UNIVERSITY OF LJUBLJANA
FACULTY OF COMPUTER AND INFORMATION SCIENCE

Blaž Zupan

MACHINE LEARNING BASED ON FUNCTION DECOMPOSITION

DISSERTATION

Supervisor: prof. dr. Ivan Bratko

Ljubljana, 1997

Abstract

The dissertation proposes a new machine learning method that, given a set of training examples, induces a definition of the target concept in terms of a hierarchy of intermediate concepts and their definitions. This effectively decomposes the problem into smaller, less complex problems, and decomposes an initial set of examples into smaller, more manageable sets. Since each example set partially represents a function, the process is called function decomposition.

Two different approaches to decomposition are proposed. A minimal-complexity approach aims at deriving subsets of examples that represent functions of minimal complexity. An minimal-error approach aims to derive a hierarchy of concepts with minimal estimated classification error. These approaches further differ in the type of data they can handle: the minimal-complexity decomposition requires consistent example sets, whereas the minimal-error variant can handle also inconsistent and noisy data, missing values, and uncertainty.

Both decomposition approaches are implemented in program HINT (HIerarchy Induction Tool). They are experimentally evaluated using a set of artificial and real-world learning problems. It is shown that decomposition performs well in terms of classification accuracy and discovery of meaningful concept hierarchies.

The experimental findings further indicate that the decomposition may efficiently be used to support data structuring, hierarchical data analysis, and knowledge discovery. As such, the proposed decomposition method contributes not only to machine learning, but also to constructive induction, knowledge discovery in data bases, data mining, and intelligent data analysis.

Keywords

machine learning
inductive concept learning
classification
function decomposition
minimal-complexity decomposition
minimal-error decomposition
discovery of intermediate concepts
constructive induction
concept hierarchy

Contents

Chapter 1

Introduction

Now that we have gathered so much data, what do we do with it?
Usama Fayyad and Ramasamy Uthurusamy
editorial, *Communications of ACM*, Special issue on Data Mining, November 1996

Recently, many opening statements of this kind appeared in journals, conference proceedings, and other printed materials that deal with data analysis, knowledge discovery, and machine learning. They all express a concern about how to "make sense" from the large volumes of data being generated and stored in almost all fields of human activity.

Especially in the last few years, the digital revolution provided relatively inexpensive and available means to collect and store the data. For example, in the domain of medicine, only four years ago one of the fathers of "Artificial Intelligence in Medicine" Edward H. Shortliffe partially blamed the underdeveloped infrastructure for the failure to fulfill the initial promise of the field (Shortliffe 1993). Recently, however, the situation is changing rapidly: modern hospitals are well equipped with monitoring and other data collection devices, and data is gathered and shared in inter- and intra-hospital information systems. In fact, medical informatics has become a must and an integral part of every successful medical institution (Spackman, Elert & Beck 1993).

The increase of data volume causes greater difficulties to extract useful information for decision support, discovery of underlying principles, or different analysis tasks. The traditional manual data-analysis has become insufficient, and methods for efficient computer-based analysis indispensable. From this need, a new interdisciplinary field of knowledge discovery in databases (KDD) was born. By end large, KDD encompasses statistical, pattern recognition, and machine learning (artificial intelligence) tools to support the analysis of data and discovery of principles that are encoded within the data.

By any means, the results of computer-based analysis have to be communicated to humans in some understandable way. In this respect, the analysis tools have to deliver transparent results and most often facilitate human intervention in the analysis process. A good example of such methods are symbolic machine learning algorithms that, as a result of data analysis, aim to derive a symbolic model (e.g. a decision tree, or a set of rules) of preferably low complexity but high transparency. Their inclusion as a core of KDD substantially increased the interest in machine learning and recently facilitated the research in this area.

When dealing with a complex problem, the approach most often applied is that of "divide-and-conquer": decompose a problem to less complex and more manageable subproblems. This strategy seems to have an obvious parallel in data analysis. This dissertation builds on this particular idea and proposes a machine learning and data analysis method based on dataset decomposition. We show that the proposed decomposition does not only ease the task of data analysis by providing smaller datasets, but it also derives the corresponding hierarchical structure that may introduce new concepts which, once interpreted by humans, may be a discovery by itself.

1.1 Motivation from machine learning perspective

In artificial intelligence, according to Michie (1995), finding a good decomposition is a major tactic both for avoiding the combinatorial explosion and for ensuring the transparent end-product. Michie specifically refers to the decomposition approach to machine learning. Most often the task of machine learning is to induce a general concept from a set of training examples. The induced classification rules can then be used either to classify new examples or, if of sufficient transparency and comprehensibility, to provide the means to discover the underlying principles. For both comprehensibility and classification, it may be beneficial if instead of learning a single complex classification rule from examples, one defines a goal-subgoal (concept-intermediate concepts) hierarchy and learn the rules for each of the subgoals.

Let us illustrate the above idea by an example. Suppose we are given neurophysiological data that describes the neuron conduction block as a function of six neuronal properties (Figure 1.1). The task is to analyze *how* these properties influence the conduction block. Instead of inducing a complex classifier directly from the features, we may decompose the conduction block problem into subproblems by introducing two intermediate concepts. Each sub-problem may now be studied separately by inducing the classifier for each of the concepts. The possible benefits of such an approach are:

- The subproblems are expected to be less complex and easier to analyze than the original problem. For example, the intermediate concept `source/sink capacity` is described using only 3 properties, while the description of the original target concept uses 6.
- The introduction of meaningful intermediate concepts may itself add to the transparency of the result.

However, along with the benefits there are two major problems when using such a structured approach:

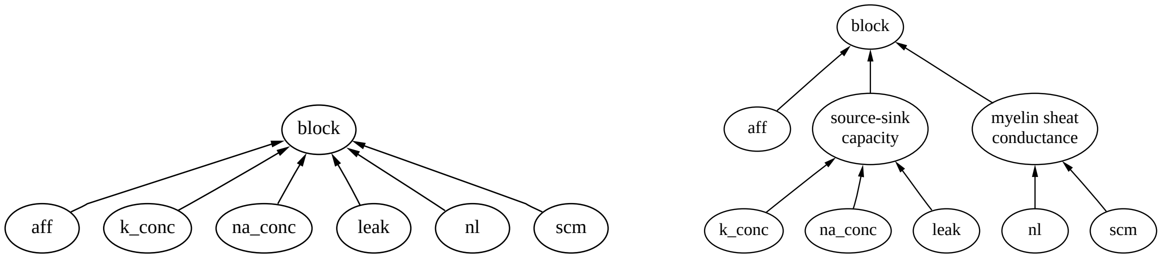


Figure 1.1: Dependency graphs in the conduction block domain for an unstructured and a structured problem with two intermediate concepts.

- The intermediate concepts and their placement in the concept structure have to be given in advance.

- The originally unstructured dataset has to be decomposed into several subsets, each describing a subconcept within the concept structure.

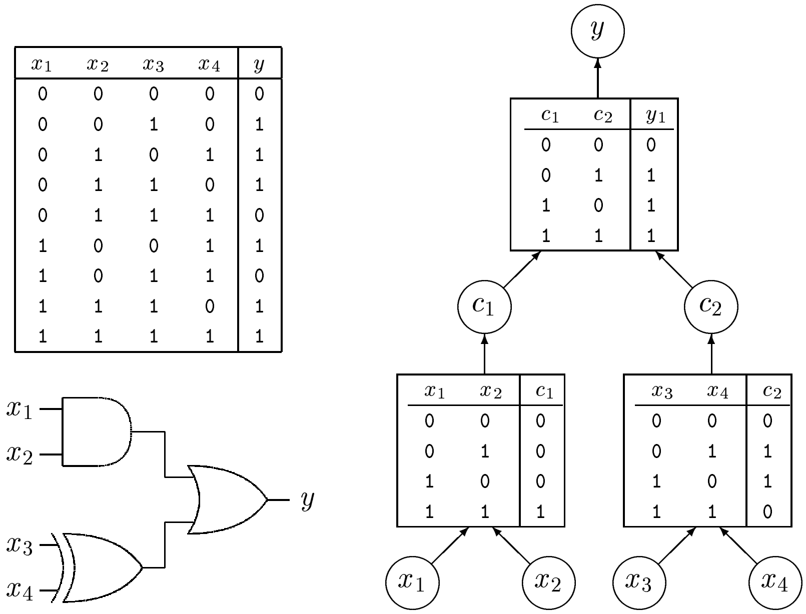
A possible solution to these problem is proposed by an approach called *structured induction* (Shapiro & Niblett 1982). The concept structure is elicited from the domain-expert and the result is a so-called *human-generated problem decomposition.* The next stage involves the expert to select from the training dataset the relevant sets of examples for each of the concepts. For each of these sets a separate decision tree is induced. The structured classifier is then checked against the examples in the dataset and refined by means of changing the concept representations until all examples are correctly classified. Shapiro (1987) used this approach for the classification of a fairly complex chess endgame. He demonstrated that the complexity and comprehensibility ("brain-compatibility") of the obtained solution was to a large extend superior to the unstructured one.

The major drawback of the structured induction method is a manual development of the hierarchy and the selection of examples to induce the classification rules; typically this is a tiresome process that requires active availability of a domain expert over long periods of time, which is usually difficult to achieve. Considerable improvements in this respect could be expected from methods that would automate or at least actively support the designer in the problem decomposition task. The open questions in this respect are:

- Given the concept hierarchy, can the unstructured data be machine-decomposed to several datasets, each describing its own subconcept in the hierarchy?

- Can a meaningful concept hierarchy be discovered directly from the unstructured data?

- How does the structured classifier obtained in such a way perform when classifying unseen examples?



x_1	x_2	x_3	x_4	y
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	1	0	1
1	1	1	1	1

c_1	c_2	y_1
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	c_1
0	0	0
0	1	0
1	0	0
1	1	1

x_3	x_4	c_2
0	0	0
0	1	1
1	0	1
1	1	0

Figure 1.2: A truth table for a Boolean function (upper left), its circuit implementation (lower left), and corresponding hierarchy of concepts with two intermediate concepts $c_1 = x_1$ AND x_2 and $c_2 = x_3$ XOR x_4 and the target concept $y = c_1$ OR c_2.

- Can such a decomposition method handle noisy data, data with possible missing attribute values, and continuously-valued data?

This dissertation deals with the above questions and introduces methods and tools to answer them affirmatively. Our work shares the motivation with structured induction, but uses a rather different approach inspired by the Boolean function decomposition technique used to support the design of digital circuits. This method was first proposed by Ashenhurst (1952) and Curtis (1962). The method iteratively decomposes a Boolean function represented by a truth-table to derive a hierarchical structure of smaller functions preferably realizable with a simple logic gate. For example, such decomposition was used to find the circuit of the Boolean function in Figure 1.2. Represented as a concept structure, the Ashenhurst-Curtis decomposition can be regarded as "automated" structured induction with a limitation to handle Boolean datasets only.

The dissertation thus borrows from two different research areas: it shares the motivation with structured induction while the core of the method is based on the Ashenhurst-Curtis decomposition. Other approaches that most influenced this work include DEX—a structured approach to decision support (Bohanec & Rajkovič 1990)—and the minimal-error pruning approach to noise handling by Niblett & Bratko (1986) and Cestnik & Bratko (1991). The dissertation aims to introduce and experimentally evaluate the decomposition as a new machine learning paradigm.

1.2 Related work

The decomposition approach to machine learning was used early by a pioneer of artificial intelligence, A. Samuel. He proposed a method based on a *signature table system* (Samuel 1967) and used it as an evaluation mechanism for checker playing programs. A signature table system is a tree of input, intermediate, and a single output variable (the nodes). Each internal node is assigned a signature table which is, based on the values of its immediate descendants, used to derive node's value. Each example in the signature table gives a corresponding value of the node for a given combination of values of its immediate descendants. The value of an output variable is determined by a bottom-up derivation that first assigns the values to the intermediate variables, and finally derives the value of the output variable. Given a set of training examples and a signature table system's structure, Samuel proposed an approach to learn the corresponding signature tables by incrementally adjusting the examples describing the intermediate concepts. Compared to his previous approach that was based on the learning of a linear evaluation polynomial (Samuel 1959), Samuel showed that using a signature table system the performance can be significantly improved. Samuel's approach was later improved by Biermann, Fairfield & Beres (1982). Their method, however, did not address the problem of deriving the structure of variables.

While, within machine learning, Samuel and Biermann et al. may be the first to realize the power of decomposition applied to a set of training examples, a very similar approach had been defined earlier in the area of switching circuit design. Curtis (1962) reports that in the late 1940's and 1950's several switching circuit theorist considered this subject and in 1956 Ashenhurst reported on a unified theory of decomposition of switching functions (Ashenhurst 1952). The method proposed by Ashenhurst decomposes a truth table of a Boolean function to be then realized with standard binary gates. Most of other related work of those times is reported and reprinted in (Curtis 1962), where Curtis compares the decomposition approach to other switching circuit design approaches and further formalizes and extends the decomposition theory. Besides a disjunctive decomposition, where each variable can appear as input in just one of the derived tables, Curtis defines a non-disjunctive decomposition where the resulting structure is an acyclic graph, rather than a tree. Furthermore, Curtis defines a decomposition algorithm that aims at constructing a switching circuit of the lowest complexity, i.e., with the lowest number of gates used. Curtis' method is defined over two-valued variables and, as with Biermann et al., requires a complete set of examples.

Recently, the Ashenhurst-Curtis approach was substantially improved by research groups of M. A. Perkowski, T. Luba, and T. D. Ross. Perkowski (1995) reports on the decomposition approach for incompletely specified switching functions. Luba (1995) proposes a method for the decomposition of multi-valued switching functions in which each multi-valued variable is encoded by a set of Boolean variables. The authors identify the potential usefulness of function decomposition for machine learning. Goldman (1994) indicates that the decomposition approach to switching function design might be termed knowledge discovery, since a functions not previously anticipated can be discovered. Similar, but using different terminology, was suggested already by Curtis (1962), who observed that a same training truth table might have different decompositions.

Using 10 different Boolean functions, Goldman (1994) demonstrated the power of generalization of decomposition. Similarly encouraging conclusions were also drawn by Ross, Noviskey, Axtell, Gadd & Goldmann (1994), further exposing the power of function decomposition approach for the Boolean feature extraction and illustrating this by Boolean feature discovery from a simple two-valued training set.

Feature discovery has been at large investigated by constructive induction, a recently active field within machine learning. The term was first used by Michalski (1986), who defined it as an ability of the system to derive and use new attributes in the process of learning. Following this idea and perhaps closest to the function decomposition are the constructive induction systems that use a set of constructive operators to derive new attributes. Examples of such systems are described in (Michalski 1983, Pfahringer 1994, Ragavan & Rendell 1993).

The main limitation of these approaches is that the set of constructive operators has to be defined in advance. Moreover, in constructive induction, the new features are primarily introduced for the purpose of improving the classification accuracy of derived classifier, while the above described function decomposition approaches focused primarily on reduction of complexity, where the impact on classification accuracy can be regarded rather as a side-effect of decomposition-based generalization.

In first-order learning of relational concept descriptions, constructive induction is referred to as predicate invention. An overview of recent achievements in this area can be found in (Stahl 1991).

Within machine learning, there are other approaches that are based on problem decomposition, but where the problem is decomposed by the expert and not discovered by a machine. A well-known example is structured induction (a term introduced by Donald Michie) applied by Shapiro (1987). Their approach is based on a manual decomposition of the problem and an expert-assisted selection of examples to construct rules for the concepts in the hierarchy. In comparison with standard decision tree induction techniques, structured induction exhibits about the same classification accuracy with the increased transparency and lower complexity of the developed models. Michie (1995) emphasized the important role of the structured induction in the future and listed several real problems that were solved in this way.

The concept hierarchy has also been used by a multi-attribute decision support expert system shell DEX (Bohanec & Rajkovič 1990) which has its roots in DECMAK methodology (Efstathiou & Rajkovič 1979, Bohanec, Bratko & Rajkovič 1983). There, a tree-like structure of variables is defined by an expert, and several tools assist in the acquisition of decision tables. These are, like Samuel's signature tables, used to derive the values of intermediate and output variables. DEX also allows different representation of defined decision tables, including decision trees (as in (Shapiro 1987)) and decision rules (Rajkovič & Bohanec 1991). DEX has been successively applied and was used in more than 50 realistic decision making problems.

Although developed independently, the hierarchical structure-based representations of a signature table system, DEX models, or of the systems discovered by a switching function decomposition are surprisingly similar. We refer to such structures as concept structures or concept hierarchies. For the decomposition terminology, we closely follow the definitions used by Curtis (1962), and those used by Perkowski (1995) when dealing with incomplete set of training examples.

1.3 Contributions of the dissertation

The major and original contributions of the dissertation are:

- the minimal-complexity decomposition method,
- the minimal-error decomposition method that can deal with noise, incomplete and imperfect data,
- the implementation of the decomposition system called HINT,
- the experimental assessment of the decomposition method showing that
    - the decomposition can discover useful and interpretable structures,
    - the decomposition can inductively learn concepts with at least comparable classification accuracy to those build by other known machine learning tools.

Other contributions include the definition of a supervised and an unsupervised approach to decomposition, introduction of formal partition selection measures and decomposability criteria for nominal-valued data, redundancy checking and attribute subset selection based on decomposition, and decomposition-specific handling of incomplete data and uncertainty.

1.4 Experimental methodology and datasets used

A function decomposition method as a machine learning and knowledge discovery technique can be regarded from two different perspectives: as a concept induction tool intended to classify new examples, and as a concept discovery tool. The dissertation proposes the decomposition method, and also evaluates it from these two perspectives. To evaluate the classification accuracy, in most of the cases the performance of the decomposition is compared to that of a state-of-the-art induction tool C4.5 (Quinlan 1993).

To assess the decomposition's ability to derive meaningful structures with useful concepts, whenever possible the domains were chosen for which such structures are either known or anticipated. These include a set of artificial domains expressed as functions where the structure was anticipated, and real-world decision support models where the structures were defined by a domain expert.

The dissertation also uses several medical and pharmacology domains. These are:

- lenses prescription domain (section 2.7.3),
- a domain from neuroscience that addresses the nerve conduction block (section 2.7.5),
- protein secondary structure prediction (section 3.4.3),
- seven medical domains from UCI machine learning repository (section 4.6.3).

1.5 An overview of the dissertation

The dissertation is organized as follows.

The minimal-complexity decomposition is described in Chapter 2. This chapter first proposes a single-step decomposition and, after defining the necessary heuristics, introduces the overall decomposition algorithm. Two different means of using decomposition (supervised and unsupervised) are discussed next, followed by a discussion on the utility of the decomposition-derived concept structure for classifications. The chapter includes the experimental evaluation of the proposed method and heuristics.

The decomposition can be used to detect and remove redundancies of attributes and attribute values. This is the topic of Chapter 3, which further proposes the inclusion of these mechanisms into the attribute subset selection algorithm.

Minimal-complexity decomposition can handle only consistent example sets. Most often, the real-world data is inconsistent and includes noise and uncertainty. To handle such types of data, and specifically to handle noise, Chapter 4 proposes the minimal-error decomposition approach. In comparison with the minimal-complexity decomposition, the minimal-error decomposition uses a different single-step decomposition algorithm and defines different partition selection measures and decomposability criteria. The method is evaluated on several selected domains from Chapter 2, and on the set of medical domains for which the classification accuracies of other machine learning tools were known from the literature.

The dissertation concludes with Chapter 5, which includes the discussion of proposed methods and discussion of experimental results, and lists the ideas for further work.

Chapter 2

Minimal-complexity function decomposition

This chapter first defines the function decomposition. Next, it introduces the single-step decomposition. The overall decomposition method is given next, with the description of supervised and unsupervised approach to decomposition. We experimentally evaluate the proposed methods using a set of artificial and real-world learning problems. The chapter concludes with a summary.

2.1 Introduction to function decomposition

This dissertation is concerned with the decomposition method for the functions of type $y = F(a_1, \ldots, a_n)$, where $a_1 \ldots a_n$ are input attributes (or input variables or properties) and y is a class variable (or output variable or property). The symbol y will also be used to denote a target concept to be learned by decomposition. Both input attributes and class are required to be nominal-valued with a finite cardinality of their set of values, denoted with $||a_i||$ and $||y||$, respectively.

Let the function $y = F(a_1, \ldots, a_n)$ be partially specified with a set of examples E_F. The value of class variable defined by an example $e_i \in E_F$ is denoted with $F(e_i)$. In this chapter we will constraint these sets to be consistent, i.e., no two examples may use the same values of attributes but define a different value for the class. Let us first define a decomposition of a function F.

Definition 2.1 Decomposition of function $y = F(a_1, \ldots, a_n)$ is a set of functions $c_i = F_i(x_1, \ldots, x_{n_i})$, where x_i is either an input attribute a_j or an intermediate concept c_j, which is again a function. Arguments x_i are referred to as attributes of function F_i.

Such a decomposition is obtained by inducing the target concept y in terms of a hierarchy of intermediate concepts (c_i) and their definitions. Each function F_i that defines an interme-

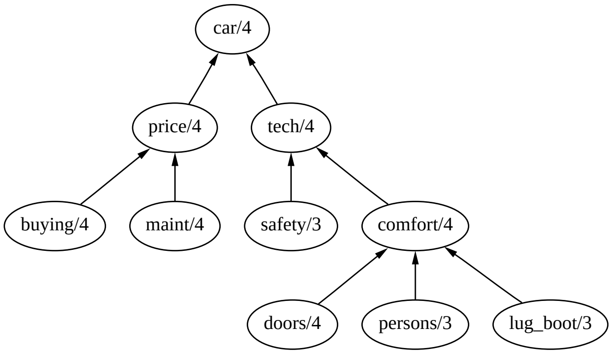


Figure 2.1: A concept tree for `car`.

diate concept c_i within the hierarchy is partially specified by the set of examples E_{F_i}. The decomposition of F is thus a decomposition of the set of examples E_F to the sets E_{F_i}.

The decomposition aims to derive the example sets E_{F_i} that are less complex than the initial example set, i.e., that use less attributes and contain fewer examples. Such sets may be easier to interpret and may introduce meaningful intermediate concepts.

Throughout the dissertation, we refer to such hierarchical system interchangeably as a *concept structure* or *concept hierarchy*. When the disjoint sets of attributes are used by the concepts in the hierarchy, we will call such a hierarchy also a *concept tree*.

Example 2.1 A concept tree for the target concept `car` is given in Figure 2.1. It uses three intermediate concepts `price`, `tech`, and `comfort`, and six input attributes `buying`, `main`, `doors`, `persons`, `lug_boot`, `safety`.

The core of the decomposition algorithm is a *single-step decomposition* which, given a set of examples E_F that partially represent $c_i = F(X)$ decomposes it to E_G and E_H. Two new example sets partially represent $c_i = G(A, c_j)$ and $c_j = H(B)$, where X is a set of F's attributes, A and B are its subsets such that $A \cup B = X$ and $A \cap B = \emptyset$, and c_j is a new intermediate concept. Before the decomposition, a concept c_i used the attributes X, while after the decomposition it is described using the attributes from the set $A \cup \{c_j\}$. The example sets E_G and E_H obtained in the decomposition process are not predefined in any way.

A single-step decomposition can be further applied to the new example sets E_G and E_H. Curtis (1962) has shown that any decomposition that complies with Definition 2.1 can be obtained by iterative applications of a single-step decompositions.

Example 2.2 A decomposition from Figure 2.1 can be obtained by three successive simple decompositions:

1. decompose `car`$= F_0($`buying,maint,doors,persons,lugboot,safety`$)$ to
   `car`$= F_1($`buying,maint,tech`$)$ and `tech`$= F_2($`doors,persons,lugboot,safety`$)$,

2. decompose `tech`$=F_2($`doors`,`persons`,`lugboot`,`safety`$)$ to `tech`$=F_3($`comfort`,`safety`$)$ and `comfort`$=F_4($`doors`,`persons`,`lugboot`$)$,

3. decompose `car`$=F_1($`buying`,`maint`,`tech`$)$ to `car`$=F_5($`price`,`tech`$)$ and `price`$=F_6($`buying`,`maint`$)$

Figure 2.2 shows the evolving hierarchical structure that corresponds to these decomposition steps.

We are concerned only with disjoint decomposition, i.e., a decomposition which yields the functions F_i with disjoint sets of attributes. A more general decomposition is *non-disjoint decomposition*, where functions can share attributes. A non-disjoint decomposition is obtained by iterative applications of non-disjoint single-step decompositions. Curtis (1962), Perkowski (1995), and Zupan & Bohanec (1996) have shown that single-step non-disjoint decomposition is a special case of simple disjunctive decomposition and that its implementation is a straightforward extension of simple disjunctive decomposition.

2.2 Single-step decomposition

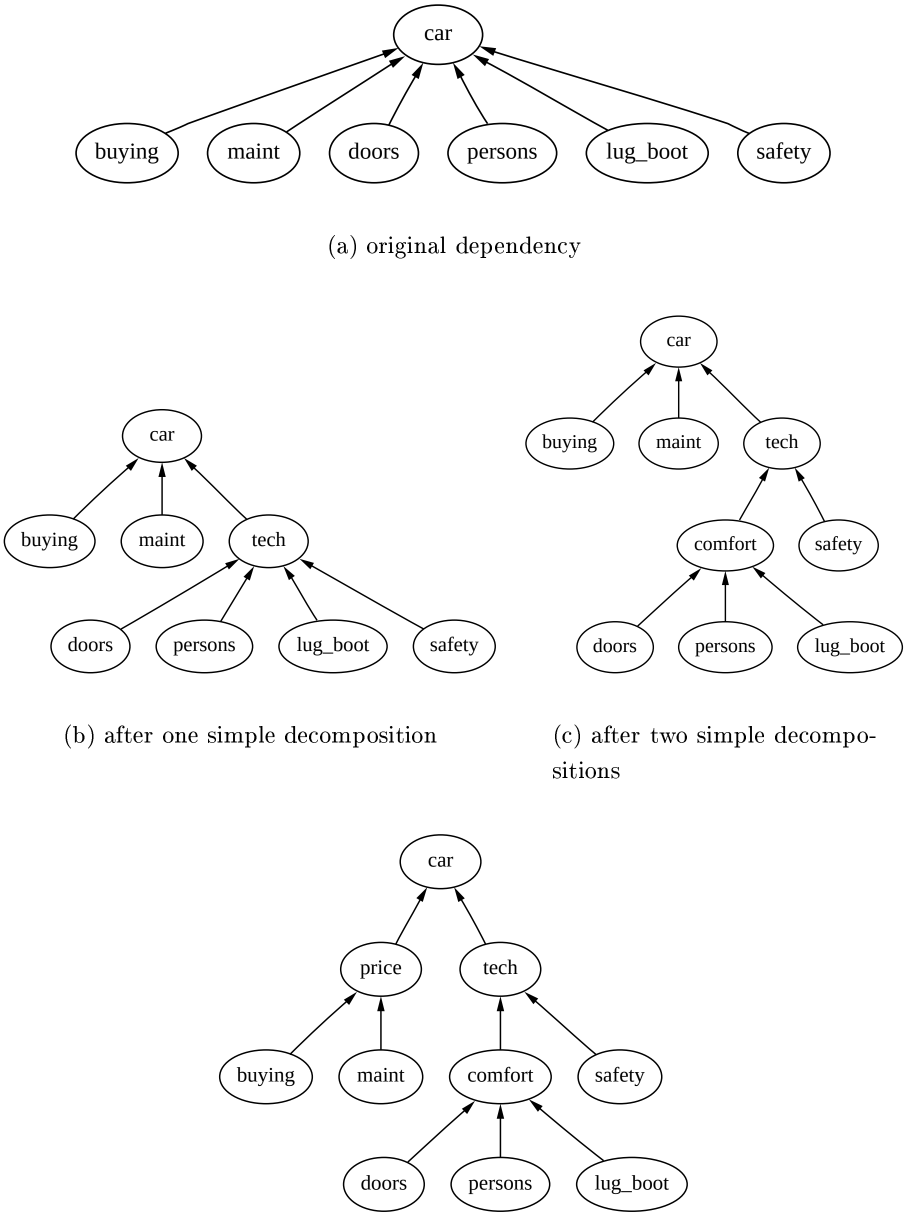
Definition 2.2 Given a set of examples E_F that partially specify the function $c_i = F(X)$ and a partition of attributes X to sets A and B, a **single-step decomposition** of F are functions $c_i = G(A, c_j)$ and $c_j = H(B)$. Sets G and H are partially specified by the example sets E_G and E_H, respectively, that are derived and are consistent with example set E_F. E_G and E_H are discovered in the decomposition process and are not predefined in any way. X is a set of attributes $x_1, \ldots, x_m$, and A and B are a nontrivial partition of attributes in X, such that $A \cup B = X$, $A \cap B = \emptyset$, $A \neq \emptyset$, and $B \neq \emptyset$.

We will use the names *free set* and *bound set* for attribute sets A and B, respectively, and use a notation $A|B$ for the partition of attributes X into these two sets. Before the decomposition, the concept c_i was described by an example set E_F and is after the decomposition described by an example set E_G. Decomposition discovers a new intermediate concept c_j which is described by an example set E_H.

Given the partition $A|B$ of attributes X, the single-step decomposition aims to decompose E_F to E_G and E_H. We first present an example of such a decomposition and then define the single-step decomposition method.

2.2.1 Example

Let us assume a function $y = F(x_1, x_2, x_3)$ where x_1, x_2, and x_3 are attributes and y is the target concept. y, x_1, and x_2 can take the values $\{$`lo`, `med`, `hi`$\}$; x_3 can take the values `lo`, `hi`. The function F is partially specified with a set of examples in Table 2.1.



(a) original dependency

(b) after one simple decomposition

(c) after two simple decompositions

(d) final concept structure after three simple decompositions

Figure 2.2: An evolving concept tree for `car`.

x_1	x_2	x_3	y
lo	lo	lo	lo
lo	lo	hi	lo
lo	med	lo	lo
lo	med	hi	med
lo	hi	lo	lo
lo	hi	hi	hi
med	med	lo	med
med	hi	lo	med
med	hi	hi	hi
hi	lo	lo	hi
hi	hi	lo	hi

Table 2.1: A set of examples that partially describe the function $y = F(x_1, x_2, x_3)$.

There are three partitions of the attributes: $\langle x_1\rangle|\langle x_2, x_3\rangle$, $\langle x_2\rangle|\langle x_1, x_3\rangle$, and $\langle x_3\rangle|\langle x_1, x_2\rangle$, and three corresponding decompositions: $y = G_1(x_1, H_1(x_2, x_3))$, $y = G_2(x_2, H_2(x_1, x_3))$, and $y = G_3(x_3, H_3(x_1, x_2))$. These decompositions are given in Figure 2.3. The comparison shows that:

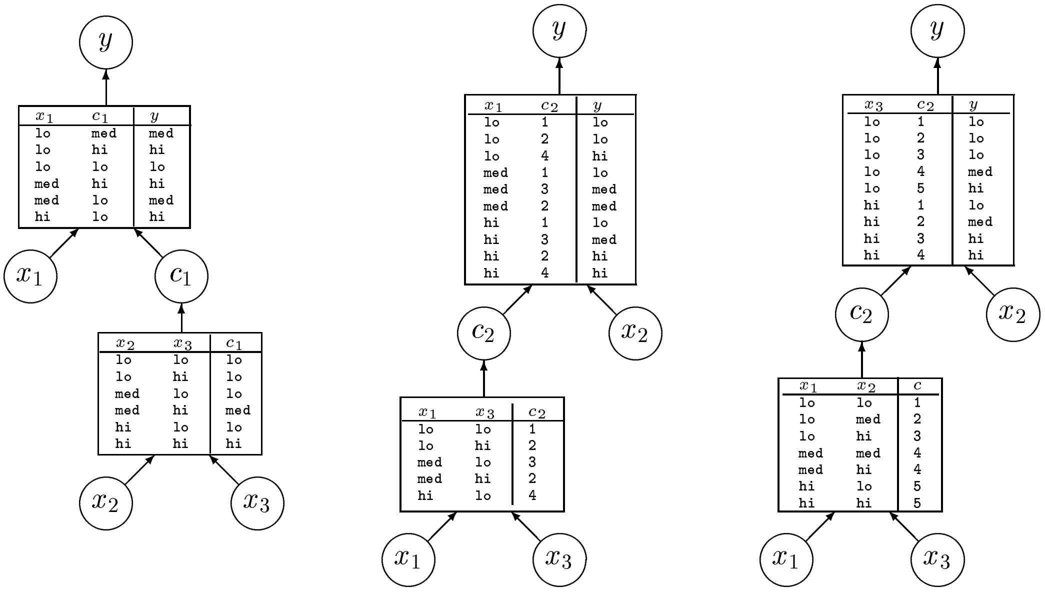
- Example sets in the decomposition $y = G_1(x_1, H_1(x_2, x_3))$ are overall smaller than those for the other two decompositions.
- The new concept $c_1 = H_1(x_2, x_3)$ uses only three values, whereas that for $H_2(x_1, x_3)$ uses four and that for $H_3(x_1, x_2)$ uses five.
- By inspecting the example sets for H_1 and G_1 it is easy to see that c_1 corresponds to MIN(x_2, x_3) and y to MAX(x_1, c_1). It is harder to interpret the sets of examples for G_2, H_2, G_3, and H_3.

Among the three attribute partitions it is therefore beneficial to decide for $\langle x_1\rangle|\langle x_2, x_3\rangle$ and decompose $y = F(x_1, x_2, x_3)$ to $y = G_1(x_1, c_1)$ and $c_1 = H_1(x_2, x_3)$). An intuitive criterion that would indeed prefer this decomposition is for example the minimal number of values required for the new intermediate concept c_1.

2.2.2 Single-step decomposition method

Let E_F be a set of examples that partially specify the function $c_i = F(X)$ and let $A|B$ be a partition of attributes X. The single-step decomposition derives new example sets E_G and E_H from E_F, such that they partially specify functions $c_i = G(A, c_j)$ and $c_j = H(B)$, respectively. The single-step decomposition starts with the derivation of a partition matrix.

Definition 2.3 Given a partition of X to $A|B$, a **partition matrix** $\mathcal{P}_{A|B}$ is a tabular representation of example set E_F with all combinations of values of attributes in A as row labels



x_1	c_1	y
lo	med	med
lo	hi	hi
lo	lo	lo
med	hi	hi
med	lo	med
hi	lo	hi

x_2	x_3	c_1
lo	lo	lo
lo	hi	lo
med	lo	lo
med	hi	med
hi	lo	lo
hi	hi	hi

x_1	c_2	y
lo	1	lo
lo	2	lo
lo	4	hi
med	1	lo
med	3	med
med	2	med
hi	1	lo
hi	3	med
hi	2	hi
hi	4	hi

x_1	x_3	c_2
lo	lo	1
lo	hi	2
med	lo	3
med	hi	2
hi	lo	4

x_3	c_2	y
lo	1	lo
lo	2	lo
lo	3	lo
lo	4	med
lo	5	hi
hi	1	lo
hi	2	med
hi	3	hi
hi	4	hi

x_1	x_2	c
lo	lo	1
lo	med	2
lo	hi	3
med	med	4
med	hi	4
hi	lo	5
hi	hi	5

Figure 2.3: Three different decompositions of the decision table from Figure 2.1.

and of B as column labels. Each example $e_i \in E_F$ has its corresponding entry in $\mathcal{P}_{A|B}$ with a row index $A(e_i)$ and a column index $B(e_i)$. Elements of $\mathcal{P}_{A|B}$ with no corresponding examples in E_F are denoted by "-". A column a of $\mathcal{P}_{A|B}$ is called non-empty if there exists an example $e_i \in E_F$ such that $B(e_i) = a$.

