DISSERTATION APPROVAL

The abstract and dissertation of Stanislaw Grygiel for the Doctor of Philosophy in Electrical and Computer Engineering were presented November 19, 1999, and accepted by the dissertation committee and the doctoral program.

COMMITTEE APPROVALS:	Marek Perkowski, Chair
	George Lendaris
	Michael Driscoll
	Jack Riley
	Martin Zwick
	Marek Elzanowski Representative of the Office of Graduate Studies

ABSTRACT

An abstract of the dissertation of Stanislaw Grygiel for the Doctor of Philosophy in Electrical and Computer Engineering presented November 19, 1999.

Title: Decomposition of Relations as a new Approach to Constructive Induction in Machine Learning and Data Mining

Machine learning usually refers to either making enhancements to an existing, or a synthesis of a new system for performing tasks like prediction, recognition, diagnosis, planning, robot control, etc. Data mining refers usually to machine learning methods designed specifically for information retrieval from large data sets.

The quality of a system created to perform a specific task is usually scored based on its predictive accuracy. In some domains however, predictive accuracy is not the only measure of interest for the users. What is often more important is interpretability of rules encoded in the system. The interpretation of results is especially important in applications where not only a correct decision is desired but also an explanation underlying that decision. That is why, for some applications, systems which provide human understandable results like decision diagrams are considered to be more acceptable than black box classifier models like neural networks.

The method proposed in this dissertation is based on **decomposition** as a way of simplification of the description of data by extracting relationships and concepts hidden in the data. The method follows the general **Occam Razor principle**, which advises a user to select the simplest of available descriptions of data in order to obtain better generalization properties. The decomposition type is the well known (from the logic synthesis area) Ashenhurst-Curtis [29] simple serial decomposition, but its application to **decomposition of relations** and **constructive induction** in Machine Learning and Data Mining is novel.

The decomposition method uses a new **data representation** specially designed for effective storage and decomposition of large **multiple-valued, incompletely specified relations**. The new representation may be particularly advantageous for decomposition and data mining of **distributed data bases**. Within the framework of the decomposition process, methods for effective **data reduction** (removing vacuous variables) and dealing with unknown values are developed to further simplify data description and speed up the decomposition process. Methods for dealing with **incomplete** and **noisy** data are proposed in the same framework.

DECOMPOSITION OF RELATIONS AS A NEW APPROACH TO CONSTRUCTIVE INDUCTION IN MACHINE LEARNING AND DATA MINING

by

STANISLAW GRYGIEL

A dissertation submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
2000

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

The data collected in many research, business, medical and manufacturing domains contain valuable information. Since the volume of the data is often large (the number of factors (features) it depends on may be on the order of 10^2 or even 10^3 in some medical applications and the number of records on the order of 10^9) the problem of automatic data analysis and information retrieval becomes more important than ever.

Knowledge Discovery in Databases (KDD) and **Data Mining** is the research area focused on problems of automating data analysis and information retrieval from large data sets. **Distributed Data Mining** deals with the problem of finding data patterns in an environment with distributed data and computations. While KDD usually refers to the overall process of data analysis and information retrieval, including selection, preprocessing and transformation of data, data mining refers to the process of applying specific algorithms to extract useful information from the data. A data mining algorithm is any algorithm that fits theories to data or enumerates patterns from data [38].

Learning general concepts from a limited number of instances of data is the domain of inductive inference. It consists of generating hypotheses, fitting them to the data and selecting the one(s) which best describe the data. To represent hypotheses, a learning system can use a variety of models, such as linear discriminator, nearest neighbor classifier, decision tree, neural network, etc. The learning process consists of adjusting model parameters according to certain performance criteria. There is no universally best model for all possible problems. For instance, Bayes' classifiers seem to provide good results in medical domains [60], neural networks are likely to perform well in parallel domains, while decision diagrams are better when dealing with sequential domain problems [101]. Selection of the model depends on the type of the problem and constitutes the learning bias.

1.1 Learning Bias

Learning processes are always biased. One teacher suits our needs better than the other. One person prefers studying in a library, the other at home. Night is the best study time for one individual and morning for the other. We are all biased by our customs, culture, friends, environment. Learning systems are biased too. We make assumptions and choices when building learning systems, implementing them and adjusting their parameters. These are all biases. Learning biases can be divided into two major categories: model selection bias and hypothesis selection bias [31].

1.1.1 Model Selection Bias

To build a classifier, a model for it has to be selected. The choice depends on the problem the classifier will be used for, and includes, but is not limited to, factors like simplicity of implementation, speed, accuracy, and personal preferences of the designer. The list of possible choices includes the following:

- **Linear Discriminator** uses hyper-plane decision boundaries to classify data. An example of a linear discriminator is the McCulloch-Pitt neuron model.
- **Nearest Neighbor Classifiers** assign the data sample to the class the closest training data sample belongs to.
- **Decision Diagrams** partition data samples into a set of covering decision rules [100]. An example application is Quinlan's C4.5 [99]
- **Neural Networks** use a network of artificial neurons to store information contained in data. The information is transferred in a learning process which

assigns weight values to the network interconnections. An example is a back-propagation neural network, the most often used type of neural network.

- **Feature Vectors**, a table of feature vectors or a lookup table, the most natural representation of data. Feature vector consists of a set of values, each value corresponds to a different variable (feature).
- **First Order Logic** as hypotheses language. First-order predicate calculus is used to store a set of hypotheses (rules, equations) describing the data.
- **Genetic Algorithms** [46] use ideas borrowed from genetic reproduction to generate populations of hypotheses. The best hypotheses from the current population are selected to create a new, more promising, population. This evolutionary process of selection continues until a satisfactory hypothesis is found.

1.1.2 Hypothesis Selection Bias

Within a framework of a given classifier model (or methodology) many hypotheses or theories may be formulated. The choice of the best one is often a difficult task. To facilitate this task, selection criteria have to be determined which are simple and robust enough to make hypothesis selection an efficient process. The list of possible choices includes but is not limited to the following:

Principle of Multiple Explanations was formulated by Epicur (341-270BC) and it states the following: *If more than one hypothesis (theory) is consistent with the data, keep them all.* The contemporary name for principle of multiple explanations is **Principle of Indifference**.

Occam's razor principle was formulated by William of Ockham (1299-1350) and states that: *Among theories which are all consistent with the observed phenomena (data) one should use the simplest one until more evidence comes along.*

Bayesian learning Thomas Bayes (1702-1761) takes a probabilistic view of Nature and formulates what we know today as **Bayes' rule**:

$$P(H_i|D_j) = \frac{P(D_j|H_i)P(H_i)}{P(D_j)}$$

where:

$P(H_i)$	is *a priori* probability of hypothesis H_i (learner's initial belief)
$P(D_j\|H_i)$	is *a priori* probability of data vector D_j given hypothesis H_i
$P(H_i\|D_j)$	is *inferred* probability of hypothesis H_i given data vector D_j
$P(D_j)$	is the probability of data vector D_j

In essence, Bayes' rule is a mapping from *a priori* probability $P(H)$ to *a posteriori* probability $P(H|D)$ determined by data D. It basically defines a way of updating the probability of hypothesis H, as more and more data D comes along. The main problem with Bayes' rule is that an *a priori* probability for hypothesis $P(H)$ may be not known, and can be difficult, or even impossible to calculate or estimate.

Maximum Likelihood principle The Maximum Likelihood principle states that a reasonable hypothesis for explaining a particular set of data is the one which is most likely to have provided the data. In other words, it is the hypothesis H which maximizes $P(D|H)$. The maximum likelihood principle is based on Bayes' rule. Bayes' rule leads us to discovering the rule H which maximizes $P(H|D)$. It is equivalent to finding the hypothesis H maximizing $P(D|H)P(H)$. Since $P(H)$, the *a priori* probability of hypothesis H, is difficult to determine, instead of maximizing $P(D|H)P(H)$, as it is in the Bayes' rule, we maximize **likelihood** $P(D|H)$ and assume that it also maximizes $P(H|D)$. The maximum likelihood ratio $P(D|H)/P(D)$ is related to the Shannon-Wiener measure of information and to the well known (from statistics) χ^2 (chi-square) distribution. It was shown [86] that the maximum likelihood estimate $L^2 = -2n \sum P(D) \log_e P(D|H)/P(D)$ approximates χ^2 asymptotically. Hence, given sample size n, number of degrees of freedom of hypothesis H, and standard χ^2 tables, the significance level (probability) that hypothesis H reflects sampling biases rather than true hypothesis can be obtained.

Maximum Entropy principle Shannon's entropy, or uncertainty [112]:

$$U(p_1,\ldots,p_n) = -\sum_i p_i \log p_i$$

where $p_i = P(H_i)$ is a probability of hypothesis H_i, can be used to determine the set of *a priori* probabilities $P(H_i)$ which maximize $U(p_1,\ldots,p_n)$ subject to $\sum_i p_i = 1$ and constraints of our prior knowledge of the problem. The hypothesis H_i with the highest probability $P(H_i)$ is the one to be chosen. The entropy principle has direct application in cases when the data D is not available, Bayes' theorem can not be used and the decision on which hypothesis to chose has to be made directly from the *a priori* probabilities $P(H_i)$.

Kolmogorov complexity A different point of view is based on the concept of Kolmogorov complexity. Kolmogorov complexity, or algorithmic entropy, is the basic concept of algorithmic information theory and was developed independently by Solomonoff [88],[116],[115], Kolmogorov and Martin-Lof [57],[58], [59], [77],[78],[79], and Chaitin [21],[22],[23],[24], [25]. The Kolmogorov complexity of an object can be defined as the *length of the shortest computer program used for the description of that object.* It is an attempt to address a problem of determination of absolute information content of individual objects rather than average information content of objects generated by a random source with underlying probability distribution (entropy or uncertainty).

Let us consider for instance a set of n binary strings of length k, $S = \{s_i\}$, $|S| = n$. The Shannon's uncertainty (entropy) of the set S is equal to $-\sum_{i=1}^{n} p_i \log pi$, where p_i is a probability of drawing s_i. In the absence of any other information on the probability distribution p, we apply the principle of indifference and assume all strings to be equally probable $p_i = 1/n$. In this case, Shannon's uncertainty will be equal to the amount of information (in bits) required to count the strings, or equivalently, to the amount of information required to communicate each individual string.

Strings containing regular patterns however, as for example a string consisting of 1 repeated k times, don't need as many bits to be communicated. Random strings, the ones which can not be compressed, require more information to be communicated. Kolmogorov complexity $K(x)$ addresses the problem of expressing properties of individual objects x, their individual entropies. There exist an interesting relation between these individual entropies (Kolmogorov complexities) and Shannon's entropy: the expected value of Kolmogorov complexity over a set of objects is equal to Shannon's entropy of the set.

Unfortunately, Kolmogorov complexity $K(x)$ is not computable in general [74] and has to be replaced by computable approximations in practical applications.

Minimum Description Length principle Another method of selecting the best hypothesis is the Minimum Description Length (MDL) principle developed by Rissanen [106]. The best theory (hypothesis) to explain a set of data is the one which minimizes the sum of:

- the length, in bits, of the description of the theory,
- the length, in bits, of the description of data given the theory.

The above formulation can be directly derived from Bayes' rule . Taking the negative logarithm of both sides of Bayes' formula, we obtain:

$$-\log P(H|D) = -\log P(D|H) - \log P(H) + \log P(D)$$

Selecting the best hypothesis corresponds to maximizing $P(H|D)$, or correspondingly, minimizing $-\log P(H|D)$. Since $\log P(D)$ is constant, then minimizing $-\log P(H|D)$ corresponds to minimizing $-\log P(D|H) - \log P(H)$ which, as is shown in [74], corresponds to minimizing $K(D|H) + K(H)$ where $K(\cdot)$ is Kolmogorov complexity.

The second part of the sum refers to the length of the hypothesis describing the available data. Usually, the more accurate the hypothesis is (fewer data mismatches), the longer its length (it describes data in more details). Fewer data mismatches means the first part of the sum requires fewer bits to be encoded, and vice versa.

PAC learning The model of PAC (Probably Approximately Correct) learning was introduced by Valiant [124] and, roughly speaking, specifies the meaning of *well* or *good* when evaluating learned concepts and learning algorithms used to learn them. Let us assume that H_T is the (true) hypothesis we try to learn and H is the hypothesis we have learned. Then the error of hypothesis H is defined as:

$$error(H) = Probability(H(x) \neq H_T(x) \; for\ a\ data\ sample\ x)$$

We can say that a learned hypothesis H is *good* if:

$$Probability(error(H) > \varepsilon) < \delta$$

where: $0 < \varepsilon < 1$ and $0 < \delta < 1$.

Different degrees of *goodness* will correspond to different values of ε and δ. The smaller ε and δ are, the better (closer to H_T) the learned hypothesis H will be. A class $\mathcal{C}$ of hypotheses is **Efficiently PAC Learnable** if the algorithm to learn hypothesis is polynomial in ε, δ and $\ln N$, where N is the cardinality of $\mathcal{C}$ (number of hypotheses). A class $\mathcal{C}$ is **Polynomial PAC Learnable** if m, cardinality of the training set required for given ε and δ, is polynomial in ε, δ and the size of minimal descriptions of the hypothesis and the data.

Chapter 2

OVERVIEW OF THE DISSERTATION

In Chapter 1 we briefly introduced a learning process in the context of machine learning and data mining. We stated that every learning process is, roughly speaking, doubly biased by model and hypothesis selection biases. These two biases correspond to two major components of any learning process: memorization and inference. The model selected for a classifier is responsible for memorizing the information inferred from data. The hypothesis selection process is a process of extracting useful knowledge from data. It is an inference process. In his inferential theory of learning R. Michalski proposed the following equation:

$$memory + inference = learning$$

The general organization of this dissertation reflects this simple scheme:

$$representation + decomposition = learning$$

Chapter 3 describes a new data structure, lr-partitions, used for storing (memorizing) data. Chapter 4 is devoted to decomposition as a constructive induction approach to inferencing useful information from data. The next chapter, Chapter 5, closes the learning equation. It presents both decomposition and lr-partitions in action, learning from various data sets. The last chapter, Chapter 6 concludes this dissertation and proposes future work.

The description presented in the following sections is not intended to provide all the details of the respective algorithms and methods, it is an overview. Missing details can be found in corresponding chapters of the dissertation, and the reader is encouraged to consult them as needed.

2.1 Representation

The representation developed in this dissertation consists of a set of lr-partitions, where each lr-partition is based on a set of variables (attributes) and consists of a set of blocks. Figure 2.1 shows the process of creating of lr-partitions (Figure 2.1*b*) from a set of data tuples (Figure 2.1*a*). The process begins with assigning a unique key to each tuple (row label in Figure 2.1*a*). Each lr-partition is defined by a pair of sets:

- a set of variables (subset of column labels in Figure 2.1*a*)
- a set of combinations of values these variables take in the data tuples

In the example in Figure 2.1 we have four lr-partitions $P(x_1)$, $P(x_2)$, $P(y_1)$, and $P(y_2)$. They are defined by the following pairs of sets: $\{\{x_1\},\{0,1,2\}\}$, $\{\{x_2\},\{0,1\}\}$, $\{\{y_1\},\{0,1,2\}\}$, and $\{\{y_2\},\{0,2\}\}$. Each value from the second set of the pair is used to specify a different lr-partition block; a block label. The lower part of each corresponding block contains the Key Designator (row labels in 2.1*a*) for those tuples (all the data in each row) which contain **values** equal to the block label.

An lr-partition block is a set of keys corresponding to tuples in which set of variables takes values equal to the block label.

The primary reason for developing this new data representation was its potential for **more compact than tuples** storage of information. Lr-partition blocks (designated via sets of keys) can be implemented using any representation of a set of integer numbers. We experimented with two such representations: binary decision diagrams (BDD) and bit-sets (BS) and compared them to the most often used representation of binary functions binary decision diagrams.

The secondary reason is flexibility of lr-partitions when used for data decomposition, the procedure we use in this dissertation for developing quality classifiers for machine learning and data mining. The sets of variables for lr-partitions can be selected in an arbitrary way, but, if any extra information is available on how

key/variable	x_1	x_2	y_1	y_2
a	0,2	1	0,1,2	2
b	0,1	0	0,2	0
c	2	0	1,2	0
d	1	1	1,2	2

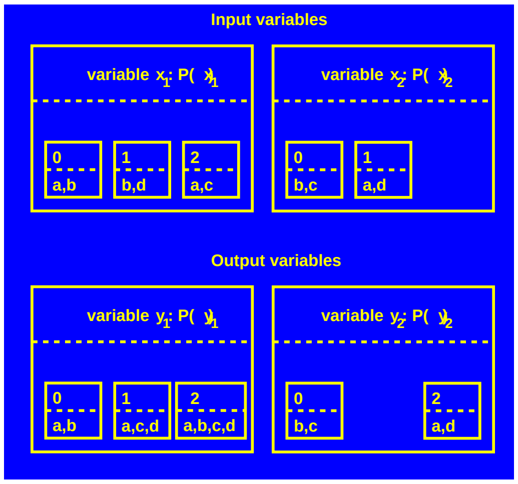

Input variables
variable x_1: P(x_1)
0 a,b
1 b,d
2 a,c
variable x_2: P(x_2)
0 b,c
1 a,d
Output variables
variable y_1: P(y_1)
0 a,b
1 a,c,d
2 a,b,c,d
variable y_2: P(y_2)
0 b,c
2 a,d

Figure 2.1: Lr-partitions.

set of tuples

key/variable	x_1	x_2	x_3	x_4	y_1
a	0,2	1	0,2	2	0,1
b	0,1	0	0,1,2	0	2
c	2	0	1,2	0,1	0,2
d	1	1	1,2	1,2	0

site A

key/variable	x_1	x_2	y_1
a	0,2	1	0,1
b	0,1	0	2
c	2	0	0,2
d	1	1	0

site B

key/variable	x_3
b	0,1,2
a	0,2
d	1,2
c	1,2

site C

key/variable	x_4	y_1
d	1,2	0
b	0	2
a	2	0,1
c	0,1	0,2

Figure 2.2: Set of tuples and its storage in a distributed data base with vertically partitioned data space.

these variables are related, it can be used to create lr-partitions which facilitate decomposition and, as a consequence, lead to better classifiers. A direct example of how this external information can be used is in creating lr-partitions for distributed data bases. In Figure 2.2, data tuples are assumed distributed among three different sites. Such a split is usually based on some relationship between data variables and each site can be directly mapped into separate lr-partition.

2.2 Decomposition

The inference method used in this dissertation is decomposition. Using decomposition, a complex structure is replaced with a set of interrelated simpler structures that can be more easily comprehended (Figure 2.3). They correspond to concepts which, when extracted from data, can be used to provide higher level abstractions to simplify the description of data. According to the Occam Razor Principle a simpler decomposed structure is likely to have better predictive properties than the original one (see [70][73][72] for more detailed treatment of this topic).

