ELSEVIER

Learning by discovering concept hierarchies

Blaž Zupan [a,b,*], Marko Bohanec [b,1], Janez Demšar [a], Ivan Bratko [a,b]

[a] *Faculty of Computer and Information Sciences, University of Ljubljana, Ljubljana, Slovenia*
[b] *Department of Intelligent Systems, J. Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia*

Abstract

We present 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. The method is inspired by the Boolean function decomposition approach to the design of switching circuits. To cope with high time complexity of finding an optimal decomposition, we propose a suboptimal heuristic algorithm. The method, implemented in program HINT (Hierarchy INduction Tool), is experimentally evaluated using a set of artificial and real-world learning problems. In particular, the evaluation addresses the generalization property of decomposition and its capability to discover meaningful hierarchies. The experiments show that HINT performs well in both respects. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Function decomposition; Machine learning; Concept hierarchies; Concept discovery; Constructive induction; Generalization

1. Introduction

To solve a complex problem, one of the most general approaches is to decompose it into smaller, less complex and more manageable subproblems. In machine learning, this principle is a foundation for structured induction [45]: instead of learning a single complex classification rule from examples, define a concept hierarchy and learn rules for each of the (sub)concepts. Shapiro [45] used structured induction for the classification of a fairly complex chess endgame and demonstrated that the complexity and comprehensibility ("brain-compatibility") of the obtained solution was superior to the unstructured one. Shapiro was

---

* Corresponding author. Email: blaz.zupan@ijs.si.
[1] Email: marko.bohanec@ijs.si.

helped by a chess master to structure his problem domain. Typically, applications of structured induction involve a manual development of the hierarchy and a manual selection and classification of examples to induce the subconcept classification rules; usually this is a tiresome process that requires an active availability of a domain expert over long periods of time. Considerable improvements in this respect may be expected from methods that automate or at least actively support the user in the problem decomposition task.

In this article we present a method for automatically developing a concept hierarchy from examples and investigate its applicability in machine learning. The method is implemented in the program called HINT (Hierarchy INduction Tool). As an illustration of the effectiveness of this approach, we present here some motivating experimental results in reconstruction of Boolean functions from examples. Consider the learning of Boolean function y of five Boolean attributes $x_1, \ldots, x_5$:

$$y = (x_1 \text{ OR } x_2) \text{ XOR } \big(x_3 \text{ OR } (x_4 \text{ XOR } x_5)\big).$$

Out of the complete 5-attribute space of 32 points, 24 points (75%) were randomly selected as examples for learning. The examples were stated as attribute-value vectors, hiding from HINT any underlying conceptual structure of the domain. In nine out of ten experiments with different randomly selected subsets of 24 examples, HINT found that the most appropriate structure of subconcepts is as shown in Fig. 1. HINT also found a definition of the intermediate functions corresponding to:

$$f_1 = \text{OR}$$
$$f_2 = \text{XOR}$$
$$f_3 = \text{OR}$$
$$f_4 = \text{XOR}.$$

This corresponds to complete reconstruction of the target concept. It should be noted that HINT does not use any predefined repertoire of intermediate functions; the definitions of the four intermediate functions above were induced solely from the learning examples.

The following results show how much the detection of a useful structure in data, like the one in Fig. 1, helps in terms of classification accuracy on new data. "New data" in our case was the remaining 25% of the points (other than those 24 examples used for
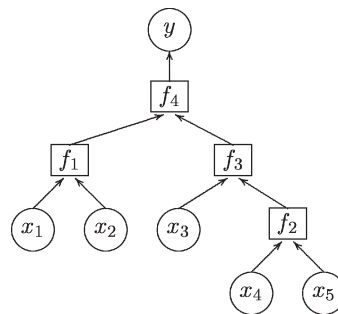

Fig. 1. Hierarchy of intermediate concepts induced by HINT for the example Boolean function.
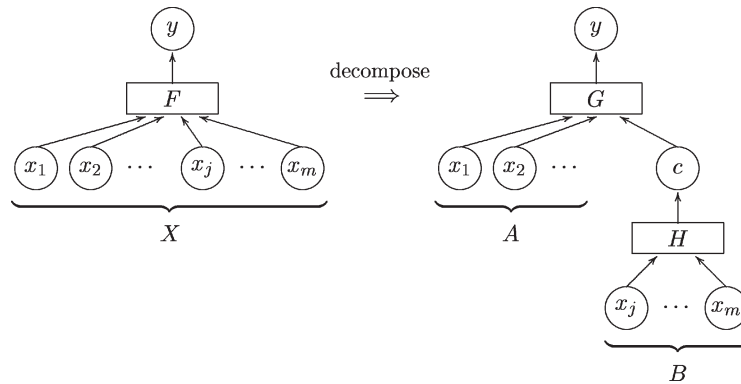
Fig. 2. Basic decomposition step.

learning). The average accuracy on new data over the 10 experiments was 97.5% with standard deviation 7.9%. For a comparison with a "flat" learner (one that does not look for concept structure in data), the program C4.5 [36] was run on the same 10 datasets. Its accuracy was 60% with standard deviation 16.5%. This result is typical of the difference in performance between HINT and flat learners in similar domains where there exist useful concept hierarchies and illustrates dramatic effects of exploiting a possible structure in the domain. A more thorough experimental evaluation of the HINT method is given later in the paper.

The HINT method is based on function decomposition, an approach originally developed for the design of switching circuits [1,10]. The goal is to decompose a function $y = F(X)$ into $y = G(A, H(B))$, where X is a set of input attributes $x_1, \ldots, x_m$, and y is the class variable (Fig. 2). F, G, and H are functions partially specified by examples, i.e., by sets of attribute-value vectors with assigned classes. A and B are subsets of input attributes such that $A \cup B = X$. The functions G and H are determined in the decomposition process and are not predefined in any way. Their joint complexity (determined by some complexity measure) should be lower than the complexity of F. Such a decomposition also discovers a new intermediate concept $c = H(B)$. Since the decomposition can be applied recursively on H and G, the result in general is a hierarchy of concepts. For each concept in the hierarchy, there is a corresponding function (such as $H(B)$) that determines the dependency of that concept on its immediate descendants in the hierarchy.

A method for discovery of a concept hierarchy from an unstructured set of examples by function decomposition can be regarded as a process that comprises the following mechanisms:

Basic function decomposition step which, given a function $y = F(X)$ partially represented by examples E_F, and a partition of the attribute set X to sets A and B, finds the corresponding functions G and H, such that $y = G(A, c)$ and $c = H(B)$. The new functions are partially defined by examples E_G and E_H.

Attribute partition selection is a process which, given a function $y = F(X)$, examines candidate partitions of X to A and B and the corresponding functions G and H. It then

selects the preferred partition of X to A and B that minimizes some complexity measure defined over G and H.

Overall function decomposition is then a recursive invocation of the above two operations on an initial example set that partially defines $y = F(X)$. In each step, the best attribute partition of X to A and B for $y = F(X)$ is selected. A function $y = F(X)$ is decomposed to $y = G(A, c)$ and $c = H(B)$ provided that G and H are overall less complex than F. If this is the case, this step is recursively repeated on newly constructed functions G and H.

Generalization usually occurs in the basic function decomposition step. When constructing example sets E_G and E_H, some points not included in E_F may be assigned a class value, thereby inductively generalizing the definition of F to points other than those explicitly stated in the examples E_F.

One of the most important problems with function decomposition is its time complexity. An algorithm for finding an optimal decomposition would consist of steps of exponential time complexity in the number of attributes. To cope with reasonably sized problems, these steps must be replaced by heuristic methods. The method presented here is "greedy" in the sense that it tries to optimize only a single step of the decomposition process; the whole discovered hierarchy, however, might not be optimal. The time complexity of splitting the attributes into sets A and B in a single decomposition step is reduced by bounding the size of B. For the task of determining the required number of values of a newly discovered concept c, which is equivalent to the graph coloring problem, we use a sub-optimal but efficient algorithm.

The proposed decomposition method is limited to nominal-valued attributes and classes. It only does disjoint partitions of attributes: $A \cap B = \emptyset$. This constrains the discovered concept hierarchies to concept trees. Furthermore, because of constraining the size of the bound set B to, say, b attributes, each internal node in the tree can have at most b descendants. In this article we do not describe the specific noise handling mechanism in HINT.

Although the function decomposition approach results in a tree, it should be noted that it is quite different from the well-known top down induction of decision trees [37]. In decision trees, nodes correspond to attributes and leaves correspond to classes. In function decomposition trees, nodes correspond to functions, and leaves correspond to attributes.

The remainder of this article first starts with the detailed description of each of the above mentioned decomposition components (Sections 2, 3, and 4). A method that uses function decomposition to detect the redundancy of attributes and to select nonredundant and most relevant attributes is given in Section 5. Section 6 experimentally evaluates the decomposition method and in particular addresses its ability to generalize and to construct meaningful concept hierarchies. The related work on the use and discovery of concept hierarchies is presented in Section 7. Section 8 gives conclusions and points to some directions for further research.

2. Basic decomposition step

Given a set of examples E_F that partially specify a function $y = F(X)$ and a partition of attributes X to subsets A and B, the *basic decomposition step* of F constructs the functions $y = G(A, c)$ and $c = H(B)$ (Fig. 2). Functions G and H are partially specified by the example sets E_G and E_H, respectively, that are derived from and are consistent with the example set E_F. Example sets 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 disjoint partition of attributes in X, such that $A \cup B = X$, $A \cap B = \emptyset$, $A \neq \emptyset$, and $B \neq \emptyset$.

The decomposition requires both the input attributes $x_i \in X$ and class variable y to be nominal-valued with domains D_{x_i} and D_y, respectively. The cardinality of these domains, denoted by $|D_{x_i}|$ and $|D_y|$, is required to be finite. The set E_F is required to be consistent: no two examples may have the same attribute values and different class values.

As proposed by Curtis [10], we will use the names *free set* and *bound set* for attribute sets A and B, respectively, and use the notation $A|B$ for the partition of attributes X into these two sets. Before the decomposition, the concept y is defined by an example set E_F and is after the decomposition defined by an example set E_G. Basic decomposition step discovers a new intermediate concept c which is defined by an example set E_H. We first present an example of such a decomposition and then define the method for the basic decomposition step.

Example 1. Consider 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. The domain of y, x_1, and x_2 is $\{\texttt{lo}, \texttt{med}, \texttt{hi}\}$ and the domain for x_3 is $\{\texttt{lo}, \texttt{hi}\}$. The function F is partially specified with a set of examples shown in Table 1.