Each column in the partition matrix denotes the behavior of F when the attributes in the bound set are constant. Columns that exhibit the same behavior are called compatible and can be represented by the same value of c_j. Example partition matrices are given in Figure 2.4.

Definition 2.4 Columns a and b of partition matrix $\mathcal{P}_{A|B}$ are **compatible** if $F(e_i) = F(e_j)$ for every pair of examples $e_i, e_j \in E_F$ with $A(e_i) = A(e_j)$ and $B(e_i) = a$, $B(e_j) = b$. The number of such pairs is denoted by $d(a, b)$.

Note that according to this definition the unspecified $\mathcal{P}_{A|B}$ elements are compatible with any value. The number of values for c_j corresponds to the number of groups of mutually compatible columns. The lowest number of such groups is called *column multiplicity* and denoted by $\nu(A|B)$. It is derived by the coloring of column incompatibility graph.

Definition 2.5 Column incompatibility graph $\mathcal{I}_{A|B}$ is a pair (V, E), where each non-empty column i of $\mathcal{P}_{A|B}$ is represented by a vertex $v_i \in V$, and an edge $(v_i, v_j) \in E$ connects two vertices if the corresponding columns of v_i and v_j are incompatible.

x_2	lo	lo	med	med	hi	hi
x_1 x_3	lo	hi	lo	hi	lo	hi
lo	lo	lo	lo	med	lo	hi
med	-	-	med	-	med	hi
hi	hi	-	-	-	hi	-
c_1	1	1	1	2	1	3

x_1	lo	lo	med	med	hi	hi
x_2 x_3	lo	hi	lo	hi	lo	hi
lo	lo	lo	-	-	hi	-
med	lo	med	med	-	-	-
hi	lo	hi	med	hi	hi	-
c_2	1	2	3	2	4	-

x_1	lo	lo	lo	med	med	med	hi	hi	hi
x_3 x_2	lo	med	hi	lo	med	hi	lo	med	hi
lo	lo	lo	lo	-	med	med	hi	-	hi
hi	lo	med	hi	-	-	hi	-	-	-
c_3	1	2	3	-	4	4	5	-	5

Figure 2.4: Partition matrices and their column labels for three different partitions of attributes x_1, x_2, and x_3, and the example set from Table 2.1.

Then, $\nu(A|B)$ is the number of colors needed to color $\mathcal{I}_{A|B}$. Namely, the proper coloring guarantees that two vertices representing incompatible columns are not assigned the same color. The same colors are only assigned to the columns that are compatible. Therefore, the optimal coloring discovers the lowest number of groups of compatible $\mathcal{P}_{A|B}$ columns. An example of a colored incompatibility graph is given in Figure 2.5.

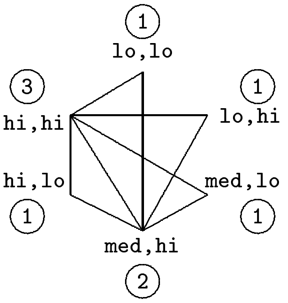


Figure 2.5: Incompatibility graph for the partition $\langle x_1\rangle|\langle x_2, x_3\rangle$ and the partition matrix of Figure 2.4. Numbers in circles represent different colors (labels) of vertices.

Graph coloring is an NP-hard problem and the computation time of an exhaustive search algorithm is prohibitive even for small graphs with about 15 vertices. Instead, Perkowski (1995) suggested a Color Influence Method of polynomial complexity and showed that the method performs well compared to the optimal algorithm. The Color Influence Method sorts the vertices by their decreasing connectivity and then assigns to each vertex a color that is different from the colors of its neighbors so that a minimal number of colors is used. We use the same coloring method, with the following improvement: when a color is to be assigned to vertex v and several compatible vertices have already been colored with different colors, the color is chosen that is used for a group of colored vertices $v_1, \ldots, v_k$ that are *most compatible* to v. The degree of compatibility is estimated as $\sum_1^k d(v, v_i)$ (see Definition 2.4 for d).

Each vertex in $\mathcal{I}_{A|B}$ denotes a distinct combination of values of attributes in B, and its label (color) denotes a value of c_j. It is therefore straightforward to derive an example set E_H from the colored $\mathcal{I}_{A|B}$. The attribute set for these examples is B. Each vertex in $\mathcal{I}_{A|B}$ is an example in set E_H. Color c_j of the vertex is the class of the example.

Set E_G is derived as follows. For any value of c_j and combination of values of attributes in A, $c_i = G(A, c_j)$ is determined by looking for an example e_i in row $A(e_i)$ and in any column labeled with the value of c_j. If such example exists, an example with attribute set $A \cup \{c_j\}$ and class $c_i = F(e_i)$ is added to E_G.

Decomposition generalizes every undefined ("-") element of $\mathcal{P}_{A|B}$ in row a and column b, if a corresponding example e_i with $a = A(e_i)$ and column $B(e_i)$ with the same label as b is found. For example, an undefined element $\mathcal{P}_{A|B}$[`<hi>`,`<lo,hi>`] of the first partition matrix in Figure 2.4 was generalized to `hi` because the column `<lo,hi>` has the same label as columns `<lo,lo>` and `<hi,lo>`.

An interesting situation occurs when the number of values required for c_j is 1. That means that $c_j = H(B)$ is constant. H is therefore not needed and the attributes in B are redundant. The example set E_F is thus not decomposed but rather transformed to E_G, which uses only the attributes in A. The utility of decomposition to discover redundancy and select a subset of attributes to consistently represent a given example set is further investigated in Chapter 3.

Most often the machine learning algorithms deal with sparse data-sets. For these, the implementation using the partition matrix is memory inefficient. Instead, the incompatibility graph $\mathcal{I}_{A|B}$ can be derived directly from the example set E_F. Such derivation requires a different definition for the edges in $\mathcal{I}_{A|B}$:

Definition 2.6 An edge (v_i, v_j) of incompatibility graph $\mathcal{I}_{A|B}$ connects two vertices v_i and v_j if there exist e_k and e_l with $F(e_k) \neq F(e_l)$ such that $A(e_k) = A(e_l)$, $i = B(e_k)$, and $j = B(e_l)$.

Demšar (1996) proposed an algorithm that efficiently implements the construction of $\mathcal{I}_{A|B}$ using the above definition. The algorithm first sorts the examples E_F based on the values of attributes in A and values of c_i. The examples with the same $A(e_i)$ constitute groups that correspond to rows in partition matrix $\mathcal{P}_{A|B}$. Within each group, examples with the same value of c_i constitute subgroups. Two examples that are in the same group but in different subgroups have a corresponding edge in $\mathcal{I}_{A|B}$.

Again, E_H is derived directly from the colored $\mathcal{I}_{A|B}$. The sorted examples of E_F are then used to efficiently derive E_G. With coloring, each subgroup has obtained a label (value of c_j). Each subgroup then defines a single example of E_H with the values of attributes in A and a value of c_j, and a value of c_i which is the same and given by any example in the subgroup.

Example 2.3 For an example set from Table 2.1 and for the partition $\langle x_1\rangle|\langle x_2, x_3\rangle$, the examples sorted on the basis of the values of attributes in A and values of y are:

x_1	x_2	x_3	y
lo	**lo**	**lo**	**lo**
lo	lo	hi	lo
lo	med	lo	lo
lo	hi	lo	lo
lo	**med**	**hi**	**med**
lo	hi	hi	hi
med	med	lo	med
med	hi	lo	med
med	hi	hi	hi
hi	lo	lo	hi
hi	hi	lo	hi

The double lines delimit the groups and the single lines the subgroups. Now consider the two instances printed in bold. Their corresponding vertices in $\mathcal{I}_{A|B}$ are (lo,lo) and (med,hi). Because these instances are in the same group but in different subgroups, there is an edge in $\mathcal{I}_{A|B}$ connecting (lo,lo) and (med,hi). After the coloring (see Figure 2.5), these two subgroups obtained the labels 1 and 2 (1= H(lo,lo) and 2= H(med,hi)) and thus constitute two examples in E_G, such that lo= G(lo,1) and med= G(lo,2).

2.2.3 Properties of single-step decomposition

Theorem 2.1 The example sets E_G and E_H obtained by single-step decomposition are coherent with E_F, i.e., every example in E_F is correctly classified using the functions H and G.

Proof: Coherence is guaranteed by the E_G and E_H derivation method. Namely, for $e_i \in E_F$ and $y_i = F(e_i)$, the value $c_j = H(B_i)$ is assigned only to the $\mathcal{P}_{A|B}$ columns with the row entries that are either undefined or have an instance with a class value y_i. Therefore there exist no instance e_j in $\mathcal{P}_{A|B}$ in the same row $A(e_i)$ and in the column labeled c_j that would have $F(e_j) \neq F(e_i)$.

Theorem 2.2 $\nu(A|B)$ obtained by optimal coloring of $\mathcal{I}_{A|B}$ is the lowest number of values for c_j to guarantee the coherence of example sets E_G and E_H with respect to example set E_F.

Proof: Suppose that $\mathcal{I}_{A|B}$ is colored with less than $\nu(A|B)$ colors. Then, at least two non-compatible $\mathcal{P}_{A|B}$ columns are assigned the same label and for these by the Definition 2.4 exist two examples $e_i, e_j \in F$ with $F(e_i) \neq F(e_j)$ and $A(e_i) = A(e_j)$. Since $H(e_i) = H(e_j)$ then $G(A(e_i), c) = G(A(e_j), c)$, which is a contradiction and at least one instance in E_F is miss-classified.

Theorem 2.3 Let N_G, N_H, and N_F are the numbers of examples in E_G, E_H, E_F, respectively. Sets E_G and E_H are derived by decomposition of E_F using the attribute partition $A|B$. Then, E_G and E_H use less attributes than E_F ($||B|| < ||X||$ and $||A||+1 < ||X||$, where X is the initial attribute set) and include fewer or equal number of examples ($N_G \leq N_F$ and $N_H \leq N_F$).

Proof: The lower number of attributes for new example sets follows from the requirement and type of decomposition we are using: $||A|| + ||B|| = ||X||$, $||A|| \geq 1$ and $||B|| \geq 1$. N_H is equal to the number of non-empty columns in $\mathcal{P}_{A|B}$ and thus less than or equal to the number of examples in E_F. Suppose, as described above, a sorted set E_F is used to derive E_G. Then, N_G is equal to the number of subgroups, and this can not be greater than the number of examples in E_F.

2.3 Decomposition algorithm

The decomposition aims to discover a hierarchy of concepts described with example sets that are overall less complex than the initial one. Since an exhaustive search is prohibitively complex, the decomposition uses a suboptimal iterative algorithm (Algorithm 3.1).

Input: Set of examples E_{F_0} describing a single output concept
Output: Its hierarchical decomposition

```
initialize $\mathcal{E} \leftarrow \{E_{F_0}\}$
initialize $j \leftarrow 1$
while $\mathcal{E} \neq \emptyset$
    arbitrarily select $E_{F_i} \in \mathcal{E}$ that partially specifies $c_i = F_i(x_1, \ldots, x_m)$
    $\mathcal{E} \leftarrow \mathcal{E} \setminus \{E_{F_i}\}$
    $A_{best}|B_{best} = \arg\min_{A|B} \Psi(A|B)$,
        where $A|B$ are all possible partitions of $X =< x_1, \ldots, x_m >$
        such that $A \cup B = X$, $A \cap B = \emptyset$, and $||B|| \leq b$
    if $E_{F_i}$ is decomposable using $A_{best}|B_{best}$ then
        decompose $E_{F_i}$ to $E_G$ and $E_{F_j}$, such that $c_i = G(A_{best}, c_j)$ and $c_j = F_j(B_{best})$
                and $E_G$ and $E_{F_j}$ partially specify $G$ and $F_j$, respectively
        $E_{F_i} \leftarrow E_G$
        if $||A_{best}|| > 1$ then $\mathcal{E} \leftarrow \mathcal{E} \cup \{E_{F_i}\}$ end if
        if $||B_{best}|| > 2$ then $\mathcal{E} \leftarrow \mathcal{E} \cup \{E_{F_j}\}$ end if
        $j \leftarrow j + 1$
    end if
end while
```

Algorithm 2.1 The decomposition algorithm

In each step the algorithm tries to decompose a single example set of the evolving structure.

It evaluates all possible partitions of the attributes and selects the best one. This step requires a so-called *partition selection measure* denoted with $\Psi(A|B)$. The best partition with the lowest value of this measure is then selected. The decomposition algorithm will decompose E_{F_i} and the function F_i it partially represents only if its decomposed functions G and H are overall less complex than F_i. In such case we say that example set E_{F_i} is decomposable using partition $A|B$. The decomposability is evaluated using the *decomposability criterion.* If the best partition found satisfies this criterion, then this partition is used to decompose E_{F_i} to sets E_G and E_H.

2.3.1 Partition selection measures

We will denote the partition selection measures with $\Psi(A|B)$. The best partition is the one that minimizes this measure:

$$A_{best}|B_{best} = \arg\min_{A|B} \Psi(A|B) \tag{2.1}$$

This section introduces three partition selection measures, one based on column multiplicity of partition matrix ($\Psi_{\mathrm{CM}}(A|B)$) and the other two based on complexity of resulting functions ($\Psi_{\mathrm{C}}(A|B)$ and $\Psi_{\mathrm{SC}}(A|B)$).

Column multiplicity (CM)

This is the simplest partition selection measure and is equal to $\nu(A,B)$, i.e., a column multiplicity of partition matrix $\mathcal{P}_{A|B}$. The idea for this measure came from the practical experience with the DEX decision support system (Bohanec & Rajkovič 1990). There, the hierarchical system of decision tables is constructed manually and it has been found that decision tables that describe concepts with a small number of values are easier to construct and interpret. Formally,

$$\Psi_{\mathrm{CM}}(A|B) = \nu(A|B) \tag{2.2}$$

Example 2.4 For the partitions in Figure 2.4 the ν-based partition selection measures are 3, 5, and 6 respectively. As expected, the best partition according to Ψ_{CM} is therefore $\langle x_1\rangle|\langle x_2, x_3\rangle$.

Complexity-based measures

Let F be defined on attributes $x_i \in X_F$ with class variable y_F. In this attribute-class space, there are a total of

$$N(X_F, y_F) = ||y_F||^{\prod_{x_i\in X_F} ||x_i||} \tag{2.3}$$

possible functions, where $||y_F||$ and $||x_i||$ are the cardinalities of value sets of y_F and x_i, respectively. The number of bits to encode F is therefore

$$\Theta_{\mathrm{C}}(F) = \log_2 N(X_F, y_F) = (\log_2 ||y_F||) \prod_{x_i \in X_F} ||x_i|| \tag{2.4}$$

The Θ_{C} *complexity-based partition selection measure* is defined as a sum of complexities of G and H: $\Psi_{\mathrm{C}}(A|B) = \Theta_{\mathrm{C}}(G) + \Theta_{\mathrm{C}}(H)$.

A similar measure was indeed proposed by Abu-Mostafa (1988) as a general measure of complexity and used in the decomposition of Boolean functions. The authors called this measure a *decomposed function cardinality* (DFC) and computed it as $\prod_{x_i \in X_F} ||x_i||$. Although its ability to guide the decomposition of Boolean functions has been illustrated in several references including (Ross, Noviskey, Gadd & Goldman 1994), DFC has been recently criticized by Perkowski (1995) for deficiencies in handling some classes of functions including multi-output symmetric functions. Moreover, we are not aware of any study of the applicability of DFC measures for the decomposition of example sets that use multi-valued attributes and classes.

Θ_{C} seems an appropriate complexity measure for function G. However, it can not be used as such for H because of the following. When decomposing $y = F(X)$ to $y = G(A, c)$ and $c = H(B)$, we assign a single value from the set $\{1, \dots, \nu(A|B)\}$ to each of the columns of partition matrix $A|B$. But, each of these values has to be used in at least one example from E_H. In other words, from $|y|^{\prod_{x_i \in B} ||x_i||}$ different functions we have to exclude all those that use less than $||c||$ values. The number of different functions with exactly $||c||$ possible values is therefore $N(\nu(A|B))$, where N is defined as:

$$\begin{aligned} N(x) = &\; x^{\prod_{x_i \in B} ||x_i||} - \sum_{i=1}^{||c||-1} \binom{||c||}{i} N(x-1) \\ N(1) = &\; 1 \end{aligned} \tag{2.5}$$

Furthermore, since the actual label (value of c) of the column in $\mathcal{P}_{A|B}$ is not important, there are $\nu(A|B)!$ equivalent assignments and therefore $||c||!$ equivalent functions H. H therefore uniquely represents $N(\nu(A|B))/(\nu(A|B)!)$ functions with exactly $\nu(A|B)$ output values, and the number of bits needed to encode H is therefore:

$$\Theta'_{\mathrm{C}}(H) = \log_2 N(||c||) - \log_2(||c||!) \;\; \text{bits} \tag{2.6}$$

The Θ_{C} and Θ'_{C}-based partition selection measure is then:

$$\Psi_{\mathrm{C}}(A|B) = \Theta_{\mathrm{C}}(G) + \Theta'_{\mathrm{C}}(H) \tag{2.7}$$

Example 2.5 For the partitions in Figure 2.4, the corresponding complexity-based partition selection measures are: $\Psi_{\mathrm{C}}(\langle x_1 \rangle | \langle x_2, x_3 \rangle) = 20.76$ bits, $\Psi_{\mathrm{C}}(\langle x_2 \rangle | \langle x_1, x_3 \rangle) = 27.68$ bits, and $\Psi_{\mathrm{C}}(\langle x_3 \rangle | \langle x_1, x_2 \rangle) = 30.39$ bits. The preferred partition is therefore $\langle x_1 \rangle | \langle x_2, x_3 \rangle$.

The implementation of Ψ_{C} may be quite inefficient because of the complex and recursive formula (2.5). As an alternative, we introduce a *simplified complexity-based partition selection*

	$\langle x_1\rangle\vert\langle x_2,x_3\rangle$			$\langle x_2\rangle\vert\langle x_1,x_3\rangle$			$\langle x_3\rangle\vert\langle x_1,x_2\rangle$		
	$\Psi(A\vert B)$	$\Theta(F)$		$\Psi(A\vert B)$	$\Theta(F)$		$\Psi(A\vert B)$	$\Theta(F)$	
C	28.61	28.53	✗	25.04	28.53	✓	20.76	28.53	✓
SC	29.84	28.53	✗	26.44	28.53	✓	21.19	28.53	✓

Table 2.2: Complexity measures and decomposability for partitions of Figure 2.4.

measure Ψ_{SC}, which is similar to Ψ_{C} except that for H it does not subtract the cases where H uses less than $||c||$ values. Ψ_{SC} uses the following complexity measures:

$$\Theta_{\mathrm{SC}}(F) = (\prod_{x_i \in X_F} ||x_i||) \log_2 ||y|| \text{ bits} \tag{2.8}$$

$$\Theta'_{\mathrm{SC}}(H) = (\prod_{x_i \in B} ||x_i||) \log_2 ||c|| - \log_2(||c||!) \text{ bits} \tag{2.9}$$

Ψ_{SC} partition selection measure is then defined as $\Psi_{\mathrm{SC}}(A|B) = \Theta_{\mathrm{SC}}(G) + \Theta'_{\mathrm{SC}}(H)$.

2.3.2 Decomposability criteria

The decomposition algorithm will decompose E_F and the function F it partially represents only if its decomposed functions G and H are overall less complex than F. Therefore, the partition $A|B$ can be used to decompose E_F to E_G and E_H if and only if $\Psi_{\mathrm{C}}(A|B) < \Theta_{\mathrm{C}}(F)$, or, using a simplified complexity-based measures if and only if $\Psi_{\mathrm{SC}}(A|B) < \Theta_{\mathrm{SC}}(F)$. We say that example set E_F is *decomposable* if there exists a partition $A|B$ with this property.

Note that because Ψ_{CM} is not based on function complexity, it can not be similarly used for decomposability criteria. Therefore, when using Ψ_{CM} for partition selection criteria, either Ψ_{C} or Ψ_{SC} is used to determine decomposability.

Example 2.6 The application of both Ψ_{C} and Ψ_{SC} decomposability criteria on the decomposition of example set from Table 2.1 are compared in Table 2.2. Neither one allows the decomposition using the partition $\langle x_1\rangle|\langle x_2, x_3\rangle$. Of other two partitions, the partition $\langle x_3\rangle|\langle x_1, x_2\rangle$ is the best partition according to both partition selection measures.

2.3.3 Joint complexity of decomposed function

Complexity-based measures can be used to asses the overall complexity of the functions in the concept hierarchy. This complexity is equal to the sum of complexities of functions in the hierarchy, where Θ_{C} (or Θ_{SC}) is used to estimate the complexity of the function at the root of the hierarchy and Θ'_{C} (or Θ'_{SC}) is used for all other functions.

2.3.4 Complexity of decomposition algorithm

The time complexity of a single step decomposition of E_F to E_G and E_H, which consists of sorting of example set, deriving the incompatibility graph and coloring it, is $O(N \log N) + O(Nk) + O(k^2)$ where N is the number of examples in E_F and k is the number of vertices in $\mathcal{I}_{A|B}$. For any bound set B, the upper bound of k is

$$k_{max} = (\max_{x_i \in X} ||x_i||)^b \tag{2.10}$$

where $b = ||B||$. The number of disjoint partitions considered by decomposition when decomposing E_F with m attributes is

$$\sum_{j=2}^{b} \binom{m}{j} \leq \sum_{j=2}^{b} (\frac{\mathrm{e}m}{j})^j = O(m^b) \tag{2.11}$$

The highest number of $n-2$ decompositions is required when the hierarchy is a binary tree, where n is the number of attributes in the initial example set. The time complexity of the decomposition algorithm is thus

$$O\Big((N \log N + Nk_{max} + k_{max}^2) \sum_{m=3}^{n} m^b\Big) = O\Big(n^{b+1}(N \log N + Nk_{max} + k_{max}^2)\Big) \tag{2.12}$$

Therefore, the algorithm's complexity is polynomial in N, n, and k_{max}. Note that the bound b is a user-defined constant. This analysis clearly illustrates the benefits of setting b to a sufficiently low value. In our experiments, b was usually set to 3.

2.4 Unsupervised and supervised approach to decomposition

The concept hierarchy may be obtained with or without human supervision. In the first case, the user is involved in deciding if, which, and how to decompose the example sets of the evolving hierarchical system. The second case is an algorithmic implementation of successive simple decomposition steps where partition selection measures and decomposition criteria as defined above are used. The user is involved in unsupervised decomposition only to set its parameters (bound b on cardinality of the set B) and specifies which measures and criteria to use.

Unsupervised decomposition discovers a hierarchical structure of concepts automatically, i.e., without user's interaction. Although the structure and discovered functions may be optimal with respect to used complexity measures, the essential drawback of the approach is that the user may prefer a different structure, i.e., a different dependency of concepts and attributes. Although such a hierarchy and corresponding functions may be more complex than those discovered by unsupervised decomposition, they can be more comprehensible and introduce recognizable concepts.

Both arguments are related to “brain compatibility” as introduced by Shapiro (1987). Decomposition should not only address the complexity, but rather provide methods to assist the user in the discovery of the structure and underlying principles of the domain. We therefore require the supervised decomposition to support the following operations:

- Given a user-defined hierarchical structure of concepts, derive the associated example sets.
- Allow the user to control the decomposition process by selecting the example set to decompose next.
- Given a example set, allow the user to select a partition for decomposition by providing a list of partitions and their associated partition selection measures.

2.5 Concept hierarchy as a classifier

A concept hierarchy can be used to classify new and possibly previously unseen examples. When an example is presented to a concept hierarchy, the target concept value is derived by bottom-up derivation where the lower-level concepts are derived first and are then used to derive the values of the higher-level concepts.

For each concept in the hierarchy, given the values of its attributes, an example is found that has the same value of attributes and its class is assigned to that concept. If such example is not found, we use a *default rule*: assign the concept a class that is the most frequently used in the set of its examples. Ties are resolved arbitrarily. According to the experimental evaluation (section 2.7.6) the default rule does help when the training examples rather sparsely cover the attribute set, but with a better coverage the role of default rule becomes minor.

2.6 Implementation

The minimal-complexity decomposition is implemented within the program HINT(Hierarchy INduction Tool). HINTis written in the C programming language and runs in the UNIX environment.

2.7 Experimental evaluation

The experimental evaluation addresses the classification accuracy of HINT and its ability to derive a comprehensible and meaningful structure, possibly similar to the anticipated one.

In most of the cases, the classification accuracy was assessed by means of learning curves. The datasets were split to training and test sets of sizes p and $1 - p$, respectively, for p from 10% to 90%. HINT derived a concept hierarchy and corresponding classifier using the examples in the training set and was tested for classification accuracy on the test set. For each p, the results are the average of 10 randomly chosen splits.

The learning curve is compared to the one obtained by C4.5 (Quinlan 1993) inductive decision tree learner run on the same data. Although we could choose from a number of different learners, and possibly for a different learner with optimized performance for a specific domain, we have decided for C4.5 for the following reasons: (1) it is one of the most known and used learning algorithms, (2) it performs well in terms of classification accuracy when compared to other learners (Michie, Spiegelhalter & Taylor 1994). We have used the release 8 of the C4.5 program, and run it with default options except for `-m1`, which was observed to obtain a better classification accuracy than the default `-m2`. Accuracy is measured on unpruned decision trees for the same reason. For each p, the significance of the difference between C4.5 and HINT is determined using a t-test with $\alpha = 0.01$ (99% confidence level). Learning curves use symbols $\circ$ (HINT) and $\diamond$ (C4.5) when the difference is not significant, $\bullet$ when HINT is significantly better, and $\blacklozenge$ when C4.5 is significantly better.

The derived concept hierarchies were, where possible, compared to the anticipated ones. The comparison was either qualitative or by means of structure dissimilarity coefficient SDC (Appendix A) to assess the similarity of the discovered structure and the structure that is anticipated. For a qualitative comparison, the comprehensibility of the discovered structure and the appropriateness of intermediate concepts were evaluated. The appropriateness of the discovered structure was further evaluated by means of the complexity of its functions by the simplified complexity measure Θ_{SC}.

We have considered the training sets of three different types:

- Artificially generated training sets using binary and multi-valued functions. The structure and intermediate concepts for these domains are anticipated.

- The training sets obtained from multi-attribute decision models originally developed using DEX (Bohanec & Rajkovič 1990). The DEX models are hierarchical, so both structure and intermediate concepts for these domains are known.

- Several training sets from machine learning repository (Murphy & Aha 1994) for which the structure is not known to us.

The first group of examples is to illustrate the main characteristics of decomposition method: discovery of concepts and functions. The examples also illustrate the method's ability to learn from an incomplete set of examples. DEX examples are specifically interesting to evaluate the decomposition method on more complex domains and to assess the ability of decomposition to reconstruct multi-attribute decision models. Additional convenience of DEX examples is the availability of the decision support expert (Marko Bohanec) who was involved in the development of the models, for the evaluation of comprehensibility and appropriateness of the structures discovered by decomposition. The experiments with domains from machine learning repository mainly focused on the evaluation of the classification accuracy.

Throughout this section we use an unsupervised decomposition algorithm. The decomposition program HINT was run on a HP J210 workstation. If not stated otherwise, the partition selection measure used is the column multiplicity, the simplified complexity measure Ψ_{SC} is used as a decomposability criterion, and the bound of the number of attributes in the bound set is set to 3.

2.7.1 Artificial domains

For this set of experiments, we generated several simple data sets based on predefined functions and tried to reconstruct them. These functions are listed in Table 2.3. They are defined using standard functions XOR, OR, AND, MIN, and MAX. The function AVG used with MM3, MM4, and MM5 computes the average of its arguments and rounds it to the closest integer. Prior probabilities of class values for each domain are given in Table 2.4. In all cases the examples in the data sets completely cover the attribute space.

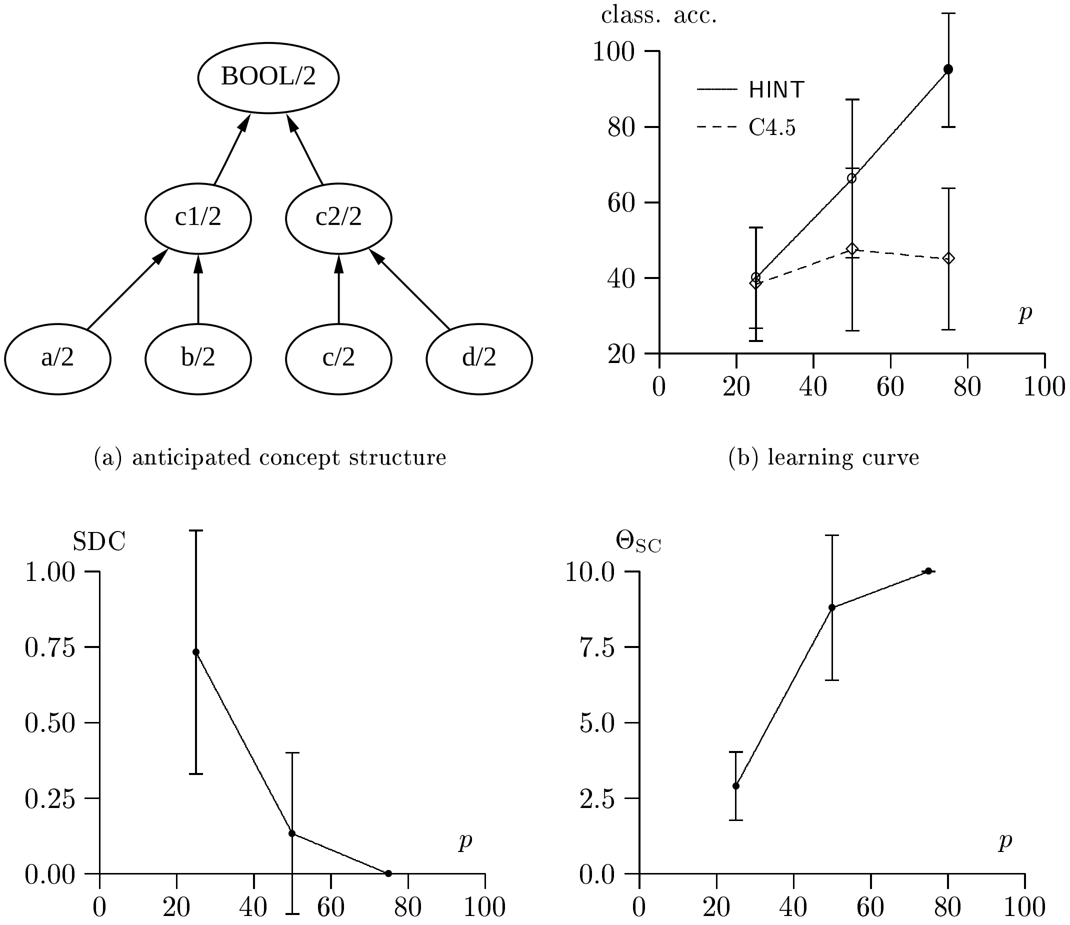
Name	Function	Number and type of attributes	Data set size
BOOL	$y = (x_1 \text{AND } x_2)\text{OR}(x_3 \text{XOR } x_4)$	4 binary attributes	16
MM3 MM4 MM5	$y = \text{MIN}(x_1,$ $\text{AVG}(x_2, \text{MAX}(x_3, x_4), x_5))$	5 k-valued attributes $k = 3, 4, 5$	243 (for $k = 3$) 1024 (for $k = 4$) 3125 (for $k = 5$)
PAL2 PAL3	$y =$ $palindrome(x_1, x_2, x_3, x_4, x_5, x_6) =$ $(x_1 = x_6)\text{AND}(x_2 = x_5)$ $AND(x_3 = x_4)$	6 k-valued attributes $k = 2, 3$	64 (for $k = 2$) 729 (for $k = 3$)

Table 2.3: Datasets based on binary and multi-valued functions.

Dataset	Prior class probabilities
BOOL	0/0.375, 1/0.625
MM3	0/0.383, 1/0.543, 2/0.074
MM4	0/0.268, 1/0.416, 2/0.291, 3/0.025
MM5	0/0.208, 1/0.293, 2/0.342, 3/0.146, 4/0.011
PAL2	0/0.875, 1/0.125
PAL3	0 0.963 1/0.037

Table 2.4: Priori probabilities of classes in the artificial datasets.

Figure 2.6 shows the results for BOOL. The anticipated structure (Figure 2.6.a) uses two intermediate concepts $c_1 = x_1 \mathrm{AND} x_2$ and $c_2 = x_3 \mathrm{XOR} x_4$, where the final concept is $y = c_1 \mathrm{OR} c_2$. Because of the small size of this domain, the training sets used either 25%, 50%, or 75% samples from the original data set. The learning curve (Figure 2.6.b) indicates that decomposition learned significantly better than C4.5 at 75%, whereas when using training sets of lower sizes the differences are not significant. Also at 75% the decomposition discovered the anticipated structure in all the 10 trials. The joint complexity of the functions in a classifier monotonically increases with the size of the training set. Namely, for sets with lower number of examples, fewer intermediate concepts and fewer attributes may be required to describe them, and consequently fewer and simpler example sets are discovered (see Chapter 3 for a formal discussion of such cases).



(a) anticipated concept structure

(b) learning curve

(c) dissimilarity with the anticipated structure

(d) joint complexity of discovered functions

Figure 2.6: Results for learning the BOOL function.

For all three MM functions, the anticipated structure is shown in Figure 2.8.a. The structure to which decomposition converged to with an increasing number of training examples is slightly different, with the intermediate concept c_1 of anticipated structure decomposed to two intermediate concepts. Although the concept c_2 of such structure uses relatively higher number of values (5 for MM3, 7 for MM4, 9 for MM5) when compared to other attributes and concepts in the structure (k for MMk), Θ_{SC} of such structure is lower than the for the anticipated (66.1 instead of 66.1 for MM3, 143.4 instead of 182.8 for MM4, 267.5 instead of 392.5 for MM5).

The experimental results for MM functions are shown in Figure 2.7. For larger domains, HINT's classificiation accuracy converges faster to 100%. Similar is true for the convergence to the anticipated structure. In comparison with C4.5, HINT is significantly more accurate for all the training set sizes for MM5, for all but 10% size for MM4, and for all sizes larger or equal to 50% for MM3.

