GA operations to produce diversity in the population.

Figure 4.2 shows examples of three individuals generated by MFE2/GA for the concept of Figure 4.1. Colons are used to separate subsets in each individual. Ind_1 represents a set of three irrelevant subsets. Ind_2 represents two subsets of interacting attributes. The function that is constructed over the first subset encapsulates two interactions $WL3_{1,4}$ and $WL3_{3,6}$. The second function defined over the other subset represents interactions $WL3_{3,6}$ and $WL3_{5,8}$. Ind_3 corresponds to the best individual since its three attribute subsets match perfectly those involved in the three complex interactions shown in Figure 4.1. Functions defined over subsets in Ind_3 are smaller and easier to construct comparing to those defined over subsets in Ind_2.

F_{S_i}-Search assigns a high-level phenotype to each low-level phenotype by constructing new features as functions defined over attribute subsets. It receives a set of attribute subsets and for each subset analyzes data to capture and abstract the relation among attributes in the subset into a new function. F_{S_i}-Search is a function that maps each individual as a set of attribute subsets onto a set of new features; that is, $\mathcal{F} : \mathcal{S}_{SS} \rightarrow \mathcal{S}_{SF_i}$.

Thus, for each set of attribute subsets, there is one and only one set of features (recall that in the low-level phenotype representation, a subset S_i where $|S_i| = 1$ or $S_i = S$ is not appeared and, therefore, does not participate in FC). Non-algebraic form is applied for representing functions. This form of representation permits extracting the function directly from data. The function f_i for any given subset $S_i = \{x_{i1}, \ldots, x_{im}\}$ is defined by assigning binary class labels (as outcomes of the function) to all the tuples in the Cartesian product $x_{i1} \times \ldots \times x_{im}$ (as inputs of the function). The class assigned to each tuple t_j depends on the class labels of the training samples that match the tuple. A training sample matches a tuple t_j if its values for attributes in S_i are equal to the corresponding values in the j^{th} combination of attribute values in the Cartesian product. The class labels are assigned as discussed case by case next:

Case 1: Unknown tuple. If there are no training samples matching t_j, a class label is assigned to $f_i(t_j)$ stochastically, according to the class distribution in the training data.

Case 2: Pure tuple. If all training samples matching t_j belong to the same class, this is the class assigned to $f_i(t_j)$.

Case 3: Mixed tuple. If there is a mixture of classes in the samples matching t_j, the class assigned to $f_i(t_j)$ depends on the numbers of tuples labeled by Case 2 as positive (class label '1') and negative (class label '0'), p_2 and n_2 respectively. If $p_2 > n_2$, the negative class is assigned; and otherwise, the positive class is assigned to $f_i(t_j)$ (i.e., the opposite label of the majority label in function).

Note that, F_{S_i}-Search does not search for a function. It induces the function directly from data with a bias for labeling pure, mixed, and unknown tuples.

Comparing MFE2/GA and DCI, the pure and unknown tuples are labeled in the same manner in both methods. The label of mixed tuples in DCI is determined by the majority label in the training data; if there is no majority, the tuple is considered as an unknown tuple. In MFE2/GA the label of mixed tuples depends on the definitive labels in the function under construction, which are the labels of pure tuples. The opposite label to the most frequent label among pure tuples is selected. Figure 4.3 illustrates how MFE2/GA constructs a function defined over subset $S_i = \{x_1, x_2\}$ using the training data set of Figure 3.2.

Recall that MFE2/GA is an improved version of MFE/GA. These two versions only differ in labeling tuples; more precisely, in Case 3, which is labeling mixed tuples. Both determine the label of mixed tuples by the label of pure tuples, but behave differently when there is no pure tuples; that is, when $p_2 = n_2 = 0$. In that case, MFE/GA considers mixed tuples as unknown tuples (Case 1). Hence, all the tuples in the Cartesian product of attributes are labeled stochastically. Selecting labels stochastically may generate a function that is accidentally consistent with the training data (i.e., overfits the

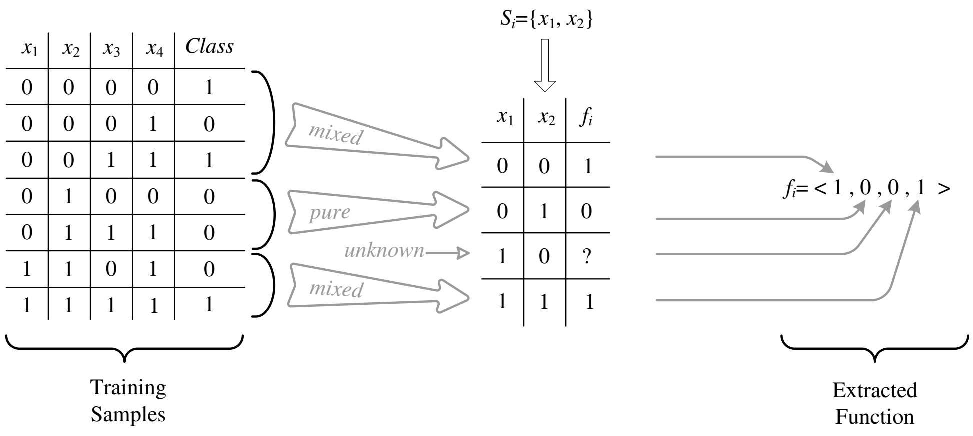


Figure 4.3: Extracting function f_i, defined over S_i, from training data in MFE2/GA: $(x_1, x_2) = (0, 0)$ and $(x_1, x_2) = (1, 1)$ are mixed tuples. Their labels are set to the opposite label of the majority label in pure tuples. There is only one pure tuples $(x_1, x_2) = (0, 1)$ with defined label '0'. So, '1' is assigned to mixed tuples. No sample in data matches with $(x_1, x_2) = (1, 0)$. The outcome of f_i for this tuple is determined stochastically with the probability of 0.6 for '0' and 0.4 for '1'.

training data). Thus, the individual that contains this function and its corresponding attribute subset is evaluated as a good individual, but results in a low predictive accuracy when evaluated by unseen data. In fact, if none of the tuples in a function are pure tuples, the attribute subset used for constructing the function is an irrelevant subset and should be discarded. Therefore, the label assignment for Case 3 is improved in the new version; when there are no pure tuples ($p_2 = n_2 = 0$), the positive label is assigned to mixed tuples, which are all the tuples that match with the training data. Then, the function will be equivalent to 'always true' function. Its outcome will be consistent only with positive samples in data. So, the individual that includes this function and its corresponding attribute subset is not evaluated as a good individual by the fitness function. Consequently, the irrelevant subset will disappear after generations. Apart from this improvement, the other parts of MFE/GA are kept unchanged in MFE2/GA. Appendix C provides an empirical comparison of the two versions of the method.

The procedure for extracting the definition of f_i from data partitions the subspace defined by S_i into four areas, as illustrated in Figure 4.4. Each f_i identifies similar patterns of interaction among attributes in S_i and *compresses* them into the negative or positive area (Case 2). The unseen area (Case 1) is covered by stochastically predicting the most frequent class. Covering unseen areas means *generalization*, and thus may involve prediction errors.

Note that GA aims to generate an individual that encapsulates interactions into several features. The representation of individuals in MFE2/GA permits constructing several features simultaneously. Moreover, the fitness function in MFE2/GA evaluates each individual composed of a set of attribute subsets and their corresponding functions, as described in the next section. Thus, features are also evaluated simulta-

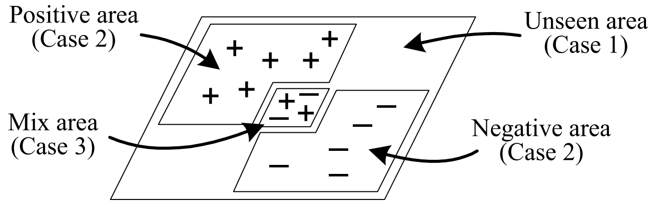


Figure 4.4: Space of samples defined by attributes in subset S_i

neously. The multi-feature construction and evaluation helps MFE2/GA to outperform other CI methods in presence of several complex interactions in data (as illustrated in Section 4.4).

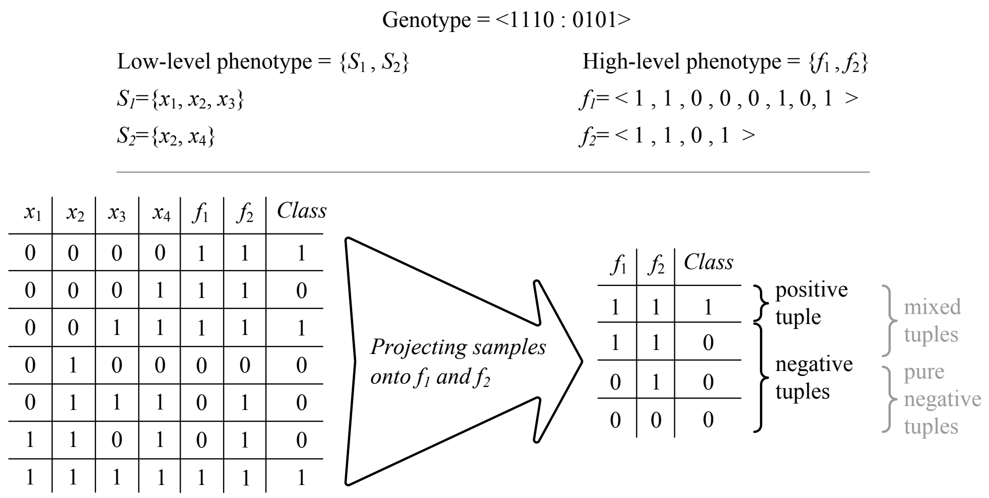
4.2.2 Fitness Function

The fitness function in MFE2/GA considers each individual in its high-level phenotype representation, evaluating the constructed features. Section 2.3 showed that genetic CI methods which construct and evaluate several features together usually apply a hypothesis-driven fitness function, applying a learning system for evaluating each individual. This form of evaluation, in addition to being limited by the quality of the learner, increases the execution time of the genetic search. Since the computation time of a data-driven fitness measure is usually less than a hypothesis-driven one, the former is applied in MFE2/GA.

The fitness function estimates both the goodness of the individual as a set of new features and the complexity of each feature in the individual. When functions f_1 to f_k corresponding to individual $\{S_1, \ldots, S_k\}$ are defined, training data are projected onto the new feature set $\{f_1, \ldots, f_k\}$ and fitness is evaluated by the following formula:

$$Fitness(Ind) = \frac{\min(|\pi^+ - \pi^-|, |\pi^- - \pi^+|)k + \|\pi^+ \cap \pi^-\|(k+1)}{N(k+1)} + \frac{\sum |S_i|}{k|S|} \,. \tag{4.1}$$

In this formula, π^+ is set of positive tuples and π^- is set of negative tuples obtained by projecting data onto $\{f_1, \ldots, f_k\}$. N is the total number of samples in training data. The single bars $|z|$ denote the number of attributes (or tuples) in subset (or relation) z; and the double bars $\|r\|$ denote the number of samples in training data that match with tuples in relation r. The formula consists of two terms. The first term approximates the cost of representing data using new features. In this term, $\min(|\pi^+ - \pi^-|, |\pi^- - \pi^+|)$ calculates the minimum between the number of positive and the number of negative tuples after removing inconsistent tuples (which are those belonging to both π^+ and π^-). This value is actually the minimum between pure positive and pure negative tuples. It is multiplied by the number of features to roughly measure the minimum



Genotype = <1110 : 0101>

Low-level phenotype = $\{S_1, S_2\}$

$S_1 = \{x_1, x_2, x_3\}$

$S_2 = \{x_2, x_4\}$

High-level phenotype = $\{f_1, f_2\}$

$f_1 = < 1, 1, 0, 0, 0, 1, 0, 1 >$

$f_2 = < 1, 1, 0, 1 >$

x_1	x_2	x_3	x_4	f_1	f_2	*Class*
0	0	0	0	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	1	1	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

f_1	f_2	*Class*
1	1	1
1	1	0
0	1	0
0	0	0

Figure 4.5: Calculating fitness value for features f_1 and f_2 defined over S_1 and S_2 using training data of Figure 4.3: $\mathit{fitness}(Ind) = \frac{2\times2+4\times3}{7\times3} + \frac{5}{2\times4} = 1.387$

code length required to classify data using new features. $\|\pi^+ \cap \pi^-\|$ denotes the number of samples in training data that match with inconsistent tuples (i.e., mixed tuples). To discriminate these samples, at least one more attribute $(k+1)$ is required. The sum $\min(|\pi^+ - \pi^-|, |\pi^- - \pi^+|) + \|\pi^+ \cap \pi^-\|$ is divided by $N(k+1)$ to be normalized. The second term evaluates the complexity of features by measuring the fraction of attributes that participate in constructing features. Including k in denominator of fractions in both terms favors individuals with larger number of subsets. The aim is to prefer several simple features, that are, features defined over smaller attribute subsets (e.g., Ind_3 in Figure 4.2) to few complex features, that are, features defined over larger subsets of attributes (e.g., Ind_2 in Figure 4.2). Figure 4.5 illustrates how fitness of the individual $\langle 1110\mathbf{:}0101 \rangle$ is evaluated using training data of Figure 4.3. GA intends to minimize the value of $\mathit{Fitness}(Ind)$. Formula 4.1 will be improved in Chapter 6 by introducing an MDL-based fitness function [Rissanen, 1983; Grunwald, 2007].

It is important to point out that features are not evaluated individually. They form a set of related parts of a theory that is used for representing interactions. Each feature by itself may not give enough information about regularities in data. For this reason, features are evaluated together, as a set of characteristics. Data are projected onto the set of features to measure how well features, as a *set*, represent the regularities in the concept. Evaluating several features together is essential when data contains several complex interactions.

Recall from Section 4.2.1 that a subset S_i where $S_i = 1$ or $S_i = S$ is ignored in the phenotype representation and FC; therefore, it will not be considered in fitness measurement. In case that none of subsets in an individual participate in FC, the worst value (a large number) is assigned to the individual as its fitness to force GA to ignore this individual.

4.2.3 Genetic Operators

When fitness of individuals is calculated, genetic operators are applied to generate a new population. Reproduction, crossover, and mutation are used for generating next population as follows.

Reproduction

Similarly to DCI, overlapping approach is used for reproduction. If the population size is p, then m best individuals are copied to the next population and $p - m$ individuals are selected for mating pool to undergo mutation and crossover operations.

However, the individual selection differs in DCI and MFE2/GA. DCI uses proportional selection. This form of selection causes two main problems [Bäck, 1996; Michalewicz, 1999; Freitas, 2002]. Firstly, it may cause premature convergence to a local optimum due to diversity reduction in the population. *Super individuals*, which are individuals with a fitness much better than population's average fitness, are much more likely to be selected than other individuals. These individuals do not allow the others to contribute in reproduction. Thus, population is dominated by these individuals and loses diversity. The domination by super individuals is desirable if it occurs in later generations when such individuals approximate the global optimal solution. Otherwise, it causes a premature convergence to a local optimum. Then, either GA terminates with a local optimal solution or requires so many generations that eventually some diversity is reintroduced in the population by mutation and better individuals are reproduced. Secondly, when GA is converging to the optimal solution, most individuals have almost the same fitness values and, therefore, have the same probability of being selected. Thus, proportional selection becomes almost equivalent to a random selection which ignores differences in fitness values.

To avoid above problems *binary tournament selection* [Goldberg, 1989] is applied in MFE2/GA. This approach randomly picks two individuals from the population, and then, selects the one with better fitness value to be copied to the mating pool. It limits the amount of selection in favor of super individuals and also exaggerates the difference between close fitness values (see [Bäck, 1996] for a comparison of selection methods). Although binary tournament selection could be seen to hinder the influence of super individuals, since overlapping approach is used, super individuals are copied to the next population; and therefore, their influence is maintained.

Crossover and Mutation

Crossover and mutation operators in MFE2/GA are distinct from those in DCI since GA's goal changes. MFE2/GA searches through the spaces $\mathcal{S}_{SS}$ and $\mathcal{S}_{SF_i}$ (as described at the beginning of Section 4.2). GA intends to generate different sets of attribute

Mutation in Attributes Level
(Mutation Type-1)

Parent1 = $\langle 10010010{:}01010100{:}000\underline{1}0111 \rangle$
Child1 = $\langle 10010010{:}01010100{:}00000111 \rangle$

Mutation in Subsets Level
(Mutation Type-2)

Parent2 = $\{S_1, \underline{S_2}, S_3, S_4, S_5\}$
Child2 = $\{S_1, S'_2, S_3, S_4, S_5\}$

Crossover in Attributes Level
(Crossover Type-1)

Parent3 = $\langle 1001|\underline{0010{:}010}|10100{:}00010111 \rangle$
Parent4 = $\langle 0010|\underline{1011{:}10010001{:}111}|01100 \rangle$
Child3 = $\langle 10011011{:}10010001{:}11110100{:}00010111 \rangle$
Child4 = $\langle 00100010{:}01001100 \rangle$

Crossover in Subsets Level
(Crossover Type-2)

Parent5 = $\{S_{11}, \underline{S_{12}}\}$ Mask1= $\langle 10 \rangle$
Parent6 = $\{S_{21}, \underline{S_{22}, S_{23}}, S_{24}, \underline{S_{25}}\}$ Mask2= $\langle 01101 \rangle$
Child5 = $\{S_{11}, S_{22}, S_{23}, S_{25}\}$
Child6 = $\{S_{12}, S_{21}, S_{24}\}$

Figure 4.6: Mutation and crossover in MFE2/GA: colons are used to separate subsets of attributes, bars mark the crossover points, and underlined are genes in the parents that will be substituted by the operators.

subsets with their associated functions and alter them by genetic operators to eventually find the proper set of subsets of interacting attributes and functions defined oven them.

Crossover and mutation operators change segments of information in parents to generate better offspring. There are two kinds of information incorporated in individuals: attributes that form a subset, and subsets that form an individual. The operators are required to generate better subsets of attributes and better sets of subsets. To achieve this aim, they are applied in two complementary levels. One level is *attributes level*, where individuals are considered as bit-strings representing presence or absence of attributes in subsets. In this level the intention is to generate better attribute subsets and corresponding functions by changing segments of bits (attributes). The other one is *subsets level*, where individuals are considered as sets of attribute subsets. Segments of information here are subsets. In this level, different sets of attribute subsets and features are produced. Hence, two types of mutation and two types of crossover are designed, as explained next and illustrated by examples in Figure 4.6.

Mutation in attributes level, Mutation Type-1, considers the individual as a bit-string of size $k.N$, where k is the number of subsets and N is the number of original attributes. It randomly negates bits in the string. Bits are selected with a probability equal to 0.01. By negating a bit, we eliminate (add) an attribute from (to) any subset in any given individual. This operation aims to introduce a new subset by a tiny change in the previously generated subset.

The Mutation Type-2, that is, mutation in subsets level, considers an individual as a sequence of subsets. This operator randomly selects a subset and replaces it with a new one, which is a randomly generated bit-string of length N. Thus, after this operation, a subset is eliminated from the given individual and a new subset is added to produce a new set of subsets. This operator introduces more variability into the population and gives more diversity than Mutation Type-1.

Similarly, two types of crossover operators are designed. Crossover Type-1, that is, *crossover in attributes level*, exchanges segments of individuals considering them as bit-strings. It applies *two-point crossover*. This operator is similar to one-point crossover (Section 2.2.1) except that two points are selected and bits between them in each parent are swapped. The two crossover points in the first parent are selected randomly. In the second parent, the crossover points are selected randomly, subject to the restriction that these points must have the same distance from the subsets boundary (in bit-string representation) as the crossover points in the first parent had [DeJong *et al.*, 1993]. Then the segments separated by points are exchanged between parents. This operator may change the length of the individual. But the limitation of $k \leq K$ is imposed, where k is the number of subsets in individual (see Section 4.2.1). If the produced offspring has more than K subsets, new crossover points are selected in the parents until the produced offspring have $k \leq K$ subsets. Depending on where the crossing points are situated this operator may generate new subsets from subsets in the parents and/or exchange subsets.

Crossover in subsets level, Crossover Type-2, aims to generate different sets of subsets by exchanging subsets in the parents. It considers individuals as sequences of subsets and performs uniform crossover. A *crossover mask* is randomly generated for each parent to determine whether a subset in a parent is given to the first or the second child. Each mask is a bit-string with a length equal to the number of subsets in the corresponding parent. Each bit equal to one (zero) in the i^{th} position of the mask means the first (second) child inherits the i^{th} subset of this parent. The probability that a subset is inherited from the first or the second parent (i.e., the probability that a bit in masks is set to one or zero) is $p_u = 0.5$. This operation may change the length of the individuals and, therefore, has the restriction of $k \leq K$, same as above. Crossover Type-2 only exchanges subsets to produce new sets of subsets. It does not generate any new attribute subset.

Mutation operators in MFE2/GA, similar to other GAs, play a secondary role [Goldberg, 1989]. They introduce small changes in the population to produce diversity. Crossover operators have the main role in constructing more promising solutions and converging GA to the optimal solution. As mentioned earlier, applying operators in two levels is required for MFE2/GA to converge faster to the optimal solution. This claim is empirically supported in Section 4.3.

It is important to note that genetic operators are performed over low-level phenotype representation of individuals, that is, a set of attribute subsets. Changing an individual implies modifying attribute subsets and/or generating a new set of subsets. Furthermore, each attribute subset in low-level phenotype representation determines a function in high-level phenotype representation that is extracted from data (as explained in Section 4.2.1). Hence, a modified attribute subset means a new constructed feature. Similarly, a modified set of attribute subsets means a new set of features. Therefore, the application of genetic operators to low-level phenotype representation of MFE2/GA's

Table 4.1: Modified parameters of PGAPack for MFE2/GA

GA Parameter	New Value
`Max Iteration`	350
`Max no Change Iter.`	100
`No. of Strings to be Replaced` $(p-m)$	90

population indirectly produces evolution of features and the set of features in high-level phenotype representation. Recall that the fitness evaluation is performed over high-level phenotypes.

4.2.4 MFE2/GA's Algorithm

To implement GA, PGAPack library [Levine, 1996] is used with default parameters, except those indicated in Table 4.1 (see Appendix B for default parameters of PGAPack). Two stopping rules are applied as termination conditions of GA. The first rule is to stop when reaching the maximum number of iterations limit, determined by "`Max Iteration`". The second one is to stop when no change in the best solution is found in a given number of iterations, specified by "`Max no Change Iter`". When either condition is satisfied, GA is terminated. The mating pool size (see Section 4.2.3) is determined by "`No of Strings to be Replaced`" (PGAPack default population size, p, is 100 individuals). Note that the mutation and crossover operators in PGAPack are replaced with those described in Section 4.2.3. For reproducibility purposes the PGAPack random seed is initialized to one.

DCI and MFE2/GA are designed based on the framework of Figure 2.7. Hence, they have similar algorithms. Figure 4.7 summarizes MFE2/GA's algorithm (see Figure 3.3 for DCI's algorithm). MFE2/GA receives training data and original attribute set and performs GA to find the set of functions that best represent interactions among attributes. To reduce overfitting, 90% of training data are used for constructing functions and all training data, for fitness evaluation using Formula 4.1. When GA is finished the best individual represents the set of functions constructed to highlight interactions. The constructed functions as new features are added to the original attribute set and data are redescribed using the new set of attributes. The new data are then given to a learner for learning.

4.3 Evaluating Crossover Operators

Section 4.2.3 described the genetic operators used in MFE2/GA, namely two types of mutation and two types of crossover. These operators aim to generate the set of attribute subsets with their corresponding functions that best represent interactions among attributes. Crossover plays an important role in convergence of GA to optimal

```
MFE2/GA
Receives: training and test data and the set of $N$ original attributes
Returns: a new representation of training and test data

  1. call $S$-Search.
  2. add the output of $S$-Search as new features to the set of original attributes,
     and redescribe training and test data.
  3. return new attribute set and data.

$S$-Search
Receives: training data and a set of $N$ original attributes
Returns: a set of new features

  1. Generate a population of bit-strings $Ind_i$ of length $k_i.N$, each representing
     a set of attribute subsets $\{S_{i1}, \ldots, S_{ik_i}\}$, where $k_i \leq K$.
  2. For each individual $Ind_i$, call $F_{S_i}$-Search, assign the set of functions gener-
     ated by $F_{S_i}$-Search to the individual, and calculate the fitness according to
     Formula 4.1.
  3. Unless predefined stopping conditions are achieved, apply genetic operators,
     generate a new population, and go to step 2.
  4. Return the set of functions assigned to the best individual.

$F_{S_i}$-Search
Receives: 90% of training data and a set of $k$ attribute subsets $\{S_1, \ldots, S_k\}$
Returns: $\{f_1, \ldots, f_k\}$

  1. For each $S_i$ in $\{S_1, \ldots, S_k\}$ define $f_i$ as follows:
     (a) consider Cartesian product of attributes in $S_i$ and classify each tuple
         in the product as pure, unknown, or mixed tuple according to the
         samples' labels in training data.
     (b) for each tuple $t$ in the product do:
           i. if $t$ is an unknown tuple, assign a class label to $f_i(t)$ stochastically,
              according to the class distribution in the training data
          ii. if $t$ is a pure tuple, assign the label of matching samples to $f_i(t)$
         iii. if $t$ is a mixed tuple, there are $p$ pure positive tuples and $n$ pure
              negative tuples, and $p > n$, then assign negative class to $f_i(t)$;
              otherwise, assign positive class
  2. return $\{f_1, \ldots, f_k\}$
```

Figure 4.7: MFE2/GA, S-Search, and F_{S_i}-Search algorithms

solution, while mutation plays a secondary role [Goldberg, 1989]. This section empirically analyzes crossover operators in attributes level and in subsets level. The effects of these two operators on performance of MFE2/GA and their importance for converging GA to optimal solution are evaluated.

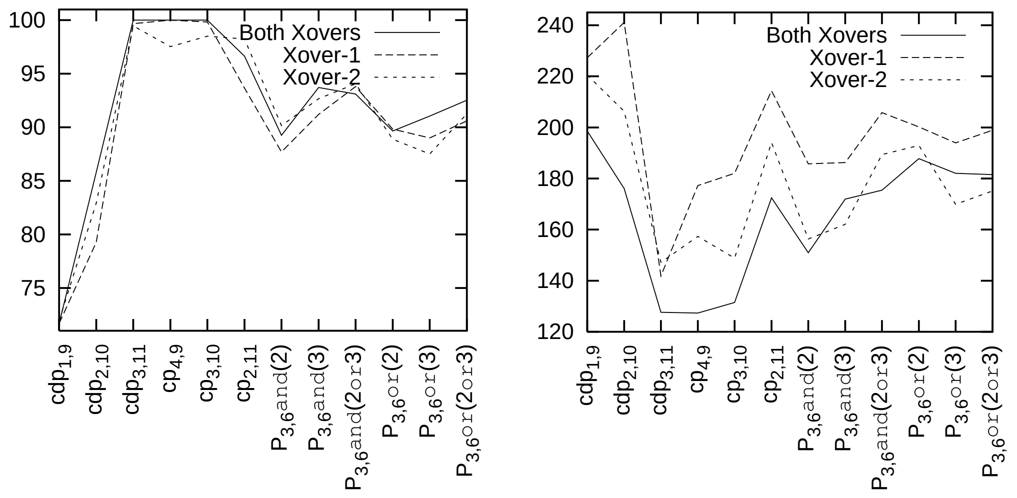
To empirically compare operators, MFE2/GA is run in three different ways: with crossover in attributes level (Type-1) only, with crossover in subsets level (Type-2) only, and with both crossover operators, named *Xover-1*, *Xover-2*, and *Both Xovers* respectively. When both crossovers are permitted, each time that individuals are selected for crossover operation, the type of crossover is randomly selected for application by flipping a coin. Both mutation operators are applied together in all experiments. For each individual that undergoes mutation, the type of mutation is determined randomly.

Synthetic concepts are used for experiments. Concepts consist of several complex interactions. Appendix A gives definitions of these concepts. For each concept, each experiment is run 20 times independently over 20 sets of shuffled data. For each experiment 5% of all data are used for training and the rest (95%) are kept unseen as test data for final evaluation. When GA is finished the new features are added to the original set of attributes and data are updated. C4.5 with default parameters is applied as a standard learner for final evaluation. The new representation of training data are used for learning by C4.5 and the prediction accuracy is measured after tree pruning using modified 95% unseen data.

Figure 4.8 shows the average results for each concept obtained by the three ways of running MFE2/GA described above (Xover-1, Xover-2, and Both Xovers). The horizontal axes represent synthetic concepts. In Figures 4.8(a) the vertical axis shows the average predictive accuracy of C4.5 on 95% test data using the features constructed by MFE2/GA. This axis in Figure 4.8(b) shows the average number of generations needed by GA in order to achieve the result for each concept.

The figure illustrates that Xover-1 (MFE2/GA with crossover in attributes level), in most cases, results in lower accuracy and slower performance than Xover-2 (MFE2/GA with crossover in subsets level). Recall from Section 4.2.3 that the crossover in attributes level is a two-point crossover that may generate new attribute subsets as well as new sets of subsets. Two point crossover has a lower recombination potential and produces less diversity in the population than uniform crossover [Spears and DeJong, 1991]. Due to applying two-point approach, this operator does not explore the search space as well as crossover in subsets level, which is a uniform crossover. For this reason, Xover-1's performance is worse than Xover-2's.

However, crossover in subsets level does not produce any new subset of attributes, but exchanges subsets. Thus, it is not enough for exploiting the search space to generate possible solutions. Generation of new subsets is achieved only by mutation. Since, mutation produces minor changes in the population, Xover-2 takes longer to converge



(a) Accuracy (b) Number of generations

Figure 4.8: Comparing the use of crossover operators

to optimal solution comparing to Both Xovers.

Applying both crossovers provides a balance of exploitation and exploration of the search space and guides MFE2/GA toward better solutions. Using crossover in attributes level or in subsets level alone causes GA to fail sometimes. Although the differences are not significant in Figure 4.8(a), Both Xovers gives a higher accuracy for most concepts. Also, comparing the results in Figure 4.8(b) illustrates that using two operators together helps GA to converge faster to the optimal solution. As expected, the results support the claim that the two crossover operators defined in Section 4.2.3 complement each other to accelerate the convergence of MFE2/GA to the optimal solution. After having chosen the best way to run MFE2/GA, the method will now be compared with other systems.

4.4 Empirical Comparison

This section describes experiments conducted to empirically analyze MFE2/GA. Section 4.4.1 explains primary experiments performed to compare MFE2/GA with DCI to evaluate the multi-feature construction property of MFE2/GA. Section 4.4.2 explains experiments conducted to compare MFE2/GA with other CI methods. Similarly to Section 3.2, experiments use synthetic concepts, which allow analyzing results deeply before moving on to deal with real-world problems.

4.4.1 MFE2/GA and DCI

Chapter 3 showed that the main characteristics of DCI, namely genetic-based search and non-algebraic representation of features, make this method work better than others if

Table 4.2: Comparing average predictive accuracies of DCI and MFE2/GA

Concept	Relev. atts	Irrel. atts	Maj %	C4.5 Org. Atts	DCI + C4.5	MFE2/GA+ C4.5
mx_6	6	6	50	94.9(3.1)	97.8(2.1)	98.8(1.8)
$cdp_{3,11}$	6	6	62.5	81.3(8.2)	97.6(1.6)	**100(0.0)**
$cdp_{2,10}$	9	3	62.5	66.0(9.8)	67.7(2.0)	**85.8(8.6)**
$cdp_{1,9}$	12	0	62.5	56.6(4.9)	56.4(2.5)	**71.7(3.8)**
$cp_{4,9}$	6	6	75.0	73.1(2.7)	97.2(1.8)	**100(0.0)**
$cp_{3,10}$	8	4	75.0	73.1(2.6)	81.6(1.6)	**100(0.0)**
$cp_{2,11}$	10	2	75.0	72.6(4.0)	68.0(1.2)	**96.7(6.1)**
$P_{3,6} \wedge (2)$	10	2	88.3	87.9(1.6)	81.0(1.5)	**93.1(5.7)**
$P_{3,6} \wedge (3or2)$	10	2	72.7	68.7(2.0)	65.4(2.6)	**89.3(5.7)**
$P_{3,6} \wedge (3)$	10	2	84.4	84.4(0.1)	75.9(2.4)	**93.8(5.2)**
$P_{3,6} \vee (2)$	10	2	61.7	59.4(5.0)	57.9(2.2)	**89.7(5.5)**
$P_{3,6} \vee (3or2)$	10	2	77.3	75.5(2.2)	69.5(2.2)	**92.5(5.7)**
$P_{3,6} \vee (3)$	10	2	65.6	60.9(2.9)	59.3(1.7)	**91.1(7.6)**
Average	8.6	3.4	70.5	73.8(4.0)	76.8(1.8)	**93.0(4.0)**

the target concept is composed of one complex interaction. However, DCI is incapable of constructing more than one feature. When several complex interactions exist in concept a CI method needs to construct and evaluate several features simultaneously to highlight interactions to the learner. MFE2/GA is designed to fulfil this requirement. This section compares MFE2/GA with DCI by conducting experiments over synthetic concepts that are known to be difficult for DCI (see Section 3.2.2). Concepts are composed of several interactions and the total number of interacting attributes for each concept is high (see Appendix A for concept definitions).

Similar experiments to those described in Section 3.2.2 and reported in [Perez, 1997] are performed. For each concept, MFE2/GA is run 20 times independently over 20 sets of shuffled data. For each trial, 5% of all data are used for training and the rest (95%) are kept unseen for final evaluation. When MFE2/GA is finished its performance is evaluated by the predictive accuracy of C4.5 [Quinlan, 1993] on modified data after adding constructed features, using 95% unseen data as test data. Similar experiments are performed with DCI. The results are also compared with the predictive accuracy of C4.5 using the original attribute set as a baseline performance.

Table 4.2 gives a summary of concepts and the average results obtained by each method. Columns two and three show the number of relevant and irrelevant attributes for each concept, and column four shows the majority class percentage. The average predictive accuracies of C4.5, DCI, and MFE2/GA are shown in fifth, sixth, and seventh columns respectively. Numbers between parentheses indicate standard deviations. Bold means that with a significant level of 0.02 MFE2/GA's accuracy is better than DCI's, using t-distribution test.

Comparing the results of the fifth and sixth columns in Table 4.2 illustrate that, as expected, DCI cannot ease learning task when the number of interacting attributes increases since this method is incapable of constructing more than one feature. MFE2/GA intends to find subsets of interacting attributes and their corresponding functions, each encapsulating one interaction. It generates individuals representing sets of attribute subsets and their corresponding functions as potential solutions. Constructing and evaluating several features at a time allows MFE2/GA to successfully break down the high-order interaction into smaller interactions represented by several features. Thus, it significantly outperforms DCI for almost all concepts.

4.4.2 MFE2/GA and Greedy CI Methods

MFE2/GA is also compared with some greedy methods used in Section 3.2.2. These methods are C4.5 and C4.5-Rules [Quinlan, 1993], which are similarity-based learners, Fringe, Grove and Greedy3 [Pagallo and Haussler, 1990], and LFC [Ragavan and Rendell, 1993], which are CI methods that use algebraic representation of features, and MRP [Pérez and Rendell, 1995], which is a CI method with non-algebraic form of feature representation. Among greedy CI methods, only Fringe constructs several features at once. Other methods construct and evaluate features one at a time.

