Constructing Intermediate Concepts by Decomposition of Real Functions

Janez Demšar[1], Blaž Zupan[2], Marko Bohanec[2], Ivan Bratko[1,2]

[1] Faculty of Computer and Information Sciences, 1000 Ljubljana, Slovenia
{janez.demsar, ivan.bratko}@fri.uni-lj.si
[2] Jozef Stefan Institute, 1000 Ljubljana, Slovenia
{blaz.zupan, marko.bohanec}@ijs.si

Abstract. In learning from examples it is often useful to expand an attribute-vector representation by intermediate concepts. The usual advantage of such structuring of the learning problem is that it makes the learning easier and improves the comprehensibility of induced descriptions. In this paper, we develop a technique for discovering useful intermediate concepts when both the class and the attributes are real-valued. The technique is based on a decomposition method originally developed for the design of switching circuits and recently extended to handle incompletely specified multi-valued functions. It was also applied to machine learning tasks. In this paper, we introduce modifications, needed to decompose real functions and to present them in symbolic form. The method is evaluated on a number of test functions. The results show that the method correctly decomposes fairly complex functions. The decomposition hierarchy does not depend on a given repertoir of basic functions (background knowledge).

1 Introduction

A learning problem can often be formulated as the problem of reconstructing a function f of a number of arguments $\mathbf{x} = x_1, x_2, \ldots$ from a given set of example points $f(\mathbf{x}_j)$. In the usual machine learning terminology, $x_1, x_2, \ldots$ are called attributes, and f is called the class. The usual induction algorithms reconstruct f by considering all the attributes at the same time. However, often it is beneficial to find useful "intermediate" concepts which would allow a decomposition of the learning problem. Hopefully, f would be easier to express in terms of suitable intermediate concepts, and in turn, these would be easy to express in terms of the original attributes or further intermediate concepts. It is generally believed that such a structuring of the learning domain would also lead to more comprehensible description of the learned concepts.

In this paper, we develop a technique for discovering useful intermediate concepts when both the class and the attributes are real-valued. It is based on function decomposition that results in a hierarchy of intermediate functions which can be illustrated by a kind of dataflow diagram (Fig. 1). The technique works bottom-up by selecting a subset of the original attributes, say $\{x_2, x_3\}$, and constructing a function ϕ_1 so that it would succesfully replace the two attributes x_2, x_3. As a result of this step, a new attribute ϕ_1 is constructed and the process recursively combines the attribute set $\{x_1, \phi_1, x_4, x_5\}$.

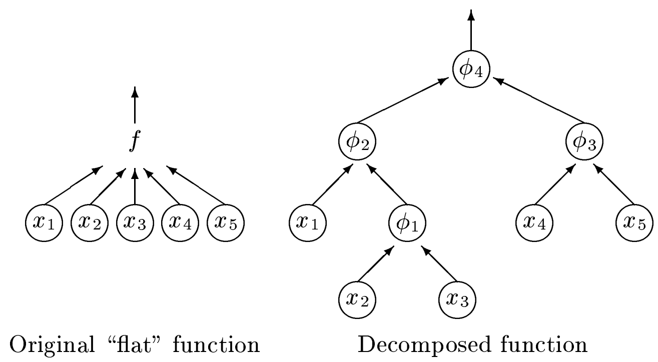

Fig. 1. "Flat" and decomposed function

Functional decomposition is a method which, given a tabular representation of a function, discovers a hierarchy of appropriate subfunctions and variables. The output of the algorithm is a decomposition tree or, generally, a directed acyclic graph with input variables as leaves and subfunctions as internal nodes. A decomposition algorithm was originally developed in late 1940's and 1950's by Ashenhurst [1] and Curtis [2] to be used for decomposition of boolean functions in switching circuits design. However, the method was rarely used in practice, mostly because of its computational intractability. Much later, the interest in the algorithm has been renewed. Perkowski et al. [5] improved the original algorithm to handle incompletely specified functions, and Luba [4] proposed to decompose multi-valued functions by representing a multi-valued variable by a set of Boolean variables. Zupan and Bohanec [7] developed an algorithm that induces a hierarchy of multi-valued variables without the need to represent them as Boolean. Also, their work shows that the algorithm is applicable in fairly complex machine learning tasks.

Not much work has been done to extend the algorithm to decomposition of real-valued functions. Ross [6] discusses the possible use of the method for functions with real valued outputs and inputs but he does not propose any algorithm for general use.