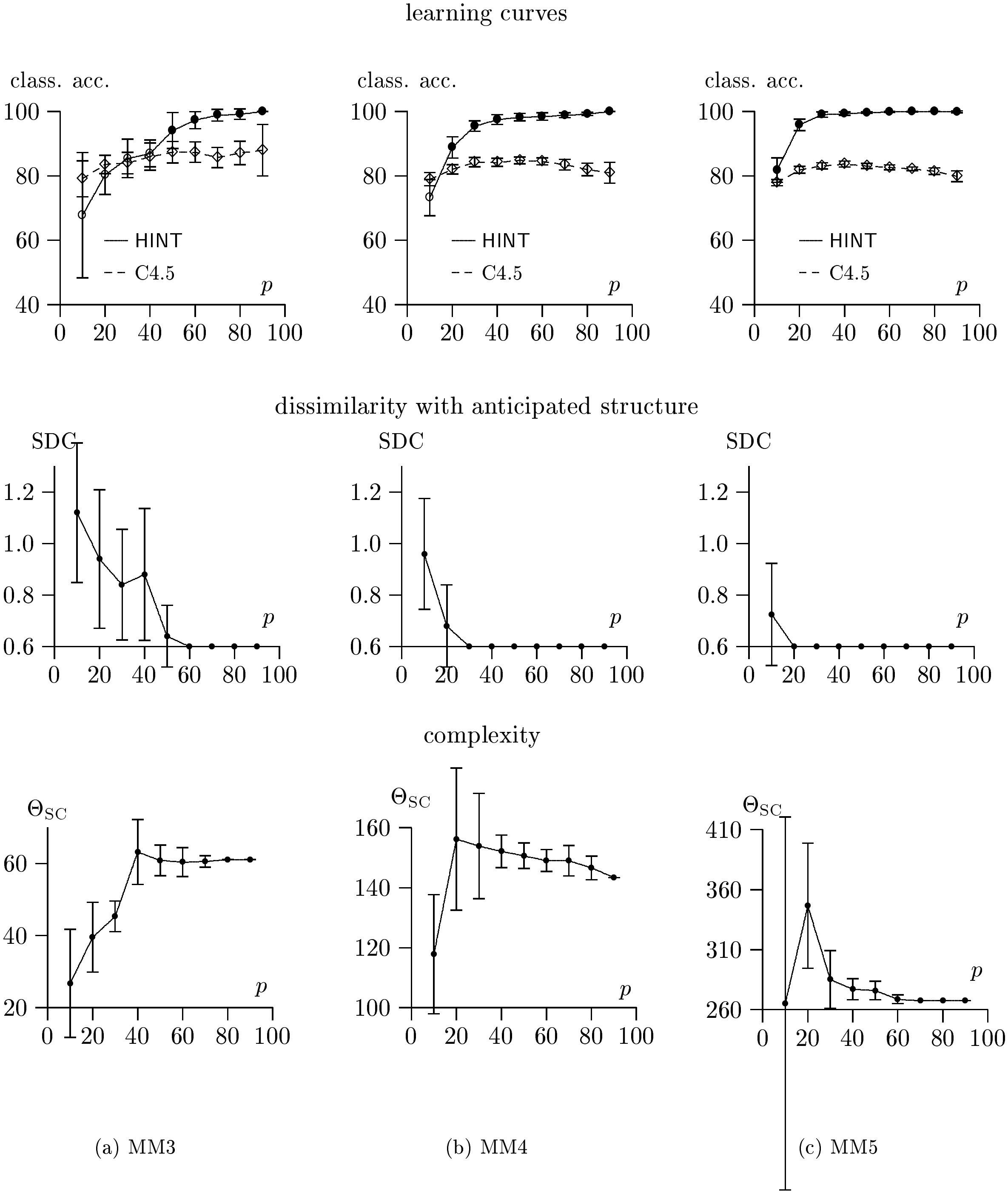
An interesting phenomenon is the dependency of the joint complexity of functions Θ_{SC} on the size of the training set. In all three cases, it can be observed that Θ_{SC} has a local extreme (maximum). This can be contributed to the interplay of two effects:

- with a training set that sparsely covers the attribute space, the discovered structure may use a lower number of atributes and a lower number of intermediate concepts which may require lower number of values to describe the training set,
- when increasing the training set size the decompositon may have more evidence for column (in)compatibility of partition matrices considered to find better structures with simpler concepts.

Similar conclusions as for MM function can be drawn for PAL2 and PAL3 (see Figure 2.9. The anticipated structure is either one of $y = ((x_1 = x_6)\mathrm{AND}(x_2 = x_5))\mathrm{AND}(x_3 = x_4)$, $y = (x_1 = x_6)\mathrm{AND}((x_2 = x_5)\mathrm{AND}(x_3 = x_4))$, or $y = ((x_1 = x_6)\mathrm{AND}(x_3 = x_4))\mathrm{AND}(x_2 = x_5)$. HINT performs significantly better than C4.5 for trainig set sizes of 60% or larger for PAL2 and 30% or larger for PAL3. It needs at least 70% of examples from the original dataset to discover one of the anticipated structures for PAL2, and at least 40% for PAL3.

2.7.2 DEX domains

An area where concept hierarchies have been used extensively is decision support in problems, in which an option is to be selected from a list of given options that best satisfies the aims or goals of the decision maker. The decision is made on the basis of the evaluation of options by a multi-attribute hierarchical model. In addition to evaluation, such models are used also for various analyses and simulations of options. Most common are the models that use numerical values and analytically expressed functions (Mallach 1994). A typical example of such an approach is Analytic Hierarchy Process (Saaty 1993). In all cases the structure of the model is defined manually. However, the acquisition of functions is well supported by various methods based on interactive dialogs, graphs, and tables.



learning curves

dissimilarity with anticipated structure

complexity

(a) MM3

(b) MM4

(c) MM5

Figure 2.7: Results of learning for MM functions.

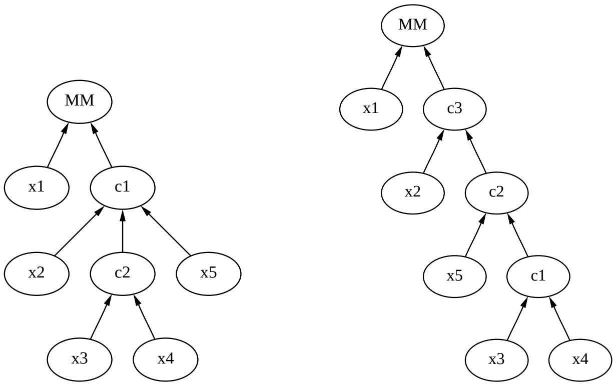
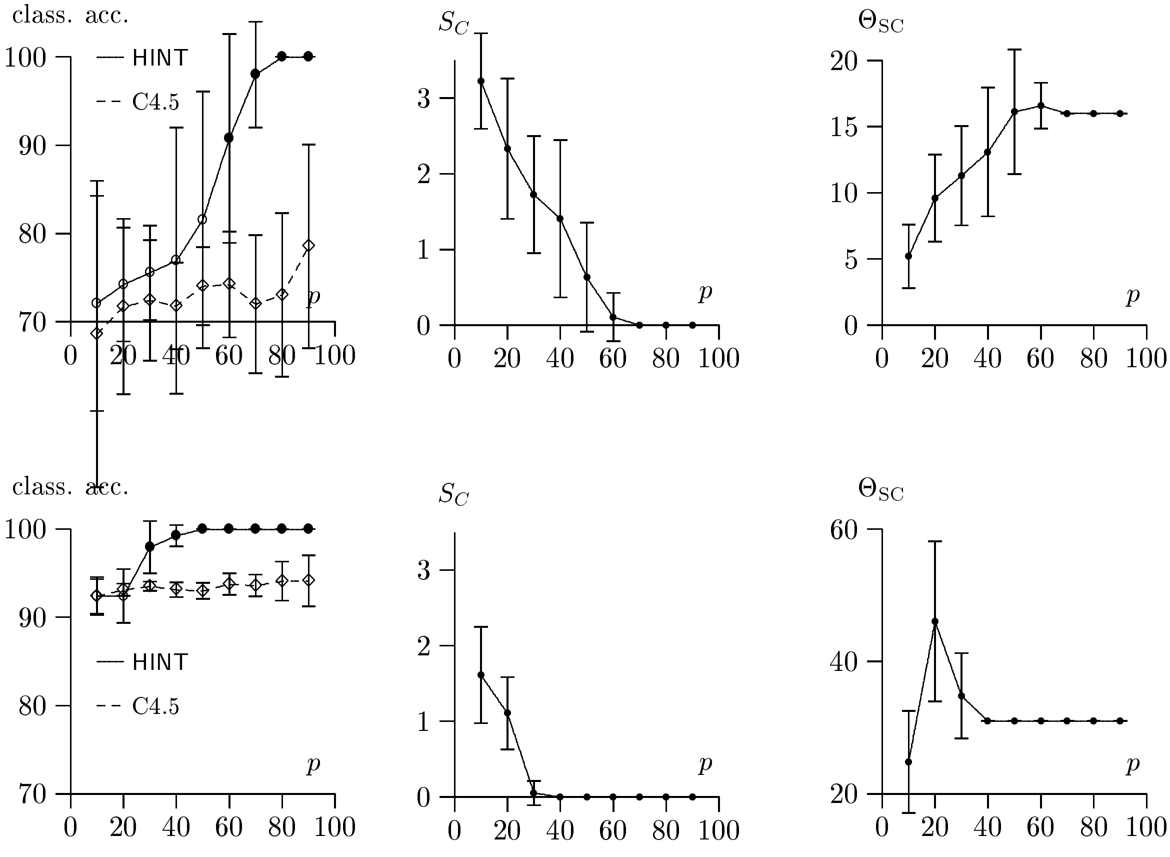


Figure 2.8: The anticipated structure for MM functions (left) and the structure to which decomposition converges with an increasing number of training examples (right).



(a) learning curves

(b) similarity with anticipated structure

(c) complexity

Figure 2.9: Results for learning the PAL2 (top) and PAL3 (bootom) functions.

DEX (Bohanec & Rajkovič 1990) is a multi-attribute decision support system that has been extensively use to solve realistic decision making problems. DEX uses categorical attributes and expects the structure and the functions to be given by the expert. The formalism used to describe the resulting model and its interpretation is essentially the same as those derived through the decomposition process described in this paper. This makes models developed by DEX ideal benchmarks for the evaluation of decomposition.

For this experiment, we have chosen six existing DEX models:

CAR: A model for evaluating cars based on their price and technical characteristics. This simple model was developed for educational purposes and is described in (Bohanec & Rajkovič 1988).

EMPLOY1 and EMPLOY2: These are simplified versions of the models that were developed with DEX for a common problem of personnel management selecting the best candidate for a particular job. While the realistic models that were practically used in several mid- to large-size companies in Ljubljana and Sarajevo consisted of more than 40 attributes, the simplified versions used here have only 7 and 9 attributes, respectively. EMPLOY1 composes the attributes into education of the candidate, age and experience, and personal characteristics. In EMPLOY2, the latter is replaced by other characteristics. EMPLOY1 is presented in (Bohanec, Urh & Rajkovič 1992), while EMPLOY2 is previously unpublished.

NURSERY: This model was developed in 1985 (Olave, Rajkovič & Bohanec 1989) to rank applications for nursery schools. It was used during several years when there was excessive enrollment to these schools in Ljubljana, and the rejected applications frequently needed an objective explanation. The final decision depended on three subproblems: profession of parents and child's nursery, family structure and financial standing, and social and health conditions of the family.

HOUSING: A model to determine the priority of housing loans applications (Bohanec, Cestnik & Rajkovič 1996). This model is a part of a management decision support system for allocating housing loans that has been used since 1991 in the Housing found of the Republic of Slovenia. So far, 11 floats of loans have been completed and various models have been used to process over 20.000 applicants. Basically, the application priority is determined on applicants housing conditions, their status, and social and health conditions.

ENTERPRISE: A model developed in 1987 at the International Center for Public Enterprises, Ljubljana, for performance evaluation of public enterprises. Actually, this is a single model from a suite of models developed for this purpose and used for the evaluation of enterprises in Pakistan and Peru (Bohanec & Rajkovič 1990). The performance of enterprises is measured by three main groups of concepts: financial, economic, and social.

The goal of this experiment was to reconstruct these models from the examples. The examples were derived from the original models, where for specific combination of input attributes the class was determined by a corresponding original model. The learning set consisted of all possible combinations of input attributes' values. When less than 20.000 examples constituted the learning set, this whole set was used for training, otherwise, 20.000 examples were randomly selected. Some characteristics of the original models and corresponding training sets are given in Table 2.5 and the prior probabilities of classes are given in Table 2.6.

Domain	n	i	N	$N[\%]$
CAR	6	3	1728	100
EMPLOY1	7	3	9600	100
EMPLOY2	9	5	18000	100
NURSERY	8	4	12960	100
HOUSING	12	6	20000	10.29
ENTERPRISE	12	8	20000	2.31

Table 2.5: Some characteristics of DEX domains used in the experiments. n is the number of attributes, i the number of concepts, N the dataset size, and $N[\%]$ is its coverage of the attribute space.

Dataset	Prior probabilities
CAR	unacc/0.700, acc/0.222, good/0.040, v-good/0.038
EMPLOY1	unacc/0.942, acc/0.017, good/0.030, exc/0.012
EMPLOY2	unacc/ 0.949, acc/ 0.048, good/ 0.002, exc/ 0.002
NURSERY	unacc/ 0.333, acc/ 0.000, v-acc/ 0.025, prior/0.329, h-prior/ 0.312
HOUSING	1/0.008, 2/0.150, 3/0.092, 4/0.299, 5.1/0.099, 5.2/0.060, 5.3/0.082, 5.4/0.051, 5.5/0.159
ENT	bad/0.190, less-acc/0.429, acc/0.143, good/0.190, exc/0.048

Table 2.6: Prior probabilities of classes in the DEX datasets.

For CAR and NURSERY, the quanititative results of experiments are given in Figures 2.10 to 2.11. For DEX domains, these results are rather typical and we include the results for other four DEX domains in Appendix C. The following can be concluded:

- In terms of the classification accuracy and in almost all of the cases, HINT performs significantly better than C4.5. The only exceptions where there is no significant difference is with training sets of 10% in the CAR and ENTERPRISE domains.

- The convergence of classification accuracy to 100% is much faster for HINT than for C4.5. Moreover, while HINT usually correctly classifies all the examples in the test set when it has learned from the training set of the size higher than 30%, the accuracy of C4.5 always stays below 100%.
- For both structure disimilarity coeficient SDC and overall complexity of functions in concept hierarchy Θ_{SC} the standard deviation is quite high for small traing sets, but it decreases to 0 when learning from larger traing sets.

For each of the domains and with increasing the size of the traing set, the decomposition converges to a single and specific structure. For CAR and NURSERY, these structures are shown in Figure 2.12 and Figure 2.13, respectively, while for other DEX domains the corresponding structures are given in Appendix C. The structures discovered by HINTare presented along with the original structure of the corresponding DEX model. The qualitative comparison of the concepts structures shows that:

- The overall complexity of discovered functions is for all domains lower than of corresponding original ones (Table 2.7). This is due to the fact that the discovered models are in general more decomposed than original ones. Another reason is that original DEX models included redundancies that were removed by the decomposition. Section 3.4.1 further discusses the redundancy issue.
- For CAR, EMPLOY1, NURSERY, and HOUSE the structures derived are very similar to the original ones, with a distinction that each of them can be obtained from the original one by further decomposing one or several of its concepts. For example, the original intermediate concept `comfort`$= F($`doors,persons,lug_boot`$)$ of CAR is represented with $c_1 = F'$`lug_boot,`$c_4)$ and $c_4 = G($`doors,persons`$)$, i.e., the discovered structure uses one additional intermediate concept c_4. Similarly, additional intermediate concepts for EMPLOY1 is c_5, for NURSERY c_6, and for HOUSING c_5, c_9, c_3, and c_10.
- Among the discovered structures, it is that of EMPLOY2 that least resembles to the original one. HINT discovered the intermediate concepts `formal` (c_2) and `work_app` (c_4), but other intermediate concepts used do not relate to the original ones. Still, the discovered structure has functions that are overall less complex than the original ones. However, this affects the transparency of the structure since the attributes that were combined together, such as **`manag_ab`** and **`health`**, do not represent a concept easy to comprehend. This is a nice example that illustrates that the most compact representation need not be the most comprehensible. Clearly, supervised decomposition is preferable for such cases.

Overall, the unsupervised decomposition algorithm performed extremely well when discovering DEX models. The derived structures were the same or only slightly different to

	CAR	EMPLOY1	EMPLOY2	NURSERY	HOUSING	ENTERPRISE
original	146.245	145.098	159.947	169.403	299.895	231.883
discovered	108.680	128.513	108.323	134.259	236.493	180.752

Table 2.7: Comparison of the overall SDTIC complexity of functions in original concept structure and in the structures discovered by HINT.

CAR	EMPLOY1	EMPLOY2	NURSERY	HOUSING	ENTERPRISE
1.2 s	25 s	29 s	28 s	346 s	130 s

Table 2.8: HINT's run-times on HP J210 workstation when decomposing the complete datasets. These are also the upper bounds for the run-times when there are fewer examples in the traing sets.

original ones. This success can be contributed to the intrinsic structure and relatively large number of the training examples. In spite of the large number of training examples, the time needed for learning was small or moderate (Table 2.8).

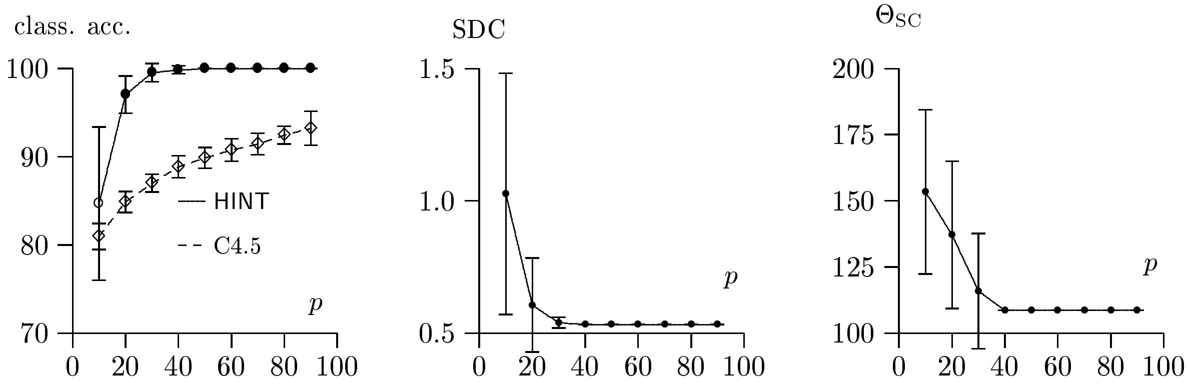


Figure 2.10: Results for the CAR domain.

2.7.3 Several domains from the ML repository

The following set of examples was taken from UCI machine learning repository (Murphy & Aha 1994):

LENSES A small domain that uses patient's age, spectacle prescription, astigmatism, and tear production rate and describes whether the patient should wear soft or hard contact lenses or no lenses at all.

SHUTTLE The 6 attribute data base is intended for determining the conditions under which autolanding would be preferable to manual control of the spacecraft.

MONK1 and **MONK2** Well-known six-attribute binary classification problems (Murphy & Aha 1994, Thrun et al. 1991). Attributes are 2 to 4-valued. MONK1 has an underlying

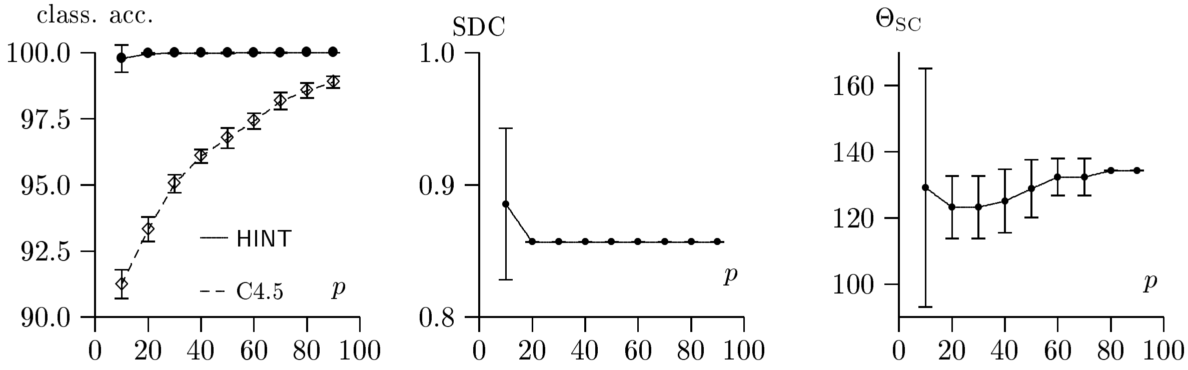


Figure 2.11: Results for the NURSERY domain.

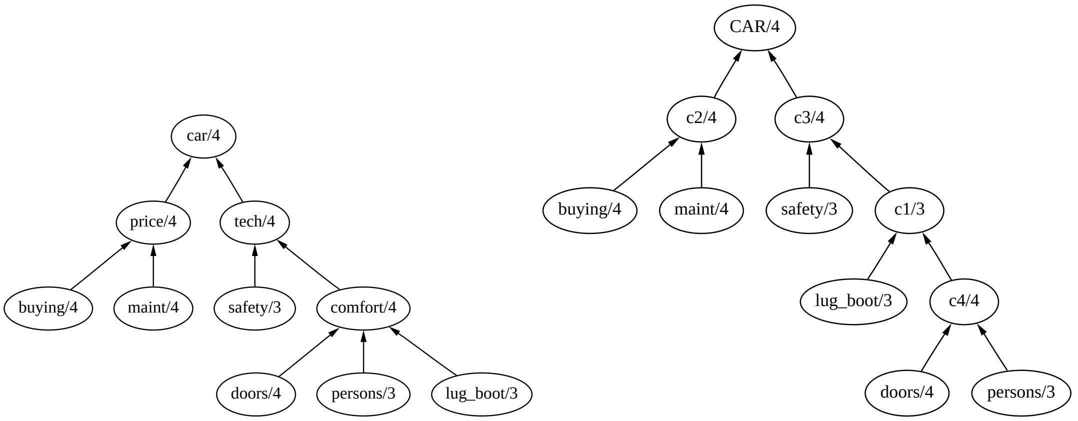


Figure 2.12: The original (left) and decomposition-derived (right) structure for the CAR domain.

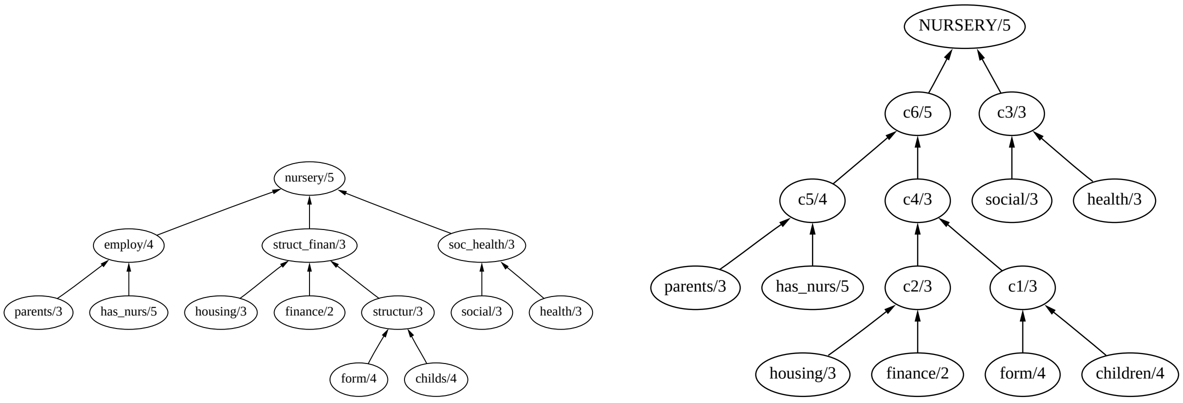


Figure 2.13: The original (left) and decomposition-derived (right) structure for the NURSERY domain.

concept `a=b OR e=1` and MONK2 the concept $x = 1$ for exactly two choices of attributes $x \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$.

The results of experiments in the form of the learning curves and overall complexities of discovered functions are given in Figures 2.14 to 2.17. While HINT and C4.5 perform similarly on LENSES and SHUTTLE, HINT is significantly better for MONK domains for $p > 20\%$. Interestingly, for LENSES and SHUTTLE the complexity curves are rather of different shape than of all other domains investigated so far and monotonically increase with higher p.

The discovered structures for each of the domains again converges to a single specific structure. For all four domains these are shown in Figure 2.18. For LENSES and SHUTTLE we did not try to interpret the structures without the domain expert. For MONK1, HINT discovered the hierarchy very similar to the anticipated one: MONK1 $= F_1(c_1, c_2)$, $c_1 = F_2(x_1, x_2)$, $c_2 = F_3(x_5, x_6)$. In F_3, x_6 is redundant and by removing it c_2 becomes x_5 while F_1 and F_2 are equal to the expected disjunctive and equality functions. How such redundancies are automatically discovered is described in Chapter 3.

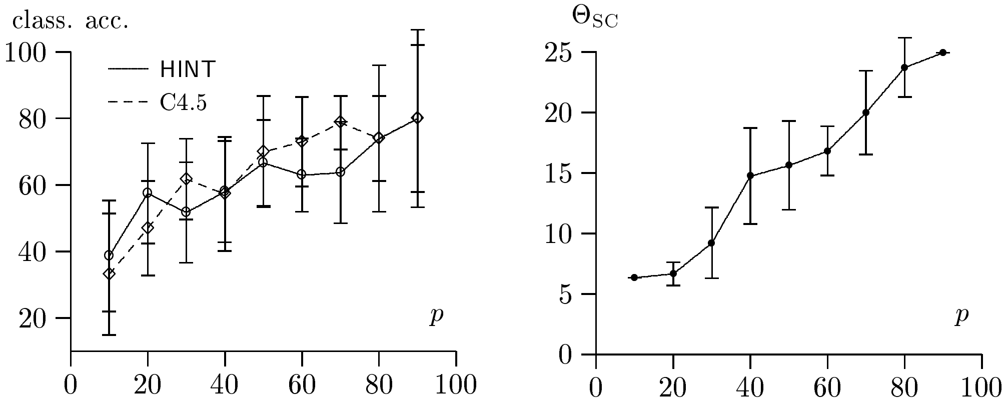


Figure 2.14: Results for learning for the LENSES domain.

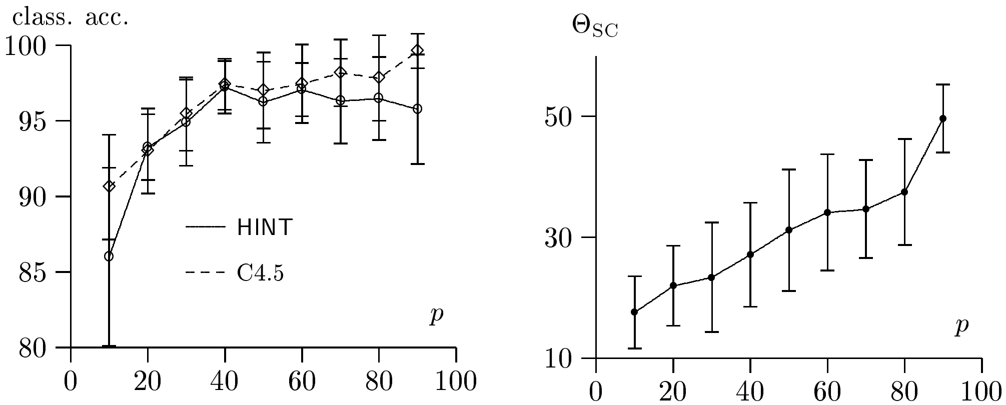


Figure 2.15: Results for learning for the SHUTTLE domain.

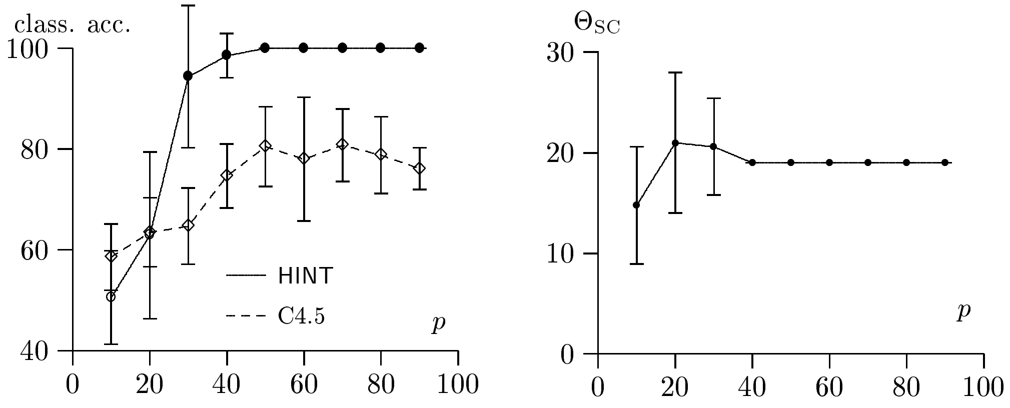


Figure 2.16: Results for learning for the MONK1 domain.

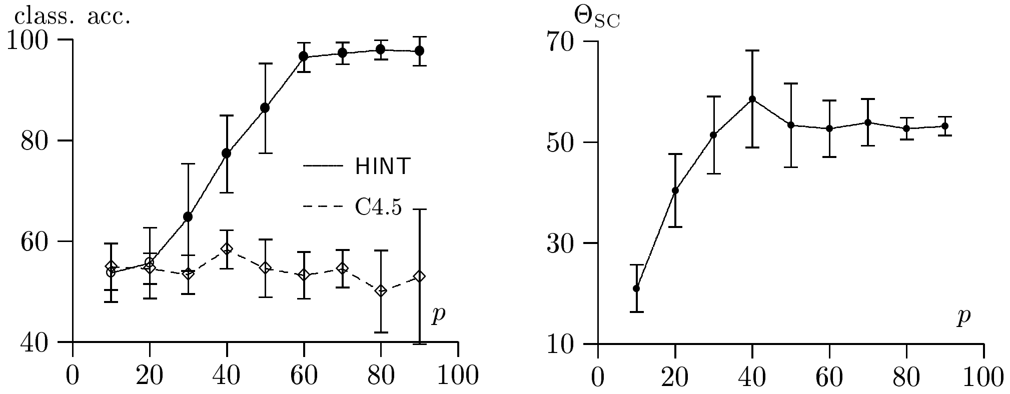
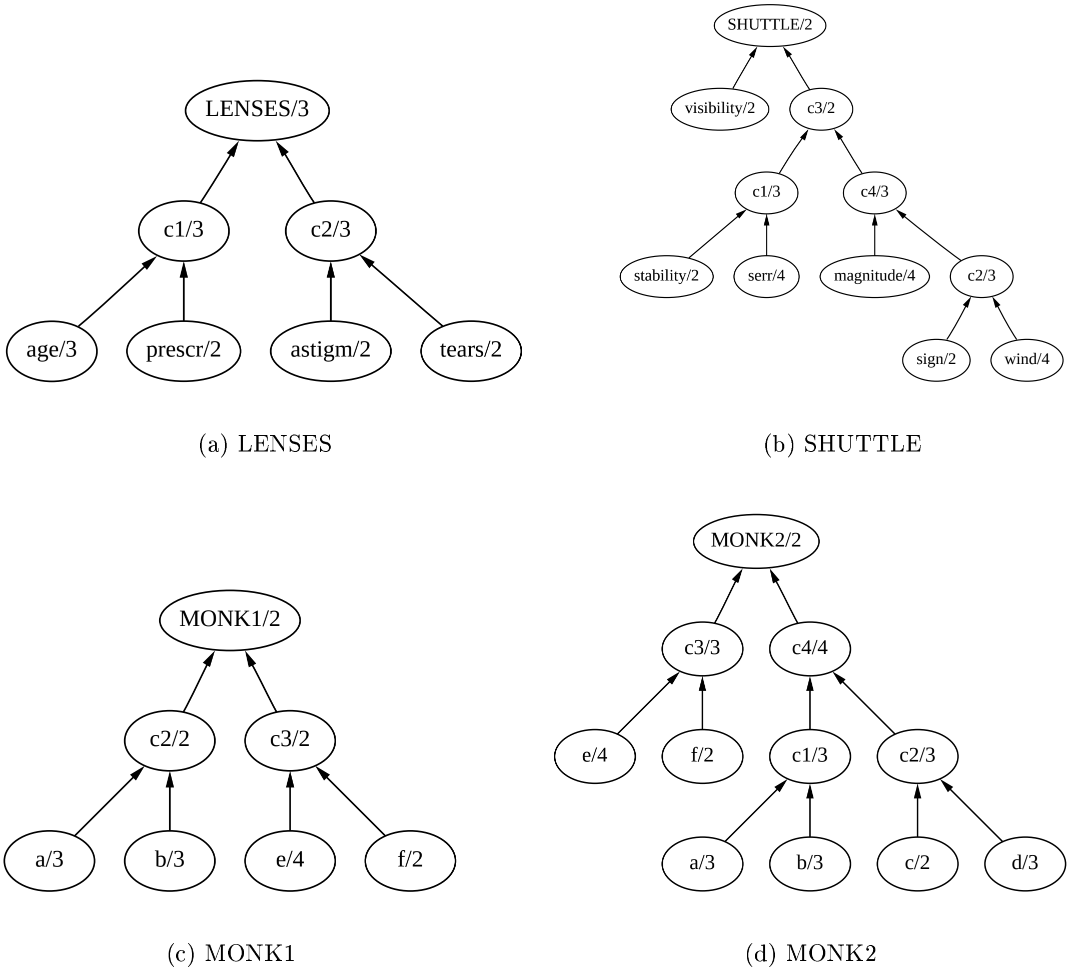


Figure 2.17: Results for learning in the MONK2 domain.

For MONK2, because of the disjunctive condition on a bound and free set, it was impossible to derive concepts comparable to the original concept definition. However, the discovered concept hierarchy is a reformulation of the target concept using functions that count 1's.



(a) LENSES

(b) SHUTTLE

(c) MONK1

(d) MONK2

Figure 2.18: Structures discovered for ML-repository domains

The results indicate that the domains LENSES and SHUTTLE are different than MONK1 and MONK2 from the viewpoint of decomposition. Namely, when compared to C4.5, for MONK1 and MONK2 HINT performs much better than in case of LENSES and SHUTTLE. Furthermore, for MONK1 and MONK2 the complexity curves settle to some value while those of LENSES and SHUTTLE this can not be observed. The possible explanation for such behavior may be that LENSES and SHUTTLE do not possess a structure that would be recognizable by decomposition and would have a positive effect on classification accuracy.

2.7.4 Further experiments on MONK1 and MONK2

HINT was further tested on the data sets for MONK1 and MONK2 that were used in the detailed study of 25 machine learning algorithms (Thrun et al. 1991). The summary of experimental results for this study are given in Appendix D. For both MONK1 and MONK2, the training set was the same as our original data set described above. The two test sets used in the study consisted of 432 examples that completely covered the attribute space.

For MONK1, the accuracy of HINT is 100%. This score was in the study (Thrun et al. 1991) achieved by 9 learners (three variants of AQ17, Assistant Professional, mFOIL, CN2, two variants of Backpropagation, and Cascade Correlation).

For MONK2, the accuracy of HINT is 97.7%, which was in the study outscored by 4 learners which achieve accuracy 100% (AQ17-DCI, two variants of Backpropagation, and Cascade Correlation). It should be noted that these results were obtained by HINT without tuning in less then 0.3 seconds of CPU time on HP J210 workstation. When, for example, the training examples were preprocessed by HINT to remove the redundant attributes (see Chapter 3), HINT missclassified just one example in the test set and thus achieved the accuracy of 99.77%.

2.7.5 Conduction-block of partially demyelinated fiber

This section demonstrates how decomposition can be used to analyze neurophysiological data. The data was obtained by using a physiologically realistic model of the myelinated axon of a neuron developed at Baylor College of Medicine in Houston. The model is given in the form of a system of multiple cross-coupled parabolic partial differential equations that are solved by an implicit numerical integration method. The model is used to predict the functional implications of neuronal structural and biophysical properties (Halter & Clark 1991, Halter, Carp & Wolpaw 1995) and has recently been employed to guide laboratory experiments and a study of pathophysiologic significance of abnormal myelination.

In this problem, we used a supervised decomposition to analyze the data used to study the influence of six nerve fiber properties (Table 2.9) to the conduction of action potential. For this study, just a part of the nerve fiber (several internodes) was demyelinated and the corresponding change of several properties was studied. The attributes and class are listed in Table 2.9. Table gives also the range of the values for `aff` and `nl`, and the ranges of orders of magnitude in which `k_conc`, `na_conc`, `scm`, and `leak` were scaled from their normal values. The data-set was generated using a realistic nerve-fiber model and consisted of 3000 random samples.