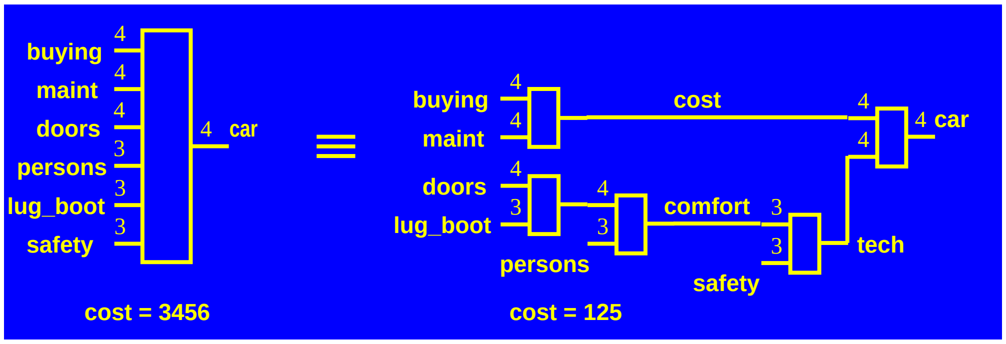


Figure 2.3: Extracting concepts by decomposition.

2.2.1 Decomposition strategy

The decomposition process is iterative. One iteration step consists of decomposing a block into two smaller blocks of lower cost. The decomposition strategy is depicted in Figure 2.4. A block to be decomposed is drawn from a pool of blocks to be decomposed. If its size is smaller than the minimum user specified size, then the block is transfered to the pool of final blocks. Otherwise an attempt is made to extract a smallest possible (user specified) block from it. If such a decomposition exists, the resulting blocks are added to the pool of blocks to be decomposed. Otherwise the minimum size is increased and decomposition attempted again. This process continues until the pool of blocks to be decomposed is empty. As a result of the decomposition we obtain the set of blocks in the pool of final blocks. This strategy is referred to as a bottom-up procedure in this dissertation. An alternative strategy, top-down, is also analyzed in this dissertation and both approaches compared.

The decomposition of any block from the pool of blocks to be decomposed is independent of the decomposition of any other block. This means that the decomposition process at this level can be parallelized, i.e., each block from the pool of blocks to be decomposed can be decomposed on a different processor or computer.

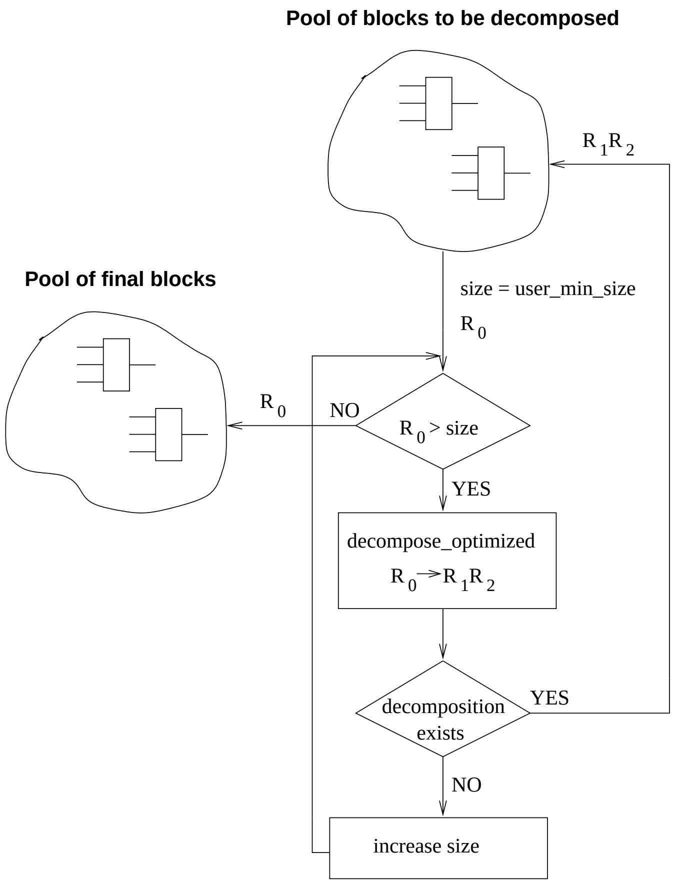
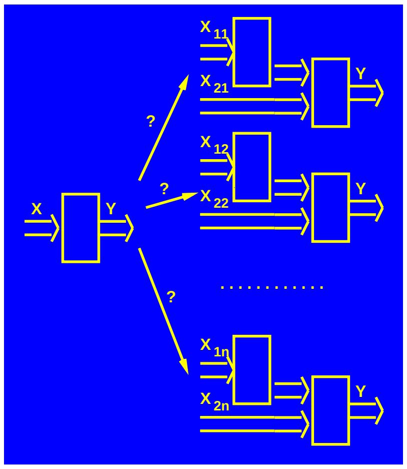


Figure 2.4: Decomposition strategy.

There are many possible decompositions of one block into two smaller blocks and all of them would have to be performed in order to select the optimal solution (see Figure 2.5). The number of all such decompositions is too large to evaluate them all so we only generate a limited number of the most promising solutions. The limited number of solutions is created via generating sets of variables X_1 and X_2. This is referred to as variable partitioning process.



Selection criteria:

- Cardinality cost measure
- Functionality cost measure
- Minimum Description Length (for incomplete data)

Figure 2.5: How to select the best solution?

The process of decomposition of one block into two blocks is in more details shown in Figure 2.6. Given sets $X_1 = \{x_3, x_4\}$ and $X_2 = \{x_1, x_2\}$ the task is to determine the output variables of block f_1 in such a way as to minimize the cost of the solution. This is done by reducing the original problem to the clique covering or graph coloring problem. The number of cliques (colors) in the graph corresponds to the cardinality (number of possible values) of variable y_1. Minimizing this value minimizes the cost of the solution.

The decomposition procedure described above is developed in Section 4.2 (p.74) for non-probabilistic directed relations, and is an essential part of all other decomposition procedures developed in this dissertation. A decomposition procedure for non-probabilistic neutral relations is developed in Section 4.4 (p.86). Two different decomposition procedures for probabilistic relations, are developed in Section 4.3

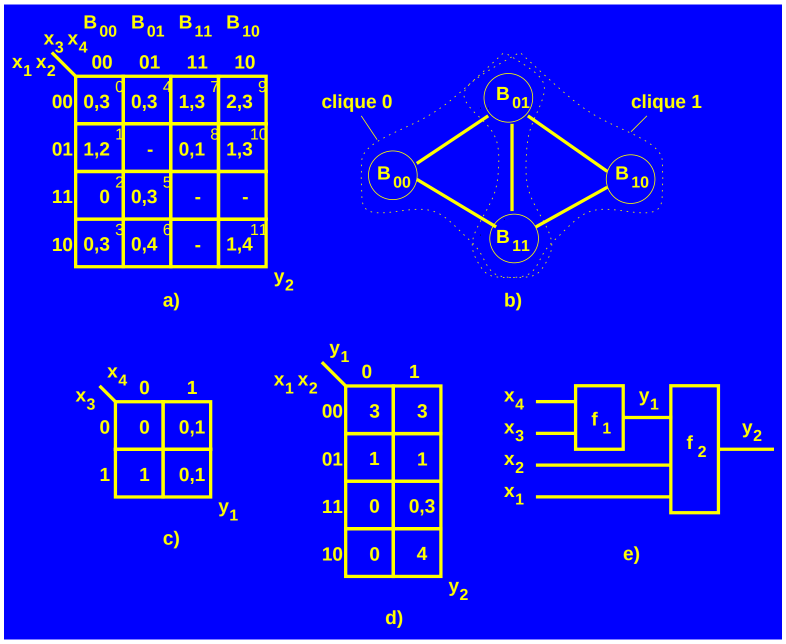


Figure 2.6: Block decomposition.

(p.79).

2.2.2 Cost measures

To evaluate a goodness of the solution at each decomposition step an appropriate cost measure is needed. In this dissertation we propose and compare two such cost measures. First of them, we call it **cardinality cost measure**, is closely related to the maximum number of tuples that may be used for the description of a problem given a set of attributes (variables). For binary functions the following formula can be applied to one step decomposition result (based on Abu-Mostafa):

$$C_c = 2^{|X_1|}|Y_1| + 2^{|X_2|+|Y_1|}|Y_2|$$

We extended this formula onto multiple-valued functions to obtain:

$$C_c = p_{X_1}\log_2 p_{Y_1} + p_{X_2}p_{Y_1}\log_2 p_{Y_2}$$

where:	X_1, X_2, Y_1, Y_2	are sets of variables defined in Figure 2.9
	$\lvert X\rvert$	is a cardinality of set X
	p_X	$= \prod_{x_i \in X} \lvert x_i\rvert$
	$\lvert x_i\rvert$	is a cardinality of variable x_i

The second cost measure, **functionality cost measure**, is equal to the number of functions that can be realized for a given structure. For binary functions and disjoint sets X_1, X_2 the following formula was developed by Lendaris and Stanley:

$$C_f = \frac{1}{2} 2^{p_{X_2}} (2^{p_{X_2}} - 1)(2^{p_{X_1}} - 2) + 2^{p_{X_2}}$$

We extended their formula to the more general case of multiple-valued functions to obtain:

$$C_f = \sum_{i=0}^{p_{Y_1}-1} P(p_{Y_2}^{p_{X_2}}, p_{Y_1} - i) S(p_{X_1}, p_{Y_1} - i)$$

where:	$P(n,r)$	is an r-permutation of n distinct things
	$S(n,m)$	is a Stirling number of the second kind
	p_X	$= \prod_{x_i \in X} \lvert x_i\rvert$
	$\lvert x_i\rvert$	is a cardinality of variable x_i

The formula above applies to disjoint sets X_1 and X_2 but its extension to the non-disjoint case is also derived in Section 4.5.2 (p.93).

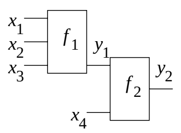
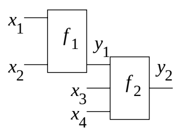
decomposed structure	cardinality C_c	log-functionality $\log_2 C_f$ (C_f)	original data cost
 x_1 x_2 x_3 f_1 y_1 f_2 y_2 x_4	12	10.6 (1528)	16
 x_1 x_2 f_1 y_1 f_2 y_2 x_3 x_4	12	10.7 (1696)	16

Figure 2.7: Cardinality vs. Functionality.

What are advantages and disadvantages of both measures? Both cost measures are in fact very similar. The functionality cost measure, however, provides a finer distinction between structures than the cardinality cost measure does. Figure 2.7 provides a simple example of such a distinction (the functions are binary). On the other hand cardinality cost measure is much simpler from the computational point of view.

2.2.3 Variable Partitioning for Decomposition

As we already stated above, the process of selecting sets X_1 and X_2 is referred to as variable partitioning. The algorithm we developed in this dissertation is based on the ordering of input variables according to their significance to the output variable determination. The most significant variables are selected to be inputs of the block f_2 (see Figure 2.9) due to their direct link to the output of this block. The remaining variables form the set X_1.

Determination of each variable significance is based on the conditional uncertainty calculation, and is depicted in Figure 2.8.

- **Uncertainty:**

$$u(a) = -\sum_i p(a = a_i) \log_2 p(a = a_i)$$

- **Conditional Uncertainty:**

$$u(y|b) = u(yb) - u(b)$$

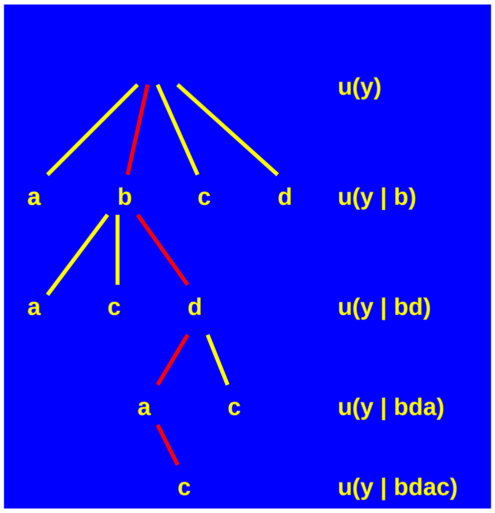


Figure 2.8: Variable partitioning for decomposition.

The conditional uncertainty of every input variable with respect to the output variable is computed and the variable which reduces the uncertainty of the output variable the most is selected as the most significant one. In the following steps, computations are repeated for the remaining input variables until all the variables are ordered. In Figure 2.8 the final variable order is *bdac*.

2.2.4 Data reduction

Real life data often contain many vacuous variables, variables which are not necessary for the description of a given problem. Such variables can be removed from the data without compromising accuracy of the description. Furthermore, their removal usually simplifies the process of creating a good model for the data. Therefore, efficient algorithms to discover such variables are an important part of the induction process.

In this dissertation, a simple but efficient algorithm for discovering vacuous variables is proposed as a part of the decomposition process:

if Y_1 (Figure 2.9) is constant then all the variables $x_i \in X_1$ are vacuous.

If the above condition is satisfied, then the result of one step decomposition is a single block obtained from the original block by removing all the vacuous variables $x_i \in X_1$.

2.2.5 Discretization

The inference method presented in this dissertation requires variable values to be discrete. The continuous variables have to be discretized before the decomposition process can even be started. In this dissertation we describe an interesting new discretization procedure derived within the framework of the decomposition strategy.

In the general decomposition strategy, the user specifies a minimal decomposition block which is the smallest size unit (block f_1 in Figure 2.9) that can be extracted from the block being decomposed. The inference procedure attempts to extract f_1 in such a way as to minimize the cost of the decomposed structure. If the block f_1 is specified to have only one input variable x_1, then its output variable cardinality $|y_1|$ must be smaller than the cardinality of the input variable $|x_1|$ in order for the cost to be decreased. Therefore, block f_1 represents a mapping from a set of values the variable x_1 can take into a smaller set of values the variable y_1 can take, and f_1 defines a new discretization scheme for variable x_1. In order to apply the described procedure, the continuous input variable x_1 has to be initially discretized using any simple discretization method, uniform binning for instance. If the number of bins used is large enough, then the decomposition procedure provides a mean for optimizing both the number and the size of the bins. Moreover, the discretization scheme discovered is directly related to the way the dependent and independent variables relate to each other and, as such, can better fit the data than any other independent discretization procedure.

The discretization procedure is described in more detail in Section 4.6.4 (p.121).

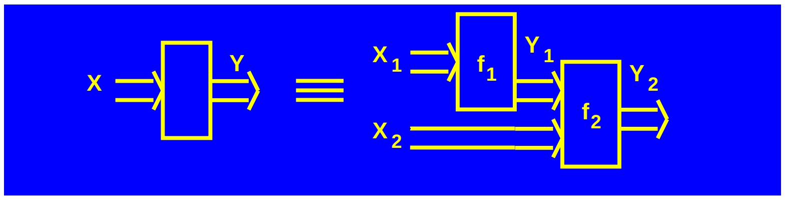
2.3 Learning

The learning method used in this dissertation can be classified as a constructive induction learning method. In other words our method learns from examples. In the learning process concepts are extracted from data forming a new representation space. Then a theory is formed in this new representation space which is as consistent as possible with the set of training examples.

A practical example of a theory discovered via a multi-level constructive induction learning method is presented in Figure 2.3 where the higher level concepts are `cost` and `tech`, and a lower level concept is `comfort`.

2.3.1 Learning from noisy or incomplete data

If the training data are noisy or incomplete, then creating an error-free classifier may lead to an overfitting situation. The classifier will perfectly fit the training



New concept:	Y_1
New representation space:	Y_1, X_2
A theory:	$f_2(Y_1, X_2)$

Figure 2.9: Learning concepts.

data but perform poorly on the data not present in the training set (compare Figure 2.10a and b, where open circle is a new piece of data). A better overall performance of a classifier can usually be achieved when allowing for non zero error rate on the training set (lossy but simpler classifier). A tradeoff between simplicity and error rate may be determined by minimizing the sum of description lengths of the cost of classifier and error rate (so called *minimum description length principle*).

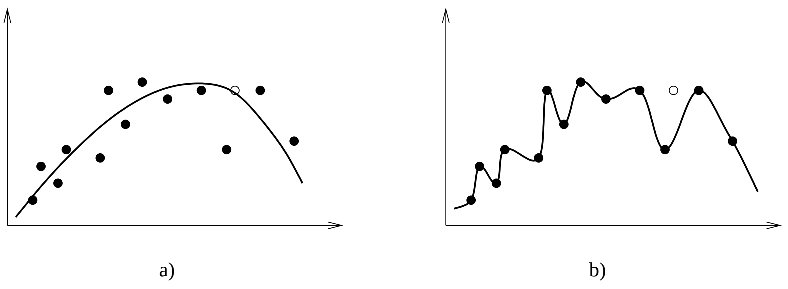


Figure 2.10: Learning from noisy data.

The algorithm for creating such a lossy classifier is embedded into the decomposition procedure when creating a compatibility/incompatibility graph for determination of a set of variables Y_1 (Figures 2.9, 2.6, and 2.11).

The graph in Figure 2.11b is a full graph with edges weighted by the ratio of mismatches for the corresponding table columns. By selecting a threshold value and removing all the edges with weights higher than the threshold value we can

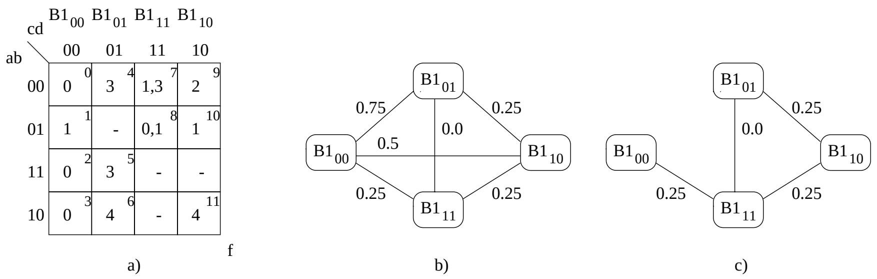


Figure 2.11: Lossy decomposition.

control the cost of the result of decomposition. The graph in Figure 2.11*c* is a result of selecting threshold value equal to 0.5. Lossless decomposition results from the threshold value equal to 0 (see Section 5.2.1 p.149). This algorithm was developed for the very general case of nominal data spaces. If the data variables are metric (a distance measure can be calculated), then more precise algorithms can be developed (see Section 5.2.1 p.149).

Chapter 3

REPRESENTATION

3.1 Introduction

In the last few years, there has been an increasing trend to apply logic representations such as BDDs in image processing, machine learning, knowledge discovery, data-base optimization, AI, image coding, automatic theorem proving and formal verification. Multiple-valued (MV) relations (functions in particular) which include very many don't cares are becoming increasingly important in these new areas of applications [98] as well as in classical logic synthesis and state machine problems. When a state machine is represented as a relation, the present and next states, as well as symbolic i/o variables are represented as multiple-valued variables that have arbitrary number of values. It is very important to have a good representation for such relations. By good representation, we understand one that is compact and allows for fast processing. For instance, success of many *binary* decomposers resulted entirely from appropriate innovative representations of Boolean functions: cube calculus [126], spectral transforms [113], decision diagrams [63, 110], or rough partitions [75] rather than from new algorithmic ideas. The same is true for tautology verifiers. Better representation allows storing larger functions, and also, to carry out efficiently appropriate calculations.