In this paper, we extended the algorithm to handle continuous variables and functions. The proposed method not only discovers a suitable function hierarchy but enables symbolic representation of the discovered function, using a predefined set of basic functions, like $\sin, \cos, \exp$ or $\ln$.

The paper is organized as follows. Section 2 introduces the real function decomposition method, which is experimentally evaluated in Section 3. Section 4 concludes the paper and outlines possible directions of further work.

2 Method

This section first introduces the basic algorithm for functional decomposition of nominal functions. Then, it focuses on the changes of this algorithm that are needed to perform the decomposition of real function.

2.1 The Basic Decomposition Algorithm

The input for algorithms that are based on Curtis' function decomposition algorithm [2] is a function $f(X)$, "sampled" in a finite number of points, where X is an argument vector. The function is presented as a table of attribute-value vectors, each consisting of values of input variables $\mathbf{x}_k$ and a function value $z_k = f(\mathbf{x}_k)$. In the usual machine learning terminology, each row of this table corresponds to an example. The basic step of the decomposition consists of two substeps:

- find a suitable partition of the set of input variables (X) into "free" (A) and "bound" (B) sets, $A \cup B = X$,
- find appropriate functions F and ϕ such that $f(X) = F(A, \phi(B))$.

The basic step is then recursively repeated on functions F and ϕ.

The partition can be selected using heuristic methods [5]. Alternative approach is to investigate all possible partitions and choose the one that induces the best functions ϕ and F according to some criterion. Often, we speed up the partition selection by examining only disjunctive splits, $A \cap B = \emptyset$, and/or decompositions with only two bound variables, $|B| = 2$. However, there are cases when such restrictions prevent the algorithm from discovering an appropriate function hierarchy or even from discovering any hierarchy at all.

Function ϕ is not uniquely defined by the set of bound variables. The algorithm first determines which combinations of values of input variables from B must not yield the same value of ϕ. It does so by finding all pairs of examples $f(\mathbf{x}_j) = z_j$, $f(\mathbf{x}_k) = z_k$ with pairwise equal values of free variables, $\mathbf{a}_j = \mathbf{a}_k$, and different function value, $z_j \neq z_k$. The intermediate function ϕ must have different values for $\mathbf{b}_j$ and $\mathbf{b}_k$, otherwise there would not exist any function F such that $F(\mathbf{a}_j, \phi(\mathbf{b}_j)) = z_j$ and $F(\mathbf{a}_k, \phi(\mathbf{b}_k)) = z_k$. This fact can be easily proved by contradiction; if $\mathbf{a}_j = \mathbf{a}_k$ and we set $\phi(\mathbf{b}_j) = \phi(\mathbf{b}_k)$ then if F existed, it would obviously have the same value for both examples, $F(\mathbf{a}_j, \phi(\mathbf{b}_j)) = F(\mathbf{a}_k, \phi(\mathbf{b}_k))$, which leads to $z_j = z_k$.

Pairs $(\mathbf{b}_j, \mathbf{b}_k)$ that must not give the same value of ϕ are called incompatible. The incompatibility relation is presented in the form of the incompatibility graph. By coloring the graph (or by finding the maximum clique on its complement, the compatibility graph), possible inputs for ϕ, vectors $\mathbf{b_j}$, are divided into M subsets; members of the same subset M_i yield the same value $\phi(\mathbf{b}_j) = m_i$. The intermediate function ϕ therefore maps each $\mathbf{b}_j$ to the corresponding set M_i. The value of induced variable m_i is added to each rule to replace variables from $B \setminus A$.

For purposes of switching circuits design, the algorithm was first used on Boolean functions. An extension of this approach to handle nominal functions with more than two different output values is presented in [7].

2.2 Algorithm for Real-Function Decomposition

The basis of our method is the decomposition algorithm described in the previous section, limited to disjunctive splits, $A \cap B = \emptyset$, with two bound variables, $|B| = 2$. When adapting it for decomposition of real functions, several problems have to be dealt with.

First, since each variable generally has an infinite domain, it is practically impossible that the learning set contains any instances with pairwise equal values of free variables which are needed to show the incompatibilities of bound values.