From the original dataset a set with 2543 examples was derived where each of the attributes was discretized using 5 intervals. The method used a genetic algorithm to find the discretization intervals that minimize the classification error when new example set is used to classify the instances of original set. If several examples from the original set are discretized

`block`	a two valued class indicating whether or not the nerve fiber conducts
`aff`	number of affected internodes, $[1, 6]$
`nl`	number of myelin layers in the internode, $[0, 100]$
`k_conc`	concentration of K^+ channel density in the internode, $[-2, 2]$
`na_conc`	concentration of Na^+ channel density in the internode, $[-2, 2]$
`scm`	specific conductivity of the myelin in the internode, $[-2, 2]$
`leak`	nodal leakage through all the nodes of the fiber, $[-2, 2]$

Table 2.9: Class and attributes for the conduction-block domain.

to the same set of attribute values, they are represented by a single example in the new set. Such a new example describes the class that is the majority class from the corresponding examples of the original set. The discretization method is further described in (Zupan, Halter & Bohanec to appear in 1997). When used to classify the items in the data-set, the discretized example set was found to missclassify only 7 instances of the original dataset (0.23%).

The decomposition algorithm checked all the possible partitions and the first one found was `affnlscm|k_concna_concleak` with $\nu = 12$. This partition was clearly the best one for decomposition, and to illustrate this, Table 2.10 lists ν for all the partitions with the cardinality of the bound set of 3.

This decomposition yielded the structure $\texttt{block} = G(\texttt{aff}, \texttt{nl}, \texttt{scm}, \texttt{c1})$ and $\texttt{c1} = H(\texttt{na\_conc}$, $\texttt{k\_conc}$, $\texttt{leak})$. The two example sets E_F and E_G were separately checked for decomposition. For E_F, decomposition with the measure $\nu(\texttt{affc1}|\texttt{na\_concscm}) = 16$ was found and `block=` G'`(aff,c1,c2)` was further decomposed using the partition $\nu(\texttt{aff}|\texttt{c1 c2}) = 13$. E_H could also be decomposed, but since all decomposition yielded an intermediate concept with 25 values, i.e. with one value for each combination of attributes in the bound set, it was decided that the hierarchy discovered so far would be the final one.

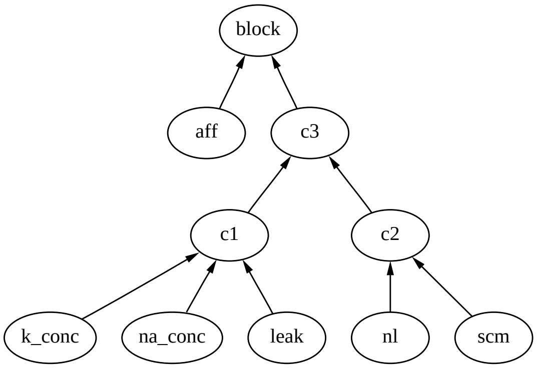


Figure 2.19: Discovered concept hierarchy for the conduction-block domain.

The discovered concept hierarchy is presented in Figure 2.19. When interpreted by the

free set	bound set	ν
aff, nl, scm	k_conc, na_conc, leak	12
nl, scm, leak	aff, k_conc, na_conc	15
k_conc, na_conc, leak	aff, nl, scm	16
nl, na_conc, scm	aff, k_conc, leak	16
k_conc, na_conc, scm	aff, nl, leak	17
k_conc, scm, leak	aff, nl, na_conc	17
aff, nl, leak	k_conc, na_conc, scm	17
nl, k_conc, scm	aff, na_conc, leak	17
na_conc, scm, leak	aff, nl, k_conc	18
aff, scm, leak	nl, k_conc, na_conc	18
aff, na_conc, scm	nl, k_conc, leak	18
aff, k_conc, leak	nl, na_conc, scm	18
aff, k_conc, scm	nl, na_conc, leak	18
aff, k_conc, na_conc	nl, scm, leak	18
aff, nl, na_conc	k_conc, scm, leak	18
aff, nl, k_conc	na_conc, scm, leak	18
nl, k_conc, na_conc	aff, scm, leak	19
aff, na_conc, leak	nl, k_conc, scm	19
nl, na_conc, leak	aff, k_conc, scm	20
nl, k_conc, leak	aff, na_conc, scm	20

Table 2.10: Partition selection measure (partition matrix column multiplicity) for undecomposed discretized condition-block example set.

domain expert (J. A. Halter), it was found that the discovered intermediate concepts are physiologically interpretable and constitute useful intermediate biophysical properties. Intermediate concept c2, for example, depends on nl and scm, the properties of the myelin sheath, which are indeed coupled within the realistic model through model-specific function $f(\mathtt{nl}, \mathtt{scm})$. c1 couples na_conc, k_conc, and leak, the axonal properties and the combined current source/sink capacity of the axon which is the driving force for all propagated action potentials.

The initial discretized set had 2543 examples, while after the decomposition the discovered concepts c1, c2, c3, and an output concept block are described with significantly simpler sets 125, 25, 184, and 65 examples. For this reason and for the reason that they represent physiologically relevant properties, they were easier to interpret than the non-decomposed sets. For instance, it was found that c2 exhibits monotonic dependency of c2 to nl and scm. Furthermore, c1 represents a principle known in physiology: for a conduction of the nerve fiber, an increase of leak will require an increase of na_conc.

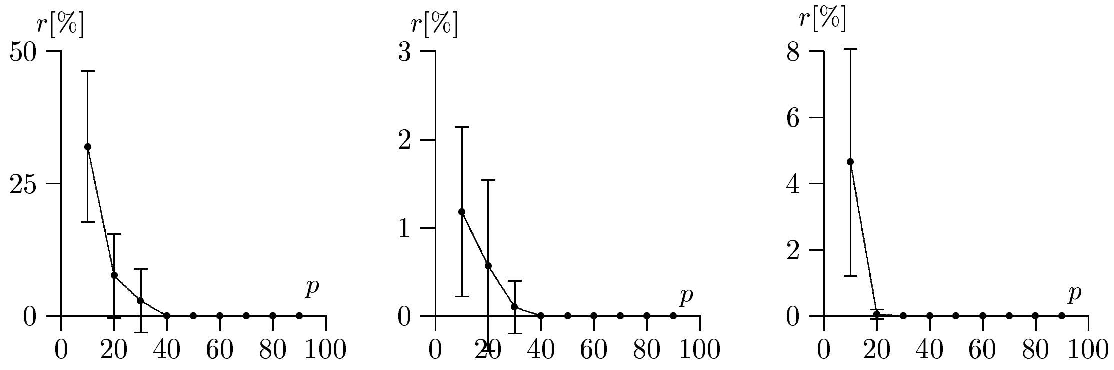
2.7.6 Generalization of decomposition

When the concept hierarchy is presented an attribute-value vector, it derives it corresponding class value by bottom-up derivation of intermediate concepts. For each intermediate concept, its example set may or may not include the appropriate example to be used for classification. In the later case, the default rule is used that assigns the value of most frequently used class (see Section 2.5) to the intermediate concept. The question is, how good is the generalization of the concept hierarchy without the default rule, i.e., given a test set, for how many of its examples the default rule is used.

To answer this question, we use three domains and show the percentage (r) of examples in the test set for which the default rule was used to derive the value of at least one concept in the hierarchy (Figure 2.20). Results show that this percentage decreases with the number of examples used in the training set and converges to 0. The results for other domains were similar. We conclude that the default rule does not contribute significantly to the generalization property of decomposition, especially when the training set covers the attribute space sufficiently well. The default rule may be beneficial though in the case of small training sets that relatively sparsely cover the attribute space.

2.7.7 Comparison of partition selection measures

The examples from previous sections use column multiplicity as the partition selection measure (Ψ_{CM}). We further tested HINT with other complexity-based measures. The experimental comparison of Ψ_{C} and Ψ_{SC} shows a large similarity of these two measures not just in terms of the ordering of partitions, but also in the similarity of their values. The examples in Appendix B clearly demonstrate this similarity. For this reason, it is sufficient to use Ψ_{SC}



(a) MONK2

(b) CAR

(c) ENTERPRISE

Figure 2.20: Percentage of examples from the test set r where a default rule was used to derive the class of at least one concept of the hierarchy.

alone to assess the effects of complexity based partition selection measures.

Concerning the classification accuracy and for all the domains used, the differences when the decomposition used either Ψ_{CM} or Ψ_{SC} where minor and most often insignificant. This was different for the discovered structures, which were (especially for DEX domains) to some extent more complex than those discovered with the use of Ψ_{CM}. However, when the maximum cardinality b of the bound set was raised from 3 to 4, these differences were less apparent, and both Ψ_{CM} and Ψ_{SC} guided decomposition discovered the same structures for CAR, EMPLOY1, and NURSERY, while the structures for ENTERPRISE were very similar. Ψ_{SC}-guided decomposition still discovered a less similar structure compared to the original DEX structure for EMPLOY2 and HOUSING, and this did not change when b was raised to 5 and higher. Interestingly, Ψ_{CM}-guided decomposition discovered the same structure for any value of b equal to or larger than 3. In part, the results of this experiments are shown in Table 2.11.

	b	CAR	EMPLOY1	EMPLOY2	NURSERY	HOUSING	ENTERPRISE
Ψ_{CM}	3	0.533	0.571	1.833	0.857	1.697	0.833
	4	0.533	0.571	1.833	0.857	1.697	0.833
Ψ_{SC}	3	0.533	0.857	1.694	0.929	1.758	1.636
	4	0.533	0.571	1.972	0.857	1.833	0.758

Table 2.11: Comparison of structures derived by Ψ_{CM} and Ψ_{SC}-guided decomposition with the original DEX models using the structure dissimilarity coefficient SDC.

The similarity of the results using Ψ_{CM} and Ψ_{SC} is rather surprising. The reason is that these two measures seem to prefer rather different partitions, where Ψ_{SC} is more biased towards the partitions with a larger number of attributes in the bound set. In such cases, the complexity of $\Theta_{\mathrm{C}}(G)$ tends to be smaller than in the cases with fewer bound attributes, and this decreases the sum $\Theta_{\mathrm{C}}(G)+\Theta_{\mathrm{C}}(H)$.

Let us illustrate this on an example. Figure 2.21 shows a structure for NURSERY discovered by Ψ_{SC}-guided decomposition with $b=4$. It is identical to that of Figure 2.13 discovered by Ψ_{CM}-guided decomposition, except that the intermediate concepts were discovered in a different order (note the indices). For example, the first partition used for decomposition by Ψ_{SC} had the bound set `<form,children,housing,finance>`, while that of Ψ_{CM} had `<cult_hist,advantage>`.

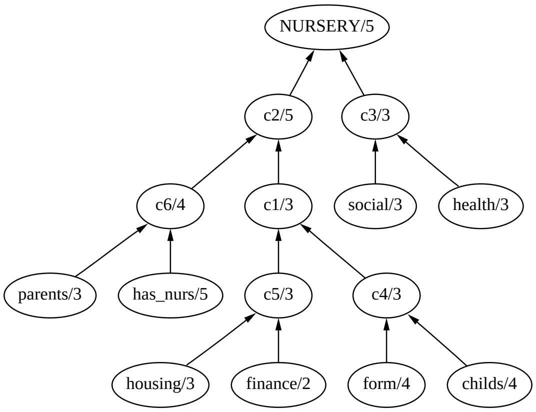


Figure 2.21: The structure discovered for NURSERY using Ψ_{SC}-guided decomposition with $b=4$.

The difference between partition selection measure can be further illustrated on the examples in Appendix B. There, the partition selection measures and the corresponding ranks are given for the initial example sets of LENSES, CAR, and NURSERY. Table 2.12 gives the correlation between these sets of ranks (the correlation was derived using the same formula as used in Appendix A). As expected, ranks given by Ψ_{C} and Ψ_{SC} are almost the same, while those of Ψ_{CM} very weakly correlate with the other two, especially for more complex domains CAR and NURSERY.

	Ψ_{C}	Ψ_{SC}
Ψ_{CM}	0.76	0.86
Ψ_{C}		0.87

(a) LENSES

	Ψ_{C}	Ψ_{SC}
Ψ_{CM}	0.23	0.23
Ψ_{C}		1.00

(b) CAR

	Ψ_{C}	Ψ_{SC}
Ψ_{CM}	0.17	0.17
Ψ_{C}		1.00

(c) NURSERY

Table 2.12: Correlation of partition ranks obtained by three different partition selection measures.

2.8 Summary and discussion

This chapter introduced a new machine learning approach based on function decomposition. A distinguishing feature of this approach is its capability to discover new, intermediate concepts, organize them into a hierarchical structure, and define the relationships between the attributes, newly discovered concepts, and target concept. In their basic form, these relationships are specified by newly constructed example sets. In a way, the learning process can thus be viewed as a process of generating new, equivalent example sets, which are consistent with the original example set. The new sets are smaller, have smaller numbers of attributes, and introduce intermediate concepts. Generalization also occurs in this process. The original undecomposed set of examples is consistent with the decomposed sets.

We have evaluated the decomposition-based learning method on several datasets. In terms of the classification accuracy, the decomposition significantly outperformed C4.5 in all but two domains (SHUTTLE and LENSES), where the differences were not significant. The examples also show that the decomposition is useful for the discovery of new intermediate concepts. For example, the decomposition was able to discover an appropriate concept hierarchy approved by domain experts for most of rather complex DEX real-world domains.

The classification accuracy results may be biased because we have mostly used the domains where we anticipated the hierarchies discoverable by decomposition. However, MONK2 is a counter example where the decomposition is not able to discover the original definition of the target concept, but rather unexpectedly its reformulation.

The decomposition approach as presented in this chapter is limited by that there is no special mechanism for handling noise and continuous attributes. The extensions that allow the decomposition to be used in such domains are presented in the following chapters.

Chapter 3

Decomposition-based attribute selection and discovery of redundancies

The attributes used in the example sets are usually of different relevance when used to determine the target concept. In this respect, some attribute may even be redundant. For example, to describe the concept "has-flu", a patient's temperature may provide substantially more information about the target concept than a patient's hight.

The attribute redundancy may be caused either because of its "true" irrelevancy to the target concept, or because the examples provided weakly cover the attribute space and do not provide evidence for such an attribute to be non-redundant. In both cases, it may be beneficial to preprocess such an example set and describe it only with a subset of most relevant attributes. Such preprocessing is called *attribute subset selection*, often referred to also as *feature subset selection.* The process aims at reducing the number of attributes and increasing the coverage of attribute space by examples, while constructing a new example set that adequately represents the concept as given by the original set of examples.

From the point of view of machine learning, the attribute subset selection attempts to find a subset of attributes under some objective function. Common objective functions are (John, Kohavi & Pfleger 1994): prediction accuracy, the complexity of the derived classifier, and minimal use of input attributes. The induced classifier may thus be more accurate at one hand, and less complex and therefore potentially more transparent on the other hand. Attribute subset selection may also be useful in the cases where the time complexity of machine learning tools strongly depends on the number of attributes. Function decomposition, for example, belongs to this category. Specifically for the decomposition, preprocessing by means of attribute selection may allow to consider also more complex intermediate concepts that would use higher number of attributes in the bound set.

This chapter describes a function decomposition-based attribute subset selection method that aims at minimizing the number of attributes and constructing a corresponding example set that is consistent to the original set of examples. First, section 3.1 describes the decomposition-specific approach to detect and remove redundant attributes. Decomposition may also handle cases when attribute itself is not redundant, but may be described with fewer values. The corresponding method is proposed in section 3.2. Both approaches are then incorporated in the attribute subset selection algorithm given in section 3.3. Section 3.4 experimentally evaluates the proposed method. The chapter concludes by a summary and discussion.

3.1 Redundancy of attributes

Let us first observe the redundancy of attributes with respect to the given example set:

Definition 3.1 Given an example set E_F that uses attributes $a_1, \dots, a_n$, let E'_F be a new example set that is derived from E_F by removing attribute a_j. a_j is *redundant* (or *irrelevant*) if E'_F is consistent with E_F.

In other words, if E_F partially specifies a function $F(a_1, \dots, a_n)$ and E'_F a function $F(a_1, \dots, a_{j-1}, a_{j+1} \dots, a_n)$, then a_j is redundant if $F'(e_i) = F(e_i)$ for every example $e_i \in E_F$.

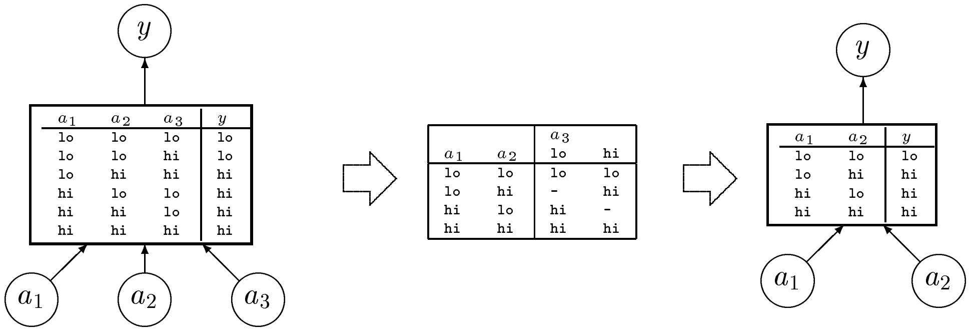
By above definition, attribute a_j is redundant if function F does not depend on it: with all other attributes constant, and varying the value of a_j, the value of function F does not change. In terms of decomposition, this inspires the following definition of redundancy:

Theorem 3.1 Let E_F be an example set that partially specifies $y = F(X) = F(a_1, \dots, a_n)$. Let $A|B$ be a disjoint partition of attributes in X such that $B = \langle a_j \rangle$ and $A = X \setminus \langle a_j \rangle$. Then a_j is redundant if $\nu(A|B) = 1$.

Proof: To prove this theorem, we refer to the single-step decomposition as defined in Chapter 2. There, $y = F(X)$ represented with a set of examples E_F was decomposed to $y = G(A, c)$ and $c = H(B)$. $\nu(A|B)$ denoted the number of values that are used by c in order for G and H to be consistent with F. When $\nu(A|B) = 1$, c uses a single value ($||c|| = 1$) and H is a constant. c can the be removed from examples E_G that partially defines G. E_G is then a new set of examples and can be used instead of E_F.

Example 3.1 Figure 3.1 shows an example set with redundant attribute a_3. The redundancy is discovered by evaluating the column multiplicity of the corresponding partition matrix. Since $\nu(\langle a_1, a_2 \rangle | \langle a_3 \rangle) = 1$, a_3 is redundant and a new example set is constructed that uses only the attributes a_1 and a_2.

Discovered redundancy can either indicate a true redundancy of attributes, or can simply mean that a training set is insufficient to pinpoint the need for this attribute. Moreover, removing one redundancy can make other redundant attributes non-redundant.

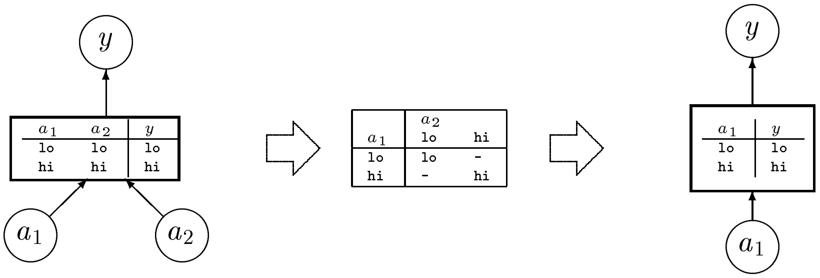


a_1	a_2	a_3	y
lo	lo	lo	lo
lo	lo	hi	lo
lo	hi	hi	hi
hi	lo	lo	hi
hi	hi	lo	hi
hi	hi	hi	hi

		a_3	
a_1	a_2	lo	hi
lo	lo	lo	lo
lo	hi	-	hi
hi	lo	hi	-
hi	hi	hi	hi

a_1	a_2	y
lo	lo	lo
lo	hi	hi
hi	lo	hi
hi	hi	hi

Figure 3.1: Discovery of redundant attribute a_3 and its removal from the training set.



a_1	a_2	y
lo	lo	lo
hi	hi	hi

	a_2	
a_1	lo	hi
lo	lo	-
hi	-	hi

a_1	y
lo	lo
hi	hi

Figure 3.2: a_1 and a_2 are both redundant in the original training set, but removal of a_2 makes a_1 non-redundant.

Example 3.2 An example in Figure 3.2 explains this case. There, x_1 and x_2 were both found redundant when considered separately. But once one of the attributes, say x_2, is removed, the other attribute (x_1) becomes non-redundant.

The decomposition-based redundancy definition can be simply modified so that to denote the redundancy of the set of attributes:

Definition 3.2 Let E_F be an example set that partially specifies $y = F(X) = F(a_1, \ldots, a_n)$. Let $A|B$ be some disjoint partition of attributes in X. The attributes in the set B are redundant if $\nu(A|B) = 1$.

Using the above definition, one may be tempted to use it in order to find the largest set of attributes B that are redundant and can be removed from the original example set. However, such an exhaustive search may be prohibitively complex even for small domains because of large number of partitions to consider. Such an approach would then be similar to the FOCUS algorithm of Almuallim & Dietterich (1991), which was originally defined for noise-free Boolean domains. As an alternative, section 3.3 presents a sub-optimal approach where only one attribute at a time is checked for redundancy.

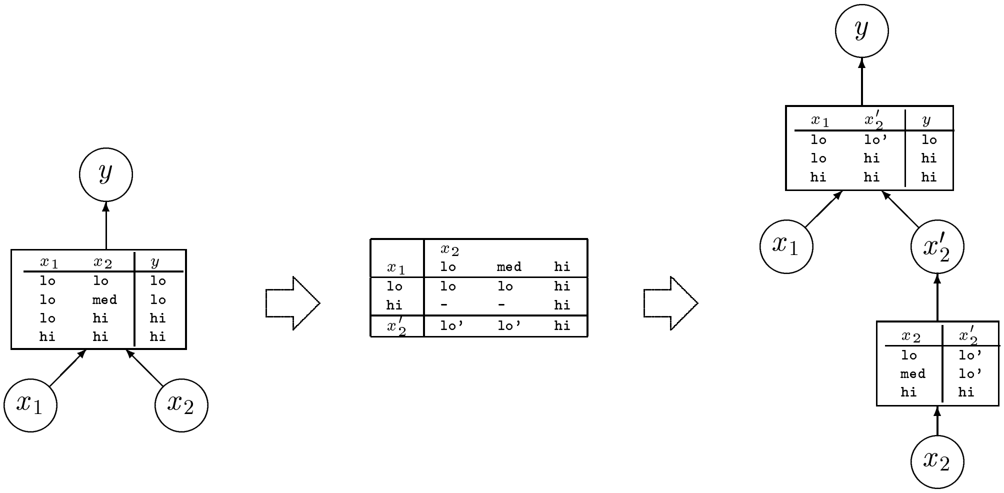


Figure 3.3: Discovery of redundant values of attribute x_2 (`lo` and `med` are replaced by `lo'`).

3.2 Redundancy of attribute values

Most often attribute subset selection methods, for example like the one proposed in (Kohavi & John 1997), focus only on the relevancy of the attributes, but do not specifically consider their value sets. An attribute may however be described by a set of values of which some are irrelevant. Furthermore, some attribute values may exhibit the same behavior with respect to the target concept. In both cases, it may be beneficial to derive the minimal description of such attribute and construct a corresponding set of examples. Again, we propose to handle such cases by a single-step decomposition:

Let E_F be an example set that partially specifies $y = F(X) = F(a_1, \ldots, a_n)$. Let $A|B$ be a disjoint partition of attributes in X such that $B = \langle a_j \rangle$ and $A = X \setminus \langle a_j \rangle$. If $\nu(A|B) < ||a_j||$, then a_j can be replaced with a new attribute a'_j, such that $y = G(A, a'_j)$ and $a'_j = H(a_j)$. G and H are functions represented with sets E_G and E_H and derived by single-step decomposition.

Note that E_G is a new set of examples that replaces E_F, while E_H is a mapping from the set of values of a_j to the new and reduced set of values of a'_j. In case an attribute would have a value defined but never used in the original example set, the corresponding column in the partition matrix would be empty and such value would not be considered by decomposition and consequently not represented in the example set E_H.

Example 3.3 Figure 3.3 shows an example where a_2 can be described with fewer values that are used in the original set. Namely, its values `lo` and `med` exhibit the same behavior in respect to the target concept, i.e., the two corresponding columns in the partition matrix are compatible. In other words, with all other attributes constant, changing a_2 from `lo` to `med` or vice versa never changes the value of y. Thus, `lo` and `med` can be replaced by a single value `lo'`.

3.3 Attribute and attribute values subset selection algorithm

The above two sections described the detection and removal of either redundant attribute or its values. Example 3.2 also demonstrated that removing one redundant may cause the other, previously redundant attribute, to become non-redundant. Therefore, the selection of attributes may depend on the order in which redundant attributes are removed.

Most often, the attribute subset selection algorithms remove the attributes in the order of their relevancy. The measure that estimate the relevance of the attributes has been extensively studied within statistics and machine learning. In our implementation incorporated within program HINT, the user can choose between any of the following: information measure (Quinlan 1979), GINI index of diversity (Breiman, Friedman, Olshen & Stone 1984), gain-ratio measure (Quinlan 1986*b*), and ReliefF and MDL-based measures (Kononenko 1994).

The attribute subset selection method is given as Algorithm 4.1. $\mathcal{R}(a_i)$ is the relevance estimated for attribute a_i, such that if $\mathcal{R}(a_i) > \mathcal{R}(a_j)$, the attribute a_i is more relevant than a_j. Note that the relevancy and redundancy of attributes are re-evaluated every time some redundancy is discovered and removed. The method falls under the category of filter algorithms (John et al. 1994) of which the aim is solely the reduction of number of attributes used. It is, however, different from other attribute subset selection algorithms in attempt not only to remove the redundant attributes but also to minimize the number of attribute values.

3.4 Experimental evaluation

3.4.1 DEX domains

DEX example sets introduced in section 2.7.2 were a subject to feature subset selection and redundancy removal. For neither of six datasets a completely redundant attribute was found. However, several redundancies in terms of the number of values used for specific attribute were detected. The proposed feature subset selection algorithm used information measure to estimate the relevancy of attributes.

Figure 3.1 gives the results. For example, for domain CAR, it was discovered that the concept does not distinguish between the cars with `4` or `more` doors, and that this two values can be merged into a single value of the new attribute `doors'`. More interesting is a redundancy discovery with EMPLOY1 and attribute `age`. There, the final concept did not distinguish between the youngest and the oldest applicants: attribute values `18-20` and `>55` can be represented with a single value instead.

The attribute value redundancies for DEX models are listed in Figure 3.1 and show that the attribute value minimization may not be only regarded as a data preprocessing, but may be a discovery by itself. When a data engineer (Marko Bohanec) was asked to comment on redundancies from Figure 3.1, he indeed recognized most of them as those that were intentionally used in DEX models with future extension and specialization of model functions

Input: Initial set of examples E_F describing a single output concept y using attribute set X
Output: A set consistent to the initial set of examples with
potentially fewer attributes and/or reduced attribute value sets

```
mark($a_i$) $\leftarrow$ TRUE for every $a_i \in X$
while $\exists a_i$ such that mark($a_i$) do
    for every $a_i \in X$ with mark($a_i$) do
        derive $\mathcal{R}(a_i)$
        $B \leftarrow \langle a_i \rangle$, $A \leftarrow X \setminus \langle a_i \rangle$
        redundant($a_i$) $\leftarrow \nu(A|B) < ||a_i||$
    end for
    if $\nexists$ redundant($a_i$) then stop
    select $a_i$ with lowest $\mathcal{R}(a_i)$, such that redundant($a_i$)
    $B \leftarrow \langle a_i \rangle$, $A \leftarrow X \setminus \langle a_i \rangle$
    decompose $E_F$ to $E_G$ and $E_H$, such that $y = G(A, a_i')$ and $a_i' = H(B)$
    if $||a_i'|| = 1$ then
        remove attribute $a_i'$ from $G$
        $X \leftarrow A$
    else
        $X \leftarrow A \cup \langle a_i' \rangle$
        mark($a_i$) $\leftarrow$ FALSE
    end if
    $E_F \leftarrow E_G$
end while
```

Algorithm 3.2 Attribute and attribute's values subset selection algorithm

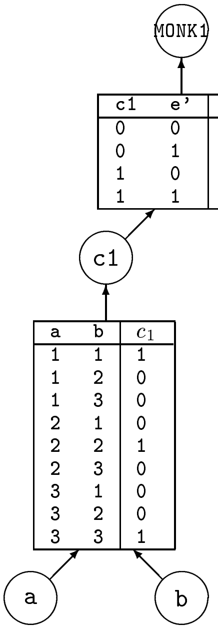
in mind.

3.4.2 MONK1

MONK1 training set from section 2.7.4 was used to investigate the utility of decomposition-based feature subset selection. Three attributes were discovered to be redundant: `f`, `c`, and `d` (in this order) and for attribute `e` an attribute `e'` with only two values instead of four was derived. The set obtained by attribute subset selection was then given to decomposition, and the resulting concept structure is presented in Figure 3.4. This structure is relatively easy to interpret: the target concept can be represented as `a=b OR e=1`, which is exactly the function originally used to generate the examples for MONK1. Preprocessing by means of attribute and values selection therefore removed the “truly” redundant attributes and found the appropriate transformation of values of `e`.

domain	attribute	#values	values merged
CAR	doors	$4 \rightarrow 3$	2, 3, (4, 5-more)
EMPLOY1	exper	$5 \rightarrow 4$	none, 0-1, 1-5, (6-10, more)
	degree	$4 \rightarrow 3$	prim.school, high sch., BSc, (MSc, PhD)
	age	$5 \rightarrow 4$	(18-20,>55) 21-25, 26-40, 41-55
EMPLOY2	exper	$6 \rightarrow 4$	none, 0-1, (1-5, 6-10), more
	degree	$4 \rightarrow 3$	prim.school, high sch., BSc, (MSc, PhD)
ENT	prd	$5 \rightarrow 4$	(e,d),c,b,a
	prf	$3 \rightarrow 2$	(neg, zero, pos)
	liq	$3 \rightarrow 2$	lt 1.0, 1-1.24, (1.25-1.5, ge 1.5)
	p-a	$5 \rightarrow 4$	(e, d), c, b, a
NURSERY	childreen	$4 \rightarrow 3$	1,2,(3,more)
	social	$3 \rightarrow 2$	(none,med),hi
HOUSING	earnings	$3 \rightarrow 2$	(below-ave,ave),high
	family	$5 \rightarrow 3$	single,married,(family,extend.family,single-wage)
	suitab	$3 \rightarrow 2$	(high,normal),low
	employed	$3 \rightarrow 2$	one,(two,more)

Table 3.1: DEX model attributes whose descriptions were found redundant in the number of values. 3rd column indicates the original and reduced number of values for specific attribute. The values of original attributes that are represented with single attribute value in the reduced set are given in brackets.



c1	e'	MONK1
0	0	0
0	1	1
1	0	1
1	1	1

a	b	c_1
1	1	1
1	2	0
1	3	0
2	1	0
2	2	1
2	3	0
3	1	0
3	2	0
3	3	1

e	e'
1	1
2	0
3	0
4	0

Figure 3.4: MONK1 concept structure as discovered by HINT. Original example set was preprocessed by attribute subset selection.

3.4.3 Protein secondary structure prediction

Determination of 3D protein structure is of great importance for the prediction of their function in pharmacology and medicine. So far, there are over 50.000 known proteins, but the structure is known for around 500 only. Experimental methods for protein structure determination (X-ray diffraction, NMR) are expensive and time consuming.

An ongoing project at Center for Applied Mathematics and Theoretical Physics, University of Maribor, Jožef Stefan Institute, and National Institute of Chemistry, Ljubljana, has a long-term goal of the prediction of 3D protein structure for a given amino-acid sequence, on the basis of a database of known protein structures. The project started with a design of Symbolic Protein Data Base, that now includes selected proteins with known structure and their symbolic descriptions in terms of their relevant properties (Mozetič & Hodoščček 1997).

A short-term goal of this project is the prediction of a secondary structure for a given primary structure (sequence of amino-acids). One of the approaches is to use learning from examples. There, the problem to start with is to appropriately formulate the learning task: which attributes to use and what concepts to learn. The first attempt focused on selection of attributes to determine the secondary structure. This section describes the preliminary results with the decomposition-based attribute subset selection.

Three different example sets were investigated. For all three, the goal was to determine the structure at specific amino-acid in the protein chain. Each example used a set of attributes that describe the k surrounding amino-acids. k is referred to as a window size: for the window of k, the attributes are given for the central amino-acid of the window plus k previous and k

next amino-acids in the chain. In all three case, the example set included over 51000 examples that specify the corresponding structure of amino-acid at the center of a window, which can be either helix, sheet, or random.

The first example set used a widow of 9. The set used 19 (9+1+9=19) attributes giving only the names of amino-acids. As there are 20 different amino-acids, the attributes were 21-valued (one additional value for "unknown"). Attribute subset selection by HINT excluded all the attributes but those from window of 4, thus constructing an example set consistent to the original set but containing only 9 attributes.

The second example set used a window of 5, but besides the amino-acid name used additional 9 different attributes to describe each amino-acid. The total number of attributes used was thus $(5+1+5)*(9+1) = 110$. After feature subset selection, only 9 attributes remain. These were all the attributes giving the names of the amino-acids in the window of 4.

The last experiment used an example set with the window of 4, where each amino-acid was described with 9 attributes as in the previous data set. The names of amino-acids were not given. The original set thus used $(4+1+4)*9 = 81$ attributes. Attribute subset selection reduced this set to 21 attributes, most of them describing the amino-acids closer to the center of the window (see Figure 3.2). Again, only the attributes of amino-acids within the window of 4 remained.

Furthermore, it was noticed that the attributes that used higher number of values remained. That could be accounted to the fact that HINT used information measure to estimate the relevancy of attributes, which is biased toward such attributes (Kononenko 1994). However, when an alternative and differently biased MDL-based measure was used instead, the results of attribute subset selection were not significantly different. A possible explanation that is also consistent with the results of the second experiment is that attributes with higher number of values remain because they better distinguish between different amino-acids, and therefore better approximate the attribute containing its name.

Overall, the conclusion using the above attribute subset selection analysis are:

- the attributes within the window of 4 seem sufficient,
- among all the attributes studied the amino-acid names seem most relevant.

The results and conclusions above are preliminary. However, the result regarding the windows size was expected and can be explained by physical properties of proteins. Further experiments will address the classification performance of different classifiers, and a performance-based attribute selection. The first experiments of this type confirmed the above conclusions.

relative position	-5	-4	-3	-2	-1	0	1	2	3	4	5
# attributes	0	1	1	3	4	3	5	2	1	1	0

Table 3.2: The number of the remaining attributes in the third experiment that describe an amino-acid at a given position within a window.

3.5 Summary and discussion

In machine learning, the use of attribute subset selection aims at constructing example sets on which inductive learners perform better (Kohavi & John 1997). For decomposition, the data preprocessing by means of subset selection may be necessary for the following reason. When considering different partitions, several bound sets as a whole may be found redundant. If not equipped with with subset selection techniques, the decomposition would select one such set arbitrarily, and subsequently remove it because of its corresponding column multiplicity of 1. Instead, attribute subset selection removes the redundant attributes based on their relevancy. Furthermore, the process may result in the example sets with fewer attributes, which can reduce the running time of decomposition or allow it to consider bigger bound sets and consequently more complex intermediate concepts.