Three essentially different representation methods for multiple-valued functions have been successfully used in logic synthesis programs and AI applications:

Multiple-Valued Cubes (MVC, using positional notation) [120], stores cubes of the table (such as Table 3.1) one by one (row by row), value by value (linearly) and apply *cube calculus* for cube operations.

Multiple-Valued Decision Diagrams (MVDD) [100],[117],[30],[55] stores cubes in a Directed Acyclic Graph (DAG). Each DAG level corresponds to a different variable; the number of node's children is equal to the number of values a variable can take.

Rough Partitions (r-partitions) [75] stores the table (such as Table 3.1) column-wise, and not row-wise as MVC does. In r-partition every variable (a column of a table), both input and output, induces a partition of the set of rows (cubes) to blocks, one block for each value the variable can take (there are two blocks for a binary variable, and k blocks for a k-valued variable).

Cube representation seems to be superior in problems with a limited number of levels, such as sum-of-products (SOP) or exclusive-sum-of-products (ESOP) synthesis. The disadvantage of cube representation is that large multi-level netlists or BDDs may produce too many cubes after flattening, so that their cube arrays can not be stored. Even if the initial data is in a form of large arrays of cubes (as is the case in ML or controller design applications), cubes may be too slow for effective manipulation and alternative representations may considerably improve the processing speed.

Decision Diagram representation seems to be superior for general purpose Boolean function manipulation, simulation, tautology, technology mapping, and verification, but can be exponential for functions of certain classes. For some classes of functions, such as parity, the decision diagram storage requirement is polynomial in terms of the number of variables. For other functions however, as shown in [32] or that occur in ML, logic or controller design [118], it is exponential.

Rough Partitions are an interesting and novel idea but they don't really form a representation of a function. Since the values of a variable are not stored together with partition blocks, the essential information on the function is lost in this representation and the original data can not be recovered from it.

None of the above representations addresses the problem of binary or multiple-valued *strongly unspecified* functions and relations which occur in Machine Learning (ML), Knowledge Discovery in Databases (KDD), Artificial Intelligence (AI), Finite State Machine (FSM) and controller design, and in decompositional approaches to logic synthesis [98]. Although the logic and FSM methodologies that produce a very high percent of don't cares are not very popular yet, there exist practical industrial applications with more than 95% of don't cares (on the average, the percent of don't cares in industrial benchmarks is not greater than 80%) [114]. In contrast, benchmarks with more than 99% of don't cares are common in ML.

With the exception of [75],[44], and [43], the representation problem has not been addressed for *multiple-valued functional and state machine decomposers*, and other synthesis programs. The data structure presented in this chapter is particularly useful for the decomposition of incompletely specified MV *functions, relations, and state machines.*

To briefly introduce the new representation, we will show how to construct lr-partitions from a set of tuples shown in Table 3.1 representing a relation $y_1(x_1, x_2)$ and function $y_2(x_1, x_2)$. In the table, a set of values separated by ',' means that a variable can take any value from that set.

	X		Y	
	x_1	x_2	y_1	y_2
a	0,2	1	0,1,2	2
b	0,1	0	0,2	1
c	2	0	1,2	0
d	1	1	1,2	2

Table 3.1: Multiple-valued, multi-output relation.

The data structure consists of a set of lr-partitions. Each lr-partition corresponds to a set of columns S_c and partitions the data horizontally. Each set of columns S_c is partitioned vertically by blocks. A block corresponds to a set of tuples S_t, each tuple in S_t having the same projection on the set of columns S_c. This projection constitutes the block label. Each tuple in the table is associated with a different symbol (number) and a block corresponding to S_t is a set of symbols

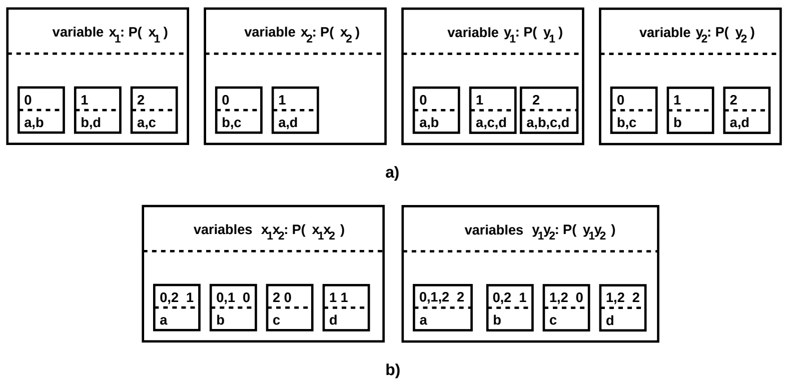

variable x_1: P(x_1)
0 a,b
1 b,d
2 a,c
variable x_2: P(x_2)
0 b,c
1 a,d
variable y_1: P(y_1)
0 a,b
1 a,c,d
2 a,b,c,d
variable y_2: P(y_2)
0 b,c
1 b
2 a,d
a)
variables x_1x_2: P(x_1x_2)
0,2 1 a
0,1 0 b
2 0 c
1 1 d
variables y_1y_2: P(y_1y_2)
0,1,2 2 a
0,2 1 b
1,2 0 c
1,2 2 d
b)

Figure 3.1: a) Single variables b) Sets of variables.

corresponding to the tuples in S_t. Both horizontal and vertical partitions can be non-disjoint, i.e. sets of columns (tuples) corresponding to different lr-partitions (blocks) can overlap.

In Figure 3.1, the small squares correspond to partition blocks, the upper part of each block contains a label and the lower part, set of symbols. Figures 3.1a, b show lr-partitions based on single variables and on sets of variables, respectively.

Due to their vertical-horizontal structure, lr-partitions have a potential for capturing both vertical and horizontal patterns in data. Most of the operations on lr-partitions can be reduced to set operations on the blocks (sets of integer numbers). Hence, by selecting a set representation for blocks we can significantly change the properties of the whole representation! We experiment with two such set representations in this chapter: BDDs and Bit Sets.

The representation presented in this chapter may be particularly advantageous in distributed data and computation environments. For large distributed data bases the sets of variables observed in different sites may in general be different. This is sometimes called a *vertically partitioned data set* and is illustrated in Figure 3.2. Each site stores only a part of the data tuple and a set of lr-partitions can be independently created for every site (this is not possible for cubes or decision diagram representations). Alignment between sub-tuples stored in different sites is in lr-partitions representation made automatically via tuple numbers (keys) that are stored in lr-partition blocks.

set of tuples

key	x_1	x_2	x_3	x_4	y_1
0	0,2	1	0,2	2	0,1
1	0,1	0	0,1,2	0	2
2	2	0	1,2	0,1	0,2
3	1	1	1,2	1,2	0

site A

key	x_1	x_2	y_1
0	0,2	1	0,1
1	0,1	0	2
2	2	0	0,2
3	1	1	0

site B

key	x_3
1	0,1,2
0	0,2
3	1,2
2	1,2

site C

key	x_4	y_1
3	1,2	0
1	0	2
0	2	0,1
2	0,1	0,2

Figure 3.2: Set of tuples and its storage in a distributed data base with vertically partitioned data space.

The chapter is organized as follows. Section 3.2 contains basic definitions of cube calculus and the necessary extensions for manipulating cubes based on different sets of variables. Such cubes are used to represent *block labels*. Sections 3.3 and 3.4 introduce the *labeled rough partitions* (lr-partitions) representation. Section 3.5 presents basic operations on multiple-valued relations represented by lr-partitions and Section 3.6 discusses the corresponding time complexities. Section 3.7 discusses memory requirements for two different representations of partition blocks: BDDs and Bit Sets (BS). Section 3.8 presents experimental results and Section 3.9 concludes the chapter.

3.2 Multiple-valued cubes

Definitions in this section are based on multiple-valued cube calculus definitions ([120] [108] [109] [107]) but extend them on the case of cubes based on different sets of variables and on multiple-valued output relations. The reason for such an extension is that lr-partitions can in general be based on overlapping sets of variables (see Section 3.3) and can represent multiple-valued relations in general.

For the special case of lr-partitions based on disjoint sets of variables, we could use cubes based on the same sets of variables along with the concatenation operation.

Definition 3.1 (literal) *Let x_i be a variable taking values from the set P_i and let S_i be a subset of P_i. Let $\mathbf{U}$ be the universal set. Then $x_i^{S_i}$ is defined as:*

$$x_i^{S_i} = \begin{cases} \emptyset & \textit{if } x_i \notin S_i \\ \mathbf{U} & \textit{if } x_i \in S_i \end{cases}$$

will be called a literal of variable x_i.

The literal $x_i^{S_i}$ can be viewed as a pair $(variable, value) = (x_i, S_i)$ where *value* is a set. In particular $S_i = P_i$, which is a multiple-valued equivalent of a binary don't care (variable x_i can take any value from P_i), and $S_i = \emptyset$ (none of the values can be assigned to the variable).

Using set-values allows for expressing situations where there is an uncertainty on what value a variable should take but the set of acceptable values can be clearly specified. For instance: "the color was red or yellow but not black or white". An example of application in the logic synthesis area is a modulo-3 counter that counts in sequence (non-deterministic state machine is a special case of multi-valued, multi-output, relation) $s0 \rightarrow s1 \rightarrow s2 \rightarrow s0$ and if the state $s3$ happens to be the initial state of the counter, counter should transit to any of the states $s0, s1, s2$, but not to the state $s3$ itself. There exist examples [114] of large controllers from industry specified with non-deterministic transitions (like the modulo-3 counter above), but the current VHDL-based design methodologies do not encourage this kind of design style.

Our representation also allows for representing multiple-valued relations in Machine Learning. For instance, for a given set of attribute values, a tissue cell can be classified as cancerous by one expert and as non-cancerous by another and both experts' opinions need to be taken into account in the data analysis. Examples of multiple-valued relations are benchmarks `hayes`, `flare1`, `flare2` from [121].

Set-values for input variables are already known from cube calculus, but using set-values for output variables is a new concept which allows for representation and manipulation of relations.

Definition 3.2 (cube) *The set of literals $c(X) = \{x_i^{S_i} : x_i \in X\}$ will be called a cube based on the set of variables X.*

Later in this chapter we will denote a cube by $c = x_1^{S_1} x_2^{S_2} \cdots x_n^{S_n}$ and a set of cubes by C. We will use notation $c(X)$ and $C(X) = \{c_i(X)\}$ to specify a cube and a set of cubes based on the set of variables X, respectively.

If $S_i = \emptyset$ then the literal $x_i^{S_i}$ can be used to represent situations where variable x_i is not present in a given cube. An example from ML domain is the well known Michalski's `trains` benchmark which describes a set of 10 trains [85]. A set of attributes corresponds to every car in the train. Since the number of cars varies from train to train, some cubes (trains) contain attributes which correspond to non existing cars and can be represented by literals with $S_i = \emptyset$. Another application is for representing output variables of multi-output functions and relations corresponding to unspecified cubes ('~' in Espresso format).

Definition 3.3 (proper or improper cube) *Cube $c(X)$ will be called improper if there exists $x_i \in X$ such that $S_i = \emptyset$. Otherwise the cube will be called proper.*

For instance cube $x_1^{\{1,3\}} x_2^{\{0,1\}} x_3^{\emptyset} x_4^{\{1\}}$ is improper and cube $x_1^{\{1,3\}} x_2^{\{0,1\}} x_3^{\{1\}} x_4^{\{1\}}$ proper.

Definition 3.4 (minterm) *Cube $c(X)$ will be called minterm if for every $x_i \in X$ cardinality of the set S_i is equal to 1.*

For instance the cube $x_1^{\{3\}} x_2^{\{0\}} x_3^{\{1\}} x_4^{\{1\}}$ is a minterm and cube $x_1^{\{2,3\}} x_2^{\{0\}} x_3^{\{1\}} x_4^{\{1\}}$ is not.

Definition 3.5 (projection) *Let $c(X)$ be a cube and let X' be a subset of X. The cube $c(X')$ created from $c(X)$ by removing all the literals $x_i^{S_i}$ such that $x_i \in X - X'$ will be called a projection of cube $c(X)$ on the set X'.*

Later in this chapter we will use notation $c_i(X_1)$ to denote a projection of cube $c_i(X_2)$ on the set $X_1 \subseteq X_2$ (same index i) and notation $c_i(X_1)$, $c_j(X_2)$ to denote two different cubes based on sets X_1 and X_2 (different indexes i, j).

In the definitions that follow, cube operations for cubes $c_1(X_1)$ and $c_2(X_2)$ based on different sets of variables are always reduced to the operations on cubes based on the same sets of variables $X_3 = X_1 \cup X_2$. This is done by extending corresponding cubes with literals $x_i^{S_i}$ such that $x_i \in X_3 - X_2$ for $c_1(X_1)$ or $x_i \in X_3 - X_1$ for $c_2(X_2)$, and $S_i = \emptyset$.

Definition 3.6 (product term) *An intersection of literals $\bigcap_i x_i^{S_i}$ will be called a product term.*

The product term contains a given minterm if it evaluates to **U** for that minterm.

Theorem 3.1 (relation) *Any multiple-valued directed relation can be expressed in the form:*

$$f = \bigcup_i (Q_i \cap T_i)$$

where Q_i is a set and T_i is a product term.

Proof *It is enough to show how to convert any other form to the one given above. For a relation expressed in the form of a set of tuples the correspondence is established by creating one product term for each tuple. The sets S_i for the literals in the product term are equal to the values of independent variables in the data tuples and sets Q_i equal to the corresponding values of dependent variables.*

For instance the multiple-valued relation $y_1(x_1, x_2)$ from Table 3.1 can be expressed as:

$$\begin{aligned} y_1 = & (\{0,1,2\} \cap x_1^{\{0,2\}} \cap x_2^{\{1\}}) \cup \\ & (\{0,2\} \cap x_1^{\{0,1\}} \cap x_2^{\{0\}}) \cup \\ & (\{1,2\} \cap x_1^{\{2\}} \cap x_2^{\{0\}}) \cup \\ & (\{1,2\} \cap x_1^{\{1\}} \cap x_2^{\{1\}}) \end{aligned}$$

In the definitions that follow $c_1(X_1) = \{x_i^{S_{1i}} : x_i \in X_1\}$ and $c_2(X_2) = \{x_i^{S_{2i}} : x_i \in X_2\}$.

Definition 3.7 (containment) *It is said that the cube $c_1(X_1)$ is contained in the cube $c_2(X_2)$, $c_1(X_1) \subseteq c_2(X_2)$, if $S_{1i} \subseteq S_{2i}$ for every $x_i \in X_1 \cup X_2$.*

For instance cubes $c_1 = x_2^{\{0\}}x_3^{\{1\}}x_4^{\{1\}} = x_1^{\emptyset}x_2^{\{0\}}x_3^{\{1\}}x_4^{\{1\}}$ and $c_2 = x_1^{\{1\}}x_2^{\{0\}} = x_1^{\{1\}}x_2^{\{0\}}x_3^{\emptyset}x_4^{\emptyset}$ are both contained in the cube $c_3 = x_1^{\{1,3\}}x_2^{\{0,1\}}x_3^{\{1\}}x_4^{\{1\}}$, but c_2 is not contained in c_1.

Definition 3.8 (intersection) *The intersection of cubes $c_1(X_1)$ and $c_2(X_2)$ is the cube $c_3(X_3) = c_1(X_1) \sqcap c_2(X_2)$, $X_3 = X_1 \cup X_2$, such that $c_3(X_3) = \{x_i^{S_{1i} \cap S_{2i}} : x_i \in X_3\}$.*

For instance if $c_1 = x_1^{\{3\}}x_2^{\{0\}}x_3^{\{1\}}x_4^{\{1\}}$, $c_2 = x_1^{\{1,3\}}x_2^{\{0,1\}}x_3^{\{1\}}x_4^{\{1\}}$, and $c_3 = x_1^{\{1\}}x_2^{\{0\}}$ then $c_1 \sqcap c_2 = c_1$ and $c_2 \sqcap c_3 = x_1^{\{1,3\}}x_2^{\{0,1\}}x_3^{\{1\}}x_4^{\{1\}} \sqcap x_1^{\{1\}}x_2^{\{0\}}x_3^{\emptyset}x_4^{\emptyset} = x_1^{\{1\}}x_2^{\{0\}}x_3^{\emptyset}x_4^{\emptyset} = x_1^{\{1\}}x_2^{\{0\}} = c_3$.

In the standard cube calculus if $S_i = \emptyset$ for any variable of a cube then the cube is empty. In our case different base sets for cubes are allowed and if $S_i = \emptyset$ for any x_i then the cube is equivalent to one with the literal $x_i^{S_i}$ removed from the set of literals defining the cube.

Definition 3.9 (supercube) *The supercube of cubes $c_1(X_1)$ and $c_2(X_2)$ is the cube $c_3(X_3) = c_1(X_1) \$ c_2(X_2)$, $X_3 = X_1 \cup X_2$, such that $c_3(X_3) = \{x_i^{S_{1i} \cup S_{2i}} : x_i \in X_3\}$.*

For instance: $x_1^{\{3\}}x_2^{\{1,2\}} \$ x_2^{\{0\}}x_3^{\{1\}}x_4^{\{1\}} = x_1^{\{3\}}x_2^{\{1,2\}}x_3^{\emptyset}x_4^{\emptyset} \$ x_1^{\emptyset}x_2^{\{0\}}x_3^{\{1\}}x_4^{\{1\}} = x_1^{\{3\}}x_2^{\{0,1,2\}}x_3^{\{1\}}x_4^{\{1\}}$.

Definition 3.10 (sharp product) *The sharp product of cubes $c_1(X_1)$ and $c_2(X_2)$ $(c_1 \# c_2)$ is equal to c_1 if their intersection is an improper cube, and is empty if $c_1 \subseteq c_2$. Otherwise, it is equal to:*

$$c_1 \# c_2 = \bigcup_{i=1}^{n} x_1^{S_{11}} \cdots x_i^{S_{1i} \cap \overline{S_{2i}}} \cdots x_n^{S_{1n}}$$

$c_1 \# c_2$ is a set of cubes that contains all the cubes of c_1 which are not contained by c_2.

Definition 3.11 (compaction) *Let $c_3(X_3) = c_1(X_1)\ \$\ c_2(X_2)$ and $X_3 = X_1 \cup X_2$. We will say that the cube $c_3(X_3)$ is a compaction of cubes $c_1(X_1)$ and $c_2(X_2)$, denoted by $c_1(X_1)\ \$\$\ c_2(X_2)$, iff there exists only one $x_i \in X_3$ such that $S_{1i} \neq S_{2i}$.*

The result of compaction is a cube which contains all the information contained in the original cubes so the compaction operations can be used for compression of a set of cubes.

For instance if $c_1 = x_1^{\{3\}} x_2^{\{0\}} x_3^{\{1\}} x_4^{\{1\}}$, $c_2 = x_1^{\{1\}} x_2^{\{0\}} x_3^{\{1\}} x_4^{\{1\}}$, $c_3 = x_1^{\{1\}} x_2^{\{0\}} x_3^{\{1\}}$, and $c_4 = x_1^{\{3\}} x_2^{\{0\}} x_3^{\{1\}}$ then cubes c_1 and c_2 can be replaced with cube $x_1^{\{1,3\}} x_2^{\{0\}}$ $x_3^{\{1\}} x_4^{\{1\}}$, cubes c_2 and c_3 with cube $x_1^{\{1\}} x_2^{\{0\}} x_3^{\{1\}} x_4^{\{1\}}$, and cubes c_3 and c_4 with cube $x_1^{\{1,3\}} x_2^{\{0\}} x_3^{\{1\}}$, Cubes c_2 and c_4 can not be compacted because they differ on more than one position.