The next problem is the interpretation of intermediate function ϕ: the original algorithm defines values of ϕ to serve as indices to sets M_i. The function ϕ is nominal rather than ordinal and since the algorithm was developed for the design of switching circuits where the interpretation is unnecessary, it does not intend to present the discovered function in the symbolic form, i.e. to recognize it as, for example, an **and** function. Instead, functions ϕ and F are left in tabular form. As shown in [7], the interpretation of a discrete function ϕ can be done manually provided that its input and output variables have only small number of possible values. When using the algorithm to *learn* the function of *real values*, rule tables that are constructed by the original algorithm are normally very large, and the search for a symbolical form of the intermediate functions cannot be left to the expert.

Interpretation of an intermediate function is also important because it ensures that the learned function is not defined only on points that appear in the rule table.

Discretization of examples. One possible way to generate incompatible pairs is to discretize the examples. After the discretization, some pairs of examples might have pairwise equal values of all the arguments. A straightforward discretization of function values may assign different discrete function values to such pairs. This way, the function would become ambiguous. To avoid this, the function value is "granulized": all pairs of examples with the same discretized values of variables $\mathbf{x}'_j = \mathbf{x}'_k$ are examined and the maximal difference of their function values $g = \max_{j,k} |f(\mathbf{x}_j) - f(\mathbf{x}_k)|$ is used for the size of the grain. Function values z_{j_1} and z_{j_2} are considered different if $|z_{j_1} - z_{j_2}| > g$. This approach works optimally for functions f with a constant gradient over the whole definition area. If this is not the case, the discretized learning examples accurately describe the areas with larger gradient, but underrepresents all other areas. The consequences and solutions of the problem shall be discussed later.

A crucial problem of discretization is determining the most suitable number of intervals. Coarse discretization can significantly lower the accuracy of constants in derived functions or even cause an incorrect decomposition. On the other hand, a finer discretization results in a sparser coverage of the domain of the function. As a consequence, the incompatibility graph has low connectivity and bears almost no information on intermediate function ϕ since there exist many different optimal colorings, yielding many different functions.

The interpretation of the intermediate function. The result of coloring the incompatibility graph is a division of the bound variables' space into the areas M_i with points which will yield the same value of ϕ, i.e. $M_i = \{(x_{i,j}, y_{i,j}) : \phi(x_{i,j}, y_{i,j}) = c_i\}$. In other words, the coloring proposes contour lines – or, because of discretization, contour strips – of the real function ϕ. Unfortunately, it provides neither the numerical values c_i nor the difference between function values on neighbouring contour strips (the gradient of ϕ). The symbolic form of ϕ has to be obtained from the shape of the strips.

First, consider a linear function ϕ. Its contour strips are straight lines. If we plot a colored graph of a discretized linear function in a coordinate system, contour strips are visible as strips of the same color. The function ϕ that we are looking for is of the form

$$\phi(x, y) = k_1(ax + y) + k_2 \tag{1}$$

Coefficients k_1 and k_2 cannot be determined at this stage since they do not have any impact on the shape of the strips. They should be determined in further steps of decomposition and incorporated in the intermediate function at the parent node.

The coefficient a is derived from the slope of the strips. Ideally, all the points $(x_{i,j}, y_{i,j}) \in M_i$ would lie on the same line $ax + y = c_i$. Because of discretization and, possibly, noise, the points are actually scattered around this line. As the measure of fit we choose the sum of squared Euclidean distances from the line,

$$E_i(a, c_i) = \sum_{j=1}^{n_i} \left[\frac{ax_{i,j} + y_{i,j} - c_i}{\sqrt{a^2 + 1}} \right]^2 \tag{2}$$

with $(x_{i,j}, y_{i,j}) \in M_i$ and $n_i = |M_i|$. To obtain the optimal a, the total sum of squared distances is minimized using the partial derivative

$$\frac{\partial E(a, \mathbf{c})}{\partial a} = \frac{\partial \sum_{i=1}^{M} E_i(a, c_i)}{\partial a} = 0 \tag{3}$$