Experiments are similar to those in the previous section and in [Perez, 1997]. For each experiment, 5% of shuffled data are used for training and the rest are kept unseen for test data. Table 4.3 gives a summary of concepts used for experiments in columns one to four and the average accuracies over 20 runs obtained by different methods in columns five to seven. Concepts are divided into two groups. The top part of the table groups concepts composed of several complex interactions and the bottom part shows those composed of one complex interaction. In each group concepts are ordered by the number of relevant attributes. The best results among C4.5, C4.5-Rules, Fringe, Grove, Greedy3, and LFC are shown in the fifth column of the table, as reported in [Perez, 1997]. The letters $c4$, cr, fr, gr, $g3$, and lf indicate that the best competitor is C4.5, C4.5-Rules, Fringe, Grove, Greedy3, or LFC respectively. The average predictive accuracy of MRP, the only greedy method in these experiments with non-algebraic feature representation, is shown in column six. Last column represents the predictive accuracy of C4.5 after adding features constructed by MFE2/GA to the original attribute set and updating data. This result is marked by † when it is equal or better than the one in column five (Prior best result) and by $\triangleleft$ when it is equal or better than the one in column six (MRP). The overall average of the results of each group of concepts is also given. Numbers between parentheses indicate standard deviations. Bold means that, with a significant level of 0.02 with t-distribution test, the accuracy is higher between MRP and MFE2/GA using.

Table 4.3 shows that MRP and MFE2/GA achieve better results than other greedy methods. This is due to their form of representing features. When a complex interaction exists among attributes, a more complex algebraic feature is needed to abstract the interaction (see Section 2.1.3). A non-algebraic representation captures the structure of

Table 4.3: Comparing average predictive accuracies of MFE2/GA with greedy methods

Concept	Relev. atts	Irrel. atts	Maj %	Prior best result	MRP	MFE2/GA + C4.5
$\mathrm{cp}_{5,8}$	4	8	75.0	100 *fr*	100(0.0)	100(0)†◁
mx_{6}	6	6	50.0	100 *fr*	96.1(7.3)	98.8(1.8) ◁
$\mathrm{cdp}_{3,11}$	6	6	62.5	98.4 *fr*	97.3(3.9)	100(0)†◁
$\mathrm{cp}_{4,9}$	6	6	75.0	86.1 *fr*	99.0(1.6)	**100(0)**†◁
$\mathrm{cp}_{3,10}$	8	4	75.0	73.4 *cr*	88.9(1.2)	**100(0)**†◁
$\mathrm{cdp}_{2,10}$	9	3	62.5	78.1 *fr*	**92.0(6.5)**	85.8(8.6)†
$\mathrm{cp}_{2,11}$	10	2	75.0	73.9 *c4*	91.7(5.7)	**96.7(6.1)**†◁
$P_{3,6} \wedge (2)$	10	2	88.3	88.1 *c4*	90.4(4.5)	93.1(5.7)†◁
$P_{3,6} \wedge (3or2)$	10	2	72.7	70.4 *fr*	79.0(2.3)	**89.3(5.7)**†◁
$P_{3,6} \wedge (3)$	10	2	84.4	83.4 *c4*	87.6(5.4)	**93.7(5.2)**†◁
$P_{3,6} \vee (2)$	10	2	61.7	70.5 *fr*	**97.1(2.9)**	89.7(5.6)†
$P_{3,6} \vee (3or2)$	10	2	77.3	75.1 *fr*	80.4(6.0)	**92.5(5.7)**†◁
$P_{3,6} \vee (3)$	10	2	65.6	68.0 *fr*	95.9(4.4)	91.1(7.6)†
$\mathrm{cdp}_{1,9}$	12	0	62.5	65.5 *cr*	**81.3(3.0)**	71.7(3.8)†
Average	8	3	70.5	80.8	91.2	93.0†◁
$P_{1,4}$	4	8	50.0	99.6 *fr*	100(0.0)	100(0)†◁
$\mathrm{gw}_{5,8}$	4	8	68.8	100 *g3*	100(0)	100(0)†◁
$\mathrm{sw}_{5,8}$	4	8	62.5	100 *fr*	100(0)	100(0)†◁
$\mathrm{mx}_6\mathrm{c}_{6,7}$	4	8	50.0	98.7 *fr*	100(0)	100(0)†◁
$\mathrm{mj}_{4,8}$	5	7	50.0	99.0 *gr*	99.8(0.7)	100(0)†◁
$P_{1,6}$	6	6	50.0	51.1 *fr*	97.0(1.6)	98.0(1.5)†◁
$\mathrm{gw}_{4,9}$	6	6	65.6	93.3 *gr*	95.1(8.2)	98.1(2)†◁
$\mathrm{sw}_{4,9}$	6	6	68.8	73.1 *fr*	98.5(1.6)	98.6(1.4)†◁
$\mathrm{mx}_6\mathrm{c}_{5,8}$	6	6	50.0	76.4 *fr*	96.4(3.7)	97.6(1.7)†◁
$\mathrm{mj}_{3,9}$	7	5	50.0	86.6 *gr*	85.2(5.1)	**90.1(2.4)**†◁
$\mathrm{nm}_{4,5,7}$	7	5	56.2	74.1 *fr*	86.7(7.6)	89.8(2.5)†◁
$\mathrm{rk}_{5,7}$	7	5	83.6	89.9 *gr*	91.4(4.7)	93.7(3.7)†◁
$\mathrm{rk}_{6,7}$	7	5	94.5	95.6 *fr*	96.8(2.3)	95.9(2.7)†
$P_{1,8}$	8	4	50.0	49.4 *cr*	75.1(1.4)	74.7(7.7)†
$\mathrm{gw}_{3,10}$	8	4	63.7	84.9 *gr*	**80.7(3.9)**	76.5(5.1)
$\mathrm{sw}_{3,10}$	8	4	72.7	71.8 *lf*	**87.8(1.3)**	75.9(5.4)†
$\mathrm{mx}_6\mathrm{c}_{4,9}$	8	4	50.0	73.5 *fr*	**80.6(5.2)**	75.5(5.2)†
Average	6	6	61.0	83.4	92.4	92.0†

† no worse than Prior best result
◁ no worse than MRP

the interaction and abstracts it more easily. For this reason MFE2/GA and MRP have less difficulty to construct features than other methods that apply algebraic form.

In the top part of the table, MFE2/GA outperforms MRP for most of the concepts. These concepts are composed of several interactions. It was illustrated in Section 3.2.2 that in spite of using a genetic-based search and finding interacting attributes, DCI could not construct an appropriate feature that encapsulates all interactions. Thus, its accuracy was lower than MRP. Multi-feature construction in MFE2/GA overcomes the problem and beats MRP. MFE2/GA breaks down the difficult task of constructing one function defined over interacting attributes into two or more easier tasks of constructing simpler functions over smaller subsets of interacting attributes using a GA. This method successfully groups attributes involved in each interaction into subsets and constructs a function over each subset to capture and abstract each interaction. It simultaneously constructs and evaluates several features at a time, which is necessary in presence of several complex interactions. Thus, it achieves better results.

MRP considers attributes individually when searching for subsets of interacting attributes, and constructs features one by one. Backward elimination approach in MRP eliminates irrelevant attributes one at a time to find interacting attributes. As explained in Section 2.1.1, once it eliminates all irrelevant attributes, due to one-by-one elimination, it may not go forward to get a smaller subset of those attributes that participate only in one interaction. So, it remains with the subset of all interacting attributes and constructs a function over this larger subset. Constructing a function that outlines all interactions is complex. Thus, MRP fails to achieve high accuracy if one-by-one attribute elimination does not allow this method to find a smaller subset of interacting attributes. Even if MRP successfully goes forward and finds the subset of attributes that participate only in one interaction, it still may fail to achieve high results due to its greedy FC. When concepts are composed of several complex interactions and few training data are available one-by-one construction of features produces problems. MRP constructs a function over the subset, splits data using this function, and continues looking for attributes that participate in other interactions. Since the number of interacting attributes is high and few training data are available, FC is difficult, and is likely to fail. If the feature constructed in the first step is not appropriate all the subsequent features will be irrelevant to. Therefore, even if this method correctly finds the proper subset of attributes, it still may construct an incorrect feature and converge to a local optimum due to its greedy FC. For this reason, while the number of interacting attributes grows, concepts become harder to be learned by MRP. Thus, its performance degrades most of the time comparing to MFE2/GA.

Note that in the top part of the table, Fringe outperforms all CI results summarized by the column "Prior best result" for 9 out of 14 concepts. This could be due to its strategy for constructing features. As mentioned before, among these methods, Fringe is the only one that constructs and evaluates several features together, which proves to be essential for this group of concepts. For the remaining 5 concepts, C4.5 or C4.5-Rules outperform other methods in the column "Prior best result", although often its accuracy is lower than the majority class percentage. This means classifying data by the label of majority class gives better result; that is, the learner did not learn these concepts.

The concepts $cdp_{i,j}$ illustrates well the different behaviors and advantages of MRP and MFE2/GA. Each of the concepts $cdp_{1,9}$, $cdp_{2,10}$, and $cdp_{3,11}$ involves three parity interactions combined by simple relations (conjunction and disjunction). The three concepts differ in the degree of parity involved (4, 3, and 2, respectively); but perhaps more importantly, they also differ in the ratio of relevant attributes (12/12, 9/12, and 6/12, respectively). This is the reason why, obviously, all results in Table 4.3 indicate that $cdp_{1,9}$ is the most difficult concept to learn: it has no irrelevant attributes that when projected away allow the complex structure of the intermediate concepts (parities) to become apparent, when learning from only 5% of data.

This affects MFE2/GA in a higher degree than it affects MRP, probably due to differences between both systems' biases. In particular, considering $cdp_{1,9}$, that is defined as:

$$Parity(a_1, \ldots, a_4) \wedge [Parity(a_5, \ldots, a_8) \vee Parity(a_9, \ldots, a_{12})] \ ,$$

MRP's focus on learning one single best relation probably guides learning toward $Parity(a_1, \ldots, a_4)$. So MRP easily finds its way toward that parity feature, which captures most of the concept. Had it used only that single feature to classify unseen data, it would have obtained even higher accuracy than it does (up to 87%). Perhaps, due to overfitting, MRP's heuristic function does not allow the system to reach such theoretically best possible performance on this concept. However, MRP's bias gets it closer to the goal than MFE2/GA.

MFE2/GA's bias is, in some sense, opposite to MRP's. It focuses on learning multiple features at once to evaluate them in combination. This higher flexibility in searching a large and complex feature space makes the system more dependent on data quality (since features are extracted from training data). MFE2/GA exploits better than MRP the redundancy in data for $cdp_{3,11}$. Since this concept is defined over 6 attributes of a total of 12 attributes, the 5% training data is likely to contain repeated parts of the concept structure, that become more apparent when projecting away the irrelevant attributes. However, this does not make the concept easy to learn by non-CI methods, due to the complexity of the interactions involved. MRP tries to learn this concept as a single relation, whereas MFE2/GA beats it by seeing the multiple features involved at once. On the other hand, for $cdp_{2,10}$, 9 attributes of 12 attributes are relevant, so there is little or no redundancy in the 5% training data. Therefore, MFE2/GA's more flexible search gets trapped in a local minimum, overfitting data; whereas MRP is favored by its strong bias for *one best* relation, which in this case, does indeed exists and it is easy to find.

For the second group of concepts, MFE2/GA gives higher accuracy than MRP when the number of interacting attributes is small. But when more attributes are involved in interaction MFE2/GA's performance degrades. Nevertheless, differences are not as significant as in the top part of the table. These concepts are composed of one high-order complex interaction. For these concepts, MFE2/GA cannot take advantage of its multi-feature construction property to break down the interaction into smaller ones. So, it becomes similar to DCI; that is, when number of interacting attributes grows its performance is scaled down. MRP tries to eliminate one attribute at a time to find interacting attributes. Because of the structure of these concepts one-by-one elimination of attributes is less problematic for MRP here than in the first group of concepts. Once MRP finds the interacting attributes, it constructs a function defined over these attributes and induced from data. The greedy FC of MRP is not a problem here since there is only one function to be constructed. Note that these concepts were designed to evaluate MRP and match well with the bias of this method [Perez, 1997]. MFE2/GA's flexibility in finding interacting attributes causes this method to overfit data and fail to find the correct subset of attributes when more attributes interact.

It is important to mention that in all experiments MFE2/GA generates close approximations to the set of subsets of interacting attributes; in more than 81% of experiments this method successfully finds the exact subsets of interacting attributes. However, the 19% failure indicates that GA is not always guided properly toward optimal solution. Chapter 6 proposes a new fitness function based on MDL Principle, which improves the performance of GA in MFE2/GA and reduces the failure amount to 11%.

4.5 Conclusion

This chapter presented MFE2/GA, a CI method that maintains all the advantages of DCI, while allows constructing and evaluating several features together. MFE2/GA uses genetic-based search to find the set of subsets of interacting attributes and the set of functions that abstract interactions. Each individual is a bit-string that represents a set of attribute subsets as a low-level phenotype, which in turn represents a set of constructed functions as a high-level phenotype (Section 4.2.1). This kind of representation allows the application of GA, which is simpler than GP. Moreover, it prevents the corresponding problems related to GP with algebraic form of representing features (Section 2.3). The individual's representation in MFE2/GA also provides the facility to capture and encapsulate interactions into several features.

Genetic operators in MFE2/GA aim to generate better attribute subsets and set of attribute subsets in low-level phenotypes. Since each low-level phenotype represents a high-level phenotype, that is a set of functions defined over attribute subsets, genetic operators indirectly evolve the set of constructed functions. Two types of mutation and two types of crossover operators are designed (Section 4.2.3). These operators complement each other and provide a balance of exploitation and exploration (Section 4.3).

Another important aspect of MFE2/GA is the data-driven fitness function. Genetic methods that evaluate several features as an individual usually apply a hypothesis-driven fitness evaluation (Section 2.3). A new data-driven fitness function is proposed for MFE2/GA, which is faster than a hypothesis-driven fitness function. This fitness function allows evaluating features together as a set of characteristics that highlight regularities.

The genetic approach of MFE2/GA along with its individual's representation, fitness evaluation, and genetic operators provide the ability of constructing and evaluating several features at once which is necessary for a CI method (Section 4.1). This method fulfills all requirements for a CI method mentioned in Chapter 2.

Experiments illustrates that multi-feature construction in MFE2/GA facilitates the task of FC when concept is composed of several interactions and few training data are available (Section 6.4.1). Thus, MFE2/GA overcomes the deficiencies of DCI (Section 3.2.2). Empirical comparison of MFE2/GA and greedy methods clearly outlines the advantages of this method (Section 6.4.2). The non-algebraic representation eases the construction of complex features; therefore, MFE2/GA outperforms methods that apply algebraic feature representation. Also, experiments shows that in presence of several complex interactions in concepts a greedy strategy for one-by-one selecting attributes and constructing features fails. MFE2/GA successfully constructs and evaluates several features together, which proves to be essential for concepts with several interactions.

However, experiments show that GA sometimes fails to conduct MFE2/GA toward the optimal solution. Fitness function has the main role in guiding GA to converge to the optimal solution. Chapter 6 reviews different kinds of fitness functions and introduces a new fitness evaluation based on MDL Principle. The MDL-based fitness function is integrated into a new system called MFE3/GA [Shafti and Pérez, 2007a; 2007b].

Before that, Chapter 5 compares MFE2/GA with the relevant non-algebraic method, HINT [Zupan *et al.*, 2001]. It analyzes design aspects that highlight similarities and differences between two systems. Since both systems heavily depend on data to construct features, the chapter focuses the empirical evaluation of both systems on their sensitivity to training data size when learning hard concepts with different types of complex interactions, and relates the empirical results to prior analysis of system designs.

Chapter 5

Data-based FC by MFE2/GA and HINT: Sensitivity to Training Data Size

Chapter 2 described requirements for success of a CI method when confronted with complex interactions. It explained that a global search strategy based on genetic algorithms facilitates both identifying interacting attributes and constructing features. Chapter 2 also described that a CI method needs to simultaneously construct and evaluate several features together; such requirement was not considered in DCI, as illustrated in Section 3.2.2. In addition, a non-algebraic (operator-free) form of representing features is required to facilitate constructing features in a CI method when the only available information about the problem is training data. Based on these requirements, Chapter 4 proposed MFE2/GA and illustrated that this method outperforms greedy CI methods which apply algebraic feature representation.

MFE2/GA was also compared, in Chapter 4, with MRP that uses non-algebraic representation. It was shown that MRP's performance degrades when several complex interactions exist among attributes, as a consequence of applying backward elimination search to find interacting attributes one by one. Due to complex interactions, each attribute by itself does not give enough information and, therefore, may be considered as an irrelevant attribute by MRP's search strategy.

Alternatively, some methods avoid one-by-one attribute selection or elimination by evaluating subsets of attributes as candidates and selecting heuristically the best subset. Since the search space grows exponentially with the number of attributes, some limitations are usually imposed in forming subsets to reduce complexity. Among these methods the most relevant to this thesis is HINT [Zupan *et al.*, 2001], which also uses non-algebraic form of representing constructed features. This method considers all subsets of a predefined bound size, and successively selects attribute subsets and constructs

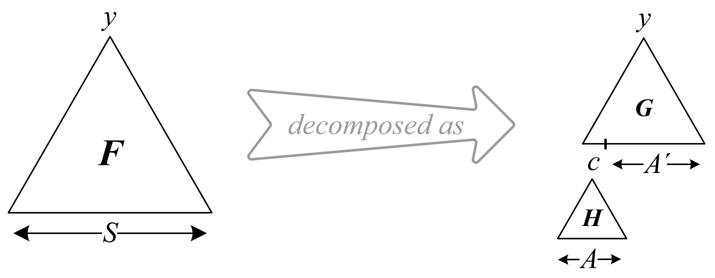


Figure 5.1: Function decomposition by HINT

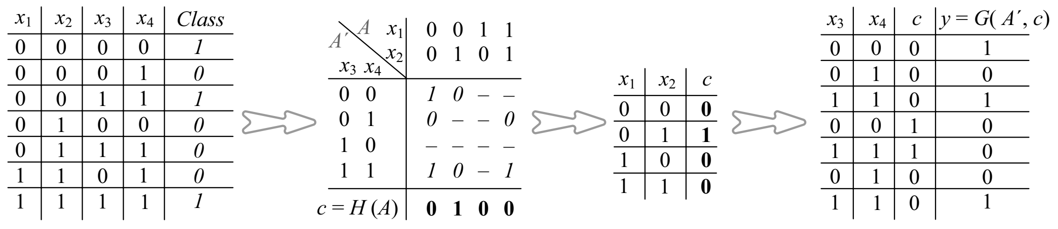
features, by labeling data samples represented using attributes in the subset, as will be detailed in this chapter.

This chapter theoretically and empirically compares HINT and MFE2/GA. Since both methods extract features directly from data, they depend heavily on training data to construct features. Data-based non-algebraic representation may require more training data samples. The focus of this chapter is on the sensitivity to training data size of these two methods. HINT is reviewed in Section 5.1. The design properties of the two systems are compared in Section 5.2, and empirically evaluated in Section 5.3, using problems where attribute interaction is a major difficulty. Experiments indicate that both methods have some important functionalities that could be combined to obtain better results. Conclusions of this chapter are summarized in Section 5.4.

5.1 HINT: Multi-value Feature Construction

HINT is a greedy CI method that generates a classifier represented by a hierarchy of intermediate concepts, which are constructed features. HINT assumes that the class labels and attributes are nominal. It applies a *function decomposition* step recursively to induce features directly from training data and represent them by a set of labeled samples, as shown bellow.

Consider the target concept is $y = F(S)$, where S is the set of given attributes and y is the class label. The function decomposition aims to partition S into subset A and its complementary A' (i.e., $A \cap A' = \emptyset$ and $A \cup A' = S$) and to define functions H and G, such that $c = H(A)$ and $y = G(A', c)$ (Figure 5.1). Functions H and G are represented by sets of attribute values with assigned labels as explained next. Then, S is replaced by $S' = A' \cup \{c\}$ and the decomposition step is recursively applied to $y = G(S')$ and to $c = H(A)$. The attribute partition is determined by evaluating all possible partitions of S into A and A'. To reduce the complexity, the size of A is limited to the bound size $b \leq 3$. The partition that minimizes some complexity measure defined over G and H is selected (see [Zupan and Bohanec, 1998] for details). If there is no such partition, function decomposition is terminated.



x_1	x_2	x_3	x_4	*Class*
0	0	0	0	*1*
0	0	0	1	*0*
0	0	1	1	*1*
0	1	0	0	*0*
0	1	1	1	*0*
1	1	0	1	*0*
1	1	1	1	*1*

$A' \backslash A$ $x_3\ x_4$ / x_1 x_2	0 0	0 1	1 0	1 1
0 0	*1*	*0*	–	–
0 1	*0*	–	–	*0*
1 0	–	–	–	–
1 1	*1*	*0*	–	*1*
$c = H(A)$	**0**	**1**	**0**	**0**

x_1	x_2	c
0	0	**0**
0	1	**1**
1	0	**0**
1	1	**0**

x_3	x_4	c	$y = G(A', c)$
0	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	1	1	0
0	1	0	0
1	1	0	1

Figure 5.2: Inducing features from data by HINT: Considering the concept $y = F(S)$ represented by the training data of Figure 4.3, HINT constructs $c = H(A)$ and $y = G(A', c)$, where $A = \{x_1, x_2\}$ and $A' = \{x_3, x_4\}$.

The function $c = H(A)$ is derived from training samples as follows. A *partition matrix* is defined, where each column and row correspond to a distinct combination of values of attributes in A and A' respectively. Each element m_{kj} in the matrix shows the class label of the samples whose attributes in A and A' take the values identifying column j and row k respectively. If there is no such sample, a "`don't know`" symbol is assigned to m_{kj}, indicating that this cell of matrix is *empty*. Two non-empty columns j and l are called *compatible* if there is no contradiction between them; that is, $m_{kj} = m_{kl}$ or $m_{kj} =$"`don't know`" or $m_{kl} =$"`don't know`" for each k. The function $c = H(A)$ is then defined by labeling (*coloring*) columns of the matrix. Each group of mutually compatible columns is labeled equally. This is performed by generating column incompatibility graph and applying Color Influence Method [Wan and Perkowski, 1992] to assign a color or label to each vertex in a way that two incompatible vertices have different colors and the optimal number of colors is used (for details see [Zupan *et al.*, 1999]). When all non-empty columns are labeled, *the default rule* is used for labeling empty columns that assigns the most frequent label (color) of non-empty columns. The function c is, then, represented by attribute values in each column and the assigned label. The function $y = G(A', c)$ is the set of training samples redescribed using the new attribute set $S' = A' \cup \{c\}$.

To better illustrate the function decomposition step of HINT, recall the training data of Figure 4.3, representing concept $y = F(S)$, where $S = \{x_1, x_2, x_3, x_4\}$. Figure 5.2 shows how HINT generates a partition matrix, constructs a function and redescribes training data. The function decomposition partitions S into sets $A = \{x_1, x_2\}$ and $A' = \{x_3, x_4\}$, and generates a partition matrix where each column corresponds to values of x_1 and x_2, and each row corresponds to values of x_3 and x_4. Each element of the partition matrix shows the label of each sample in training data (represented in italics in the figure). The "`don't know`" elements are marked by "–". Then, HINT identifies compatible columns and assigns colors (labels, in boldface) to them. The first and forth columns are compatible and the same color, represented by '0', is assigned to

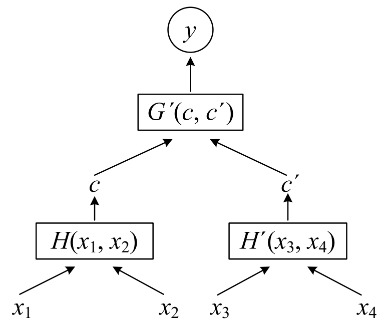


Figure 5.3: The classifier generated by HINT for the concept of Figure 5.2

them. The second column is colored as '1'. The third column is an empty column. The default rule assigns '0' (i.e., the most frequent color) to this column. The bottom row in the matrix shows the colors assigned to each column (represented in Boldface), which determines the outcome of the function $c = H(A)$ for different values of attributes in $A = \{x_1, x_2\}$. Then, the set of attributes S is replaced by the new one $S' = \{x_3, x_4, c\}$, and data samples are redescribed to generate the new sample set $y = G(x_3, x_4, c)$ using the constructed function $c = H(A)$. The decomposition step is then applied to the new attribute set and training data recursively. At the end, HINT generates a classifier by a hierarchy of constructed functions as shown in Figure 5.3.

Note that in this example, only two labels were needed for coloring columns, though, HINT is capable of constructing multi-value functions by assigning more than two colors to columns (as described in Section 5.2). Because of its ability to generate multi-value features by *multi coloring*, HINT achieves high accuracy on many artificial and real-world domains of UC Irvine Database [Blake and Merz, 1998]. However, to achieve these results, HINT requires to have enough data samples available.

It is important to mention that the decomposition step *generalizes* labels for "`don't know`" elements. If $m_{kj} =$"`don't know`" and column j is non-empty and compatible with column l, then the same label as m_{kl} is given to m_{kj}. If $m_{kj} =$"`don't know`" for all k in column j, that is the column j is empty, the most frequent label of non-empty columns is assigned to m_{kj}.

Note also that attribute x_i may be irrelevant if, for $A = \{x_i\}$, all columns are compatible. Thus, during a preprocessing *attribute-redundancy remover*, HINT applies the decomposition step, partitioning attribute set into $A_i = \{x_i\}$ and $A'_i = S - \{x_i\}$ in order to remove all irrelevant attributes. Since the order in which attributes are selected for evaluating their redundancy may affect the outcome, attributes are processed in the reverse order of their relevancy measure as estimated by the ReliefF algorithm [Kononenko, 1994].

5.2 MFE2/GA and HINT

HINT and MFE2/GA share two important factors. First, they apply non-algebraic form to represent constructed features, which is preferred when concept is complex and no prior information is available for choosing the appropriate algebraic operators. Second, both methods induce features directly from data to represent them by non-algebraic representation. However, they are different in many aspects.

The first important difference between two systems is the procedure applied for inducing features from data. The procedure used by each method differs, mainly, in the labeling of cases or samples. HINT groups compatible samples and assigns a label to each group (Section 5.1). It represents features by the set of labeled samples. MFE2/GA groups samples into *pure*, *mixed* and *unknown* cases and labels each case differently (Section 4.2.1). The feature is represented by a vector of values, which indicates the outcome of the function for each combination of attribute values.

MFE2/GA creates features with binary labels (no more labels than the target concept itself), whereas HINT can use more labels. This property allows HINT to break down a high-order complex interaction into smaller interactions (as empirically shown in Section 5.3.1). When there is an interaction among smaller interactions, there are several mixed cases that should be treated differently. Thus, different labels are needed for different mixed cases.

To illustrate how HINT differs from MFE2/GA in constructing functions, consider the concept $F_1(x_1, x_2, x_3, x_4) \overset{\text{def}}{=} w(x_1, x_2) > w(x_3, x_4)$, where $w(x_i, x_j)$ is weight of Boolean attributes x_i and x_j. The complex interaction F_1 can be considered as an interaction (i.e., "being greater than") between two smaller interactions $w(x_1, x_2)$ and $w(x_3, x_4)$. Partitioning the attribute set into $A = \{x_1, x_2\}$ and $A' = \{x_3, x_4\}$ gives a partition matrix as shown in Figure 5.4(a). The first column of the matrix is considered as a *pure* case, and columns 2, 3, and 4 as *mixed* cases. After projecting data onto A, MFE2/GA labels the first column as '0', and the three impure columns as '1'. Such labeling gives an irrelevant feature equivalent to $x_1 \vee x_2$. Thus, MFE2/GA ignores this subset eventually and projects data onto the larger set $S = \{x_1, x_2, x_3, x_4\}$ to extract a function representing F_1. Since the size of S is larger, more training data are needed to extract the function from S itself. HINT, after partitioning attribute set into A and A' and considering columns compatibility, assigns three different color labels to columns: '0' to the first column, '1' to compatible second and third columns, and '2' to the forth column. Note that this labeling is equivalent to the interaction $w(x_1, x_2)$. Since HINT can break down the function F_1 into smaller ones, it needs less data to construct better features.

Note that for this example all data were available only for the purpose of illustration. When few data samples are provided, multi-coloring may not work as expected. Multi-coloring may cause overfitting, making HINT sensitive to data size. For example, con-

x_3 x_4 \ x_1	0	0	1	1
\ x_2	0	1	0	1
0 0	0	1	1	1
0 1	0	0	0	1
1 0	0	0	0	1
1 1	0	0	0	0
MFE2/GA labels	0	1	1	1
HINT labels	0	1	1	2

(a) $F_1 \overset{\text{def}}{=} w(x_1, x_2) > w(x_3, x_4)$

x_2 x_3 x_4 x_5 \ x_1	0	0	1	1
\ x_6	0	1	0	1
0 0 0 0	–	0	–	1
0 0 0 1	0	0	0	0
0 0 1 0	0	0	0	0
0 0 1 1	0	0	0	0
0 1 0 0	0	0	0	0
0 1 0 1	0	0	0	0
0 1 1 0	1	–	0	–
0 1 1 1	0	0	0	0
1 0 0 0	0	0	0	0
1 0 0 1	1	–	0	–
1 0 1 0	0	0	0	0
1 0 1 1	0	0	0	0
1 1 0 0	0	0	0	0
1 1 0 1	0	0	0	0
1 1 1 0	0	0	0	0
1 1 1 1	–	0	–	1
MFE2/GA labels	1	0	0	1
HINT labels	0	0	1	1

(b) $F_2 \overset{\text{def}}{=} (x_1 = x_6) \wedge (x_2 = x_5) \wedge (x_3 = x_4)$

Figure 5.4: Comparing the functions constructed by MFE2/GA and HINT

sider the partition matrix illustrated in Figure 5.4(b) for the concept $F_2(x_1, x_2, \ldots, x_6)$, which is true if $x_1 = x_6$, $x_2 = x_5$, and $x_3 = x_4$ (palindrome of six Boolean attributes). F_2 is composed of a simple relation among three complex interactions. In spite of eight missing values (88% training data), MFE2/GA projects data onto $A = \{x_1, x_6\}$, and labels second and third columns (*pure* cases) as zero and first and forth columns (*mixed* cases) as one, which is equivalent to $x_1 = x_6$. HINT, because of missing data, cannot see the incompatibility between first and second columns; therefore, it may assign the same color label to these columns and other color label to columns three and four, which gives a function equivalent to attribute x_1 itself. Note that in many real-world domains fewer training data will be available, which is worsening the problem (see Section 5.3.2 for empirical results).

HINT needs to see two samples with different labels in the same row to see the incompatibility of columns. When the number of different class labels in two columns is small and few data are available, HINT colors columns incorrectly. This problem affects MFE2/GA less than HINT because it looks at all cases in each column. If the column is a *pure*, in spite of missing data, MFE2/GA classifies it as *pure* case. If the column is a *mixed*, then MFE2/GA misclassifies it only when it cannot see the mixture of class labels, i.e., either all positive cases or all negatives are missed. Therefore, MFE2/GA is less sensitive to data size in this case.

Labeling of empty columns (which are cases that are not seen in training data) differs also in HINT and MFE2/GA. HINT assigns the most frequent color of non-empty columns to an empty column. MFE2/GA considers the empty column as an unknown case and assigns a label stochastically, according to the class distribution in the training data. As an example see the constructed functions by MFE2/GA and HINT in Figures 4.3 and 5.2.

The second important difference between these two systems relates to subset selection and feature construction. HINT successively selects subsets of attributes and constructs features one by one. If, in early steps, attributes are selected incorrectly and irrelevant features are constructed, all successive features will be irrelevant too. This greedy technique of HINT degrades its performance when several complex interactions among attributes exists. Conversely, through the use of GA, MFE2/GA is able to generate individuals, each composed of several subsets of attributes with their corresponding functions. Thus, several functions are generated and evaluated together in MFE2/GA. This multi-feature construction is very important when the target concept is complex and few data samples are provided. Section 5.3.2 empirically shows the advantage of multi-feature construction.

The third important distinction to note is the way the two methods deal with irrelevant attributes. HINT, during its preprocessing attribute-redundancy remover, eliminates irrelevant attributes and represents the concept $y = F(S)$ as $y = F'(S')$, where S' is set of relevant attributes. Since attributes are evaluated and removed one by one, when complex interaction exists among attributes, a relevant attribute may be considered irrelevant and eliminated from set of attributes. MFE2/GA generates features F_1 to F_k in parallel, where $y = R(F_1(A_1), ..., F_k(A_k))$, $\bigcup A_i \subseteq S$, $A_i \cap A_j$ is the set of attributes shared between two interactions, and $S - \bigcup A_i$ is the set of irrelevant attributes. Thus, MFE2/GA implicitly ignores irrelevant attributes by selecting subsets of relevant attributes.

Another distinction relates to the attributes participating in more than one interaction, which are called *shared attributes*. When the problem consists of several interactions that share attributes, parallel feature construction allows MFE2/GA to discover and highlight interactions. HINT, due to its greediness for constructing features one by one, cannot discover interactions with shared attributes since this kind of interactions cannot be represented by a hierarchy of constructed functions.

Finally, HINT imposes construction of functions over attribute subsets of a limited size to reduce the search complexity. Recall that HINT explores all partitions of S into A and A' where $|A|$ is bellow the limit established by a system parameter. This limitation may leave out some promising functions. MFE2/GA does not need this restriction. GA in MFE2/GA allows a broader search yet in a reasonable computation time.

5.3 Empirical Evaluation

This section presents an empirical analysis of MFE2/GA and HINT performed [Shafti and Pérez, 2007c] to support the theoretical comparison given in Section 5.2. It studies the performance of the two systems over concepts of synthetic and real-world domains, to evaluate different functionalities of these methods when few training data are provided. The synthetic problems are divided into two groups whose complexity has different nature: concepts that are composed of one complex interaction, and concepts that are composed of several complex interactions. They represent sources of difficulty that appear in real-world problems where primitive attributes are used for representing data. The concept defined over real-world domain is the Braille-detection problem.

For each problem, target concept, and size of training data, each system is run 20 times independently over 20 sets of shuffled data, and the performance is evaluated by calculating the average predictive accuracy using unseen test data. Default parameters are used for HINT[1] and MFE2/GA, unless otherwise stated. When MFE2/GA finishes, data modified after adding the constructed features are used by C4.5 [Quinlan, 1993] for learning and accuracy is evaluated by unseen data. To see the utility of features constructed by HINT and MFE2/GA, experiments are repeated with C4.5 and C4.5-Rules [Quinlan, 1993] on original data as baseline performance.