Table 1
Set of examples that partially define
the function $y = F(x_1, x_2, x_3)$

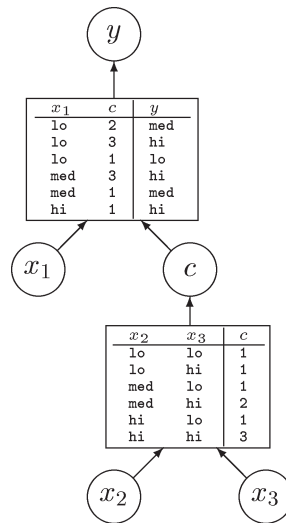| x_1	x_2	x_3	y
lo	lo	lo	lo
lo	lo	hi	lo
lo	med	lo	lo
lo	med	hi	med
lo	hi	lo	lo
lo	hi	hi	hi
med	med	lo	med
med	hi	lo	med
med	hi	hi	hi
hi	lo	lo	hi
hi	hi	lo	hi

Fig. 3. Decomposition $y = G_1(x_1, H_1(x_2, x_3))$ of the example set from Table 1.

Consider the decomposition $y = G(x_1, H(x_2, x_3))$, i.e., a decomposition with attribute partition $\langle x_1 \rangle | \langle x_2, x_3 \rangle$. This is given in Fig. 3. The following can be observed:

- The new concept hierarchy is consistent with the original example set. This can be verified by classifying each example in E_F. For instance, for attribute values $x_1 = \mathtt{med}$, $x_2 = \mathtt{med}$, and $x_3 = \mathtt{lo}$, we derive $c = 1$ and $y = \mathtt{med}$, which is indeed the same as the value of $F(\mathtt{med}, \mathtt{med}, \mathtt{lo})$.
- The example sets E_G and E_H are overall smaller than the original E_F and also easier to interpret. We can see that the new concept c corresponds to $\mathrm{MIN}(x_2, x_3)$, and E_G represents the function $\mathrm{MAX}(x_1, c)$.
- The decomposition generalizes some undefined entries of F. For example, $F(\mathtt{hi}, \mathtt{lo}, \mathtt{hi})$, which does not appear in example set E_F, is generalized to $\mathtt{hi}$ ($c = H(\mathtt{lo}, \mathtt{hi}) = 1$ and $y = G(\mathtt{hi}, 1) = \mathtt{hi}$).

2.1. The method

Let E_F be a set of examples that partially specify the function $y = F(X)$ and let $A|B$ be a partition of attributes X. The basic decomposition step derives new example sets E_G and E_H from E_F, such that they partially specify functions $y = G(A, c)$ and $c = H(B)$, respectively. Functions G and H are consistent with F, so that each example from E_F is classified equally by F and by its decomposition to G and H.

The decomposition starts with the derivation of a partition matrix.

Definition 1. 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 each row corresponding to a distinct combination of values of attributes in A, and each column corresponding to a distinct combination of

	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	2	1	3

Fig. 4. Partition matrix for the example set from Table 1 and the attribute partition $\langle x_1 \rangle | \langle x_2, x_3 \rangle$. The bottom row shows the column labels (values of c for combinations of x_2 and x_3) obtained by the coloring of incompatibility graph.

values of attributes in B. Each example $e_i \in E_F$ has its corresponding entry in $\mathcal{P}_{A|B}$ with row index $A(e_i)$ and column index $B(e_i)$. The elements of $\mathcal{P}_{A|B}$ with no corresponding examples in E_F are denoted by "-" and treated as *do not care*. A column a of $\mathcal{P}_{A|B}$ is called *nonempty* if there exists an example $e_i \in E_F$ such that $B(e_i) = a$.

An example partition matrix is given in Fig. 4. Note that the columns represent all possible combinations of the values of the attributes in B. Each column thus denotes the behavior of F when the attributes in the bound set are constant. To find a function $c = H(B)$, it is necessary to find a corresponding value (label) for each nonempty column of the partition matrix. Columns that exhibit noncontradicting behavior are called *compatible*. We will show that the necessary condition for consistency of F with G and H is that the same labels are assigned only to compatible columns.

Definition 2. 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 called *degree of compatibility* between columns a and b and is denoted by $d(a, b)$. The columns not being compatible are *incompatible* and their degree of compatibility is zero.

Theorem 1. *Example sets E_G and E_H are consistent with E_F only if E_H is derived from labeled partition matrix $\mathcal{P}_{A|B}$ so that no two incompatible columns are labeled with the same label.*

Proof. Let $A(e)$ denote the values of attributes in A for example e and $B(e)$ is defined similarly. Suppose we have a pair of incompatible columns b_i and b_j. By Definition 2, there exists a pair of examples $e_i, e_j \in F$ with

$$A(e_i) = A(e_j) = a, \qquad B(e_i) = b_i, \qquad B(e_j) = b_j, \qquad F(e_i) \neq F(e_j).$$

If decomposition $F(X) = G(A, H(B))$ is consistent, $F(e_i) = G(a, H(b_i))$ and $F(e_j) = G(a, H(b_j))$. Therefore $F(e_i) \neq F(e_j)$ leads to $G(a, H(b_i)) \neq G(a, H(b_j))$, which is clearly not possible if $H(b_i) = H(b_j)$.   □

Theorem 1 provides a necessary condition for consistency. Let us define a partition matrix to be *properly labeled* if the same label is not used for mutually incompatible

columns. Below we introduce a method that constructs E_G and E_H that are consistent with E_F and derived from any properly labeled partition matrix. The labeling preferred by decomposition is the one that introduces the fewest distinct labels, i.e., the one that defines the smallest domain for intermediate concepts c. Finding such labeling corresponds to finding the lowest number of groups of mutually compatible columns. This number is called *column multiplicity* and is denoted by $\nu(A|B)$.

Definition 3. *Column incompatibility graph* $\mathcal{I}_{A|B}$ is a graph where each nonempty column of $\mathcal{P}_{A|B}$ is represented by a vertex. Two vertices are connected if and only if the corresponding columns are incompatible.

The partition matrix column multiplicity $\nu(A|B)$ is then the number of colors needed to color the incompatibility graph $\mathcal{I}_{A|B}$. 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, and thus induces the assignment of y to every nonempty column of $\mathcal{P}_{A|B}$ such that $|D_c|$ is minimal. An example of colored incompatibility graph is given in Fig. 5.

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 of optimal coloring, a heuristic approach should be used. For proper labeling of partition matrix, an efficient heuristic algorithm called Color Influence Method was proposed by Perkowski and Uong [34] and Wan and Perkowski [50]. They empirically showed that the method generates solutions close to optimal. Essentially, Color Influence Method uses similar idea to a heuristic algorithm for graph coloring by Welsh and Powell [51], which 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 the 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 $\sum_{i=1}^{k} d(v, v_i)$ (see Definition 2 for d).
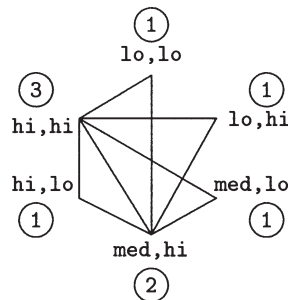

Fig. 5. Incompatibility graph for the partition $\langle x_1 \rangle | \langle x_2, x_3 \rangle$ and the partition matrix of Fig. 4. Numbers in circles represent different colors (labels) of vertices.

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. 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 of the vertex is the class of the example.

Example set E_G is derived as follows. For any value of c and combination of values a of attributes in A, $y = G(a, c)$ is determined by looking for an example e_i in row $a = A(e_i)$ and in any column labeled with the value of c. If such an example exists, an example with attribute values $A(e_i)$ and c and class $y = F(e_i)$ 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 column b is found. For example, an undefined element $\mathcal{P}_{A|B}[$`<hi>`$,$`<lo,hi>`$]$ of the first partition matrix in Fig. 4 was generalized to `hi` because the column `<lo,hi>` had the same label as columns `<lo,lo>` and `<hi,lo>`.

2.2. Some properties of the basic decomposition step

Here we give some properties of the basic decomposition step. We omit the proofs which rather obviously follow from the method of constructing example sets E_G and E_H.

Theorem 2. *The example sets E_G and E_H obtained by the basic decomposition step are consistent with E_F, i.e., every example in E_F is correctly classified using the functions H and G.*

Theorem 3. *The partition matrix column multiplicity $v(A|B)$ obtained by optimal coloring of $\mathcal{I}_{A|B}$ is the lowest number of values for c to guarantee the consistency of example sets E_G and E_H with respect to example set E_F.*

Theorem 4. *Let N_G, N_H, and N_F be the numbers of examples in E_G, E_H, E_F, respectively. Decomposition derives E_G and E_H from E_F using the attribute partition $A|B$. Then, E_G and E_H use fewer or the same number of attributes as E_F ($|B| < |X|$ and $|A| + 1 \leqslant |X|$, where X is the initial attribute set) and include fewer or the same number of examples ($N_G \leqslant N_F$ and $N_H \leqslant N_F$).*

2.3. Efficient derivation of the incompatibility graph

Most often, machine learning algorithms deal with sparse datasets. 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. According to Definition 3, an edge (v_i, v_j) of incompatibility graph $\mathcal{I}_{A|B}$ connects two vertices v_i and v_j if there exist examples $e_k, e_l \in E_F$ with $F(e_k) \neq F(e_l)$ such that $A(e_k) = A(e_l)$, $i = B(e_k)$, and $j = B(e_l)$. We propose an algorithm that efficiently implements the construction of $\mathcal{I}_{A|B}$ using this definition. The algorithm first sorts the examples E_F based on the values of attributes in A and values of y. Sorting uses a combination of *radix* and *counting sort* [9], and thus runs $|A| + 1$ intermediate sorts of time complexity $|E_F|$. After sorting, the exam-

Table 2
Examples from Table 1 sorted by x_1 and y; double lines delimit the groups, single lines the subgroups

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

ples with the same $A(e_i)$ constitute consecutive groups that correspond to rows in partition matrix $\mathcal{P}_{A|B}$. Within each group, examples with the same value of y constitute consecutive subgroups. Each pair of examples from the same group and different subgroups has 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). Each subgroup then defines a single example of E_G with the values of attributes in A and a value of c, and a value of y which is the same and given by any example in the subgroup.

Example 2. For the example set from Table 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 given in Table 2. 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).