Solution of this equation for a gives the optimal value of a. The quality of the approximation can be measured by Pearson's correlation coefficient r which, in its original form

$$r = \frac{\sum_j (x_j - \overline{x})(y_j - \overline{y})}{\sqrt{\sum_j (x_j - \overline{x})^2} \sqrt{\sum_j (y_j - \overline{y})^2}} \tag{4}$$

measures the linear correlation between variables x_j and y_j. Since we deal with more than one group of points, the Pearson's coefficient must be generalized to

$$r_g = \frac{\sum_{i,j}(x_{i,j} - \overline{x}_i)(y_{i,j} - \overline{y}_i)}{\sqrt{\sum_{i,j}(x_{i,j} - \overline{x_i})^2}\sqrt{\sum_{i,j}(y_{i,j} - \overline{y_i})^2}} \tag{5}$$

where $\overline{x_i} = \frac{1}{n_i}\sum_{j=1}^{n_i} x_{i,j}$ and $\overline{y_i} = \frac{1}{n_i}\sum_{j=1}^{n_i} y_{i,j}$. In this form, the coefficient is still normalized to be between -1 and 1, with $|r_g| = 1$ meaning the maximum linearity of the strips and $|r_g| = 0$ no linearity.

Other, non-linear functions are sought by transformation to a linear function. For example, contour strips of $\phi(x, y) = x^a y$ are same as those of $\phi(x, y) = a\ln(x) + \ln(y)$ or $\phi(x, y) = aX + Y$ where $X = \ln(x)$ and $Y = \ln(y)$. Our algorithm searches for all the functions of the form $\phi(x, y) = ag(x) + h(y)$ and $\phi(x, y) = [g(x)]^a\, h(y)$ by using transformations $x_{i,j} \to g(x_{i,j})$, $y_{i,j} \to h(y_{i,j})$ and $x_{i,j} \to a\ln g(x_{i,j})$, $y_{i,j} \to \ln h(y_{i,j})$, respectively. Functions g and h are from a predefined set of basic functions, for example $\{\mathrm{Id}, \sin, \exp, \ln\}$. All the possible functions are evaluated and the one with the greatest $|r_g|$ is chosen.

The root of the decomposition tree. The decomposition process stops when the free set of attributes is empty. The values of the decomposed function F are not necessarily equal to the values of the original function f. However, for an analyticaly expressible function f, if the algorithm finds the correct decomposition and there is no noise, the method guarantees that the value of $f(\mathbf{x})$ can be reconstructed from $F(\mathbf{x})$. Our program tries to find a function g and constants a and n, such that $f(\mathbf{x}) = ag(F(\mathbf{x})) + n$. The function g is from the same set of basic functions as mentioned above. For each function, a and n are found by the classical least-squares method and the difference between $f(\mathbf{x})$ and $ag(F(\mathbf{x}))+n$ is measured by corrected relative error [3]. The most accurate function is added to the decomposition tree as the root's parent.

Discarding invalid contour strips. Besides the noise in the data, the algorithm also encounters the noise caused by discretization of variables and granulation of function value. The noise of variables is partially reduced by robust statistic methods used for deriving ϕ. A more serious problem occurs as a consequence of the granulation of function value which affects the graph coloring, especially when function's gradient strongly changes across the definition area. Areas with small gradient are covered with much wider contour strips than areas with larger gradient; in some cases they also differ in shape.

A simple and effective method that can overcome this problem calculates Pearson's r for each strip and discards all the strips with $|r|$ significantly lower than the average $|r|$.

The problem of similar functions. Another problem that the algorithm has to cope with is the problem of distinguishing between similar functions. For example, when the width of discretization interval is 0.1 and $|x| < 0.75$, functions

x and $\sin(x)$ are indistinguishable. One of possible solutions of this problem is to use non-equidistant discretization, which is, however, difficult to perform. A different solution is to introduce the cost for each function used. This way, the program is given background knowledge of which functions are expected and which are less likely to occur. The cost of the function is subtracted from the absolute value of the correlation coefficient when comparing different candidates for function ϕ. Even more complex background knowledge can be given by forbidding or penalizing the function within certain contexts. For example, when observing some physical phenomena, we shall allow functions sin and ln but strongly penalize combinations $\sin + \ln$ and $\sin \cdot \ln$.

The third and the safest way to deal with similar functions is to involve an expert which intervenes when the algorithm has to decide between functions with a similar correlation coefficient.