5.3.1 Concepts Composed of One Complex Interaction

A first set of experiments is performed over synthetic concepts with one complex interaction among attributes. Concepts, which were previously used in [Pérez and Rendell, 1995], are defined over 12 Boolean attributes. Most of these concepts are also used for experiments of Section 4.4.2. See Appendix A for their definition. To observe the effect of training data size, four sets of experiments are run using 1%, 3%, 5%, and 10% of all 2^{12} instances as training data, and the rest (99%, 97%, 95%, and 90% respectively) are kept unseen as test data for final evaluation.

Table 5.1 presents the average predictive accuracies of C4.5, C4.5-Rules, HINT, and MFE2/GA over different size of training data. Concepts are ordered increasingly by the number of interacting attributes shown in column 2. The majority class percentage of each concept is given in the third column. The higher of the two average accuracies obtained by C4.5 and C4.5-Rules is reported in columns 4, 7, 10, and 13. This result is marked by c if obtained by C4.5, or by r if obtained by C4.5-Rules. Numbers between parentheses denote standard deviations. When using 1% training data, the accuracy of MFE2/GA is marked by † when it is better than HINT's average accuracy. The highest average accuracy, within each training size section, is marked by $\triangleleft$, but only when it is not lower than the majority class percentage. MFE2/GA's average accuracy is significantly better than those in bold and significantly worse than those in italic; the significantly higher accuracy of the two results obtained by HINT and C4.5/R is marked by $\circ$ (using t-distribution test, with $\alpha = 0.02$). "N/A" means that HINT's attribute-redundancy remover regards all attributes as irrelevant, due to small size of data for this concept. The overall average accuracy for each method is shown in the last row of the table.

When 1% data (41 samples) are used for training, for some concepts, none of the

[1]as implemented in Orange data mining library [Demsar *et al.*, 2004].

Table 5.1: Comparing average predictive accuracies over concepts with one complex interaction

	Rel.	Maj	1% Data			3% Data			5% Data			10% Data		
Concept	atts	%	C4.5/R	HINT	MFE2/GA	C4.5/R	HINT	MFE2/GA	C4.5/R	HINT	MFE2/GA	C4.5/R	HINT	MFE2/GA
$P_{1,4}$	4	50.0	r **50.3(2.8)**	**69.1(25.9)**○	97.8(3.7)†◁	r **54.6(5.5)**	**92.1(19.2)**○◁	100(0.0)◁	r **67.9(12.9)**	100(0.0)○◁	100(0.0)◁	r **91.8(10.5)**	100(0.0)○◁	100(0.0)◁
$gw_{5,8}$	4	68.8	r 85.3(6.0)○	**72.8(10.8)**	90.7(9.6)†◁	r 100(0.0) ◁	98.9(4.8)	100(0.0)◁	r 100(0.0) ◁	99.0(4.6)	100(0.0)◁	c 100(0.0)◁	100(0.0) ◁	100(0.0)◁
$sw_{5,8}$	4	62.5	c **58.3(5.2)**	**68.3(21.3)**	96.5(4.8)†◁	r **74.5(14.2)**	100(0.0)○◁	100(0.0)◁	r **85.0(7.5)**	100(0.0)○◁	100(0.0)◁	r 99.1(2.9)	100(0.0) ◁	100(0.0)◁
$mx_6c_{6,7}$	4	50.0	c **60.0(4.6)**	**58.7(16.5)**	97.2(3.8)†◁	r **86.3(5.3)**	95.3(14.3)○	100(0.0)◁	r **93.7(6.2)**	100(0.0)○◁	100(0.0)◁	r 100(0.0)◁	100(0.0) ◁	100(0.0)◁
$mj_{4,8}$	5	50.0	r 78.6(4.0) ◁	72.1(12.2)	78.0(9.0)†	r **89.4(4.0)**	94.6(10.8)	98.9(1.6)◁	r **98.8(1.9)**	100(0.0)○◁	100(0.0)◁	c 100(0.0)◁	100(0.0) ◁	100(0.0)◁
$P_{1,6}$	6	50.0	r 49.7(0.1)	53.7(15.9)	54.2(9.9)†◁	r **49.3(0.2)**	86.9(23.2)○	93.8(2.7)◁	r **48.9(0.3)**	**100(0.0)**○◁	98.0(1.5)	r **49.1(2.0)**	100(0.0)○◁	100(0.0)◁
$gw_{4,9}$	6	65.6	r 76.3(3.4)○◁	70.2(9.0)	74.1(5.0)†	r **84.4(2.9)**	84.1(13.9)	91.3(4.2)◁	r **90.6(2.6)**	95.9(9.3)	98.0(2.0)◁	r **99.2(0.9)**	100(0.0)○◁	100(0.0)◁
$sw_{4,9}$	6	68.8	c 60.3(5.4)	59.1(10.0)	62.5(4.7)†	c **63.1(2.9)**	87.5(19.0)○	92.3(2.3)◁	c **66.2(3.1)**	*99.8(0.5)*○◁	98.6(1.4)	r **79.0(9.1)**	100(0.0)○◁	100(0.0)◁
$mx_6c_{5,8}$	6	50.0	r *67.7(4.5)* ◁	63.6(12.9)	60.9(5.9)	c **70.8(1.4)**	82.5(19.6)○	92.9(2.7)◁	c **71.3(1.6)**	*99.4(1.1)*○◁	97.6(1.6)	r **77.3(2.9)**	100(0.0)○◁	100(0.0)◁
$mj_{3,9}$	7	50.0	r *71.8(5.4)*○◁	64.6(6.3)	66.8(6.2)†	r *80.4(2.7)*	*86.3(15.0)* ◁	75.7(7.8)	r **84.8(1.7)**	*96.0(8.0)*○◁	90.0(2.4)	r **90.9(1.5)**	*99.9(0.3)*○◁	97.7(1.2)
$nm_{4,5,7}$	7	56.2	r 64.6(3.5) ◁	60.0(8.0)	61.6(5.5)†	r 70.5(3.6)	*84.1(17.7)*○◁	73.3(7.7)	r **74.3(3.9)**	*96.0(10.0)*○◁	89.8(2.5)	r **84.0(3.1)**	*99.8(0.5)*○◁	98.3(0.9)
$rk_{5,7}$	7	83.6	c 80.5(4.2)○	**71.9(5.6)**	77.6(3.6)†	c **81.7(2.1)**	83.4(11.3)	85.0(5.2)◁	c **81.7(1.5)**	93.5(9.2)○	93.7(3.7)◁	r **84.8(1.5)**	*99.8(0.5)*○◁	98.7(1.1)
$rk_{6,7}$	7	94.5	c *93.5(3.2)*	**N/A**	90.6(4.0)†	c *94.1(2.0)*○	**90.1(3.0)**	92.7(1.4)	c **94.1(0.6)**	**93.1(4.0)**	95.9(2.7)◁	r **95.2(1.1)**	98.5(2.6)○	99.4(1.1)◁
$gw_{3,10}$	8	63.7	r *73.2(3.0)*○◁	66.9(5.3)	68.9(4.1)†	r *79.7(2.3)*	*85.3(12.2)* ◁	73.7(5.3)	r *83.4(1.5)*	*96.0(8.0)*○◁	76.5(5.1)	r 87.2(1.1)	*99.5(0.5)*○◁	89.1(3.6)
$sw_{3,10}$	8	72.7	c 64.7(7.4)○	**56.5(4.9)**	63.0(4.8)†	c 68.7(3.9)	*83.9(19.1)*○◁	68.5(4.1)	c **68.0(3.5)**	*93.2(13.6)*○◁	75.9(5.4)	c **69.4(2.3)**	*99.7(0.4)*○◁	88.1(7.6)
$mx_6c_{4,9}$	8	50.0	c *73.5(1.8)*○◁	*64.1(8.3)*	58.1(6.6)	c *73.8(0.1)*	*88.2(14.8)*○◁	66.7(3.4)	c 73.2(1.3)	*95.2(8.3)*○◁	75.5(5.2)	c **72.8(0.2)**	*98.9(0.6)*○◁	85.0(8.3)
$mj_{2,10}$	9	50.0	r *70.5(3.4)*○◁	61.3(6.4)	64.1(7.1)†	r *76.4(2.4)* ◁	*72.0(8.3)*	65.8(5.5)	r *79.3(1.5)*	*92.2(10.1)*○◁	67.3(3.5)	r *83.1(1.0)*	*99.5(0.5)*○◁	76.0(2.9)
$nm_{5,6,9}$	9	59.0	r 59.4(2.6)○◁	55.7(4.9)	58.7(4.4)†	r *65.2(2.1)*	*71.7(15.0)* ◁	59.8(3.6)	r 66.2(1.9)	*90.5(15.2)*○◁	65.9(3.4)	r 71.4(3.1)	*99.3(0.6)*○◁	73.6(4.5)
$rk_{6,9}$	9	83.6	c *79.6(4.8)*○	72.5(5.2)	74.9(4.4)†	c 80.6(3.0)	**73.0(4.5)**○	79.6(3.3)	c 81.0(2.3)	*89.4(10.4)*○◁	80.5(2.2)	c 82.7(1.4)	*99.6(0.6)*○◁	83.3(3.4)
$rk_{7,9}$	9	93.0	c *92.6(1.5)*○	86.4(3.9)	88.5(3.9)†	c *92.5(1.3)*	**87.5(2.1)**○	90.0(1.4)	c *92.7(0.8)*○	**89.5(1.8)**	91.3(1.1)	c 92.7(0.8)	*98.2(1.1)*○◁	92.0(1.7)
$mx_6c_{3,10}$	10	50.0	c *73.4(4.4)*○◁	65.8(9.8)	62.2(8.7)	r *74.3(0.1)* ◁	*66.9(15.3)*	55.9(5.7)	c *74.0(0.1)*	*96.8(8.2)*○◁	57.6(4.4)	c *73.4(0.2)*	*99.4(0.5)*○◁	65.9(7.3)
Average	7	63.0	c 70.1(3.8)	65.7(10.2)	73.7(5.7)†◁	r 76.1(3.0)	85.4(12.5) ◁	83.6(3.2)	r 79.4(2.8)	96.0(5.8) ◁	88.2(2.3)	r 84.5(2.2)	99.6(0.4) ◁	92.7(2.1)

c	results obtained by C4.5
r	results obtained by C4.5-Rules
N/A	results are not available
◁	highest average accuracy
†	better than HINT
○	significantly higher between HINT and C4.5/R
Boldface:	MFE2/GA's accuracy is significantly better than this accuracy
Italic:	MFE2/GA's accuracy is significantly worse than this accuracy

methods obtain an accuracy better than the majority class percentage. In these cases, classifying samples by the label of the majority class gives better results than those obtained by the methods. When that is not the case, for most concepts the best system is C4.5/R. MFE2/GA and HINT use data samples to induce new functions, so they are sensitive to the data quantity. Due to small data size, for most concepts, the methods cannot construct proper functions to improve learning task and achieve better accuracy than C4.5/R. However, if few attributes participate in interaction (i.e., four attributes), MFE2/GA achieves higher accuracy than the others and significantly outperforms HINT. For these concepts, MFE2/GA discovers relevant attributes and successfully constructs proper functions since 41 samples are enough for this method to capture and encapsulate the interaction among four attributes. But, for HINT, this data size is not enough for learning as well as C4.5/R or MFE2/GA. Note that for almost all concepts MFE2/GA gives better results than HINT. The small data size along with the presence of irrelevant attributes make these concepts difficult to learn by HINT.

When the training data size is increased to 3%, both CI methods get more advantage from data and obtain better results than C4.5/R. MFE2/GA achieves higher accuracy than HINT and significantly outperforms C4.5/R, when the ratio of relevant attributes to total number of attributes is not high (less than seven interacting attributes). For these concepts, HINT cannot generate functions as properly as MFE2/GA due to overfitting. It breaks down the high-order complex interaction into smaller interactions represented by functions that are consistent with training data but produce errors when evaluated by unseen test data (as seen also by en example in Figure 5.4(b)). However, when the number of interacting attributes increases, the result changes. As explained in Section 5.2, MFE2/GA cannot break down these concepts into smaller interactions because of its bias for extracting functions from data. The function that represents the smaller interaction has different *mixed* cases that should be labeled differently. MFE2/GA cannot assign different labels to *mixed* cases. So, it tries to encapsulate the whole interaction by constructing a unique function. But, the training samples are not enough for capturing the interaction among all relevant attributes. On the other hand, HINT, by means of multi-coloring, successfully groups *mixed* cases into compatible columns and assigns a label to each group of compatible columns. So, it breaks down the high-order complex interaction into smaller ones connected to each other by another interaction. Since the underlying hierarchy of intermediate concepts is not complex, despite the small size of training data, HINT constructs proper functions. Hence, it significantly outperforms MFE2/GA for most concepts with high number of interacting attributes. It also significantly obtains better results then C4.5/R.

With 5% data the differences among methods are more visible. HINT significantly achieves higher accuracy than MFE2/GA and C4.5/R for most concepts since it has enough training samples for extracting functions. However, MFE2/GA still achieves an accuracy equal to or higher than HINT when few attributes participate in interaction;

Table 5.2: Summary of Table 5.1

Description	Frequency (out of 19)			
of the event	1% Data	3% Data	5% Data	10% Data
MFE2/GA$\big/$HINT	7/1	4/8	3/11	0/11
MFE2/GA$\big/$C4.5/R	3/9	9/8	13/4	11/2
HINT$\big/$C4.5/R	1/12	11/1	17/1	17/0

and significantly outperforms C4.5/R. With 10% data similar results obtained, though, for concepts with small number of interacting attributes now there are enough data samples for HINT to avoid overfitting and achieve a predictive accuracy as high as MFE2/GA.

Note that for most concepts the standard deviation of the accuracies obtained by HINT is high. Further analysis of the results shows that there are some trials of 20 runs where HINT cannot achieve good results because it confounds irrelevant attributes with interacting ones and, consequently, constructs irrelevant functions. This illustrates that the attribute-redundancy remover of HINT sometimes cannot detect irrelevant attributes.

Table 5.2 summarizes the results of the experiments. It shows the number of times a method significantly outperformed the other method for different data sizes. For an event "$\text{method}_1\big/\text{method}_2$", a frequency "$a/b$" means method_1 outperformed method_2 for a number of concepts out of 19 concepts, and method_2 outperformed method_1 for b number of concepts. As explained before and showed in this summary, when few training data is provided C4.5/R achieves better results; and MFE2/GA outperforms HINT. When the size of training data increases, both CI methods start to get more advantage of the training data and achieve better accuracy comparing to C4.5/R. HINT due to its multi-coloring outperforms MFE2/GA.

Concepts used for these experiments are composed of one complex interaction among four to ten attributes. Since MFE2/GA cannot break down complex interactions into smaller ones it fails to improve accuracy when more attributes participate in interactions. Next section presents experiments performed over concept that are composed of several interactions over smaller sets of attributes, that is, the number of interacting attributes is high (from six to eighteen attributes) but interactions are small (among three to six attributes each).

5.3.2 Concepts Composed of Several Complex Interactions

This section evaluates the two systems when there are more than one complex interactions in the concept. The synthetic concepts are designed to focus the empirical study on situations were multiple complex attribute interactions make FC necessary for learning and difficult to achieve due to complex underlying hierarchy of intermediate concepts. Table 5.3 summarizes concepts used for the experiments and the results. For all concepts, attributes are Boolean except in the last four concepts, where for Monk1 attributes are 2 to 4-valued and for the other three concepts attributes are 3-valued. The number of relevant and irrelevant attributes is reported in columns 2 and 3 of the table. The majority class percentage is denoted in column 4. Note that for some concepts there are attributes participating in more than one underlying interaction (shared attributes). For example, in $\wedge(\mathtt{P}_{1,4}, \mathtt{P}_{3,6})$, x_3 and x_4 are shared by $\mathtt{P}_{1,4}$ and $\mathtt{P}_{3,6}$. See Appendix A for definition of concepts, including a description of the complex interactions underlying these concepts.

Similar experiments to those in Section 5.3.1 are performed. As these concepts are more complex than those in the previous section, more data samples are needed for learning. Experiments are performed using 1%, 5% and 10% of all possible instances as training data and the rest of them as test data. Results of experiments over 3% data are not reported here because they are not so relevant for our empirical comparison. The average accuracy of C4.5, C4.5-Rules, HINT, and MFE2/GA over different size of training data are reported in Table 5.3 with the same format and notation as Table 5.1.

With 1% data, for almost all cases, HINT's accuracy is significantly lower than C4.5/R. MFE2/GA's result is also lower than C4.5/R, sometimes significantly. However, C4.5/R achieves accuracies still not higher than the majority class percentages for most cases; that is, there are few $\triangleleft$ symbols in the 1% data section. Thus, if we classify all data by the label of majority class, we obtain a higher accuracy than these methods in most cases due to lack of data. These concepts are composed of six to eighteen interacting attributes. Thus, 1% of data is not enough for learning, using any of methods. Note that in spite of having few data, MFE2/GA obtains higher accuracy than HINT (significantly for most cases), although both are often below majority class percentage.

With 5% data, the result is quite different. Now, in almost all cases MFE2/GA has an accuracy significantly better than other methods. Providing more training data also helps HINT to improve its accuracy. However, the overall average accuracy of HINT in this section is almost the same as the one obtained by C4.5/R. When 10% data are used, HINT starts to get more advantage of the size of data and improves its accuracy, but still MFE2/GA has an overall average accuracy better than HINT. This indicates that HINT needs more data than MFE2/GA for learning these concepts.

As explained in Section 5.2 and empirically illustrated in Section 5.3.1, one of the main characteristics of HINT is multi-coloring, that is, assignment of labels to groups of compatible columns. When there is one high-order interaction among attributes, lots of variations exist among columns. Thus, the incompatibility across columns is discovered by HINT; then, interacting attributes are selected and labels are assigned correctly. For this reason HINT achieved higher accuracy in experiments of the previous section. However, this is not the case for concepts in this section; thus, HINT

Table 5.3: Comparing average predictive accuracies over concepts with several complex interactions

	Rel.	Irrel.	Maj	1% Data			5% Data			10% Data		
Concept	atts	atts	%	C4.5/R	HINT	MFE2/GA	C4.5/R	HINT	MFE2/GA	C4.5/R	HINT	MFE2/GA
$\wedge(\mathtt{P}_{1,4},\mathtt{P}_{3,6})$	6	6	75.0	c 68.2(7.1)∘	**N/A**	66.2(6.4)	c**72.5(3.2)**	*100(0.0)*∘◁	99.1(1.6)	r **78.7(6.1)**	100(0.0)∘◁	100(0.0)◁
$\wedge(\mathtt{P}_{1,6},\mathtt{P}_{3,8})$	8	4	75.0	c 66.3(7.0)∘	**59.1(6.9)**	64.9(3.9)	c**73.4(2.7)**	*98.6(6.3)*∘◁	92.8(5.7)	c **74.4(1.2)**	*100(0.0)*∘◁	99.3(0.7)
$\wedge(\mathtt{P}_{1,6},\mathtt{P}_{7,12})$	12	0	75.0	c 67.3(8.0)∘	**58.5(6.3)**	63.5(4.2)	c**72.6(3.9)**	82.5(16.5)∘	87.4(8.2)◁	c **74.8(1.4)**	*100(0.0)*∘◁	97.8(1.6)
$\wedge(\mathtt{P}_{1,3},\mathtt{P}_{3,5},\mathtt{P}_{4,6})$	6	6	87.5	c *86.7(2.5)*∘	**76.5(5.1)**	80.2(4.1)	c**87.6(1.2)**	**94.1(9.6)**∘	99.8(0.7)◁	r **92.3(3.9)**	100(0.0)∘◁	100(0.0)◁
$\wedge(\mathtt{P}_{1,4},\mathtt{P}_{2,5},\mathtt{P}_{3,6})$	6	6	87.5	c *86.7(2.5)*∘	**76.2(6.0)**	82.0(4.7)	c**87.5(0.3)**	97.2(7.2)∘	99.6(0.7)◁	r **89.4(3.1)**	100(0.0)∘◁	100(0.0)◁
$\wedge(\mathtt{P}_{1,4},\mathtt{P}_{3,6},\mathtt{P}_{5,8})$	8	4	87.5	c *86.6(4.2)*∘	**70.5(5.7)**	76.2(5.8)	c**87.5(0.1)**	90.3(11.0)	96.4(1.8)◁	c **87.5(0.2)**	*100(0.0)*∘◁	99.3(0.8)
$\wedge(\mathtt{P}_{1,4},\mathtt{P}_{5,8},\mathtt{P}_{9,12})$	12	0	87.5	c *86.1(4.7)*∘	**72.9(6.8)**	78.6(4.2)	c 87.5(0.1)∘	**78.4(4.1)**	90.8(7.4)◁	c **87.5(0.2)**	98.3(5.1)∘	100(0.0)◁
$\wedge(\mathtt{P}_{1,6},\mathtt{P}_{2,7},\mathtt{P}_{3,8})$	8	4	87.5	c *86.7(2.5)*∘	**73.8(5.7)**	83.1(3.8)	c**86.6(1.8)**	92.3(10.0)∘◁	92.2(3.3)	c **87.2(0.8)**	99.0(4.3)∘◁	97.2(1.6)
$\wedge(\mathtt{WL3}_{1,5},\mathtt{WL3}_{3,7})$	7	5	64.1	r 75.1(5.0)∘	**69.9(6.8)**	75.3(5.7)◁	r**90.1(3.0)**	91.2(11.6)	93.6(5.3)◁	r **97.1(1.4)**	99.2(3.0)∘◁	98.8(1.0)
$\wedge(\mathtt{WL3}_{1,5},\mathtt{WL3}_{4,8})$	8	4	68.0	r 75.4(4.8)∘◁	**67.5(8.9)**	75.3(4.8)	r**86.7(2.0)**	**88.8(8.9)**	95.5(6.5)◁	r **96.0(1.8)**	99.4(0.7)∘◁	99.4(2.4)
$\wedge(\mathtt{WL3}_{1,5},\mathtt{WL3}_{5,9})$	9	3	71.5	r 73.9(3.2)∘◁	**67.7(6.1)**	73.6(3.9)	r**84.9(2.6)**	**88.0(10.1)**	95.3(5.4)◁	r **91.8(2.9)**	99.9(0.3)∘◁	99.6(0.9)
$\wedge(\mathtt{WL3}_{1,5},\mathtt{WL3}_{6,10})$	10	2	75.0	c 75.1(2.8)∘◁	**69.0(5.9)**	74.2(2.6)	r**82.2(2.1)**∘	**78.6(5.2)**	91.1(8.2)◁	r **88.6(3.3)**	98.4(4.1)∘	99.1(4.1)◁
$\wedge(\mathtt{WL3}_{1,4},\mathtt{WL3}_{3,6},\mathtt{WL3}_{5,8})$	8	4	57.8	r *72.6(5.6)*∘◁	63.5(5.8)	66.9(6.0)	r**89.2(4.1)**	**89.3(12.0)**	97.5(2.0)◁	r 99.1(1.7)	99.6(0.6) ◁	99.6(0.9)
$\wedge(\mathtt{WL3}_{1,4},\mathtt{WL3}_{5,8},\mathtt{WL3}_{9,12})$	12	0	67.5	r 69.4(2.8)∘◁	63.3(3.8)	66.5(4.9)	r**79.5(3.2)**∘	**71.8(4.5)**	91.5(11.0)◁	r **93.0(3.2)**∘	**86.4(8.8)**	100(0.0)◁
$\wedge(\mathtt{W23}_{1,6},\mathtt{W23}_{7,12})$	12	0	70.9	c 64.1(4.3)∘	**59.0(4.2)**	63.6(3.0)	r**68.2(2.3)**∘	**65.9(3.3)**	80.1(8.9)◁	r **72.5(3.2)**	92.4(10.7)∘	96.2(6.1)◁
$\wedge(\mathtt{W23}_{1,4},\mathtt{W23}_{5,8},\mathtt{W23}_{9,12})$	12	0	75.6	c 69.9(5.4)∘	**63.2(4.7)**	67.0(4.8)	r**74.7(1.9)**∘	**69.7(3.0)**	87.8(11.2)◁	r **85.1(3.5)**	**85.1(10.4)**	100(0.0)◁
$\wedge(\mathtt{W23}_{1,5},\mathtt{W23}_{6,10},\mathtt{W23}_{11,15})$	15	0	76.0	c**72.3(1.8)**∘	**65.3(3.4)**	76.8(5.7)◁	r**88.5(3.1)**	98.9(2.8)∘	100(0.0)◁	r **95.5(2.7)**	100(0.0)∘◁	100(0.0)◁
$\wedge(\mathtt{W23}_{1,6},\mathtt{W23}_{7,12},\mathtt{W23}_{13,18})$	18	0	84.0	r**82.7(1.6)**∘	**74.1(2.9)**	98.8(3.1)◁	r**98.1(0.9)**	100(0.0)∘◁	100(0.0)◁	r 100(0.0) ◁	100(0.0) ◁	100(0.0)◁
$\wedge(\mathtt{A}_{1,4},\mathtt{A}_{5,8},\mathtt{A}_{9,12})$	12	0	82.2	c 79.2(4.4)∘	**72.1(5.2)**	76.0(4.7)	r 89.8(5.0)∘	**79.7(3.1)**	94.2(7.0)◁	r 100(0.0)∘◁	**89.6(6.5)**	99.9(0.2)
$\wedge(\mathtt{B}_{1,4},\mathtt{B}_{5,8},\mathtt{B}_{9,12})$	12	0	87.5	c *86.2(3.0)*∘	75.4(6.3)	78.7(4.0)	c**86.9(1.3)**∘	**81.1(2.1)**	89.8(4.3)◁	r **92.8(3.1)**∘	**85.8(4.1)**	100(0.0)◁
$\wedge(\mathtt{C}_{1,4},\mathtt{C}_{5,8},\mathtt{C}_{9,12})$	12	0	57.8	c 56.0(3.7)∘	**52.8(3.2)**	56.1(3.7)	r**66.2(3.8)**	**64.6(7.8)**	98.1(8.7)◁	r **79.3(5.7)**	100(0.0)∘◁	100(0.0)◁
$\wedge(\mathtt{D}_{1,4},\mathtt{D}_{5,8},\mathtt{D}_{9,12})$	12	0	87.5	c *85.8(3.1)*∘	**N/A**	82.3(4.1)	r 90.6(2.8)∘	**83.7(1.9)**	91.0(3.5)◁	r 98.6(2.7)∘	**89.0(3.0)**	98.9(2.4)◁
$\wedge(\mathtt{E}_{1,4},\mathtt{E}_{5,8},\mathtt{E}_{9,12})$	12	0	75.6	c 70.1(5.9)∘	**62.2(5.3)**	68.1(3.9)	r**77.0(3.0)**∘	**72.2(4.8)**	90.7(10.7)◁	r **91.5(5.1)**	93.9(11.1)	100(0.0)◁
$\wedge(\mathtt{A}_{1,4},\mathtt{C}_{5,8},\mathtt{E}_{9,12})$	12	0	73.6	c 69.3(4.4)∘	**64.0(6.3)**	68.1(3.6)	r**82.2(3.4)**∘	**73.7(5.6)**	97.8(4.9)◁	r **87.3(4.3)**	**94.1(8.7)**∘	100.0(0.2)◁
$\wedge(\mathtt{A}_{1,4},\mathtt{B}_{5,8},\mathtt{D}_{9,12})$	12	0	85.9	c *84.2(2.6)*∘	**N/A**	79.7(4.3)	r 87.6(3.6)∘	**81.5(3.4)**	90.4(4.5)◁	r **96.2(3.4)**∘	**91.3(6.2)**	99.8(0.7)◁
$\wedge(\mathtt{A}_{1,4},\mathtt{B}_{5,8},\mathtt{C}_{9,12})$	12	0	78.9	c 71.6(5.6)∘	67.5(4.8)	69.9(3.8)	r**86.3(3.5)**∘	**75.8(4.3)**	94.0(7.1)◁	r **89.0(2.8)**	**96.2(6.3)**∘	100(0.0)◁
$\wedge(\mathtt{B}_{1,4},\mathtt{C}_{3,6},\mathtt{A}_{7,10},\mathtt{D}_{9,12})$	12	0	86.5	c *84.0(4.7)*∘	77.4(4.8)	80.1(4.1)	r 88.5(2.0)∘	**83.2(2.8)**	90.2(2.9)◁	r **94.4(1.8)**∘	**89.0(3.9)**	98.0(2.0)◁
$\wedge(\mathtt{A}_{1,4},\mathtt{B}_{5,8},\mathtt{C}_{9,12},\mathtt{E}_{13,16})$	16	0	87.0	r**89.5(2.4)**∘	**81.3(2.4)**	96.5(5.4)◁	r**94.8(2.1)**	99.8(1.1)∘	100(0.0)◁	r **98.7(1.2)**	100(0.0)∘◁	100(0.0)◁
$\wedge(\mathtt{C}_{1,4},\mathtt{WL3}_{5,8},\mathtt{W23}_{9,12})$	12	0	67.8	r 63.3(3.6)	60.8(4.5)	63.5(3.3)	r**74.2(3.1)**	**70.6(7.2)**	97.9(6.3)◁	r **83.2(4.5)**	**93.8(10.2)**∘	100(0.0)◁
$\wedge(\mathtt{W23}_{1,5},\mathtt{C}_{5,8},\mathtt{WL3}_{8,12})$	12	0	76.6	c 71.9(3.9)∘	66.6(5.4)	70.5(4.6)	c**76.4(1.2)**∘	**71.2(2.4)**	80.7(6.3)◁	r **80.4(3.0)**	**84.0(7.8)**	98.4(3.3)◁
$\wedge(\mathtt{W23}_{1,5},\mathtt{C}_{4,7},\mathtt{WL3}_{6,10})$	10	2	76.6	c 73.0(3.6)∘	**66.6(5.5)**	70.5(4.0)	r**77.4(2.8)**	**75.9(6.6)**	84.4(6.3)◁	r **83.9(2.3)**	96.4(4.6)∘	98.5(1.8)◁
Monk1	6	0	50.0	r 49.7(2.8)∘	**N/A**	50.6(8.1))◁	r**65.2(8.3)**	70.1(13.4)	77.0(10.6)◁	r**81.0(11.7)**	**89.5(6.0)**∘	99.6(1.8)◁
$\mathtt{palindrome}_6+2$	6	2	96.0	c *96.3(0.0)*∘◁	**N/A**	93.9(2.2)	c**96.3(0.1)**∘	**93.2(2.0)**	99.6(0.7)◁	c **96.3(0.1)**	**99.0(1.8)**∘	100(0.0)◁
$\vee(\mathtt{pal}_{1,4},\mathtt{pal}_{3,6},\mathtt{pal}_{5,8})$	8	0	70.0	c 67.6(4.3)∘	**57.8(3.4)**	65.4(3.2)	r 71.2(2.6)∘◁	**63.8(4.1)**	70.6(3.1)	r *82.0(3.7)*∘◁	74.7(9.1)	76.3(4.0)
$\vee(\mathtt{pal}_{1,4},\mathtt{pal}_{4,7},\mathtt{pal}_{7,10})$	10	0	70.0	r 72.2(3.1)∘◁	**65.1(6.3)**	71.1(1.2)	r**97.5(2.3)**	100(0.0)∘◁	100(0.0)◁	r 99.9(0.3)	100(0.0) ◁	100(0.0)◁
Average	10	2	76.6	c 75.7(3.9)	67.3(5.3)	73.5(4.3)	r 82.2(3.1)	83.3(5.7)	92.5(5.0)◁	r 88.5(3.1)	95.0(4.9)	98.7(1.0)◁

Table 5.4: Summary of Table 5.3

	Frequency (out of 34)		
Description of the event	1% Data	5% Data	10% Data
MFE2/GA/HINT	28/0	23/2	13/3
MFE2/GA/C4.5/R	3/11	28/0	28/1
HINT/C4.5/R	0/33	10/16	22/7

cannot take advantage of its multi-coloring to outperform MFE2/GA. So, for almost all cases MFE2/GA has significantly better accuracy than HINT. The result of concept $palindrome_6$+2 empirically illustrates what was predicted in Section 5.2 regarding the problem of multi-coloring when few data are available.

The low accuracy of HINT is due to these concepts being composed of two or more interactions of three or more attributes each. The underlying hierarchy of intermediate concepts is large and complex. HINT applies a greedy strategy for constructing this hierarchy. The goodness of the feature constructed in each step has a direct effect on the construction of other features in next steps. As explained in Section 5.2, when few data are available, it is more probable that HINT overfits data by selecting attributes and constructing features incorrectly. Therefore, the feature constructed by HINT is irrelevant to the concept; and subsequently, the other features constructed in the next steps are irrelevant too. This explains the high standard deviation of HINT's result for most of these concepts when the size of training data is small. Either HINT gets the chance to select attributes correctly in early steps and construct a correct hierarchy, or its early mistakes misguide it toward an irrelevant hierarchy.

Additional study and analysis of the features generated by HINT indicates that HINT's attribute-redundancy remover fails for some concepts due to selecting attributes one by one for redundancy evaluation. When few data are available a relevant attribute may be considered as redundant and removed from original attribute set. In addition to losing information, removing a relevant attribute causes irrelevant attributes to be considered as relevant.

MFE2/GA overcomes these problems caused by few data, by selecting several subsets of attributes simultaneously, which are then used to construct several features; and constructed features are also evaluated together. Note that for all concepts the majority of attributes participate in interactions (for some cases all attributes), but they can be represented by a relation among smaller interactions. Thus, MFE2/GA successfully breaks down the relation among attributes into small interactions represented by constructed functions. Hence, it can find the exact target concept or a close approximation to it due to its multi-feature construction. For this reason, the standard deviation of its accuracy is lower than HINT's.

Table 5.4 summarizes the results with the same format as Table 5.2. It can be seen that when few training data is provided the result is similar to those shown in Table 5.2. But when the size of training data increases the results are quit different. MFE2/GA significantly outperforms other methods due to its multi-feature construction. HINT cannot take advantage of its multi-coloring property and, therefore, outperforms MFE2/GA only for few concepts. It outperform C4.5/R when enough training data samples (i.e., 10% data) are provided.

5.3.3 Experiments with Braille-detection Problem

This section reports on a similar empirical comparison of the two systems, but this time based on a task defined over a real-world domain. A Braille code is a 3×2 matrix of raised/unraised dots. Figure 5.5(a) shows the Braille code as it was originally invented for French alphabet (which did not include the w), where raised and unraised dots are shown by black and white circles respectively. The target concept is to distinguish Braille-coded text from randomly generated codes, using a windowing of three codes. Each sample consists of three codes, each represented by six Boolean attributes forming a total of 18 attributes. If all three codes are Braille, the sample is classified as true, and otherwise, it is classified as false. Figure 5.5(b) shows a valid sample in Braille-detection Problem.

The Braille-detection concept is composed of complex interactions among attributes, which are not easily represented by an algebraic form. For each code (a matrix of six attributes), the third and sixth attributes determine that the code pertains to which group: the first group that consists of letters a to j, the second group that is k to t, or the third one that is u to z. The other four attributes determine the position of the code in the group. Note that if the code belongs to the third group then only the first five positions are valid, as shown in Figure 5.5(a). Thus, each code is represented by several interactions among attributes. Since the concept is composed of three codes (three sets of interactions), the underling hierarchy of intermediate concepts is complex, and hard to discover by a greedy method such as HINT when few training data samples are available.