3. Partition selection measures

The basic decomposition step assumes that a partition of the attributes to free and bound sets is given. However, for each function F there can be many possible partitions, each one yielding a different intermediate concept c and a different pair of functions G and H.

Among these partitions, we prefer those that lead to a simple concept c and functions G and H of low complexity.

Example 3. Consider again the example set from Table 1. Its decomposition that uses the attribute partition $\langle x_1 \rangle | \langle x_2, x_3 \rangle$ is shown in Fig. 3. There are two other nontrivial attribute partitions $\langle x_2 \rangle | \langle x_1, x_3 \rangle$ and $\langle x_3 \rangle | \langle x_1, x_2 \rangle$ whose decompositions are given in Fig. 6. Note that, compared to these two decompositions, the first decomposition yields less complex and more comprehensible datasets. While we could interpret the datasets of the first decomposition (concepts MIN and MAX), the interpretation of concepts for other two decompositions is harder. Note also that these two decompositions both discover intermediate concepts that use more values than the one in the first decomposition. Among the three attribute partitions it is therefore best 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 $c_1 = H_1(x_2, x_3)$.

We introduce a *partition selection measure* $\psi(A|B)$ that estimates the complexity of decomposition of F to G and H using the attribute partition $A|B$. The best partition is the one that minimizes $\psi(A|B)$. This section introduces three partition selection measures, one based on column multiplicity of partition matrix and the remaining two based on the amount of information needed to represent the functions G and H. The three measures are experimentally compared in Section 6.4.
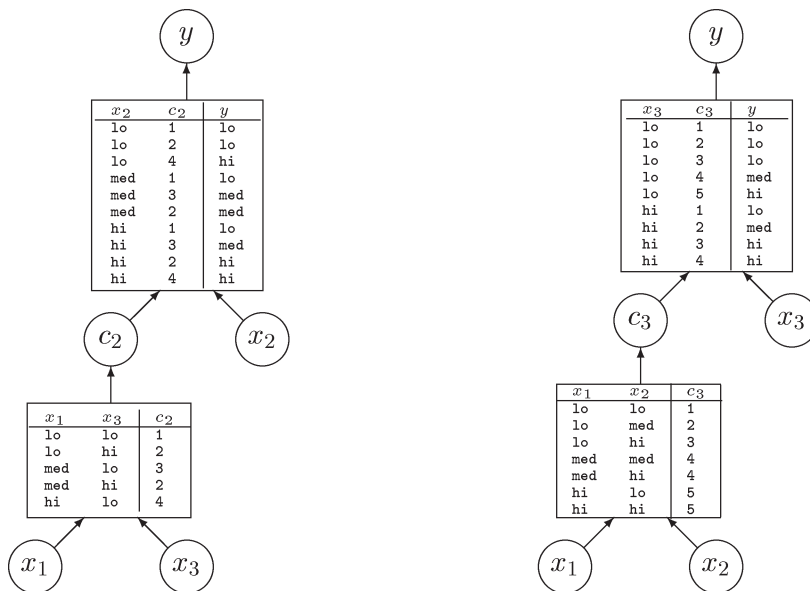

Fig. 6. The decompositions $y = G_2(x_2, H_2(x_1, x_3))$ and $y = G_3(x_3, H_3(x_1, x_2))$ of the example set from Table 1.

3.1. Column multiplicity

Our simplest partition selection measure, denoted ψ_v, is defined as the number of values required for the new feature c. That is, when decomposing F, a set of candidate partitions is examined and one that yields c with the smallest set of possible values is selected for decomposition. The number of required values for c is equal to column multiplicity of partition matrix $\mathcal{P}_{A|B}$, so:

$$\psi_v(A|B) = v(A|B). \tag{1}$$

Note that $v(A|B)$ also indirectly affects the size of instance space that defines G. The smaller the $v(A|B)$, the less complex the function G.

The idea for this measure came from practical experience with decision support system DEX [6]. There, a hierarchical system of decision tables is constructed manually. In more than 50 real-life applications it was observed that in order to alleviate the construction and interpretation, the designers consistently developed functions that define concepts with a small number of values. In most cases, they used intermediate concepts with 2 to 5 values.

Example 4. For the partitions in Fig. 4, ψ_v is 3, 4, and 5, respectively. As expected, the best partition according to ψ_v is $\langle x_1 \rangle | \langle x_2, x_3 \rangle$.

3.2. Information-based measures

The following two partition selection measures are based on the complexity of functions. Let $I(F)$ denote the minimal number of bits needed to encode some function F. Then, the best partition is the one that minimizes the overall complexity of newly discovered functions H and G, i.e., the partition with minimal $I(H) + I(G)$. The following two measures estimate I differently: the first one takes into account only the attribute-class space size of the functions, while the second one additionally considers specific constraints imposed by the decomposition over the functions.

Let us first consider some function of type $y = F(X)$. The instance space for this function is of size

$$|D_X| = \prod_{x \in X} |D_x|. \tag{2}$$

Each instance is labeled with a class value from $|D_y|$. Therefore, the number of all possible functions in the attribute-class space is

$$N_1(X, y) = |D_y|^{|D_X|}. \tag{3}$$

Assuming the uniform distribution of functions, the number of bits to encode a function F is then

$$I_1(F) = \log_2 N_1(X, y) = |D_X| \; \log_2 |D_y|. \tag{4}$$

Based on I_1 we can define our first information-based measure ψ_s, which is equal to the sum of bits to encode the functions G and H

$$\psi_s(A|B) = I_1(G) + I_1(H). \tag{5}$$

A similar measure for Boolean functions was proposed by Ross et al. [40] and called DFC (Decomposed Function Cardinality). They have used it to guide the decomposition of Boolean functions and to estimate the overall complexity of derived functions. DFC of a single function is equal to $|D_X|$. Similarly to our definition of ψ_s, the DFC of a system of functions is the sum of their DFCs.

Example 5. For the attribute partitions in Fig. 4, the ψ_s-based partition selection measures are: $\psi_s(\langle x_1 \rangle | \langle x_2, x_3 \rangle) = 23.8$ bits, $\psi_s(\langle x_2 \rangle | \langle x_1, x_3 \rangle) = 31.0$ bits, and $\psi_s(\langle x_3 \rangle | \langle x_1, x_2 \rangle) = 36.7$ bits. The preferred partition is again $\langle x_1 \rangle | \langle x_2, x_3 \rangle$.

When $y = F(X)$ is decomposed to $y = G(A, c)$ and $c = H(B)$, the function H is actually constrained so that:
- The intermediate concept c uses *exactly* $|D_c|$ values. Valid functions H include only those that, among the examples that define them, use at least one for each of the values in D_c.
- The labels for c are *abstract* in the sense that they are used for internal bookkeeping only and may be reordered or renamed. A specific function H therefore represents $|D_c|!$ equivalent functions.

For the first constraint, the number of functions that define the concept y with cardinality $|D_y|$ using the set of attributes X is:

$$\mathrm{N}_2(X, y) = S(|D_X|, |D_y|), \tag{6}$$

where $S(n, r)$ is the number of distinct classifications of n objects to r classes (Stirling number of the second kind multiplied by $r!$) defined as:

$$S(n, r) = \sum_{i=0}^{r} (-1)^{r-i} \binom{r}{i} i^n. \tag{7}$$

The formula is derived using the principle of inclusions and exclusions, which takes the total number of distributions of n objects to r classes, r^n, and subtracts the number of distributions with one class empty, $\binom{r}{1}(r-1)^n$. The distributions which have not only one but two classes empty were counted twice; $\binom{r}{2}(r-2)^n$ is added to correct this. Now, the distributions with three empty classes were subtracted three times as singles and then added three times again as pairs; $\binom{r}{3}(r-3)^n$ must be subtracted as a correction. Continuing this way, we derive the above formula. For a detailed discussion, see [15].

The number of valid functions H is therefore $\mathrm{N}_2(B, c)/|D_c|!$ and the number of bits to encode a specific function H assuming the uniform distribution of functions is:

$$\mathrm{I}_2(H) = \log_2 \frac{\mathrm{N}_2(B, c)}{|D_c|!}. \tag{8}$$

For function G, the second of the above two constraints does not apply: outputs of G are uniquely determined from examples that define F and the developed function H. We may assume that F uses all the values in D_y, and so does the resulting function G. Thus, the first constraint applies to G as well, and the number of bits to encode a specific function G is:

$$\mathrm{I}_2'(G) = \log_2 \mathrm{N}_2(A \cup \{c\}, y). \tag{9}$$

The partition selection measure ψ_c based on the above definition is therefore:

$$\psi_c(A|B) = I_2'(G) + I_2(H) = \log_2 N_2\big(A \cup \{c\}, y\big) + \log_2 \frac{N_2(B, c)}{|D_c|!}. \tag{10}$$

This measure will, for any attribute partition, always be lower than or equal to ψ_s.

Our development of ψ_c was motivated by the work of Biermann et al. [2]. They found an exact formula for counting the number of functions that can be represented by a given concept hierarchy.

In addition to the constraints on H and G mentioned above, Biermann et al. considered constraints related to the so-called reducibility of functions: if $c = G(B)$ is decomposition-constructed, then any function H should be disregarded that makes any value of c redundant (see Definition 5 for redundancy of values). We did not incorporate these constraints into ψ_c since they would considerably complicate the computation and make it practically infeasible. Namely, the computation of Biermann et al.'s formula is exponential in the number of attributes and their domain sizes. Furthermore, it would require taking into account not only the properties of the function F and its attribute-class space, but also the properties of the complete concept hierarchy developed so far.

Example 6. For the attribute partitions in Fig. 4, the ψ_c-based partition selection measures are: $\psi_c(\langle x_1 \rangle | \langle x_2, x_3 \rangle) = 20.6$ bits, $\psi_c(\langle x_2 \rangle | \langle x_1, x_3 \rangle) = 25.0$ bits, and $\psi_c(\langle x_3 \rangle | \langle x_1, x_2 \rangle) = 28.5$ bits. Again, the preferred partition is $\langle x_1 \rangle | \langle x_2, x_3 \rangle$.

4. Overall function decomposition

The decomposition aims to discover a hierarchy of concepts defined by example sets that are overall less complex than the initial one. Since an exhaustive search is prohibitively complex, the decomposition uses a suboptimal greedy algorithm.

4.1. Decomposition algorithm

The overall decomposition algorithm (Algorithm 1) applies the basic decomposition step over the evolving example sets in a concept hierarchy, starting with a single nonstructured example set. The algorithm keeps a list $\mathcal{E}$ of constructed example sets, which initially contains a complete training set E_{F_0}.