3.3 Labeled rough partitions

Definition 3.12 *Separation of the elements of a nonempty set S into nonempty subsets S_i, $\bigcup S_i = S$, is called a* rough partition *(r-partition) of S.*

Rough partitions allow subsets S_i to overlap.

Definition 3.13 (relation) *Let $S = \{S_i\}$ be a set of sets S_i. A subset R of the cartesian product $S_1 \times S_2 \times \ldots \times S_k$ will be called an k-*ary relation*.*

Cartesian product operation is associative and cartesian product is a set so we can always reduce a k-ary relation to the binary relation $R \subseteq S_X \times S_Y$ where S_X and S_Y are sets of n-ary and m-ary tuples respectively and $n + m = k$. This means that we can reduce our analysis to binary relations with no loss of generality.

Definition 3.14 (directed relation) *Let relation R from S_X to S_Y be given. If S_X is a set of tuples corresponding to independent (input) variables and S_Y is a* *set of tuples corresponding to dependent (output) variables the relation R will be called a* directed relation.

Definition 3.15 (neutral relation) *A relation which is not directed will be called neutral.*

Function is a special case of a directed relation from S_X to S_Y where every element (tuple) $s_X \in S_X$ is the first member of precisely one ordered pair $(s_X, s_Y) \in S_X \times S_Y$.

An example of a multiple-valued, multi-output directed relation is shown in Table 3.1. The set of cubes $C(X) = \{x_1^{\{0,2\}}x_2^{\{1\}}, x_1^{\{0,1\}}x_2^{\{0\}}, x_1^{\{2\}}x_2^{\{0\}}, x_1^{\{1\}}x_2^{\{1\}}\}$ corresponds to the set S_X and the set of cubes $C(Y) = \{y_1^{\{0,1,2\}}y_2^{\{2\}}, y_1^{\{0,2\}}y_2^{\{1\}}, y_1^{\{1,2\}}y_2^{\{0\}}, y_1^{\{1,2\}}y_2^{\{2\}}\}$ to the set S_Y in Definition 3.14. Multiple-valued input variables x_1 and x_2 have cardinalities $|x_1| = 3$ and $|x_2| = 2$ respectively. The output variables y_1 and y_2 have both the same cardinality $|y_1| = |y_2| = 3$. The multi-output relation represented in Table 3.1 can also be viewed as a collection of a single output relation $y_1(x_1, x_2)$ and a single output function $y_2(x_1, x_2)$.

Definition 3.16 (labeled partition block) *Let $C(X)$ be a set of multiple-valued cubes, and relation $R_{c_k(X_1)}$ be defined by a cube $c_k(X_1), X_1 \subseteq X$, as follows: $c_i(X)R_{c_k(X_1)}c_j(X)$ iff $c_k(X_1) \subseteq c_i(X_1)$ and $c_k(X_1) \subseteq c_j(X_1)$, where $c_k(X_1)$ is given and $c_i(X), c_j(X) \in C(X)$. The set of all cubes $c_i(X) \in C(X)$ being in relation $R_{c_k(X_1)}$ to each other and labeled by cube $c_k(X_1)$ will be called* labeled partition block *and denoted by $B_{c_k(X_1)}$.*

Since all the cubes in $C(X)$ can be enumerated with distinct symbols (integer numbers in particular), the partition block can be represented by a set of symbols. By marking every partition block with a label (as a subscript) we establish a correspondence between the set of symbols in the partition block and the cubes in $C(X)$. For instance in Table 3.1, $X = \{x_1, x_2\}$,$Y = \{y_1, y_2\}$ are sets of independent and dependent variables, respectively, and $C(X \cup Y) = \{a, b, c, d\}$, where a, b, c, d are symbols denoting cubes $c_1(X \cup Y) = x_1^{\{0,2\}}x_2^{\{1\}}y_1^{\{0,1,2\}}y_2^{\{2\}}$, $c_2(X \cup Y) =$

$x_1^{\{0,1\}}x_2^{\{0\}}y_1^{\{0,2\}}y_2^{\{1\}}$, $c_3(X \cup Y) = x_1^{\{2\}}x_2^{\{0\}}y_1^{\{1,2\}}y_2^{\{0\}}$, and $c_4(X \cup Y) = x_1^{\{1\}}x_2^{\{1\}}y_1^{\{1,2\}}y_2^{\{2\}}$ respectively.

Example 3.1

Let $X_1 = \{x_1\}$. Then cubes $c_k(X_1)$ define the following relations:

$c_k(X_1)$	$R_{c_k(X_1)}$
$x_1^{\{0\}}$	$\{(x_1^{\{0\}}, a), (x_1^{\{0\}}, b)\}$
$x_1^{\{1\}}$	$\{(x_1^{\{1\}}, b), (x_1^{\{1\}}, d)\}$
$x_1^{\{2\}}$	$\{(x_1^{\{2\}}, a), (x_1^{\{2\}}, c)\}$

Table 3.2: Example of a relation (a).

and the corresponding labeled partition blocks are $\{a, b\}_0$, $\{b, d\}_1$, and $\{a, c\}_2$.

Let $X_2 = \{x_2\}$. Then cubes $c_k(X_2)$ define the following relations:

$c_k(X_2)$	$R_{c_k(X_2)}$
$x_2^{\{0\}}$	$\{(x_2^{\{0\}}, b), (x_2^{\{0\}}, c)\}$
$x_2^{\{1\}}$	$\{(x_2^{\{1\}}, a), (x_2^{\{1\}}, d)\}$

Table 3.3: Example of a relation (b).

and the corresponding labeled partition blocks are $\{b, c\}_0$ and $\{a, d\}_1$.

Let $X_3 = \{x_1, x_2\}$. Then cubes $c_k(X_3)$ define the following relations:

$c_k(X_3)$	$R_{c_k(X_3)}$
$x_1^{\{0,2\}}x_2^{\{1\}}$	$\{(x_1^{\{0,2\}}x_2^{\{1\}}, a)\}$
$x_1^{\{0,1\}}x_2^{\{0\}}$	$\{(x_1^{\{0,1\}}x_2^{\{0\}}, b)\}$
$x_1^{\{2\}}x_2^{\{0\}}$	$\{(x_1^{\{2\}}x_2^{\{0\}}, c)\}$
$x_1^{\{1\}}x_2^{\{1\}}$	$\{(x_1^{\{1\}}x_2^{\{1\}}, d)\}$

Table 3.4: Example of a relation (c).

and the corresponding labeled partition blocks are $\{a\}_{0,2\ 1}$, $\{b\}_{0,1\ 0}$, $\{c\}_{2\ 0}$, and $\{d\}_{1\ 1}$.

Notice that the first element of every ordered pair of relation $R_{c_i(X_j)}$ is the same and it constitutes label of the labeled partition block defined by that relation. Observe that by adding labels all information of the initial table (or set of cubes) is preserved in labeled rough partitions, thus they form a representation of this relation (function).

Definition 3.17 (labeled rough partition) *The collection of nonempty labeled partition blocks $B_{c_k(X_1)}$ that form a rough partition of a set $C(X)$ will be called* labeled rough partition *(lr-partition) and denoted by $P(X_1) = \{B_{c_k(X_1)}\}$.*

In particular, if $X_1 = \{x\}$ then $P(X_1)$ may be denoted by $P(x)$. Given the example from Table 3.1 we have $P(X_1) = P(x_1) = \{\{a,b\}_0,\ \{b,d\}_1,\ \{a,c\}_2\}_{x_1}$, $P(X_2) = P(x_2) = \{\{b,c\}_0,\ \{a,d\}_1\}_{x_2}$, and $P(X_3) = P(X) = \{\{a\}_{0,2\ 1},\ \{b\}_{0,1\ 0}$, $\{c\}_{2\ 0},\ \{d\}_{1\ 1}\}_{x_1x_2}$.

Definition 3.18 *For lr-partitions $P(X_1)$ and $P(X_2)$ of a set of cubes $C(X)$, $X_1, X_2 \subseteq X$, it is said that $P(X_1) \leq P(X_2)$ if every block of $P(X_1)$ is included in at least one block of $P(X_2)$.*

For lr-partitions $P(X_1), P(X_3)$ from Table 3.1 we have $P(X_3) \leq P(X_1)$ because $\{a\} \in \{a,b\}$, $\{b\} \in \{a,b\}$, $\{c\} \in \{a,c\}$, and $\{d\} \in \{b,d\}$. Also, $P(X_3) \leq P(X_2)$.

Definition 3.19 (labeled partition block product) *Product of two labeled partition blocks $B_{c_i(X_1)}$ and $B_{c_j(X_2)}$ is the labeled partition block $B_{c_k(X_3)} = B_{c_i(X_1)} \cap B_{c_j(X_2)}$, which partition block is an intersection of partition blocks of $B_{c_i(X_1)}$ and $B_{c_j(X_2)}$ and which label $c_k(X_3)$ is equal to $c_i(X_1)\ \$\ c_j(X_2)$.*

The product of lr-partition blocks $B_{c_1(X_1)} = \{a,b\}_0 \in P(X_1)$ and $B_{c_2(X_2)} = \{a,d\}_1 \in P(X_2)$ from Table 3.1 is lr-partition block $\{a\}_{0\ 1}$ whose label $x_1^{\{0\}}x_2^{\{1\}} = c_1(X_1)\ \$\ c_2(X_2)$.

Definition 3.20 (lr-partition product) *The product $P(X_1)P(X_2)$ of lr-partitions $P(X_1)$ and $P(X_2)$ of a set of cubes $C(X)$, where $X_1, X_2 \subseteq X$, is lr-*

partition $P(X_3), X_3 = X_1 \cup X_2$, the blocks of which are non empty products of the labeled partition blocks of $P(X_1)$ and $P(X_2)$.

The product of lr-partitions $P(X_1)$ and $P(X_2)$ from Table 3.1 is lr-partition $P(X_3) = \{\{a\}_{0\ 1},\ \{a\}_{2\ 1},\ \{b\}_{0\ 0},\ \{b\}_{1\ 0},\ \{c\}_{2\ 0},\ \{d\}_{1\ 1}\}_{x_1 x_2}$. If lr-partition blocks contain the same sets of elements and their labels can be compacted (Definition 3.11) we can combine them into one lr-partition block with a label equal to the compaction of their labels. So the above lr-partition can be reduced to $P(X_3) = \{\{a\}_{0,2\ 1}, \{b\}_{0,1\ 0}, \{c\}_{2\ 0}, \{d\}_{1\ 1}\}_{x_1 x_2}$. For lr-partitions based on overlapping sets X_1, X_3 we have: $P(X_1)P(X_3) = \{\{a\}_{0,2\ 1}, \{b\}_{0,1\ 0}, \{c\}_{2\ 0}, \{d\}_{1\ 1}\}_{x_1 x_2} = P(X_3)$.

The following two important theorems establish the way lr-partitions can be combined or splitted which will allow us to optimize the memory requirement for lr-partitions representation (see Section 3.4).

Theorem 3.2 *For any set of cubes $C(X)$, and any set of subsets X_i of X, $P(\bigcup_i X_i) = \prod_i P(X_i)$.*

PROOF *It is enough to show that $P(X_1 \cup X_2) = P(X_1)P(X_2)$. By Definitions 3.16 and 3.17 lr-partition $P(X_i)$ consists of blocks corresponding to every combination of values of variables $x \in X_i$ present in data. Hence, by Definitions 3.19 and 3.20 product $P(X_1)P(X_2)$ consists of blocks corresponding to every combination of values of variables $x \in X_1 \cup X_2$ present in data. Hence, $P(X_1 \cup X_2) = P(X_1)P(X_2)$.*

Corollary 3.1 *If $X_1 \subseteq X_2$ then $P(X_1)\,P(X_2) = P(X_2)$.*

PROOF *If $X_1 = X_2$ then $P(X_1)\,P(X_2) = P(X_2)\,P(X_2) = P(X_2)$. If $X_1 \subset X_2$ then there exists X_1' such that $X_2 = X_1 \cup X_1'$ and $X_1 \cap X_1' = \emptyset$. Hence, by Theorem 3.2 $P(X_1)P(X_2) = P(X_1)P(X_1 \cup X_1') = P(X_1)P(X_1)P(X_1') = P(X_1)P(X_1') = P(X_1 \cup X_1') = P(X_2)$.*

Theorem 3.3 (lr-partition extraction) *For given sets of variables X and X_1, $X_1 \subseteq X$, and lr-partition $P(X) = \{B_{c_i'(X)}\}$, the lr-partition $P(X_1) = \{B_{c_j''(X_1)}\}$ extracted from $P(X)$ consists of the partition blocks $B_{c_j''(X_1)}$ satisfying the following* *condition: each $B_{c''_j(X_1)}$ is a union of those partition blocks $B_{c'_i(X)}$ of lr-partition $P(X)$ which labels satisfy the condition*

$$c'_i(X_1) \supseteq c''_j(X_1)$$

where $c'_i(X_1)$ is the projection of cube $c'_i(X)$ on the set X_1.

PROOF *By Definitions 3.12 and 3.17 it must be $\bigcup_j B_{c''_j(X_1)} = \bigcup_i B_{c'_i(X)}$. The necessary condition to satisfy this requirement is that labels $c''_j(X_1)$ are selected in such a way that for every $c''_j(X_1)$ there exists $c'_i(X_1)$ such that $c''_j(X_1) \subseteq c'_i(X_1)$. For a given $c''_j(X_1)$ there may be many $c'_i(X_1)$ satisfying condition $c''_j(X_1) \subseteq c'_i(X_1)$ so the condition $\bigcup_j B_{c''_j(X_1)} = \bigcup_i B_{c'_i(X)}$ can only be satisfied if $B_{c''_j(X_1)}$ is a union of the blocks $B_{c'_i(X)}$ corresponding to those $c'_i(X_1)$. This is the sufficient condition.*

Given the example from Table 3.1, let us extract lr-partition $P(X_1)$ from $P(X_3)$, $X_1 = \{x_1\}$, $X_3 = \{x_1, x_2\}$. Since $P(X_3) = \{\{a\}_{0,2\ 1}, \{b\}_{0,1\ 0}, \{c\}_{2\ 0}, \{d\}_{1\ 1}\}_{x_1x_2}$ then we have the following $c'_i(X_3)$ cubes: $c'_1(X_3) = x_1^{\{0,2\}}x_2^{\{1\}}$, $c'_2(X_3) = x_1^{\{0,1\}}x_2^{\{0\}}$, $c'_3(X_3) = x_1^{\{2\}}x_2^{\{0\}}$, and $c'_4(X_3) = x_1^{\{1\}}x_2^{\{1\}}$. Projecting $c'_i(X_3)$ cubes on the set X_1 we have the following $c'_i(X_1)$ cubes: $c'_1(X_1) = x_1^{\{0,2\}}$, $c'_2(X_1) = x_1^{\{0,1\}}$, $c'_3(X_1) = x_1^{\{2\}}$, and $c'_4(X_1) = x_1^{\{1\}}$. Let us select the following $c''_j(X_1)$ cubes: $c''_1(X_1) = x_1^{\{0\}}$, $c''_2(X_1) = x_1^{\{1\}}$, $c''_3(X_1) = x_1^{\{2\}}$, It can be easily verified that the necessary condition is satisfied. To satisfy sufficient conditions blocks $B_{c''_j(X_1)}$ of lr-partition $P(X_1)$ are computed as follows: $B_0 = B_{0,2\ 1} \cup B_{0,1\ 0} = \{\{a\} \cup \{b\}\}_0 = \{a, b\}_0$, $B_1 = B_{0,1\ 0} \cup B_{1\ 1} = \{\{b\} \cup \{d\}\}_1 = \{b, d\}_1$, $B_2 = B_{0,2\ 1} \cup B_{2\ 0} = \{\{a\} \cup \{c\}\}_2 = \{a, c\}_2$.

3.4 Representation of multiple-valued relations

Theorem 3.4 (representation) *The multiple-valued, multi-output relation $Y = f(X)$ can be represented as a pair of sets of lr-partitions $\{\{P(X_i)\}, \{P(Y_j)\}\}$, where $\bigcup_i X_i = X, \bigcup_j Y_j = Y$ and X, Y are sets of input (independent) and output (dependent) variables respectively.*

PROOF *It is enough to show that transition to another representation is possible. By Theorem 3.2, $P(X \cup Y) = \prod_{X_i} P(X_i)\ \prod_{Y_j} P(Y_j)$ and the set of block* *labels of* $P(X \cup Y)$ *forms a cube representation of relation* $Y = f(X)$.

Notice that the lr-partitions data structure defined by Theorem 3.4 allows not only the sets X_i and Y_j but also X and Y to overlap. The later means that systems (relations) with feedback can be represented. Neutral relations (all the variables are independent) can be represented too.

Contrary to the rough partition [75] which stores an abstraction of a function, the labeled rough partitions can be used for general purpose representation of functions and relations because no information is lost in them.

cube id	x_1	x_2	x_3	x_4	y_1	y_2
a	0	0	0	0	0,1	0,1,2
b	0	1	1	0,1	1,2	1
c	1	1	0	0	0	1,2
d	0	0,2	0	1	3	0,2
e	1	1	2	1	0,3	0

Table 3.5: Multiple-valued relation represented by cubes.

Example 3.2

Let us construct lr-partitions for the multiple-valued relation from Table 3.5.

$P(x_1) = \{\{a, b, d\}_0, \{c, e\}_1\}_{x_1}$

$P(x_2) = \{\{a, d\}_0, \{b, c, e\}_1, \{d\}_2\}_{x_2}$

$P(x_3) = \{\{a, c, d\}_0, \{b\}_1, \{e\}_2\}_{x_3}$

$P(x_4) = \{\{a, b, c\}_0, \{b, d, e\}_1\}_{x_4}$

$P(y_1) = \{\{a, c, e\}_0, \{a, b\}_1, \{b\}_2, \{d, e\}_3\}_{y_1}$

$P(y_2) = \{\{a, d, e\}_0, \{a, b, c\}_1, \{a, c, d\}_2\}_{y_2}$

$P(X) = \{\{a\}_{0\ 0\ 0\ 0}, \{b\}_{0\ 1\ 1\ 0,1}, \{c\}_{1\ 1\ 0\ 0}, \{d\}_{0\ 0,2\ 0\ 1}, \{e\}_{1\ 1\ 2\ 1}\}_{x_1 x_2 x_3 x_4}$

$P(Y) = \{\{a\}_{0,1\ 0,1,2}, \{b\}_{1,2\ 1}, \{c\}_{0\ 1,2}, \{d\}_{3\ 0,2}, \{e\}_{0,3\ 0}\}_{y_1 y_2}$

As we can see from the Example 3.2 lr-partitions based on single variables are in a sense a column representation of the table. Each block of $P(x_i)$ is a list of rows of the table that contain a given value of the variable x_i.