A total of 20 data sets of 31250 samples are generated with majority class of 50%. These data sets are used to carry out a set of experiments similar to those from the previous section. Experimental results showed that the tree generated by C4.5 using features constructed by MFE2/GA has the new features near the root, but still uses many primitive attributes at deeper levels. This indicates that the features generated were not enough for abstracting all interactions. So, the parameter K that is the maximum number of features in each individual (Section 4.2.1) is increased from five to nine, allowing MFE2/GA to generate more features. This requires more CPU time, but a single learning trial still takes only a few minutes (for 5% data about two minutes on a Pentium 4).

Table 5.5: Comparing results over Braille-detection problem

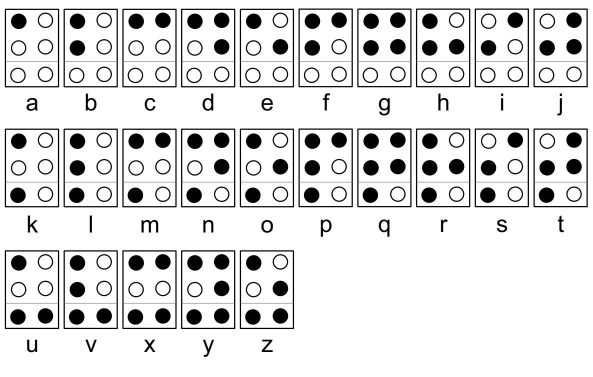
Data Size	C4.5	C4.5-Rules	HINT	MFE2/GA
1%	90.7(1.9)○	*94.8(2.2)*○◁	**75.9(4.1)**	89.9(4.6)
5%	**97.6(0.4)**○	99.6(0.3)○◁	**90.2(3.4)**	99.1(2.0)
10%	**98.6(0.3)**○	99.9(0.2)○	**95.9(3.4)**	100.0(0.1)◁
15%	**99.0(0.2)**	**99.9(0.1)**○	**99.0(0.8)**	100.0(0.0)◁
20%	**99.4(0.1)**	100.0(0.0)○	**99.4(0.6)**	100.0(0.0)◁

◁	highest average accuracy
○	significantly better than HINT
Boldface:	MFE2/GA is significantly better than this accuracy
Italic:	MFE2/GA is significantly worse than this accuracy

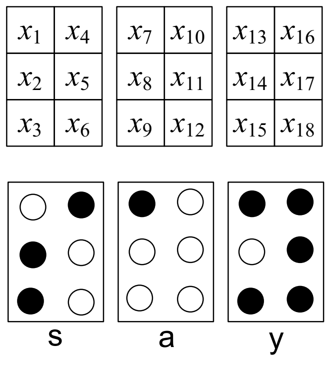
Experiments are performed increasing training data from 1% to 20% to see how data size affects these methods. Table 5.5 shows accuracies of C4.5, C4.5-Rules, HINT, and MFE2/GA for different size of training data. MFE2/GA's accuracy is significantly better than those in bold and worse than those in italic, and ○ means this accuracy is significantly better than HINT (using t-distribution test, with $\alpha = 0.02$). As shown in the table, MFE2/GA outperforms HINT and C4.5 for all data sizes except for 1%, where MFE2/GA gets slightly lower accuracy than C4.5. C4.5-Rules's accuracy is similar to MFE2/GA's, except for 1% data, where it is significantly higher and 15% data, where it is significantly lower than MFE2/GA.

The low accuracy and high standard deviation of MFE2/GA with 1% training data indicates that this method most of the times overfits data due to its dependency to data quantity for extracting new features (though it is not as sensitive as HINT). But if enough data is provided, MFE2/GA can achieve better accuracy comparing to other methods. With 10% data this method achieves 100% predictive accuracy over test data, whereas C4.5-Rules needs two times more data (20% data) to properly learn the concept and obtain 100% accuracy.

It is interesting to note that, although C4.5-Rules performs similarly to MFE2/GA,



(a) Braille code representation


x_1 x_4 x_7 x_{10} x_{13} x_{16}
x_2 x_5 x_8 x_{11} x_{14} x_{17}
x_3 x_6 x_9 x_{12} x_{15} x_{18}
s
a
y

(b) a sample classified as a valid code

Figure 5.5: Braille-detection problem

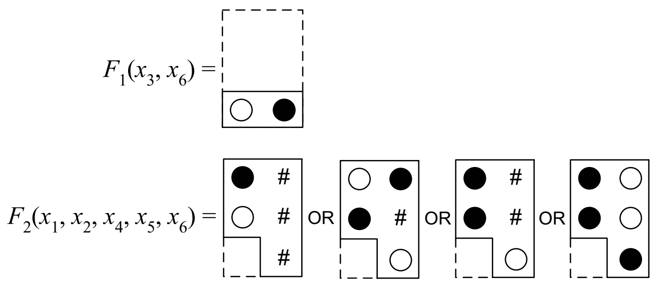


Figure 5.6: Sample features constructed by MFE2/GA for Braille-detection problem

it generates a large number of rules (often more than 35 rules for 5% data) that are difficult to interpret. Features generated by MFE2/GA and used in the tree can be easily interpreted. For all experiments MFE2/GA successfully discovers that there are three relations, of six attributes each, in the training data and constructs functions to highlight these three relations. Each relation corresponds to one position in the 3-code window. MFE2/GA usually constructs two functions for each relation of six attributes, representing the definition of a Braille code, in total six functions for a sequence of three Braille codes. Figure 5.6 shows the two functions that are usually constructed to define the valid codes represented by the first six attributes. Similar functions are found for the other groups of six attributes. The solid line in the figure shows the domain of each function. A black circle indicates the attribute value is '1' (raised dot), a white circle means the attribute value is '0' (unraised dot), a '#' means "`don't care`", i.e., it can be either '0' or '1'. The first function, F_1, highlights all codes with unraised dot 3 and raised dot 6, as invalid codes to be excluded from the target. The second function, F_2, is a disjunction of four rules to define all Braille letters ignoring dot 3. The conjunction, $\overline{F_1} \wedge F_2$, classifies all Braille codes. In algebraic form, $\overline{F_1} \wedge F_2$ is equivalent to $\overline{(\overline{x}_3 x_6)} \wedge (x_1\overline{x}_2 \vee \overline{x}_1 x_2 x_4 \overline{x}_6 \vee x_1 x_2 \overline{x}_6 \vee x_1 x_2 \overline{x}_4 \overline{x}_5 x_6))$, which will be longer when converted to DNF form. This phrase represents only one third of the concept (one code of three codes). The concept is the conjunction of three phrases like this. Therefore, a large number of rules are needed to represent this concept. Note also that when more data is available, MFE2/GA encapsulates the relation among six attributes and represents it by a single function. Thus, it constructs a total of just three functions, one function for each subset of six attributes, to represent a sequence of three Braille codes.

In spite of using non-algebraic representation similarly to MFE2/GA, HINT needs more data than the others to uncover the underlying concept structure and improve accuracy. Several interactions exist among 18 attributes in this concept. HINT needs to construct a complex hierarchy of functions representing interactions, which is a difficult task for its greedy procedure. MFE2/GA, due to use of genetic search and evaluation of multiple candidate features simultaneously, achieves better results than HINT. It can be observed from table that HINT's accuracy is significantly lower than C4.5-Rules and

MFE2/GA for all data sizes.

Experiments on the Braille-detection problem shows that there are real-world problems with multiple complex interactions which make FC necessary for learning and difficult to achieve by greedy methods.

5.4 Conclusion

This chapter focused on analytically and empirically comparison of two systems based on non-algebraic feature representation, MFE2/GA and HINT, when few training data are provided. Several distinctive aspects of these systems' design are identified and related to the functionality expected and observed from each system.

One of the main differences between these two methods and most FC methods is that the former extract features directly from data and represent them as data sets or vector of values. This becomes important when complex interaction exists and no domain knowledge is available for defining operators. However, because of this characteristic both methods depend heavily on training data to construct features.

The empirical evaluation shows that MFE2/GA is less sensitive to size of training data than HINT due to its multi-feature construction functionality. MFE2/GA constructs and evaluates several features simultaneously, as GA's individuals, which is convenient when concepts involve several complex interactions and few data are available. HINT applies a greedy strategy for constructing multiple features; therefore, if it fails to construct relevant features in early steps, features constructed in subsequent steps will be affected.

However, when the target depends mainly on one complex interaction, HINT outperforms MFE2/GA because of its multi-coloring functionality. It successfully breaks down the complex interaction into smaller parts. MFE2/GA's multi-feature construction is not an advantage when one interaction suffices to express the target, and does not help to decompose a complex interaction into parts. Although, multi-coloring of HINT sometime makes this method more sensitive to the quantity of training data than MFE2/GA.

The two methods also differ in the procedure for selecting interacting attributes. HINT applies a preprocessing method to remove irrelevant attributes. Then, it considers all attribute subsets of a predefined bound size to construct features. MFE2/GA uses GA to select attributes which makes this method more promising when the search space has high variation.

Comparing the results obtained by MFE2/GA with C4.5-Rules on Braille-detection problem shows that their accuracies are similar. But, it is important to point out that the number of rules generated by C4.5-Rules is large and, therefore, hard to comprehend while MFE2/GA encapsulates relations among attributes into a small number of features which are more easy to interpret.

This study suggests that the integration of both functionalities, multi-coloring and multi-feature construction, may create a synergy that improves learning performance (see Section 7.3).

Chapter 6

Improving the Fitness Evaluation: MFE3/GA

Fitness has a major role in guiding a genetic-based search to converge to the optimal solution. The fitness function intends to conduct the GA toward its goal and accelerate its convergence by providing a good estimate of the quality of each individual in the population. A simple fitness evaluation was proposed in Section 4.2.2 for MFE2/GA. This measure approximates the cost of representing data using new features and the complexity of the constructed features. Experiments over synthetic concepts in Section 4.4.2 illustrated that sometimes the fitness function does not perfectly guide MFE2/GA toward the global optimal solution.

Considering the importance of fitness, this chapter aims to improve the fitness evaluation in MFE2/GA to achieve better results. Section 6.1 reviews three types of fitness measures commonly used by genetic CI methods for constructing features. Sections 6.2 and 6.3 propose two new fitness functions based on entropy and the Minimum Description Length (MDL) Principle to be incorporated in MFE2/GA, leading to new versions of the method called MFE2/GA$_E$ and MFE3/GA, respectively [Shafti and Pérez, 2007a; 2007b]. Section 6.4 describes experiments performed to evaluate the new systems, showing that MFE3/GA with MDL-based fitness outperforms other methods. Once the appropriate fitness function is chosen, MFE3/GA is applied to real-world problems and the experimental results are reported in Section 6.5. Conclusions are described in Section 6.6.

6.1 Fitness Function

When a GA is applied to perform FC, the goal is to generate new features that facilitate more accurate learning when they are used to change the representation of training data. Thus, the fitness function should estimate the quality of the constructed features.

Constructed features may be evaluated in different ways. Three common forms of evaluating features are classifier error rate measure, entropy-based measure, and MDL-based measure, as described next.

Classifier Error Rate Measure

Since the main goal of CI is to improve the learning task and achieve higher predictive accuracy, most genetic CI methods apply classifier error rate as fitness measurement (i.e., a hypothesis-driven fitness function, as in [Vafaie and DeJong, 1998; Ritthoff *et al.*, 2002; Estébanez *et al.*, 2007]). These methods use the hypothesis generated by a learner to assess the usefulness of constructed features in classifying data. This approach is also referred to as a wrapper model by Kohavi and John in [1997]. For each individual, training data are redescribed using the new feature(s) in the individual. Then, the new training data is used for learning and measuring the predictive accuracy to determine the fitness function of the individual. These methods usually divide the training data into two sets: train-train data and train-test data. Then, the learner uses train-train data for learning and train-test data for evaluating the predictive accuracy. Some methods apply k-fold cross validation instead of dividing data into two sets. They partition training data into k sets. Then, learning and evaluation is repeated k times, each time using $k-1$ sets as training and one set as test, so that each set is used exactly once as test data. Then the average accuracy is calculated as the fitness of the individual. Methods with hypothesis-driven fitness function usually choose a learner with a good performance in terms of speed and accuracy, such as C4.5 [Quinlan, 1993]. According to Kohavi and John [1997], the advantage of using a learner is that, first, features are adapted to the bias of the learner; and second, this approach provides the ability to evaluate features together with original attributes. However, it is not the most appropriate for genetic-based search since it is computationally expensive comparing to other fitness measurements. The fitness is evaluated for each individual in each generation. Therefore, an accurate fitness measure with less computation time is preferable to reduce the execution time of a genetic method.

Entropy-based Measure

Some other methods (for instance, [Hu and Kibler, 1996; Larsen *et al.*, 2002; Otero *et al.*, 2003]) apply entropy-based fitness measure such as information gain or gain ratio [Quinlan, 1993]. Entropy measures the average amount of *bit* of information needed to identify the class of a sample in data [Shannon, 1948]. The entropy of a training data set, T, with c different class labels is measured as follows:

$$Entropy(T) = -\sum_{i=1}^{c} p_i \log_2 p_i \ , \tag{6.1}$$

where p_i is the probability that a sample belongs to the i^{th} class. Entropy identifies the *impurity* of data; a smaller entropy means less information is needed to classify data since data is purer; hence, the data classification is easier. Entropy-based FC methods use entropy to evaluate the effectiveness of constructed features in reducing impurity in data and improving classification. Such methods usually apply information gain, which measures the amount of reduction in entropy given the value of new feature(s) (see [Quinlan, 1993] for details about information gain and gain ratio).

MDL-based Measure

The Minimum Description Length (MDL) Principle [Rissanen, 1983; Grunwald *et al.*, 2005; Grunwald, 2007] has been also used for evaluating constructed features. The MDL Principle was originally described in terms of optimizing a communication problem. A sender has to communicate some data to a receiver. In order to send minimum information, he compresses regularities in data into a “theory” to be communicated. This theory may not cover all data and produce some “exceptions”. So, the sender also has to communicate exceptions to the receiver. He may send a complex theory with few exceptions or a simpler theory with more exceptions. The MDL Principle establishes that:

> the optimal solution is obtained by selecting a theory that minimizes the sum of the code lengths corresponding to *theory* and *exceptions.*

This introduces a trade-off between a very simple theory that produces many errors (exceptions) and a more complex one that accounts for almost all data and makes only a few errors.

The MDL Principle has been applied in several learning methods. For instance, it has been used to control the growth of decision trees [Quinlan and Rivest, 1989] and decision rules [Quinlan, 1993]. In order to apply it to learning, the learning task has to be described as a communication problem. Thus, the learner (sender) has a table of pre-classified training data that needs to be sent to the receiver. As an alternative to sending the whole table, that is, each data sample with its classification, the learner can compress the table into a hypothesis (theory), such as a decision tree, a set of rules or any other form of classifier, and send it to the receiver. Such theory may not be perfect and, hence, make errors when classifying some of the training data. So, to make the communication correct, the errors (exceptions) should also be sent to the receiver along with the theory. The learner has to choose a theory that minimizes the number of bits needed to encode both the theory and errors produced by the theory. Note that according to the MDL Principle, when two theories fit the problem equally (produce same errors), the simpler in terms of theory code length is chosen. Thus, the MDL Principle includes Occam’s razor principle, which states: “*prefer the simplest hypothesis that fits data*”.

The integration of the MDL Principle into the evolutionary approach is not as frequent as it is in other machine learning approaches. Most genetic methods have focused on optimizing a fitness based on classification errors or entropy. An exception is found in DOGMA [Hekanaho, 1997], a genetic-based theory revision system that adjusted the MDL Principle into a fitness measure suitable for evaluating sets of rules as fragments of the theory being revised. Also, there are several examples of using MDL-based fitness for GP systems [Lin and Bhanu, 2005; Zhang and Mühlenbein, 1995; Kazakov, 1997]. In these cases, since fitness is applied to individuals that represent programs or symbolic expressions that may grow unnecessarily large and complex, it becomes important to control their growth by an MDL-based fitness.

Similarly, when a genetic-based approach is used for FC and, so, individuals represent constructed features, the MDL Principle may become necessary to evaluate the complexity of constructed features and their inconsistency with the training data. This is regardless of whether the new features are represented as symbolic algebraic expressions (as in [Muharram and Smith, 2005; Otero *et al.*, 2003; Vafaie and DeJong, 1998]) or non-algebraically (as in MFE2/GA). In both cases, the proposed features correspond naturally to a theory that can grow too large and complex to produce no errors in the training data, and that we may prefer to keep simpler as long as it does not produce too many errors. In spite of this, none of the genetic CI systems integrates the MDL Principle into their fitness function. MFE2/GA is a partial exception since its fitness function measures both the cost of representing data using constructed features and the complexity of features (theory), and the cost of representing inconsistencies in data produced by new features (error) (Section 4.2.2). However, this fitness measurement is not explicitly designed as an approximation to the MDL Principle, as it will be done in Section 6.3.

Although many genetic CI methods use classification error rate as fitness function, this approach is not suitable for a genetic search due to its high computation time, as mentioned above. In addition, some preliminary experiments not reported in this thesis show that an implemented genetic CI method with entropy-based fitness function results in a slightly better accuracy than the same method with a fitness based on classification error rate [Shafti and Pérez, 2003b]. Hence, in the rest of this chapter only the last two forms of fitness measure based on entropy and MDL Principle are evaluated.

6.2 Entropy-based Fitness Function in MFE2/GA$_E$

MFE2/GA is modified to apply a fitness function based on entropy. The new system is called MFE2/GA$_E$. For each individual, the fitness is measured by calculating the entropy of the concept given the values of new features [Quinlan, 1993; Shannon, 1948]. Training data are projected onto the new feature set $\{f_1, \ldots, f_k\}$ and fitness is evaluated by the following formula:

$$Fitness(Ind) = \sum_{i=1}^{2^k} \frac{|T_i|}{|T|} Entropy(T_i), \tag{6.2}$$

where T is set of training data samples and T_i is set of samples whose values for the new attributes f_1 to f_k are equal to the i^{th} tuple in the Cartesian product $f_1 \times \ldots \times f_k$. Recall that there are 2^k tuples since constructed features are binary.

As an example, consider the data set and constructed features of Figure 4.5. The Entropies for each set of samples matching with tuples $T_1 = (0,0)$, $T_2 = (0,1)$, $T_3 = (1,0)$, and $T_4 = (1,1)$ are as follows respectively[1]:

$$\begin{aligned} Entropy(T_1) &= -\tfrac{1}{1} \times \log_2 \tfrac{1}{1} - \tfrac{0}{1} \times \log_2 \tfrac{0}{1} = 0 \ , \\ Entropy(T_2) &= -\tfrac{2}{2} \times \log_2 \tfrac{2}{2} - \tfrac{0}{2} \times \log_2 \tfrac{0}{2} = 0 \ , \\ Entropy(T_3) &= 0 \ ; \ \text{due to no matching data samples} \ , \\ Entropy(T_4) &= -\tfrac{3}{4} \times \log_2 \tfrac{3}{4} - \tfrac{1}{4} \times \log_2 \tfrac{1}{4} = 0.811 \ . \end{aligned}$$

Thus, the fitness of this individual is $\frac{|T_4|}{|T|} Entropy(T_4) = \frac{4}{7} * 0.811 = 0.463$.

To reduce overfitting, part of training data are used for constructing features and all training data are used for entropy-based fitness evaluation. Keeping part of data for fitness evaluation helps GA to construct individuals with smaller functions.

6.3 MDL-based Fitness Function in MFE3/GA

A new fitness function based on MDL Principle is proposed to be incorporated in MFE2/GA to improve its performance. The new system is called MFE3/GA. Before describing how fitness is computed, the notion of *function length* is introduced. As described in Section 4.2.2, each function f_i, defined over subset $S_i = \{X_{i_1} \ldots X_{i_m}\}$, is represented by binary labels of tuples in Cartesian product of attributes in S_i. Thus, each f_i can be represented by $\prod_{j=1}^{m} |X_{i_j}|$ bits, which is referred to as *the length* of function, $len(f_i)$, where m is the number of attributes in S_i, and $|X_{i_j}|$ is the number of values that attribute X_{i_j} can take. Since all constructed functions are defined over proper subsets of S, the longest function f_l is one defined over $S_l = S - \{X_s\}$, where X_s is the attribute that can take fewest values. The length of f_l is $\prod_{i=1, i \neq s}^{N} |X_i|$. To reduce the complexity of constructing functions, the length of each function is limited by a parameter of the system, B. By default the limit is set to 2^B, where $B = 16$, that is, 64 Kbits. In case of binary attributes, this is equivalent to a function defined over 16 attributes. Then, the longest function that can be constructed for a given training data is of length MAXLEN $= min(\prod_{i=1, i \neq s}^{N} |X_i|, 2^B)$. If $\prod_{j=1}^{m} |X_{i_j}| >$ MAXLEN for

[1]In entropy calculation, $0 \times \log_2 0$ is defined to be 0.

a subset S_i, the subset is not considered in FC and fitness evaluation by the system to reduce the complexity, but is considered for genetic operations to produce diversity. This is similar to the technique suggested in Sections 4.2.1 and 4.2.2 when $S_i = S$ or $|S_i| = 1$. In case that none of subsets in an individual are considered in FC, the worst value (a large number) is assigned to the individual as its fitness to force GA to ignore this individual.

The fitness of each individual $Ind = \langle S_1, \ldots, S_k \rangle$ is determined by evaluating the set of corresponding functions $\{f_1, \ldots, f_k\}$ as a theory and measuring two factors based on MDL Principle: the inconsistency of the set with the training data (exceptions produced representing data with the theory) and its complexity (the code needed to represent the theory).

The inconsistency measure drives GA to generate more accurate functions. For measuring the inconsistency of the set of functions with training data, the training data are projected onto the set of constructed features $\{f_1, \ldots, f_k\}$. Then, each tuple in the projection that matches with both positive and negative samples in data is considered as an inconsistent tuple. The inconsistency of the set of functions, $\|E\|$, is measured by the total number of samples that match inconsistent tuples in the projection. This value is normalized by dividing it by the maximum inconsistency, that is, the total number of samples in the training data, M.

The consistency of the individual is not the only factor to drive GA toward its goal. Recall the goal of MFE3/GA is to ease the complex relation among interacting attributes by constructing several functions each representing one complex interaction in the concept. To achieve this goal, the fitness function prefers a consistent individual with several small functions to a consistent individual with few large functions by measuring their complexities based on MDL Principle. The complexity of each individual is determined by the theory code, that is, the sum of length of functions (in bits) defined over subsets in the individual. The complexity factor is normalized by dividing it by its maximum value (code length) that is $K \times \text{MAXLEN}$.

Then, the fitness of the individual is evaluated by the following formula and GA aims to minimize this value:

$$Fitness(Ind) = \frac{\|E\|}{M} + \frac{\sum_{i=1}^{k} len(f_i)}{K \times \text{MAXLEN}} \; . \tag{6.3}$$

Therefore, given two individuals equally consistent with the training data, the fitness function prefers the one with several functions defined over smaller subsets of attributes, rather than one function defined over the union of subsets. To illustrate this, recall individuals in Figure 4.2. Among these individuals, Ind_2 and Ind_3 include subsets of interacting attributes. Thus, functions defined over Ind_2 and Ind_3 are consistent with the given training data. However, since Ind_2 has a subset of eight interacting attributes, more training data are needed to correctly construct the function defined over it. Thus, if few training data are available, the function constructed over this subset overfits training data and will be inconsistent with unseen test data. Ind_3 consists of three subsets, each one containing attributes of one of the three interactions in the concept. Since each subset has four attributes, less data are needed comparing to Ind_2 to construct the corresponding functions properly. Thus, Ind_3 is better (more fit) than Ind_2. Assuming that both individuals are equally consistent with the training data, the fitness function measures the complexity of each individual and prefers Ind_3 with complexity of $2^4 + 2^4 + 2^4 = 48$ to Ind_2 with complexity equal to $2^8 + 2^3 = 264$. Note that the complexity evaluation corresponds to measuring the length of functions and not length of individuals.

As an example, consider the constructed functions in Figure 4.5. For these concepts, there are four samples that are inconsistent with functions (those match with the mixed tuple $(f_1, f_2) = (1, 1)$). Thus, the inconsistency is $\|E\| = 4$. The total length of functions is $8 + 4$. For this data set MAXLEN $= 2^3$, $M = 7$, and by default $K = 5$. Thus, the fitness is calculated as $\frac{8+4}{5\times 2^3} + \frac{4}{7} = 0.871$.

Recall from Section 4.2.4 that MFE2/GA uses 90% of training data for constructing functions and all training data, for fitness evaluation to reduce overfitting. Keeping part of training data for fitness evaluation is not necessary in MFE3/GA because the MDL-based fitness evaluation of this method intends to conduct GA toward less complex individuals and, therefore, reduces overfitting. Hence, MFE3/GA uses all training data for constructing functions and evaluating fitness.

6.4 Experiments

This section empirically compares results obtained by two systems: MFE3/GA with MDL-based fitness, and MFE2/GA_E with entropy-based fitness. Results are also compared with two learners: the standard learner C4.5 (trees and rules) [Quinlan, 1993], and HINT [Zupan *et al.*, 2001] (see Chapter 5 for details about HINT). The first part of these experiments uses synthetic concepts designed for experiments in Section 5.3.2. The second part reports on experiments performed on Braille-detection problem of Section 5.3.3. The MDL-based fitness function of MFE3/GA has improved its performance; and therefore, the new method obtains better results than MFE2/GA. See Appendix C for empirical comparison of the two versions.

6.4.1 Experiments with Synthetic Concepts

The synthetic concepts used as a benchmark for these experiments are composed by several complex interactions. For all concepts, attributes are Boolean except in the last four concepts, where for Monk1 attributes are 2 to 4-valued and for the other three concepts attributes are 3-valued. Table 6.1 gives a summary of these concepts and the experimental results. See Appendix A for a detailed definition of concepts. Columns 2 and 3 show the number of relevant and irrelevant attributes for each concept. The majority class percentage of each concept is given in the forth column.

All experiments are run 20 times independently, each using 5% of all possible instances as training data and the rest as test data. For MFE2/GA$_E$, only part of the 5% training data are used for constructing features and all training data for fitness evaluation using entropy. The previous experimental evaluation showed that on average, MFE2/GA$_E$ achieves higher accuracy when 30% of training data are used for constructing features. So 30% of 5% training data are used for FC and all 5% training data for feature evaluation. Note that doing this is of benefit to MFE2/GA$_E$, and yet, the MDL-based MFE3/GA is believed to be able to outperform it.

Table 6.1 illustrates a summary of the empirical study. The higher of the two average accuracies obtained by C4.5 and C4.5-Rules is reported in column 5. This result is marked by c if obtained by C4.5, or by r if obtained by C4.5-Rules. The average accuracies of HINT, MFE2/GA$_E$, and MFE3/GA are reported in columns 6 to 8 respectively. Columns 9 and 10 show the average number of GA's generations for each genetic method. Numbers between parentheses indicate standard deviations. The highest average accuracy is marked by $\triangleleft$, but only when it is not lower than the majority class percentage. The accuracy of MFE2/GA$_E$ is marked by † when it is significantly better than the accuracy of HINT; MFE3/GA's result is significantly better than those in bold and significantly worse than those in italic (using t-distribution test with $\alpha = 0.02$).

As it can be seen from Table 6.1, the MDL-based fitness function of MFE3/GA guides this method toward better solutions as expected; and therefore, it significantly outperforms MFE2/GA$_E$ for most concepts. MFE2/GA$_E$ in most cases overfits data. It constructs set of features with very small entropy (most of the time with zero entropy), which means the set of features classifies training data perfectly. But when they are evaluated on test data, they produce errors. This is because entropy does not consider the complexity of the theory proposed by the constructed features. It constructs large functions that perfectly match training data and produce overfitting.

Also, comparing the average number of generations of both GA methods illustrates that MDL-based fitness function helps GA to converge to optimal solution faster than the entropy-based method.

Comparing results of MFE2/GA$_E$ and HINT indicates that, although entropy-based FC achieves lower accuracy than MDL-based FC, its overall average accuracy is still better than HINT. This shows the advantage of using GA for FC when concepts are composed by several complex interactions and few training data are available. Even a genetic FC method with not a good fitness function outperforms the greedy FC. Comparing the overall average accuracy of MFE3/GA in Table 6.1 with the one obtained

Table 6.1: Comparing results over synthetic concepts

	Rel.	Irr.	M	Accuracy				No. Generations	
Concept	atts	atts	%	C4.5/R	HINT	MFE2/GA$_E$	MFE3/GA	MFE2/GA$_E$	MFE3/GA
$\wedge(P_{1,4}, P_{3,6})$	6	6	75.0	c**72.5(3.2)**	100(0.0)◁	**98.3(2.1)**	99.8(0.5)	137(35.7)	125(18.4)
$\wedge(P_{1,6}, P_{3,8})$	8	4	75.0	c**73.4(2.7)**	*98.6(6.3)*◁	91.8(6.5)	94.1(2.8)	**219(47.0)**	131(18.1)
$\wedge(P_{1,6}, P_{7,12})$	12	0	75.0	c**72.6(3.9)**	82.5(16.5)	**77.1(6.0)**	89.8(6.8)◁	**230(82.4)**	144(16.0)
$\wedge(P_{1,3}, P_{3,5}, P_{4,6})$	6	6	87.5	c**87.6(1.2)**	**94.1(9.6)**	**96.7(4.9)**	99.8(0.7)◁	130(24.0)	141(27.6)
$\wedge(P_{1,4}, P_{2,5}, P_{3,6})$	6	6	87.5	c**87.5(0.3)**	97.2(7.2)	**96.5(4.6)**	99.6(0.7)◁	153(46.9)	130(29.6)
$\wedge(P_{1,4}, P_{3,6}, P_{5,8})$	8	4	87.5	c**87.5(0.1)**	**90.3(11.0)**	**91.7(5.4)**	98.6(1.7)◁	207(54.0)	173(43.7)
$\wedge(P_{1,4}, P_{5,8}, P_{9,12})$	12	0	87.5	c**87.5(0.1)**	**78.4(4.1)**	**86.4(4.4)**†	92.4(7.2)◁	212(63.0)	199(53.9)
$\wedge(P_{1,6}, P_{2,7}, P_{3,8})$	8	4	87.5	c**86.6(1.8)**	92.3(10.0)	**86.7(4.1)**	93.8(2.4)◁	174(43.4)	169(40.9)
$\wedge(WL3_{1,5}, WL3_{3,7})$	7	5	64.1	r 90.1(3.0)	91.2(11.6)	90.9(3.8)	93.1(5.9)◁	**201(65.7)**	132(28.8)
$\wedge(WL3_{1,5}, WL3_{4,8})$	8	4	68.0	r 86.7(2.0)	88.8(8.9)	89.2(6.1)	89.9(9.6)◁	**230(72.9)**	156(51.8)
$\wedge(WL3_{1,5}, WL3_{5,9})$	9	3	71.5	r**84.9(2.6)**	88.0(10.1)	88.5(6.2)	93.5(7.0)◁	**213(55.4)**	154(37.6)
$\wedge(WL3_{1,5}, WL3_{6,10})$	10	2	75.0	r**82.2(2.1)**	**78.6(5.2)**	83.3(3.5)†	88.1(8.4)◁	**233(80.5)**	167(43.4)
$\wedge(WL3_{1,4}, WL3_{3,6}, WL3_{5,8})$	8	4	57.8	r**89.2(4.1)**	**89.3(12.0)**	**92.9(5.7)**	97.5(2.2)◁	**208(60.2)**	162(50.2)
$\wedge(WL3_{1,4}, WL3_{5,8}, WL3_{9,12})$	12	0	67.5	r**79.5(3.2)**	**71.8(4.5)**	**81.1(6.5)**†	92.3(10.5)◁	**239(60.9)**	177(49.1)
$\wedge(W23_{1,6}, W23_{7,12})$	12	0	70.9	r**68.2(2.3)**	**65.9(3.3)**	**72.8(3.1)**†	83.4(9.3)◁	**215(65.5)**	159(40.1)
$\wedge(W23_{1,4}, W23_{5,8}, W23_{9,12})$	12	0	75.6	r**74.7(1.9)**	**69.7(3.0)**	**80.4(4.3)**†	94.1(9.4)◁	250(75.0)	207(67.3)
$\wedge(W23_{1,5}, W23_{6,10}, W23_{11,15})$	15	0	76.0	r**88.5(3.1)**	98.9(2.8)	**98.5(2.5)**	100(0.0)◁	**228(66.6)**	187(24.2)
$\wedge(W23_{1,6}, W23_{7,12}, W23_{13,18})$	18	0	84.0	r**98.1(0.9)**	100(0.0)◁	**99.5(0.5)**	100(0.0)◁	215(57.0)	200(24.9)
$\wedge(A_{1,4}, A_{5,8}, A_{9,12})$	12	0	82.2	r**89.8(5.0)**	**79.7(3.1)**	**89.1(4.0)**†	97.8(4.3)◁	243(77.0)	225(69.1)
$\wedge(B_{1,4}, B_{5,8}, B_{9,12})$	12	0	87.5	c**86.9(1.3)**	**81.1(2.1)**	88.0(1.3)†	89.6(4.0)◁	231(68.9)	190(70.3)
$\wedge(C_{1,4}, C_{5,8}, C_{9,12})$	12	0	57.8	r**66.2(3.8)**	**64.6(7.8)**	**84.6(16.0)**†	98.5(6.9)◁	**254(75.4)**	170(24.0)
$\wedge(D_{1,4}, D_{5,8}, D_{9,12})$	12	0	87.5	r 90.6(2.8)	**83.7(1.9)**	**89.7(1.4)**†	92.3(3.4)◁	217(77.6)	194(45.2)
$\wedge(E_{1,4}, E_{5,8}, E_{9,12})$	12	0	75.6	r**77.0(3.0)**	**72.2(4.8)**	**81.4(6.8)**†	93.0(10.5)◁	232(62.1)	200(65.7)
$\wedge(A_{1,4}, C_{5,8}, E_{9,12})$	12	0	73.6	r**82.2(3.4)**	**73.7(5.6)**	**84.2(7.4)**†	97.5(6.1)◁	232(77.6)	197(50.8)
$\wedge(A_{1,4}, B_{5,8}, D_{9,12})$	12	0	85.9	r**87.6(3.6)**	**81.5(3.4)**	**88.7(2.7)**†	92.0(4.7)◁	209(65.2)	206(54.3)
$\wedge(A_{1,4}, B_{5,8}, C_{9,12})$	12	0	78.9	r**86.3(3.5)**	**75.8(4.3)**	**87.2(4.2)**†	94.6(7.2)◁	248(57.1)	209(71.8)
$\wedge(B_{1,4}, C_{3,6}, A_{7,10}, D_{9,12})$	12	0	86.5	r**88.5(2.0)**	**83.2(2.8)**	**88.6(1.3)**†	90.8(3.6)◁	195(41.7)	199(48.5)
$\wedge(A_{1,4}, B_{5,8}, C_{9,12}, E_{13,16})$	16	0	87.0	r**94.8(2.1)**	99.8(1.0)	**99.2(1.2)**	100(0.0)◁	214(64.7)	235(35.1)
$\wedge(C_{1,4}, WL3_{5,8}, W23_{9,12})$	12	0	67.8	r**74.2(3.1)**	**70.6(7.2)**	**80.7(7.3)**†	93.7(11.1)◁	**231(68.2)**	178(47.4)
$\wedge(W23_{1,5}, C_{5,8}, WL3_{8,12})$	12	0	76.6	c**76.4(1.2)**	**71.2(2.4)**	**78.1(2.6)**†	84.0(8.7)◁	**219(71.8)**	169(43.1)
$\wedge(W23_{1,5}, C_{4,7}, WL3_{6,10})$	10	2	76.6	r**77.4(2.8)**	**75.9(6.6)**	**80.5(3.4)**†	88.7(8.9)◁	232(53.1)	193(55.0)
Monk1	6	0	50.0	r**65.2(8.3)**	**70.1(13.4)**	**66.0(10.0)**	**80.0(11.7)**◁	108(29.5)	109(21.6)
$\mathtt{palindrome}_6 + 2$	6	2	96.0	c**96.3(0.1)**	**93.2(2.0)**	**97.6(1.8)**†	99.6(0.7)◁	162(60.7)	133(19.4)
$\vee(pal_{1,4}, pal_{3,6}, pal_{5,8})$	8	0	70.0	r 71.2(2.6)	**63.8(4.1)**	70.0(3.6)†	71.4(1.7)◁	**213(55.2)**	138(28.1)
$\vee(pal_{1,4}, pal_{4,7}, pal_{7,10})$	10	0	70.0	r**97.5(2.3)**	100(0.0)◁	**95.7(5.4)**	100(0.0)◁	**228(70.1)**	149(13.0)
Average	10	2	76.6	r 82.2(3.1)	83.3(5.7)	87.2(4.6)†	93.2(5.1)◁	210.3(60.9)	171.6(40.8)

◁ highest accuracy
† significantly better than HINT
c results obtained by C4.5
r results obtained by C4.5-Rules

Boldface: MFE3/GA's result is significantly better than this
Italic: MFE3/GA's result is significantly worse than this

by MFE2/GA in Table 5.3 shows that the MDL-based fitness function improves the accuracy of the method. See Appendix C for detail comparison of the two versions of the method.