In each step (the **while** loop) the algorithm arbitrarily selects an example set E_{F_i} from $\mathcal{E}$ which belongs to a single node in the evolving concept hierarchy. The algorithm tries to decompose E_{F_i} by evaluating all candidate partitions of its attributes. To limit the complexity, the candidate partitions are those with the cardinality of the bound set less than or equal to a user defined parameter b. For all such partitions, a partition selection measure is determined and the best partition $A_{best}|B_{best}$ is selected accordingly. Next, the decomposition determines if the best partition would result in two new example sets of lower complexity than the example set E_{F_i} being decomposed. If this is the case, E_{F_i} is called *decomposable* and is replaced by two new example sets. This decomposition step is then repeated until a concept structure is found that includes only nondecomposable example sets.

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 $c_i = F_i(x_1, \ldots, x_m), i < j$
    $\mathcal{E} \leftarrow \mathcal{E} \setminus \{E_{F_i}\}$
    $A_{best}|B_{best} = \underset{A|B}{\arg\min}\, \psi(A|B),$
        where $A|B$ runs over all possible partitions of $X = \langle x_1, \ldots, x_m \rangle$
        such that $A \cup B = X$, $A \cap B = \emptyset$, and $|B| \leqslant b$
    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 1. Decomposition algorithm

To decompose a function further or not is determined by the *decomposability criterion*. Suppose that we are decomposing a function F and its best attribute partition would yield functions G and H. Then, either one of the two information-based complexity measures defined in Section 3.2 can be used to determine the number of bits $I(F)$, $I(G)$, and $I(H)$ to encode the three functions, where the method to compute $I(F)$ is the same as for $I(G)$. The decomposability criterion is then $I(G) + I(H) < I(F)$.

Note that because ψ_v is not based on function complexity, it can not be similarly used as decomposability criterion. Therefore, when using ψ_v as a partition selection measure, either of the two information-based complexity measures is used to determine decomposability.

Example 7. Table 3 compares the application of decomposability criteria ψ_s and ψ_c on the example set from Table 1. Neither criterion allows the decomposition with $\langle x_3 \rangle | \langle x_1, x_2 \rangle$. Of the other two partitions, the partition $\langle x_1 \rangle | \langle x_2, x_3 \rangle$ is the best partition according to both partition selection measures.

Table 3
Complexity measures and decomposability for partitions of Fig. 4

	$\langle x_1 \rangle | \langle x_2, x_3 \rangle$			$\langle x_2 \rangle | \langle x_1, x_3 \rangle$			$\langle x_3 \rangle | \langle x_1, x_2 \rangle$		
	$I(G) + I(H)$	$I(F)$		$I(G) + I(H)$	$I(F)$		$I(G) + I(H)$	$I(F)$	
ψ_s	23.7	28.5	✓	31.0	28.5	✗	36.7	28.5	✗
ψ_c	20.7	28.5	✓	25.0	28.5	✓	28.5	28.5	✗

4.2. 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 E_F and deriving and coloring the incompatibility graph is $O(Nn_c) + O(Nk) + O(k^2)$, where N is the number of examples in E_F, k is the number of vertices in $\mathcal{I}_{A|B}$, and n_c is the maximum cardinality of attribute domains and domains of constructed intermediate concepts. For any bound set B, the upper bound of k is

$$k_{\max} = n_c^b, \tag{11}$$

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

$$\sum_{j=2}^{b} \binom{n}{j} \leqslant \sum_{j=2}^{b} n^j \leqslant (b-1)n^b = O(n^b). \tag{12}$$

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

$$O\left(\left(Nn_c + Nk_{\max} + k_{\max}^2 \right) \sum_{m=3}^{n} m^b \right) = O\left(n^{b+1} \left(Nn_c + Nk_{\max} + k_{\max}^2 \right) \right). \tag{13}$$

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

5. Attribute redundancy and decomposition-based attribute subset selection

When applying a basic decomposition step to a function $y = F(X)$ using some attribute partition $A|B$, an interesting situation occurs when the resulting function $c = H(B)$ is constant, i.e., when $|D_c| = 1$. For such a decomposition, the intermediate concept c can be removed as it does not influence the value of y. Thus, the attributes in B are *redundant*, and $y = F(X)$ can be consistently represented with $y = G(A)$, which is a decomposition-constructed function $G(A, c)$ with c removed.

Such decomposition-discovered redundancy may well indicate for a true attribute redundancy. However, especially with the example sets that sparsely cover the attribute space, this redundancy may also be due to undersampling: the defined entries in partition matrix are sparse and do not provide the evidence for incompatibility of any two columns. In such cases, several bound sets yielding intermediate concepts with $|D_c| = 1$ may exist, thus misleading the partition selection measures to prefer partitions with redundant bound sets instead of those that include attributes that really define some underlying concept.

To overcome this problem, we propose an example set preprocessing by means of attribute subset selection which removes the redundant attributes. The resulting example
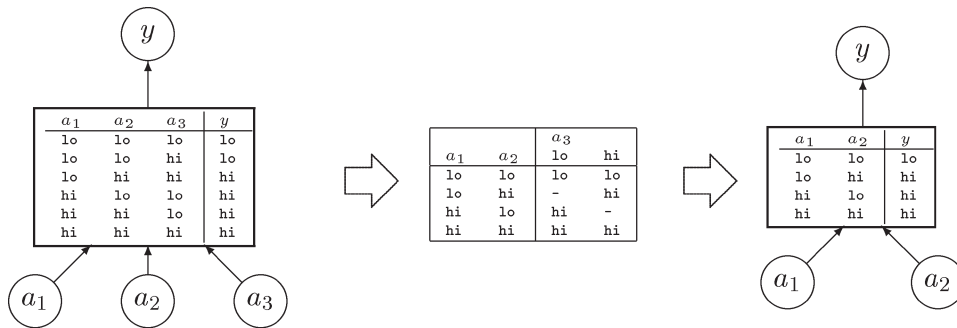
Fig. 7. Discovery of redundant attribute a_3 and its removal from the training set.

set is then further used for decomposition. Attributes are removed one at the time, their redundancy being determined by the following definition.

Definition 4. An attribute a_j is *redundant* for a function $y = F(X) = F(a_1, \ldots, a_n)$, if for the partition of attributes X to $A|B$ such that $B = \langle a_j \rangle$ and $A = X \setminus \langle a_j \rangle$, the column multiplicity of partition matrix $\mathcal{P}_{A|B}$ is $\nu(A|B) = 1$.

Fig. 7 provides an example of the discovery and removal of a redundant attribute. The original dataset (left) was examined for redundancy of attribute a_3 by constructing a corresponding partition matrix (center). Since the two columns for $a_3 = \mathtt{lo}$ and $a_3 = \mathtt{hi}$ are compatible, a_3 can be reduced to a constant and can thus be removed (right).

Besides attribute redundancy, we also define redundancy in attribute values.

Definition 5. An attribute a_j has redundant values if for a function $y = F(X) = F(a_1, \ldots, a_n)$ and for a partition of X to $A|B$ such that $B = \langle a_j \rangle$ and $A = X \setminus \langle a_j \rangle$, the column multiplicity of partition matrix $\mathcal{P}_{A|B}$ is lower than $|D_j|$.

By the above definition, such attribute can be replaced by an attribute $a'_j = H(a_j)$, and a function $y = G(A, a'_j)$ may be used instead of $y = F(X)$. An example of such attribute replacement is given in Fig. 8. Since an example set E_H may itself be of interest and point out some regularities in data, it is included in the representation and a'_j is treated as an intermediate concept.

It should be noted that not all redundancies according to Definitions 4 and 5 may simply be removed. For example, after removing a redundant attribute from X, other redundant attributes may become non redundant. Therefore, redundant attributes are processed one at a time in the reverse order of their relevance. Given an initial example set, redundancy and relevance of the attributes is determined. Next, the least relevant attribute is selected, and its redundancy removed by either removing the attribute or replacing it by a corresponding attribute with fewer values, whichever appropriate. The process is then repeated on the new example set, until no more redundancies are found.

To estimate the relevance of attributes, we use the ReliefF algorithm as proposed by Kononenko [18]. This particular algorithm is used due to its advantages over the other
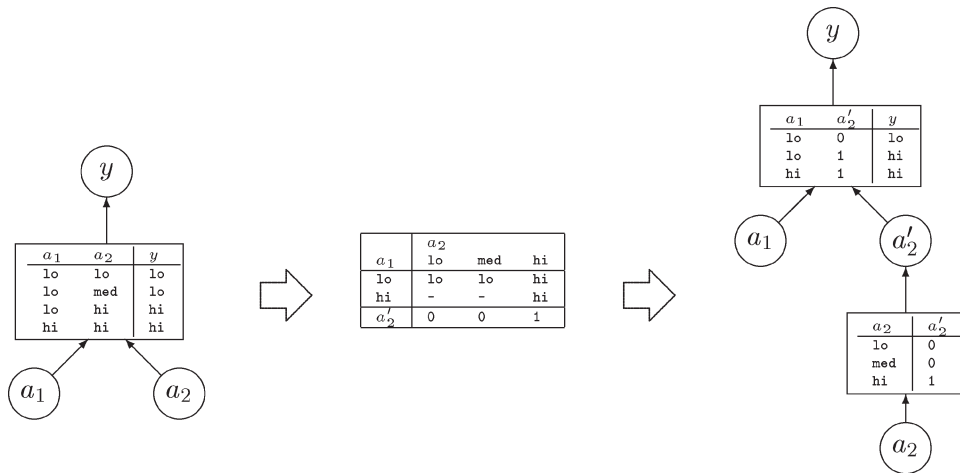
Fig. 8. Replacement of attribute a_2 with a corresponding attribute a_2'. Note that $|a_2'| < |a_2|$.

impurity functions usually used in inductive learning algorithms [19]. ReliefF estimates the attributes according to how well they distinguish among the instances that are close to each other. The relevance of attribute a is then

$$W(a) = P(\text{different value of } a \mid k\text{-nearest instances from different class})$$
$$- P(\text{different value of } a \mid k\text{-nearest instances from same class}). \qquad (14)$$

We use the version of ReliefF which determines the attribute's relevance based on at most 200 randomly selected examples from the training set, and which for every example examines $k = 5$ nearest instances of the same class and k nearest instances of different different class. The distance between two examples is measured as the number of attributes in which these two examples differ. Probabilities are estimated by relative frequencies. For further details of the ReliefF algorithm see [18].