Representation with partitions $P(X), P(Y)$ is in a sense a row representation of the table. They can have fewer blocks than those represented with partitions based on single variables but labels are longer and more memory is needed to store them. This form is basically the same as the cube representation.

The intermediate cases can be constructed too. The more variables sets X_i, Y_j contain the more representation resembles row representation of the table and vice versa.

Selection of a set of subsets X_i and Y_i of input and output variables, with which partition blocks are defined, is one of the fundamental choices one has to make when constructing lr-partitions. This is of similar importance as is the variable ordering for BDDs.

The new representation is a combination of row and column representations of the data set so the user has more ways of compressing the information:

- by compressing sets of minterms into cubes
- by encoding cubes with new variables (row numbers)
- by selecting set representation for lr-partition blocks

This way, paying the price of higher conceptual complexity, we increase flexibility of storing information contained in the data. By making use of the existence of repetitive parts of data cubes lr-partitions can facilitate factorization and decomposition of multiple-valued functions and relations, especially in the presence of many don't cares.

Lr-partitions representation has been created to enable efficient representation of incompletely specified functions and relations of many variables. If a cube has standard output don't cares for *all* its outputs, it is not stored at all, only care cubes are stored which for large functions and relations with many don't cares, results in savings in both storage and processing time. A MVDD for instance has to store pointers to the terminal node 'DC'. If there are N disjoint DC cubes in a map, there would be N such pointers, and this number can be exponential in the number of input variables. As a result, the size of lr-partition representation is at worst of the order of the number of cares, so it does not depend on the location of don't cares.

Using lr-partitions to represent relations, the set-values of input and output variables are stored and processed in the same way without loosing the information about their distinct functionalities. This is another advantage of our representation, input and output variables are represented in a uniform way.

3.5 Operations on multiple-valued relations

In this section the algorithms to perform basic operations on multiple-valued relations (functions in particular) will be presented. These few operations (restriction, satisfy, complement, compose) can be combined to perform a wide variety of more complex operations.

3.5.1 Restriction (cofactor)

Let us first define restriction for multiple-valued relations.

Definition 3.21 (restriction) *Let $Y = f(X)$ be a multiple-valued, multi-output relation defined be a set of cubes $C(X \cup Y) = \{c_i(X \cup Y)\}$. The restriction of $Y = f(X)$ in respect to the cube $c(X_1 \cup Y_1)$ is another relation $Y' = f'(X')$ defined by a set of cubes $C((X - X_1) \cup (Y - Y_1)) = \{c_i((X - X_1) \cup (Y - Y_1))\}$ such that $c_i(X_1 \cup Y_1) = c(X_1 \cup Y_1)$ and $c_i((X - X_1) \cup (Y - Y_1))$ and $c_i(X_1 \cup Y_1)$ are projections of $c_i(X \cup Y)$ on the sets $(X - X_1) \cup (Y - Y_1)$ and $X_1 \cup Y_1$ respectively.*

For instance a restriction of the function $y = f(x_1, x_2, x_3)$ in respect to the cube $x_2^{\{1\}} x_3^{\{0\}}$ is the function $y' = f'(x_1) = f(x_1, 1, 0)$.

The computational procedure for the simplest case where each of the sets X_i, Y_j contains only one variable is described by Algorithm 1 where $B_{c(X_1 \cup Y_1)}$ is an intersection of all the blocks $B_{x_i^{S_i}}$ and $B_{y_j^{S_j}}$ such that $x_i^{S_i}, y_j^{S_j} \in c(X_1 \cup Y_1)$.

```
!htp] Restriction: simple case [1] every $x_j \in X - X_1$ $P(x_j) := P(x_j)B_{c(X_1 \cup Y_1)}$ every $y_j \in Y - Y_1$ $P(y_j) :=$
$P(y_j)B_{c(X_1 \cup Y_1)}$
```

end

For the general case of sets X_i, Y_j containing arbitrary numbers of variables, a more general Algorithm 2 must be formulated. Algorithm 3 describes the process of computing intermediate blocks needed for determination of blocks $B_{c(X_1)}$, $B_{c(Y_1)}$ and $B_{c(X_1 \cup Y_1)}$ in Algorithm 2. This is denoted by `sum` in the pseudo code of Algorithm 2.

```
!htp] Restriction: general case [1]
Compute $B_{c(X_1)}$ $B_{c(X_1)} := \bigcap_{X_i \cap X_1 \neq \emptyset} sum(P(X_i), c(X_i \cap X_1))$
Compute $B_{c(Y_1)}$ $B_{c(Y_1)} := \bigcap_{Y_i \cap Y_1 \neq \emptyset} sum(P(Y_i), c(Y_i \cap Y_1))$
$B_{c(X_1 \cup Y_1)} := B_{c(X_1)} \cap B_{c(Y_1)}$
Compute lr-partitions $P(X_i - X_1)$ every $X_i$ $X_i - X_1 \neq \emptyset$ $P(X_i - X_1) := P(X_i - X_1)B_{c(X_1 \cup Y_1)}$
Compute lr-partitions $P(Y_j - Y_1)$ every $Y_j$ $Y_j - Y_1 \neq \emptyset$ $P(Y_j - Y_1) := P(Y_j - Y_1)B_{c(X_1 \cup Y_1)}$
```

end

```
!htp] $sum(P(X_i), c(X_p))$ [1]
$B_{c(X_p)} := \emptyset$ every $B_{c_j(X_i)} \in P(X_i)$ $B_{c(X_p)} := \bigcup_{c_j(X_p) \supseteq c(X_p)} B_{c_j(X_p)}$ return $B_{c(X_p)}$
```

end

Notice that the restriction of a relation, represented by sets of lr-partitions $\{P(X_i)\}$ and $\{P(Y_j)\}$, with respect to the cube $c(X_k \cup Y_k)$ consists of sets of lr-partitions $\{P(X_i - X_k)\}, \{P(Y_j - Y_k)\}$.

3.5.2 Containment

Let us consider a situation when we have an input-output cube $c'(X \cup Y)$ and we want to check if it is contained in a given function or relation $Y = f(X)$ represented by lr-partitions $\{P(X_i)\}, \{P(Y_j)\}$. In other words we want to check if the cube $c'(X \cup Y)$ was in the set of cubes used for creation of lr-partitions representation.

In the simplest case cube $c'(X \cup Y)$ is a minterm, sets X_i, Y_j contain one variable each, and f is a binary function. In this case, each partition $P(x_i), P(y_j)$ contains only two blocks (at most) labeled with 0 and 1. Now, for each variable x_i, let us select block B_0 in lr-partition $P(x_i)$ if $x_i^0 \in c'(X \cup Y)$ and block B_1 otherwise. Let us repeat the selection process for variables y_j and lr-partitions $P(y_j)$. If the intersection of all the selected blocks is not empty then cube $c'(X \cup Y)$ is contained in the function $Y = f(X)$, otherwise it is not. For multiple-valued functions variables x_i, y_j may take more than two values and the number of blocks in each lr-partition to select from may be greater than two but otherwise the algorithm remains the same.

For the general case of multiple-valued relations represented by cubes the following procedure can be used to check the containment:

Cube $c'(X \cup Y)$ is contained in relation $Y = f(X)$ iff the following is true:

$$\bigcap_i B_{c(X_i)} \cap \bigcap_j B_{c(Y_j)} \neq \emptyset$$

where:

$$B_{c(X_i)} = \bigcup_{c_k(X_i) \supseteq c'(X_i)} B_{c_k(X_i)}$$

$$B_{c(Y_j)} = \bigcup_{c_k(Y_j) \supseteq c'(Y_j)} B_{c_k(Y_j)}$$

and $c'(X_i)$, $c'(Y_j)$ are projections of $c'(X \cup Y)$ on the sets X_i and Y_j respectively.

Example 3.3

Let us check if the cube $c'(X \cup Y) = x_1^{\{0\}} x_2^{\{1\}} x_3^{\{1\}} x_4^{\{0\}} y_1^{\{1\}} y_2^{\{1\}}$, where $X = \{x_1, x_2, x_3, x_4\}$ and $Y = \{y_1, y_2\}$, is contained in the relation from Example 3.2. For $X_1 = \{x_1\}$,$X_2 = \{x_2\}$,$X_3 = \{x_3\}$, $X_4 = \{x_4\}$,$Y_1 = \{y_1\}$ and $Y_2 = \{y_2\}$ we have:

$P(X_1) = \{\{a, b, d\}_0, \{c, e\}_1\}_{x_1}$

$P(X_2) = \{\{a, d\}_0, \{b, c, e\}_1, \{d\}_2\}_{x_2}$

$P(X_3) = \{\{a, c, d\}_0, \{b\}_1, \{e\}_2\}_{x_3}$

$P(X_4) = \{\{a, b, c\}_0, \{b, d, e\}_1\}_{x_4}$

$P(Y_1) = \{\{a,c,e\}_0, \{a,b\}_1, \{b\}_2, \{d,e\}_3\}_{y_1}$
$P(Y_2) = \{\{a,d,e\}_0, \{a,b,c\}_1, \{a,c,d\}_2\}_{y_2}$
From the definition of containment we have:
$B_{c(X_i)} = \bigcup_{c_k(X_i) \supseteq c'(X_i)} B_{c_k(X_i)}$ and
$B_{c(X_1)} = \bigcup_{c_k(X_1) \supseteq x_1^{\{0\}}} B_{c_k(X_1)} = \{a,b,d\}$,
$B_{c(X_2)} = \bigcup_{c_k(X_2) \supseteq x_2^{\{1\}}} B_{c_k(X_2)} = \{b,c,e\}$,
$B_{c(X_3)} = \bigcup_{c_k(X_3) \supseteq x_3^{\{1\}}} B_{c_k(X_3)} = \{b\}$,
$B_{c(X_4)} = \bigcup_{c_k(X_4) \supseteq x_4^{\{0\}}} B_{c_k(X_4)} = \{a,b,c\}$,
$B_{c(Y_1)} = \bigcup_{c_k(Y_1) \supseteq y_1^{\{1\}}} B_{c_k(Y_1)} = \{a,b\}$,
$B_{c(Y_2)} = \bigcup_{c_k(Y_2) \supseteq y_2^{\{1\}}} B_{c_k(Y_2)} = \{a,b,c\}$,
$\bigcap_i B_{c(X_i)} = \{a,b,d\} \cap \{b,c,e\} \cap \{b\} \cap \{a,b,c\} = \{b\}$,
$\bigcap_j B_{c(Y_j)} = \{a,b\} \cap \{a,b,c\} = \{a,b\}$.
Therefore:

$$\bigcap_i B_{c(X_i)} \ \cap \ \bigcap_j B_{c(Y_j)} = \{b\} \cap \{a,b\} = \{b\} \neq \emptyset$$

and cube $c'(X \cup Y) = x_1^{\{0\}} x_2^{\{1\}} x_3^{\{1\}} x_4^{\{0\}} y_1^{\{1\}} y_2^{\{1\}}$ is contained in the relation from Example 3.2.

Let us repeat the computations for the relation represented by the second set of lr-partitions:
$P(X) = \{\{a\}_{0\ 0\ 0\ 0}, \{b\}_{0\ 1\ 1\ 0,1}, \{c\}_{1\ 1\ 0\ 0}, \{d\}_{0\ 0,2\ 0\ 1}, \{e\}_{1\ 1\ 2\ 1}\}_{x_1x_2x_3x_4}$
$P(Y) = \{\{a\}_{0,1\ 0,1,2}, \{b\}_{1,2\ 1}, \{c\}_{0\ 1,2}, \{d\}_{3\ 0,2}, \{e\}_{0,3\ 0}\}_{y_1y_2}$
From the definition of containment we have:
$B_{c(X)} = \bigcup_{c_k(X) \supseteq x_1^{\{0\}} x_2^{\{1\}} x_3^{\{1\}} x_4^{\{0\}}} B_{c_k(X)} = \{b\}$,
$B_{c(Y)} = \bigcup_{c_k(Y) \supseteq y_1^{\{1\}} y_1^{\{1\}}} B_{c_k(Y)} = \{a\} \cup \{b\} = \{a,b\}$,
and

$$B_{c(X)} \ \cap \ B_{c(Y)} = \{b\} \cap \{a,b\} = \{b\} \neq \emptyset$$

which again shows that the cube $c'(X \cup Y)$ is contained in the relation from Example 3.2.

The containment operation can be used for instance to check whether two relations are equivalent or to find a satisfiability set for a relation.

3.5.3 Complement

Let us consider first a binary function $Y = f(X)$. Each lr-partitions $P(y_i)$ of that function consist of two blocks B_0, B_1. In order to perform the complement of function f the only thing we need to do is to swap block labels of every lr-partition $P(y_i)$ so that the blocks B_0 and B_1 will retain their contents but the labels change their values from 0 to 1 and from 1 to 0 respectively. The similar procedures can be applied for multiple-valued functions depending of the definition of the complement [120][107].

3.5.4 Composition

The composition of directed relations can be described by the following equation:

$$Y_2 = f_2(Y_1, X_2) \Rightarrow Y_0 = f_0(X_0)$$

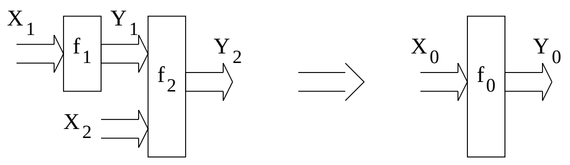


Figure 3.3: Non-disjoint serial composition.

where:

$$Y_1 = f_1(X_1)$$
$$X_1 \cap X_2 = X_3$$
$$X_0 = X_1 \cup X_2$$
$$Y_0 = Y_2$$

Directed relations (functions in particular) from Figure 3.3 are represented by lr-partitions as follows:

$f_1 = \{\{P_1(X_{1i})\}, \{P_1(Y_{1j})\}\}$:

$$\begin{array}{ll} P_1(X_{1i}) = \{B_{c_{1n}(X_{1i})}\} & \bigcup_i X_{1i} = X_1 \\ P_1(Y_{1j}) = \{B_{c_{1n}(Y_{1j})}\} & \bigcup_j Y_{1j} = Y_1 \end{array}$$

$f_2 = \{\{\{P_2(X_{2i})\}, \{P_2(Y_{1j})\}\}, \{P_2(Y_{2k})\}\}$:

$$\begin{array}{ll} P_2(X_{2i}) = \{B_{c_{2n}(X_{2i})}\} & \bigcup_i X_{2i} = X_2 \\ P_2(Y_{1j}) = \{B_{c_{2n}(Y_{1j})}\} & \bigcup_j Y_{1j} = Y_1 \\ P_2(Y_{2k}) = \{B_{c_{2n}(Y_{2k})}\} & \bigcup_k Y_{2k} = Y_2 \end{array}$$

$f_0 = \{\{\{P_0(X_{1i})\}, \{P_0(X_{2j})\}\}, \{P_0(Y_{2k})\}\}$:

$$\begin{array}{ll} P_0(X_{1i}) = \{B_{c_{1n}(X_{1i})}\} & \bigcup_i X_{1i} = X_1 \\ P_0(X_{2j}) = \{B_{c_{2n}(X_{2j})}\} & \bigcup_j X_{2j} = X_2 \\ P_0(Y_{2k}) = \{B_{c_{2n}(Y_{2k})}\} & \bigcup_k Y_{2k} = Y_2 \end{array}$$

Let $P_1(Y_1) = \{B_{c_{1n}(Y_1)}\}$ and $P_2(Y_1) = \{B_{c_{2n}(Y_1)}\}$.

The composition of relations f_1 and f_2 described by the sets of cubes $C(X_1 \cup Y_1)$ and $C(Y_1 \cup X_2 \cup Y_2)$ consists of all possible cubes $c(X_1 \cup X_2 \cup Y_2)$ such that $c(X_1 \cup X_2 \cup Y_2)$ is a concatenation of cube $c(X_1)$ of f_1 and cube $c(X_2 \cup Y_2)$ of f_2, and the value of $c(Y_1)$ of f_1 is equal to the value of $c(Y_1)$ of f_2. It is equivalent to performing the following operation (see [137]):

$$(C(X_1 \cup Y_1) \times C(X_2 \cup Y_2)) \cap (C(X_1) \times C(Y_1 \cup X_2 \cup Y_2))$$

To perform a composition of relations f_1 and f_2 into a relation f_0 represented by lr-partitions, Algorithm 4 has been formulated and for simplification the cube symbols were encoded with integer numbers. The result of composition $f_0 = \{\{\{P_0(X_{1i})\}, \{P_0(X_{2j})\}\}, \{P_0(Y_{2k})\}\}$ consists of lr-partitions based on the sets X_{1i}, X_{2j}, and Y_{2k}, the same sets as in the relations f_1 and f_2.

!htp] Composition [1]

Step 1: rr = 0 $P_1(X_3 \cup Y_1) = P_1(X_3)P_1(Y_1)$ $P_2(X_3 \cup Y_1) = P_2(X_3)P_2(Y_1)$ *Step 2: Compute new row numbers rr and partitions* $P_0(X_{2k})$ *every* $P_2(X_{2k})$ $P_2(X_{2k} \cup X_3 \cup Y_1) = P_2(X_{2k})P_2(X_3 \cup Y_1)$ *every pair of blocks* $B_{c_{1i}(X_3 \cup Y_1)} \in P_1(X_3 \cup Y_1)$ *and* $B_{c_{2j}(X_{2k} \cup X_3 \cup Y_1)} \in P_2(X_{2k} \cup X_3 \cup Y_1)$ $c_{1i}(X_3 \cup Y_1) \subseteq c_{2j}(X_3 \cup Y_1)$ *in pair* $B_{c_{1i}(X_3 \cup Y_1)}, B_{c_{2j}(X_{2k} \cup X_3 \cup Y_1)}$ *every pair* $m \in B_{c_{1i}(X_3 \cup Y_1)}$ *and* $n \in B_{c_{2j}(X_{2k} \cup X_3 \cup Y_1)}$ *the first of the sets* X_{2k} $r[m][n] = rr$ $rr = rr + 1$ *add* $r[m][n]$ *to* $B_{c_{2j}(X_{2k})} \in P_0(X_{2k})$

Step 3: Compute partitions $P_0(X_{1k})$ *every* $P_1(X_{1k})$ *every* $B_{c_{1i}(X_{1k})} \in P_1(X_{1k})$ $m \in B_{c_{1i}(X_{1k})}$ *Add row m of* $r[m][n]$ *to* $B_{c_{1i}(X_{1k})} \in P_0(X_{1k})$

Step 4: Compute partitions $P_0(Y_{2k})$ every $P_2(Y_{2k})$ every $B_{c_{2j}(Y_{2k})} \in P_2(Y_{2k})$ $n \in B_{c_{2j}(Y_{2k})}$ Add column n of $r[m][n]$ to $B_{c_{2j}(Y_{2k})} \in P_0(Y_{2k})$

end

The fact that the new row numbers rr in Algorithm 4 need to be computed only once is proved in Theorem 3.5.