2.3 Complexity of the Algorithm

A single step of decomposition consists of discretizing the attribute (time complexity is $O(N)$), sorting the rule table ($O(N \log N)$), deriving the incompatibility graph ($O(Nk)$), coloring it ($O(k^2)$) and interpreting the coloring ($O(Ns^2)$), where N is the number of examples. k is the number of combinations of discretized bound variables values, which is at most equal to the product of number of discretization intervals. s is the number of basic functions. To select a partition when decomposing a function of l variables, $l(l-1)/2$ possible partitions must be considered, which gives the complexity of

$$O\left(l^2(N + N\log N + Nk + k^2 + Ns^2)\right)$$

Since the decomposition algorithm induces a binary tree, the step above must be repeated $l-1$ times, therefore the total complexity is

$$\begin{aligned} &O\left(\sum_{i=2}^{i=l} i^2(N + N\log N + Nk + Ns^2 + k^2)\right) \\ &= O\left(l^3(N + N\log N + Nk + Nk^2 + Ns^2)\right) \end{aligned}$$

Empirical tests show that sorting is far slower than other operations even for relatively small number of examples, so we can estimate the time complexity as $O\left(l^3 N \log N\right)$.

This result shows the main advantage of this method in comparison with some existing methods of function discovery, such as GoldHorn [3], which performs an exhaustive search over the space of functions it can represent. We can note that GoldHorn's complexity increases exponentially with the depth of function and number of subfunctions but linearly in the number of examples, while our algorithm's complexity is practically independent of the number of basic functions.

3 Experimental Evaluation

The algorithm was tested on several functions specially chosen to explore its advantages and drawbacks. All the functions were within the program's search space, i.e. their hierarchical decomposition did not require any basic functions that were unknown to the program.

$f(x, y) = x + 2y + 3$. This simple linear function is used to roughly measure the number of examples that the algorithm needs to discover the correct form of the function and derive accurate coefficients. The program was run 10 times for each number of randomly chosen examples for the function ($x, y \in [0, 10]$). The results are shown in Table 1.

#examples	correct ϕ	$\overline{a}$	σa
20	2	2.041	0.164
25	3	1.862	0.160
30	6	1.980	0.132
40	7	2.024	0.064
50	10	2.032	0.044
100	10	2.049	0.032
500	10	2.053	0.016
1000	10	2.000	0.014

Table 1. Decomposing function $f(x, y) = x + 2y + 3$: Accuracy and correctness of the form of the discovered function

$f(x, y, z) = 2.5xy + 0.5z$. This function illustrates some difficulties due to equidistant discretization and granulation. The program is expected to decompose it as shown in Figure 2.

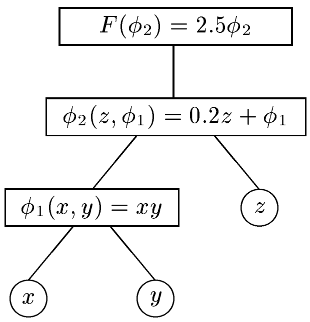

Fig. 2. Function $f(x, y, z) = 2.5xy + 0.5z$: The decomposition tree

If $x, y, z \in [0, 10]$, the gradient of $\phi_1 = xy$ is greater for larger than for smaller x and y. Since granulation is the same over the whole area, the area with the low function's gradient is colored as a wide strip (the left-bottom strip on the Figure 3). Its shape clearly differs from other areas. It cannot be used to determine the slope of the (linearized) functions so the program, after comparing its r with the average, chooses to ignore it.

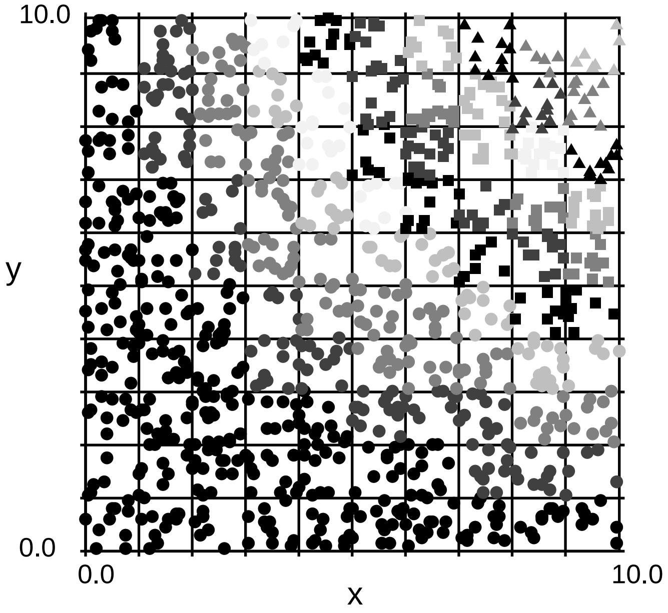

Fig. 3. Function $f(x, y, z) = 2.5xy + 0.5z$: Contour strips of the first step of the decomposition when decomposing by $B = \{x, y\}$.