6. Experimental evaluation

The decomposition methods described in this article were implemented in the program HINT (Hierarchy INduction Tool). This section attempts to evaluate HINT and the underlying methods from the aspects of generalization and discovery of concept hierarchies. For this purpose, several datasets are used for which either the underlying concepts hierarchy is known or anticipated, or unknown. The latter datasets are considered only for the evaluation of generalization.

The datasets on which experiments were performed are introduced first. This is followed by the assessment of generalization and evaluation of HINT's capabilities to discover meaningful concept hierarchies. Finally, we study how different partition selection measures influence HINT's behavior.

6.1. Datasets

Three types of datasets were used: (1) artificial datasets with known underlying concepts, (2) real-life datasets taken mostly from UCI Repository of machine learning databases [31], and (3) datasets derived from hierarchical decision support models developed with the DEX methodology. To distinguish among them, we will refer to these datasets as *artificial*, *repository*, and *DEX datasets*. Their basic characteristics are given in Table 4.

The artificial datasets are PALINDROME, PARITY, MONK1, and MONK2. PALIN-DROME is a palindrome function over six 3-valued attributes. PARITY is defined as XOR over five binary attributes; the other five attributes in this domain are irrelevant. MONK1 and MONK2 are well known six-attribute binary classification problems [31,49] that use 2-to 4-valued attributes. MONK1 has an underlying concept $x_1 = x_2$ OR $x_5 = 1$, and MONK2 the concept $x = 1$ for exactly two choices of attributes $x \in \{x_1, \ldots, x_6\}$.

VOTE is a real-world database as given with Quinlan's C4.5 package [36] that includes example votes by US House of Representatives Congressmen. The votes are simplified to yes, no, or unknown. PROMOTERS, SPLICE and MUSHROOM were all obtained from the UCI Repository [31]. PROMOTERS describes E. coli promoter gene sequences and classifies them according to biological promoter activity. Given a position in the middle of a window of 60 DNA sequence elements, instances in SPLICE are classified to donors, acceptors, or neither. Given an attribute-value description of a mushroom, the class of MUSHROOM instances is either edible or poisonous. Common to all four datasets is that they include only nominal attributes. Only MUSHROOM includes instances with

Table 4
Basic characteristics of datasets

Dataset	#class	#atts.	#val/att.	#examples	maj. class (%)
PALINDROME	2	6	3.0	729	96.3
PARITY	2	10	2.0	1024	50.0
MONK1	2	6	2.8	432	50.0
MONK2	2	6	2.8	432	67.1
VOTE	2	16	3.0	435	61.4
PROMOTERS	2	57	4.0	106	50.0
SPLICE	3	60	8.0	3191	50.0
MUSHROOM	2	22	–	5644	61.8
CAR	4	6	3.5	1728	70.0
NURSERY	5	8	3.4	12960	33.3
HOUSING	9	12	2.9	5000	29.9
BREAST	4	12	2.8	5000	41.5
EIS	5	14	3.0	10000	59.0
BANKING	3	17	2.2	5000	40.8

undefined attributes, which were for the purpose of this study removed since HINT—as described in this article—does not include explicit mechanism to handle such cases. As the concept hierarchies for these datasets are unknown to us and neither could we anticipate them, these datasets were only used for the study of generalization. That is, we were interested in HINT's accuracy on test data.

The remaining six datasets were obtained from multi-attribute decision models originally developed using DEX [6]. DEX models are hierarchical, so both the structure and intermediate concepts for these domains are known. The formalism used to describe the resulting model and its interpretation are essentially the same as those derived by decomposition. This makes models developed by DEX ideal benchmarks for the evaluation of decomposition. 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.

Six different DEX models were used. CAR is a model for evaluating cars based on their price and technical characteristics. This simple model was developed for educational purposes and is described in [5]. NURSERY is a real-world model developed to rank applications for nursery schools [33]. HOUSING is a model to determine the priority of housing loans applications [4]. This model is a part of a management decision support system for allocating housing loans that has been used since 1991 in the Housing Fund of Slovenia. BANKING, EIS and BREAST are three previously unpublished models for the evaluation of business partners in banking, evaluation of executive information systems, and breast-cancer risk assessment, respectively.

Each DEX model was used to obtain either 5000 or 10000 attribute-value instances with corresponding classes as derived from the model such that the class distribution was equal as in the dataset that would completely cover the attribute space. We have decided for either 5000 or 10000 examples because within this range HINT's behavior was found to be most relevant and diverse. The only exception is CAR where 1728 instances completely cover the attribute space.

6.2. Generalization

Here we study how the size of the training set affects HINT's ability to find a correct generalization. We construct learning curves by a variant of 10-fold cross-validation. In 10-fold cross validation, the data is divided to 10 subsets, of which 9 are used for training and the remaining one for testing. The experiment is repeated 10 times, each time using a different testing subset. Stratified splits are used, i.e., the class distribution of the original dataset and training and test sets are essentially the same. In our case, instead of learning from all examples from 9 subsets, only p percent of training instances from 9 subsets are randomly selected for learning, where p ranges from 10% to 100% in 10% steps. This adaptation of the standard method was necessary to keep test sets independent and compare classifiers as proposed in [42]. Note that when $p = 100\%$, this method is equivalent to the standard stratified 10-fold cross-validation.

HINT derived a concept hierarchy and corresponding classifier using the examples in the training set. The hierarchy was tested for classification accuracy on the test set. For each

p, the results are the average of 10 independent experiments. The attribute subset selection was used on a training set as described in Section 5. The resulting set of examples was then used to induce a concept hierarchy. HINT used the column multiplicity as a partition selection measure and determined the decomposability based on our first information-based measure ψ_v (Section 3.2). The bound set size b was limited to three.

The concept hierarchy obtained from training set was used to classify the instances in the test set. The instance's class value was obtained 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 latter case, the *default rule* was used that assigns the value of most frequently used class in the example set that defines the intermediate concept.

We compare HINT's learning curve to the one obtained by C4.5 inductive decision tree learner [36] run on the same data. As is the case with HINT, C4.5 was also required to induce a decision tree consistent with the training set. Hence, C4.5 used the default options except for -m1 (minimal number of instances in leafs was set to 1) and the classification accuracy was evaluated on unpruned decision trees. For several datasets, we have observed that subsetting (option -s) obtains a more accurate classifier: the learning curves for C4.5 were then developed both with and without subsetting, and the better one of the two was used for comparison with HINT. For each p, a binomial test [42] was used to test for significant differences between the methods using $\alpha = 0.01$ (99% confidence level).

The learning curves are shown in Figs. 9 and 10. Drawing symbols are ∘ for HINT and ◇ for C4.5. Where the difference is significant, the symbol for the better classifier is filled (● for HINT and ◆ for C4.5). The following can be observed:

- In general, for artificial datasets HINT performs significantly better than C4.5. For all four domains HINT's classification accuracy converges to 100% as the training set sizes increase. The percentage of instances needed to reach 100% accuracy varies between domains from 10 to 60%. Note that C4.5 never reaches the 100% classification accuracy.
- For VOTE, PROMOTERS, and MUSHROOM the differences are not significant, while for SPLICE C4.5 performs better. Only for MUSHROOM both classifiers reached the maximum accuracy.
- Common for all six DEX domains is that with small training sets HINT performs similar or worse than C4.5, while when increasing the training set size it gains the advantage and finally reaches the classification accuracy of close to or exactly 100%.

Note that for DEX and most of the artificial datasets, there exist useful concept structures in the form of concept trees. Given sufficient training instances, it is exactly in these domains where HINT outperforms C4.5. Repository datasets do not necessarily have such characteristics, which may be the reason why for these domains HINT's performance is worse. For example, the domain theory given with the SPLICE dataset [31] mentions several potentially useful intermediate concepts that share attributes. Thus these concepts form a concept *lattice* rather than a concept tree, and therefore cannot be discovered by HINT. Furthermore, DEX and artificial datasets indicate that although a domain possesses a proper structure discoverable by decomposition, HINT needs a sufficient number of training examples to induce good classifiers: HINT's performance suffers from undersampling more than C4.5's.
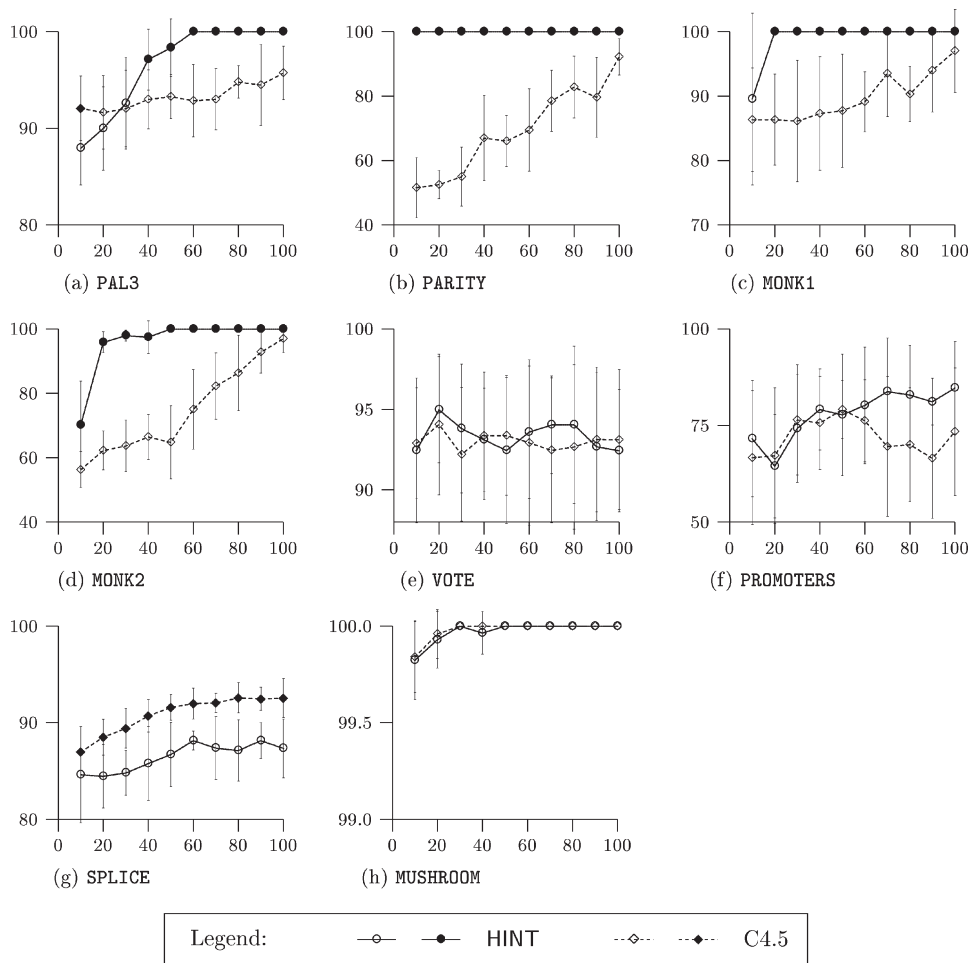
Fig. 9. Learning curves for artificial and repository datasets.