6.4.2 Experiments with Real-world Domain

This section reports on empirical comparison of the methods based on a task defined over the real-world domain. The Braille code detection problem of Section 5.3.3 is used and similar experiments to that section are performed.

Recall from Section 5.3.3 that for this concept the parameter K (explained in Sec-

Table 6.2: Comparing results over Braille-detection problem

Data Size	C4.5	C4.5 Rules	HINT	MFE2/GA$_E$	MFE3/GA
1%	90.7(1.9)	*94.8(2.2)*◃	**75.9(4.1)**	**63.6(7.4)**	87.4(8.6)
5%	**97.6(0.4)**	**99.6(0.3)**	**90.2(3.4)**	**96.5(5.0)**	99.8(0.3)◃
10%	**98.6(0.3)**	**99.9(0.2)**	**95.9(3.4)**	**98.2(3.0)**	100.0(0.1)◃
15%	**99.0(0.2)**	**99.9(0.1)**	**99.0(0.8)**	**97.1(4.9)**	100.0(0.0)◃
20%	**99.4(0.1)**	100.0(0.0)	**99.4(0.6)**	**99.4(1.4)**	100.0(0.0)◃

tion 4.2.1) is required to change from 5 to 9 for both MFE2/GA$_E$ and MFE3/GA, allowing them to generate more features.

A total of 20 data sets of 31250 samples were used for experiments and average accuracies over 20 runs are calculated. Experiments were performed increasing training data from 1% to 20% to see how data size affects the performance of the methods. Table 6.2 shows the accuracies of C4.5, C4.5Rules, HINT, MFE2/GA$_E$, and MFE3/GA. MFE3/GA's accuracy is significantly better than those in bold and worse than those in italic (using t-distribution test with $\alpha = 0.02$).

Consider the results corresponding to 1% data in Table 6.2. For this size of training data, all FC methods achieve lower accuracies than C4.5 and C4.5Rules. MFE2/GA$_E$ gets the lowest accuracy comparing to other FC methods because this method uses only 30% of 1% training data for feature generation and overfits data. MFE3/GA overfits data less than other FC methods due to its MDL-based fitness function.

When the number of training data increases all FC methods take the advantage of training data size and improve their accuracies. However, MFE3/GA is the only FC method in the table that gets higher accuracy than C4.5 and C4.5Rules. It significantly outperforms all other methods except for 20% data when both MFE3/GA and C4.5Rules get 100% accuracy. Comparing the results obtained by MFE2/GA$_E$ when 15% and 20% training data size are used, shows that when more data are provided, this method overfits data and achieves lower accuracy.

6.5 Experimental Results on UCI Benchmarks

Sections 6.2 and 6.3 presented two fitness evaluations integrated into MFE2/GA$_E$ and MFE3/GA respectively; and, experiments in Section 6.4 showed the better performance of MFE3/GA. Now that an appropriate fitness is designed, this section evaluates the new method in real-world applications using the UC Irvine machine learning benchmarks [Asuncion and Newman, 2007; Blake and Merz, 1998]. Note that the difficulty of most problems in Irvine databases is noise and lack of relevant attributes instead of complex attribute interaction. However, these sources of difficulties are outside of the scope of this thesis. For experiments in this section, Irvine databases which are expected to be closer to the learning problem addressed in this research are selected.

The first learning problem is SPECT Heart data set which contains information related to the cardiac Single Proton Emission Computed Tomography (SPECT) images of patients and the cardiologist's diagnoses. The concept consists of 22 binary attributes that summarize the SPECT's images. Each case is classified into two categories: normal and abnormal. The data set has 267 cases, which is divided into two sets: the training data with 80 cases and the test data with 187 cases.

The second problem is the German Credit data set. This data set contains information about the status of persons' accounts and their classification as good or bad credit risk. Each case is described by 20 attributes: 7 continuous and 13 nominal. All continuous attributes are discretized to nominal attributes before performing experiments. Since the data set is not partitioned into training and test, experiments are run 20 times, each time data is shuffled and 90% of samples are used for training and the rest are used for testing. Then, the average accuracy is calculated.

The next two data sets are Monks problems which are well-known artificial problems used to benchmark machine learning methods. These concepts are defined on a domain of 6 nominal attributes. Monk1 has an underlying concept $x_1 = x_2 \wedge x_5 = 1$ with 124 training samples and 432 test samples. Monk2 concept is "*exactly two of six attributes are equal to one*" with 169 training and 432 test samples. Monk3 problem has noise and, therefore, is not used for these experiments. Recall that Monk1 was also used for experiments in Sections 5.3.2 and 6.4.1.

The last Irvine concept used for experiments is Poker Hand data set. Each sample in this data set is a set of five playing cards drawn from a standard deck of 52. Each card is described using two attributes, rank (with 13 nominal values) and suit (with 4 nominal values). Thus, 10 nominal attributes are used for describing each sample. The attribute x_{2k-1} represents the suit of card k and the attribute x_{2k} represents the rank of card k for $1 \leq k \leq 5$ (i.e., attributes x_1, x_3, x_5, x_7, x_9 are suit attributes and $x_2, x_4, x_6, x_8, x_{10}$ are rank attributes). The training data contains 25010 samples and the test data is 1,000,000 samples. Note that the complete domain of this concept contains 311,875,200 samples. Thus, the proportion of data available for learning is very small. The original data set in Irvine database has 10 class labels classifying samples into Nothing in hand, One pair, Two pairs, Three of a kind, Straight, Flush, Full house, Four of a kind, Straight flush, and Royal flush. Since MFE3/GA assumes that the class labels are binary, the concept is converted to a binary class concept determining whether a sample is a valuable set of poker hand (i.e., One pair, Two pairs, Three of a kind, Straight, Flush, Full house, Four of a kind, Straight flush, or Royal flush) or not (i.e., Nothing in Hand).

Table 6.3 shows the results of running C4.5, C4.5Rules, HINT and MFE3/GA for each concept. Since the underlying interactions in these concepts were not clear, the parameter K (explained in Section 4.2.1) is increased from 5 to 12 for MFE3/GA, allowing this method to generate more features. Default parameters are used for other methods. The average result for each method is reported in the last row.

Table 6.3: Comparing results over UCI benchmarks

Concept	C4.5	C4.5 Rules	HINT	MFE3/GA
SPECT Heart	66.8	77.0	74.3	73.8
German Credit	71.8	71.1	59.0	68.4
Monk1	75.7	100	100	100
Monk2	65.0	66.2	100	78.9
Poker Hand	62.8	99.4	50.2	99.6
Average	68.4	82.7	76.7	84.1

As it can be seen from Table 6.3, MFE3/GA improves the accuracy of C4.5 and achieves similar result as HINT on SPECT Heart; but, it outperforms HINT on German Credit. However, the accuracy of both HINT and MFE3/GA is lower than C4.5Rules on SPECT Heart and German Credit concepts. Both CI methods fail to improve learning accuracy over these two concepts. These concepts are expected to contain complex interactions; however, there are other sources of difficulties which make them harder for the CI methods. Thus, neither HINT nor MFE3/GA facilitate learning these concepts. The existence of noise in data might be a source of difficulty for these methods. Note that the SPECT Heart data set available in Irvine was obtained by processing images and extracting 44 continuous attributes which were then preprocessed to obtain 22 Boolean attributes. These preprocessing steps may have added noise to data or eliminated some important information. Also, discretization of attributes performed on German Credit for these experiments might have removed relevant information. Note that in spite of these difficulties, the accuracy of MFE3/GA is similar to or better than HINT on these two concepts.

The result on Monk1 shows that this concept is an easy one for most methods. The main reason is that enough training data were provided for all methods. Recall from Sections 5.3.2 and 6.4.1 that with smaller data size the differences among methods are clearer and MFE3/GA significantly outperforms other methods, which shows the advantage of this method when few training samples are provided.

MFE3/GA improves learning Monk2. However, HINT and other CI methods [Thrun *et al.*, 1991] achieve better results for this concept. The reason is that this concept consists of one complex interaction over all given attributes. MFE3/GA assumes that the concept consists of several interactions. Thus, it establishes the limitation of not permitting the construction of a function over all attributes. It looks for smaller functions over subsets of original attributes. However, in spite of this limitation, MFE3/GA still achieves better accuracy than C4.5 and C4.5Rules on Monk2. It tries to highlight the interaction in this concept by constructing five features defined over subsets $\{x_1, x_2, x_5\}$, $\{x_1, x_3, x_4, x_5, x_6\}$, $\{x_1, x_2, x_3, x_6\}$, $\{x_2, x_3, x_4, x_5\}$, and $\{x_1, x_2, x_4, x_6\}$. These features outline the interaction among attributes in each subset; but, they are not enough to highlight the whole interaction in the concept.

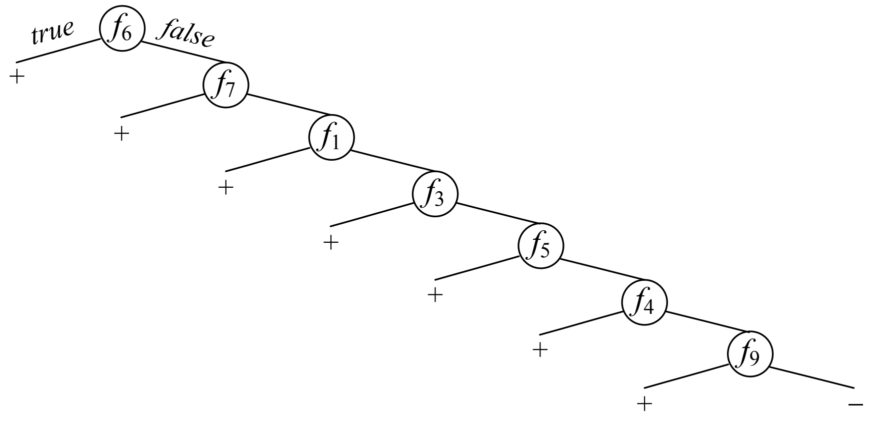


Figure 6.1: Decision tree constructed by C4.5 for Poker Hand after FC

The result on Poker Hand is quite different. For this concept HINT achieves the lowest accuracy and MFE3/GA the highest. This concept is a good example of real world to illustrate the need for a system like MFE3/GA. The concept consists of several complex interactions which are difficult to discover if not enough data samples are provided. C4.5 generates a pruned tree of 4822 nodes giving 62.8% accuracy. C4.5Rules improves accuracy to 99.4%. However, it generates a large set of rules (250 rules) which are hard to interpret. HINT due to its greedy search finds a local optimal solution and overfits training data resulting in 50.2% accuracy. MFE3/GA achieves the highest accuracy by capturing and compacting interactions into few features. This method successfully discovers the interactions by means of the global search, non-algebraic representation, and multi-feature construction.

A deep analysis of the features constructed by MFE3/GA for Poker Hand concept is important for this research. MFE3/GA constructs nine features, $\{f_1, f_2, \ldots, f_9\}$, to highlight interactions in the concept. After adding these features to the original attribute set and updating training and test data, C4.5 generates an unpruned decision tree of 376 nodes with constructed features near to the root and original attributes close to leaves. Then, pruning in C4.5 reduces the size of the tree to 15 nodes using 7 out of 9 constructed features (ignoring f_2 and f_8) and none of the original attributes, as shown in Figure 6.1. The pruned tree, which is equivalent to the disjunction of the seven features, classifies the test data with 99.6% accuracy.

It is challenging to analyze the seven features constructed by MFE3/GA and see how they help learning the Poker Hand concept. Features are defined over two or three attributes in $\{x_2, x_4, x_6, x_8, x_{10}\}$ (rank attributes) except the last feature, f_9, which is defined over $\{x_1, x_3, x_5, x_7, x_9\}$ (suit attributes). Recall that features in MFE3/GA are represented non-algebraically (Section 4.2.1). Each feature is represented by a vector of values that shows the outcome of the function for each combination of attribute values. Most of the features could easily be analyzed only by studying their vector of values. However, when the feature contains a relation among many attributes, its vector is long and not easy to interpret. C4.5 is used as a tool to interpret these long vectors. For each vector a complete data set (i.e., a set of attribute values and their corresponding outcome of the function) is generated and given to C4.5 for learning. The tree produced by C4.5 for each data set clearly shows the interaction encapsulated into each feature. Note that this process is not a learning process since all the data set representing the function is available and is given to C4.5. This process is performed only for the interpretation of long functions.

Among the seven features, f_9 is the only function that is defined over five suit attributes. This function returns true when $x_1 = x_3 = x_5 = x_7 = x_9$. So, it represents a hand which contains five cards of the same suit. This function perfectly abstracts and encapsulates the interaction Flush (a hand of same suit).

The other six functions represent relations among two or three rank attributes. They jointly abstract seven poker hands: One pair, Two pairs, Three of a kind, Full house, Four of a kind, Straight flush, and Royal flush, which are interactions among rank attributes. Features f_1, f_3, f_4, and f_5 are defined over subsets of two rank attributes, $S_1 = \{x_2, x_{10}\}$, $S_3 = \{x_6, x_{10}\}$, $S_4 = \{x_2, x_4\}$, and $S_5 = \{x_4, x_6\}$, respectively. They are true if the two attributes in the subset have the same value, that is two cards with the same rank. Features f_6 and f_7 are defined over subsets of three rank attributes, $S_6 = \{x_4, x_8, x_{10}\}$ and $S_7 = \{x_2, x_6, x_8\}$. The feature f_6 is true if at least two of the three attributes have the same value, that is two or three cards with the same rank. The feature f_7 is constructed with two errors in the non-algebraic representation due to the small size of Poker Hand training data. This function is true if at least two of the three attributes in $\{x_2, x_6, x_8\}$ have the same value or if $x_2 = 11$, $x_6 = 3$, and $x_8 = 0$, or if $x_2 = 12$, $x_6 = 8$, and $x_8 = 2$. The last two conditions are errors in f_7 which cause misclassification of 572 cases of 1,000,000 test data that is less than 0.06% error.

Figure 6.2 illustrates the relations among attributes represented by the six features. These features together contain the relation between any pair of attributes (shown by lines connecting pairs of vertices in the pentagon). The interesting point is that the relation between each pair is not contained in more than one feature (i.e., there is only one line connecting each pair of vertices). This implies that the relation between any pair of interacting rank attributes is contained in the constructed features and there is no repeated information encapsulated in the features. MFE3/GA perfectly compacts the interactions between rank attributes into six features, while the interaction among suit attributes is abstracted in f_9.

It is interesting to note that the error occurred in the construction of f_7 produces only 572 misclassification out of 1,000,000 test samples. Recall from Section 4.2.1 that each feature is represented by a vector of values or labels, each label showing the outcome of the constructed function for a combination of attribute values. An error in each label misclassifies only cases that match with the combination of attribute values

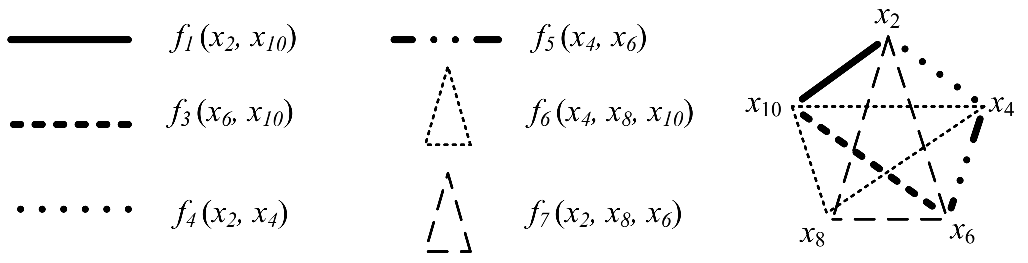


Figure 6.2: Relations between attributes represented by constructed features

corresponding to the incorrect label. Two labels in the vector representing f_7 were induced incorrectly; and only 572 cases in test data match with the attribute values corresponding to these two labels. If instead of the non-algebraic representation, an algebraic representation is applied for expressing features, an error in feature construction happens when an incorrect operator or operand is selected in an algebraic term in the feature. This error may produce a larger amount of misclassification comparing to an error in non-algebraic representation since usually more cases are covered by each algebraic term in the constructed features. This shows the advantage of non-algebraic representation.

Note that in addition to 572 misclassified cases of the test data due to the error in f_7, there are 93 cases in the training data and other 3885 cases in the test data, which are also classified incorrectly. These cases correspond to Poker hands containing five cards of sequential rank (Straight hand). MFE3/GA fails to capture this interaction. In order to encapsulate this interaction a feature defined over five attributes, each of 13 values (i.e., rank attributes), is required; that is, a feature of length $13^5 = 371,293$. Recall from Section 6.3 that the length of each feature is limited to 2^B, where $B = 16$, MFE3/GA cannot construct such feature to represent the interaction Straight. For this reason, it cannot achieve an accuracy higher than 99.6% (i.e., $\frac{3885}{1,000,000}$ error) on Poker Hand.

The analysis of the constructed features and the decision tree of Figure 6.1 illustrate how MFE3/GA can capture and highlight interactions to the learner. MFE3/GA by means of constructing 7 features helped C4.5 to generate a decision tree of only 15 nodes achieving 99.6% accuracy. Although, C4.5Rules's accuracy is close to MFE3/GA, the set of 250 rules generated by C4.5Rules are not easy to comprehend. MFE3/GA compacts the underlying relations in the concept into highly informative features. The features constructed by MFE3/GA and the resulting decision tree are easily interpretable and help the expert to understand the underlying complex interactions in the concept. Note that two factors, existence of several complex interactions and the small size of training data made the Poker Hand concept hard for HINT due to its greedy strategy for searching interacting attributes and constructing features.

6.6 Conclusion

In Chapter 4, the accuracy advantage of the MFE2/GA approach was related to the structure of the individuals in the GA population. Each individual provides a collection of new features intended to change the representation of data, in a way that highlights the underlying complex attribute interactions and, hence, simplifies learning. However, the fitness function of MFE2/GA sometimes does not guide GA to converge to the optimal solution. The purpose of Chapter 6 was to improve fitness function of the method. Different types of fitness functions are reviewed and two of them are suggested for integration into the system.

First, a fitness function based on the entropy notion is designed in a new system called MFE2/GA$_E$. This fitness evaluates the effectiveness of each individual, as a set of constructed features, in reducing impurity of data and improving classification. Second, the use of the MDL Principle is proposed for evaluating the fitness of each individual. The new MDL-based fitness function implemented in the MFE3/GA method includes two terms: one that approximates the complexity of the collection of new features (theory), and a second one that accounts for the misclassifications produced by those features (errors).

To assess the advantages introduced by new fitness functions, an empirical study is performed using a benchmark of synthetic concepts designed to involve several combinations of complex attribute interactions. The study shows that the proposed MDL-based fitness yields significantly better predictive learning accuracy than other fitness solely based on entropy. Moreover, the MDL-based fitness helps GA to converge to optimal solution faster than the entropy-based fitness. Comparing the results obtained by MFE3/GA in this chapter with those obtained by MFE2/GA in Chapter 4 and Chapter 5 shows that the MDL-based fitness function improves the accuracy of the method (see Appendix C for detailed empirical comparison). In addition, our empirical results show that even without the improvement of an MDL-based fitness, the MFE2/GA$_E$ approach with an entropy-based fitness measure retains most of its accuracy advantage over two relevant learners: a standard learner as C4.5 (trees and rules), and HINT, a non-GA CI method that, like MFE2/GA, uses non-algebraic representation for constructed features. Similar empirical results were found using Braille code detection problem defined over a real-world domain.

Finally, MFE3/GA is evaluated by performing experiments over some machine learning benchmarks from UC Irvine databases. The results show that when complex interactions among attributes exist this method successfully abstracts and encapsulates interactions into constructed features to highlight them to the learner.

Chapter 7

Conclusions and Future Work

This chapter summarizes the important achievements of the research and describes some possible directions for future work. Section 7.1 provides a summary of the dissertation; Section 7.2 describes the contribution of the research; and finally, Section 7.3 discusses the limitations and proposes possible ways in which the work can be further extended.

7.1 Summary of the dissertation

Complex interactions and Constructive Induction (CI): This research focused on the problem of learning in presence of complex attribute interactions in concepts. The low-level primitive representation of real-world data facilitates the existence of interactions. Chapter 1 explained how attribute interaction complicates learning. When complex interactions among attributes exist, the importance of each interacting attribute is masked to the learner; moreover, even if important attributes are identified by the learner, regularities are still hard to discover (Section 1.1). CI has been introduced to ease the problem of learning hard concepts. When CI is applied to concepts with complex interactions, FC in CI aims to capture and encapsulate interactions into new features to outline regularities to the learner (Section 1.2).

Greedy CI and genetic CI: In spite of many progresses in CI, current CI methods still have difficulties when learning concepts with complex interactions. Chapter 2 briefly reviewed related works and studied the requirements for a CI method when applied to such concepts. It classified CI methods into two groups. The first group consists of greedy CI methods, which apply a greedy local search to find interacting attributes and construct features (Section 2.1). The second group includes genetic CI methods which are those that apply a global search such as GA or GP (Section 2.3).

Greedy CI – Weaknesses: Three main weaknesses of current greedy CI methods were highlighted in Section 2.1, which are as follows. First, these methods apply a greedy local search to find interacting attributes (Section 2.1.1). The search space for finding interacting attributes is large and with high variation. Thus, a local search may find a local optimal solution. Second, most CI methods apply a greedy strategy to construct and evaluate features one by one (Section 2.1.2). So, the construction of each feature depends on those previously constructed. When several complex interactions exist in the target concept, a CI method may construct an incorrect feature in a primary step and mistakenly evaluate it as a correct feature. Therefore, all subsequent features constructed using this feature will be irrelevant. Third, most CI methods apply an algebraic language to represent new features by using algebraic operators (Section 2.1.3). A complex algebraic feature is required to encapsulate a complex interaction. Moreover, if no prior knowledge is provided about the concept, it is difficult to define algebraic operators.

CI requirements: Considering these weaknesses, three requirements for a CI method were specified. These requirements are as follows. First, a CI method needs a global search to skip local optimal solutions and find the global optimal solution (the optimal subset of interacting attributes). Section 2.2 presented GA and GP as global search techniques which can be applied for CI. Second, a CI method needs to construct and evaluate several features together to facilitate learning in presence of several complex interactions in the concept. A search method based on GA or GP permits constructing and evaluating several features together as a single genetic individual. Third, a CI method requires a proper representation language when interactions are complex and no prior information is provided about the concept. Section 2.1.3 introduced the notion of non-algebraic operator-free representation as a form of representing features, which is preferred to algebraic form when no prior knowledge is available to define operators. Non-algebraic representation permits extracting features directly from data, which is necessary if the training data is the only information provided about the concept. This form of representation also reduces the size of search space and the difficulty of constructing complex features.

Genetic CI – Weaknesses: Section 2.3 reviewed relevant genetic CI methods and outlined their deficiencies. The major problem of these methods is the use of algebraic form such as parse trees to represent features, which produces difficulties, as explained in Section 2.1.3. The other problem of most genetic methods is the evaluation of constructed features by means of a hypothesis-driven fitness evaluation; that is, a hypothesis generated by a learner is used as the fitness evaluation to guide the search toward the optimal solution. The success of these methods strongly depends on the learner. Moreover, using a learner for fitness evaluation slows down the performance of the method. Thus, a data-driven fitness evaluation is preferred to a hypothesis-driven one. Another deficiency of some of genetic methods is that they do not exploit GA or GP to construct and evaluate several features together as a genetic individual. In spite of using a genetic search, their strategy for constructing features is, still, greedy and one by one.

A CI framework – Decomposition: Considering the deficiencies of reviewed methods and the requirements for a CI, Section 2.4 introduced a framework for designing a CI method. This framework simplifies the CI task of searching the huge and complex search space. A CI method needs to search the space of functions defined over subsets of attributes. This space grows exponentially with the number of attributes and has high variation. The proposed framework suggests partitioning this space into two spaces: the space of attribute subsets and the space of functions defined over a given attribute subset. Thus, the task of CI is divided into two smaller tasks: searching for subsets of interacting attributes and finding a function that represents interactions among attributes in a given subset. GA is used for the first task. For each attribute subset produced by genetic operators a function is constructed by analyzing data and inducing a relation among attributes in the subset and the target concept. This relation is represented non-algebraically using a vector of values that specifies the outcome of the function for each combination of attribute values. A data-driven measurement evaluates the utility of the function. If the function represents a promising relation among given attributes, the attribute subset is considered a good subset in GA. Thus, generated subsets are used to induce functions from data; and, functions are used to evaluate subsets and guide GA toward better subsets. By this strategy the dependency between the two tasks is maintained while improving each of them.

DCI – The first instance of the framework: The rest of the dissertation focused on two new methods designed to evaluate the framework. Chapter 3 introduced DCI. This CI method was designed to analyze the utility of global search and non-algebraic feature representation when complex interaction exists in the target concept. Following the framework proposed in Chapter 2, this method applies GA to find interacting attributes. For each attribute subset as a genetic individual, DCI analyzes data to induce a non-algebraic function. The fitness of each individual is determined by evaluating the constructed feature. An entropy-based fitness evaluation, that is a data-driven measurement, was presented. It measures the amount of uncertainty introduced by constructed feature and its complexity. Genetic operators are performed on attribute subsets to generate better ones. Since each attribute subset is associated with an induced function, when subsets are changed by genetic operators, functions are changed too. Thus, GA indirectly evolves functions to construct better ones. This strategy allows DCI to decompose the search space and divide the main task of CI into two, while maintaining the effect of each one on the other. DCI meets the first and third requirements specified in Chapter 2.

DCI evaluation – Advantages: Chapter 3, then, explained experiments conducted to evaluate the use of GA and non-algebraic feature representation in DCI. Empirical results showed three important points. First, when complex interactions exist among attributes feature selection is not enough for improving accuracy of a learner; features that highlight regularities are needed to be constructed to ease learning. Features constructed by DCI significantly improve the accuracy of a standard learner such as C4.5 when a complex interaction exists among attributes. Second, when concept consists of complex interactions, non-algebraic representation helps a CI method to construct more promising features and achieve better results comparing to CI methods with algebraic representation. Third, the global search in DCI successfully finds the subset of interacting attributes and its corresponding function when concepts are composed of one complex interaction.

DCI evaluation – Weaknesses: The experiments showed a requirement for a CI method which was specified earlier in Chapter 2 but is not considered in DCI, that is the need for constructing and evaluating several features. DCI intends to abstract all interactions into one feature. Experiments showed that when number of interacting attributes is high, in spite of correctly detecting relevant attributes, DCI cannot construct a proper function that represents all interactions.

MFE2/GA – Overcome deficiencies of DCI: Chapter 4 explained the need for constructing more than one feature. When several interactions exist in concept, a function that encapsulates them tends to be complex and difficult to construct. This difficulty augments when few training data samples are provided. A CI method needs to break down the function into several smaller ones. Then, functions form a set of related parts of a theory that represents interactions. Each function by itself may not be evaluated properly; functions should be evaluated together as a set of characteristics. Section 4.2 introduced MFE2/GA, the second instance of the proposed framework. This method, while maintaining all the advantage of DCI, permits constructing and evaluating several features at the same time. MFE2/GA takes advantage of GA to construct and evaluate several features together. Each individual in MFE2/GA represents a set of attribute subsets; each subset is associated with a function that is induced from data. The aim of GA is to find the best set of attribute subsets and the best set of functions. The data-driven fitness measurement in MFE2/GA evaluates the set of constructed functions together as a theory (Section 4.2.2). Special genetic operators are designed to allow producing both new attribute subsets and new sets of attribute subsets with their corresponding functions (Section 4.2.3). These operators are empirically analyzed to show their importance for GA convergence to optimal solution (Section 4.3).

MFE2/GA – Empirical Evaluation: MFE2/GA fulfills all the requirements for a CI method specified in Chapter 2. Empirical evaluation showed that the multi-feature construction and evaluation helps MFE2/GA to outperform other relevant methods when several complex interactions among attributes exist (Section 4.4). The experiments outlined the importance of constructing and evaluating several features together as parts of a theory. These experiments illustrated that MFE2/GA successfully finds the set of subsets of interacting attributes and constructs a set of functions representing complex interactions. Therefore, it outperforms DCI and other relevant methods when concept consists of several complex interactions.

MFE2/GA – Analyzing its sensitivity to the training data: The MFE2/GA's flexibility in decomposing and searching a large and complex space of functions makes this method dependent on the quality of data. MFE2/GA is a data-based CI method. It needs to see the replication of the structure of the function in data in order to extract it properly from data. Chapter 5 aimed to evaluate the sensitivity of MFE2/GA to the training data size by comparing the method with HINT, a relevant data-based CI method. Both MFE2/GA and HINT are strongly dependent on the training data size. The designs of the two methods are theoretically studied (Section 5.2) and empirically compared (Section 5.3). Both methods have some important functionalities. The multi-value feature construction of HINT (Section 5.1) allows this method to break down a complex interaction into smaller ones. Thus, it needs less training data to construct functions when concept consists of a high-order complex interaction. However, if the concept consists of several low-order complex interactions this property of HINT may cause overfitting. The multi-feature construction of MFE2/GA along with its global search help this method to exploit data better than HINT. These properties of MFE2/GA are very important for its success when several complex interactions exist among attributes and few training data are provided.

MFE3/GA – Improving MFE2/GA with an MDL-based fitness: Experiments in Chapter 4 indicated that the GA in MFE2/GA is not always guided properly toward the optimal solution; therefore, MFE2/GA sometimes fails to find interacting attributes. Chapter 6 analyzed different types of fitness measurement and suggested the use of two of them with the aim of improving MFE2/GA's accuracy. The first fitness measure was designed based on entropy notion and integrated to the system, resulting a new version called MFE2$_E$/GA. This measure evaluates the goodness of the set of constructed features in reducing impurity in data and improving classification. The second measure is designed based on MDL Principle. It measures the sum of two values: the complexity of constructed features (theory) and the amount of misclassification produced using new features (errors). This measure is integrated into a new version of the method called MFE3/GA. An empirical analysis was performed to compare and evaluate both fitness measures. The results indicate that the MDL-based fitness guides GA significantly better than the entropy-based fitness. In addition, the MDL-based fitness helps GA to converge to the solution faster than the other one. This measure also improves the accuracy of the method comparing with the results obtained by MFE2/GA in Chapters 4 and 5.

MFE3/GA – Final evaluation on machine learning benchmarks: For final evaluation of MFE3/GA, some experiments over UC Irvine machine learning benchmarks were performed and explained in Section 6.5. These experiments showed that the new method can successfully be applied to real-world domains to abstract and highlight relations among attributes and ease learning task despite complex interactions in concepts.

7.2 Contribution of the Research

This research contributes to the fields of machine learning, constructive induction, feature selection, feature construction, and genetic and evolutionary computation. The work aimed to ease the learning task when the low-level primitive representation of real-world concepts produces complex interactions among attributes. Concepts with complex interactions pose a problem to learners. Current CI methods cannot improve learning in presence of several complex interactions when the only information available is the training data set. The research answered two important questions:

1. What are the requirements for a CI method to improve learning such concepts?
2. How these requirements can be met?

Three important requirements are specified for a CI:

1. A global search to find subsets of interacting attributes and functions that represent interactions in each subset
2. A proper representation language for abstracting and expressing interactions
3. A strategy that permits construction and evaluation of several functions together

The research discussed that previously introduced CI methods do not fulfill all these requirements; and, if a method satisfies these requirements, facilitates learning such complex concepts when the only information provided is training data.

In order to meet these requirements, the work aimed to simplify the huge and complex search space needed to be explored for constructing appropriate features. A new framework is designed to decompose the search space into two smaller spaces, while reflecting the effect of each one on the other one. The framework uses GA as a global search to find subsets of interacting attributes and induces functions directly from training data. This framework allows searching the optimal set of subsets of interacting attributes and the optimal set of functions that represent interactions, fulfilling the first and third requirements specified for a CI method. The research also intended to ease FC by defining a proper language for representing interactions. The notion of non-algebraic (operator-free) representation language is introduced and used in the framework. It was shown that when no prior knowledge is available about the domain this form of feature representation facilitates constructing complex interactions and is more appropriate than the algebraic feature representation traditionally used by other CI methods. The proposed framework can be used for designing a new machine learning application or a new tool to be integrated into a machine learning system.

Two experimental methods, DCI and MFE2/GA are designed to evaluate the utility of the framework for machine learning. DCI distinguishes interacting attributes from irrelevant attributes and defines a function over subset of interacting attributes to abstract and encapsulate interactions into a new feature. MFE2/GA in addition to detecting interacting attributes, groups them into subsets, each containing attributes that participate in one complex interaction. It also simultaneously constructs and evaluates several functions together, each defined over one subset.

MFE2/GA meets all the specified requirements and therefore differs from other CI methods. Experimental analysis showed that this method facilitates the learning task more than the others when concept consists of several complex interactions and no prior information is provided about the concept.