On the other hand, other strips do not adequately represent the goal function and the Table 2 shows that there are other functions with almost equal r_g coefficient. The reason for high ranking of functions of type $\ln + \mathrm{Id}$ and $\mathrm{Id} + \mathrm{Id}$ is that the most representative contour strips are merged in a single strip and ignored, while the rest of strips are already close to linear without any transformation.

After the first step of the decomposition is made, ϕ_1 is introduced that directly depends on x and y and has values between 0 and 100, so $2.5\phi_1 \in [0, 250]$. The other remaining variable z is between 0 and 10, $0.5z \in [0, 5]$, hence it is negligible in comparison with ϕ_1. If both variables are discretized using the same number of intervals, the algorithm discovers functions like $0.001z + \phi_1$ and the measure of quality r_g is very low (< 0.07). If we (manually) increase the number of intervals for z, the algorithm detects its role in the function and chooses the correct type of intermediate function (see Table 3).

function	r_g
$x^{1.00}y$	0.9505
$5.09\ln(x) + y$	0.9368
$0.19x + \ln(y)$	0.9276
$1.00x + y$	0.8918
$\ln(x)^{5.92}e^y$	0.8348
$\vdots$	

Table 2. Function $f(x, y, z) = 2.5xy + 0.5z$: Candidates for intermediate function ϕ_1 when decomposing by $B = \{x, y\}$

function	r_g
$0.19z + \phi_1$	0.8513
$0.54\ln(z) + \phi_1$	0.7254
$z^{0.54}e^{\phi_1}$	0.7254
$\vdots$	

Table 3. Function $f(x, y, z) = 2.5xy + 0.5z$: Candidates for intermediate function ϕ_2 when decomposing by $B = \{xy, z\}$

$\boldsymbol{f(x, y) = xy}$**.** The ordinary product xy with $x, y \in [0, 1]$ is an example of a function where discretization almost causes the wrong decomposition due to the similarity of functions, as shown in Table 4.

function	r_g
xy	0.9712
$\sin^{1.08}(x)y$	0.9702
$x^{0.91}\sin(y)$	0.9701
$\sin(x)^{0.99}\sin(y)$	0.9688
$\vdots$	

Table 4. Function $f(x, y) = xy$: Candidates for intermediate function ϕ.

The quantitative error of wrong decision would be small since $\sin(x) \approx x$ but an expert may be unable to interpret the resulting decomposition.

$\boldsymbol{f(x,y,w,z) = x + \sin(y + \ln(wz))}$. The program expresses the discovered function in a variety of different ways, which can presumably help an expert to interpret the meaning of derived functions.

Function $f(x,y,w,z) = x + \sin(y + \ln(wz))$ can be rewritten as $f(x,y,w,z) = x + \sin(y + \ln w + \ln z))$. In the first step, some of the best ranking candidates for ϕ_1 are as shown in Table 5. Besides functions $\phi_1(w,z) = wz$, $\phi_1(y,w) = y + \ln(w)$ and $\phi_1(y,z) = y + \ln(z)$, the program also proposes $\phi_1(y,z) = e^y z$ and $\phi_1(y,w) = e^y w$. These can be used later in $\phi_2(y,w,z) = \ln(\phi_1(y,z)) + \ln(w)$ or $\phi_2(y,w,z) = \ln(\phi_1(y,w)) + \ln(z)$, respectively, or even in $\phi_2(y,w,z) = w\phi_1(y,z)$ or $\phi_2(y,w,z) = z\phi_1(y,w)$ to write the function as $f(x,y,w,z) = x + \sin(\ln(wye^y))$. The last form is, however, not in algorithm's search space if the set of basic functions does not contain the composed function $\sin \circ \ln$.

function	r_g
$w^{0.97}z$	0.7901
$(0.99y + \ln(z))$	0.7630
$e^{0.99y}z$	0.7630
$0.99y + \ln(w)$	0.6640
$e^{0.99y}w$	0.6640
$\vdots$	

Table 5. Function $f(x,y,w,z) = x + \sin(y + \ln(wz))$: Candidates for an intermediate function ϕ when decomposing by $B = \{x, y\}$.