The number of attributes used in concept hierarchies depends on attribute subset selection (training data preprocessing by removing redundant attributes). This further depends on the existence of irrelevant attributes and on the coverage of attribute space by training set. Fig. 11 illustrates that with increasing coverage the number of attributes in induced structures increases and, in general, converges to a specific number of most relevant and nonredundant attributes. Interestingly, for PARITY and MONK1 domain HINT finds, as expected, that only 5 and 3 attributes are relevant, respectively. HINT converges to the use of about 10 attributes for VOTE, 12 for SPLICE, and 5 for MUSHROOM. For all DEX domains, with sufficiently large training sets HINT does not remove any of the attributes—this was expected since all attributes in these domains are relevant.

Attribute redundancy removal based on function decomposition is part of the HINT method. However, it could also be used for C4.5 as a preprocessor of the learning data. It
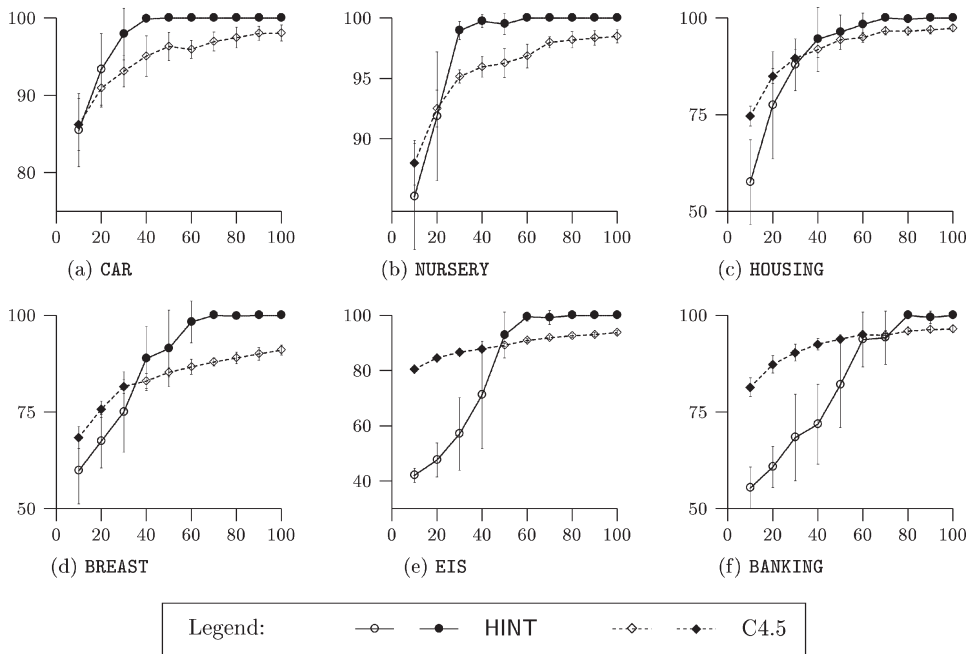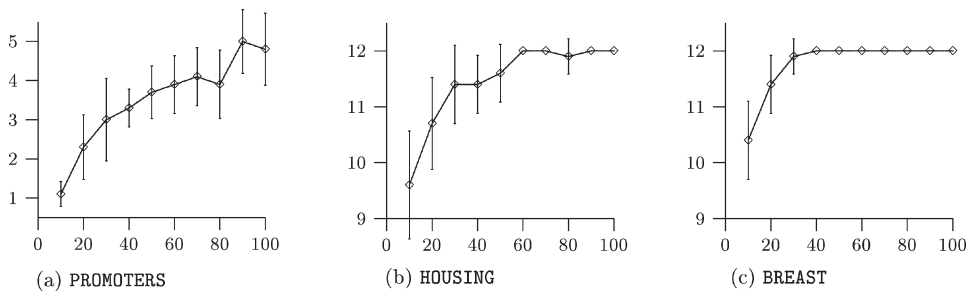
Fig. 10. Learning curves for DEX datasets.

Fig. 11. Number of attributes used in concept hierarchy as a function of training set size.

would be interesting to investigate how this would effect the performance of C4.5, but such an experiment is beyond the scope of this article.

The use of default rule for classification had a minor impact on classification accuracy. This holds even for the smallest training sets, where 95% of instances were classified without firing the default rule. In most cases, with increasing training set size this percentage monotonically increased to 100%.

6.3. Hierarchical concept structures

Induced concept structures were compared to those anticipated for artificial and DEX domains. For each of these, HINT converged to a single concept structure when increasing the training set size. For PALINDROME and PARITY, HINT induced expected structure of the type $(x_1 = x_6 \text{ AND } x_2 = x_5) \text{ AND } (x_3 = x_4)$ and $x_1 \text{ XOR } ((x_2 \text{ XOR } x_3) \text{ XOR } (x_4 \text{ XOR } x_5))$.

More interesting are the structures for MONK1 and MONK2. For MONK1, HINT develops a concept hierarchy (Fig. 12) that (1) correctly excludes irrelevant attributes x_3, x_4, and x_6, (2) transforms x_5 to x_5' by mapping four values of x_5 to only two values of x_5', (3) includes an intermediate concept c and its tabular representation for $x_1 = x_2$, and (4) relates c and x_5' with a tabular representation of the OR function. In other words, the resulting hierarchy correctly represents the target concept. For MONK2, although the discovered structure (Fig. 13) does not directly correspond to the original concept definition, it correctly reformulates the target concept by introducing concepts that count the number of ones in their arguments. Also note that all attributes that have more than two values are replaced by new binary ones.

For all DEX domains HINT converged to concept hierarchies that were very similar to original DEX models. A typical example is NURSERY, for which Fig. 14 shows the original DEX model and the concept hierarchy discovered by HINT. Note that the two structures are actually the same except that some original DEX intermediate concepts were additionally decomposed. Similarities of the same type were also observed for other DEX domains.

For NURSERY, no attributes were removed by preprocessing and redundancies were found only in attributes' domains: applicant's social status none and medium were found
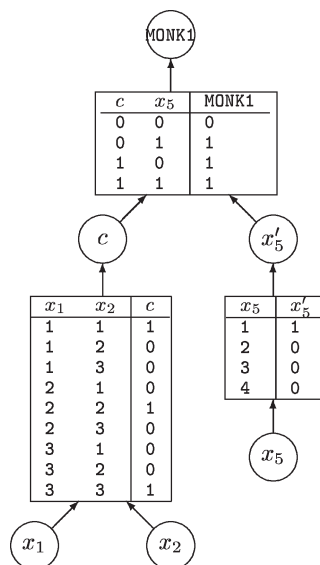

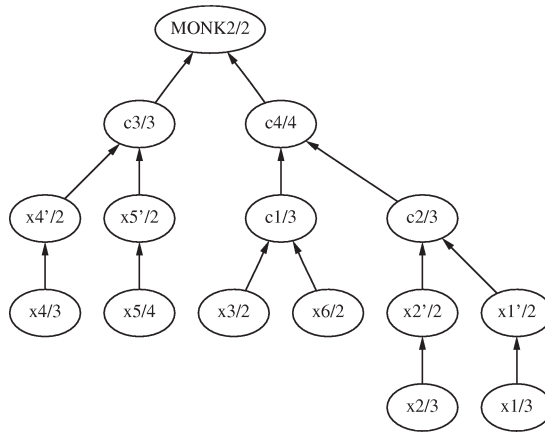Fig. 12. MONK1 feature hierarchy as discovered by HINT.

Fig. 13. The feature hierarchy discovered for `MONK2`. Each node gives a name of the feature and cardinality of its set of values.
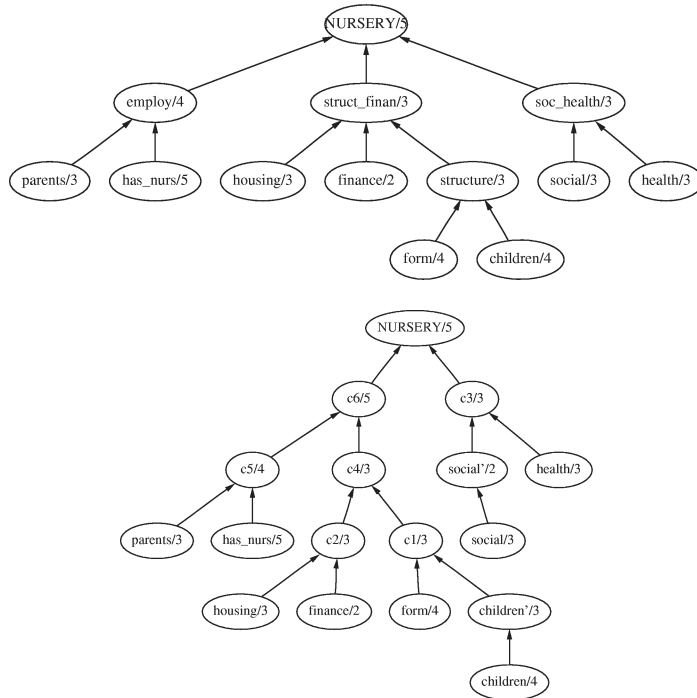

Fig. 14. Original (top) and discovered structure (bottom) for `NURSERY`.

equivalent, and there was no difference between a family having `3` or `more-than-3` children. Similar type of redundancies were also found in other DEX models. When a decision support expert that participated in the development of DEX models was asked

(a) PROMOTERS   (b) HOUSING   (c) BREAST

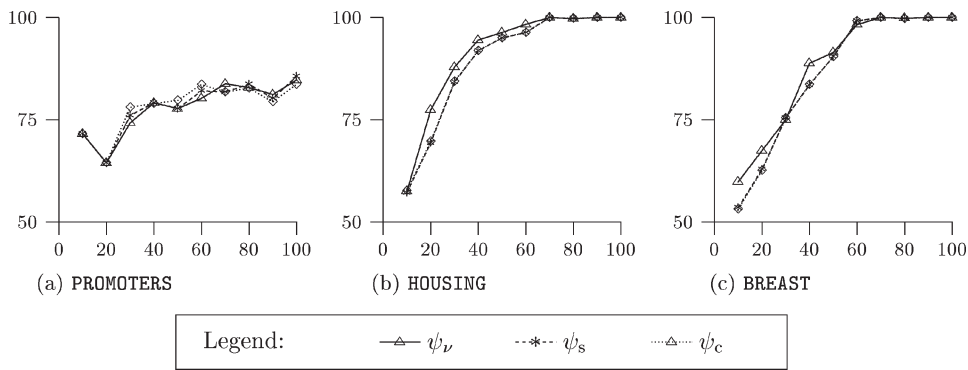Legend: ψ_ν ψ_s ψ_c

Fig. 15. Learning curves for different partition selection measures.

to comment on these findings, he indeed recognized most of them as those that were intentionally used in DEX models with future extension and specialization of model functions in mind.