Proposing this CI method raised a new question to answer that is how sensitive MFE2/GA is to the training data size. The method is a data-based CI and its performance depends strongly on the training data. Theoretical and empirical analysis showed that the design of the method makes it less sensitive to data comparing to a relevant data-based CI method.

Finally the research proposed a new version of the method, MFE3/GA. This method has the advantage of using an MDL-based measure for evaluating constructed features more accurately.

The result of the research is a multi-feature construction method based on GA and non-algebraic feature representation that eases learning concepts with several complex interactions when few data samples are available and no expert knowledge is provided. This method can be integrated into machine learning systems to facilitate learning under the specified conditions. However, like any other method, MFE3/GA has its limitations and cannot be applied to all learning problems, as explained in the next section.

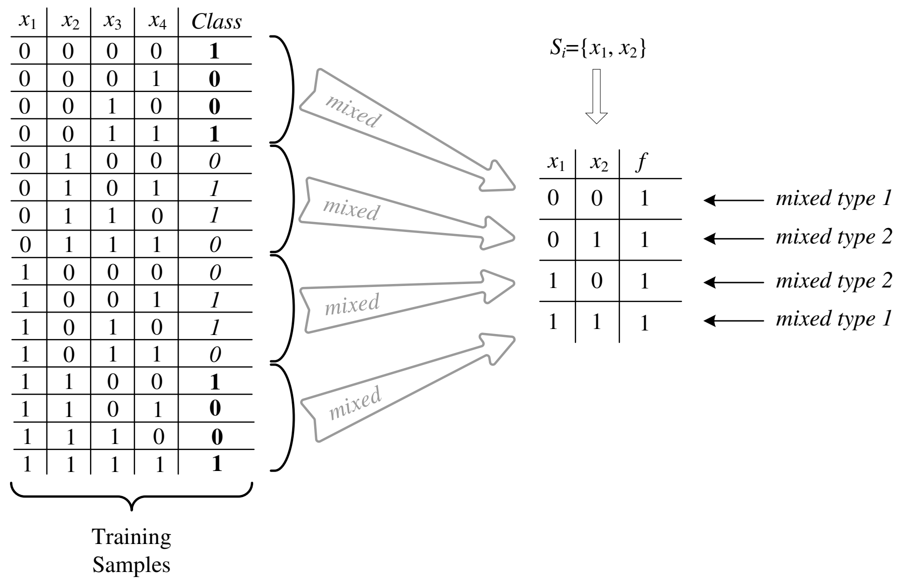
7.3 Limitations and Future Work

This thesis illustrated that a CI method that satisfies the specified requirements facilitates learning despite complex interactions. Nevertheless, MFE3/GA like any other method has some limitations and cannot be applied to all problems. This section highlights MFE3/GA's limitations and suggests approaches to overcome them. Note that these limitations relate to MFE3/GA and not the proposed CI framework and CI requirements. This section also discusses some directions for future research.

Decomposing a Complex Interaction

This research illustrated that MFE3/GA successfully facilitates learning in presence of several complex interactions by decomposing the concept into a set of functions, each representing one complex interaction. However, MFE3/GA cannot further decompose each complex interaction into smaller parts. As illustrated in Sections 4.4.2 and 5.3.1, when the number of attributes involved in the complex interaction grows, the FC task in MFE3/GA becomes more difficult. Therefore, its accuracy degrades. Recall from Section 4.2.1 that MFE3/GA induces a function by classifying tuples in the Cartesian product of attribute values into pure, unknown, and mixed tuples. If a complex interaction is broken down into smaller parts, all the tuples will be classified as mixed. However, there are different mixed tuples which can be treated differently. To illustrate this, consider the concept $P_{1,4} \overset{\text{def}}{=} (x_1 \oplus x_2) \oplus (x_3 \oplus x_4)$ that consists of a complex interaction of four attributes (parity of four). This interaction can be broken down into two smaller ones, $x_1 \oplus x_2$ and $x_3 \oplus x_4$. However, MFE3/GA cannot discover these smaller interactions. It projects data onto subset $S_i = \{x_1, x_2\}$, classifies all the tuples in the Cartesian product of attributes in S_i as mixed tuples, and assigns the same label to them, as illustrated in Figure 7.1, resulting in an irrelevant function $f = \mathbb{T}$ (i.e., "always true").

The careful analysis of data in Figure 7.1 shows that there are dissimilarities among tuples which can be considered for classifying them. Projecting data onto S_i grouped training samples into four sets resulting in four mixed tuples. Considering data class labels, it can be seen that the first and forth group of samples have the same pattern of class labels $\langle 1, 0, 0, 1\rangle$ (boldface labels in Figure 7.1), and the second and third group follow another pattern, $\langle 0, 1, 1, 0\rangle$ (italic labels in the figure). This information can be used to further classify mixed tuples into two subgroups (mixed tuples type 1 and mixed tuples type 2 in the figure), assigning a different label to each. MFE3/GA cannot distinguish different mixed tuples to assign different labels to them. Thus, it tries to capture the whole interaction at once by constructing a function over all interacting attributes, which is more complex to construct. This limitation may cause difficulty for MFE3/GA if the number of attributes involved in the complex interaction is high and few training data samples are provided (as experiments in Section 5.3.1 showed). For such concepts



x_1	x_2	x_3	x_4	*Class*
0	0	0	0	**1**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	*0*
0	1	0	1	*1*
0	1	1	0	*1*
0	1	1	1	*0*
1	0	0	0	*0*
1	0	0	1	*1*
1	0	1	0	*1*
1	0	1	1	*0*
1	1	0	0	**1**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**1**

x_1	x_2	f
0	0	1
0	1	1
1	0	1
1	1	1

Figure 7.1: High-order complex interaction and MFE3/GA

a strategy that allows classifying mixed tuples would be useful. HINT overcomes this limitation by means of multi-coloring property (Section 5.1), assigning different labels to mixed tuples. It considers the rest of attributes, x_3 and x_4, to determine compatible tuples of attribute values in $S_i = \{x_1, x_2\}$ and assign same label to them, as shown in italic and boldface in Figure 7.2.

It would be challenging to integrate the multi-coloring of HINT into multi-feature construction of MFE3/GA. However, if there are other attributes in the concept in addition to those participating in the complex interaction, the task would be more complex. HINT partitions the original attribute set into a subset and its complementary in order to label tuples of attribute values in the subset. For example, if the concept is $\wedge(P_{1,4}, P_{5,8}) \overset{\text{def}}{=} [(x_1 \oplus x_2) \oplus (x_3 \oplus x_4)] \wedge [(x_5 \oplus x_6) \oplus (x_7 \oplus x_8)]$, HINT partitions the original attribute set into $S_i = \{x_1, x_2\}$ and $S_i^c = \{x_3, x_4, x_5, x_6, x_7, x_8\}$ and labels the tuples in S_i by considering the value of all the attributes in S_i^c. But, considering only he value of two attributes, x_3 and x_4, is enough for labeling tuples in S_i. It would be easier to construct functions for this concept if the method could extract two subsets,

x_1	0	0	1	1
x_2	0	1	0	1
x_3 x_4				
0 0	**1**	*0*	*0*	**1**
0 1	**0**	*1*	*1*	**0**
1 0	**0**	*1*	*1*	**0**
1 1	**1**	*0*	*0*	**1**
c	**0**	*1*	*1*	**0**

Figure 7.2: High-order complex interaction and HINT

$S_i = \{x_1, x_2\}$ and $S'_i = \{x_3, x_4\}$, from original set of attributes to construct two functions representing $x_1 \oplus x_2$ and $x_3 \oplus x_4$. But, due to its local search, HINT cannot achieve this. This limitation is not convenient for concepts with several interacting attributes when few training data samples are provided, as shown in Section 5.3.

In order to overcome this limitation, GA in MFE3/GA can be modified to generate individuals in the form of groups of subsets; each group is used for constructing a group of functions representing one complex interaction. For example, for concept $\wedge(P_{1,4}, P_{5,8})$, the best individual can be $Ind = \langle\{x_1, x_2\}, \{x_3, x_4\}; \{x_5, x_6\}, \{x_7, x_8\}\rangle$, where the semicolon separates groups of subsets. Then, a set of two functions are generated over the first group of subsets, representing $x_1 \oplus x_2$ and $x_3 \oplus x_4$, and another set of two functions are generated over the second group, representing $x_5 \oplus x_6$ and $x_7 \oplus x_8$. However, this modification makes the search space larger and more complex. A careful analysis of the search space and search strategy is required to further develop this enhancement.

Multi-value Class Labels and Continuous Attributes

Another limitation of MFE3/GA is that the current version of the method can be used only for concepts with binary class labels. Thus, for learning a concept with n-value class labels, MFE3/GA must be repeated $n - 1$ times to learn each class label. In order to overcome this limitation, MFE3/GA needs to assign multi-value labels to tuples. Labeling pure and unknown tuples can be easily modified to include multi-value labels. But extending the binary label of mixed tuples to multi-value labels needs more studies. The current version selects the opposite label to the most frequent label among pure tuples as the label of mixed tuple (Section 4.2.1). When multi-value labels are assigned to pure tuples, there is no opposite label as it was in binary label assignment. A solution could be to select the label of mixed tuples stochastically according to the inverse distribution of pure tuples' labels, so that the less frequent label has more chance to be assigned to the mixed tuple.

MFE3/GA is also limited to concepts with nominal attribute values. Thus, a discretization preprocessing algorithm [Liu *et al.*, 2002; Boulle, 2004; Kurgan and Cios, 2004] is required to transform continuous attributes into nominal ones.

Noisy Data

The other issue is sensitivity of MFE3/GA to noisy data. Real-world data is often noisy. The noise might be due to missing or incorrect attribute values or class labels. MFE3/GA induces new features directly from data. Thus, the quality of constructed features depends strongly on the quality of the training data. This method aims to construct features that are consistent with the training data. When training data contains noise, samples itself may be inconsistent, that is two samples with the same attribute values are labeled differently. It would be interesting to evaluate and improve the performance of MFE3/GA when applied to such data.

In case that the noise is in class labels, one approach that can be applied is a preprocessing mechanism to remove or relabel mislabeled samples before running MFE3/GA. Many research have been realized to design such mechanism [Zeng and Martinez, 2001; Muhlenbach *et al.*, 2004; Venkataraman *et al.*, 2004; Malossini *et al.*, 2006; Sun *et al.*, 2007]. However, if few training samples are provided, such preprocessing mechanism may not improve accuracy. Another approach is to integrate noise handling into MFE3/GA. Some methods use statistical measures to associate a class membership probability to each sample and consider this probability in learning [Lawrence and Schölkopf, 2001; Rebbapragada and Brodley, 2007]. Zupan *et al.* in [2000] introduced a similar mechanism to be included in HINT. Their extended version of HINT aims to construct functions that minimize expected classification error. In their new approach, instead of grouping compatible columns (Section 5.1), they merge columns in a way that minimizes the training data classification error using m-error estimate [Cestnik and Bratko, 1991]. However this approach is computationally expensive and cannot be used for MFE3/GA.

In case that the noise is in attribute values, the correct value of a noisy attribute may be predictable using other samples and attributes. Then, a preprocessing mechanism is applied to modify noisy attributes [Van Hulse *et al.*, 2007]. Another approach is to switch the attribute with the class and apply the same mechanism used for noisy class labels [Teng, 1999; Zhu and Wu, 2004]. More research is required in order to extend MFE3/GA that can be applied to noisy data.

Note that the method was designed for problems when the only available information about the concept is the primitive training data. When no other knowledge is provided about the concept, enough correct data is expected to be available for learning; otherwise, an accurate learning is not possible.

Comparison with Other methods

This research did not empirically compare MFE3/GA with other genetic CI methods on concepts with complex interactions; however, a theoretical comparison is realized in Section 2.3. Genetic CI methods apply algebraic representation (often parse trees) for constructing new features. Thus, they have difficulties in learning concepts with complex interactions when the only available knowledge about the concept is the training data. Still, empirical comparison of MFE3/GA with these methods can be interesting for further study and improvement of the method. Also, it would be interesting to see how MFE3/GA can improve accuracy of other learners apart from C4.5.

Make Use of Domain knowledge

Another direction for future work is to extend MFE2/GA to accept domain knowledge for improving its performance. This research focused on problems when no domain knowledge is provided. One may consider integrating the domain knowledge into the system. For example, if the domain expert has some information about attributes, this information can be used when generating the first population or during genetic operations to guide the search toward more promising solutions.

GA Parameters, Parallel GA, and Other Search Techniques

The other important direction for future research relates to GA aspects of MFE3/GA. The most important one is the computation time of the proposed method. A parallel MFE3/GA on a computer with multiprocessor or a network of computers can speed up this method [Nowostawski and Poli, 1999]. Operations in GA such as fitness evaluation, crossover and mutation can be performed in parallel. The feature extraction of MFE3/GA is also a procedure that can be run in parallel for several attribute subsets. PGAPack library [Levine, 1996] used for implementing MFE3/GA supports parallel GA and can be used to improve the method in terms of computation time. Also, more studies about GA parameters may result in a better performance. This research did not aim to optimize these parameters. Studying other search techniques such as Particle swarm optimization [Kennedy, 1995] could be of interest too. The use of co-evolutionary GA [Bhanu and Krawiec, 2002] or multi objective optimization [Goldberg, 1989] to perform the two tasks, constructing features and finding attribute subsets, is another challenging issue.

Up from Earth's Center through the Seventh Gate
I rose, and on the Throne of Saturn sate,
And many knots unravel'd by the road,
But not the Master-Knot of Human Fate.
Omar Khayyam, 1048–1131, Iran

Appendix A

Concepts Definitions

This appendix defines synthetic concepts used for experiments in this dissertation. All the concepts are composed of one or more complex interactions.

The first group of concepts includes those previously used by Perez [1997]. These concepts are defined over 12 Boolean attributes x_1 to x_{12}. Let $w(x_{i..j}) \overset{\text{def}}{=}$ *weight of attributes x_i to x_j*; that is, the number of ones in $\{x_i, \ldots, x_j\}$, then:

- $\mathrm{P}_{i,j} \overset{\text{def}}{=} w(x_{i..j})$ *is an odd number*; i.e., $parity(x_i, \ldots, x_j)$,
- $\mathrm{cp}_{i,j} \overset{\text{def}}{=} \mathrm{P}_{i,6} \wedge \mathrm{P}_{7,j}$,
- $\mathrm{cdp}_{i,j} \overset{\text{def}}{=} \mathrm{P}_{i,4} \wedge (\mathrm{P}_{\frac{i+j}{2},8} \vee \mathrm{P}_{j,12})$,
- $\mathrm{gw}_{i,j} \overset{\text{def}}{=} w(x_{i..6}) > w(x_{7..j})$,
- $\mathrm{sw}_{i,j} \overset{\text{def}}{=} w(x_{i..6}) = w(x_{7..j})$,
- $\mathrm{nm}_{i,j,k} \overset{\text{def}}{=} w(x_{7-\lfloor k/2 \rfloor..7+\lfloor k/2 \rfloor}) \in \{i, j\}$,
- $\mathrm{rk}_{i,j} \overset{\text{def}}{=} w(x_{7-\lfloor j/2 \rfloor..7+\lfloor j/2 \rfloor}) = i$,
- $\mathrm{mj}_{i,j} \overset{\text{def}}{=} w(x_{i..j}) \geq \lfloor (j-i)/2 \rfloor + 1)$,
- $\mathrm{mx}_6 \overset{\text{def}}{=} mx(x_1, x_2, x_3, x_6, x_9, x_{12})$, where:

$$mx(0,0,x_3,x_6,x_9,x_{12}) = x_3,$$
$$mx(0,1,x_3,x_6,x_9,x_{12}) = x_6,$$
$$mx(1,0,x_3,x_6,x_9,x_{12}) = x_9,$$
$$mx(1,1,x_3,x_6,x_9,x_{12}) = x_{12},$$

- $\mathrm{mx}_6\mathrm{c}_{i,j} \overset{\text{def}}{=} mx_c(x_1, x_2, x_{i..j})$, where:

$$mx_c(0,0,x_{i..j}) = \wedge(x_i, \ldots, x_j),$$

$$mx_c(0,1,x_{i..j}) = \vee(x_i,\dots,x_j),$$
$$mx_c(1,0,x_{i..j}) = parity(x_i,\dots,x_j),$$
$$mx_c(1,1,x_{i..j}) = \neg parity(x_i,\dots,x_j),$$

- $\mathrm{P}_{i,j} \vee (l) \overset{\text{def}}{=} \mathrm{P}_{i,j} \vee w(x_{7..12}) = l$,
- $\mathrm{P}_{i,j} \wedge (l) \overset{\text{def}}{=} \mathrm{P}_{i,j} \wedge w(x_{7..12}) = l$.

The second group of concepts includes those specially designed for experiments to illustrate the difficulty of learning in presence of several complex interactions. These concepts are defined over Boolean attributes except palindrome family of concepts where attributes are 3-valued.

The concept *palindrome*$_6$+2 is palindrome of six attributes with two additional irrelevant attributes. The other concepts are defined as conjunctions $\wedge(f_1,\dots,f_n)$ or disjunctions $\vee(f_1,\dots,f_n)$ where f_m is one of the followings:

- $\mathrm{WL3}_{i,j} \overset{\text{def}}{=} w(x_{i..j}) < 3$,
- $\mathrm{W23}_{i,j} \overset{\text{def}}{=} w(x_{i..j}) \in \{2,3\}$,
- $\mathrm{pal}_{i,j} \overset{\text{def}}{=}$ *palindrome of* x_i to x_j,
- Any of functions $A_{i,i+3}$ $B_{i,i+3}$, $C_{i,i+3}$, $D_{i,i+3}$, and $E_{i,i+3}$, defined over four Boolean attributes x_i to x_{i+3} as explained below (see Figure A.1).

Functions $A_{i,i+3}$, $B_{i,i+3}$ and $E_{i,i+3}$ consider their four attributes as a 2-by-2 bitmap and are true if and only if the bitmap contains the following patterns: function $A_{i,i+3}$ detects if any two (vertically or horizontally) adjacent bits are set to one; function $B_{i,i+3}$ is as A but excluding the case of all bits set to one; and function $E_{i,i+3}$ is as A but including the case of all bits set to zero. Functions $C_{i,i+3}$ and $D_{i,i+3}$ consider their four attributes as a 4-by-1 bitmap (or just a sequence) and are true if and only if the bitmap contains the following patterns: function $C_{i,i+3}$ detects if any two adjacent bits are set to identical values but not all bits have the same value; and function $D_{i,i+3}$ detects if there are any two adjacent bits set to one.

To illustrate the complexity of these concepts, note for instance that the DNF of function $A_{1,4}$ is $x_1x_2 + x_2x_3 + x_3x_4 + x_4x_1$. Some concepts are conjunction of $A_{1,4}$, $A_{5,8}$, and $A_{9,12}$, or other three such functions from above. So the DNFs of concepts are complex and difficult to construct and represent using algebraic representation (Section 2.1.3).

The last group of concepts, Monk concepts, from UC Irvine repository [Blake and Merz, 1998], are defined over 6 attributes with 3, 3, 4, 3, 4, and 2 values, respectively, as follows:

- Monk-1 $\overset{\text{def}}{=} (x_1 = x_2) \vee (x_5 = 1)$
- Monk-2 $\overset{\text{def}}{=}$ *exactly two of six attributes are equal to one*.

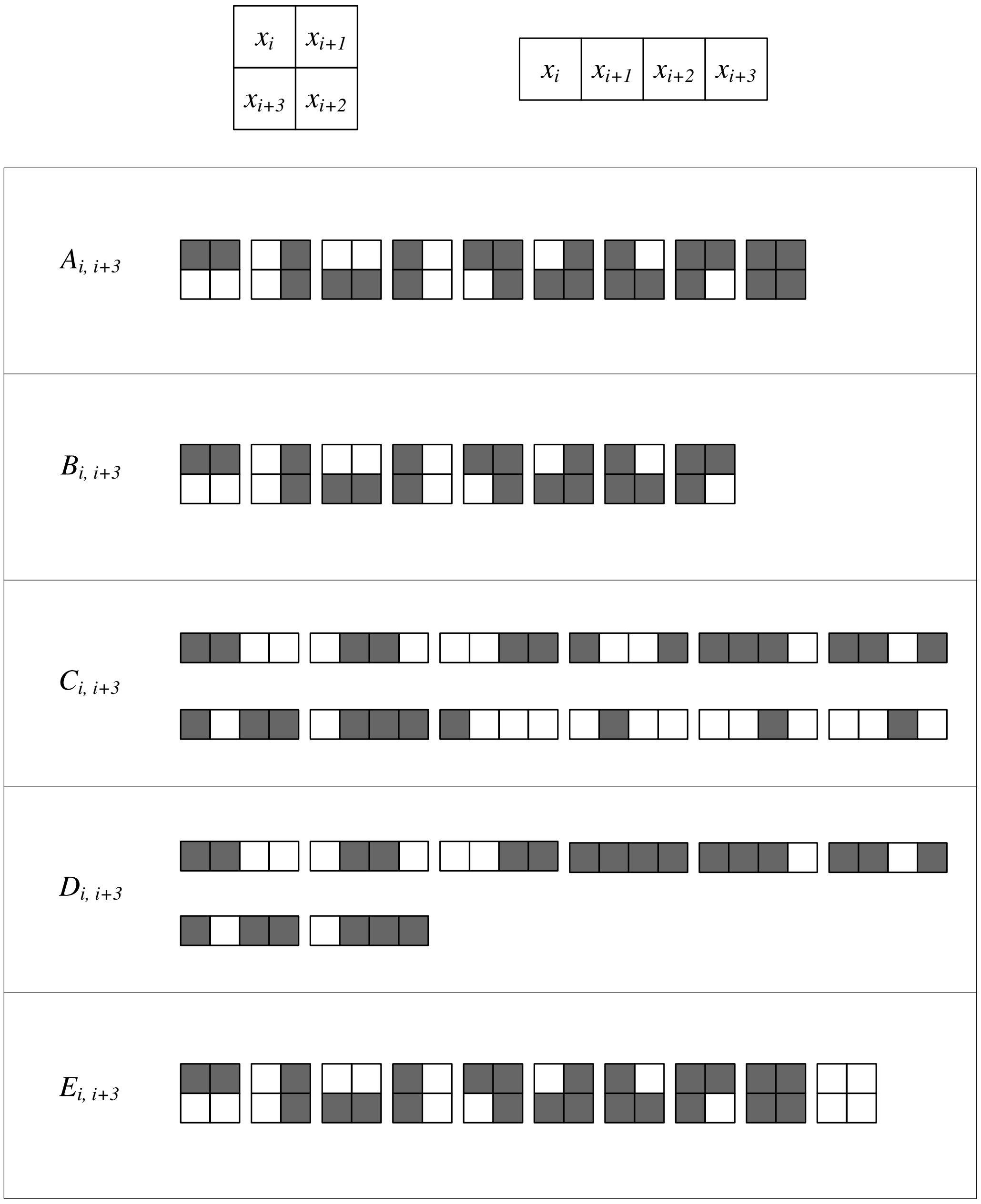


Figure A.1: Functions $A_{i,i+3}$, $B_{i,i+3}$, $C_{i,i+3}$, $D_{i,i+3}$, and $E_{i,i+3}$

Appendix B

PGAPack Default Parameters

Table B.1 illustrates default parameters of PGAPack [Levine, 1996] that are used for CI methods in this work. Some of these parameters are modified for implementing the methods. These parameters are marked in columns three and four for DCI and MFE2/GA, respectively. Same parameters as MFE2/GA are used for MFE3/GA and MFE2/GA$_E$. For more information about these parameters and other aspects of PGA-Pack see [Levine, 1996]. For modified parameters of DCI and MFE2/GA see Sections 3.1.4 and 4.2.4, respectively.

Table B.1: PGAPack Default Parameters

GA Parameter	**Default Value**	DCI	MFE2/ GA
Population size (p)	100	×	
Copied for population replacement	Most fit individuals		
Stopping rule	Max. iteration limit exceeded	×	×
Max iteration	1000	×	×
Max no change iter.	100	×	
Max similarity value	95%	×	
No. of strings to be replaced ($p - m$)	10	×	×
Mutation and/or crossover	Or	×	
Crossover type	Two point crossover	×	×
Probability of crossover	0.85		
Uniform crossover bias (p_u)	0.6		×
Mutation probability	Reciprocal of the string length		×
Selection type	Binary tournament selection	×	
Not allow duplicate strings	False		
Fitness type	Raw fitness value		
Randomly initialize population	True		
Probability of initializing a bit to one	0.5		
Seed random number with clock	True	×	×

m is the number of individuals copied to the next population

Appendix C

Comparing DCI and Different Versions of MFE Methods

This appendix compares the experimental results obtained using DCI (Chapter 3), MFE/GA and MFE2/GA (Chapter 4), and MFE3/GA (Chapter 6). The dissimilarities among DCI, MFE/GA and MFE2/GA are described in Section 4.2, and those among MFE3/GA and MFE2/GA are described in Section 6.3. Since the performance differences were not significant for some experiments, their empirical comparison is moved to this appendix. Table C.1 reports the results over synthetic concepts that are composed of one complex interaction. Note that MFE/GA is the older version of MFE2/GA and, therefore, is not used for these experiments. Table C.2 reports the results over concepts composed of several complex interactions. These experiments were also not performed with MFE/GA and DCI over some concepts (marked by N/A).

All experiments are run over 20 sets of shuffled data (5% training and 95% test data) and average accuracies are reported; except those results marked by * in Table C.2 which are average results of performing experiments over 10 sets of data. Accuracies are compared using t-distribution test with $\alpha = 0.1$.

As it can be seen from Table C.1 the results obtained by MFE2/GA and MFE3/GA are not significantly better than DCI since concepts in this table are composed of one complex interaction. The multi-feature construction property of these MFE methods does not help to construct better features than DCI. However, in Table C.2 all versions of MFE significantly outperform DCI for almost all concepts.

Table C.3 summarizes the results of Tables C.1 and C.2. It shows the number of times a method significantly outperformed the other method for each group of concepts. This table has a similar format as Tables 5.2 and 5.4. For an event "$\text{method}_1 \big/ \text{method}_2$", a frequency "$a/b$ out of c" means method_1 outperformed method_2 for a number of concepts out of c concepts, and method_2 outperformed method_1 for b number of concepts. It can be seen that MFE3/GA is the best competitor for almost all concepts.

Table C.1: Comparing methods – one interaction

Concept	Relev. atts	Irrel. atts	Maj %	DCI +C4.5	MFE2/GA +C4.5	MFE3/GA +C4.5
$gw_{5,8}$	4	8	68.8	100(0)◁	100(0)◁	100(0)◁
$mj_{3,9}$	7	5	50.0	89.1(2.7)	90.1(2.4)◀	89.4(2.3)
$mj_{4,8}$	5	7	50.0	99.7(1.5)	100(0)◁	100(0)◁
$mx_6c_{5,8}$	6	6	50.0	97.8(1.8)	97.6(1.7)	**98.5(1.5)**◀
$mx_6c_{6,7}$	4	8	50.0	100(0)◁	100(0)◁	100(0)◁
$nm_{4,5,7}$	7	5	56.2	89.9(1.6)	89.8(2.5)	90.8(2.6)◀
$rk_{5,7}$	7	5	83.6	95.1(1.7)◀	93.7(3.7)	92.5(5.2)
$rk_{6,7}$	7	5	94.5	*98.3(1.3)*◀	**95.9(2.7)**	94.2(1.9)
$sw_{5,8}$	4	8	62.5	100(0)◁	100(0)◁	100(0)◁
p_4	4	8	50.0	100(0)◁	100(0)	100(0)
p_6	6	6	50.0	98.0(1.8)	98.0(1.5)◀	98.1(1.5)
p_8	8	4	50.0	76.7(2.7)◁	74.7(7.7)	76.7(2.7)◁
$gw_{3,10}$	8	4	63.7	N/A	76.5(5.1)	**79.9(3.3)**◀
$gw_{4,9}$	6	6	65.6	N/A	98.1(2)	98.3(1.5)◀
$mj_{2,10}$	9	3	50.0	N/A	67.3(3.5)	**70.8(4.9)**◀
$mx_6c_{4,9}$	8	4	50.0	N/A	75.5(5.2)	77.5(2.6)◀
$mx_6c_{3,10}$	10	2	50.0	N/A	57.7(4.4)◀	56.8(3.1)
$nm_{5,6,9}$	9	3	59.0	N/A	65.9(3.4)	66.9(3.1)◀
$rk_{6,9}$	9	3	83.6 ◁	N/A	80.5(2.2)	**81.6(1.4)**◀
$rk_{7,9}$	9	3	93.0 ◁	N/A	91.3(1.1)	91.5(0.9)◀
$sw_{3,10}$	8	4	72.7	N/A	75.9(5.4)	77.6(5.9)◀
$sw_{4,9}$	6	6	68.8	N/A	98.6(1.4)◀	98.1(1.7)
AVERAGE	7	5	62.36	N/A	87.60	88.15◀

N/A	results are not available
◁	the higher accuracy
◀	absolutely the highest
Boldface	significantly higher between MFE3/GA and MFE2/GA
underline	significantly higher between MFE3/GA and DCI
italic	significantly higher between MFE2/GA and DCI

Table C.2: Comparing methods – several interactions

Concept	Rel. atts	Irr. atts	Maj %	DCI +C4.5	MFE/GA +C4.5	MFE2/GA +C4.5	MFE3/GA +C4.5
$\mathrm{cdp}_{3,11}$	6	6	62.5	97.6(1.6)	99.2(3.1) ○	**100(0.0)**◂○	99.7(0.7) ○
$\mathrm{cdp}_{2,10}$	9	3	62.5	67.7(2.0)	86.6(10.3) ○	85.8(8.6) ○	**90.3(5.7)**◂○
$\mathrm{cdp}_{1,9}$	12	0	62.5	56.4(2.5)	71.1(7.4) ○	71.7(3.8)◂○	71.1(3.5) ○
$\mathrm{cp}_{5,8}$	4	8	75	100(0.0)◃	100(0.0)◃	100(0.0)◃	100(0.0)◃
$\mathrm{cp}_{4,9}$	6	6	75	97.2(1.8)	98.9(4.0) ○	100(0.0)◃○	100(0.0)◃○
$\mathrm{cp}_{3,10}$	8	4	75	81.6(1.6)	95.7(8.9) ○	*100(0.0)*◃○	100(0.0)◃○
$\mathrm{cp}_{2,11}$	10	2	75	68.0(1.2)	97.1(4.8) ○	96.7(6.1) ○	97.3(6.1)◂○
$\mathrm{P}_{3,6} \wedge (2)$	10	2	88.3	81.0(1.5)	*95.9(2.9)*◂○	93.1(5.7) ○	92.8(6.2) ○
$\mathrm{P}_{3,6} \wedge (3or2)$	10	2	72.7	65.4(2.6)	90.8(5.6) ○	89.3(5.7) ○	**93.2(4.2)**◂○
$\mathrm{P}_{3,6} \wedge (3)$	10	2	84.4	75.9(2.4)	94.1(5.4) ○	93.8(5.2) ○	95.4(3.9)◂○
$\mathrm{P}_{3,6} \vee (2)$	10	2	61.7	57.9(2.2)	90.3(6.6) ○	89.7(5.5) ○	**93.1(2.0)**◂○
$\mathrm{P}_{3,6} \vee (3or2)$	10	2	77.3	69.5(2.2)	92.5(7.5) ○	92.5(5.7) ○	**95.7(1.7)**◂○
$\mathrm{P}_{3,6} \vee (3)$	10	2	65.6	59.3(1.7)	92.1(7.1) ○	91.1(7.6) ○	93.7(1.8)◂○
mx_{6}	6	6	50	97.8(2.1)	N/A	98.8(1.8)	99.1(2)◂○
$\wedge(\mathtt{P}_{1,4},\mathtt{P}_{3,6})$	6	6	75	* 97.7(2.2)	N/A	99.1(1.6) ○	**99.8(0.5)**◂○
$\wedge(\mathtt{P}_{1,6},\mathtt{P}_{3,8})$	8	4	75	* 81.7(1.3)	N/A	92.8(5.7) ○	94.1(2.8)◂○
$\wedge(\mathtt{P}_{1,3},\mathtt{P}_{3,5},\mathtt{P}_{4,6})$	6	6	87.5	* 99.2(1.4)	N/A	99.8(0.7)◃○	99.8(0.7)◃○
$\wedge(\mathtt{P}_{1,4},\mathtt{P}_{2,5},\mathtt{P}_{3,6})$	6	6	87.5	* 99.4(0.8)	N/A	99.6(0.7)◃	99.6(0.7)◃
$\wedge(\mathtt{P}_{1,4},\mathtt{P}_{3,6},\mathtt{P}_{5,8})$	8	4	87.5	* 89.2(1.2)	N/A	96.4(1.8) ○	**98.6(1.7)**◂○
$\wedge(\mathtt{P}_{1,6},\mathtt{P}_{2,7},\mathtt{P}_{3,8})$	8	4	87.5	* 89.0(1.1)	N/A	92.2(3.3) ○	**93.8(2.4)**◂○
$\wedge(\mathtt{P}_{1,6},\mathtt{P}_{7,12})$	12	0	75	N/A	N/A	87.4(8.2)	89.8(6.8)◂
$\wedge(\mathtt{P}_{1,4},\mathtt{P}_{5,8},\mathtt{P}_{9,12})$	12	0	87.5	N/A	N/A	90.8(7.4)	92.4(7.2)◂
$\wedge(\mathtt{WL3}_{1,5},\mathtt{WL3}_{3,7})$	7	5	64.1	N/A	N/A	93.6(5.3)◂	93.1(5.9)
$\wedge(\mathtt{WL3}_{1,5},\mathtt{WL3}_{4,8})$	8	4	68	N/A	N/A	**95.5(6.5)**◂	89.9(9.6)
$\wedge(\mathtt{WL3}_{1,5},\mathtt{WL3}_{5,9})$	9	3	71.5	N/A	N/A	95.3(5.4)◂	93.5(7.0)
$\wedge(\mathtt{WL3}_{1,5},\mathtt{WL3}_{6,10})$	10	2	75	N/A	N/A	91.1(8.2)◂	88.1(8.4)
$\wedge(\mathtt{WL3}_{1,4},\mathtt{WL3}_{3,6},\mathtt{WL3}_{5,8})$	8	4	57.8	N/A	N/A	97.5(2)◂	97.5(2.2)
$\wedge(\mathtt{WL3}_{1,4},\mathtt{WL3}_{5,8},\mathtt{WL3}_{9,12})$	12	0	67.5	N/A	N/A	91.5(11)	92.3(10.5)◂
$\wedge(\mathtt{W23}_{1,6},\mathtt{W23}_{7,12})$	12	0	70.9	N/A	N/A	80.1(8.9)	83.4(9.3)◂
$\wedge(\mathtt{W23}_{1,4},\mathtt{W23}_{5,8},\mathtt{W23}_{9,12})$	12	0	75.6	N/A	N/A	87.8(11.2)	**94.1(9.4)**◂
$\wedge(\mathtt{W23}_{1,5},\mathtt{W23}_{6,10},\mathtt{W23}_{11,15})$	15	0	76	N/A	N/A	100(0)◃	100(0.0)◃
$\wedge(\mathtt{W23}_{1,6},\mathtt{W23}_{7,12},\mathtt{W23}_{13,18})$	18	0	84	N/A	N/A	100(0)◃	100(0.0)◃
$\wedge(\mathtt{A}_{1,4},\mathtt{A}_{5,8},\mathtt{A}_{9,12})$	12	0	82.2	N/A	N/A	94.2(7)	**97.8(4.3)**◂
$\wedge(\mathtt{B}_{1,4},\mathtt{B}_{5,8},\mathtt{B}_{9,12})$	12	0	87.5	N/A	N/A	89.8(4.4)◂	89.6(4.0)
$\wedge(\mathtt{C}_{1,4},\mathtt{C}_{5,8},\mathtt{C}_{9,12})$	12	0	57.5	N/A	N/A	98.1(8.7)	98.5(6.9)◂
$\wedge(\mathtt{D}_{1,4},\mathtt{D}_{5,8},\mathtt{D}_{9,12})$	12	0	87.5	N/A	N/A	91(3.5)	92.3(3.4)◂
$\wedge(\mathtt{E}_{1,4},\mathtt{E}_{5,8},\mathtt{E}_{9,12})$	12	0	75.6	N/A	N/A	90.7(10.7)	93.0(10.5)◂
$\wedge(\mathtt{A}_{1,4},\mathtt{C}_{5,8},\mathtt{E}_{9,12})$	12	0	73.6	N/A	N/A	97.8(4.9)◂	97.5(6.1)
$\wedge(\mathtt{A}_{1,4},\mathtt{B}_{5,8},\mathtt{D}_{9,12})$	12	0	85.9	N/A	N/A	90.4(4.5)	92.0(4.7)◂
$\wedge(\mathtt{A}_{1,4},\mathtt{B}_{5,8},\mathtt{C}_{9,12})$	12	0	78.9	N/A	N/A	94(7.1)	94.6(7.2)◂
$\wedge(\mathtt{B}_{1,4},\mathtt{C}_{3,6},\mathtt{A}_{7,10},\mathtt{D}_{9,12})$	12	0	86.5	N/A	N/A	90.2(2.9)	90.8(3.6)◂
$\wedge(\mathtt{A}_{1,4},\mathtt{B}_{5,8},\mathtt{C}_{9,12},\mathtt{E}_{13,16})$	16	0	87	N/A	N/A	100(0)◃	100(0.0)◃
$\wedge(\mathtt{C}_{1,4},\mathtt{WL3}_{5,8},\mathtt{W23}_{9,12})$	12	0	67.8	N/A	N/A	97.9(6.3)◂	93.7(11.1)
$\wedge(\mathtt{W23}_{1,5},\mathtt{C}_{5,8},\mathtt{WL3}_{8,12})$	12	0	76.6	N/A	N/A	80.7(6.4)	84.0(8.7)◂
$\wedge(\mathtt{W23}_{1,5},\mathtt{C}_{4,7},\mathtt{WL3}_{6,10})$	10	2	76.6	N/A	N/A	84.4(6.4)	**88.7(8.9)**◂
$\vee(\mathtt{pal}_{1,4},\mathtt{pal}_{3,6},\mathtt{pal}_{5,8})$	8	0	70	N/A	N/A	70.6(3.1)	71.4(1.7)◂
$\vee(\mathtt{pal}_{1,4},\mathtt{pal}_{4,7},\mathtt{pal}_{7,10})$	10	0	70	N/A	N/A	100(0)◃	100(0.0)◃
$\mathtt{palindrome}_{6}+2$	6	2	96	N/A	N/A	99.6(0.7)◃	99.6(0.7)◃
AVERAGE	10	2	75.4	N/A	N/A	92.97	93.85◂

$*$	average accuracy over 10 runs	○	significantly better than DCI
N/A	results are not available	**Boldface**	significantly higher between MFE3/GA and MFE2/GA
◃	the higher accuracy	underline	significantly higher between MFE3/GA and MFE/GA
◂	absolutely the highest	*italic*	significantly higher between MFE2/GA and MFE/GA

Table C.3: Summary of Tables C.1 and C.2

Description of the event	Frequency	
	Concepts with one interaction	Concepts with several interactions
MFE3/GA/MFE2/GA	4/1 out of 22	10/2 out of 48
MFE3/GA/MFE/GA	N/A	3/1 out of 13
MFE3/GA/DCI	0/2 out of 12	18/1 out of 20
MFE2/GA/MFE/GA	N/A	1/1 out of 13
MFE2/GA/DCI	0/1 out of 13	17/0 out of 20
MFE/GA/DCI	N/A	12/0 out of 13

Apéndice D

Conclusión y Trabajo Futuro (Spanish)

Este capítulo resume los logros importantes de la investigación realizada y describe algunas posibles orientaciones para el trabajo futuro. La Sección D.1 proporciona un resumen de la tesis; la Sección D.2 describe la contribución de la investigación; y por último, la Sección D.3 analiza las limitaciones y propone posibles formas para ampliar el trabajo.