$\boldsymbol{f(x,y,w,z) = x + \ln(y + \ln(w + z))}$. This experiment shows the algorithm's ability to decompose complex nested functions. It also proves that the number of graph's colors and r_g are not necessarily correlated and that the latter is much more accurate criterion for selecting the appropriate partition. Table 6 lists all possible partitions, number of colors, the best intermediate functions, and their r_g for the first step of decomposition.

Among three possible partitions for the next step, the algorithm again chooses the right one, $A = \{x\}, B = \{y, \phi_1\}$ and $\phi_2 = y + \ln(\phi_1)$, as shown in Table 7.

In the last step, the only possible partition is $A = \{\}$, $B = \{x, \phi_2\}$ and the program correctly interprets the colored graph as $\phi_3 = 0.98x + \ln(\phi_2)$. Thus, the discovered function is $0.98x + \ln(1.09y + \ln(1.02w + z))$.

$\boldsymbol{f(x,y) = \sin(x + y)}$. For $x, y \in [0, 7]$, this function is non-injective and the program is unable to decompose it, as shown in Table 8. The reason is in repeating colors of contour strips, as already explained and shown on Figure 4.

bound variables	# colors	function	r_g
w, z	19	$1.02w + z$	0.8673
y, w	14	$2.27y + \ln(w)$	0.7924
y, z	13	$11.17y + z$	0.7794
x, z	19	$32.09x + z$	0.7127
x, w	11	$28.63x + w$	0.6935
x, y	6	$1.13e^x + \sin(y)$	0.5134

Table 6. Function $f(x, y, w, z) = x + \ln(y + \ln(w + z))$: Possible partitions, number of colors, the best intermediate function and its linearity for the first step of decomposition.

bound variables	# colors	function	r_g
y, ϕ_1	26	$1.09y + \ln(\phi_1)$	0.9317
x, ϕ_1	19	$2.72x + \ln(\phi_1)$	0.9030
x, y	17	$2.22x + \sin(y)$	0.8898

Table 7. Function $f(x, y, w, z) = x + \ln(y + \ln(w + z))$: The second step of decomposition with $\phi_1 = w + z$.

4 Conclusion

The experiments presented in this paper indicate that the proposed method is able to correctly decompose relatively complex functions and can be successfully used to discover a symbolic representation of a tabulated function. On the other hand, the accuracy of constants appearing in the symbolic representation is low due to the discrete nature of the method. However, as described in [3], the accuracy can be further improved by the simplex method. Since discretization and granulation also cause other difficulties, like indistinguishable similar functions and ignoring of variables with small impacts on the value of the function, future

function	r_g
$0.00e^x + \sin(y)$	0.2869
$0.26\ln(x) + \cos(y)$	0.2674
$-3.52\cos(x) + \ln(y)$	0.2656
$2139.49\sin(x) + e^y$	0.2455
$0.05x + \cos(y)$	0.2389
$\vdots$	

Table 8. Function $f(x, y) = \sin(x + y)$: Candidates for an intermediate function ϕ.

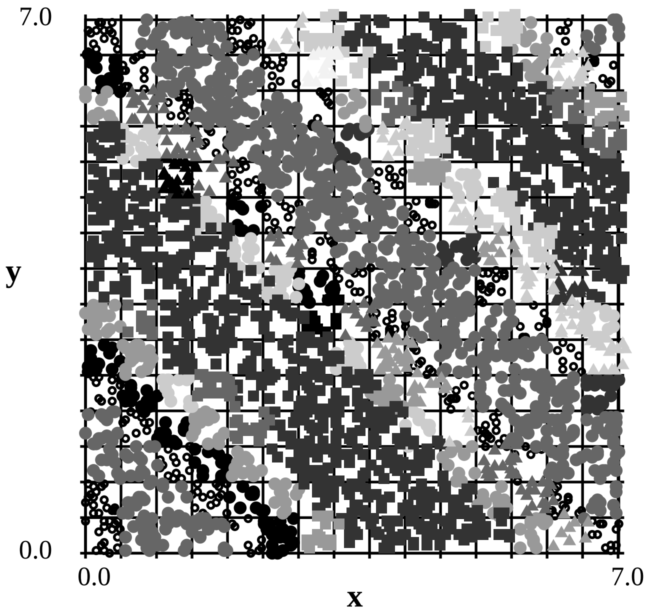

Fig. 4. Function $f(x, y, z) = f(x, y) = \sin(x + y)$: Contour strips.

research should address the design of method that avoids the use of discrete structures.