Theorem 3.5 (new row numbers) *The new row numbers rr in Algorithm 4 need to be computed only for one lr-partition $P_2(X_{2k})$.*

PROOF *By Definition 3.20 all the blocks $B_{c_{2j}(X_{2k} \cup Y_1 \cup X_3)}$ of lr-partition $P_2((X_{2k} - X_3) \cup Y_1 \cup X_3) = P_2(X_{2k})P_2(Y_1 \cup X_3)$ which have equal projections $c_{2j}(Y_1 \cup X_3)$ contain all the row numbers and only the row numbers from the block $B_{c'_{2j}(Y_1 \cup X_3)}$ of lr-partition $P_2(Y_1 \cup X_3)$ regardless of the value of k. Since m is taken from $B_{c_{1i}(Y_1 \cup X_3)}$ which doesn't depend of k, and n taken from $B_{c_{2j}(X_{2k} \cup Y_1 \cup X_3)}$ such that $c_{1i}(Y_1 \cup X_3) \subseteq c_{2j}(Y_1 \cup X_3)$ the set of pairs m, n doesn't depend of k either. Therefore it is enough to compute the set of new row numbers only for the first of the sets X_{2k}.*

Example 3.4 (composition)

Let incompletely specified relations f_1 and f_2 be defined as follows:

f_1:

#	x_1	x_2	x_3	y_1
0	1	0	1	0
1	0	1	1	1
2	1	1	0	1
3	0	1	0	0

$$P_1(X_{11}) = \{\{1,3\}_{01}\{0\}_{10}\{2\}_{11}\}_{x_1x_2}$$
$$P_1(X_{12}) = \{\{0\}_{01}\{2,3\}_{10}\{1\}_{11}\}_{x_2x_3}$$
$$P_1(Y_{11}) = \{\{0,3\}_0\{1,2\}_1\}_{y_1}$$
$$P_1(Y_1) = P_1(Y_{11})$$

f_2:

#	y_1	x_3	x_4	x_5	y_2
0	0	0	0	1	1
1	0	0	1	1	1
2	0	1	0	0	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	1	1	1	0

$$P_2(Y_{11}) = \{\{0,1,2,3\}_0\{4,5\}_1\}_{y_1}$$
$$P_2(X_{21}) = \{\{0,4\}_{00}\{1\}_{01}\{2\}_{10}\{3,5\}_{11}\}_{x_3x_4}$$
$$P_2(X_{22}) = \{\{2\}_{00}\{0,4\}_{01}\{3\}_{10}\{1,5\}_{11}\}_{x_4x_5}$$
$$P_2(Y_{21}) = \{\{2,4,5\}_0\{0,1,3\}_1\}_{y_2}$$
$$P_2(Y_1) = P_2(Y_{11})$$

where:

$$\begin{aligned}
X_{11} &= \{x_1, x_2\} \\
X_{12} &= \{x_2, x_3\} \\
Y_{11} &= \{y_1\} \\
X_{21} &= \{x_3, x_4\} \\
X_{22} &= \{x_4, x_5\} \\
Y_{21} &= \{y_2\} \\
X_1 &= X_{11} \cup X_{12} = \{x_1, x_2, x_3\} \\
Y_1 &= Y_{11} = \{y_1\} \\
X_2 &= X_{21} \cup X_{22} = \{x_3, x_4, x_5\} \\
Y_2 &= Y_{21} = \{y_2\} \\
X_3 &= X_1 \cap X_2 = \{x_3\}
\end{aligned}$$

Our goal is to find a relation f_0 to be a composition of relations f_1 and f_2. First, we compute the new row numbers and partition $P_0(X_{21})$. Following Algorithm 4 we have:

$$\begin{aligned}
P_1(X_3 \cup Y_1) &= \{\{3\}_{00}\{2\}_{01}\{0\}_{10}\{1\}_{11}\}_{x_3y_1} \\
P_2(X_3 \cup Y_1) &= \{\{0,1\}_{00}\{4\}_{01}\{2,3\}_{10}\{5\}_{11}\}_{x_3y_1} \\
P_2(X_{21} \cup X_3 \cup Y_1) &= P_2(X_{21})\, P_2(X_3 \cup Y_1) \\
&= \{\{0\}_{000}\{4\}_{001}\{1\}_{010}\{2\}_{100}\{3\}_{110}\{5\}_{111}\}_{x_3x_4y_1}
\end{aligned}$$

The pairs of blocks that satisfy condition in line 10 of Algorithm 4 are listed in Table 3.6.

$B_{c_{1i}(x_3y_1)}$	$B_{c_{2j}(x_3x_4y_1)}$
$\{3\}_{00}$	$\{0\}_{000}$
$\{3\}_{00}$	$\{1\}_{010}$
$\{2\}_{01}$	$\{4\}_{001}$
$\{0\}_{10}$	$\{2\}_{100}$
$\{0\}_{10}$	$\{3\}_{110}$
$\{1\}_{11}$	$\{5\}_{111}$

Table 3.6: Pairs of blocks (a).

Denoting by m, n the row numbers in pairs of blocks from the column 1 and 2 of Table 3.6 the row numbers for relation f_0 result as in Table 3.7.

$m \setminus n$	0	1	2	3	4	5
0			3	4		
1						5
2					2	
3	0	1				

Table 3.7: New row numbers.

Empty boxes in Table 3.7 correspond to the combinations of m, n which don't appear in the pairs of blocks from Table 3.6, for instance there is no pair with number 0 in $B_{c_{1i}(X_3 \cup Y_1)}$ and 4 in $B_{c_{2j}(X_{21} \cup X_3 \cup Y_1)}$.

The portions $c_{2j}(X_{21}) = c_{2j}(x_3x_4)$ of the labels in blocks $\{0\}_{000}$, $\{4\}_{001}$ are identical and equal to 00 so the combinations of m, n in pairs containing these two blocks will account for the new row numbers in block B_{00} of lr-partition $P_0(X_{21})$. The corresponding combinations of m, n are (3,0) and (2,4), and the new row numbers from Table 3.7 are $\{0, 2\}$. $c_{2j}(x_3x_4)$ portion of the label is equal to 01 in block $\{1\}_{010}$, combination of m, n is (3,1) and corresponding new row number from Table 3.7 is $\{1\}$. $c_{2j}(x_3x_4)$ portion of the label is equal to 10 in block $\{2\}_{100}$, combination of m, n is (0,2) and corresponding new row number from Table 3.7 is $\{3\}$. $c_{2j}(x_3x_4)$ portions of the labels are equal to 11 in blocks $\{3\}_{110}$ and $\{5\}_{111}$, combinations of m, n are (0,3),(1,5) and corresponding new row numbers from Table 3.7 are $\{4, 5\}$.

Therefore, lr-partition $P_0(X_{21})$ is equal to:

$$P_0(X_{21}) = \{\{0,2\}_{00}\{1\}_{01}\{3\}_{10}\{4,5\}_{11}\}_{x_3x_4}$$

Continuing computations of lr-partitions $P_0(X_{2i})$ we have:

$$\begin{aligned} P_2(X_{22} \cup X_3 \cup Y_1) &= P_2(X_{22})\, P_2(X_3 \cup Y_1) \\ &= \{\{2\}_{0010}\{0\}_{0100}\{4\}_{0101}\{3\}_{1010}\{1\}_{1100}\{5\}_{1111}\}_{x_4x_5x_3y_1} \end{aligned}$$

The corresponding pairs of blocks are listed in Table 3.8.

$B_{c_{1i}(x_3y_1)}$	$B_{c_{2j}(x_4x_5x_3y_1)}$
$\{3\}_{00}$	$\{0\}_{0100}$
$\{3\}_{00}$	$\{1\}_{1100}$
$\{2\}_{01}$	$\{4\}_{0101}$
$\{0\}_{10}$	$\{2\}_{0010}$
$\{0\}_{10}$	$\{3\}_{1010}$
$\{1\}_{11}$	$\{5\}_{1111}$

Table 3.8: Pairs of blocks (b).

Combinations of m, n are the same as before so from Tables 3.8 and 3.7 the lr-partition $P_0(X_{22})$ results:

$$P_0(X_{22}) = \{\{3\}_{00}\{0,2\}_{01}\{4\}_{10}\{1,5\}_{11}\}_{x_4x_5}$$

To compute lr-partitions $P_0(X_{1i})$ we check the row numbers contained in every block $B_{c_{1i}(X_{1k})} \in P_1(X_{1k})$. For every row number $m \in B_{c_{1b}(X_{1i})}$ we add the row m of Table 3.7 to the block $B_{c_{1i}(X_{1k})} \in P_0(X_{1k})$. Block B_{00} of lr-partition $P(X_{11})$ contains row numbers 1 and 3, so the block B_{00} of lr-partition $P_0(X_{11})$ will contain new row numbers from the rows 1 and 3 of Table 3.7: {0,1,5}. Following the above procedure we get the following lr-partitions $P_0(X_{11})$ and $P_0(X_{12})$:

$$P_0(X_{11}) = \{\{0,1,5\}_{01}\{3,4\}_{10}\{2\}_{11}\}_{x_1x_2}$$

$$P_0(X_{12}) = \{\{3,4\}_{01}\{0,1,2\}_{10}\{5\}_{11}\}_{x_2x_3}$$

To compute lr-partitions $P_0(Y_{2k})$ we follow the same procedure as for $P_0(X_{1k})$ but instead of adding rows we add columns of Table 3.7 to the corresponding blocks of $P_0(Y_{2k})$. The column numbers correspond to the numbers contained in blocks of lr-partition $P_2(Y_{2k})$. The resulting lr-partition is:

$$P_0(Y_{21}) = \{\{2,3,5\}_0\{0,1,4\}_1\}_{y_2}$$

This corresponds directly to the following set of cubes:

f_0:

#	x_1	x_2	x_3	x_4	x_5	y_2
0	0	1	0	0	1	1
1	0	1	0	1	1	1
2	1	1	0	0	1	0
3	1	0	1	0	0	0
4	1	0	1	1	0	1
5	0	1	1	1	1	0

Another situation when the composition is needed is the one shown in Figure 3.4.

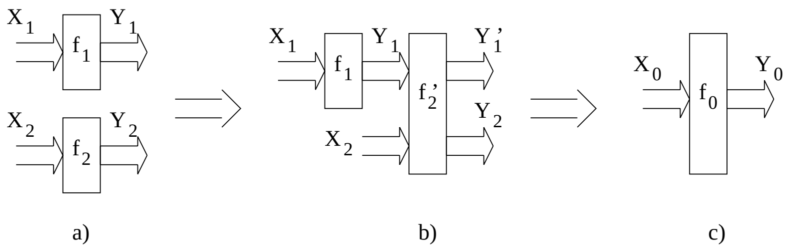


Figure 3.4: Parallel composition.

This case can be described by the following equation:

$$\{Y_1 = f_1(X_1), Y_2 = f_2(X_2)\} \Rightarrow \{Y_1', Y_2\} = f_2'(X_2, Y_1) \Rightarrow Y_0 = f_0(X_0)$$

where $Y_1' = Y_1, X_0 = X_1 \cup X_2$, $Y_0 = Y_1' \cup Y_2$, and sets X_1, X_2 and Y_1, Y_2 can overlap.

As it is shown on Figure 3.4, this case can be reduced to the case from Figure 3.3 and then Algorithm 4 used to obtain relation f_0. Before Algorithm 4 can be applied however, lr-partition $P_2(Y_1)$ needs to be determined in Figure 3.4*b*. We assume that variables in the set Y_1 at the input to f_2' can assume any values, they are all don't cares. Therefore $P_2(Y_1)$ will consist of blocks labeled by the same labels as blocks of lr-partition $P_1(Y_1)$, each block will contain all the row numbers of the relation f_2.

After applying Algorithm 4 we still need to compute lr-partitions $\{P_0(Y_{1k})\}$. They can be computed by applying the algorithm used for computing lr-partitions $\{P_0(X_{1k})\}$ in Algorithm 4, and is repeated below for clarification.

```
!htp] Compute partitions $P_0(Y_{1k})$ [1]
every $P_1(Y_{1k})$ every $B_{c_{1i}(Y_{1k})} \in P_1(Y_{1k})$ $m \in B_{c_{1i}(Y_{1k})}$ Add row $m$ of $r[m][n]$ to $B_{c_{1i}(Y_{1k})} \in P_0(Y_{1k})$
```

end

Notice that the composition operation allows us to create lr-partitions representation not only from a set of cubes but also from a netlist, decision diagrams, or BLIF format description.

3.6 Time complexity analysis

3.6.1 Restriction (cofactor)

Time complexity analysis will be based on the number of basic set operations $\cap, \cup$ performed on lr-partition blocks B.

If the sets X_i,Y_j contain single binary variables the number of set intersection operations performed to compute $B_{c(X_1 \cup Y_1)}$ is equal to $|X_1| + |Y_1|$. Computation of lr-partition $P(x_j)$ ($P(y_j)$) requires two set intersections per binary variable $x_i \in X - X_1$ ($y_j \in Y - Y_1$). Hence, the total number of set intersection operations is equal to $|X_1| + |Y_1| + 2(|X - X_1| + |Y - Y_1|)) = 2(|X| + |Y|) - (|X_1| + |Y_1|)$ (if $X' \subseteq X''$ then $|X' - X''| = |X'| - |X''|$) where $|X|$ denotes the cardinality of set X.

In general, when X_i, Y_j contain arbitrary number of multiple-valued variables, the worst case computing $B_{c(X_1 \cup Y_1)}$ is when every variable $x_m \in X_1$ ($y_n \in Y_1$) is in a different set X_i (Y_j) and every block $B_{c(X_p)}$ ($B_{c(Y_p)}$) is a union of all the blocks in $P(X_i)$ ($P(Y_j)$). The number of set union operations (worst case) is therefore equal to $|X_1||P(X_i)| + |Y_1||P(Y_j)| = \sum_i |P(X_i)| + \sum_j |P(Y_j)|$. Computation of every $P(X_i)B_{c(X_1 \cup Y_1)}$ ($P(Y_j)B_{c(X_1 \cup Y_1)}$) requires $|P(X_i)|$ ($|P(Y_j)|$) block intersections. This totals to $\sum_i |P(X_i)| + \sum_j |P(Y_j)|$ block intersections, which is equal to the total number of lr-partition blocks.

Therefore, the worst case time complexity of the restriction operation is equal to $O(\sum_i |P(X_i)| + \sum_j |P(Y_j)|)$.

3.6.2 Containment

Let us consider first the situation when $|X_i| = |Y_j| = 1$, i.e. each set X_i, Y_j contains only one variable. The time complexity of selecting a block that corresponds to a variable value from the cube $c(X \cup Y)$ is $O(1)$. Each time the selection is made, intersection of the selected block with the current intersection result is performed. This process has to be repeated $|X| + |Y|$ times, hence, the total time complexity (expressed in terms of the number of basic set operations $\cup, \cap$ performed on lr-partition blocks) is equal to $O(|X| + |Y|)$. For sets X_i, Y_j containing more than one variable, the time complexity of computing $B_{c(X_i)}, B_{c(Y_i)}$ is equal to $O(|P(X_i)|), O(|P(Y_j)|)$, respectively. Hence, the total time complexity is equal to $O(\sum_i |P(X_i)| + \sum_j |P(Y_j)|)$ and can be smaller than $O(|X| + |Y|)$.

3.6.3 Complement

Complement operation of binary functions, as described in section 3.5.3, is performed by swapping block labels of every lr-partition $P(y_i)$ so that the blocks B_0 and B_1 will retain their contents but the labels change their values from 0 to 1 and from 1 to 0 respectively. Hence, the time complexity of the complement operation is $O(|Y|)$.

3.6.4 Composition

Let us first consider the simplest case of binary functions and all the sets X_i, Y_j containing single variables. Binary function implies two blocks (labeled with 0 and 1) per lr-partition. One variable per each of the sets X_i, Y_j implies the number of lr-partitions (number of sets X_i, Y_j) equal to the total number of variables. Let us also denote $n_1 = |C(X_1 \cup Y_1)|$ (the total number of row numbers of relation f_1) and $n_2 = |C(X_2 \cup Y_1 \cup Y_2)|$ (the total number of row numbers of relation f_2).

Step 1:

Computation of a partition products $P_1(X_3)P_1(Y_1)$, $P_2(X_3)P_2(Y_1)$ requires performing at most $|P_1(X_3)||P_1(Y_1)| + |P_2(X_3)||P_2(Y_1)|$ set intersections. In the worst case $X_3 = X_1$ or $X_3 = X_2$. If $X_3 = X_2$ then $X_2 \subseteq X_1$ and the above expression takes the form $|P_1(X_2)||P_1(Y_1)| + |P_2(X_2)||P_2(Y_1)| = 2(|P_1(X_2)| + |P_2(X_2)|) < 2(n_1 + n_2)$ for set Y_1 containing one binary variable. For the disjoint case ($X_3 = \emptyset$), $P_1(X_3 \cup Y_1) = P_1(Y_1)$, $P_2(X_3 \cup Y_1) = P_2(Y_1)$, the above expression simplifies to $|P_1(Y_1)| + |P_2(Y_1)| = 4$.

Step 2:

- Computation of a partition product $P_2(X_{2k})P_2(X_3 \cup Y_1)$ requires performing $|P_2(X_{2k})||P_2(X_3 \cup Y_1)|$ intersections. The worst case takes place if $X_3 = X_1$ or $X_3 = X_2$. Let us assume $X_3 = X_2$, then $|P_2(X_3 \cup Y_1)| = |P_2(X_2 \cup Y_1)| = n_2$ and $|P_2(X_{2k})||P_2(X_3 \cup Y_1)| = 2n_2$ for sets X_{2k}, Y_1 containing one binary variable each. This step has to be repeated the number of times equal to the number of sets X_{2k}. For single element sets X_{2k} that number is equal to $|X_2|$. Hence, the worst case number of "intersection" operations on blocks is equal to $2n_2|X_2|$. For the disjoint case ($X_3 = \emptyset$) we have $|P_2(X_{2k})||P_2(X_3 \cup Y_1)| = |P_2(X_{2k})||P_2(Y_1)| = 4$ and total number of "intersection" operations is equal to $4|X_2|$.
- Operation $c_{1i}(X_3 \cup Y_1) \subseteq c_{2j}(X_3 \cup Y_1)$ needs to be performed $|P_1(X_3 \cup Y_1)| \cdot |P_2(X_{2k} \cup X_3 \cup Y_1)|$ times. In the worst case when $X_3 = X_1$ or

$X_3 = X_2$ we have $|P_1(X_3 \cup Y_1)| \cdot |P_2(X_{2k} \cup X_3 \cup Y_1)| = |P_1(X_2 \cup Y_1)| \cdot |P_2(X_{2k} \cup X_2 \cup Y_1)| \leq n_1 n_2$ (we assume $X_3 = X_2$). One operation $c_{1i}(X_3 \cup Y_1) \subseteq c_{2j}(X_3 \cup Y_1)$ requires $|X_3 \cup Y_1|$ subset operations ($S_{1i} \subseteq S_{2i}$) so the number of set operations for one set X_{2k} is not greater than $n_1 n_2 \cdot |X_3 \cup Y_1|$ which is equal to $n_1 n_2 |X_2 \cup Y_1|$ in the worst case. This has to be repeated for every set X_{2k}. Hence the total number of $\subseteq$ operations for single variable sets X_{2k} is equal to $n_1 n_2 \cdot |X_2 \cup Y_1| \cdot |X_2| = n_1 n_2 \cdot (|X_2|^2 + |X_2|)$. Notice that for binary variables each $\subseteq$ operation is performed on sets of cardinality of at most two and is very fast (constant time) comparing to set operations on lr-partition blocks which may contain many elements. For the disjoint case ($X_3 = \emptyset$) we have $|P_1(X_3 \cup Y_1)| \cdot |P_2(X_{2k} \cup X_3 \cup Y_1)| = |P_1(Y_1)| \cdot |P_2(X_{2k} \cup Y_1)| = 8$ for sets X_{2k}, Y_1 containing one binary variable each. Hence the total number of $\subseteq$ operations for the disjoint case is equal to $8|X_2|$.