6.4. Comparison of partition selection measures

So far, all experiments with HINT used column multiplicity as the partition selection measure. The same experiments were also performed under the same settings but using two information-based partition selection measures. The study revealed that there are no significant differences in terms of classification accuracy. Fig. 15 depicts typical examples of learning curves; only average classification accuracies are shown, which are for all training set sizes insignificantly different for all three measures. Moreover and especially for artificial and DEX datasets, HINT converged to the same concept structure for either of the selection measures. To conclude, it is interesting that a measure as simple as column multiplicity performed equally well as the other two more complex measures.

7. 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* [44] and used it as an evaluation mechanism for his checkers playing programs. A signature table system is a tree consisting of input, intermediate, and a single output variable, and is essentially an identical representation of concept trees as used in this article. Signature tables define the intermediate concepts and use signatures (examples) that completely cover the attribute space. 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. Samuel used manually defined concept structures with two layers of intermediate concepts. Learning was based on presenting a set of book moves to the concept hierarchy and adjusting the output values of the signatures according to the correlation coefficient computed from learning examples.

Compared to his previous approach that was based on the learning of the coefficients in a linear evaluation polynomial [43], Samuel showed that the use of a signature table system significantly improves the performance. Samuel's approach was later studied and improved by Biermann et al. [2], but still required the concept structure to be given in advance.

While, within machine learning, Samuel and Biermann et al. may be the first to realize the power of using concept hierarchies, fundamentals of the approach that can discover such hierarchies were defined earlier in the area of switching circuit design. Curtis [10] reports that in the late 1940s and 1950s several switching circuit theorists considered this subject and in 1952 Ashenhurst reported on a unified theory of decomposition of switching functions [1]. The method proposed by Ashenhurst decomposes the truth table of a Boolean function to be realized with standard binary gates. Most of other related work of that time is reported and reprinted in [10], where Curtis compares the decomposition approach to other switching circuit design approaches and further formalizes and extends the decomposition theory. Besides a disjoint decomposition, where each variable can appear as input in just one of the derived tables, Curtis defines a nondisjoint 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 requires a set of examples that completely cover the attribute space.

Recently, the Ashenhurst–Curtis approach was substantially improved by research groups of M.A. Perkowski, T. Luba, and T.D. Ross. Perkowski and Uong [34] and Wan and Perkowski [50] propose a graph coloring approach to the decomposition of incompletely specified switching functions. A different approach is presented by Luba and Selvaraj [23]. Their decomposition algorithms are able to generalize. A generalization of function decomposition when applied to a set of simple Boolean functions was studied by Ross et al. [40] and Goldman [14]. The authors indicate that the decomposition approach to switching function design may be termed knowledge discovery as functions and features not previously anticipated can be discovered. A similar point, but using different terminology, was made already by Curtis [10], who observed that the same truth table representing a Boolean function might have different decompositions.

Feature discovery has been at large investigated by constructive induction, a recently active field within machine learning. The term was first used by Michalski [25], 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 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 [24,35,38]. 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 the induced classifier, while the above described function decomposition approaches focused primarily on the 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 [47].

Decomposition with nominal-valued attributes and classes may be regarded as a straightforward extension of Ashenhurst–Curtis approach. Such an extension was described by Biermann et al. [2]. Alternatively, Luba [22] proposes a decomposition where multi-valued intermediate concepts are binarized. Files et al. [13] propose a decomposition approach for k-valued logic where both attributes and intermediate concepts take at most k values.

A concept structure as used in this article defines a declarative bias over the hypothesis space. Biermann et al. [2] showed that concept structure significantly limits the number of representable functions. This was also observed by Russell [41], who proved that tree-structured bias can reduce the size of concept language from doubly-exponential to singly exponential in the number of attributes. Tadepalli and Russell [48] show that such bias enables PAC-learning of tabulated functions within concept structure. Their approach for decomposition of Boolean functions requires the concept structure to be given in advance. Their learning algorithm differs from the function decomposition approaches in that it uses both examples and queries, i.e., asks the oracle for the class value of instances that are needed in derivation but not provided in the training examples. Similar to function decomposition, the learning algorithm of Tadepalli and Russell induces intermediate concepts that are lower in the hierarchy first. As with Ashenhurst–Curtis decomposition, the resulting classifiers are consistent with training examples. Queries are also used in PAC-learning described by Bshouty et al. [8]. Their algorithm identifies both concept structures and their associated tabulated functions, but can deal only with Boolean functions with symmetric and constant fan-in gates. Within PAC-learning, Hancock et al. [16] learn nonoverlapping perceptron networks from examples and membership queries. An excellent review of other related work in PAC-learning that uses structural bias and queries is given in [48].

Function decomposition is also related to construction of oblivious read-once decision graphs (OODG). OODGs are rooted, directed acyclic graphs that can be divided into levels [17]. All nodes at a level test the same attribute, and all edges that originate from one level terminate at the next level. Like with decision trees, OODG leaf nodes represent class values. OODGs can be regarded as a special case of decomposition, where decomposition structures are of the form $f_1(x_1, f_2(x_2, \ldots, f_n(x_n)))$ and where x_n is at the top of a decision graph and the number of nodes at each level equals the number of distinct output values used by corresponding function f_i. In fact, decision graphs were found as a good form of representation of examples to be used by decomposition [13,20,21]. Within machine learning, the use of oblivious decision graphs was studied by Kohavi [17]. Graphs induced by his learning algorithm are consistent with training examples, and for incomplete datasets the core of the algorithm is a graph coloring algorithm similar to the one defined by Perkowski and Uong [34].

Of other machine learning approaches that construct concept hierarchies we here mention Muggleton's DUCE [29,30] which uses transformation operators to compress the given examples by successive generalization and feature construction. Nevill-Manning and Witten [32] describe SEQUITUR, an algorithm that infers a hierarchical structure from a sequence of discrete symbols. Although there are some similarities with function decomposition (e.g., maintaining consistency and induction of new features), DUCE and SEQUITUR are essentially different in both the algorithmic and representational aspects.

Within machine learning, there are other approaches based on problem decomposition, but where the problem is decomposed by an expert and not discovered by a machine. A well-known example is structured induction, a term introduced by Donald Michie and applied by Shapiro and Niblett [46] and Shapiro [45]. Their approach is based on a manual decomposition of the problem and an expert-assisted selection and classification of examples to construct rules for intermediate 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 [26] emphasized the important role of structured induction in the future and listed several real problems that had been solved in this way.

Mozetič [7,27,28] employed another scheme for structuring the learning problem. That approach was particularly aimed at automated construction of system models from input–output observations of the system's behavior. The structure of the learning problem, specified by a Prolog clause, corresponded to the physical structure of the modeled system in terms of the system's components and connections among them. In an experiment, a substantial part of a qualitative model of the heart was induced from examples of the behavior of the heart. It was shown that the structuring of the domain very significantly improved the effectiveness of learning compared to unstructured learning. Again, the structure of the system was specified by the user and not induced automatically.

Concept hierarchy has also been used in a multi-attribute decision support expert system shell DEX [6] which has its roots in DECMAK methodology [3,12]. There, a tree-like structure of variables is defined by an expert, and several tools assist in the acquisition of decision rules. These are, like Samuel's signature tables, used to derive the values of intermediate and output variables. DEX also allows different representations of user-defined decision tables, including decision trees [45] and decision rules [39]. DEX has been applied in more than 50 real decision making problems.

The method presented in this article essentially borrows from three different research areas: it shares the motivation with structured induction and structured approach to decision support, while the core of the method is based on Ashenhurst–Curtis function decomposition. In comparison with related work, the present article is original in the following respects: new method for handling multi-valued attributes and classes, improved decomposition heuristics, treatment of redundancies, emphasis on generalization effects of decomposition, paying strong attention to the discovery of meaningful concept hierarchies, and experimental evaluation on machine learning problems. Our earlier experiments in function decomposition applied to DEX domains were presented in [53,54].

8. Conclusion

We 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 induce the relationships between the attributes, newly discovered concepts, and the 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 number of attributes, and introduce intermediate concepts. Generalization also occurs in this process.

We have evaluated the decomposition-based learning method on several datasets. In particular, we studied the accuracy of the induced descriptions by HINT and its capability to discover meaningful hierarchies. For all datasets where useful hierarchies existed, HINT significantly outperformed C4.5 and found relevant concept hierarchies, provided that enough examples were used for training. Experiments show that decomposition is more sensitive to undersampling and, especially in more complex datasets, C4.5 performed relatively better with small training sets. For other datasets, with no useful concept structure, C4.5 and HINT performed similarly in all but one domain.

In terms of the meaningfulness of discovered structures, the most significant experiments were those with DEX domains. For these domains HINT's task was to reconstruct the underlying concept hierarchy. We have observed that for all six domains investigated, HINT converges to concept hierarchies that are very similar or identical to those anticipated. It should be emphasized that we consider these similarities of concept structures as the most significant indicator of HINT's success.

The approach described in this article is limited to consistent datasets and nominal features. It is therefore desired to extend the approach to discover new features from noisy data, and from data that comprises continuous features. To handle noisy data, a minimal-error decomposition was recently proposed [52]. It is based on a representation of training examples with class distributions and uses successive column merging of partition matrix, so that the expected error of classification is minimized. For continuously-valued datasets, the function decomposition method was proposed in [11]. They both present preliminary results which strongly encourage further development in this direction and integration of their techniques into common function decomposition framework. The feature construction aspect of HINT is investigated in more detail in [55].