An interesting question is what happens if the algorithm misses the right intermediate function, for example, if it chooses $\phi(x, y) = x\sin(y)$ or $\phi(x, y) = x + y$ instead of $\phi(x, y) = xy$. If the wrong decision is made because of the similarity of functions in the definition area (like Id and sin), the mistake should not affect the upper layers of decomposition tree. When the wrong decision is a consequence of discarding some contour strips, our decomposition can go astray completely. The algorithm should therefore be improved not to rely on local decisions but to perform a beam search, where several candidate intermediate functions are chosen and later discarded if they show to be unusable.

Classical coloring algorithms are appropriate for graphs with vertices that correspond to nominal values. If they are used on ordinal values, they obviously ignore the information about the position of vertices in the space of bound attributes. In our case, ignoring the location of vertices may cause the discrepancy between coloring and the next phase of the process, the interpretation of colors. Classical graph coloring heuristics that try to minimize the number of colors used are suitable for the decomposition of nominal functions, where the number of colors influences the cardinality of the intermediate function and the criterion, used to choose the partition, normally chooses the partition with less colors. For decomposition of ordinal and real functions, the functions and partitions are evaluated using quite a different criterion, the r_g coefficient, which in some cases even encourages non-optimal colorings. Hence, the standard coloring method should be replaced by an alternative method that tries to make the areas of same color continuous and linear, thus optimizing r_g rather than the number of colors.

Non-injective functions produce an incompatibility graph in which, after it is linearized, the strips of the same color are repeated in a pattern that depends on the type of a function. The problem of decomposing such function is not solved yet.

Another unsolved problem is the decomposition of functions with more than one occurrence of the same variable, for example $f(x, y) = x + \sin(x + y)$. The method presented in this paper fails to give any meaningful result. However, the methods to support such decompositions do exist for Boolean and multi-valued functions [5, 7]. We are working on extension of these methods to handle real-valued functions as well.

The most important problem that is yet to be solved is the problem of coefficients k_1 and k_2 in (1) when they are to appear in non-linear functions such as $\sin(k_1(ax + y) + k_2)$. We are currently investigating a promising method that decomposes such functions by using splits with one bound variable.

In comparison with some existing methods for function discovery, for example GoldHorn [3], we can conclude that our method is able to reconstruct relatively complex functions but with low accuracy of coefficients, while GoldHorn offers high accuracy on functions of limited complexity. The time complexity of our method is low, since all the slow phases (like graph coloring) can be replaced by faster, yet efficient heuristic algorithms. GoldHorn performs an exhaustive search of all possible functions to a given depth. This grows exponentially with the depth and the number of basic functions (background knowledge). On the other hand, our functional decomposition uses a "divide and conquer" approach which significantly improves the efficiency. Also it should be noted that the complexity in our case is relatively low. The time complexity of the algorithm is cubic in the number of attributes, at most $O(N \ln N)$ in the number of examples, and practicallly independent of the number of basic functions (size of background knowledge).

References

1. R. L. Ashenhurst (1952): *The Decomposition of Switching Functions*, Technical report, Bell Laboratories BL-1(11), 541-602
2. H. A. Curtis (1962): *Design of Switching Circuits*, D. Van Nostrand Company
3. V. Križman (1993): *Noise handling in dynamic system modelling* Master thesis (in Slovene), University Ljubljana, Faculty of Computer and Information Science
4. T. Luba (1995): Decomposition of Multiple-valued Functions, *25th Intl. Symposium on Multiple-valued Logic*, 256-261, Bloomington, Indiana.
5. M. A. Perkowski *et al.* (1996): *Unified Approach to Functional Decomposition of Switching Functions*, Unpublished technical report, Wright Laboratory WL/AART-2, Ohio

6. T. D. Ross *et al.* (1994): On the Decomposition of Real-valued Functions, *3rd International Workshop of Post-Binary VLSI Systems*
7. B. Zupan, M. Bohanec (1996): *Learning Concept Hierarchies from Examples by Function Decomposition*, Technical Report, J. Stefan Institute, URL `ftp://ftp-e8.ijs.si/pub/reports/IJSDP-7455.ps`

This article was processed using the LaTeX macro package with LLNCS style