- Insertion of new row numbers rr into the blocks of lr-partition $P_0(X_{2k})$ has to be performed as many times as the number of different rr is, which is $n_1 n_2$ (maximum). This has to be repeated for every lr-partition $P_2(X_{2k})$. For single element sets X_{2k}, that number is equal to $|X_2|$. Hence, the number of "insert" operations into a block is equal to $n_1 n_2 \cdot |X_2|$.

Step 3:

For every block of lr-partition $P_1(X_{1k})$, every row number m contained in it implies inserting new row numbers from row m of table $r[m][n]$ into a corresponding block of lr-partition $P_0(X_{1k})$. The whole process has to be repeated for every $P_1(X_{1k})$ which is $|X_1|$ times for X_{1k} containing one variable each. For binary variables there are only two blocks (at most) per lr-partition, each containing at most n_1 row numbers. Therefore the worst case number of "insert" operations is equal to $2 \cdot n_1 \cdot |X_1|$. The worst case takes place in the situation when every block of lr-partition $P_1(X_{1k})$ contains all numbers

m. Since such situation doesn't happen very often the number of "insert" operations is usually much smaller.

Step 4:

Following the reasoning as in the previous step the number of "insert" operations is equal to $2 \cdot n_2 \cdot |Y_2|$.

Let as denote by $n_i, n_a$the number of operations required to perform set operations "intersect" and "insert" respectively. Then, the total number of operations required to perform composition in the worst case ($X_3 = X_2$) is equal to:

$$\begin{aligned} N = {} & 2(n_1 + n_2)n_i + \\ & + 2n_2|X_2|n_i + n_1n_2(|X_2|^2 + |X_2|) + n_1n_2|X_2|n_a + \\ & + 2n_1|X_1|n_a + \\ & + 2n_2|Y_2|n_a \end{aligned}$$

The total number of operations in the disjoint case ($X_3 = \emptyset$) is equal to:

$$\begin{aligned} N = {} & 4n_i + \\ & + 4|X_2|n_i + 8|X_2| + n_1n_2|X_2|n_a + \\ & + 2n_1|X_1|n_a + \\ & + 2n_2|Y_2|n_a \end{aligned}$$

For the sets represented by bit sets or hash tables for instance, the values for n_a and n_i are 1, and n respectively, where n is equal to n_1 or n_2. Hence, the time complexity of the compose operation is equal to $O(a)$ where $a = n_1n_2|X_2|^2$ for the worst case and $a = n_1n_2|X_2|$ for the disjoint case.

For the sets represented by binary trees the values for n_a and n_i are $\log n$ and n, respectively (a node corresponds to one element of the set and n is the number of nodes in a tree). Since the maximum number of elements in a set is n_2 (n_1), the time complexity of the compose operation in this case will be equal to $O(a \log_2 n_2)$.

In the case of BDD set representation, the elements are not stored in separate nodes of a tree. An element to be stored in BDD requires the number of nodes equal to the maximum number of bits needed to represent one element of the set and nodes can be shared between different elements (because of node sharing, the BDD representation can be more compact than the binary tree one). For BDD representation of a set, the values for n_a and n_i are n and n respectively, and $n = \log_2 n_2$ (or $\log_2 n_1$) is equal to the maximum number of bits needed to represent one element of the set. Hence, the time complexity of the compose operation will be equal to $O(a \log_2 n_2)$, which is the same as for the binary tree.

For the general case of multiple-valued directed relations and $X_{1k}, X_{2k}, Y_{1k}, Y_{2k}$ containing arbitrary number of variables, the following time complexities result:

Step 1:

The number of "intersection" operations on blocks:
maximum $|P_1(X_3)||P_1(Y_1)| + |P_2(X_3)||P_2(Y_1)|$,
disjoint case: $|P_1(Y_1)| + |P_2(Y_1)|$.

Step 2:

- The number of "intersection" operations on blocks: $|P_2(X_3 \cup Y_1)| \sum_k |P_2(X_{2k})|$
- The number of subset operations $S_{1i} \subseteq S_{2j}$: $|P_1(X_3 \cup Y_1)| \sum_k |P_2(X_{2k} \cup X_3 \cup Y_1)|$.
- The number of "insert" operations into a block: $n_1 n_2 \cdot |X_2|$.

Step 3:

The worst case number of "insert" operations: $n_1 \cdot \sum_k |P_1(X_{1k})|$.

Step 4:

The worst case number of "insert" operations: $n_2 \cdot \sum_k |P_2(Y_{2k})|$.

3.7 Memory Requirements

The starting point for lr-partitions representation is a relation or function given in a form of multiple-valued cubes (see Table 3.5). Then lr-partitions are built for selected subsets of input and output variables (see Example 3.2). Memory requirement for lr-partition representation depends on two main factors:

1. Selection of sets X_i and Y_j.
2. Representation of sets of cubes in the partition blocks.

3.7.1 Selection of sets X_i, Y_i

The data structure described by Theorem 3.4 gives us a lot of freedom in selecting sets X_i and Y_i. Since the partition block can be considered an atomic data structure the analysis will focus on minimizing the number of partition blocks required to represent a set of cubes. To simplify analysis let us assume that the number of values each variable can take is equal to m, each set X_i contains the same number of variables k, and sets X_i are disjoint. Then, for a completely specified function represented by minterms, the number of blocks n_B is equal to:

$$n_B = \frac{n}{k} m^k$$

where $n = |X|$, $\bigcup X_i = X$, and X_i are disjoint.

In the above formula m, n are constants so $n_B = f(k)$ which takes minimum value for $k = 1/\ln m$. Computation of k for different values of m results in $k = 1.44, 0.91, 0.72, \ldots$ for $m = 2, 3, 4, \ldots$. Since k has to be an integer number greater than 0 the best choice for k is $k = 1$ which leads to $\{\{P(x_1), \ldots, P(x_n)\}, \{P(y_1), \ldots, P(y_k)\}\}$ in Theorem 3.4.

Situation is different, however, if the relation is incompletely specified and the number of care cubes is a small fraction of the number of all minterms specifying the relation (as it is often the case for ML data). For instance, if the number of care cubes grows linearly with the number of input variables, n_B will be described by the following formula:

$$n_B = \frac{n}{k} K n$$

where K is a constant.

The value of n_B in this equation decreases if the value of k increases. Since k can not be greater than n, n_B takes minimal value for k equal to n and the set of partitions in Theorem 3.4 reduces to $\{P(X), P(Y))\}$. In this case the number of blocks and their size (one element) are small but the storage required for labels (cubes) increases and becomes the primary factor determining the memory requirement.

Between these two extreme cases there are many other possible choices for X_i and Y_j. Selection of sets X_i and Y_i can for instance be done based on some heuristic measures of closeness of variables. Such operation would correspond to definition of a higher level abstraction and can easily by represented by lr-partitions. Example 3.2 on page 36 illustrates the two cases described above and shows that a relation represented with partitions $P(X), P(Y)$ has smaller number of blocks than the one represented with partitions based on single variables.

3.7.2 Set representation

All operations on lr-partitions are *set operations* on the corresponding sets of symbols (integer numbers in particular) representing blocks. Therefore, any computer package for representing and manipulating sets (and in particular any decision diagrams package that allows set-theoretical operations), can be used to implement lr-partitions with no modification: for instance the packages for BMDDs, EVDDs, KFDDs, K*BMDs, ZBDDs, etc. [87]. In this chapter we present results of comparison of two set representations: Bit Sets (BS), and Binary Decision Diagrams (BDDs). Notations lr-BDD and lr-BS will refer to lr-partition with blocks represented by BDDs and BSs respectively. Notation BDD will refer to representation of a function by a single BDD.

Binary Decision Diagrams (BDD)

One of the most efficient representations of a large set of objects is a decision diagram data structure, in particular *Binary Decision Diagram* (BDD), which has been very successfully applied to the representation of large binary functions [18].

A question of comparison of BDDs and cube arrays is a much discussed one in logic synthesis [32, 118]. It is well-known that there are functions, such as parity, for which BDDs are obviously better, and there are other functions, such as the one shown by Devadas [32] (or that occur in ML, logic or controller design [118]), that are more efficiently described using an array of cubes. Let us analyze these two extreme cases.

One extreme example is a completely specified binary function, similar to parity, and with many input variables. Obviously, in this case, a BDD is better than an array of cubes because it has a polynomial number of nodes while the array of cubes has an exponential number of cubes. In this case the minterm symbols for lr-partitions are selected to be integer numbers equal to the decimal values of the corresponding minterms. Thus the size of the output variable block labeled by 1 is the same as that of the BDD for this function. All the input blocks have one node each. Hence, both representations are comparable in space.

For the other extreme case, let us consider a binary function like those discussed by [32] that have polynomial number of cubes and exponential number of nodes in BDD. If the function is specified with cubes, it has n variables and k cubes, $k << 2^n$. Very conservatively estimating: in the worst case there is $2(n+1)$ partition blocks, each represented by a BDD with k nodes (for $\{\{P(x_i), \{P(y_j)\}\}$). So, the total number of nodes is $O(2nk)$ while the number of nodes in the single BDD representation would be $O(2^n)$.

It is advantageous that with a good selection of a cube symbol encodings in these two extreme cases the lr-partitions with blocks represented by BDDs *are comparable in size to the better representation of the two: arrays of cubes, or BDDs.*

Examples of multi-output multiple-valued relations can be constructed for which the advantage over MVDDs would be dramatic for large values of n and k. There exist practical functions with similar, although not that extreme, properties [118]. To this category belong functions with many cubes and many variables, but with still very small ratio of cares to don't cares. This is the kind of functions from ML benchmarks [121], but with larger k, n and number of terms. It is our hope that for larger multi-valued functions or relations the storage advantage of lr-partition representation will be even more clearly observable. Such benchmarks, however, although they exist, are not included into popular sets of benchmarks such as ISCAS or MCNC [81].

Bit Sets (BS)

To be able to store any subset of a set of n elements, BS is represented by a vector of n bits. If the i-th element of the set is contained in the subset, i-th bit of the BS is set to 1, otherwise it is set to 0. Hence, the memory requirement is $\lceil n/8 \rceil$ bytes.

3.8 Experimental results

Notations lr-BDD and lr-BS will refer to lr-partition representations with BDDs and BSs representing partition blocks respectively. Notation BDD refers to representing a function by a single BDD. The size of lr-BS is computed according to the following formula:

$$\text{size} = \left\lceil \frac{\#\text{ of cubes} \cdot \#\text{ of partition blocks}}{8} \right\rceil \text{[bytes]}$$

Where the number of partition blocks for binary function is double the number of input and output variables (two blocks, labeled 0 and 1, per variable). To compute lr-BS/BDD we assume that one BDD node requires 22 bytes of memory [16]. All the tests were performed on DECstation 5000/240 with 64MB of RAM. Times are user times measured with accuracy of 1/10 second by the Unix command

`/bin/time` and are given in seconds. For lr-BDD and BDD representations we used U.C. Berkeley BDD package with sifting variable reordering method. All lr-partitions were based on single variables $\{\{P(x_1), P(x_2), \ldots\}, \{P(y_1), P(y_2), \ldots\}\}$.

3.8.1 Binary functions

The testing has been performed on four types of functions: parity, Devadas, multiply, and on two-level MCNC benchmarks [81]. The reason for using two-level MCNC benchmarks is that current implementation of lr-BDD accepts input data only in the form of multiple-valued cubes similar to Espresso format. This is a natural representation of input data in ML and controller design problems.

The examples analyzed in this section are completely specified binary functions (to allow for comparison with BDD representation) specified by sets of cubes. Lr-partitions however, allow for representation of multiple-valued functions and relations which can be incompletely specified.

Parity functions

For *parity* functions linearly-sized BDD representation ($2n-1$ nodes) can always be found. As the comparison of BDD and lr-BDD representations presented in Table 3.10 shows, lr-BDD is equally good ($2n$ nodes) for this type of functions. The lr-BS representation however, compares poorly to both BDD and lr-BDD. This is due to particularly good compression capabilities of BDDs for this type of functions.

In Table 3.9, time to create lr-BDD representation is roughly twice as long as for BDD and this ratio remains constant when the function size increases. The time to create lr-BS however is much shorter than for the two other cases.

Devadas functions

Another type of functions to be tested were the ones discussed in [32]. The function has $2n + \log n$ inputs and n^2 product terms in sum-of-product repre-

	i/o	# cubes	t_{BDD} [s]	t_{lr-BDD} [s]	t_{lr-BS} [s]
p9	9/1	512	0.3	0.6	0.0
p10	10/1	1024	0.6	1.6	0.1
p11	11/1	2048	1.5	3.6	0.2
p12	12/1	4096	3.7	8.3	0.5
p14	14/1	16384	19.2	40.8	2.7
p16	16/1	65536	97.1	194.8	11.8

Table 3.9: Parity functions: construction time.

	lr-BDD nodes	BDD nodes	$\frac{\text{lr-BS}}{\text{BDD}}$	$\frac{\text{lr-BDD}}{\text{BDD}}$
p9	18	17	3.42	1.06
p10	20	19	6.74	1.05
p11	22	21	13.30	1.05
p12	24	23	26.31	1.04
p14	28	27	103.43	1.04
p16	32	31	408.40	1.03

Table 3.10: Parity functions: size.

sentation and $O(2^{n/2})$ nodes in BDD representation under any possible variable ordering. The functions d8, d10, d11, d12 in Tables 3.11 and 3.12 correspond to $n = 8, 10, 11,$and 12.

In terms of memory requirement both lr-BS and lr-BDD are better than BDD in this case. However, lr-BS memory requirement increases with the number of cubes and eventually may become greater than BDD. On the other hand, lr-BDD to BDD ratio decreases with the number of cubes. The larger the number of cubes the better lr-BDD comparing to BDD.

The time $t_{\text{lr}-\text{BDD}}$ needed to construct lr-BDD increases slower than the time t_{BDD} needed to build BDD and the ratio $t_{\text{BDD}}/t_{\text{lr}-\text{BDD}}$ increases from 1.79 for d8 to 4.76 for d12. The time $t_{\text{lr}-\text{BS}}$ to create lr-BS is, as before, much shorter than for BDD and lr-BDD.

As we can see from the results of testing on Devadas functions, BDD doesn't compare well to any of lr-BDD and lr-BS.

	i/o	# cubes	t_{BDD} [s]	t_{lr-BDD} [s]	t_{lr-BS} [s]
d8	19/1	2038	23.3	12.8	0.3
d10	24/1	10308	251.2	104.8	2.7
d11	26/1	22631	831.6	249.9	7.0
d12	28/1	49151	2984.5	632.3	16.4

Table 3.11: Devadas functions: construction time.

	lr-BDD nodes	BDD nodes	$\frac{\text{lr-BS}}{\text{BDD}}$	$\frac{\text{lr-BDD}}{\text{BDD}}$
d8	1743	1610	0.29	1.08
d10	3372	5331	0.55	0.63
d11	2403	16445	0.42	0.15
d12	11930	20784	0.78	0.57

Table 3.12: Devadas functions: size.

Multiplier functions

Another type of function is n-bit *multiplier* function which requires $O(2^{n/8}) = O(1.09^n)$ nodes in single BDD representation [19]. Functions m6, m7, m8, and m9 in Table 3.14 correspond to $n = 6, 7, 8$ and 9. As it can be seen from Table 3.14 the size of lr-BDD increases slower than that of BDD. This would indicate that the number of nodes of lr-BDD is less than $O(1.09^n)$.

The time needed to construct lr-BDD is about 1.5 times smaller than for BDD for multiplier functions in Table 3.13. For lr-BS that factor is even greater and is of order of 30.

MCNC benchmarks

The result of testing on MCNC benchmarks [81] is shown in Tables 3.15 and 3.16. The lr-BS representation appears to be smaller than BDD representation in

	i/o	# cubes	t_{BDD} [s]	t_{lr-BDD} [s]	t_{lr-BS} [s]
m6	12/12	4096	11.0	22.4	1.0
m7	14/14	16384	75.0	124.8	5.1
m8	16/16	65536	461.2	663.7	23.2
m9	18/18	262144	2419.5	3930.9	108.3

Table 3.13: Multiplier functions: construction time.

	lr-BDD nodes	BDD nodes	$\frac{\text{lr-BS}}{\text{BDD}}$	$\frac{\text{lr-BDD}}{\text{BDD}}$
m6	1109	1103	1.01	1.0050
m7	3116	3109	1.68	1.0020
m8	8849	8841	2.70	1.0009
m9	25063	25054	4.20	1.0004

Table 3.14: Multiplier functions: size.

73% of cases. The lr-BDD representation however, is larger than BDD in most of the cases.

For most of the functions in the table the number of cubes is much smaller than the number of minterms required to represent the same functions and, as the analysis in Section 3.7.1 shows, lr-partitions based on sets of variables ($P(X_i)$) instead of single variables ($P(x_i)$) should be less memory consuming here.