Acknowledgements

References

[1] R.L. Ashenhurst, The decomposition of switching functions, Technical Report, Bell Laboratories BL-1(11), 1952, pp. 541–602.

[2] A.W. Biermann, J. Fairfield, T. Beres, Signature table systems and learning, IEEE Trans. Syst. Man Cybern. 12 (5) (1982) 635–648.

[3] M. Bohanec, I. Bratko, V. Rajkovič, An expert system for decision making, in: H.G. Sol (Ed.), Processes and Tools for Decision Support, North-Holland, Amsterdam, 1983.

[4] M. Bohanec, B. Cestnik, V. Rajkovič, A management decision support system for allocating housing loans, in: P. Humphreys, L. Bannon, A. McCosh, P. Migliarese (Eds.), Implementing System for Supporting Management Decisions, Chapman & Hall, London, 1996, pp. 34–43.

[5] M. Bohanec, V. Rajkovič, Knowledge acquisition and explanation for multi-attribute decision making, in: Proceedings 8th Internat. Workshop on Expert Systems and Their Applications, Avignon, France, 1988, pp. 59–78.

[6] M. Bohanec, V. Rajkovič, DEX: An expert system shell for decision support, Sistemica 1 (1) (1990) 145–157.

[7] I. Bratko, I. Mozetič, N. Lavrač, KARDIO: A Study in Deep and Qualitative Knowledge for Expert Systems, MIT Press, Cambridge, MA, 1989.

[8] N.H. Bshouty, T.R. Hancock, L. Hellerstein, Learning boolean read-once formulas over generalized bases, J. Comput. System Sci. 50 (3) (1995) 521–542.

[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1989.

[10] H.A. Curtis, A New Approach to the Design of Switching Functions, Van Nostrand, Princeton, NJ, 1962.

[11] J. Demšar, B. Zupan, M. Bohanec, I. Bratko, Constructing intermediate concepts by decomposition of real functions, in: M. van Someren, G. Widmer (Eds.), Proceedings European Conference on Machine Learning, ECML-97, Prague, April 1997. Springer, Berlin, 1997, pp. 93–107.

[12] J. Efstathiou, V. Rajkovič, Multiattribute decisionmaking using a fuzzy heuristic approach, IEEE Trans. Syst. Man Cybern. 9 (1979) 326–333.

[13] C. Files, R. Drechsler, M. Perkowski, Functional decomposition of MVL functions using multi-valued decision diagrams, in: Proceedings International Symposium on Multi-Valued Logic, May 1997.

[14] J.A. Goldman, Pattern theoretic knowledge discovery, in: Proceedings 6th Internat. IEEE Conference on Tools with AI, 1994.

[15] R.L. Graham, D.E. Knuth, O. Patashnik, Concrete Mathematics: A Foundation for Computer Science, Addison-Wesley, Reading, MA, 1994.

[16] T.R. Hancock, M. Golea, M. Marchand, Learning nonoverlaping perceptron networks from examples and membership queries, Machine Learning 16 (3) (1994) 161–183.

[17] R. Kohavi, Bottom-up induction of oblivious read-once decision graphs, in: F. Bergadano, L. De Raedt (Eds.), Proceedings European Conference on Machine Learning (ECML-94), Springer, Berlin, 1994, pp. 154–169.

[18] I. Kononenko, Estimating attributes, in: F. Bergadano, L. De Raedt (Eds.), Proceedings European Conference on Machine Learning (ECML-94), Springer, Berlin, 1994, pp. 171–182.

[19] I. Kononenko, E. Šimec, M. Robnik Šikonja, Overcoming the myopia of inductive learning algorithms with ReliefF, Applied Intelligence J. 7 (1) (1997) 39–56.

[20] Y.-T. Lai, K.-R.R. Pan, M. Pedram, OBDD-based function decomposition: Algorithms and implementation, IEEE Trans. Computer Aided Design of Integrated Circuits and Systems 15 (8) (1996) 977–990.

[21] Y.-T. Lai, M. Pedram, S. Sastry, BDD-based decomposition of logic functions with application to FPGA synthesis, in: Proceedings 30th DAC, 1993, pp. 642–647.

[22] T. Luba, Decomposition of multiple-valued functions, in: Proceedings 25th Internat. Symposium on Multiple-Valued Logic, Bloomington, IN, May 1995, pp. 256–261.

[23] T. Luba, H. Selvaraj, A general approach to boolean function decomposition and its application in FPGA-based synthesis, VLSI Design 3 (3–4) (1995) 289–300.

[24] R.S. Michalski, A theory and methodology of inductive learning, in: R. Michalski, J. Carbonnel, T. Mitchell (Eds.), Machine Learning: An Artificial Intelligence Approach, Morgan Kaufmann, Paolo Alto, CA, 1983, pp. 83–134.

[25] R.S. Michalski, Understanding the nature of learning: Issues and research directions, in: R. Michalski, J. Carbonnel, T. Mitchell (Eds.), Machine Learning: An Artificial Intelligence Approach, Morgan Kaufmann, Los Altos, CA, 1986, pp. 3–25.

[26] D. Michie, Problem decomposition and the learning of skills, in: N. Lavrač, S. Wrobel (Eds.), Machine Learning: ECML-95, Lecture Notes in Artificial Intelligence Vol. 912, Springer, Berlin, 1995, pp. 17–31.

[27] I. Mozetič, Learning of qualitative models, in: I. Bratko, N. Lavrač (Eds.), Progress in Machine Learning, Sigma Press, Wilmslow, England, 1987.

[28] I. Mozetič, The role of abstractions in learning of qualitative models, in: Proceedings 4th Internat. Workshop on Machine Learning, Irvine, CA, Morgan Kaufmann, Los Altos, CA, 1987.

[29] S. Muggleton, Structuring knowledge by asking questions, in: I. Bratko, N. Lavrač (Eds.), Progress in Machine Learning, Sigma Press, Wilmslow, England, 1987, pp. 218–229.

[30] S. Muggleton, Inductive Acquisition of Expert Knowledge, Addison-Wesley, Wokingham, England, 1990.

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

[32] C.G. Nevill-Manning, I.H. Witten, Identifying hierarchical structure in sequences: A linear-time algorithm, J. Artificial Intelligence Res. 7 (1997) 67–82.

[33] M. Olave, V. Rajkovič, M. Bohanec, 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 (North-Holland), Amsterdam, 1989, pp. 145–160.

[34] M. Perkowski, H. Uong, Automatic design of finite state machines with electronically programmable devices, in: Record of Northcon '87, Portland, OR, 1987, pp. 16/4.1–16/4.15.

[35] B. Pfahringer, Controlling constructive induction in CiPF, in: F. Bergadano, L. De Raedt (Eds.), Proceedings European Conference on Machine Learning (ECML-94), Springer, Berlin, 1994, pp. 242–256.

[36] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA, 1993.

[37] R. Quinlan, Induction of decision trees, Machine Learning 1 (1) (1986) 81–106.

[38] H. Ragavan, L. Rendell, Lookahead feature construction for learning hard concepts, in: Proceedings 10th Internat. Machine Learning Conference, Morgan Kaufmann, San Mateo, CA, 1993, pp. 252–259.

[39] V. Rajkovič, M. Bohanec, Decision support by knowledge explanation, in: H.G. Sol, J. Vecsenyi (Eds.), Environments for Supporting Decision Process, Elsevier Science, Amsterdam, 1991.

[40] T.D. Ross, M.J. Noviskey, D.A. Gadd, J.A. Goldman, Pattern theoretic feature extraction and constructive induction, in: Proceedings ML-COLT-'94 Workshop on Constructive Induction and Change of Representation, New Brunswick, NJ, 1994.

[41] S.J. Russell, Tree-structured bias, in: Proceedings AAAI-88, St. Paul, MN, Morgan Kaufmann, San Mateo, CA, 1988, pp. 641–645.

[42] S.L. Salzberg, On comparing classifiers: Pitfalls to avoid and a recommended approach, Data Mining and Knowledge Discovery 1 (1997) 317–328.

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

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

[45] A.D. Shapiro, Structured induction in expert systems, Turing Institute Press (in association with Addison-Wesley), 1987.

[46] A.D. Shapiro, T. Niblett, Automatic induction of classification rules for a chess endgame, in: M.R.B. Clarke (Ed.), Advances in Computer Chess 3, Pergamon, Oxford, 1982, pp. 73–92.

[47] I. Stahl, An overview of predicate invention techniques in ILP, in: ESPRIT BRA 6020: Inductive Logic Programming, 1991.

[48] P. Tadepalli, S. Russell, Learning from examples and membership queries with structured determinations, Machine Learning, to appear.

[49] S.B. Thrun et al., A performance comparison of different learning algorithms, Technical Report, Carnegie Mellon University CMU-CS-91-197, Pittsburgh, PA, 1991.

[50] W. Wan, M.A. Perkowski, A new approach to the decomposition of incompletely specified functions based on graph-coloring and local transformations and its application to FPGA mapping, in: Proceedings IEEE EURO-DAC-'92, Hamburg, 1992, pp. 230–235.

[51] D.J.A. Welsh, M.B. Powell, An upper bound on the chromatic number of a graph and its application to timetabling problems, Computer J. 10 (1967) 85–86.

[52] B. Zupan, Machine learning based on function decomposition, Ph.D. Thesis, University of Ljubljana, 1997. Available at `http://www-ai.ijs.si/BlazZupan/papers.html`.

[53] B. Zupan, M. Bohanec, I. Bratko, B. Cestnik, A dataset decomposition approach to data mining and machine discovery, in: D. Heckerman, H. Mannila, D. Pregibon, R. Uthurusamy (Eds.), Proceedings 3rd Internat. Conference on Knowledge Discovery and Data Mining (KDD-97), Irvine, CA, AAAI Press, 1997, pp. 299–303.

[54] B. Zupan, M. Bohanec, I. Bratko, J. Demšar, Machine learning by function decomposition, in: D.H. Fisher Jr. (Ed.), Proceedings 14th Internat. Conference on Machine Learning (ICML-97), Morgan Kaufmann, San Mateo, CA, 1997, pp. 421–429.

[55] B. Zupan, M. Bohanec, J. Demšar, I. Bratko, Feature transformation by function decomposition, IEEE Intelligent Systems and Their Applications 13 (2) (1998) 38–43.