D.1 Resumen de la Tesis

Interacciones complejas e Inducción Constructiva (IC): Esta investigación se centró en el problema del aprendizaje automático en presencia de interacciones complejas entre atributos en el concepto. La representación de bajo nivel y primitiva de los datos en problemas reales facilita la existencia de interacciones. El Capítulo 1 explicó cómo la interacción complica el aprendizaje. Cuando existen interacciones complejas entre los atributos, la importancia de cada atributo relevante está oculta al sistema de aprendizaje; además, aunque los atributos relevantes se identifiquen por el aprendizaje, las regularidades son todavía difíciles de descubrir (Sección 1.1). IC ha sido introducida para aliviar el problema de aprendizaje de conceptos difíciles. Cuando IC se aplica a los conceptos con interacciones complejas, la construcción de características en IC tiene como objetivo capturar y encapsular las interacciones en unas nuevas características para destacar las regularidades del concepto al sistema de aprendizaje (Sección 1.2).

IC ávida e IC genética: A pesar de muchos avances en IC, los métodos IC actuales todavía tienen dificultades a la hora de aprender conceptos con interacciones complejas. El Capítulo 2 revisó brevemente los trabajos relacionados y estudió los requisitos de un método de IC cuando se aplica a tales conceptos. Este capítulo clasificó los métodos de IC en dos grupos. El primer grupo contiene los métodos de IC ávidas (greedy), que aplican una búsqueda ávida local para encontrar los atributos que interactúan y para construir características (Sección 2.1). El segundo grupo incluye los métodos de IC genética que son los que aplican una búsqueda global como Algoritmos Genéticos (AG) o Programación Genética (PG) (Sección 2.3).

IC ávida - Debilidades: Se resaltaron tres debilidades principales de los actuales métodos ávidos de IC en la Sección 2.1. En primer lugar, estos métodos aplican una búsqueda ávida local para encontrar los atributos que interactúan (Sección 2.1.1). El espacio de búsqueda para encontrar los atributos es grande y con gran variación. Entonces, una búsqueda local puede encontrar una solución óptima local. En segundo lugar, la mayoría de los métodos IC aplican una estrategia ávida para construir y evaluar las características una por una (Sección 2.1.2). Así, la construcción de cada elemento depende de los que previamente se hayan construido. Cuando existan varias interacciones complejas en el concepto objetivo, un método IC puede construir una función incorrecta en un paso primario y evaluarla erróneamente como una función correcta. Por lo tanto, todas las subsiguientes características construidas usando esta función serán irrelevantes. En tercer lugar, la mayoría de los métodos IC aplican un lenguaje algebraico para representar las nuevas características, utilizando los operadores algebraicos (Sección 2.1.3). Para encapsular una interacción compleja se requiere una característica algebraica compleja. Además, si no se proporciona conocimientos previos sobre el concepto, es difícil definir los operadores algebraicos.

Los requisitos para IC: Teniendo en cuenta estas debilidades, se especificaron tres requisitos para un método de IC. Estos requisitos son los siguientes. Primero, un método IC necesita una búsqueda global para saltar las soluciones óptimas locales y encontrar la solución óptima global (el subconjunto óptimo de atributos que interactúan). La Sección 2.2 presentó AG y PG como las técnicas de búsqueda global que pueden aplicarse para IC. Segundo, un método de IC tiene que construir y evaluar varios elementos a la vez para facilitar el aprendizaje en presencia de varias interacciones complejas en el concepto. Un método de búsqueda basado en AG o PG permite la construcción y evaluación de varias características representadas como un solo individuo en la evolución genética. En tercer lugar, un método de IC requiere un lenguaje de representación adecuado cuando las interacciones son complejas y no se dispone de información previa sobre el concepto. La Sección 2.1.3 introdujo la noción de representación no-algebraica (libre de operadores) como una forma de representar las características. Esta representación se prefiere a la representación algebraica cuando el conocimiento previo no está disponible para definir los operadores. La representación no-algebraica permite la extracción directa de características a partir de los datos, lo cual es necesario si los datos de entrenamiento son la única información facilitada sobre el concepto. Esta forma de representación también reduce tanto el tamaño del espacio de búsqueda como la dificultad de construir las características complejas.

IC Genética - Debilidades: La Sección 2.3 revisó los métodos genéticos de IC y marcó sus deficiencias. El problema principal de estos métodos es el uso de la representación algebraica como árboles sintácticos (parse trees [Koza, 1992]) para representar características, lo que produce dificultades, como se explicó en la Sección 2.1.3. El otro problema de la mayoría de los métodos genéticos es la evaluación de características nuevas a través de una función de evaluación derivada de una hipótesis para orientar la búsqueda hacia la solución óptima; es decir, la hipótesis generada por un sistema de aprendizaje auxiliar se usa para la evaluación de las características construidas. El éxito de estos métodos depende en gran medida de dicho sistema de aprendizaje auxiliar. Además, utilizando un sistema de aprendizaje para la evaluación dentro de AG disminuye el rendimiento del método. Por lo tanto, una evaluación derivada de los datos se prefiere a una evaluación derivada de una hipótesis. Otra deficiencia de algunos métodos genéticos es que no emplean AG o PG para construir y evaluar varias características juntamente como un individuo de la población. A pesar de utilizar una búsqueda genética, su estrategia para la construcción de funciones es, todavía, ávida (greedy), construyendo funciones una a una.

Un marco de IC - Descomposición: Teniendo en cuenta las deficiencias de los métodos revisados y los requisitos para la IC, la Sección 2.4 presentó un marco para diseñar un método de IC. Este marco simplifica la tarea de buscar en el inmenso y complejo espacio de búsqueda de IC. Un método de IC debe buscar en el espacio de funciones definidas sobre subconjuntos de atributos. Este espacio crece exponencialmente con el número de atributos y tiene una variación alta. El marco propuesto sugiere partir este espacio en dos: el espacio de subconjuntos de atributos y el espacio de funciones definidas sobre un determinado subconjunto de atributos. Por lo tanto, la tarea de IC se divide en dos pequeñas tareas: buscar los subconjuntos de atributos que interactúan y encontrar una función que representa la interacción entre los atributos de un determinado subconjunto. Se utiliza AG para la primera tarea. Para cada subconjunto de atributos producidos por operadores genéticos se construye una función mediante el análisis de los datos e induciendo una relación entre atributos en el subconjunto y el concepto objetivo. Esta relación se representa no-algebraicamente utilizando un vector de valores que especifica la salida de la función para cada combinación de valores de atributo. Una medida derivada por los datos evalúa la utilidad de la función. Si la función representa una prometedora relación entre los atributos, el subconjunto de atributos se considera un buen subconjunto en AG. De este modo subconjuntos generados se utilizan para inducir funciones a partir de los datos, y estas funciones se utilizan para evaluar subconjuntos y orientar el AG hacia mejores subconjuntos. Mediante esta estrategia se mantiene la dependencia entre las dos tareas mientras se mejora cada una de ellas.

DCI - La primera instancia del marco: El resto de la tesis se centró en dos nuevos métodos diseñados para evaluar el marco. El Capítulo 3 introdujo el DCI. Este método de IC se diseño para analizar la utilidad de la búsqueda global y la representación no-algebraica de las funciones cuando en el concepto objetivo existe una interacción compleja. Tras el marco propuesto en el Capítulo 2, este método aplica AG para encontrar los atributos que interactúan. Para cada subconjunto de atributos como individuo genético, DCI analiza los datos para inducir a una función no-algebraica. La aptitud de cada individuo se determina evaluando la función construida. Se presentó una evaluación basada en entropía, es decir, una medida derivada de los datos. Esta evaluación mide la cantidad de incertidumbre introducida por cada característica construida y su complejidad. Los operadores genéticos se aplican sobre los subconjuntos de atributos para generar mejores individuos. Dado que cada subconjunto de atributos se asocia con una función inducida, cuando se cambian los subconjuntos por los operadores genéticos, las funciones también se cambian. Por lo tanto, AG indirectamente evoluciona funciones para construir mejores funciones. Esta estrategia permite DCI a descomponer el espacio de búsqueda y dividir la tarea principal de IC en dos, manteniendo al mismo tiempo el efecto de cada uno sobre el otro. DCI cumple el primero y el tercero de los requisitos especificados en el Capítulo 2.

Evaluación de DCI - Ventajas: El Capítulo 3 explicó los experimentos realizados para evaluar el uso de AG y la representación no-algebraica de las características en DCI. Resultados empíricos demuestran tres puntos importantes. Primero, cuando existen interacciones complejas entre los atributos, un método de selección atributos no es suficiente para mejorar la precisión del sistema de aprendizaje; se necesita construir unas características que subrayen las regularidades para facilitar aprendizaje. Características construidas por DCI mejoran significativamente la precisión de un sistema de aprendizaje estándar como C4.5 [Quinlan, 1993] cuando existe sólo una interacción compleja entre los atributos. Segundo, cuando el concepto consiste en varias interacciones complejas, la representación no-algebraica ayuda al método de IC a construir características más prometedoras y lograr mejores resultados en comparación con los métodos de IC que usan la representación algebraica. Tercero, la búsqueda global de DCI encuentra con éxito el subconjunto de los atributos que interactúan y su correspondiente función cuando los conceptos se componen de una sola interacción compleja.

Evaluación de DCI - Debilidades: Los experimentos mostraron un requisito para un método de IC que se especificó anteriormente en el Capítulo 2, pero no se consideró en DCI, que es la necesidad de construcción y evaluación de varias características. DCI intenta resumir todas las interacciones en una sola función. Los experimentos mostraron que cuando el número de atributos que interactúan sea alto, a pesar de detectar correctamente los atributos, DCI no puede construir una función que representa todas las interacciones.

MFE2/GA - Superar las deficiencias de DCI: El Capítulo 4 explicó la necesidad de construir más de una función. Cuando existen varias interacciones en el concepto, una función que encapsula las interacciones tiende a ser compleja y difícil de construir. Esta dificultad aumenta cuando se dispone de pocos datos de entrenamiento. Un método de IC tiene que romper la función en varias funciones más pequeñas. Entonces, estas funciones forman un conjunto de las partes relacionadas de una teoría que representa las interacciones. Cada función por sí misma no puede ser evaluada adecuadamente; las funciones deben ser evaluadas conjuntamente como un conjunto de características. La Sección 4.2 presentó MFE2/GA, la segunda instancia del marco propuesto. Este método, mientras que mantiene todas las ventajas de DCI, permite la construcción y la evaluación de varios elementos al mismo tiempo. MFE2/GA emplea AG para construir y evaluar múltiples características. Cada individuo en MFE2/GA representa un conjunto de subconjuntos de atributos; cada subconjunto se asocia con una función que es inducida a partir de los datos. El objetivo de AG es encontrar el mejor conjunto de subconjuntos de atributos y el mejor conjunto de funciones. La evaluación de individuos en MFE2/GA derivada de los datos evalúa el conjunto de funciones construidas como una teoría (Sección 4.2.2). Se diseñaron operadores genéticos especiales para permitir la producción tanto de nuevos subconjuntos de atributos como de nuevos conjuntos de subconjuntos de atributos previamente construidos. Recuérdese que la evolución genética de subconjuntos de atributos lleva asociada implícitamente la evolución de sus correspondientes funciones (Sección 4.2.3). Estos operadores son analizados empíricamente para demostrar su importancia para la convergencia de AG a la solución óptima (Section 4.3).

MFE2/GA - evaluación empírica: MFE2/GA cumple todos los requisitos especificados en el Capítulo 2 para un método de IC. La evaluación empírica mostró que la construcción y evaluación de múltiples características ayudan a MFE2/GA a superar a otros métodos relevantes cuando varias interacciones complejas existan entre los atributos (Sección 4.4). Los experimentos resaltaron la importancia de la construcción y la evaluación de multiples características juntas como partes de una teoría. Estos experimentos demuestran que MFE2/GA encuentra con éxito el conjunto de subconjuntos de atributos que interactúan y construye un conjunto de funciones que representan interacciones complejas. Por lo tanto, supera DCI y otros métodos relevantes cuando el concepto contiene varias interacciones complejas.

MFE2/GA - El análisis de sensibilidad a los datos de entrenamiento: La flexibilidad de MFE2/GA en descomposición y búsqueda de un espacio grande y complejo de funciones hace que este método dependa de la calidad de los datos. MFE2/GA es un método de IC basado en los datos. Este método necesita ver la replicación de la estructura de la función en los datos con el fin de extraerla correctamente. El Capítulo 5 tuvo el objetivo de evaluar la sensibilidad de MFE2/GA al tamaño de los datos de entrenamiento mediante la comparación con el método HINT [Zupan *et al.*, 2001], un método relevante de IC basado a los datos. Ambos MFE2/GA y HINT dependen del tamaño de los datos de entrenamiento en gran medida. Los diseños de los dos métodos se estudiaron analíticamente (Sección 5.2) y se compararon empíricamente (Sección 5.3). Cada método tiene unas funcionalidades importantes. La construcción de características de múltiples valores en HINT (Sección 5.1) le permite romper una interacción compleja en interacciones más pequeñas. Por lo tanto, necesita menos datos de entrenamiento para construir funciones cuando el concepto consiste en una sola interacción compleja de orden alto. Sin embargo, si el concepto está compuesto de varias interacciones complejas de orden bajo, esta propiedad de HINT puede causar sobreajuste (overfitting). La propiedad de construcción de múltiples características de MFE2/GA junto con su búsqueda global ayuda a este método a aprovechar los datos mejor que HINT. Estas propiedades de MFE2/GA son muy importantes para su éxito cuando existen varias interacciones complejas entre los atributos y se dispone de pocos datos de entrenamiento.

MFE3/GA - Mejorar MFE2/GA con una evaluación basada en MDL: Los experimentos del Capítulo 4 indican que el AG en MFE2/GA no siempre es guiado correctamente hacia la solución óptima, por lo que MFE2/GA a veces no encuentra atributos relacionados. El Capítulo 6 analizó los diferentes tipos de funciones de evaluación de aptitud (fitness) y sugirió el uso de dos de ellos con el objetivo de mejorar la exactitud de MFE2/GA. La primera evaluación de fitness se diseñó usando el concepto de entropía y se integró en el sistema, resultando una nueva versión llamada MFE2$_E$/GA. Esta medida evalúa la aptitud del conjunto de características construidas para reducir la impureza en los datos y para mejorar la clasificación. La segunda medida se diseñó basada en los principios de la descripción mínima (MDL [Grunwald, 2007]). Esta evaluación mide la suma de dos valores: la complejidad de características construidas (la teoría) y la cantidad de errores de clasificación producidos por nuevas características (error). Este fitness se integró en una nueva versión del método llamada MFE3/GA. Se realizó un análisis empírico para comparar y evaluar ambas medidas de aptitud. Los resultados indican que el fitness basado en MDL guía AG significativamente mejor que el fitness basado en la entropía. Además, el fitness basado en MDL ayuda a AG a converger a la solución más rápido que el basado en entropía. Este fitness también mejora la exactitud del método comparando con los resultados obtenidos por MFE2/GA en los Capítulos 4 y 5.

MFE3/GA - Evaluación final con los bancos de pruebas de aprendizaje: Para la evaluación final de MFE3/GA, se realizaron y se explicaron en la Sección 6.5 unos experimentos sobre los bancos de pruebas (benchmarks) de UCI (UC Irvine Machine Learning Repository [Blake and Merz, 1998]). Estos experimentos mostraron que el nuevo método se puede aplicar con éxito a los problemas reales para encapsular y subrayar las relaciones entre los atributos y facilitar la tarea de aprendizaje a pesar de existencia de interacciones complejas en los conceptos.

D.2 Contribución de la Investigación

Esta investigación contribuye a los campos de aprendizaje automático, inducción constructiva, selección de atributos, construcción de características, y computación genética y evolutiva. El trabajo facilita la tarea de aprendizaje cuando la representación primitiva y de bajo nivel de los problemas reales produce interacciones complejas entre los atributos. Los conceptos con interacciones complejas suponen un problema para los sistemas de aprendizaje. Los métodos actuales de IC no pueden mejorar el aprendizaje en presencia de varias interacciones complejas cuando la única información disponible son los datos de entrenamiento. La investigación responde a dos importantes preguntas:

1. ¿Cuáles son los requisitos para un método de IC para mejorar el aprendizaje de tales conceptos?
2. ¿Cómo deben cumplirse estos requisitos?

Se especificaron tres importantes requisitos para un método de IC:

1. Una búsqueda global para encontrar subconjuntos de atributos que interactúan y funciones que representan las interacciones en cada subconjunto,
2. Un lenguaje adecuado para encapsular y expresar las interacciones,
3. Una estrategia que permita la construcción y evaluación de varias funciones a la vez.

La investigación argumentó que los métodos de IC presentados anteriormente no cumplen todos los requisitos; y si un método cumple estos requisitos, facilita el aprendizaje de los conceptos complejos cuando la única información proporcionada son datos de entrenamiento.

Con el fin de satisfacer estos requisitos, el trabajo simplifica el enorme y complejo espacio de búsqueda que se debe explorar con el fin de construir características apropiadas. Se ha diseñado un nuevo marco para descomponer el espacio de búsqueda en dos espacios más pequeños, mientras que se refleja el efecto de cada uno sobre el otro. El marco utiliza AG como la búsqueda global para encontrar subconjuntos de atributos que interactúan e induce funciones directamente de los datos de entrenamiento. Este marco permite buscar el conjunto óptimo de subconjuntos de atributos que interactúan y el conjunto óptimo de funciones que representan dichas interacciones, cumpliendo el primero y el tercer requisito especificados para un método de IC. La investigación también intentó facilitar la construcción de características utilizando un lenguaje adecuado para representar las interacciones. La noción de lenguaje no-algebraico (libre de operadores) se presentó y se utilizó en el marco propuesto. Se demostró que cuando el conocimiento previo sobre el dominio no está disponible, esta forma de representación de características facilita la construcción de interacciones complejas y es más apropiada que una representación algebraica tradicionalmente utilizada por otros métodos de IC. El marco propuesto se puede utilizar para diseñar una nueva aplicación de aprendizaje automático o una nueva herramienta para integrarse en un sistema de aprendizaje automático.

Se diseñaron dos métodos experimentales, DCI y MFE2/GA, con el objetivo de evaluar la utilidad del marco para el aprendizaje automático. DCI distingue los atributos que interactúan de los atributos irrelevantes y define una función sobre un subconjunto de atributos que interactúan para resumir y encapsular las interacciones en una nueva característica. MFE2/GA además de detectar los atributos que interactúan, los agrupa en subconjuntos, cada uno conteniendo los atributos que participan en una interacción compleja. El método también construye y evalúa varias funciones a la vez como un individuo de la población del AG, cada una definida sobre un subconjunto de atributos.

MFE2/GA cumple todos los requisitos especificados y, por tanto, difiere de otros métodos de IC. Los análisis experimentales mostraron que este método facilita la tarea de aprendizaje más que otros cuando el concepto se compone de varias interacciones complejas y se proporciona ninguna información previa sobre el concepto.

Proponiendo este método de IC surgió una nueva cuestión a responder que es qué sensibilidad muestra MFE2/GA frente al tamaño de los datos de entrenamiento. El método es una IC basada en los datos y su rendimiento depende en gran medida a los datos de entrenamiento. Los estudios analíticos y empírico mostraron que el diseño del método hace que sea menos sensible a los datos que otros método de IC basado en los datos.

Por último, la investigación propone una nueva versión del método, MFE3/GA. Este método tiene la ventaja de utilizar una medida basada en MDL (Longitud de Descripción Mínima) para evaluar con mayor precisión las características construidas.

El resultado de esta investigación desarrolla un método de construcción de múltiples características basado en AG y la representación no-algebraica de características para facilitar el aprendizaje de conceptos con varias interacciones complejas cuando pocos datos de entrenamiento están disponibles y no está proporcionado ningún conocimiento de experto. Este método puede ser integrado en los sistemas de aprendizaje automático para facilitar el aprendizaje en las condiciones especificadas. Sin embargo, al igual que cualquier otro método, MFE3/GA tiene sus limitaciones y no puede aplicarse a todos los problemas de aprendizaje, como se explicará en la siguiente sección.

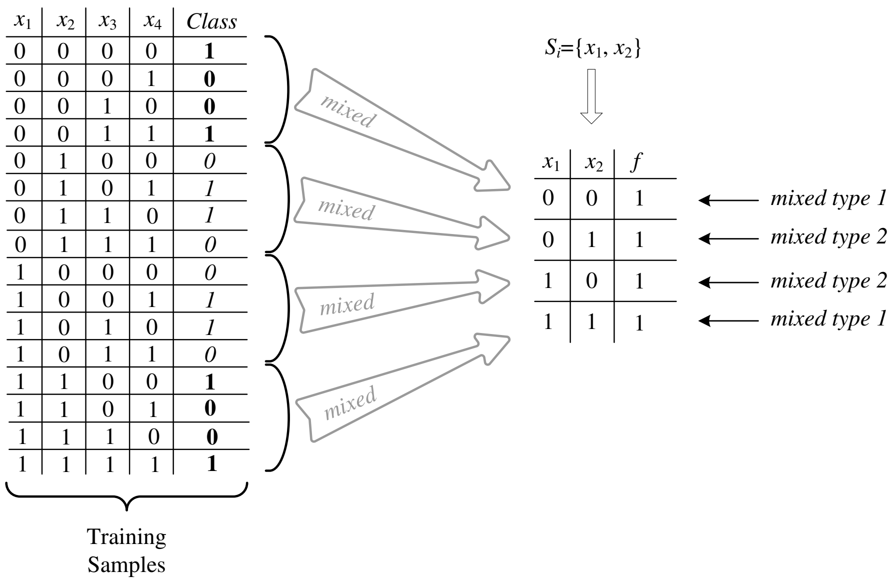
D.3 Limitaciones y Trabajo Futuro

La tesis demostró que un método de IC que satisfaga los requisitos especificados en el marco propuesto facilita el aprendizaje a pesar de complejas interacciones. Sin embargo, MFE3/GA, igual que cualquier otro método, tiene algunas limitaciones y no puede aplicarse a todos los problemas. Esta sección destaca las limitaciones de MFE3/GA y sugiere enfoques para superarlas. Nótese que estas limitaciones son de MFE3/GA y no del marco de IC propuesto ni de los requisitos de IC. Esta sección también describe algunas líneas para futuras investigaciones.

Descomposición de una Interacción Compleja

Esta investigación ilustró que MFE3/GA facilita con éxito el aprendizaje en presencia de varias interacciones complejas a través de descomposición del concepto en un conjunto de funciones, cada una representando una interacción compleja. Sin embargo, MFE3/GA no puede descomponer aún más cada interacción compleja en partes más pequeñas. Como se ilustró en las Secciones 4.4.2 y 5.3.1, cuando el número de atributos que participan en la interacción compleja crezca, la tarea de constricción de características en MFE3/GA se hará más difícil. Por lo tanto, su exactitud se degradará. Recordar de la Sección 4.2.1 que MFE3/GA induce una función mediante la clasificación de las tuplas en el producto Cartesiano de los atributos en tres grupos: tuplas puras, tuplas desconocidas, y tuplas mixtas. Si una interacción compleja se descompone en partes más pequeñas, todas las tuplas serán clasificadas como mixtas. Sin embargo, hay diferentes tuplas mixtas que pueden tratarse de maneras diferentes. Para ilustrar esto, considérese el concepto $P_{1,4} \overset{\text{def}}{=} (x_1 \oplus x_2) \oplus (x_3 \oplus x_4)$, que consiste en una interacción compleja de cuatro atributos (la paridad de cuatro atributos). Esta interacción puede desglosarse en dos más pequeñas, $x_1 \oplus x_2$ y $x_3 \oplus x_4$. Sin embargo, MFE3/GA no puede descubrir estas interacciones de orden menor. Este método proyecta los datos en subconjunto $S_i = \{x_1, x_2\}$, clasifica todas las tuplas en el producto Cartesiano de atributos en S_i como tuplas mixtas, y asigna la misma etiqueta a ellas, como se ilustra en la Figura D.1, dando la función irrelevante$f = \mathbb{T}$ (es decir, "siempre verdadero").

El análisis de los datos en la Figure D.1 indica que hay diferencias entre las tuplas que se pueden utilizar para su clasificación. La proyección de datos en S_i agrupa las muestras de entrenamiento en cuatro conjuntos resultando cuatro tuplas mixtas. Teniendo en cuenta las etiquetas de los datos, se puede observar que el primer y el cuarto grupo de los ejemplos tienen el mismo patrón de las etiquetas de clases, $\langle 1, 0, 0, 1 \rangle$ (las etiquetas en negritas en la Figura D.1), y el segundo y el tercer grupo siguen otro patrón, $\langle 0, 1, 1, 0 \rangle$



x_1	x_2	x_3	x_4	*Class*
0	0	0	0	**1**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**1**
0	1	0	0	*0*
0	1	0	1	*1*
0	1	1	0	*1*
0	1	1	1	*0*
1	0	0	0	*0*
1	0	0	1	*1*
1	0	1	0	*1*
1	0	1	1	*0*
1	1	0	0	**1**
1	1	0	1	**0**
1	1	1	0	**0**
1	1	1	1	**1**

S_i={x_1, x_2}

x_1	x_2	f	
0	0	1	*mixed type 1*
0	1	1	*mixed type 2*
1	0	1	*mixed type 2*
1	1	1	*mixed type 1*

Figura D.1: Interacción compleja y MFE3/GA

(las etiquetas cursivas en la figura). Esta información puede utilizarse para clasificar aún más las tuplas mixtas en dos subgrupos (tuplas mixtas de tipo 1 y tuplas mixtas de tipo 2 en la figura), asignando una etiqueta diferente a cada uno de ellos. MFE3/GA no puede distinguir entre diferentes tuplas mixtas para asignarles etiquetas distintas. Entonces, intenta capturar toda la interacción a la vez mediante la construcción de una función definida sobre todos los atributos que interactúan, lo cual es más difícil de conseguir. Esta limitación puede causar dificultades para MFE3/GA si el número de atributos que participan en la interacción compleja es alto y se dispone de pocos datos de entrenamiento (como se mostró en los experimentos de la Sección 5.3.1). Para estos conceptos una estrategia que permita la clasificación de tuplas mixtas sería útil. HINT supera esta limitación con la propiedad de coloración múltiple (Sección 5.1), asignando diferentes etiquetas a las tuplas mixtas. Este método considera el resto de atributos, x_3 y x_4, para determinar la compatibilidad de tuplas de valores de atributo en $S_i = \{x_1, x_2\}$ y asignar misma etiqueta a tuplas compatibles, como se muestra en cursiva y negrita en la Figura D.2.

$x_3\ x_4$ \ x_1 / x_2	0 / 0	0 / 1	1 / 0	1 / 1
0 0	**1**	*0*	*0*	**1**
0 1	**0**	*1*	*1*	**0**
1 0	**0**	*1*	*1*	**0**
1 1	**1**	*0*	*0*	**1**
c	**0**	*1*	*1*	**0**

Figura D.2: Interacción compleja y HINT

Sería un reto integrar la propiedad de coloración múltiple de HINT con la propiedad de construcción de múltiples características de MFE3/GA. Aunque, si hay otros atributos en el concepto además de los que participan en la interacción compleja, la tarea será más compleja. HINT, para etiquetar las tuplas de valores de atributos en el subconjunto, divide el conjunto original de atributos en un subconjunto y su complementario. Por ejemplo, si el concepto es $\wedge(P_{1,4}, P_{5,8}) \overset{\text{def}}{=} [(x_1 \oplus x_2) \oplus (x_3 \oplus x_4)] \wedge [(x_5 \oplus x_6) \oplus (x_7 \oplus x_8)]$, HINT divide el conjunto original en $S_i = \{x_1, x_2\}$ y $S_i^c = \{x_3, x_4, x_5, x_6, x_7, x_8\}$ y etiqueta las tuplas en S_i considerando los valores de los atributos en S_i^c. Sin embargo, teniendo en cuenta sólo los valores de dos atributos x_3 y x_4 es suficiente para etiquetar las tuplas en S_i. Sería más fácil construir funciones para este concepto si el método pudiera extraer dos subconjuntos, $S_i = \{x_1, x_2\}$ y $S_i' = \{x_3, x_4\}$, del conjunto original de atributos para la construcción de dos funciones que representan $x_1 \oplus x_2$ y $x_3 \oplus x_4$. No obstante, debido a su búsqueda local, HINT no puede lograr esto. Esta limitación no es conveniente cuando el concepto contenga varios atributos que interactúen y se disponga de pocos datos de entrenamiento, como se mostró en la Sección 5.3.

Para superar esta limitación de MFE3/GA, AG en este método puede modificarse para generar individuos en forma de grupos de subconjuntos de atributos, cada grupo se utiliza para construir un grupo de funciones que representan una interacción compleja. Por ejemplo, para el concepto $\wedge(P_{1,4}, P_{5,8})$, el mejor individuo puede ser $Ind = \langle \{x_1, x_2\}, \{x_3, x_4\}; \{x_5, x_6\}, \{x_7, x_8\} \rangle$, donde el punto y coma separa los grupos. Entonces, un conjunto de dos funciones se genera sobre el primer grupo de subconjuntos, representando $x_1 \oplus x_2$ y $x_3 \oplus x_4$, y otro conjunto de dos funciones se genera sobre el segundo grupo, representando $x_5 \oplus x_6$ y $x_7 \oplus x_8$. Sin embargo, esta modificación hace que el espacio de búsqueda sea más grande y más complejo. Un profundo análisis del espacio de búsqueda y la estrategia de búsqueda es necesario para desarrollar y ampliar esta mejora.

Clases con Etiquetas de Múltiples Valores y Atributos Continuos

Otra limitación de MFE3/GA es que la versión actual del método puede utilizarse únicamente para los conceptos con las etiquetas de clase binarias. Entonces, para un concepto con n valores de etiquetas de clase, MFE3/GA debe repetirse $n-1$ veces para aprender cada etiqueta de clase. Para superar esta limitación, MFE3/GA necesita poder asignar etiquetas de múltiples valores a las tuplas. El proceso de etiquetar tuplas puras y desconocidas puede modificarse fácilmente para incluir etiquetas de múltiples valores. Sin embargo, la ampliación de las etiquetas binarias de tuplas mixtas a etiquetas de múltiples valores necesita más estudio. La versión actual para etiquetar la tupla mixta selecciona la etiqueta opuesta a la etiqueta más frecuente entre tuplas puras (Sección 4.2.1). Pero cuando se asigna etiquetas de múltiples valores a las tuplas puras, no hay una noción de etiqueta contraria como la que había en la asignación de etiquetas binarias. Una solución podría ser seleccionar la etiqueta de las tuplas mixtas estocásticamente de acuerdo con la distribución inversa de tuplas etiquetadas como puras, de modo que las etiquetas menos frecuentes tengan más oportunidad de ser asignadas a las tuplas mixtas.