The proposed method falls into category of *filter* approaches to attribute subset selection which minimize the number of attributes independently of inducer (Kohavi & John 1997). A different, *wrapper* approach instead removes one candidate attribute at a time and tests the performance (classification accuracy, complexity) of the inducer on the resulting example set. If this has degraded, a different attribute is considered for removal. We can use both induction by decomposition and attribute redundancy estimation by decomposition for the wrapper approach as well. The concern is however the complexity, which in wrapper approach increases by a factor of $O(n^2)$. For this reason, we have, as of now, implemented only the filter approach.

The attribute subset selection as introduced in this chapter was based on decomposition-based definition of redundancy of either attribute or their values. We consider these as the main contributions of the proposed approach. The techniques developed do not include any particular mechanisms for noise handling. However, in the next chapter a decomposition that can deal with noise is introduced, and section 4.2.7 shows how to straightforwardly extend the method proposed here to (1) handle noise and (2) remove the attribute so as to minimize the expected classification error.

The results of experimental evaluation of the proposed method are encouraging. In all cases considered, the algorithm's behavior in terms of which attributes were removed and which value sets minimized were explainable. The decomposition-based attribute subset selection was found not only to be useful for attribute reduction, but was also as a tool for preliminary data analysis.

Chapter 4

Minimal-error function decomposition

The minimal-complexity function decomposition described in Chapter 2 imposes the constraint of consistency over the example sets it can handle: there can not exist a pair of examples with the same attribute values but of different class. The real-world data, however, most often violates this constraint. Its inconsistency may be contributed to either noise or the incompleteness of description language (Clark & Niblett 1987). Other properties of data that the minimal-complexity decomposition could not deal with further include missing attribute values and uncertainty.

This Chapter proposes a different decomposition method that is based on the minimization of classification error. It was designed with a purpose to appropriately handle noise, but the example set representation schema the algorithm uses allows to handle other real-world data properties mentioned above.

The new decomposition algorithm requires a new representation schema for the set of examples, where each example defines a distribution of classes rather than a single class. This representation is introduced in section 4.1. Section 4.2, which is the core of this Chapter, describes minimal-error decomposition algorithm. Following are two sections that deal with various data properties and show (1) how to transform such data to the examples of the new representation schema in order to be used for for the decomposition (section 4.3), and (2) once a concept hierarchy has been derived, classify such data (section 4.4). Notes on implementation are given in section 4.5. The new decomposition algorithm is experimentally evaluated in section 4.5. Section 4.7 summarizes the method and experimental results.

4.1 Representing examples with class distributions

In Chapter 2 we have proposed a decomposition that requires that no two examples have the same values of attributes. This constraint also imposed the consistency of such dataset, so that no two examples with same attribute values can describe different class. In the beginning of this chapter we argued that data consistency is seldom present in the real-world, and to handle inconsistent data we have to extend the data representation and decomposition algorithm accordingly.

The requirement of example set consistency by minimal-complexity decomposition can be contributed to the specific use of partition matrix: as defined in Chapter 2, each entry of partition matrix can hold just a single class value. Consequently, each combination of attribute values in the example set can have a single associated class value.

To alleviate such constraint, we redefine the entries of partition matrix to instead represent the *class distributions*. Class distribution is a vector. Each i-th element gives the number of examples in the corresponding example set that define the i-th class, and use the same attribute values as the corresponding partition matrix entry.

Example 4.1 Let us illustrate this new representation on a simple example. Consider the example set from Table 4.1.a where each example defines a single class. The set is inconsistent because of the examples with attributes ⟨med,lo,lo⟩, and also includes several other examples that use same attribute values. Table 4.1.b gives an equivalent example set that uses class distributions. The distributions are given in brackets where, for example, a vector (0,2,1) denotes that a corresponding example represents two examples from the original set with class med and one example with class hi. Note that in the set that are no examples with same attribute values. Such set can then straightforwardly be represented with a partition matrix. This is for our example given in Table 4.4.a.

The example sets that use class distributions can then be regarded as a transformation of some possibly inconsistent example set: for each combination of attribute values used in the original set, the corresponding example with class distribution is created, such that the i-th element of class distribution vector is equal to the number of examples in the original set that describe the i-th value of class.

From the point-of-view of decomposition, the most important property of set of examples with class distributions is the uniqueness of the examples in regard to their attribute values. As introduced in the next section, not only will the new decomposition be able to consider such sets, but will also use the same representation for the discovered example sets.

4.2 Minimal-error decomposition algorithm

This section introduces a decomposition algorithm that is driven by error-minimization: given the initial set of examples, it tries to decompose it to a system of example sets with minimized classification error of obtained classifier. The method was inspired by a noise-handling approach to decision tree pruning (section 4.2.1). Section 4.2.2 describes minimal-error single-step decomposition. This method is essentially different from the corresponding minimal-

x_1	x_2	x_3	y
lo	lo	hi	lo
lo	hi	lo	lo
lo	hi	hi	hi
med	lo	lo	med
med	lo	lo	hi
med	lo	lo	med
med	hi	lo	med
med	hi	lo	med
med	hi	hi	hi
hi	lo	lo	hi
hi	lo	hi	med
hi	hi	lo	hi
hi	hi	hi	hi
hi	hi	hi	hi

(a)

x_1	x_2	x_3	y
lo	lo	hi	(1,0,0)
lo	hi	lo	(1,0,0)
lo	hi	hi	(0,0,1)
med	lo	lo	(0,2,1)
med	hi	lo	(0,2,0)
med	hi	hi	(0,0,1)
hi	lo	lo	(0,0,1)
hi	lo	hi	(0,1,0)
hi	hi	lo	(0,0,1)
hi	hi	hi	(0,0,2)

(b)

Table 4.1: An inconsistent set of examples where several examples have the same value of attributes (left) and its representation with examples that use class distributions (right).

complexity algorithm (section 2.2), as it does not construct an incompatibility graph but rather uses a reduction of partition matrix by column merging. On the other hand, overall minimal-complexity and minimal-error decomposition algorithms are rather similar, with the exception of partition selection measures and decomposability condition (section 4.2.4). The new decomposition algorithm estimates the error using m-error estimate, where the parameter m has to be appropriately assessed from the data. The corresponding method is given in section 4.2.6. The section concludes with a note on complexity of the new algorithm.

4.2.1 Related work on noise handling

The real-world data most often contains errors that are due to the nature of data collection or measurement. For example, errors occur because of imperfect measuring equipment or because of mistakes at data entry. Early machine learning systems often made a “noiseless domain assumption” and introduced the classifiers that were consistent with the learning set of examples. However, when the domains contained noise, the induced classifiers were unnecessary complex and did not perform well. An alternative to such induction system was proposed by Clark & Niblett (1987), who suggested to instead induce less complex classifiers that possibly missclassify some examples but potentially perform better on the unseen examples. Their method, specific to the rule induction system CN2, used a significance test to judge which rules to retain in the classifier.

For another, nowadays probably most popular machine learning paradigm of top-down induction of decision trees, the early noise handling mechanisms were described by Quinlan (1986*a*) and Kononenko, Bratko & Roškar (1984) (pruning of decision tree during the construction), and by Niblett & Bratko (1986) and Cestnik & Bratko (1991) (pruning the decision tree after it has been constructed).

The pruning of Cestnik & Bratko (1991) is based on the Bayesian approach to estimate probabilities. Consider a set of N examples, of which n_c represent a class c of apriori probability p_c. The probability p that the new example will be of class c is:

$$p = \frac{n_c + p_c m}{N + m} \tag{4.1}$$

This formula is referred to as *m-probability estimate* and was proposed by Cestnik (1990). He demonstrated that several machine learning classifiers (naive Bayesian classifier, decision trees) perform significantly better when the induction uses m-probability estimate instead of other known estimates like frequency or Laplace law of succession (Niblett & Bratko 1986).

m-probability estimate can also be used to estimate the error when classifying to the class c. Given an example set as above, the m-error estimate is:

$$\varepsilon = 1 - \frac{n_c + p_c m}{N + m} \tag{4.2}$$

The value of parameter m is domain dependent. Cestnik & Bratko (1991) propose that m should be adjusted to some essential properties of the learning domain, e.g., to noise level. The value of m can be either defined by a domain expert or derived directly from the set of examples (see section 4.2.6).

Cestnik & Bratko (1991) used m-error estimate for decision tree pruning. Their approach was based on the pruning method of Niblett & Bratko (1986). They all used the following idea. Given a set of examples E_0, the condition at the node of decision tree splits it to sets E_1 and E_2. Let the error estimates when classifying to the majority class of E_0, E_1, and E_2 be ε_0, ε_1, and ε_2, respectively. Let p_1 and p_2 be the probabilities of sets E_1 and E_2, respectively. The Niblett-Bratko pruning rule states that if $\varepsilon_0 < p_1\varepsilon_1 + p_2\varepsilon_2$, the tree has to be pruned at node n. Originally, the method used Laplace error estimate, which was by Cestnik & Bratko (1991) replaced with m-error estimate. This pruning method aims to minimize the expected error of decision tree, and is thus referred to as *minimal-error pruning*. In the following sections, we present the decomposition that is driven by the same goal: given a set of examples, the minimal-error decomposition tries to decompose it in order to minimize the expected classification error of the obtained classifier.

4.2.2 Single-step decomposition

Like for the single-step decomposition algorithm from Chapter 2, this single-step decomposition decomposes a set of examples E_F to sets E_G and E_H, where these sets partially describe the functions $c_i = F(X)$, $c_i = G(A, c_j)$, and $c_j = H(B)$, respectively. $A|B$ is a non-trivial partition of attributes X. The example sets E_F, E_G, and E_H use class distributions.

Definition 4.1 Let E_F be a set of *examples with class distributions* that represents the function $y = F(x_1, \ldots, x_n)$. Each example $e_i \in E_F$ is given with the value of attributes $x_1, \ldots, x_n$ and has associated a *class distribution vector* $\mathbf{d_i} = (d_i^1, \ldots, d_i^{||y||})$.

Let us start by defining the estimated error of example set E_F that represents the function $c = F(x_1, \ldots, x_m)$. Consider first a single example $e_i \in E_F$. When this is used for classification, it classifies to class c_k such that of all the elements in the distribution vector of e_i the k-th element is the highest ($d_i^k = \max(\mathbf{d_i})$). Using m-error estimate as introduced above, the error estimate for example e_i is:

$$\varepsilon(e_i) = 1 - \frac{\max(\mathbf{d_i}) + p(c_k)m}{\sum_j d_i^j + m} \tag{4.3}$$

$p(c_k)$ is the apriori probability of class c_k as estimated from the class distributions of example set E_F:

$$p(c_k) = \frac{\sum\limits_{i=1}^{||E_F||} d_i^k}{\sum\limits_{i=1}^{||E_F||} \sum\limits_{j=1}^{||c||} d_i^j} \tag{4.4}$$

We further define the probability of example e_i in the set E_F:

$$p(e_i) = \frac{\sum\limits_{k=1}^{||y||} d_i^k}{\sum\limits_{l=1}^{||E_F||} \sum\limits_{j=1}^{||y||} d_l^j} \tag{4.5}$$

The overall classification error estimated for the example set E_F is then the sum of products of errors estimates and probabilities of examples in E_F:

$$\varepsilon(E_F) = \sum_{i=1}^{||E_F||} p(e_i)\varepsilon(e_i) \tag{4.6}$$

Example 4.2 Table 4.2 shows the probabilities and m-error estimates for the examples from the set given in Table 4.2. Value 2 for the parameter m was used. Also given are apriori class probability as estimated from this set. For instance, to compute $p(\mathtt{y} = \mathtt{lo})$ one must first observe that the sum of all elements of distribution vectors is 14, and that the sum of all elements of vectors for class lo is 2. Therefore, $p(\mathtt{y} = \mathtt{lo}) = \frac{2}{12} = 0.143$. Let us now, for instance, compute the m-error estimate for an example 4. This example classifies to class med, and has $\max(\mathbf{d}_4) = 2$, and $\sum_j d_4^j = 3$. The error estimate is $\varepsilon(e_4) = 1 - \frac{2+0.357\times 2}{3+2} = 0.457$.

	x_1	x_2	x_3	y	$p(e_i)$	$\varepsilon(e_i)$	$p(e_i)\varepsilon(e_i)$
1	lo	lo	hi	(1,0,0)	0.071	0.429	0.041
2	lo	hi	lo	(1,0,0)	0.071	0.429	0.041
3	lo	hi	hi	(0,0,1)	0.071	0.333	0.024
4	med	lo	lo	(0,2,1)	0.214	0.457	0.098
5	med	hi	lo	(0,2,0)	0.143	0.250	0.046
6	med	hi	hi	(0,0,1)	0.071	0.333	0.024
7	hi	lo	lo	(0,0,1)	0.071	0.333	0.024
8	hi	lo	hi	(0,1,0)	0.071	0.762	0.031
9	hi	hi	lo	(0,0,1)	0.071	0.333	0.024
10	hi	hi	hi	(0,0,2)	0.143	0.250	0.036
							$\sum = 0.393$

(a)

$p(\texttt{y} = \texttt{lo})$	$p(\texttt{y} = \texttt{med})$	$p(\texttt{y} = \texttt{hi})$
0.143	0.357	0.5

(b)

Table 4.2: m-error estimates and class probabilities for the example set from Table 4.1.

To derive the error that this example contributes to the set E_F, we have to multiply $\varepsilon(e_4)$ with the probability of this example. Since the sum of values of distribution vector for e_4 is 3, it follows that $p(e_4) = \frac{3}{14} = 0.214$ and consecutively $p(e_4)\varepsilon(e_4) = 0.214 \times 0.457 = 0.098$. Note that the overall m-error estimate for this set of examples is 0.393.

For the purpose of single-step decomposition, we again represent the set E_F using the partition matrix. For our example set in Table 4.1.b and for the partition $A|B = \langle x_1\rangle|\langle x_2, x_3\rangle$, the partition matrix $\mathcal{P}_{A|B}$ is given Table 4.4.a. This time, we label each column of $\mathcal{P}_{A|B}$ with its unique column index l. Again, the column labels represent the value of the new intermediate concept c_j. We can then represent the examples from Table 4.1.b with an equivalent set of examples in Table 4.3.a, where each combination of values of attributes in B was replaced by a corresponding partition matrix column label. We will refer to such set of examples as the *initial set of examples* E_0. Note that because of equivalence of the sets E_0 and E_F, its estimated errors are the same: $\varepsilon(E_0) = \varepsilon(E_F)$.

The idea for the single-step decomposition is based on *partition matrix column merging.* Suppose that for $\mathcal{P}_{A|B}$ in Table 4.4.a we merge the columns with indices 1 and 3. The columns 1 and 3 are then replaced with a single column, where, row by row, the class distributions of columns 1 and 3 have been summed up. This derives a new partition matrix (Table 4.4.c) with one column less, which in turn represents a new example set E_1 (Table 4.3.b). E_1 can be regarded as E_0 with the examples that belong to either columns 1 or 3 and have equal values of the attributes in A merged.

Let us now compute the estimated error for the set E_1. This is $\varepsilon(E_1) = 0.3629$, which is less than the error of the initial example set: $\varepsilon(E_0) = 0.3933$. In other words, partition matrix column merging may result in a new example set with decreased estimated error.

x_1	c_j	y
lo	2	(1,0,0)
lo	3	(1,0,0)
lo	4	(0,0,1)
med	1	(0,2,1)
med	3	(0,2,0)
med	4	(0,0,1)
hi	1	(0,0,1)
hi	2	(0,1,0)
hi	3	(0,0,1)
hi	4	(0,0,2)

(a)

x_1	c_j	y
lo	1	(1,0,0)
lo	2	(1,0,0)
lo	3	(0,0,1)
med	1	(0,4,1)
med	3	(0,0,1)
hi	1	(0,0,2)
hi	2	(0,1,0)
hi	3	(0,0,2)

(b)

Table 4.3: An initial set of examples E_0 and a set E_1 after columns 1 and 3 of $\mathcal{P}_{A|B}$ in Table 4.4.a have been merged. Furthermore, E_0 and E_1 correspond to partition matrices in Tables 4.4.a and 4.4.c, respectively.

The potential decrease of estimated error of the example set after merging of two columns provides the motivation for the core of the new single-step decomposition algorithm. Using the successive steps of $\mathcal{P}_{A|B}$ column merging, the algorithm aims to derive the partition matrix with a corresponding example set of the lowest estimated error.

The minimal-complexity single-step decomposition is described as follows. At each step, all pairs of columns (l_i, l_j) are considered for merging, and the error $\varepsilon(l_i, l_j)$ of each new corresponding example set that would be derived by merging the columns l_i and l_j is computed. The pair of columns is then selected which yields the new example set E_{j+1} with the lowest estimated error. If this error is lower than or equal to the estimated error of the example set before merging, i.e., if $\varepsilon(E_{j+1}) \leq \varepsilon(E_j)$, then the merge is performed. Else, the merging stops and we will refer to the partition matrix obtained as a *reduced partition matrix.*

Note that to estimate the error for every new set of examples obtained by column merging, we can instead assign the error to each of the columns of $\mathcal{P}_{A|B}$. Let $B(e_i)$ be a column index for the example e_i. Then, the estimated error of column l of $\mathcal{P}_{A|B}$ is:

$$\varepsilon(\mathcal{P}_{A|B,l}) = \sum_{B(e_i)=l} p(e_i)\varepsilon(e_i) \tag{4.7}$$

Let us denote the partition matrix after k successive column merging as $\mathcal{P}_{A|B}(k)$. The error estimate for its corresponding example set E_k is then

$$\varepsilon(E_k) = \sum_l \varepsilon(\mathcal{P}_{A|B,l}(k)) \tag{4.8}$$

When merging two columns l_i and l_j of $\mathcal{P}_{A|B}(k)$ resulting in a new column l_{ij} of $\mathcal{P}_{A|B}(k+1)$ with the error estimate $\varepsilon\mathcal{P}_{A|B,l_{ij}}(k)$, the error of new example set E_{k+1} is computed as

$$\varepsilon(E_{k+1}) = \sum_{l\neq l_i \& l\neq l_j} \varepsilon(\mathcal{P}_{A|B,l}(k)) + \varepsilon(\mathcal{P}_{A|B,l_{ij}}(k)) \tag{4.9}$$

To compute the error of the new set E_k, it is therefore sufficient to compute the error of the new merged column l_{ij}, since the error estimates of other columns in the above equation were already computed in the previous merging steps.

The method for partition matrix column merging is presented as Algorithm 4.1. The algorithm is sub-optimal and greedy, as at each step considers the merging of only a pair of columns.

Input: Initial partition matrix $\mathcal{P}_{A|B}(0)$
Output: Reduced partition matrix $\mathcal{P}_{A|B}(k)$ obtained by k steps of column merging

```
compute $\varepsilon(\mathcal{P}_{A|B,l_i}(0))$ for every column $l_i$ of $\mathcal{P}_{A|B}(0)$
$k \leftarrow 0$
do
    for every pair of columns $(l_i, l_j)$ in $\mathcal{P}_{A|B}(k)$ do
        compute $\varepsilon(\mathcal{P}_{A|B,l_{ij}}(k))$
        $\varepsilon(l_i, l_j) = \sum_{l\neq l_i \& l\neq l_j} \varepsilon(\mathcal{P}_{A|B,l}(k)) + \varepsilon\mathcal{P}_{A|B,l_{ij}}(k)$
    end for
    select $(l_i, l_j)$ with minimal $\varepsilon(l_i, l_j)$
    if $\varepsilon(l_i, l_j) \leq \varepsilon(\mathcal{P}_{A|B,l}(k))$ then
        derive $\mathcal{P}_{A|B,l}(k+1)$ by merging columns $l_i$ and $l_j$ of $\mathcal{P}_{A|B,l}(k)$
        $\varepsilon(\mathcal{P}_{A|B,l}(k+1)) \leftarrow \varepsilon(l_i, l_j)$
        $k \leftarrow k+1$
    end if
while $\varepsilon(l_i, l_j) \leq \varepsilon(\mathcal{P}_{A|B,l}(k))$
```

Algorithm 4.1 Partition matrix column merging for single-step decomposition

Example 4.3 Table 4.4 shows a sequence of column merging for our example partition matrix. The value 2 of parameter m was used to compute the error estimates. Each partition matrix has at its right side a corresponding error estimation for the new matrix when two of its columns are merged. The error of the best candidate for merging is marked in bold. Note that the merging stops after 2 steps: merging the columns 1 and 2 of the partition matrix in Table 4.4.e would result in an increase of estimated error $(0.3959 > 0.3486)$.

The reduced partition matrix $\mathcal{P}_{A|B}(k)$ is therefore the one that, by merging any pair of its columns, the estimate error of the corresponding example set does not decrease. It is now straightforward to use $\mathcal{P}_{A|B}(k)$ to construct the new example sets E_G and E_H. In fact, E_G

x_2	lo	lo	hi	hi
x_1 x_3	lo	hi	lo	hi
lo	-	(1,0,0)	(1,0,0)	(0,0,1)
med	(0,2,1)	-	(0,2,0)	(0,0,1)
hi	(0,0,1)	(0,1,0)	(0,0,1)	(0,0,2)
l	1	2	3	4
$\varepsilon(\mathcal{P}_{A\|B,l})$	0.1267	0.0714	0.1119	0.0833

(a) $\mathcal{P}_{A|B}(0)$ with $\varepsilon = 0.3933$

	2	3	4
1	0.4076	**0.3510**	0.3929
2		0.3886	0.4195
3			0.4152

(b) $\varepsilon(l_i, l_j)$

x_2	lo hi	lo	hi
x_1 x_3	lo lo	hi	hi
lo	(1,0,0)	(1,0,0)	(0,0,1)
med	(0,4,1)	-	(0,0,1)
hi	(0,0,2)	(0,1,0)	(0,0,2)
l	1	2	3
$\varepsilon(\mathcal{P}_{A\|B,l})$	0.1963	0.0714	0.0833

(c) $\mathcal{P}_{A|B}(1)$ with $\varepsilon = 0.3510$

	2	3
1	**0.3486**	0.3726
2		0.3772

(d) $\varepsilon(l_i, l_j)$

x_2	lo hi lo	hi
x_1 x_3	lo lo hi	hi
lo	(2,0,0)	(0,0,1)
med	(0,4,1)	(0,0,1)
hi	(0,1,2)	(0,0,2)
l	1	2
$\varepsilon(\mathcal{P}_{A\|B,l})$	0.2563	0.0833

(e) $\mathcal{P}_{A|B}(2)$ with $\varepsilon = 0.3486$

	2
1	**0.3956**

(f) $\varepsilon(l_i, l_j)$

Table 4.4: The sequence of column merging of partition matrix. The reduced partition matrix (e) is obtained by merging the columns of the initial partition matrix (a). Estimated error for partition matrix that would be derived by merging the columns l_i and l_j of corresponding matrix is given on the right-hand side.

is exactly the example set that corresponds to the reduced partition matrix and is therefore equal to E_k. The example set $e_l \in E_H$ is derived as a mapping of values of attributes in B to the corresponding labels of columns. Each example in E_H therefore represents a single class that is equal to column label. The the class distribution vector for example $e_l \in E_H$ derived from column l has a single non-zero element d_l^l equal to

$$d_l^l = \sum_{B(e_i)=l}^{||E_F||} \sum_{j=1}^{||c||} d_i^j \tag{4.10}$$

Example 4.4 For our example set E_F from Table 4.1.b and its final partition matrix in Table 4.4.e, the example sets E_G and E_H are given in Table 4.5.a and 4.5.b, respectively. For each example, these Tables give a class distribution and a class each example classifies to. This is equal to the class with the highest corresponding value in example's distribution vector. E_H actually represents a function $c = \mathrm{MIN}(x_2, x_3)$ and E_G a function $y = \mathrm{MAX}(x_1, c)$. c is a new intermediate concept. Note that the derived examples are not consistent with the original examples from Table 4.1.a. Namely, the derived concept hierarchy missclassifies the examples hi $= F($med, lo, lo$)$ and med $= F($hi, lo, hi$)$. These examples were intentionally included in the original example set to demonstrate the ability of the new single-step decomposition algorithm to handle noise.

x_2	x_3	distribution	c
lo	lo	(2 0)	0
lo	hi	(2 0)	0
hi	lo	(3 0)	0
hi	hi	(0 3)	1

(a) E_H with interpretation $c = \mathrm{MIN}(x_2, x_3)$

x_1	c	distribution	y
lo	0	(2 0 0)	lo
lo	1	(0 0 1)	hi
med	0	(0 4 1)	med
med	1	(0 0 1)	hi
hi	0	(0 1 2)	hi
hi	1	(0 0 2)	hi

(b) E_G with interpretation $y = \mathrm{MAX}(x_1, c)$

Table 4.5: Example sets E_G (left) and E_H (right) that are decompositions of the set E_F from Table 4.1.b.

As was the case for the single step decomposition algorithm of Chapter 2, the partition matrix for the implementation of the minimal-error single-step decomposition is used only implicitly. Again, the reason for this is the usual sparseness of the machine learning datasets and more memory efficient implementation when merging is done directly on the set of examples. That is, the single step decomposition starts with the example set $E_0 = E_F$. Next, it replaces the attributes in B with the new attribute c and its initial values which correspond to the partition matrix column indices. Instead of joining the two columns of partition matrix explicitly, just the corresponding examples in this set are merged. For efficiency of such procedure, the original example set is sorted by the values of attributes in A. The result of such merging is the new example set E_G, while E_H is obtained by remembering which of the original columns are now represented with merged columns of the reduced partition matrix.

4.2.3 Properties of minimal-error single-step decomposition

Theorem 4.1 Let E_F be an example set decomposed to E_G and E_H using the attribute partition $A|B$. Then the estimated classification error of the decomposed system $y = F_G(A, c)$ and $c = F_H(B)$ is lower than or equal to that of original example set E_F.

Proof: The estimated error $\varepsilon(E_G)$ is equal to $\varepsilon(E_F)$ only in the case when no columns of $\mathcal{P}_{A|B}$ can be merged. Namely, it is the condition for $\mathcal{P}_{A|B}$ column merging that the estimated error does not increase. Therefore, $\varepsilon(E_G) \leq \varepsilon(E_F)$, since the entries in reduced $\mathcal{P}_{A|B}$ dirrectly correspond to examples in E_G. The sole role of the function $c = F_H(B)$ in such system is then to define to which of the merged columns does an example with specific values of attributes in B belong to.

Theorem 4.2 Each of the example sets E_G and E_H that are derived by decomposition of E_F uses less attributes and includes fewer or equal number of examples than the original set E_F.

Proof: The proof for the decreased number of attributes is the same as for Theorem 2.3. E_H would include the same number of examples only if in the corresponding partition matrix has a single entry per column and merging would not take place. In other cases, when at least two columns are merged, E_H contains fewer examples than E_F, i.e., $||E_H|| < ||E_F||$. The number of examples in E_G is equal to the number of nonempty entries in the partition matrix, which, when not reduced, is in turn equal to the number of examples in E_F. When merging takes place, the number of nonempty entries never increases but may decrease since some non-empty entries may be merged. Therefore, $||E_G|| \leq ||E_F||$.

Theorem 4.3 The decomposition of $y = F(X)$ to $y = G(A, c)$ and $c = H(B)$, where functions are represented with E_F, E_G, and E_H, respectively, preserves the class probability of class variable y when estimated from examples E_F and E_G.

Proof: This follows from the way decomposition handles the class distribution vectors. Namely, when merging two columns of partition matrix, the new class distributions are only introduced when two distributions are merged. Since this is done by summing up of two distributions, (1) the overall sum of all elements of the distributions remains the same, and (2) the overall sum of the distribution vector elements for particular class remains the same. Since this two define the class probabilities (Eq. 4.4), the latter is not changed after a single step decomposition.

4.2.4 Decomposition algorithm

The decomposition algorithm that can be used on the (possibly noisy) example sets with class distribution is essentially the same as the one described in the Chapter 2 (Algorithm 3.1). The only difference is the core of the algorithm, which is now the minimal-error single-step decomposition algorithm.

The minimal-error single-step decomposition algorithm derives the intermediate concept with specific number of values. In Chapter 2, this number was then used to compute different measures for partition selection and decomposability of the example set. Similarly, this measures can be used to guide decomposition of examples with class distributions as well. However, as a new single-step decomposition aims at minimizing the estimated classification error, we are inclined to design an overall decomposition with the same goal. The overall algorithm is then still the same as Algorithm 1.1, with the following changes. We assign each candidate partition $A|B$ an error $\varepsilon(A|B)$ which is equal to the estimated error of the reduced partition matrix for that partition (Algorithm 3.1). The new partition selection measure is then:

$$\Psi_\varepsilon(A|B) = \varepsilon(A|B) \tag{4.11}$$

The best partition is therefore the one that the most minimizes the expected error of the example set. Similarly, we can define the decomposability criteria of a set E_F as:

$$\Psi_\varepsilon(A|B) < \varepsilon(E_F) \tag{4.12}$$

The above criteria states that E_F is decomposable only if the partition $A|B$ is found with the reduced partition matrix with lower estimated error than that of E_F.

Example 4.5 In the example presented in the section 4.2.2 we have used the partition $A|B = \langle x_1\rangle|\langle x_2, x_3\rangle$ and for it derived a reduced partition matrix with $\varepsilon = 0.3486$. This error is also the corresponding partition selection measure, i.e., $\varepsilon(\langle x_1\rangle|\langle x_2, x_3\rangle) = 0.3486$. For the other two nontrivial partitions, these measures are $\varepsilon(\langle x_2\rangle|\langle x_1, x_3\rangle) = 0.3705$ and $\varepsilon(\langle x_3\rangle|\langle x_1, x_2\rangle) = 0.3558$. Therefore, the best partition to be used for decomposition is $\langle x_1\rangle|\langle x_2, x_3\rangle$. Using this partition the example set is also decomposable, since the estimated error of reduced partition matrix is lower than the error of the original example set $(0.3486 < 0.3933)$.

4.2.5 Complexity of decomposition algorithm

The two decomposition algorithms, i.e., the one described in this chapter and the one introduced in the chapter 2, are of the same complexity except for the complexity of the single-step decomposition.

Suppose that partition matrix has k columns, and the function being decomposed $c_i = F_i(X)$ is represented with a set E_F of N examples. The single-step decomposition first sorts the examples of E_F $(O(N \log N))$, estimates the error of each column of partition matrix

$(O(N||c_i||))$, tests all pairs of columns for merging $(O(Nkk^2||c_i||))$, and if for the best pair with the lowest error estimate this is lower then the error of the current partition matrix, merges the two columns $(O(Nk||c_i||))$. The whole procedure is repeated at most $k-1$ times. The complexity of single-step decomposition algorithm is therefore:

$$O\Big(N\log N + (k-1)(N||c_i|| + Nkk^2||c_i|| + Nk||c_i||)\Big) = O\Big(N\log N + k^4N||c_i||\Big) \tag{4.13}$$

Let k_{max} be the maximal number of columns of any partition matrix considered by decomposition. Let $||c||_{max}$ be the maximal number of classes used by the target or any concept discovered by decomposition. Using the derivation of the number of partitions to evaluate and the number of single-step decompositions to perform in section 2.3.4, the overall complexity of the decomposition algorithm is:

$$O\Big(n^{b+1}(N\log N + k_{max}^4 N||c||_{max}))\Big) \tag{4.14}$$

The time complexity of the minimal-error decomposition algorithm is therefore higher than that of the minimal-complexity decomposition algorithm. Comparison of Eq. 4.14 and Eq. 2.12 shows that this can be contributed to the term $k_{max}^4 N||c||_{max}$ of Eq. 4.14, which has its corresponding terms $Nk_{max} + k_{max}^2$ in Eq. 2.12.

Again, the algorithm's complexity is polynomial in N, n, and k_{max}, and to reduce the execution time, bound sets with just a few attributes should be used. Lower values for b do not only reduce the number of partitions being evaluated for decomposition, but also reduce k_{max}.

4.2.6 Selection of parameter m

Our new decomposition algorithm bases its estimation of error on m-error estimate (Eq. 4.2). m is the parameter of the estimation method, and, kept constant through the process of decomposition, a parameter of minimal-error decomposition method itself. Originally, Cestnik (1990) introduced m as a domain dependent parameter related to the level of noise in the domain: if little noise noise is expected, m should be small and should grow if the amount of noise is substantial. Further interpretations of m-probability estimate and m were proposed by Cestnik & Bratko (1991). They observed that m-probability estimate combines the apriori probability and the new evidence obtained from N examples, such that apriori probability was obtained from m prior examples and estimated with relative frequency. In other words, m can be related to the number of examples needed to estimate the probability of the class they represent.

Experimental study of Cestnik & Bratko (1991) indicated that the performance of the classifier may to a large extent depend on the choice of m. They propose two approaches to select m. In the first one, m can be adjusted by the domain expert that has a knowledge about the noisiness of the data, i.e., that knows how truthworthy particular sources of data are. The second approach involves two learning sets called *growing set* and *pruning set*, where growing set is used for learning and pruning set for testing. Then, the inducer learns on the growing set with different values of m. For each m, the performance of induced classifier is evaluated on the pruning set. Finally, m which yields a classifier of best performance is used to induce a classifier from the complete learning set.

The implementation and a detailed study of the second approach applied for the top-down induction of decision trees is presented in (Mladenić 1995). To estimate m, Mladenić uses an optimization technique that assesses the performance of the classifier induced using a specific value of m by k-fold cross validation: given a learning set, this is randomly split to k mutually exclusive subsets (folds) of approximately equal size. The decision tree is then induced and tested k times, each time using the k-th fold for testing and all other folds for learning. The accuracy is then estimated as the average accuracy from k folds. Usually, $k = 10$ is used. Note that the core idea is again to determine m from the learning set only.

The derivation of m is then defined as an optimization that uses the accuracy estimated by k-fold cross-validation as an optimization criterion. Mladenić applied different optimization algorithms (greedy search, simulated annealing, etc.) and by experimental study observed their similar performance.

We use a similar approach to select the value of m for decomposition. In the current implementation, no optimization algorithm is used and instead a predefined set of m values are evaluated on the learning set. Consequently, m that yields the best concept structures in terms of classification accuracy is selected. In our experiments (section 4.6), 10-fold cross-validation was used.

Although such approach has a nice property of automatically selecting the value of m, the definite drawback is its increased running time. Namely, for each m the decomposition has to be run 10 times, and if there are n_m different candidate values of m to be tested, the decomposition is run kn_m times. Nevertheless, for the decomposition to be completely unsupervised, this method is still used in our experiments. However, for practical uses, one should probably combine the automatic search with the adjustment of parameter m by the domain expert.

4.2.7 On (un)supervised decomposition and attribute subset selection

The minimal-error decomposition can be either applied in unsupervised or supervised mode. Again, the motivation and approaches to these are the same as given in section 2.4. In the experiments at the end of this chapter we use the decomposition only in the unsupervised mode. However, we strongly believe that in the detailed studies and analysis of particular dataset a supervised decomposition has to be a method of choice.

In regard to the attribute subset selection and discovery of redundancies, the methods described in chapter 3 can be directly applied. The only difference is that instead of using minimal-complexity the method now use minimal-error single-step decomposition. The method can be further extended so that instead of proposed measures for attribute relevance, only the potentially redundant attributes are considered and the one which yields a partition matrix of lowest estimated error is removed.

4.3 Representing imperfect data by examples with class distributions

The real-world data most often includes noise and/or uses incomplete description language to describe the examples. Noise, for example, may occur because of imperfect measuring equipment used to collect the data. The incompleteness of description language may contribute to the existence of the examples with same attribute values but different class.

Another common problem, in particular with data in medical domains (Diamantidis & Giakoumakis 1996) are also missing attribute values. There, the attribute values are either unavailable (say some expensive medical test was not made for that subject) or they are irrelevant. These two cases are usually referred to as the problem of *don't know* and *don't care* values. It is important for a machine learning tool to distinguish between these two cases and appropriately deal with them.

In the previous chapters the decomposition assumed that all examples are equally important. However, there may be cases where additional information about the weight of each example is provided. Furthermore, each example may describe not just one but rather several classes, each with its own factor of belief. Such cases may be common if human is involved in assigning the classes to each of examples. For example, a patient may be given several possible diagnoses, each with corresponding belief.

Lavrač & Džeroski (1994) refer to the data that has any combination of the properties above as *imperfect data*. This section shows how such data can be transformed in the set of examples that use class distributions and can be further used for decomposition.

4.3.1 Noisy and inconsistent data

The conversion from noisy and inconsistent set of examples to the set that uses class distributions was discussed in section 4.1. Let us here further illustrate the approach by a simple example. Suppose that we are given the following inconsistent example set:

med, lo, lo	med
med, lo, lo	hi
med, lo, lo	med

This set can be represented with a single example that uses class distribution and is defined as:

```
med, lo, lo       (0,1,2)
```

4.3.2 Uncertain data

We distinguish among the uncertainty of examples, their classes, and their values of attributes. For the later, we examine the case of missing attribute values that may be either treated as *don't-knows* or *don't-cares.*

Uncertainty of examples

Each example in the original example set may be assigned a degree of confidence to which it is believed that such example really represents the assigned class. This may also be regarded as the weight of the example. For instance, the examples dataset

```
med, lo, lo       med/3
med, lo, lo       hi/2
med, lo, hi       med/0.5
```

where the numbers beside the classes denote their corresponding weight. These are converted to the following set of examples with class distributions

```
med, lo, lo       (0,3,2)
med, lo, hi       (0,0.5,0)
```

Again, the conversion is straightforward, with the weight of each example being used to form the distribution vector of the corresponding new example.

Uncertainty of classes

Similarly as above, we can for instance define an example

```
med, lo, lo       lo/0.2, med/0.8
```

which defines two classes with corresponding certainties. Note that this example could be equivalently represented with two separate examples

```
med, lo, lo       lo/0.2
med, lo, lo       med/0.8
```

An obvious representation with class distribution is

```
med, lo, lo       (0.2,0.8,0)
```

Missing attribute values

When an example is defined with some values of attributes missing, there may be two cases for each unknown values: either (1) such value is unknown for example due to lack of measurement, or (2) is not specified because it is irrelevant for that particular example. The first case is most often referred to as the problems of *don't-know*, and the second as the problem of *don't-care* values.

Recent work of Diamantidis & Giakoumakis (1996) states that the distinction of the two by inductive algorithm may be specifically important in the medical domains. They claimed that usually the inductive algorithms treated missing-values as don't-knows, whereas it may be beneficial if those that really express don't-cares are treated separately. The experimental argument they provided was based on a set of medical domains, for which the missing values were usually treated as don't-knows. They show that by replacing some, randomly chosen don't-cares with don't-knows, the induction may result in significantly higher classification accuracy. Although such experimentation may look superficial and a better experimental setup would involve medical experts that would indeed classify each of the missing values to two different cases, it provides a motivation for induction algorithm to treat these cases separately.

Again, we handle both don't-cares and don't-knows by conversion of corresponding examples to the example set that uses class distributions. For don't-cares, the conversion is as follows. Let an example use the attributes $a_1, \ldots, a_n$ with missing values of don't care type. This example is then represented by an expanded set of $\prod_{i=1}^{n} ||a_i||$ examples, each one using its own combination of values of attributes $a_1, \ldots, a_n$. For instance, an example

```
med, -, -      lo
```

of which the attributes and class have domains as defined in Example 4.1, is expanded to examples

```
med, lo, lo    lo
med, lo, hi    lo
med, hi, lo    lo
med, hi, hi    lo
```

To treat the missing values as don't-knows, the similar expansion to the set of examples is used, with a distinction that this time a corresponding weight is assigned to each of the new examples. Let $p(a_i^j)$ be a probability of j-th value of i-th attribute estimated from the example set. Let v_i be a value of i-th attribute from the set of don't-know attributes $a_1, \ldots, a_n$. Let $p(v_i) = p(a_i^j)$ be the probability of this value, such that $j = v_i$. The weight for this example is then defined as $\prod_{i=1}^{n} p(v_i)$.

For example, let the attribute x_2 with possible values `lo` and `hi`, have estimated probabilities $p(x_2 = \texttt{lo}) = 0.2$, $p(x_2 = \texttt{hi}) = 0.8$, and attribute x_3 with same set of possible values have $p(x_2 = \texttt{lo}) = 0.5$ and $p(x_2 = \texttt{hi}) = 0.5$. Then, the example

```
med, -, -       lo
```

has the following corresponding representation using class distributions:

```
med, lo, lo     lo/0.1
med, lo, hi     lo/0.1
med, hi, lo     lo/0.4
med, hi, hi     lo/0.4
```

This set can be straightforwardly converted to representation with class distributions as explained in section 4.3.2.

The main distinction between don't-cares and don't-knows regarding the performance of the decomposition algorithm is that expanding the examples with don't-cares may be particularly useful, since they may contribute to better coverage of the attribute space. On the other hand, the examples with many don't-knows may be expanded to a large number of examples of very small weight; these may affect very much the algorithms running time, but not its classification accuracy. Namely, the examples with small weights may play no role both because of being out-weighted by other examples or because of their insignificant affect when used for error estimation. A straightforward way to deal with this is to set a weight limit for an example to be considered at all. In our experiments, we have used this limit just for a single domain (HEPA), where the limit was set to 0.001.

4.4 Classification

In section 2.5 we have described the classification of attribute-value vectors using the concept hierarchy derived by decomposition.

Minimal-error decomposition derives the example sets with class distribution, but when used for classification, each example classifies to a single class, i.e., a class with the corresponding highest value in the distribution vector. Therefore, the classification algorithm is the same as the one used for concept structures derived by minimal-complexity decomposition (see section 2.5).

However, to be able to handle attribute-value vectors with some attribute values missing, we have to extend this algorithm. The extension first assumes that each of the attributes is given by its normalized value distribution vector. Let E_F represent a function $c = F(x_1, \ldots, x_m)$. Let each attribute x_i be given with a normalized distribution of its values $\mathbf{v_i}$, such that $\prod_{j=1}^{||x_i||} v_i^j = 1$. The goal is now to determine the value of $c = F(\mathbf{v_1}, \ldots, \mathbf{v_m})$, which is expressed as a distribution v_c. To derive v_c, we first construct a set of attribute-value vectors with all possible combinations of attribute values. Each such vector with attribute values $\langle x_1^k, \ldots x_m^k \rangle$ is then assigned a weight w_k

$$w_k = \prod_{i=1}^{m} v_i^{x_i^k} \tag{4.15}$$

Note that since $\mathbf{v_i}$ are normalized, the sum of weights $\sum_k w_k$ is equal to 1. Only the attribute-value vectors with non-zero weight are considered. For each such k-th vector, a corresponding value of c denoted by c^k is determined by either:

- finding an example in $e_i \in E_F$ with the same value of the attributes; the value v_c^k is then the value of c that example e_i classifies to,
- if such example is not found, the default rule is used which assigns v_c^k the most probable of the values of c as estimated from the set of examples E_F.

The elements of resulting c value distribution are then computed as

$$v_c^i = \sum_{c^k = i} w_k \tag{4.16}$$

and then normalized to be further used in the derivation of the higher-level concepts in the hierarchy.

For missing attribute values and in case of don't-cares, the value of attribute x_i is then represented with a value distribution vector with elements equal to $1/||x_i||$. In case of don't-knows, the corresponding distribution vector is equal to a vector of apriori probabilities of attribute values as estimated from the learning set that was originally used for decomposition.

Example 4.6 Consider a system of functions $y = G(x_1, c)$ and $c = H(x_2, x_3)$ which are represented by decomposition-derived example sets in Table 4.5. Let us determine the value of y for $x_1 =$ `hi`, $x_2 =$ `hi`, and x_3 not defined and treated as don't-care. First, we need to derive $c = H(hi, \_)$. The attribute x_3 is then represented as a distribution $(0.5, 0.5)$. Using the values of x_2 and x_3 the set of vectors with non-zero weight and their corresponding c values from example set E_G are:

k	x_2	x_3	w_k	c^k
1	hi	lo	0.5	0
2	hi	hi	0.5	1

The distribution for values of c is then (0.5,0.5). This, together with a value of x_1, is now used to derive y. The attribute-vectors, their weights, and corresponding values of y needed for this derivation are:

k	x_1	c	w_k	c^k
1	hi	lo	0.5	hi
2	hi	hi	0.5	hi

The resulting normalized distribution for y is therefore $(0, 0, 1)$, so the final value of y is `hi`.

4.5 Implementation

The error-guided decomposition is implemented within the program HINT(Hierarchy INduction Tool), which is thus a common environment for minimal-complexity and minimal-error decompositions. Further details on HINT are given in Appendix F.

4.6 Experimental evaluation

In this section, we experimentally assess the performance of the minimal-error decomposition to discover the structures inherited in the data, and the performance of discovered concept hierarchies in terms of classification accuracy. In particular, we investigate how this performance is affected by noise in the data.

To address both issues, we have selected three typical domains used in section 2.7. Originally, these are noiseless domains, but were for the study in this section corrupted with class noise.

Next, the decomposition performance on the domain MONK3 is evaluated. This domain, like MONK1 and MONK2 (see section 2.7.4), was originally used in the study of classification accuracy of several machine learning algorithms (Thrun et al. 1991). We compare the results obtained in this study to the classification accuracy of the minimal-error decomposition and also compare the structure discovered to the original definition of the problem.

To a further extent, it is hard to evaluate the performance of minimal-error decomposition and compare it to other machine learning methods. Namely, for most of real-world domains available (for example, those at UCI Machine Learning Repository (Murphy & Aha 1994)), insufficient description is provided for someone not familiar with the domain to evaluate the appropriateness of the concept structures derived by HINT. Therefore, we could further evaluate HINT's ability to handle imperfect data only by means of classification accuracy. For this, we have selected seven medical domains and compared the performance of HINT to those of other inducers investigated in the study of Kononenko, Šimec & Robnik Šikonja (1996).

4.6.1 MM4, CAR, and SHUTTLE

We have selected the domains MM4, CAR, and SHUTTLE from section 2.7 and repeated similar experiments to derive the learning curves and to observe how the concept structures differ from the anticipated ones (for MM4 and CAR only). This time, however, we have corrupted the learning sets with 10%, 20%, or 30% of class noise. This effectively means that, for example, with 10% of class noise the class of 10% of examples in the learning set was set arbitrarily.

Again, the results were averaged across 10 experiments for each size of the learning set. For each of this experiments, the value of m was determined solely on the base of the learning set using 10-fold cross-validation technique discussed in section 4.2.6. The possible set from which m were selected included 20 different values: 0.1 to 0.9 with steps of 0.1, 1 to 3 with steps of 0.2, 4 to 10 with steps of 1. To determine each data point in the learning curve of HINT, decomposition was therefore run $10 \times 20 = 200$ times. This number could be significantly reduced if the user would be involved in setting more specific set of m values, but since we were interested in the unsupervised performance of decomposition, we refrained to do that.

To evaluate the classification accuracy of HINT, the results were compared to those obtained by C4.5. This time, C4.5 was run with the default options. For the classification the pruned C4.5 trees were used. The significance of the difference between C4.5 and HINT was determined using a t-test with $\alpha = 0.01$ (99% confidence level). Where no inducer is significantly better, the learning curves use symbols $\circ$ for HINT and $\diamond$ for C4.5. When HINT is significantly better, the symbol $\bullet$ is used, while when this is true for C4.5, the data points are marked with $\blacklozenge$.

The learning curves for CAR are given in Figure 4.1. They are quite similar to that with noiseless domain and minimal-complexity decomposition (see Figure 2.10), but the convergence to 100% classification accuracy is slower. As the noise level increases, this seems to most affect the performance of HINT with small learning sets. Nevertheless, for this domain, HINT's classification was significantly better than C4.5's in most cases.

CAR's structure dissimilarity coefficient when derived concept hierarchies are compared to the original DEX model are given in Figure 4.2. For all levels of noise, the decomposition converged to the same concept structure. This was exactly the same one as for the noiseless case (see Figure 2.12). With higher percentage of class noise, the convergence is slower, and the standard deviations of structure dissimilarity index are higher.

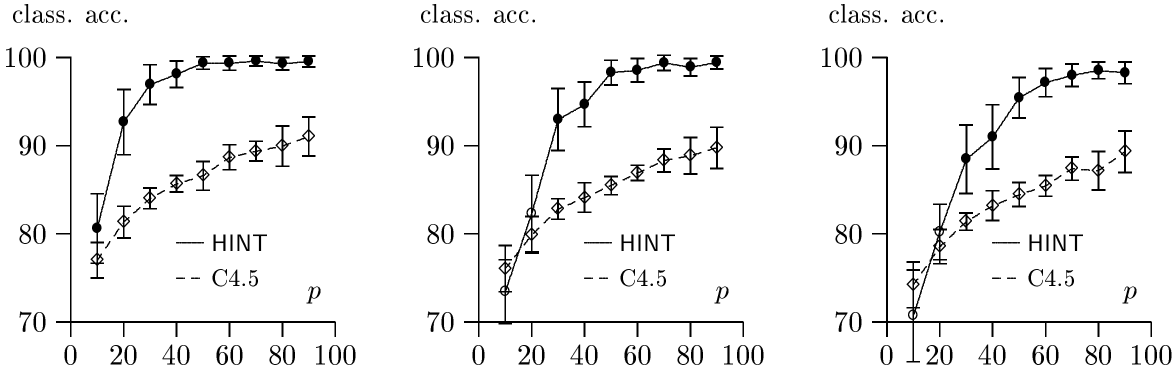


Figure 4.1: Learning curves for CAR with 10%, 20%, and 30% of class noise.

Compared to the CAR domain, similar conclusion can be drawn about the convergence of learning curves for MM4. The structure dissimilarity index for MM4, which compares the derived structure to the anticipated one (Figure 2.8.a), does not exhibit any recognizable dependence on the size of learning set. However, the averages of SDC are around 0.6, which was also the expected value (compare with the corresponding graph in Figure 2.7).

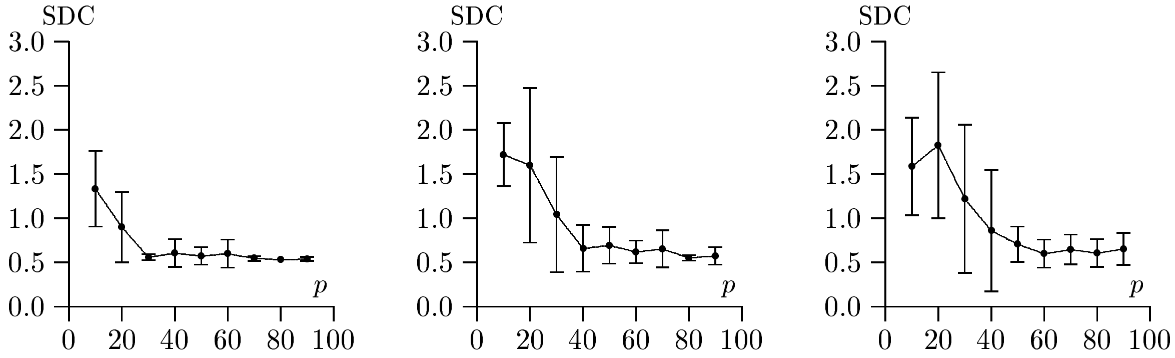


Figure 4.2: Structure disssimilarity index for CAR domain and 10%, 20%, and 30% noise.

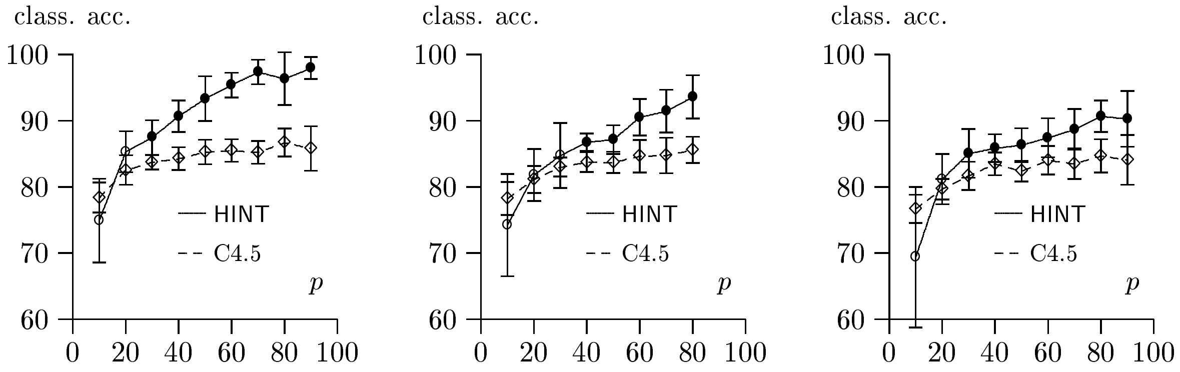


Figure 4.3: Learning curves for MM4 with 10%, 20%, and 30% of class noise.

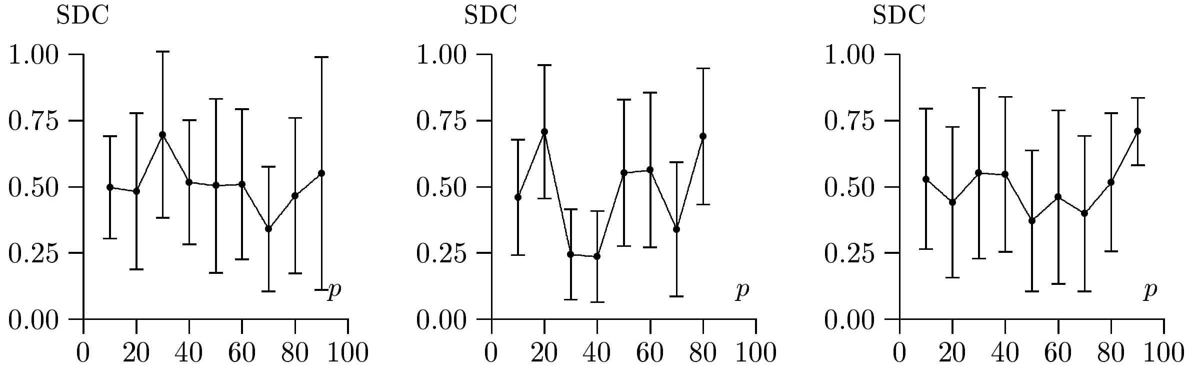


Figure 4.4: Structure disssimilarity index for MM4 domain and 10%, 20%, and 30% noise.

For the SHUTTLE domain, Figure 4.5 gives the learning curves for different values of class noise. This time, the minimal-error decomposition does not outperform C4.5, and when learning sets are small, C4.5 is significantly better. This is interesting to compare to noiseless case and minimal-complexity decomposition: there, HINT also performed relatively worse that with other domains like MM4 and CAR. A possible explanation may be that SHUTTLE does not possess a structure that would be recognizable by decomposition and that would be beneficial in terms of decomposition-induced classifier. We should not forget, though, that in both cases unsupervised decomposition was used, and that such problems may still benefit from the supervision of the domain expert.

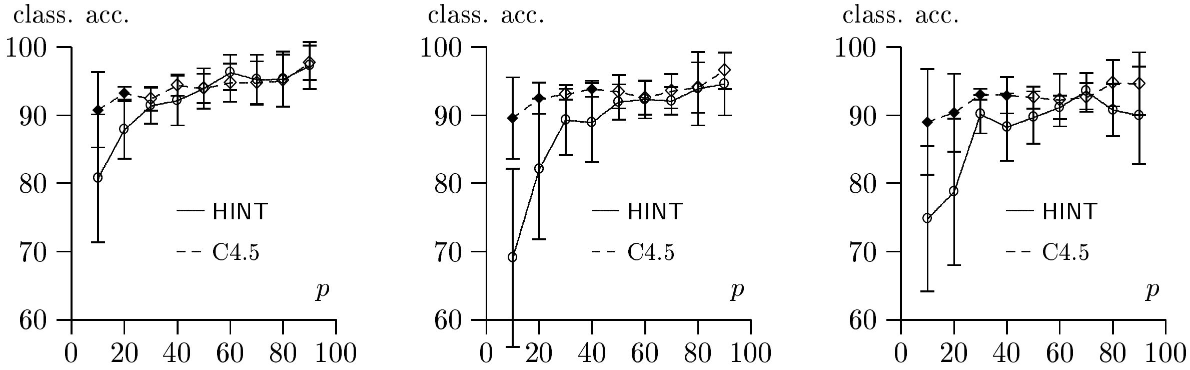


Figure 4.5: Results for learning in SHUTTLE domain

As described above, for every learning set HINT derived a value of m which minimized the classification error as estimated by 10-fold cross validation on learning set. Figure 4.6 gives these values when averaged over all experiments with the same noise level for each domain. As expected, for all the domains used in this section the value of m increases with noise level.

domain	10%	20%	30%
MM4	0.9 ±0.6	2.0 ±1.1	3.6 ±1.9
CAR	0.7 ±0.3	1.3 ±0.6	1.7 ±0.5
SHUTTLE	1.4 ±0.5	2.6 ±0.3	2.8 ±0.7

Table 4.6: Discovered values for m for different noise levels

4.6.2 MONK3

The detailed study of 25 machine learning algorithms in (Thrun et al. 1991) mentioned in section 2.7.4 used also a domain MONK3 with the target concept defined as:

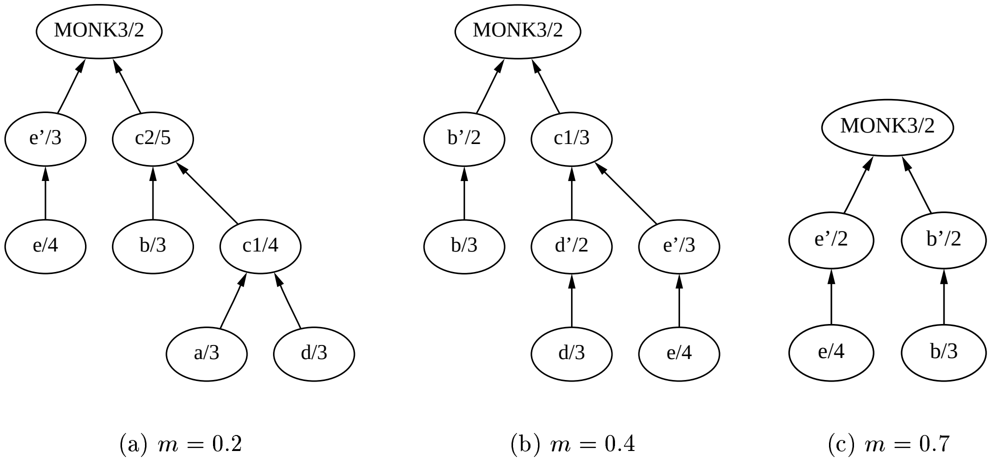
```
MONK3 = e=3 AND d=1 OR e≠4 AND b≠3
```

This binary classification problem uses two 2-valued attributes (c and f), three 3-valued attributes (a, b, and d), and a 4-valued attribute e. The problem defined in the study was to learn this concept from a sample with 122 examples where 5% of examples were subject to class noise. To test the classifier, a set of 432 examples that covered the complete attribute space and are consistent with the target concept definition were provided.

HINT used attribute subset selection and decomposition, both driven by error-minimization. Using only the examples in the learning set, HINT derived the value of $m = 0.4$ to be the most appropriate for this domain. The concept structure induced with this value of m (Figure 4.6.b) correctly classified all the examples in the test set. This score was in the study of Thrun et al. (1991) achieved only by a few machine learning tools (see Appendix D).

It is also interesting to observe the example sets HINT discovered. These are given in Table 4.7. First, note that the redundant attributes a, c, and f are not present and were removed by HINT's attribute subset selection algorithm. Next, the example sets for b', d', and e' reveal that HINT successfully found the groups of values which are relevant to the target concept. The interpretation of examples for the intermediate concept c1 reveals that c1=2 if e=4, c1=1 if d=1 and e=3, and c1=0 otherwise. Then, MONK3=1 if c1=2, or if c1=1 and b≠3, which is an equivalent redefinition of the original concept for MONK3.

Further experiments show that for MONK3 HINT induces the same concept structure when $0.4 \leq m \leq 0.6$. This structure is given in Figure 4.6.b. When $m < 0.4$, the discovered structures are more complex and less accurate. Figure 4.6.a gives a structure discovered with $m = 0.2$ that misclassified 4.63% of examples in the test set. When $m > 0.6$, HINT discovers less complex but again less accurate concept structures. For example, when $m = 0.7$, the structure discovered is the one at Figure 4.6.b, with the misclassification rate of 2.78%.



(a) $m = 0.2$ (b) $m = 0.4$ (c) $m = 0.7$

Figure 4.6: Concept structures for MONK3 discovered by HINT when different values of m were used.

b	b’	
1	0	(8.00 0.00)
2	0	(7.00 0.00)
3	1	(0.00 8.00)

d	d’	
1	0	(35.00 0.00)
2	1	(0.00 31.00)
3	1	(0.00 40.00)

e	e’	
1	0	(4.00 0.00 0.00)
2	0	(4.00 0.00 0.00)
3	1	(0.00 4.00 0.00)
4	2	(0.00 0.00 4.00)

d’	e’	c1	
1	2	2	(0.00 0.00 2.00)
1	1	0	(2.00 0.00 0.00)
1	0	0	(2.00 0.00 0.00)
0	2	2	(0.00 0.00 2.00)
0	1	1	(0.00 2.00 0.00)
0	0	0	(2.00 0.00 0.00)

b’	c	MONK3	
1	2	0	(11.00 1.00)
1	1	1	(0.00 2.00)
1	0	0	(27.00 0.00)
0	2	0	(19.00 0.00)
0	1	1	(1.00 5.00)
0	0	1	(4.00 52.00)

Table 4.7: Decomposed example sets discovered by HINT. For each example the majority class and class distribution vector is given.

4.6.3 Seven medical domains

To further compare the performance of HINT to other machine learning tools, this section uses seven medical datasets. Although used in many different studies, because of the recency and inclusion of several learning algorithms we have chosen to compare the results with those of Kononenko et al. (1996).

The medical datasets used are:

- PRIM: the problem of locating a primary tumor in patients with metastases,
- BREA: the problem of predicting the recurrence of breast cancer five years after the removal of the tumor,
- LYMP: the problem of determining the type of cancer in lymphografy,
- RHEU: the diagnostic problem in rheumatology,
- HEPA: the prognostics of survival of patients suffering from hepatitis,
- DIAB: the diagnostic data for diabetes,
- HEART: diagnosis of hearth diseases.

Originally, PRIM, BREA, LYMP, and RHEU are the datasets from University Medical Center in Ljubljana, Slovenia, HEPA was provided by Gail Gong from Carnegie-Mellon University, and DIAB and HEART were obtained from StatLog database (Michie et al. 1994).

Most of the datasets include continuously-valued attributes. We use the same attribute discretization as used in the study of Kononenko et al. (1996) discretized.

Basic description of the medical datasets is given in Figure 4.8. It should be further noted that these datasets include noise, and most include examples with missing values as well.

domain	#class	#atts.	#val/att.	#examples	maj. class (%)
PRIM	22	17	2.2	339	25
BREA	2	10	2.7	288	80
LYMP	4	18	3.3	148	55
RHEU	6	32	9.1	355	66
HEPA	2	19	3.8	155	79
DIAB	2	8	8.8	768	65
HEART	2	13	5.0	270	56

Table 4.8: Basic description of the medical datasets

The classification accuracy evaluation method was the same as in (Kononenko et al. 1996): each set was randomly split to the learning set that included 70% of examples and to test set with 30% of examples. For each domain, the results are an average of 30 such experiments.

The performance of HINT is compared to several other classifiers. The results are taken from (Kononenko et al. 1996) and were not obtained using the same learning and test sets. Assistant-I, Assistant-R, and LFC all induce decision trees. Assistant-I uses information gain and Assistant-R uses RELIEFF as an attribute selection criterion. LFC is a decision-tree inducer additionally equipped with constructive induction algorithm (Lookahead Feature Construction). Other two classification tools used are naive Bayesian classifier and k-nearest neighbor algorithm. For further details on these methods see (Kononenko et al. 1996).

The classification accuracy results are shown in Table 4.9. HINT was run in two different ways: for the first, HINT derived an appropriate value of m as described in section 4.2.6. Another technique tried to estimate the performance of HINT if one would know the appropriate value of m in advance: for all 30 experiments, HINT was run using a constant value for m. Such evaluation was repeated for a set of m values, and classification accuracy of experiments with m that yielded the best result is shown (row labeled with HINT*). We will denote the later experimental setup with HINT*. In all cases, HINT preprocessed the example sets with attribute subset selection algorithm as proposed in Chapter 3.

We were especially interested to compare the performance of HINT to that of Assistant and LFC tools. The reason is that these are all state-of-the-art symbolic inducers, i.e., induce a classifier in the form of decision tree that can be further used to reason about the properties that are hidden in the data. HINT performs worse than these inducers for PRIM, LYMP, and HEART domains, better for RHEU and DIAB domain, and comparable for BREA and HEPA domain. Overall, it can be concluded that none of these learning methods exhibits a clear advantage over the others.

The Bayesian classifier performed better than any of symbolic inducers used. Kononenko et al. (1996) contribute this success to the property of medical data sets where attributes are typically conditionally independent given the class.

Also note that HINT* always performed better than HINT, though for most of domains the differences were rather small. This result was expected, as HINT may have difficulties to estimate m when datasets sparsely cover the attribute space. The values of m found by HINT and the value used for the best set of experiments by HINT* are given in Figure 4.10.

	PRIM	BREA	LYMP	RHEU	HEPA	DIAB	HEART
HINT	36.8±4.4	78.2±3.7	72.9±6.6	65.7±3.9	78.2±5.0	73.1±2.6	73.0±8.4
HINT*	36.9±4.9	79.6±3.7	74.1±6.3	66.3±4.0	79.0±5.6	73.7±2.7	76.2±5.5
Assistant-I	40.8±5.1	76.8±4.6	77.0±5.5	64.8±4.0	77.2±5.3	71.1±2.8	77.6±4.5
Assistant-R	38.9±4.7	78.5±3.9	77.0±5.9	63.8±4.9	82.3±5.4	71.5±2.6	77.6±4.5
LFC	37.1±4.9	76.1±4.3	82.4±5.2	60.6±4.7	79.0±5.3	69.2±3.0	77.3±5.2
naive Bayes	48.6±4.1	78.7±4.6	84.7±4.2	66.5±4.0	86.1±3.9	76.3±2.4	84.5±3.0
k-NN	42.1±5.0	79.5±2.7	82.6±5.7	66.0±3.6	82.6±4.9	73.9±2.5	82.9±3.7

Table 4.9: Classification accuracy of different learning systems on medical datasets

	PRIM	BREA	LYMP	RHEU	HEPA	DIAB	HEART
HINT	4.9±2.3	8.5±5.6	4.3±2.21	7.1±1.9	4.2±2.43	6.0±2.2	3.7±2.43
HINT*	4.0	18.0	3.2	9.7	7.5	6.7	2.4

Table 4.10: The value of m derived by HINT using cross-validation on learning set (first row), and the value of m consistently used for 30 experiments that yield the best classification accuracy (second row).

To further study the effects of m on performance of HINT, we have used DIAB and HEPA domains and the experimental setup of HINT*. The results are shown on Figures 4.7 and 4.8. Classification accuracy first raises with m, and then settles down after some value of m. A higher value of m that are not given in these figures are required for the accuracy to drop down again. As expected, the number of attributes that remain after preprocessing by attribute selection decreases with m. Similar is true for the overall complexity of discovered functions. This is consistent with the use of m for minimal-error pruning of decision trees (Cestnik & Bratko 1991), where with higher value of m the resulting trees are less complex.

The concept structures discovered for the medical domains from this section are given in Appendix E. Because of the unavailability of the domain experts, we could not interpret these structures. Nevertheless, one can notice that they all use substantially fewer attributes

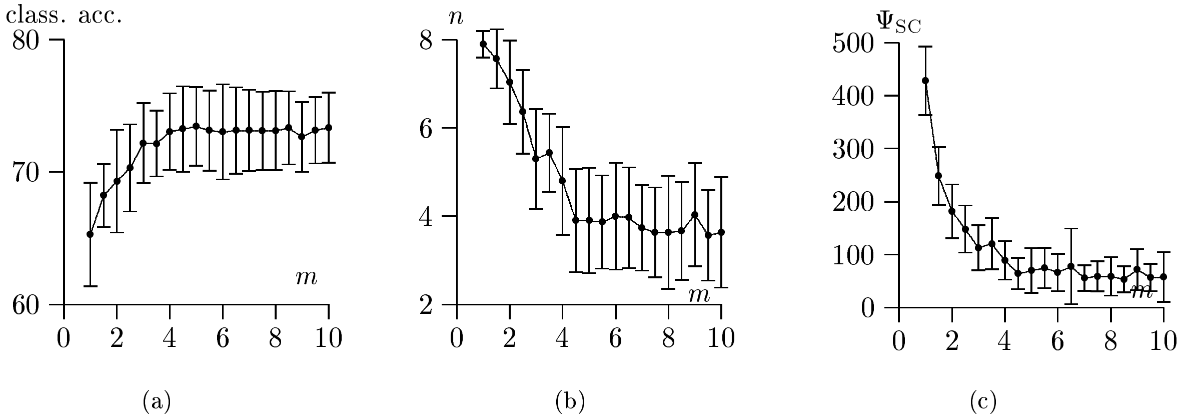


Figure 4.7: Classification accuracy (a), number of attributes used (b), and complexity of the concept structure (c) for DIAB as a function of m.

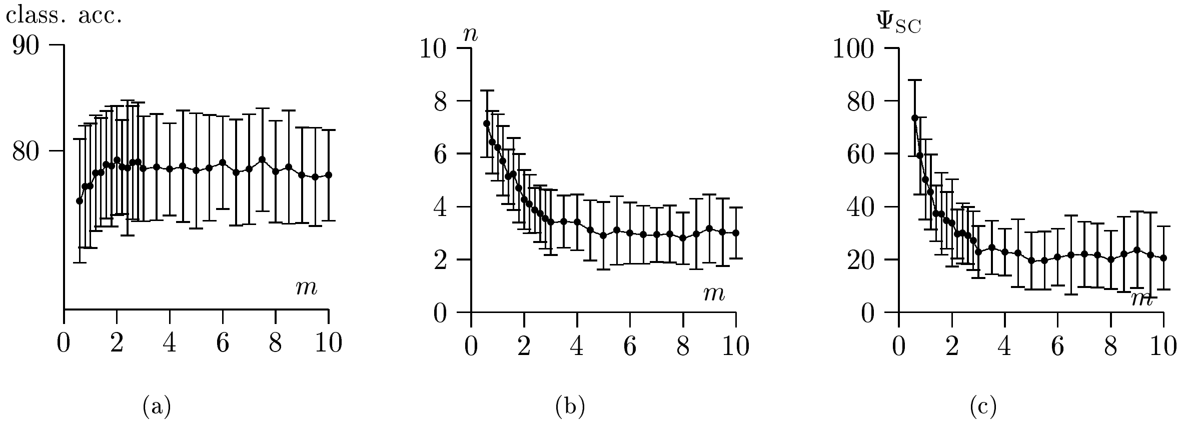


Figure 4.8: Classification accuracy (a), number of attributes used (b), and complexity of the concept structure (c) for HEPA as a function of m.

than included in the original example sets. For example, structure for RHEU uses only 11 attributes instead of 32, and DIAB only 3 instead of 8. Therefore, for these domains, the critical phase may be the attribute subset selection that removed the redundant attributes.