MFE3/GA también está limitado a conceptos con atributos nominales. Entonces, un preprocesamiento de discretización de atributos [Liu *et al.*, 2002; Boulle, 2004; Kurgan and Cios, 2004] es necesario para transformar los atributos continuos en atributos nominales.

Datos con Ruido

La otra cuestión es la sensibilidad de MFE3/GA a datos con ruido. Los datos de problemas reales normalmente tienen errores considerados ruido. El ruido puede ser debido al valor incorrecto o perdido de un atributo o clase. MFE3/GA induce directamente las nuevas características a partir de los datos. Por lo tanto, la calidad de características construidas depende en gran medida de la calidad de los datos de entrenamiento. El objetivo del método es construir características que sean coherentes con los datos de entrenamiento. Cuando los datos contienen ruido, las muestras pueden ser inconsistentes, es decir, dos muestras con mismos valores de atributos estén etiquetadas diferentemente. Sería interesante evaluar y mejorar el rendimiento de MFE3/GA cuando se aplica a dichos datos.

En caso de que el ruido esté en las etiquetas de clase, se puede aplicar un mecanismo de preprocesamiento para eliminar o volver a etiquetar los ejemplos mal etiquetados antes de ejecutar MFE3/GA. Varias investigaciones se han realizado para diseñar tal mecanismo [Zeng and Martinez, 2001; Muhlenbach *et al.*, 2004; Venkataraman *et al.*, 2004; Malossini *et al.*, 2006; Sun *et al.*, 2007]. Sin embargo, si se proporcionan pocas muestras de entrenamiento, tal mecanismo de preprocesamiento puede no mejorar la precisión. Otro enfoque consiste en integrar el manejo de ruido dentro de MFE3/GA. Algunos métodos utilizan medidas estadísticas para asociar a cada una de las muestras una probabilidad de pertenencia a una clase y consideran esta probabilidad en el proceso de aprendizaje [Lawrence and Schölkopf, 2001; Rebbapragada and Brodley, 2007]. Zupan *et al.* en [2000] introdujo un mecanismo similar para integrarlo en HINT. Su versión ampliada de HINT intenta construir funciones que minimicen el error esperado de clasificación. En su nuevo enfoque, en lugar de agrupar columnas según su compatibilidad (Sección 5.1), las columnas se agrupan en una forma que minimice el error de clasificación de los datos de entrenamiento utilizando estimación de m-error [Cestnik and Bratko, 1991]. Sin embargo, este enfoque es computacionalmente costoso y no se puede utilizar para MFE3/GA.

En caso de que el ruido esté en atributos, el valor correcto de un atributo con ruido puede ser previsible utilizando otras muestras y atributos. En este caso, se aplica un mecanismo de preprocesamiento para modificar los atributos con ruido [Van Hulse *et al.*, 2007]. Otro enfoque consiste en intercambiar el atributo con la clase y aplicar el mismo mecanismo utilizado para el caso de ruido en la etiqueta de la clase [Teng, 1999; Zhu and Wu, 2004]. Se necesita más investigaciones para ampliar MFE3/GA con el fin de poder aplicarse a datos con ruido.

Nótese que MFE3/GA ha sido diseñado para problemas cuando la única información disponible sobre el concepto proviene de los datos primitivos de entrenamiento. Cuando no se proporciona conocimientos sobre el concepto, se espera tener disponible bastantes datos precisos para el aprendizaje; de lo contrario, un aprendizaje preciso no es posible.

Comparación Con otros Métodos

Esta investigación no comparó empíricamente MFE3/GA con otros métodos genéticos de IC sobre los conceptos con interacciones complejas; sin embargo, una comparación desde punto de vista analítica se realizó en la Sección 2.3. Los métodos de IC genéticos aplican la representación algebraica (normalmente árboles sintácticos) para construir nuevas características. Por lo tanto, tienen dificultades en aprender conceptos con interacciones complejas cuando el conocimiento disponible sobre el concepto se limita a los datos de entrenamiento. Sin embargo, la comparación empírica de MFE3/GA con estos métodos puede resaltar de interés y conducir a mejoras en el método actual. Además, sería interesante ver cómo MFE3/GA puede mejorar la precisión de otros sistemas de aprendizaje aparte de C4.5.

Hacer Uso de Conocimientos de Dominio

Otra dirección de trabajo futuro es ampliar MFE2/GA para aceptar conocimientos sobre el dominio con el objetivo de mejorar su rendimiento. Esta investigación se centró en los problemas cuando no se proporciona tal conocimiento. Se puede considerar la integración de los conocimientos sobre el dominio. Por ejemplo, si el experto en el dominio tiene cierta información acerca de los atributos, esta información puede ser utilizada para generar la primera población o durante las iteraciones genéticas para converger la búsqueda hacia soluciones más prometedoras.

Los Parámetros de AG, AG paralelo, y Otras Técnicas de Búsqueda

La otra dirección importante para investigación futura se refiere a los aspectos de AG dentro de MFE3/GA. Lo más importante es el tiempo de ejecución del método propuesto. Un MFE3/GA paralelo en un ordenador con multiprocesadores o una red de ordenadores puede acelerar este método [Nowostawski and Poli, 1999]. Las operaciones en AG, como por ejemplo la evaluación de individuos, el cruzamiento y la mutación, pueden realizarse en paralelo. La extracción de características de MFE3/GA también es un procedimiento que puede ejecutarse en paralelo sobre varios subconjuntos de atributos. La librería PGAPack [Levine, 1996] utilizada para la implementación de MFE3/GA proporciona AG paralelo y puede utilizarse para mejorar el método en términos de tiempo de cálculo. Por otra parte, más estudios sobre los parámetros de AG pueden resultar en un rendimiento mejor. La investigación presentada en esta tesis no tenía el objetivo de optimizar estos parámetros. Estudiar las otras técnicas de búsqueda como la optimización de "Particle Swarm" [Kennedy, 1995] podría ser de interés. El uso de AG coevolucionario [Bhanu and Krawiec, 2002] u optimización multi-objetiva [Goldberg, 1989] para realizar las dos tareas, la construcción de funciones y encontrar subconjuntos de atributos, es otra idea tentadora.

Bibliography

[Aha, 1991] David W. Aha. 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 Publishers, Inc.

[Anglano *et al.*, 1997] C. Anglano, A. Giordana, and G. Lo Bello L. Saitta. A network genetic algorithm for concept learning. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, San Francisco, CA, 1997. Morgan Kaufmann Publishers, Inc.

[Antonie *et al.*, 2001] Maria-Luiza Antonie, Osmar R. Zaane, and Alexandru Coman. Application of data mining techniques for medical image classification. In *Proceedings of Second Intl. Workshop on Multimedia Data Mining (MDM/KDD'2001) in conjunction with Seventh ACM SIGKDD*, pages 94–101, San Francisco, USA, 2001.

[Araujo *et al.*, 1999] D. L. A. Araujo, H. S. Lopes, and Alex Alves Freitas. A parallel Genetic Algorithm for rule discovery in large databases. In K Ilto, editor, *Proceedings of 1999 IEEE International Conference on Systems, Man and Cybernetics Conf*, volume 3, pages 940–945, Tokyo, October 1999. IEEE.

[Asuncion and Newman, 2007] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[Bäck, 1996] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford University Press, Oxford, UK, 1996.

[Bensusan and Kuscu, 1996] Hilan Bensusan and Ibrahim Kuscu. Constructive induction using genetic programming. In T. Fogarty and G. Venturini, editors, *Proceedings of Evolutionary Computing and Machine Learning Workshop (ICML'96)*, Bari, Italy, July 1996.

[Bhanu and Krawiec, 2002] Bir Bhanu and Krzysztof Krawiec. Coevolutionary construction of features for transformation of representation in machine learning. In Alwyn M. Barry, editor, *Proceedings of the Bird of a Feather Workshops, Genetic* *and Evolutionary Computation Conference*, pages 249–254, New York, 8 July 2002. AAAI.

[Blake and Merz, 1998] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[Bloedorn and Michalski, 1998] Eric Bloedorn and Ryszard S. Michalski. Data-driven constructive induction: Methodology and applications. In Liu and Motoda [1998], pages 51–68.

[Boulle, 2004] Marc Boulle. Khiops: A statistical discretization method of continuous attributes. *Machine Learning*, 55:53–69, 2004.

[Brodley and Friedl, 1999] Carla E. Brodley and Mark A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research (JAIR)*, 11:131–167, 1999.

[Cestnik and Bratko, 1991] Bojan Cestnik and Ivan Bratko. On estimating probabilities in tree pruning. In Yves Kodratoff, editor, *Proceedings of the European Working Session on Machine Learning (EWSL)*, Lecture Notes in Computer Science, pages 138–150, London, UK, 1991. Springer-Verlag.

[Chan and Mourad, 1994] Pak K. Chan and Samiha Mourad. *Digital Design Using Field Programmable Gate Arrays*. Prentice Hall, 1994.

[Danyluk and Provost, 1993] Andrea Pohoreckyj Danyluk and Foster J. Provost. Small disjuncts in action: Learning to diagnose errors in the local loop of the telephone network. In *Proceedings of the Tenth International Conference of Machine Learning*, pages 81–88, University of Massachusetts, Amherst, MA, USA, June 1993. Morgan Kaufmann Publishers, Inc.

[Davis, 1989] L Davis. Adapting operator probabilities in genetic algorithms. In J. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69, San Mateo, CA, 1989. Morgan Kaufmann Publishers, Inc.

[DeJong *et al.*, 1993] Kenneth A. DeJong, William M. Spears, and Diana F. Gordon. Using Genetic Algorithms for concept learning. *Machine Learning*, 13(2-3):161–188, 1993.

[Demsar *et al.*, 2004] L. Demsar, B. Zupan, and G. Leban. Orange: From experimental machine learning to interactive data mining. White Paper (www.ailab.si/orange), 2004.

[Dhar *et al.*, 2000] Vasant Dhar, Dashin Chou, and Foster Provost. Discovering interesting patterns for investment decision making with glower - a genetic learner overlaid with entropy reduction. *Data Mining and knowledge Discovery*, 4(4):251–280, 2000.

[Dietterich and Michalski, 1981] Thomas G. Dietterich and Ryszard S. Michalski. Inductive learning of structural descriptions: Evaluation criteria and comparative review of selected methods. *Artificial Intelligence*, 16(3):257–294, July 1981.

[Draper and Smith, 1981] Norman R. Draper and Harry Smith. *Applied Regression Analysis*. Wiley-Interscience, 1981.

[Dzeroski and Lavrac, 1993] Saso Dzeroski and Nada Lavrac. Inductive learning in deductive databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):939–949, 1993.

[Estébanez *et al.*, 2007] César Estébanez, José M. Valls, and Ricardo Aler. Gppe: a method to generate ad-hoc feature extractors for prediction in financial domains. *Applied Intelligence*, 2007.

[Freitas and Lavington, 1997] Alex A. Freitas and Simon H. Lavington. *Mining Very Large Databases with Parallel Processing*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[Freitas, 2001] Alex Alves Freitas. Understanding the crucial role of attribute interaction in data mining. *Artificial Intelligence Review*, 16(3):177–199, November 2001.

[Freitas, 2002] Alex Alves Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, New York, Inc., 2002.

[Freitas, 2003] Alex A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery. In Ashish Ghosh and Shigeyoshi Tsutsui, editors, *Advances in evolutionary computing: theory and applications*, Natural Computing Series, pages 819–845. Springer-Verlag, New York, Inc., New York, NY, USA, 2003.

[Gelfand *et al.*, 1998] B. Gelfand, M. Wulfekuhler, and W. Punch. Discovering concepts in raw text: Building semantic relationship graphs. In *Proceedings of International Conference on Machine Learning ICML/AAAI workshop on Learning for Text Categorization*, 1998.

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[Griffith, 1966] Arnold K. Griffith. A new machine-learning techniques applied to the game of checkers. Technical Report AIM-094, Massachusetts Institute of Technology, March 1966.

[Grunwald *et al.*, 2005] Peter D. Grunwald, In Jae Myung, and Mark A. Pitt. *Advances in Minimum Description Length: Theory and Applications*. The MIT Press, 2005.

[Grunwald, 2007] Peter D. Grunwald. *The Minimum Description Length Principle*. The MIT Press, 2007.

[Grzymala-Busse and Hu, 2001] Jerzy W. Grzymala-Busse and Ming Hu. A comparison of several approaches to missing attribute values in data mining. In *RSCTC '00: Revised Papers from the Second International Conference on Rough Sets and Current Trends in Computing*, pages 378–385, London, UK, 2001. Springer-Verlag.

[Hekanaho, 1997] Jukka Hekanaho. GA-based rule enhancement in concept learning. In David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*, pages 183–186. AAAI Press, 1997.

[Holland, 1968] John H. Holland. Hierarchical description of universal spaces and adaptive systems. Technical report, University of Michigan, Department of Computer and Communication Science, Ann Arbor, 1968. ORA Projects 01252 and 08226.

[Holland, 1975] John H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, Michigan, 1975.

[Holte *et al.*, 1989] Robert Holte, Liane Acker, and Bruce Porter. Concept learning and the problem of small disjuncts. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, volume 1, pages 813–818, 1989.

[Holte, 1993] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91, 1993.

[Hsu *et al.*, 2002] William H. Hsu, Cecil P. Schmidt, and James A. Louis. Genetic algorithm wrappers for feature subset selection in supervised inductive learning. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, page 680, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[Hu and Kibler, 1996] YuhJyh Hu and Dennis F. Kibler. Generation of attributes for learning algorithms. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 806–811. AAAI, The MIT Press, August 1996.

[Hu, 1998a] YuhJyh Hu. Constructive induction: Covering attribute spectrum. In Liu and Motoda [1998], pages 257–272.

[Hu, 1998b] YuhJyh Hu. A genetic programming approach to constructive induction. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Proceedings of the Third Annual Genetic Programming Conference*, pages 146–151, University of Wisconsin, Madison, Wisconsin, USA, July 1998. Morgan Kaufmann Publishers, Inc.

[Indurkhya and Weiss, 1991] Nitin Indurkhya and Sholom M. Weiss. Iterative rule induction methods. *Applied Intelligence*, 1(1):43–54, June 1991.

[Jakulin and Bratko, 2003] Aleks Jakulin and Ivan Bratko. Analyzing attribute dependencies. In Nada Lavrac, Dragan Gamberger, Hendrik Blockeel, and Ljupco Todorovski, editors, *Proceedings of PKDD '03*, volume 2838 of *Lecture Notes in Computer Science*, pages 229–240, Cavtat, Croatia, Jan 2003. Springer-Verlag.

[Jakulin and Bratko, 2004] Aleks Jakulin and Ivan Bratko. Testing the significance of attribute interactions. In Carla E. Brodley, editor, *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, pages 409–416, New York, NY, USA, 2004. ACM Press.

[Jakulin *et al.*, 2003] Aleks Jakulin, Ivan Bratko, Dragica Smrke, Janez Demsar, and Blaz Zupan. Attribute interactions in medical data analysis. In Michel Dojat, Elpida T. Keravnou, and Pedro Barahona, editors, *Artificial Intelligence in Medicine, 9th Conference on Artificial Intelligence in Medicine in Europe*, volume 2780 of *Lecture Notes in Computer Science*, pages 229–238, Protaras, Cyprus, October 2003. Springer-Verlag.

[John *et al.*, 1994] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the eleventh International Conference on Machine Learning*, pages 121–129, New Brunswick, NJ, USA, July 1994.

[Kazakov, 1997] Dimitar Kazakov. Unsupervised learning of naive morphology with genetic algorithms. In W. Daelemans, A. Van den Bosch, and A. Weijters, editors, *Workshop Notes of the ECML / MLnet Workshop on Empirical Learning of Natural Language Processing Tasks*, pages 105–112, Prague, Czech Republic, April 1997.

[Kennedy, 1995] R. Kennedy, J. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[Kohavi and John, 1997] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[Kononenko, 1994] Igor Kononenko. Estimating attributes: analysis and extensions of relief. In *Proceedings of European Conference on Machine Learning, ECML*, Lecture Notes in Computer Science, pages 171–182, Catania, Italy, 1994. Springer-Verlag, New York, Inc.

[Koza, 1991] John R. Koza. A hierarchical approach to learning the boolean multiplexor function. In G.J.E. Rawlins, editor, *Foundation of Genetic Algorithms.* Morgan Kaufmann Publishers, Inc., 1991.

[Koza, 1992] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

[Kurgan and Cios, 2004] Lukasz A. Kurgan and Krzysztof J. Cios. Caim discretization algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 16(2):145–153, 2004.

[Langdon and Poli, 2002] William B. Langdon and Riccardo Poli. *Foundations of Genetic Programming.* Springer-Verlag, 2002.

[Langley and Simon, 1995] Pat Langley and Herbert A. Simon. Applications of machine learning and rule induction. *Communications of ACM*, 38(11):54–64, 1995.

[Larsen *et al.*, 2002] Otavio Larsen, Alex Alves Freitas, and Júlio C. Nievola. Constructing X-of-N attributes with a genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, page 1268, San Francisco, July 2002. Morgan Kaufmann Publishers, Inc.

[Lawrence and Schölkopf, 2001] Neil D. Lawrence and Bernhard Schölkopf. Estimating a kernel fisher discriminant in the presence of label noise. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 306–313, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[Levine, 1996] D. Levine. Users guide to the PGAPack parallel genetic algorithm library. Technical Report 18, Argonne National Laboratory, 1996.

[Lin and Bhanu, 2005] Yingqiang Lin and Bir Bhanu. Object detection via feature synthesis using MDL-based genetic programming. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35(3):538–547, 2005.

[Liu and Motoda, 1998] Huan Liu and Hiroshi Motoda, editors. *Feature Extraction, Construction and Selection: A Data Mining Perspective*, volume 453 of *The International Series in Engineering and Computer Science.* Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[Liu *et al.*, 2002] Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6(4):393–423, 2002.

[Malossini *et al.*, 2006] Andrea Malossini, Enrico Blanzieri, and Raymond T. Ng. Detecting potential labeling errors in microarrays by data perturbation. *Bioinformatics*, 22(17):2114–2121, 2006.

[Markovitch and Rosenstein, 2002] Shaul Markovitch and Dan Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 49(1):59–98, 2002.

[Matheus and Rendell, 1989] C.J. Matheus and L.A. Rendell. Constructive induction on decision trees. In *Proceedings of the 11'th International Joint Conference on Artificial Intelligence*, pages 645–650, Detroit, MI, 1989.

[Michalewicz, 1999] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York, Inc., 1999.

[Michalski, 1978] Ryszard S. Michalski. Pattern recognition as knowledge-guided computer induction. Technical Report 927, Department of Computer Science, University of Illinois, Urbana, June 1978.

[Minsky and Papert, 1968] Marvin Minsky and Seymour Papert. Linear separation and learning. Technical Report AIM-167, Massachusetts Institute of Technology, October 1968.

[Mitchell, 1980] Tom M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, New Brunswick, New Jersey, 1980.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning.* McGraw-Hill Company, Inc., 1997.

[Muharram and Smith, 2005] Mohammed Muharram and George D. Smith. Evolutionary constructive induction. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1518–1528, 2005.

[Muhlenbach *et al.*, 2004] Fabrice Muhlenbach, Stéphane Lallich, and Djamel A. Zighed. Identifying and handling mislabelled instances. *Journal of Intelligent Information Systems*, 22(1):89–109, 2004.

[Murphy and Pazzani, 1991] P. Murphy and M. Pazzani. ID2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 183–187, Evanston, IL, 1991. Morgan Kaufmann Publishers, Inc.

[Neri and Giordana, 1995] Filippo Neri and Attilio Giordana. A parallel genetic algorithm for concept learning. In *Proceedings of the sixth International Conference on Genetic Algorithms*, pages 436–443, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[Neter *et al.*, 1996] John Neter, Michael H. Kutner, William Wasserman, and Christopher J. Nachtsheim. *Applied Linear Statistical Models.* McGraw-Hill/Irwin, February 1996.

[Nowostawski and Poli, 1999] M. Nowostawski and Riccardo Poli. Parallel genetic algorithm taxonomy. In *Proceedings of the Third International Conference on Knowledge-based Intelligent Information Engineering Systems KES'99*, pages 88–92. IEEE Computer Society, Augost 1999.

[O'Reilly and Oppacher, 1994] Una-May O'Reilly and Franz Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 73–88, Estes Park, Colorado, USA, 1994. Morgan Kaufmann Publishers, Inc.

[Otero *et al.*, 2003] Fernando E. B. Otero, Monique M. S. Silva, Alex Alves Freitas, and Júlio C. Nievola. Genetic programming for attribute construction in data mining. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward P. K. Tsang, Riccardo Poli, and Ernesto Costa, editors, *Proceedings of the Sixth European Conference in Genetic Programming*, volume 2610 of *Lecture Notes in Computer Science*, pages 384–393. Springer-Verlag, April 2003.

[Pagallo and Haussler, 1990] Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5(1):71–99, 1990.

[Pagallo, 1990] Giulia Pagallo. *Adaptive Decision Tree Algorithms for Learning from Examples.* PhD thesis, University of California at Santa Cruz, 1990.

[Pappa *et al.*, 2002] Gisele L. Pappa, Alex Alves Freitas, and Celso A. A. Kaestner. Attribute selection with a multi-objective genetic algorithm. In *SBIA '02: Proceedings of the 16th Brazilian Symposium on Artificial Intelligence*, Lecture Notes in Computer Science, pages 280–290, London, UK, 2002. Springer-Verlag.

[Pazzani, 1998] M. Pazzani. Constructive induction of cartesian product attributes. In Liu and Motoda [1998], pages 341–354.

[Pérez and Rendell, 1995] Eduardo Pérez and Larry A. Rendell. 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 Publishers, Inc.

[Perez, 1997] Eduardo Perez. *Learning Despite Complex Interaction: An Approach Based on Relational Operators.* PhD thesis, University of Illinois, Urbana-Champaign, 1997.

[Poli and Langdon, 1998] Ricardo Poli and W. B. Langdon. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation Journal*, 6(3):231–252, 1998.

[Qian and Sejnowski, 1988] Ning Qian and Terrence J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 202(4):865–884, August 1988.

[Quinlan and Rivest, 1989] J. Ross Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Inf. Comput.*, 80(3):227–248, 1989.

[Quinlan, 1983] J. Ross Quinlan. Learning efficient classification procedures and their application to chess end games. In *Machine Learning: An Artificial Intelligence*

Approach, pages 463–482. Morgan Kaufmann Publishers, Inc., Tioga, Palo Alto, CA, 1983.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Quinlan, 1993] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, Inc., San Mateo, California, 1993.

[Ragavan and Rendell, 1993] Harish Ragavan and Larry A. Rendell. Lookahead feature construction for learning hard concepts. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 252–259, University of Massachusetts, Amherst, MA, USA, June 1993. Morgan Kaufmann Publishers, Inc.

[Rebbapragada and Brodley, 2007] Umaa Rebbapragada and Carla E. Brodley. Class noise mitigation through instance weighting. In *Proceedings of European Conference on Machine Learning, ECML*, pages 708–715. Springer-Verlag, New York, Inc., 2007.

[Rendell and Ragavan, 1993] Larry A. Rendell and Harish Ragavan. Improving the design of induction methods by analyzing algorithm functionality and data-based concept complexity. In *Proceedings of International Joint Conferences on Artificial Intelligence*, pages 952–959, 1993.

[Rendell and Seshu, 1990] Larry A. Rendell and Raj Seshu. Learning hard concepts through constructive induction: Framework and rationale. *Computational Intelligence*, 6:247–270, 1990.

[Rissanen, 1983] Jorma Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, Jun. 1983.

[Ritthoff *et al.*, 2002] Oliver Ritthoff, Ralf Klinkenberg, Simon Fischer, and Ingo Mierswa. A hybrid approach to feature selection and generation using an evolutionary algorithm. In *UK Workshop on Computational Intelligence*, Birmingham, U.K., September 2002.

[Rothlauf, 2006] Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Samuel, 1959] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Developement*, 3(3):210–229, July 1959.

[Schaffer, 1993] Cullen Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10(2):153–178, 1993.

[Shafti and Pérez, 2003a] Leila S. Shafti and Eduardo Pérez. Constructive induction using non-algebraic feature representation. In *Proceedings of the Third IASTED*

International Conference on Artificial Intelligence and Applications, pages 134–139, Benalmadena, Spain, 8-10 September 2003. Acta Press.

[Shafti and Pérez, 2003b] Leila S. Shafti and Eduardo Pérez. Genetic approach to constructive induction based on non-algebraic feature representation. In Michael R. Berthold, Hans-Joachim Lenz, Elizabeth Bradley, Rudolf Kruse, and Christian Borgelt, editors, *Advances in Intelligent Data Analysis V, Proceedings of the Fifth International Symposium on Intelligent Data Analysis (IDA)*, Lecture Notes in Computer Science, pages 599–610, Berlin, Germany, 28-30 August 2003. Springer-Verlag.

[Shafti and Pérez, 2004] Leila S. Shafti and Eduardo Pérez. Machine learning by multi-feature extraction using genetic algorithms. In Christian Lemaître, Carlos A. Reyes, and Jesús A. González, editors, *Advances in Artificial Intelligence, Proceedings of the Ninth Ibero-American Conference on AI (IBERAMIA)*, Lecture Notes in Computer Science, pages 246–255, Puebla, México, 22-26 November 2004. Springer-Verlag.

[Shafti and Pérez, 2005] Leila S. Shafti and Eduardo Pérez Pérez. Constructive induction and genetic algorithms for learning concepts with complex interaction. In Hans-Georg Beyer *et al.*, editor, *Proceedings of the 7th Genetic and Evolutionary Computation Conference (GECCO)*, pages 1811–1818, New York, NY, USA, 25-29, june 2005. ACM Press.

[Shafti and Pérez, 2007a] Leila S. Shafti and Eduardo Pérez. Fitness function comparison for GA-based feature construction. In Daniel Borrajo, Luis Castillo, and Juan Manuel Corchado, editors, *Current Topics in Artificial Intelligence, Selected Papers from the 12th Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, volume 4788 of *Lecture Notes in Computer Science*, pages 249–258. Springer-Verlag, 12-16 November 2007.

[Shafti and Pérez, 2007b] Leila S. Shafti and Eduardo Pérez. MDL-based fitness for feature construction. In Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors, *Proceedings of the 9th Genetic and Evolutionary Computation Conference (GECCO)*, volume 2, page 1875, New York, NY, USA, 7-11 July 2007. ACM Press.

[Shafti and Pérez, 2007c] Leila S. Shafti and Eduardo Pérez. Reducing complex attribute interaction through non-algebraic feature construction. In V. Devedžic, editor, *Proceedings of the 5th IASTED International Conference on Artificial Intelligence and Applications*, pages 359–365, Innsbruck, Austria, 12-14 February 2007. Acta Press.

[Shannon, 1948] Claude E. Shannon. A mathematical theory of communication. *Bell System Tech. Journal*, 27:379–423 and 623–656, July 1948.

[Sierra and Corbacho, 2002] Alejandro Sierra and Fernando J. Corbacho. Input and output feature selection. In *Proceedings of the International Conference on Artificial Neural Networks*, Lecture Notes in Computer Science, pages 625–630, London, UK, 2002. Springer-Verlag.

[Smith and Bull, 2003] Matthew G. Smith and Larry Bull. Feature construction and selection using genetic programming and a genetic algorithm. In Conor Ryan, Terence Soule, Maarten Keijzer, Edward P. K. Tsang, Riccardo Poli, and Ernesto Costa, editors, *Proceedings of the sixth European Conference in Genetic Programming*, volume 2610 of *Lecture Notes in Computer Science*, pages 229–237, Essex, UK, April 2003. Springer-Verlag.

[Spears and Anand, 1991] William M. Spears and Vic Anand. A study of crossover operators in genetic programming. In *Proceedings of the 6th International Symposium on Methodologies for Intelligent Systems*, Lecture Notes in Computer Science, pages 409–418, London, UK, 1991. Springer-Verlag.

[Spears and DeJong, 1991] William M. Spears and Kenneth A. DeJong. On the virtues of parameterized uniform crossover. In *Proceedings of the Internationa Conference on Genetic Algorithms*, pages 230–236, 1991.

[Spears, 1992] William M. Spears. Crossover or mutation? In L. Darrell Whitley, editor, *Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, pages 221–237, Vail, Colorado, USA, July 1992. Morgan Kaufmann Publishers, Inc.

[Srinivasan and King, 1999] A. Srinivasan and R. D. King. Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and knowledge Discovery*, 3(1):37–57, March 1999.

[Sun *et al.*, 2007] Jiang-wen Sun, Feng-ying Zhao, Chong-jun Wang, and Shi-fu Chen. Identifying and correcting mislabeled training instances. *Future generation communication and networking (FGCN 2007)*, 1:244–250, December 2007.

[Teng, 1999] Choh-Man Teng. Correcting noisy data. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 239–248, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[Thrun *et al.*, 1991] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Džeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The

MONK's problems: A performance comparison of different learning algorithms. Technical Report CS-91-197, Carnegie Mellon University, Pittsburgh, PA, 1991.

[Todorovski and Dzeroski, 1997] Ljupco Todorovski and Saso Dzeroski. Declarative bias in equation discovery. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 376–384, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[Utgoff and Brodley, 1991] Paul E. Utgoff and Carla E. Brodley. Linear machine decision trees. Technical Report COINS, 91-10, Department of Computer Science, University of Massachusetts, Amherst, Massachusetts, January 1991.

[Vafaie and DeJong, 1998] Haleh Vafaie and Kenneth DeJong. Feature space transformation using genetic algorithms. *IEEE Intelligent Systems*, 13(2):57–65, March-April 1998.

[Vafaie, 1998] Haleh Vafaie. *Using genetic algorithms for restructuring feature-based representation space.* PhD thesis, George Mason University, Fairfax, VA, USA, 1998.

[Van Hulse *et al.*, 2007] Jason D. Van Hulse, Taghi M. Khoshgoftaar, and Haiying Huang. The pairwise attribute noise detection algorithm. *Knowledge and Information Systems*, 11:171–190, 2007.

[Venkataraman *et al.*, 2004] Sundara Venkataraman, Dimitris Metaxas, Dmitriy Fradkin, Casimir Kulikowski, and Ilya Muchnik. Distinguishing mislabeled data from correctly labeled data in classifier design. In *ICTAI '04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 668–672, Washington, DC, USA, 2004. IEEE Computer Society.

[Wan and Perkowski, 1992] Wei Wan and Marek A. Perkowski. A new approach to the decomposition of incompletely specified multi-output functions based on graph coloring and local transformations and its application to fpga mapping. In *EURO-DAC '92: Proceedings of the conference on European design automation*, pages 230–235, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[Weiss, 2003] Gary Mitchell Weiss. *The effect of small disjuncts and class distribution on decision tree learning.* PhD thesis, Rutgers, The State University of New Jersey, New Brunswick, NJ, USA, 2003.

[Weiss, 2005] Gary M. Weiss. Mining with rare cases. In Oded Maimon and Lior Rokach, editors, *The Data Mining and Knowledge Discovery Handbook*, pages 765–776. Springer-Verlag, 2005.

[Yang and Honavar, 1998] Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. In Liu and Motoda [1998], pages 117–136.

[Yang *et al.*, 1991] Der-Shung Yang, Larry A. Rendell, and Gunnar Blix. A scheme for feature construction and a comparison of empirical methods. In *Proceedings of the twelfth International Joint Conference on Artificial Intelligence*, pages 699–704, Sydney, Australia, August 1991. Morgan Kaufmann Publishers, Inc.

[Zeng and Martinez, 2001] Xinchuan Zeng and Tony R. Martinez. An algorithm for correcting mislabeled data. *Intelligent Data Analysis*, 5(6):491–502, 2001.

[Zhang and Mühlenbein, 1995] Byoung-Tak Zhang and Heinz Mühlenbein. MDL-based fitness functions for learning parsimonious programs. In E. V. Siegel and J. R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 122–126, MIT, Cambridge, MA, USA, 10–12 1995. AAAI.

[Zhao and Liu, 2007] Zheng Zhao and Huan Liu. Searching for interacting features. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1156–1161, Hyderabad, India, January 2007.

[Zheng, 1995] Zijian Zheng. Constructing nominal X-of-N attributes. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1064–1070, Montréal, Québec, Canada, August 1995. Morgan Kaufmann Publishers, Inc.

[Zheng, 2000] Zijian Zheng. Constructing X-of-N attributes for decision tree learning. *Machine Learning*, 40(1):35–75, 2000.

[Zhu and Wu, 2004] Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: a quantitative study of their impacts. *Artificial Intelligence Revew*, 22(3):177–210, 2004.

[Zupan and Bohanec, 1998] Blaz Zupan and Marko Bohanec. Experimental evaluation of three partition selection criteria for decision table decomposition. *Informatica*, 22(2), 1998.

[Zupan *et al.*, 1999] Blaz Zupan, Marko Bohanec, Ivan Bratko, and Janez Demsar. Learning by discovering concept hierarchies. *Artificial Intelligence*, 109(1-2):211–242, 1999.

[Zupan *et al.*, 2000] Blaz Zupan, Ivan Bratko, Marko Bohanec, and Janez Demsar. Induction of concept hierarchies from noisy data. In *Proceedings of International Conference on Machine Learning*, pages 1199–1206, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers, Inc.

[Zupan *et al.*, 2001] Blaz Zupan, Ivan Bratko, Marko Bohanec, and Janez Demsar. Function decomposition in machine learning. *Machine Learning and Its Applications, Advanced Lectures*, pages 71–101, 2001.

List of Acronyms and Symbols

CI	Constructive Induction
DCI	Decomposed Constructive Induction
DNF	Disjunctive Normal Form
EC	Evolutionary Computation
FC	Feature Construction
GA	Genetic Algorithm
GP	Genetic Programming
MDL	Minimum Description Length
MFE	Multi-Feature Extraction

2^B	maximum length of constructed features
f_i	a function defined over S_i
Ind_i	an individual in GA
K	maximum number of constructed features
M	number of samples in training data
N	number of attributes in S
p	population size in GA
S	original set of attributes
$\mathcal{S}_{F_i}$	search space of all functions defined over the attribute subset S_i
S_i	a subset of attributes
$\mathcal{S}_S$	search space of all subsets of attributes
SS_i	a set of subsets of attributes
$\mathcal{S}_{SF_i}$	search space of all sets of functions defined over SS_i
$\mathcal{S}_{SS}$	search space of all sets of subsets of attributes
T	training data
x_i	the i^{th} attribute in S
x_{ij}	the j^{th} attribute in S_i

Index