To conclude, let us note that the presented set of experiments used HINT in the unsupervised mode. The real challenge for decomposition would rather be to be used in the data-exploratory process with a medical expert at the side. Such process may more efficiently reveal the potential intermediate concepts that may be hidden in these medical domains. We expect that such supervised approach to decomposition may not only result in more meaningful and appropriate structures, but may also improve the performance of HINT in terms of classification accuracy.

4.7 Summary and discussion

The development of decomposition method from this chapter was motivated by the inability of minimal-complexity decomposition to handle noise and inconsistent data. The resulting minimal-error decomposition algorithm is from minimal-complexity variant substantially different in techniques for single-step decomposition, heuristics for partition selection measure, and decomposability criteria. Overall, though, the fundamental principle of decomposing the set of examples to smaller and more manageable sets remains. This is demonstrated also by the overall decomposition algorithm, which is the same for the two approaches. Consequently, the attribute subset selection algorithm introduced in the previous chapter can be with no difference used in the minimal-error decomposition to detect and remove the redundancies, and can therefore be used on noisy and inconsistent data as well.

The minimal-error decomposition requires a representation of examples that use class distributions. Eventually, this not only allows to extend the decomposition algorithm, but also to appropriately handle a wider variety of data properties: inconsistencies, noise, and uncertainty. Most often, these are also the properties of real-world data, and in this sense the minimal-error decomposition can be used on a wider range of domains than its minimal-complexity variant.

It has to be noted that a combined variant of the algorithms can be considered as well. For example, a single-step decomposition may be driven by error minimization, while some complexity-based measure may be used for partition selection. Not to investigate over too broad variants of decomposition, we intentionally kept the two overall decomposition approaches being driven by a single bias.

The experimental evaluation again confirmed that decomposition may, even in the presence of noise, perform well both in terms of classification accuracy and derivation of meaningful concept hierarchies. For example, for MONK3, the decomposition not only correctly classified all the examples in the test set, but also induced an interpretable and appropriate concept structure. The toughest test on decomposition in regard to its classification accuracy performance were seven different medical domains. There, the sparse coverage of attribute space may present the problem for decomposition. Still, the classification accuracy was overall comparable to that of state-of-the-art symbolic inducers.

Chapter 5

Conclusion

5.1 Summary and discussion on decomposition method

The dissertation proposes a new machine learning paradigm that is based on function decomposition. Given a set of training examples, the proposed decomposition induces a definition of the target concept in terms of a hierarchy of intermediate concepts and their definitions. This effectively decomposes the problem into smaller, less complex problems by decomposing an initial set of examples to smaller and more manageable sets.

The core of the decomposition algorithm is a *single-step decomposition*: given a set of examples E_F that partially specify a function $c_i = F(X)$ it decomposes it to example sets E_G and E_H. The new sets partially describe the functions $c_i = G(A, c_j)$ and $c_j = H(B)$. X is an initial set of attributes, and A and B its nontrivial partitions. c_j is a new, intermediate concept.

The overall decomposition algorithm uses the single-step decomposition to (1) find the best partition of attribute set X to sets A and B, (2) to determine if E_F is decomposable using sets A and B and (3) to decompose E_F to E_G and E_H. Since the decomposition can be applied recursively on E_G and E_H, the result in general is a hierarchy of concepts.

Two different approaches to decomposition are proposed. The minimal-complexity approach aims at deriving a final set of examples that represent functions of minimal complexity. The minimal-error approach aims to derive a hierarchy of concepts with minimal estimated classification error.

The proposed approaches to decomposition are further different in the type of data they can handle. The minimal-complexity approach can appropriately handle only the example sets that are consistent. On the other hand, the minimal-error approach allows to decompose imperfect examples sets that potentially include noise, missing values and uncertainty. In this sense, the minimal-error decomposition approach is more general that minimal-complexity variant. On the other hand, the minimal-error decomposition algorithm is of higher computational complexity, and for this reason one may prefer minimal-complexity method when dealing with consistent set of examples. Both approaches handle continuous attributes by discretization.

Both decomposition approaches can be used to detect and remove potential redundancies in the data. These include either the redundancies of attributes or redundancies of attribute values. The dissertation proposes to include these redundancy handling mechanism into the attribute subset selection algorithm, that aims to (1) reduce the number of attribute used in non-decomposed example set and thus speed-up the decomposition, and (2) make a judged decision based on attribute relevancy on which attributes to exclude from the dataset. Namely, both decomposition approaches can both detect and handle the redundancies of the set of attributes while the decomposition takes place, but if several such sets exist, the one excluded would be chosen arbitrarily.

5.2 Discussion of experimental results

The dissertation experimentally evaluates the proposed methods. There are two evaluation criteria considered: (1) the decomposition's ability to derive a comprehensible and meaningful concept structure, and (2) the classification accuracy of the derived concept structure.

To assess the first one, artificial and real-world example sets were used for which the structure was either known in advance (DEX models), anticipated (different functions, including those of MONK domains), or could be evaluated by a domain expert (an example from neurophysiology). The experiments for minimal-error decomposition used a selection of training sets used to evaluate the minimal-complexity variant, this time spoiled with class noise. The experimental evaluation showed that decomposition can discover useful and interpretable concept hierarchy that were in most cases at least similar if not equal to those expected.

The classification accuracy of decomposition was compared to the state-of-the-art inducer C4.5. The comparison showed that decomposition generalizes well and can, in some cases, even have a significantly better classification accuracy than C4.5.

Relative to C4.5, the classification performance of decomposition seems to be better when training sets better cover the attribute space. The learning curves of C4.5 are more "flat" than those of decomposition, which tend to a higher extend lean downward with the decreased training set sizes. Specific to noise-handling, this sensitivity to attribute space coverage seems to increase with the level of noise in the data.

The experiments further indicated an interesting parallel between minimal-complexity and minimal-error approaches: on specific domain and in regard to C4.5, the two approaches seem to perform similarly. For example, when the minimal-complexity algorithm performed better compared to C4.5, the minimal-error approach on the same domain spoiled with class noise performed better as well. On the other side, where there was no significant difference between C4.5 and minimal-complexity decomposition, same was noticed for the minimal-error approach.

The possible explanation is that the decomposition as a paradigm prefers the datasets that have some inherited structure that can be discovered by decomposition. It is a problem to apriori know if such a structure exists. For instance, in MONK2 domain, based on the definition of the concept it was evident that such structure could not be discovered by decomposition and one would consequently expect for decomposition not to perform well in this case. However, the decomposition discovered a structure which was an equivalent reformulation of the solution, and in terms of classification accuracy performed significantly better than C4.5.

5.3 Contributions

The contributions of the dissertation include:

- the minimal-complexity decomposition approach to inductive concept learning, which is inspired by a Boolean function decomposition approach to design of digital circuits, extended by:
  - new method for handling multi-valued attributes,
  - improved decomposition heuristics, which include partition selection measures, decomposability criteria, and an extension of graph coloring approach by using the degree of compatibility.
- the minimal-error decomposition approach that features:
  - the decomposition of example sets that use class distribution,
  - minimal-error approach to single-step decomposition,
  - minimal-error definition of partition selection measure and decomposability,
  - mechanisms to handle inconsistent and noisy data, and data with missing values and uncertainty,
  - possibility to derive structures of different complexity by accordingly setting the m parameter,
- the definition of supervised and unsupervised approach to decomposition,
- the attribute subset selection algorithm that uses decomposition-based discovery and redundancy removal,
- the implementation of the proposed methods in the program called HINT(Hierarchy INduction Tool),

- the experimental assessment of the decomposition algorithms both from the viewpoint of classification accuracy and discovery of concept structures; in brief, the experiments show that:
  - the decomposition can discover useful and interpretable structures,
  - the decomposition can inductively learn concepts with at least comparable classification accuracy to those build by other machine learning tools,
  - the minimal-error decomposition can efficiently handle noise and missing data.

The dissertation contributes to the related research areas as follows:

- to *machine learning* as a new inductive concept learning paradigm,
- to *constructive induction* as a method to construct new features that are not predefined; other constructive induction methods most often use a background knowledge of constructive operators (known functions),
- to *structured induction* as a means of automating the discovery of structure and its associated functions,
- to *decision support* as a method to induce decision models from data,
- to *knowledge discovery and data-mining* as a "divide-and-conquer" paradigm that can be used either in supervised or unsupervised mode, and that decomposes a data analysis problem to more manageable subproblems while deriving the concept structure which characterizes the data and may be a discovery by itself,
- to *intelligent data analysis* for the reasons as above, with an example for potential use in *intelligent data analysis in medicine* as given in Section 2.7.5.

5.4 Further work

The dissertation opens new avenues for both practical use of decomposition on real-world problems and for further improvements and extensions of the method.

From the methodological point of view, we propose the following extensions and improvements of decomposition method:

- The use of the background knowledge to constraint the possible set of partitions being considered, and to list the preferred functions. Inclusion of such background knowledge may also help to alleviate the decomposition's problem of sensitivity to smaller training sets, and may help to derive more transparent concept hierarchies. The background knowledge on functions does not need to be exact, but may specify only the function's type, for example monotone or symmetric functions,

- The use and appropriately handling of typed attributes, i.e., the decomposition should distinguish between nominal and ordinal attributes,

- A heuristic approach to determine the best partition: currently, the decomposition considers all possible bound sets of limited size. A heuristics to propose a candidate partition without performing an exhaustive search may result in substantial reduction of time-complexity. The method may be based on some of existing partition construction approaches that were developed for Boolean function decomposition (Perkowski 1995),

- An extension of decomposition to appropriately handle continuously-valued attributes, but does not require to discretize them in advance; the work of Demšar, Zupan, Bohanec & Bratko (1996) may be a starting point for such extension,

- In this thesis, the attribute partitions were disjoint, which limited the concept structure to be a tree. To derive more complex concept structures (in the form of acyclic graphs), the extension of the method that handles non-disjoint partitions is required. A variants of non-disjoint decompositions are presented in (Perkowski 1995) and in (Zupan & Bohanec 1996). Both show that a non-disjoint decomposition may be a straightforward extension of the disjoint algorithm.

Another focus of the further work is to enhance HINT's capabilities to assist in the interpretation of derived example sets. Any of existing machine learning tools may be used for this. A possible alternative is also to use the appropriate tools included in a DEX decision support system (Bohanec & Rajkovič 1988). However, the idea that we are most excited about is the inclusion of decomposition within an Inductive Logic Programming environment (ILP, see (Lavrač & Džeroski 1994)). A decomposition may benefit from ILP's ability to efficiently use the background knowledge, while ILP can benefit from decomposition's ability to derive smaller and less manageable example sets and from its methodological approach that may be adopted for predicate invention.

Finally and most importantly, the further work will focus on practical applications of decomposition methods. We plan to use HINT as a data analysis and knowledge discovery support tool and address real-world problems, where the concept structures are not necessary known in advance and where the domain experts are available to assist in supervised decomposition.

Bibliography

Abu-Mostafa, Y. S. (1988), *Complexity in Information Theory*, Springer-Verlag, New York.

Almuallim, H. & Dietterich, T. G. (1991), Learning with many irrelevant features, *in* 'Ninth National Conference on Artificial Intelligence', MIT Press, pp. 547–552.

Ashenhurst, R. L. (1952), The decomposition of switching functions, Technical report, Bell Laboratories BL-1(11), pages 541–602.

Biermann, A. W., Fairfield, J. & Beres, T. (1982), 'Signature table systems and learning', *IEEE Trans. Syst. Man Cybern.* **12**(5), 635–648.

Bohanec, M., Bratko, I. & Rajkovič, V. (1983), An expert system for decision making, *in* H. G. Sol, ed., 'Processes and Tools for Decision Support', North-Holland.

Bohanec, M., Cestnik, B. & Rajkovič, V. (1996), A management decision support system for allocating housing loans, *in* 'Proc. IFIP WG 8.3 Conference on Implementing Systems for Supporting Management Decisions: Concepts, Methods and Experiences', London, U.K. to appear as a book chapter.

Bohanec, M. & Rajkovič, V. (1988), Knowledge acquisition and explanation for multi-attribute decision making, *in* '8th Intl Workshop on Expert Systems and their Applications', Avignon, France, pp. 59–78.

Bohanec, M. & Rajkovič, V. (1990), 'DEX: An expert system shell for decision support', *Sistemica* **1**(1), 145–157.

Bohanec, M., Urh, B. & Rajkovič, V. (1992), 'Evaluating options by combined qualitative and quantitative methods', *Acta Psychologica* **80**, 67–89.

Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984), *Classification and regression trees*, Wadsworth International Group.

Cestnik, B. (1990), Estimating probabilities: A crucial task in machine learning, *in* 'Proc. 9th European Conference on Artificial Intelligence ECAI90', pp. 147–149.

Cestnik, B. & Bratko, I. (1991), On estimating probabilities in tree prunning, *in* 'Proc. European Workshop on Symbolic Learning EWSL-91'.

Clark, P. & Niblett, T. (1987), Induction in noisy domains, *in* I. Bratko & N. Lavrač, eds, 'Progress in Machine Learning', Sigma Press, Wilmslow, pp. 11–30.

Curtis, H. A. (1962), *A New Approach to the Design of Switching Functions*, Van Nostrand, Princeton, N.J.

Demšar, J. (1996), 'Decomposition of real functions'. B. Sc. Thesis (in Slovene).

Demšar, J., Zupan, B., Bohanec, M. & Bratko, I. (1996), Constructing intermediate concepts by decomposition of real functions, *in* 'Proc. European Conference on Machine Learning, ECML-96', Prag. to appear.

Diamantidis, N. & Giakoumakis, E. G. (1996), 'Don't care values in induction', *Artificial Intelligence in Medicine* **8**, 505–514.

Efstathiou, J. & Rajkovič, V. (1979), 'Multiattribute decisionmaking using a fuzzy heuristic approach', *IEEE Trans. on Systems, Man and Cybernetics* **9**, 326–333.

Goldman, J. (1994), Pattern theoretic knowledge discovery, Technical report, GSRP Wright Laboratories.

Halter, J. A., Carp, J. S. & Wolpaw, J. W. (1995), 'Operantly conditioned motoneuron plasticity: possible role of sodium channels', *J. Neurophysiology* **73**(2), 867–871.

Halter, J. A. & Clark, J. W. (1991), 'A distributed-parameter model of the myelinated nerve fiber', *J. Theo. Biol.* **148**, 345–382.

John, G. H., Kohavi, R. & Pfleger, K. (1994), Irrelevant features and the subset selection problem, *in* 'Machine Learning: Proceedings of the Eleventh International Conference', Morgan Kaufmann Publishers, San Francisco, CA, pp. 121–129.

Kohavi, R. & John, G. H. (1997), 'Wrappers for feature subset selection', *Aritificial Intelligence Journal* . to appear.

Kononenko, I. (1994), Estimating attributes, *in* 'Proc. European Conference on Machine Learning'.

Kononenko, I., Bratko, I. & Roškar, E. (1984), Experiments in automatic learning of medical diagnostic rules, Technical report, Jožef Stefan Institute, Ljubljana.

Kononenko, I., Šimec, E. & Robnik Šikonja, M. (1996), 'Overcoming the myopia of inductive learning algorithms with ReliefF', *Applied Intelligence Journal* . (in press).

Lavrač, N. & Džeroski, S. (1994), *Inductive logic programming: techniques and applications*, Ellis Horwood.

Luba, T. (1995), Decomposition of multiple-valued functions, *in* '25th Intl. Symposium on Multiple-Valued Logic', Bloomigton, Indiana, pp. 256–261.

Mallach, E. G. (1994), *Understanding decision support systems and expert systems*, Irwin.

Michalski, R. S. (1983), A theory and methodology of inductive learning, *in* R. Michalski, J. Carbonnel & T. Michell, eds, 'Machine Learning: An Artificial Intelligence Approach', Kaufmann, Paolo Alto, CA, pp. 83–134.

Michalski, R. S. (1986), Understanding the nature of learning: issues and research directions, *in* R. Michalski, J. Carbonnel & T. Michell, eds, 'Machine Learning: An Artificial Intelligence Approach', Kaufmann, Los Atlos, CA, pp. 3–25.

Michie, D. (1995), Problem decomposition and the learning of skills, *in* N. Lavrač & S. Wrobel, eds, 'Notes in Artificial Intelligence 912', Springer-Verlag, Springer, pp. 17–31.

Michie, D., Spiegelhalter, D. J. & Taylor, C. C., eds (1994), *Machine learning, neural and statistical classification*, Ellis Horwood.

Mladenić, D. (1995), Domain-tailored machine learning, Master's thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana.

Mozetič, I. & Hodoščeck, M. (1997), Symbolic protein data base, Technical report, Institute Jožef Stefan, IJS-DP 7505.

Murphy, P. M. & Aha, D. W. (1994), UCI Repository of machine learning databases `[http://www.ics.uci.edu/~mlearn/mlrepository.html]`. Irvine, CA: University of California, Department of Information and Computer Science.

Niblett, T. & Bratko, I. (1986), Learning decision rules in noisy domains, *in* 'Expert Systems 86', Cambridge University Press, pp. 15–18. (Proc. EWSL 1986, Brighton).

Olave, M., Rajkovič, V. & Bohanec, M. (1989), An application for admission in public school systems, *in* I. T. M. Snellen, W. B. H. J. van de Donk & J.-P. Baquiast, eds, 'Expert Systems in Public Administration', Elsevier Science Publishers (North Holland), pp. 145–160.

Perkowski, M. A. (1995), A survey of literature on function decomposition, Technical report, GSRP Wright Laboratories, Ohio OH.

Pfahringer, B. (1994), Controlling constructive induction in CiPF, *in* F. Bergadano & L. D. Raedt, eds, 'Machine Learning: ECML-94', Springer-Verlag, pp. 242–256.

Quinlan, J. R. (1979), Discovering of rules by induction from large collections of examples, *in* D. Michie, ed., 'Expert systems in micro-electronic age', Edinburgh University Press.

Quinlan, J. R. (1986*a*), Learning from noisy data, *in* 'Proc. International Machine Learning Workshop', University of Illinois at Urbana Champaign, pp. 58–64.

Quinlan, J. R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers.

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

Ragavan, H. & Rendell, L. (1993), Lookahead feature construction for learning hard concepts, *in* 'Proc. Tenth International Machine Learning Conference', Morgan Kaufman, pp. 252–259.

Rajkovič, V. & Bohanec, M. (1991), Decision support by knowledge explanation, *in* H. G. Sol & J. Vecsenyi, eds, 'Environments for supporting Decision Process', Elsevier Science Publishers B.V.

Ross, T. D., Noviskey, M. J., Axtell, M. L., Gadd, D. A. & Goldmann, J. A. (1994), Pattern theoretic feature extraction and constructive induction, Technical report, Wright Laboratory, USAF, WL/AART, WPAFB, Ohio, OH.

Ross, T. D., Noviskey, M. J., Gadd, D. A. & Goldman, J. A. (1994), Pattern theoretic feature extraction and constructive induction, *in* 'Proc. ML-COLT '94 Workshop on Constructive Induction and Change of Representation', New Brunswick, New Jersey.

Saaty, T. L. (1993), *Multicriteria decision making: The analytic hierarchy process*, RWS Publications.

Samuel, A. (1959), 'Some studies in machine learning using the game of checkers', *IBM J. Res. Develop.* **3**, 221–229.

Samuel, A. (1967), 'Some studies in machine learning using the game of checkers II: Recent progress', *IBM J. Res. Develop.* **11**, 601–617.

Shapiro, A. D. (1987), *Structured induction in expert systems*, Turing Institute Press in association with Addison-Wesley Publishing Company.

Shapiro, A. D. & Niblett, T. (1982), Automatic induction of classificiation rules for a chess endgame, *in* M. R. B. Clarke, ed., 'Advances in Computer Chess 3', Pergamon, Oxford, pp. 73–92.

Shortliffe, E. H. (1993), 'The adolescence of AI in medicine: will field come of age in the '90s?', *Artificial Intelligence in Medicine* **5**(2), 93–106.

Spackman, K., Elert, J. D. & Beck, J. R. (1993), The CIO and the medical informaticist: alliance for progress, *in* 'Proc. Annual Symposium on Computer Applications in Medical Care', pp. 525–528.

Spiegel, M. R. (1991), *Theory and Problems of Probability and Statistics*, McGraw-Hill.

Stahl, I. (1991), An overview of predicate invention techniques in ILP, *in* 'ESPRIT BRA 6020: Inductive Logic Programming'.

Thrun et al., S. B. (1991), A performance comparison of different learning algorithms, Technical report, Carnegie Mellon University CMU-CS-91-197.

Zupan, B. (1996), HINT script language: Description with examples, Technical report, J. Stefan Institute, Ljubljana, Slovenia.

Zupan, B. & Bohanec, M. (1996), Learning concept hierarchies from examples by function decomposition, Technical report, IJSDP–7455, J. Stefan Institute, Ljubljana. URL `ftp://ftp-e8.ijs.si/pub/reports/IJSDP-7455.ps`.

Zupan, B., Demšar, J. & Bohanec, M. (1997), Structure dissimilarity coefficient: definition and experiments, Technical report, Institute Jožef Stefan.

Zupan, B., Halter, J. A. & Bohanec, M. (to appear in 1997), Concept discovery by function decomposition and its application in neurophysiology, *in* N. Lavrač, E. Keravnou & B. Zupan, eds, 'Intelligent Data Analysis in Medicine and Pharmacology', Kluwer.

Appendix A

Quantitative assessment of structure similarities

During the development and testing of decomposition method, there has often been a need to compare two structures, i.e., to determine their degree of similarity. Since qualitative comparison by means of visual inspection can be very time consuming and not practical when many such comparisons are needed, we designed a quantitative and computable measure of dissimilarity of two structures. In this appendix we first propose the structure dissimilarity coefficient (SDC) and then experimentally demonstrate its appropriateness.

A.1 Structure dissimilarity coefficient

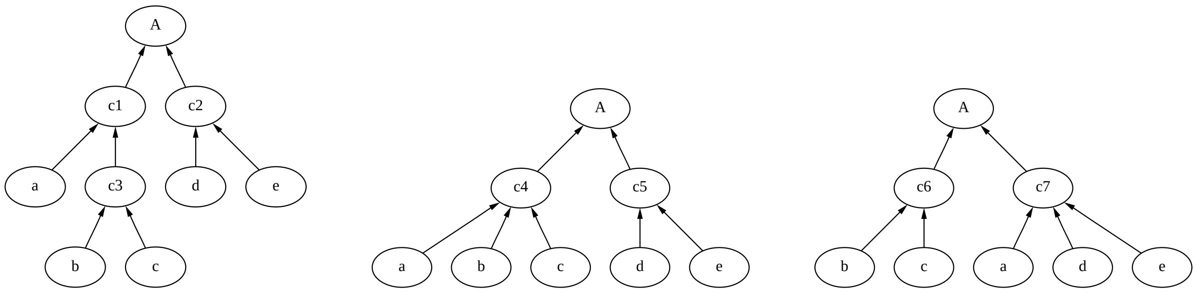
Given a concept structure S and its two leaf nodes v_i and v_j, let their distance $d(v_i, v_j)$ be equal to the number of nodes one has to traverse to come from v_i to v_j. For example, for the structure A in Figure A.1.a, $d_A(a,c) = 2$, $d_A(b,c) = 1$, and $d(a,d) = 3$.

Given two structures S_a and S_b with leaf nodes V_a and V_b, respectively, their set of common leaf nodes is $V = V_a \cap V_b$, The structure dissimilarity coefficient for S_a and S_b is then

$$\mathrm{SDC}(S_a, S_b) = \frac{1}{n(n-1)} \sum_{v_i \in V} \sum_{v_j \in V, j \neq i} |d_{S_a}(v_i, v_j) - d_{S_b}(v_i, v_j)| \tag{A.1}$$

where n is the number of leaf nodes in V. The bigger the structure dissimilarity coefficient, the less similar are two structures. Identical structures have SDC = 0.

For example, let us consider the structures A and B from Figure A.1. Their matrices with the leaf node distances are given in Figures A.2.a and A.2.b, and the absolute values of the difference of the elements of these matrices are given in Figure A.2.c. The mean value of the off-diagonal elements of the later is structure dissimilarity coefficient and is equal to 0.6. The pairwise dissimilarity coefficient for structures A, B, and C from Figure A.1 are given in Table A.1. From these (and also by the visual inspection) it may be concluded the most similar structures of the three are A and B, and the least are B and C.



(a) (b) (c)

Figure A.1: Three example structures

	a	b	c	d	e
a	0	2	2	3	3
b	2	0	1	4	4
c	2	1	0	4	4
d	3	4	4	0	1
e	3	4	4	1	0

(a) d_A

	a	b	c	d	e
a	0	1	1	3	3
b	1	0	1	3	3
c	1	1	0	3	3
d	3	3	3	0	1
e	3	3	3	1	0

(b) d_B

	a	b	c	d	e
a	0	1	1	0	0
b	1	0	0	1	1
c	1	0	0	1	1
d	0	1	1	0	0
e	0	1	1	0	0

(c) $|d_A - d_B|$

Figure A.2: Leaf node distances their absolute difference for the structures A and B

	A	B	C
A	0	0.6	1.0
B	0.6	0	1.2
C	1.0	1.2	0

Table A.1: Dissimilarity coefficients for the structures from Figure A.1

A.2 Experimental evaluation

To assess the appropriateness of the structure dissimilarity coefficient, we have designed the following experiment. Given were six different sets, each of eleven different structures and a single reference structure. Three individuals (Marko Bohanec, Janez Demšar, and Blaž Zupan) were asked to rank the structures of each set according to their similarity to the reference structure. The corresponding ranks were also derived using SDC.

The complete set of structures and ranks is presented in (Zupan, Demšar & Bohanec 1997). We here give a single set of structures (Figures A.4 to A.7) and their reference structure (Figure A.3). The reference structure is the original HOUSING model (see section 2.7.2) and the structures in the set where obtained by HINT when using various (more or less successful) heuristics. The corresponding ranks are given in Table A.2.

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11
SDC	8	6	9	4	7	5	10	11	2	3	1
MB	10	8	10	5	9	5	4	5	1	2	3
BZ	9	6	10	4	10	4	7	8	1	2	2
JD	10	7	11	4	9	5	7	6	1	2	2

Table A.2: Structures #1 to #11 in Figure A.4 to A.7 ranked by their similarity to structure in Figure A.3

To mutually compare such sets of ranks each set was first normalized, such that when m structures were given the same rank i, the new rank is $i + \frac{m-1}{2}$. Then, the correlation coefficient r_{XY} was computed for each pair of rank sets X and Y and their elements $x_i \in X$ and $y_i \in Y$ (Spiegel 1991, page 263):

$$r_{XY} = \frac{n \sum x_i y_i - (\sum x_i)(\sum y_i)}{\sqrt{[n \sum x_i^2 - (\sum x_i)^2][n \sum y_i^2 - (\sum y_i)^2]}} \tag{A.2}$$

n is the number of ranks in each set.

The correlations for ranks in Table A.2 are given in Table A.3. The lowest correlation was that between the ranks of SDC and MB, and the highest between ranks of BZ and JD. Overall, ranks obtained by SDC correlate well with the other ranks, and they are are comparable to the correlations between MB, BZ, and JD.

Table A.4 shows the averaged results over six structure sets used in this experimental study. The correlation between SDC and other sets of ranks is relatively high and again comparable with correlations between MB, BZ, and JD. We can thus conclude that SDC may be an appropriate measure to approximately assess the likeness of the two structures. However, the likeness of two structures is domain and user dependent and should be assessed by human where necessary. For this reason, we use SDC in this dissertation merely to show

r	SDC	MB	BZ	JD
SDC	1.00	0.59	0.83	0.78
MB	0.59	1.00	0.88	0.91
BZ	0.83	0.88	1.00	0.95
JD	0.78	0.91	0.95	1.00

Table A.3: Correlation coefficients for ranks from Table A.2

	SDC	MB	BZ	JD
SDC	1.000 ±0.000	0.662 ±0.226	0.828 ±0.073	0.771 ±0.074
MB	0.662 ±0.226	1.000 ±0.000	0.814 ±0.159	0.829 ±0.141
BZ	0.828 ±0.073	0.814 ±0.159	1.000 ±0.000	0.934 ±0.043
JD	0.771 ±0.074	0.829 ±0.141	0.934 ±0.043	1.000 ±0.000

Table A.4: The average and standard deviations of correlation coefficients between ranks of six sets of structures

the convergence of decomposition to a single structure and to estimate whether this was different or the same to anticipated one.

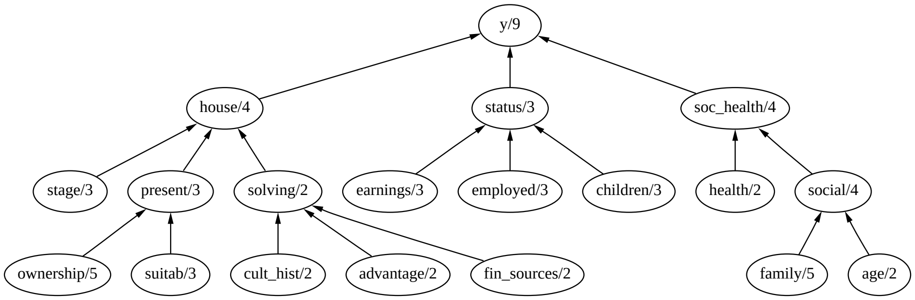


Figure A.3: Reference structure

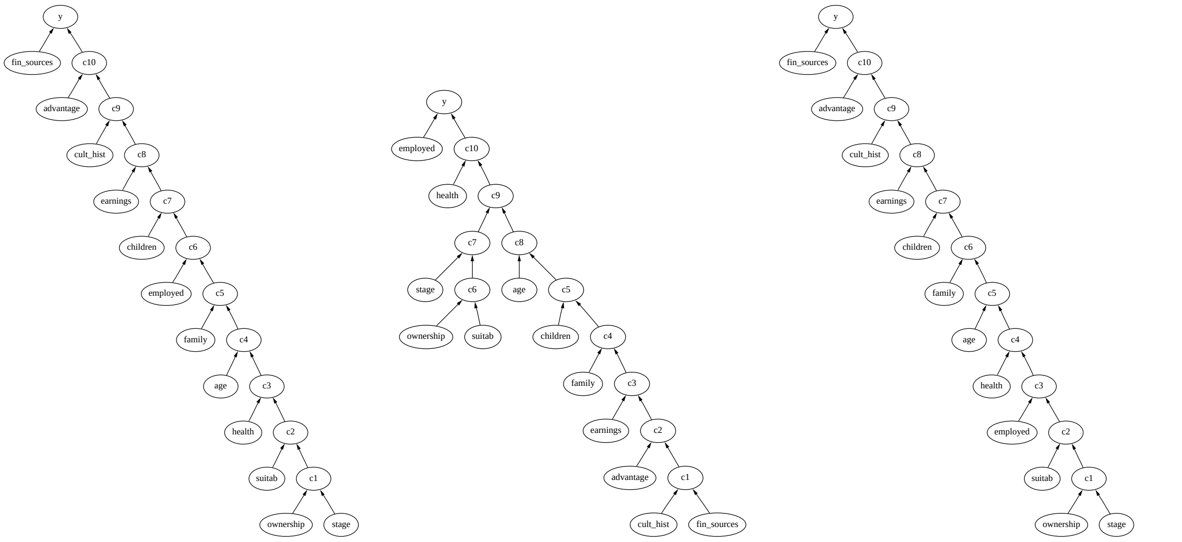


Figure A.4: **Structures #1, #2, and #3**

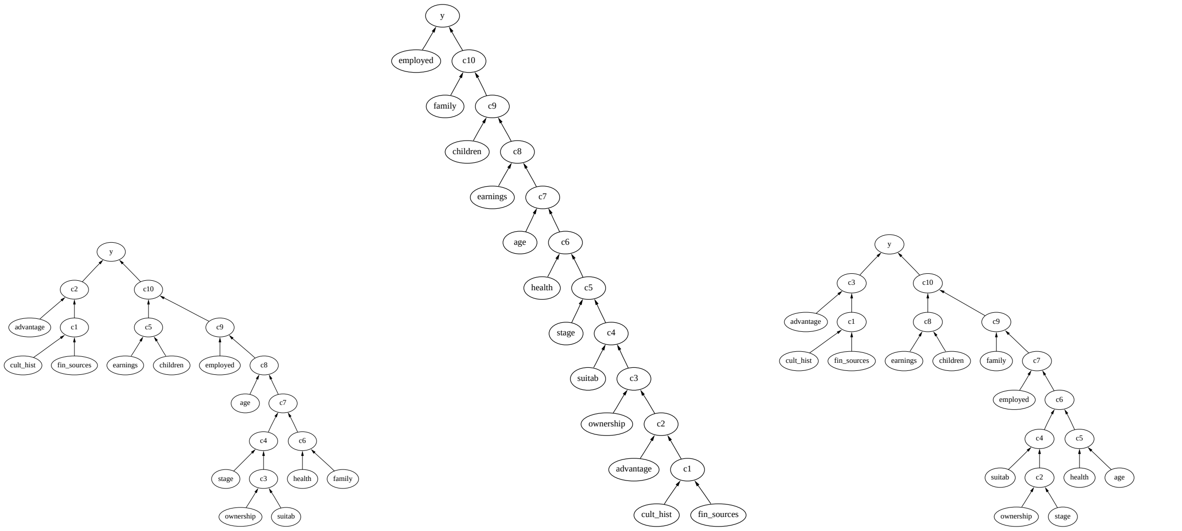


Figure A.5: **Structures #4, #5, and #6**

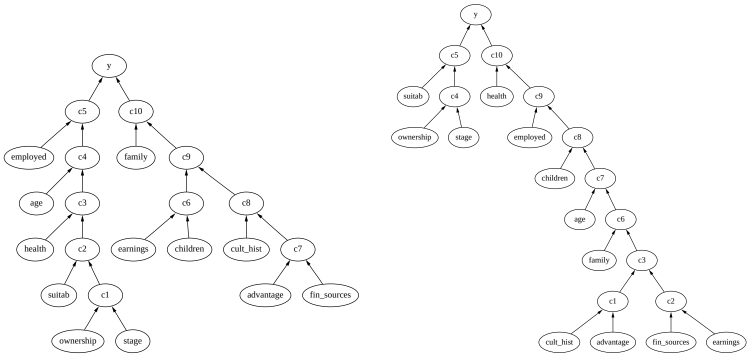


Figure A.6: **Structures #7 and #8**

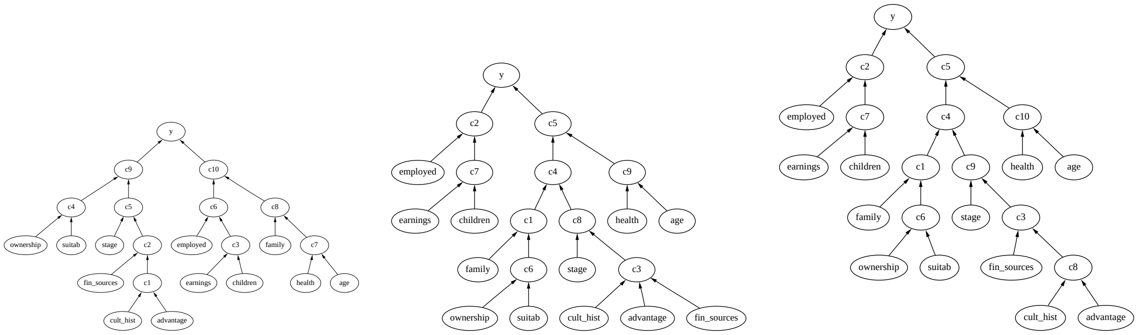


Figure A.7: **Structures #9, #10, and #11**

Appendix B

Three examples for partition selection measures

This appendix presents an evaluation of attribute partitions for three example domains LENSES, CAR, and NURSERY and their complete set of examples as described in section 2.7.2. For LENSES and NURSERY, the partitions with up to 3 attributes and for CAR with up to 4 attributes in the bound sets are presented. For each partition, the partition selection measures Ψ_{CM}, Ψ_{C}, and Ψ_{SC} are shown together with its rank according to these measures. The results are discussed in section 2.7.7.

Partition	Ψ_{CM}		Ψ_{C}		Ψ_{SC}	
`<astigm,tears>|<age,prescr>`	3	1	25.94	1	25.51	1
`<prescr,tears>|<age,astigm>`	4	5	32.77	6	31.38	5
`<prescr,astigm>|<age,tears>`	4	5	32.77	6	31.38	5
`<age,tears>|<prescr,astigm>`	4	5	41.45	10	38.04	10
`<age,astigm>|<prescr,tears>`	3	1	32.28	4	31.11	3
`<age,prescr>|<astigm,tears>`	3	1	32.28	4	31.11	3
`<tears>|<age,prescr,astigm>`	3	1	25.94	1	25.91	2
`<astigm>|<age,prescr,tears>`	4	5	32.09	3	31.90	7
`<prescr>|<age,astigm,tears>`	5	9	36.81	9	36.25	9
`<age>|<prescr,astigm,tears>`	5	9	35.44	8	33.81	8

Table B.1: Partitions for LENSES

Partition	Ψ_{CM}		Ψ_{C}		Ψ_{SC}	
<doors,persons,lug_boo,safety>|<buying,maint>	4	2	891.42	16	891.36	16
<maint,persons,lug_boo,safety>|<buying,doors>	12	26	2620.50	46	2613.39	46
<maint,doors,lug_boo,safety>|<buying,persons>	9	14	2611.60	42	2606.44	42
<maint,doors,persons,safety>|<buying,lug_boo>	12	26	3470.20	49	3456.00	49
<maint,doors,persons,lug_boo>|<buying,safety>	9	14	2611.60	42	2606.44	42
<buying,persons,lug_boo,safety>|<maint,doors>	12	26	2620.50	46	2613.39	46
<buying,doors,lug_boo,safety>|<maint,persons>	9	14	2611.60	42	2606.44	42
<buying,doors,persons,safety>|<maint,lug_boo>	12	26	3470.20	49	3456.00	49
<buying,doors,persons,lug_boo>|<maint,safety>	9	14	2611.60	42	2606.44	42
<buying,maint,lug_boo,safety>|<doors,persons>	4	2	1171.40	25	1171.22	25
<buying,maint,persons,safety>|<doors,lug_boo>	4	2	1171.40	25	1171.22	25
<buying,maint,persons,lug_boo>|<doors,safety>	7	9	2037.40	38	2035.26	38
<buying,maint,doors,safety>|<persons,lug_boo>	6	7	2317.80	41	2315.37	41
<buying,maint,doors,lug_boo>|<persons,safety>	5	6	1934.00	37	1932.76	37
<buying,maint,doors,persons>|<lug_boo,safety>	7	9	2701.00	48	2696.85	48
<persons,lug_boo,safety>|<buying,maint,doors>	10	20	730.81	8	730.80	8
<doors,lug_boo,safety>|<buying,maint,persons>	7	9	626.45	4	626.45	4
<doors,persons,safety>|<buying,maint,lug_boo>	10	20	857.66	12	857.57	12
<doors,persons,lug_boo>|<buying,maint,safety>	6	7	546.59	3	546.58	3
<maint,lug_boo,safety>|<buying,doors,persons>	13	31	1081.10	22	1080.66	21
<maint,persons,safety>|<buying,doors,lug_boo>	16	35	1299.70	28	1298.60	28
<maint,persons,lug_boo>|<buying,doors,safety>	23	48	1798.70	34	1793.68	34
<maint,doors,safety>|<buying,persons,lug_boo>	21	44	2108.70	39	2101.26	39
<maint,doors,lug_boo>|<buying,persons,safety>	17	37	1730.80	31	1727.34	31
<maint,doors,persons>|<buying,lug_boo,safety>	17	37	1730.80	31	1727.34	31
<buying,lug_boo,safety>|<maint,doors,persons>	13	31	1081.10	22	1080.66	21
<buying,persons,safety>|<maint,doors,lug_boo>	16	35	1299.70	28	1298.60	28
<buying,persons,lug_boo>|<maint,doors,safety>	24	49	1869.00	35	1863.21	35
<buying,doors,safety>|<maint,persons,lug_boo>	21	44	2108.70	39	2101.26	39
<buying,doors,lug_boo>|<maint,persons,safety>	17	37	1730.80	31	1727.34	31
<buying,doors,persons>|<maint,lug_boo,safety>	19	41	1920.20	36	1914.96	36
<buying,maint,safety>|<doors,persons,lug_boo>	3	1	342.47	2	342.47	2
<buying,maint,lug_boo>|<doors,persons,safety>	7	9	760.77	10	760.73	10
<buying,maint,persons>|<doors,lug_boo,safety>	8	13	860.70	13	860.60	13
<buying,maint,doors>|<persons,lug_boo,safety>	10	20	1347.90	30	1346.96	30
<lug_boo,safety>|<buying,maint,doors,persons>	10	20	796.02	11	796.02	11
<persons,safety>|<buying,maint,doors,lug_boo>	12	26	875.48	14	875.48	14
<persons,lug_boo>|<buying,maint,doors,safety>	15	33	979.87	18	979.87	18
<doors,safety>|<buying,maint,persons,lug_boo>	15	33	882.34	15	882.34	15
<doors,lug_boo>|<buying,maint,persons,safety>	11	25	736.91	9	736.91	9
<doors,persons>|<buying,maint,lug_boo,safety>	10	20	696.57	7	696.57	7
<maint,safety>|<buying,doors,persons,lug_boo>	9	14	654.00	5	654.00	5
<maint,lug_boo>|<buying,doors,persons,safety>	19	41	1010.90	19	1010.93	19
<maint,persons>|<buying,doors,lug_boo,safety>	17	37	948.26	17	948.25	17
<maint,doors>|<buying,persons,lug_boo,safety>	21	44	1080.90	21	1080.74	23
<buying,safety>|<maint,doors,persons,lug_boo>	9	14	654.00	5	654.00	5
<buying,lug_boo>|<maint,doors,persons,safety>	22	47	1100.20	24	1100.19	24
<buying,persons>|<maint,doors,lug_boo,safety>	20	43	1041.30	20	1041.26	20
<buying,doors>|<maint,persons,lug_boo,safety>	26	50	1251.30	27	1250.70	27
<buying,maint>|<doors,persons,lug_boo,safety>	4	2	339.42	1	339.42	1

Table B.2: Partitions for CAR

Partition	Ψ_{CM}		Ψ_{C}		Ψ_{SC}	
<form,childs,housing,finance,social,health>|<parents,has_nur>	4	4	8050.00	11	8049.92	11
<has_nur,childs,housing,finance,social,health>|<parents,form>	12	33	30106.00	50	30092.19	48
<has_nur,form,housing,finance,social,health>|<parents,childs>	9	23	22589.00	38	22583.58	38
<has_nur,form,childs,finance,social,health>|<parents,housing>	9	23	30102.00	49	30092.19	48
<has_nur,form,childs,housing,social,health>|<parents,finance>	6	7	30098.00	47	30092.19	48
<has_nur,form,childs,housing,finance,health>|<parents,social>	6	7	20075.00	27	20072.83	27
<has_nur,form,childs,housing,finance,social>|<parents,health>	7	12	23418.00	41	23413.89	41
<parents,childs,housing,finance,social,health>|<has_nur,form>	17	40	25612.00	44	25597.86	44
<parents,form,housing,finance,social,health>|<has_nur,childs>	13	35	19601.00	25	19595.75	25
<parents,form,childs,finance,social,health>|<has_nur,housing>	13	35	26103.00	45	26092.05	45
<parents,form,childs,housing,social,health>|<has_nur,finance>	9	23	27096.00	46	27088.46	46
<parents,form,childs,housing,finance,health>|<has_nur,social>	9	23	18084.00	24	18081.31	24
<parents,form,childs,housing,finance,social>|<has_nur,health>	10	30	20089.00	30	20085.05	30
<parents,has_nur,housing,finance,social,health>|<form,childs>	3	1	5665.10	5	5665.05	5
<parents,has_nur,childs,finance,social,health>|<form,housing>	7	12	17575.00	19	17573.03	19
<parents,has_nur,childs,housing,social,health>|<form,finance>	8	20	30101.00	48	30092.19	48
<parents,has_nur,childs,housing,finance,health>|<form,social>	8	20	20082.00	29	20078.74	29
<parents,has_nur,childs,housing,finance,social>|<form,health>	9	23	22589.00	38	22583.58	38
<parents,has_nur,form,finance,social,health>|<childs,housing>	7	12	17575.00	19	17573.03	19
<parents,has_nur,form,housing,social,health>|<childs,finance>	6	7	22580.00	37	22577.20	37
<parents,has_nur,form,housing,finance,health>|<childs,social>	6	7	15068.00	15	15066.43	15
<parents,has_nur,form,housing,finance,social>|<childs,health>	7	12	17575.00	19	17573.03	19
<parents,has_nur,form,childs,social,health>|<housing,finance>	3	1	15053.00	14	15052.59	14
<parents,has_nur,form,childs,finance,health>|<housing,social>	6	7	20075.00	27	20072.83	27
<parents,has_nur,form,childs,finance,social>|<housing,health>	7	12	23418.00	41	23413.89	41
<parents,has_nur,form,childs,housing,health>|<finance,social>	4	4	20069.00	26	20067.48	26
<parents,has_nur,form,childs,housing,social>|<finance,health>	5	6	25084.00	43	25080.73	43
<parents,has_nur,form,childs,housing,finance>|<social,health>	3	1	10042.00	12	10042.29	12
<childs,housing,finance,social,health>|<parents,has_nur,form>	13	35	6709.50	7	6709.31	7
<form,housing,finance,social,health>|<parents,has_nur,childs>	10	30	5192.90	3	5192.86	3
<form,childs,finance,social,health>|<parents,has_nur,housing>	10	30	6814.80	8	6814.72	8
<form,childs,housing,social,health>|<parents,has_nur,finance>	7	12	7093.40	9	7093.33	9
<form,childs,housing,finance,health>|<parents,has_nur,social>	7	12	4795.00	1	4795.03	1
<form,childs,housing,finance,social>|<parents,has_nur,health>	8	20	5469.40	4	5469.39	4
<has_nur,housing,finance,social,health>|<parents,form,childs>	9	23	5776.00	6	5775.93	6
<has_nur,childs,finance,social,health>|<parents,form,housing>	21	46	17646.00	22	17639.04	22
<has_nur,childs,housing,social,health>|<parents,form,finance>	24	48	30123.00	51	30092.18	47
<has_nur,childs,housing,finance,health>|<parents,form,social>	24	48	20147.00	33	20135.86	33
<has_nur,childs,housing,finance,social>|<parents,form,health>	25	50	20981.00	34	20967.54	34
<has_nur,form,finance,social,health>|<parents,childs,housing>	21	46	17646.00	22	17639.04	22
<has_nur,form,housing,social,health>|<parents,childs,finance>	18	41	22617.00	40	22605.57	40
<has_nur,form,housing,finance,health>|<parents,childs,social>	18	41	15144.00	17	15139.42	17
<has_nur,form,housing,finance,social>|<parents,childs,health>	19	44	15978.00	18	15972.95	18
<has_nur,form,childs,social,health>|<parents,housing,finance>	9	23	15085.00	16	15082.72	16
<has_nur,form,childs,finance,health>|<parents,housing,social>	18	41	20122.00	32	20113.00	32
<has_nur,form,childs,finance,social>|<parents,housing,health>	19	44	21234.00	35	21223.69	35
<has_nur,form,childs,housing,health>|<parents,finance,social>	12	33	20097.00	31	20091.69	31
<has_nur,form,childs,housing,social>|<parents,finance,health>	13	35	21767.00	36	21760.15	36
<has_nur,form,childs,housing,finance>|<parents,social,health>	7	12	7865.20	10	7865.02	10
<parents,housing,finance,social,health>|<has_nur,form,childs>	13	35	5153.50	2	5153.45	2
<parents,childs,finance,social,health>|<has_nur,form,housing>	28	51	14234.00	13	14227.78	13

Table B.3: Partitions for NURSERY

Appendix C

Experimental results on DEX domains

This appendix includes graphs and structures that are results of experiments with DEX domains in section 2.7 and were not included in that section. The results included in the appendix are those of DEX domains EMPLOY1, EMPLOY2, HOUSING, and ENTERPRISE. The reader is referred to section 2.7 for the discussion of the results.

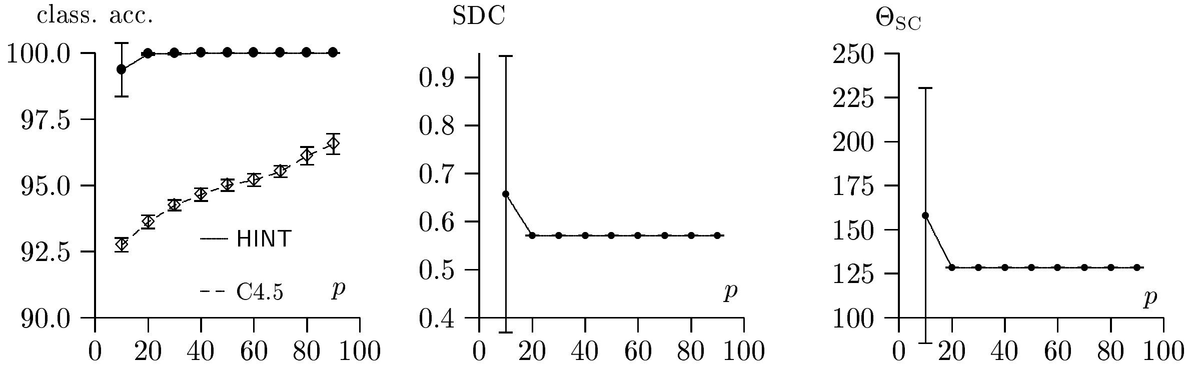


Figure C.1: Results for EMPLOY1 domain

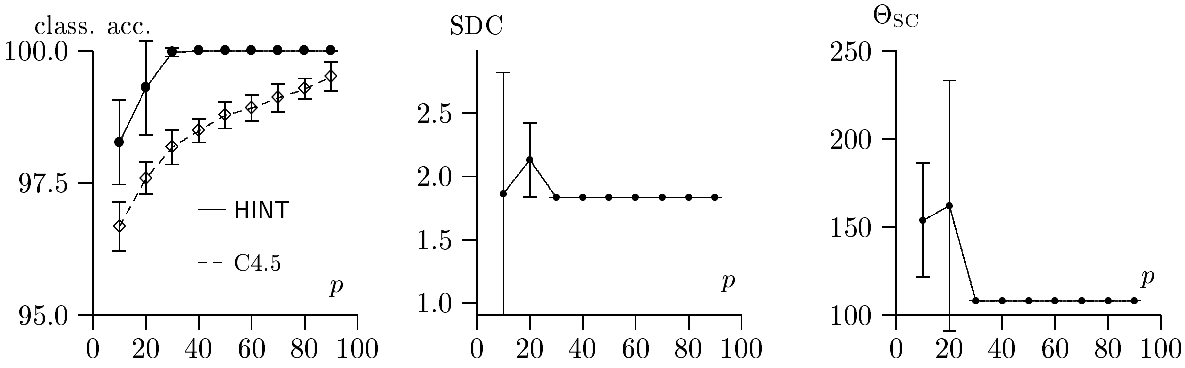


Figure C.2: Results for EMPLOY2 domain

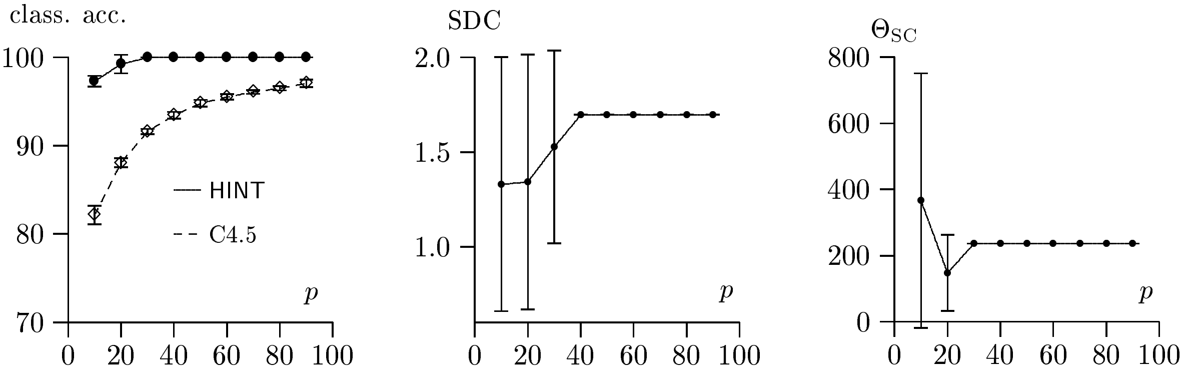


Figure C.3: Results for HOUSING domain

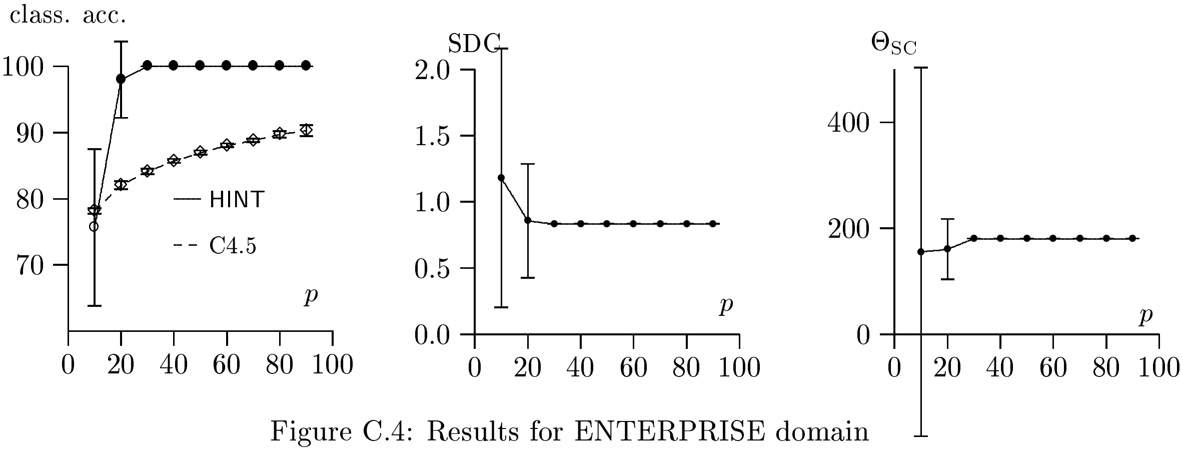


Figure C.4: Results for ENTERPRISE domain

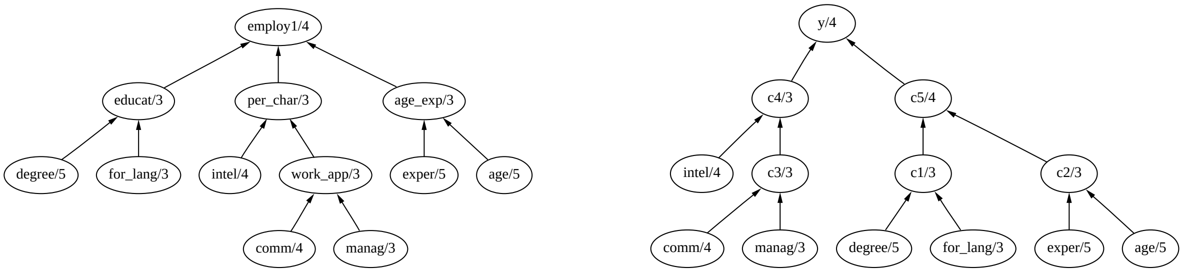


Figure C.5: The original (left) and decomposition-derived (right) structure for the EMPLOY1 domain.

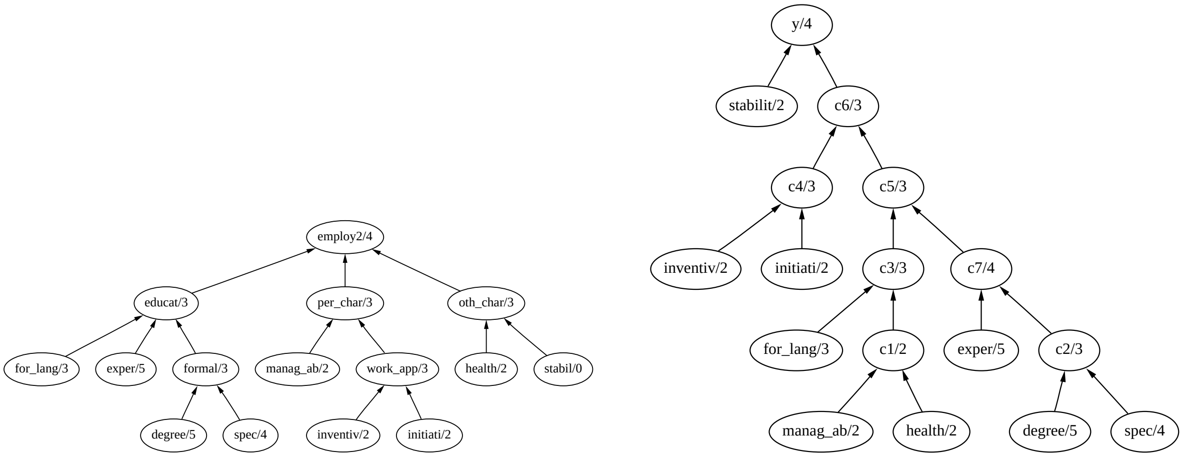


Figure C.6: The original (left) and decomposition-derived (right) structure for the EMPLOY2 domain.

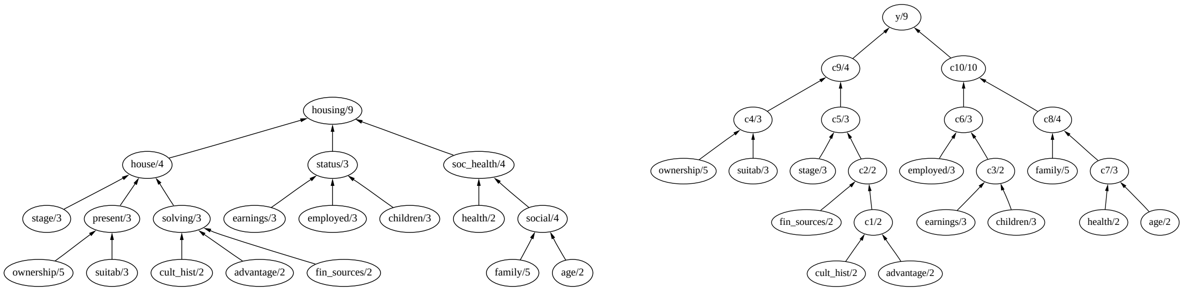


Figure C.7: The original (left) and decomposition-derived (right) structure for the HOUSING domain.

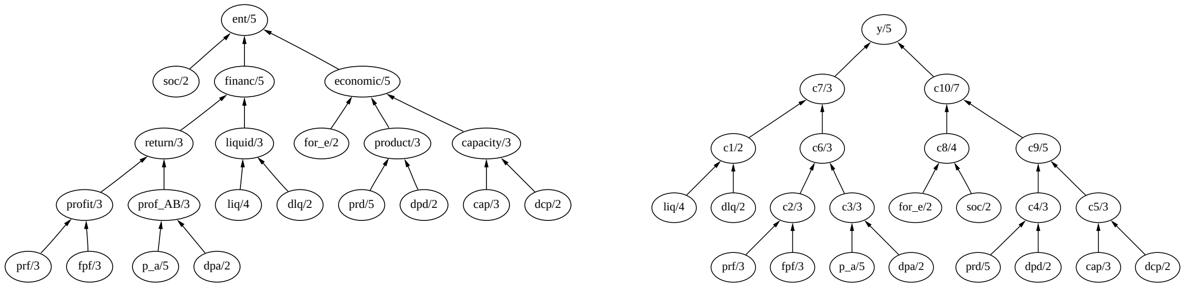


Figure C.8: The original (left) and decomposition-derived (right) structure for the ENTERPRISE domain.

Appendix D

Results on MONK domains

Table D.1 gives the performance of different learning algorithms on MONK domains as derived in reported by (Thrun et al. 1991). The domains and the corresponding experiments with HINT are described and evaluated in section 2.7.4 (MONK1 and MONK2) and in section 4.6.2.

learner	MONK1	MONK2	MONK3
HINT	**100**	99.8	**100**
AQ17-DCI	**100**	**100**	94.2
AQ17-HCI	**100**	93.1	**100**
AQ17-FCLS		92.6	97.2
AQ14-NT			**100**
AQ15-GA	**100**	86.8	**100**
Assistant Professional	**100**	81.3	**100**
mFOIL	**100**	69.2	**100**
ID5R	81.7	61.8	
IDL	97.2	66.2	
ID5R-hat	90.3	65.7	
TDIDT	75.7	66.7	
ID3	98.6	67.9	94.4
ID3, no widowing	83.2	69.1	95.6
ID5R	79.9	69.2	95.2
AQR	95.9	97.9	87.0
CN2	**100**	69.0	89.1
CLASSWEB 0.10	71.8	64.8	80.8
CLASSWEB 0.15	65.7	61.6	85.4
CLASSWEB 0.20	63.0	57.2	75.2
PRISM	86.3	72.7	90.3
ECOWEB leaf prediction	71.8	67.4	68.2
ECOWEB l.p. & information utility	82.7	71.3	68.8
Backpropagation	**100**	**100**	93.1
Backprop. with weight decay	**100**	**100**	97.2
Cascade Correlation	**100**	**100**	97.2

Table D.1: A classification accuracy of various learning algorithms on MONK domains.

Appendix E

Concept structures for seven medical domains

In this appendix we present the concept structures derived for seven medical domains from section 4.6.3. Structures were derived from a complete set of examples, and value of m used for derivation is that of HINT* from Table 4.10.

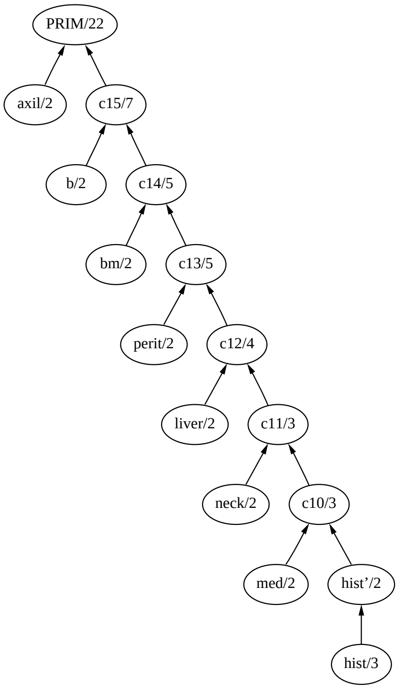


Figure E.1: Concept structure for PRIM domain

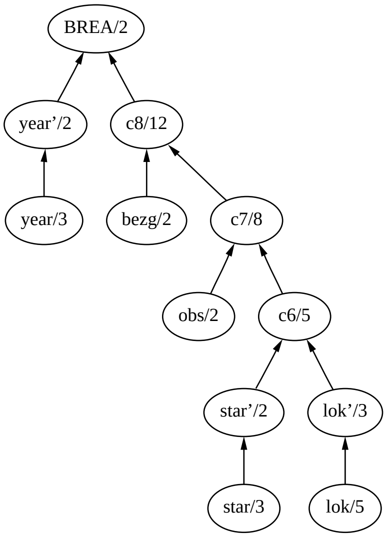


Figure E.2: Concept structure for BREA domain

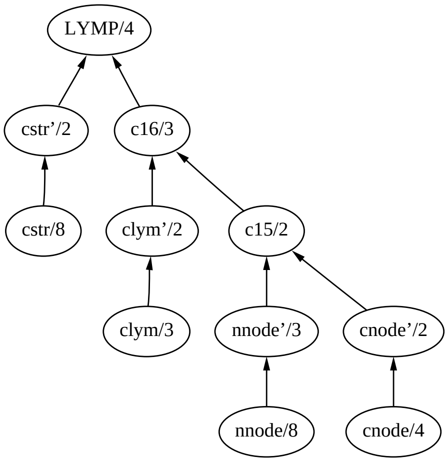


Figure E.3: Concept structure for LYMP domain

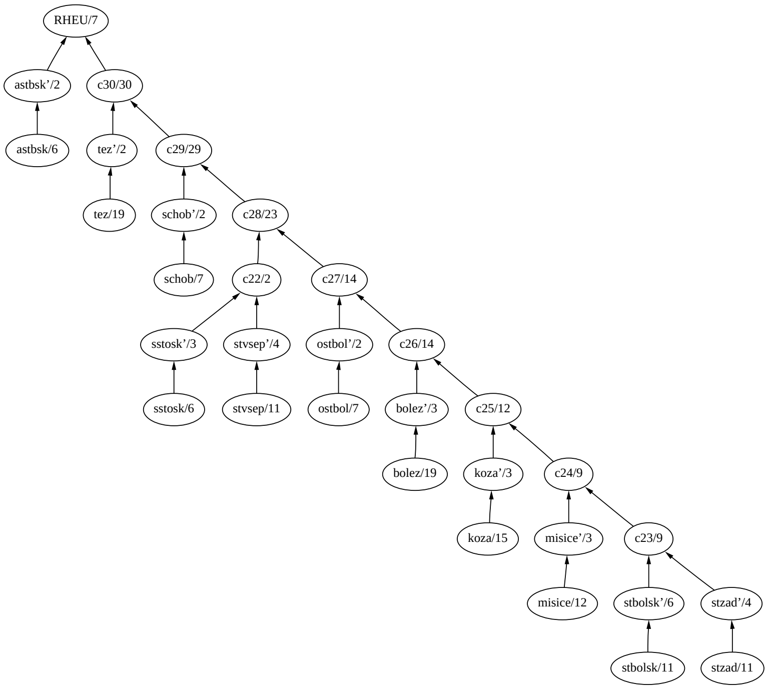


Figure E.4: Concept structure for RHEU domain

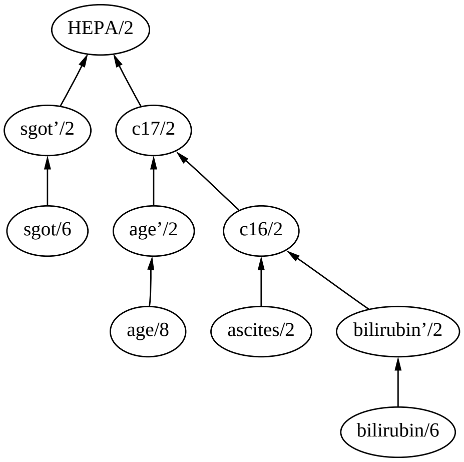


Figure E.5: Concept structure for HEPA domain

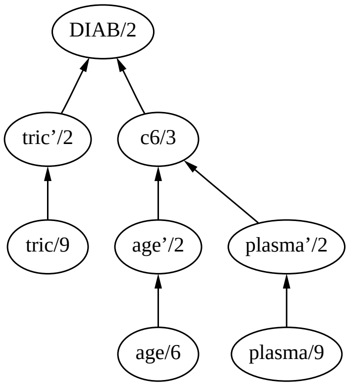


Figure E.6: Concept structure for DIAB domain

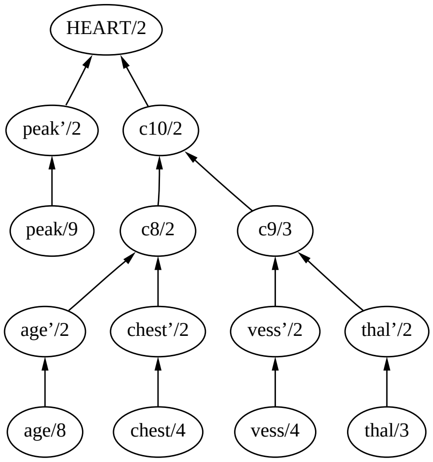


Figure E.7: Concept structure for HEART domain

Appendix F

HINT: A hierarchy induction tool

The methods proposed in this dissertation were implemented in a system called HINT (Hierarchy INduction Tool). HINT is written in the C programming language and runs in various UNIX environments, including HP/UX, SGI Iris, and SunOS. HINT is an interactive, command-driven tool that supports both supervised and unsupervised decomposition. The series of commands may be stored in scripts, which may facilitate the use of HINT in case of more complex and longer experiments.

In particular, HINT supports:

- minimal-complexity and minimal-error decomposition,
- attribute and attribute values subset selection,
- setup of decomposition parameters (b, m, etc.),
- definition of domain, which include definition of attribute names, values and example sets,
- supervised decomposition by
  - manual selection of best attribute partition,
  - definition of target concept structure,
  - manual definition of examples for function H which is used by decomposition,
- operations on a single example set and between example sets (copy, move, delete, etc.)
- classification-accuracy estimation by k-fold cross-validation, hold-out, and leave-one-out
- derivation of learning curves,
- save and restore of structures and example sets from files; C4.5 data format for example sets is supported.

The script language used by HINT is given in (Zupan 1996). We here illustrate it by an example. The domain used is the one from Section 2.2.1. The following gives the attribute names and their sets of values:

```
y depends on {x1,x2,x3}

y, x1, x2 in {lo,med,hi}
x3 in {lo, hi}
```

The set of examples is defined with

```
sel y
example set {
    lo  lo  lo    lo
    lo  lo  hi    lo
    lo  med lo    lo
    lo  med hi    med
    lo  hi  lo    lo
    lo  hi  hi    hi
    med med lo    med
    med hi  lo    med
    med hi  hi    hi
    hi  lo  lo    hi
    hi  hi  lo    hi
}
```

Within HINT, the partition selection measures for partitions with size of bound sets of 2 can be obtained by the following command (">>" is HINT's prompt):

```
>> set decomposition 2
>> test decomposition y
Decompose: y = y(x1 x2 x3 )
dfc=18 sdtic=28.529325  dtic=28.529325
Part: x3 | x1 x2 : 2x9= 5, 19,  29.84,  28.61263, ok
Part: x2 | x1 x3 : 3x6= 4, 18, 26.435, 25.041918, ok
Part: x1 | x2 x3 : 3x6= 3, 15, 21.189, 20.756516, ok

>> set decompose cm
>> decompose y
>> list struct
y/3
```

```
  x1/3
  c1/3
    x2/3
    x3/2
```

The resulting example sets can be further examined by:

```
>> select c1
>> list examples
    x2  x3  = c
    lo  lo    1
    lo  hi    1
    med lo    1
    med hi    2
    hi  lo    1
    hi  hi    3
>> select y
>> list examples
    x1  c   = y
    lo  2     med
    lo  3     hi
    lo  1     lo
    med 3     hi
    med 1     med
    hi  1     hi
```