BDD representation for benchmarks `apex3` and `seq` failed to terminate successfully as it didn't fit into computer memory. The lr-BDD representation terminated without any problem for the same benchmarks. This would indicate that lr-BDD is less memory consuming when creating the representation. This can be explained by the fact that lr-BDD consists of many small shared BDDs which are incrementally created and processed while reading the data. On the other hand, BDD representation consists of one large DAG which may temporarily grow beyond capacity of the available memory while reading the data and performing necessary transformations.

	i/o	# cubes	t_{BDD} [s]	t_{lr-BDD} [s]	t_{lr-BS} [s]
apex1	45/45	1440	82.4	31.0	0.8
apex2	39/3	1576	42.2	31.0	0.6
apex3	54/50	1036	-	20.1	0.7
apex4	9/19	1907	1.9	13.8	0.3
apex5	117/88	2710	7.2	120.9	4.6
seq	41/35	2014	-	47.5	1.0
table3	14/14	1686	1.8	19.5	0.3
table5	17/15	1600	1.7	21.8	0.3
cps	24/109	855	3.2	10.3	0.5
cordic	23/2	2105	4.0	14.0	0.5
duke2	22/29	404	0.7	4.1	0.1
e64	65/65	327	3.5	5.0	0.2
ex1010	10/10	1297	2.4	9.7	0.1
ex4p	128/28	654	2.5	19.8	0.8
misex2	25/18	101	0.1	0.5	0.0
misex3	14/14	1391	4.0	15.5	0.2
misex3c	14/14	1566	2.3	16.3	0.3
pdc	16/40	822	1.6	7.9	0.2
spla	16/46	837	1.3	8.5	0.2
t481	16/1	841	0.7	4.5	0.1
vg2	25/8	304	0.9	2.8	0.0
alu4	14/8	1184	2.6	12.4	0.2
5xp1	7/10	141	0.0	0.4	0.0
9sym	9/1	158	0.1	0.6	0.0
bw	5/28	93	0.1	0.2	0.0
clip	9/5	271	0.2	1.3	0.0
ex5p	8/63	208	0.4	0.8	0.0
inc	7/9	94	0.0	0.3	0.0
rd53	5/3	67	0.0	0.1	0.0
rd73	7/3	274	0.1	1.1	0.0
rd84	8/4	511	0.2	2.3	0.0
sao2	10/4	137	0.1	0.5	0.0

Table 3.15: MCNC benchmarks: construction time.

	lr-BDD nodes	BDD nodes	$\frac{\text{lr-BS}}{\text{BDD}}$	$\frac{\text{lr-BDD}}{\text{BDD}}$
apex1	4877	1345	1.09	3.63
apex2	5594	730	1.03	7.66
apex3	3384	-	-	-
apex4	4012	892	0.68	4.50
apex5	7435	1130	5.59	6.58
seq	6202	-	-	-
table3	4311	778	0.69	5.54
table5	4387	711	0.82	6.17
cps	2550	1072	1.21	2.38
cordic	2136	61	9.84	35.02
duke2	1364	392	0.60	3.48
e64	918	229	2.12	4.01
ex1010	3605	1314	0.23	2.74
ex4p	2952	535	2.17	5.52
misex2	369	78	0.65	4.73
misex3	4616	695	0.64	6.64
misex3c	3976	499	1.00	7.97
pdc	2820	609	0.86	4.63
spla	2598	576	1.03	4.51
t481	1368	32	5.12	42.75
vg2	1139	301	0.38	3.78
alu4	3478	800	0.37	4.35
5xp1	388	55	0.51	7.05
9sym	485	26	0.70	18.65
bw	228	105	0.34	2.17
clip	759	92	0.47	8.25
ex5p	562	242	0.69	2.32
inc	265	68	0.26	3.90
rd53	180	19	0.34	9.47
rd73	661	35	0.91	18.89
rd84	928	48	1.45	19.33
sao2	459	89	0.26	5.16

Table 3.16: MCNC benchmarks: size.

3.8.2 Multiple-valued functions

Table 3.17 shows a comparison of selected benchmarks from [121] and [17] in terms of memory requirements for representation of partition blocks by BDDs and BSs. Value in column 4 (part blocks) is equal to the total number of partition blocks for a given benchmark.

	i/o	cubes	part blocks	lr-BDD nodes	$\frac{\text{lr-BS}}{\text{lr-BDD}}$
zoo	16/1	101	46	412	0.07
shuttle	6/1	15	18	43	0.04
breastc	9/1	699	92	3638	0.10
balance	4/1	625	23	652	0.13
lenses	4/1	24	12	23	0.07
trains	32/1	10	107	98	0.10
trains20	29/1	20	109	185	0.08
car	6/1	1728	25	1163	0.21
employ1	7/1	18000	33	4292	0.79
employ2	9/1	9600	31	1802	0.94
programm	12/1	20000	47	70447	0.08

Table 3.17: Multiple-valued benchmarks.

As it can be seen from Table 3.17, in all the cases lr-BS is smaller than lr-BDD, even for functions with a large number of cubes. However, the ratio lr-BS/lr-BDD not only depends on the number of cubes but also on the structure of the function. For instance, function `employ2`, which is much smaller than `programm`, has lr-BS/lr-BDD = 0.98, much larger than the value of 0.08 for the `programm` function. If that ratio depended only on the number of cubes the relation would have been opposite.

3.9 Summary

We have presented a new data structure (lr-partitions) and shown that it can be used to represent not only binary functions but also a broader class of multiple-valued, completely and incompletely specified relations (functions in particular) which are typical in Machine Learning and complex FSM controller optimization applications. It can easily be used to represent distributed data sets for distributed data mining applications.

Characteristics of lr-partition representation can be summarized as follows:

- lr-partitions representation is a natural representation for distributed data bases with vertically partitioned datasets (in opposite to decision diagrams and cubes representations).
- lr-BDD based on single variables ($P(x_i)$, large partition blocks, small labels) have characteristics similar to BDD representation. If lr-BDD is based on larger sets ($P(X)$, small partition blocks, large labels) then they more resemble cube representation of the function.
- Multiple values of both input and output variables can be easily represented. This is especially important in ML and complex FSM controller optimization applications to express uncertainty of choice of variable's values.
- It can easily handle situations where a variable is not present in a given cube (Michalski's train benchmark [85] and '~' in Espresso format).
- By selection of sets X_i and Y_j, lr-partitions can be dynamically adjusted to a given type of data (completely vs. incompletely specified, many cubes vs. few cubes) to minimize memory requirements.
- by selecting a set representation for lr-partition blocks, characteristics of the whole representation can be significantly changed. Two such representations, BDDs and BSs, have been compared in this chapter, but other representations (OBDDs, BMDDs, EVDDs, KFDDs, hash tables, etc.) can be used too.
- by making use of the existence of repetitive parts of data cubes, lr-partitions can facilitate factorization and decomposition of multiple-valued functions

and relations, especially in the presence of many don't cares; we implemented a decomposer of multiple-valued relations which can decompose large functions and relations from ML and controller domains. It was shown in [97] that this representation is not only compact but also allows for a fast processing.

- lr-partition representation constructed from a canonical representation (BDD, canonical cubes [9] [8][10][11], minterms with specified order) can be made canonical.

Comparison of lr-partitions, with BS representing partition blocks (lr-BS), and single BDD shows superiority of lr-partitions in most of the binary MCNC test cases (73% of the benchmarks), lr-BS was also smaller than lr-BDD on all the multiple-valued benchmarks. If the number of cubes describing a function or relation is large (tens of thousands) then representing partition blocks with BDDs (lr-BDD) is usually less memory consuming than with BSs (lr-BS).

We believe therefore that *Labeled Rough Partitions* are a new and very promising general purpose data structure for large binary and multiple-valued functions and relations especially in distributed data and computations environment.

Chapter 4

DECOMPOSITION

4.1 Introduction

The idea of decomposition of a complex system into an organized set of simpler subsystems to simplify the system description is not new. In cognitive science it is known as **Functional Analysis** [28]. The basic idea is that the system is viewed as computing a function. Functional analysis is a process of decomposing that function into a structure of simpler subfunctions in a hope that the result will be easier to explain (Occam razor principle). Each subfunction can be viewed as a definition of a certain concept[1].

There are two main approaches to the analysis of complex systems: **probabilistic** and **non-probabilistic**.

Probabilistic approach requires an existence of a global probability distribution over the variables of the system. In practice the true probability distribution is rarely available and we can only collect frequencies associated with the combinations of system variables values (tuples). These frequency values can be normalized to the total number of observations and used for approximation of the true probability distribution over the variables of the system. The decomposition of probabilistic systems consists on determination of a set of simplest marginal probabilities describing the system.

Non-probabilistic approach results from the situations where collecting a statistically reliable information on the global frequency (probability) distribution may be impossible or unreasonable. The decomposition in this case consists on determination of a set of simplest possible relations describing the system.

[1]Webster: Concept - an idea of what a thing in general should be

A system is called **directed** if there exist a dependency relation between groups of variables, one set of variables acts on another set of variables, the values of some variables depend on the values of some other variables. If the system is not directed it is called **neutral** (see also Definitions 3.14 and 3.15 p.30).

A directed relation is called a **function** if the dependent variables can not take different values for the same combinations of independent variables. The notions of directed relation and function were constructed by imposing constraints on the most general notion of relation.

The decomposition of complex systems was analyzed by many researchers in the past. In the terminology of general systems science both decomposition and composition are known under the name of **reconstructability analysis** [52].

The approach presented by Ashby [6], Klir [51], Krippendorff [61], and Conant [26] consists of generating a lattice of possible decomposition structures and evaluating them in terms of both complexity and accuracy using either **set-theoretic** (non-probabilistic approach, Hartley's uncertainty [45]) or **information-theoretic** (probabilistic approach, Shannon's uncertainty [112], transmission and derivatives) measures. A structure that results in the smallest complexity and yet describes the data with a high accuracy is selected to be a best solution. Reconstructability analysis of directed systems was further clarified by Zwick in [135].

An excellent, concise overview of decomposition approaches developed within the framework of general systems methodology (reconstructability analysis) is presented in [137] and extended bibliography of reconstructability analysis as a whole is provided in [1]. Some additional details on set-theoretic approach to reconstructability analysis are presented in [27] [136].

The approaches presented above can be characterized as using uncertainty measures as a main tool for discovering and extracting simple subsystems from a complex system. The number of system variables remains unchanged in the process of decomposition. A distinctively different approach to the decomposition of complex systems emerged as an extension of approaches used for the decomposition of binary functions. A broader range of methods is used here (mostly non-probabilistic) to decompose complex structures and new variables are introduced to the resulting structure to achieve further complexity reduction.

The theory of decomposition of binary functions [7], [29] was extended to multiple-valued, completely and incompletely specified functions by Karp [50]. Lendaris and Stanley use Ashenhurst-Curtis type decomposition to construct a cascade of functional blocks (concepts) matching given data [71]. The approach presented by Walliuzzaman [125] resembles Curtis approach for binary functions [29] but when Curtis investigates all possible decompositions to select the best one, Walliuzzaman develops a set of conditions for easy selection of decomposition that leads to a simple solution. Abugharrbieh and Lee [3] and [4] extend Shen's algorithm for binary functions [113] on the multiple-valued functions. Their method operates on functions that may be given either by truth tables or algebraic expressions. Luba [75] uses partition based method to decompose multiple-valued functions. In general, Ashenhurst-Curtis type decompositions correspond to extracting new concepts from data while the approach presented by Fang and Wojcik, to describing data in terms of predefined, existing concepts.

All the above algorithms start from an initial function and decompose it step by step by extracting smaller subfunctions. Another, compositional approach to decomposition of multiple-valued functions is presented by Fang and Wojcik [37]. In their approach the original function is expressed by a composition of subfunctions taken from a library of already predefined functions. Another approach presented in [40], [34], [47] [48], [49] is based on representation of multiple-valued functions in terms of MTMDD (multi-terminal, multiple-valued decision diagrams). MTMDD approach was used only for disjoint decomposition of completely specified functions (with the exception of [40] where disjoint decomposition of incompletely specified functions is presented). Most of these algorithms present decomposition results for functions with no more than 10 input variables and they assume that functions are homogeneous (all the variables are of equal cardinality). MTMDD based methods were used for disjoint decomposition only.

Comprehensive review of existing methods, and presentation of new ideas for functional decomposition of binary, multiple-valued and continuous functions is presented in [96] and [94]. Based on that work new algorithms for functional decomposition of incompletely specified, multiple-valued functions were developed and implemented in program GUD [95]. GUD was the first program able to perform multi-level decomposition of large, incompletely specified, multiple-valued functions. Some of the ideas presented in [96], [94], and [95] were next implemented in program HINT [134][133]. Theory and implementation of the first multi-level decomposer (MVGUD) for large multiple-valued directed relations were presented in [97].

In this chapter we will present algorithms for decomposition of non-probabilistic directed and neutral relations as well as probabilistic neutral relations. These three approaches will be analyzed separately as they require slightly different algorithms.

The method of decomposition of directed relations described in this dissertation follows the main ideas from [97] but uses a different data structure to represent relations. It transforms a multiple-valued incompletely specified function or relation into a multi-level structure and doesn't depend on particular assumptions about the nature of the blocks of the structure. The transformation process is based on Ashenhurst-Curtis type serial decomposition and introduces new variables to reduce the cost of the final solution.

One step of Ashenhurst-Curtis type decomposition consists of forming a description of the initial relation $f_0(X_0)$ in terms of other, less complex relation $f_1(X_1)$ and input variables X_2:

$$y_0(X_0) = f_2(f_1(X_1), X_2)$$

where X_1, X_2 are sets of input variables and $X_0 = X_1 \cup X_2$, sets X_1 and X_2 may overlap. If X_1 and X_2 overlap ($X_1 \cap X_2 \neq \emptyset$) decomposition is called non-disjoint, otherwise it is called disjoint.

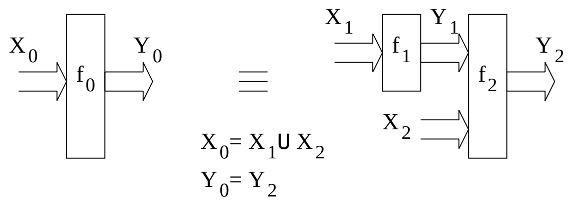


Figure 4.1: One level serial decomposition: $f_0(X_0) = f_2(f_1(X_1), X_2)$.

Original relation $f_0(X_0)$ is being represented in terms of variables of a new representation space $\{Y_1, X_2\}$, more suitable for the problem description. The initial representation space $X_0 = X_1 \cup X_2$ was divided into two subspaces (not necessarily disjoint) and one of them used to define a new concept $f_1(X_1)$ (see Figure 4.1). The selection of X_1, X_2, and determination of f_1 is carried out in such a way as to minimize the overall complexity measure of the result. According to the general Occam razor principle this should result in better generalization properties of the final solution. The decomposition process is repeated iteratively until the terminating criteria are met. At each decomposition level a local optimization is performed and the resulting blocks represent relations (functions in particular). Once the decomposition is terminated these relations may provide additional choices for the final, global optimization.

An example of a relation with binary inputs and a single multiple-valued output is shown in Table 4.1. Observe, that not all the combinations (minterms) of the input variable values are present in the table. Values in the column corresponding to the output variable y_1 include the so-called *set-values* defined in Section 3.2. If, for instance, the values the variable y_1 can take have the following meanings: 0 - a chair, 1 - an armchair, 2 - a desk, 3 - a table, 4 - a bench, then the set-value {0,1} in the first row means “a chair or an armchair”. The value 0 means a definite answer “a chair”, and the set-value {0,1,2,3,4} corresponds a complete unknown, often referred to as **don’t care** and denoted by “-”. Minterms for which the value of the dependent variable y_1 is unknown (don’t care) are not stored in the table (there is no row for the minterm $x_1x_2x_3x_4 = 1111$). In general, for the relations

	x_1	x_2	x_3	x_4	y_1
0	0	0	0	0	0,1
1	0	1	0	0	1,2
2	1	1	0	0	0
3	0	0	0	1	0,3
4	1	1	0	1	0,3
5	1	0	0	1	0,4
6	1	0	0	0	0,3
7	0	0	1	1	1,3
8	0	1	1	1	0,1
9	0	0	1	0	2,3
10	1	0	1	0	1,4
11	0	1	1	0	2,3

Table 4.1: Multiple-valued relation.

with an arbitrary number of dependent variables, the minterm is not stored only if the values of all the dependent variables for this minterm are equal to *don't care*.

In multiple-valued systems, the entire classical decomposition approach is considerably more complex than for binary systems because of the associated combinatorial explosion. However, this is not the case for weakly specified relations and functions, and an appropriate decomposition approach can be made efficient by utilizing *don't cares*. It can be observed that in the area of circuit design the percent of *don't cares* is not more than 90%. In Machine Learning, this percent is usually larger than 99%. Arbitrarily, we will define the relations with more than 95% *don't cares* to be weakly specified (they will be also referred to as strongly unspecified relations). Let us also observe, that the greater the cardinality of a *don't care* is ($\{0,1\}$ vs. $\{0,1,2\}$ for instance), the more strongly unspecified relation is. The smaller the cardinality of a *don't cares* is, the more the relation resembles a function.

The chapter is organized as follows: Section 4.2 presents decomposition of multiple-valued functions and directed relations (non-probabilistic), Section 4.3 presents a new approach to decomposition of probabilistic neutral relations, Section 4.4 presents an approach to decomposition of non-probabilistic neutral relations, Section 4.5 introduces cost measures used for the evaluation of decomposed structures, Section 4.6 discusses optimization issues related to minimization of the cost of decomposed structures, Section 4.7 presents examples of decomposition of various multiple-valued and binary relations, and Section 4.8 summarizes the chapter.

4.2 Decomposition of non-probabilistic directed relations

In this section we only consider a decomposition of directed relations and functions that are discrete and non-probabilistic. The method presented here performs a multi-level Ashenhurst-Curtis type decomposition [7, 29]. One decomposition step consists of forming a description of an initial relation $f_0(X_0)$ in terms of another, less complex relation $f_1(X_1)$ and variables from the set X_2 (Figure 4.1 p.72).

$$f_0(X_0) = f_2(f_1(X_1), X_2)$$

The relation $f_1(X_1)$ is extracted (discovered) in the decomposition process and introduces a new variable to the original set of system variables.

Lemma 4.1 (conditions for $P(X) \leq P(Y)$) *Let the directed relation $Y = f(X)$ be defined by a set of input-output cubes $C(X \cup Y)$. Then $P(X) \leq P(Y)$ iff for every subset of input cubes $\{c_i(X)\} \subseteq C(X)$ such that their intersection $c_k(X)$ is a proper cube, the intersection of the output cubes $\{c_j(Y)\} \subseteq C(Y)$ corresponding to the input cubes in $B_{c_k(X)}$ is also a proper cube.*

Proof

1. *Let us assume that $P(X) \leq P(Y)$. According to Definitions 3.16 (p.31) and 3.8 (p.29) the intersection of block cubes of $P(X)$ is not empty iff there exists a cube $c_k(X)$ which is contained in every cube of that block. The same is true for blocks of $P(Y)$. By Definition 3.18 (p.33) each block of $P(X)$ is contained*

in at least one block of $P(Y)$. Since $P(X)$ and $P(Y)$ both partition the same set of cubes $C(X \cup Y)$ then for every $c(X)$ in the block B of $P(X)$ there exists $c(Y)$ in a block of $P(Y)$ such that $c(X)$ and $c(Y)$ are part of the same input-output cube $c(X \cup Y)$.

2. *Let us now assume that for every subset of input cubes $\{c_i(X)\} \subseteq C(X)$ such that their intersection $c_k(X)$ is a proper cube, the intersection of the output cubes $\{c_j(Y)\} \subseteq C(Y)$ corresponding to input cubes in $B_{c_k(X)}$ is also a proper cube. If the intersection $c_k(X)$ of a subset of the input cubes is proper, then, by Definition 3.16 (p.31), $c_k(X)$ defines an lr-partition block $B_{c_k(X)}$ containing that set of cubes. The same is true for the subset of output cubes. $B_{c_k(X)}$ contains the largest subset of input cubes having $c_k(X)$ as its intersection. The corresponding block of $P(Y)$ contains at least the same cubes as the block $B_{c_k(X)}$ does. Hence by Definition 3.18 (p.33) $P(X) \leq P(Y)$.*

Corollary 4.1 *If the directed relation $Y = f(X)$ is defined by a set of input-output cubes $C(X \cup Y)$ and for every $c_i(X), c_j(X) \in C(X)$, cubes $c_i(X), c_j(X)$ are disjoint then $P(X) \leq P(Y)$.*

PROOF *Since input cubes are disjoint then by Definitions 3.16 (p.31) and 3.17 (p.33) $P(X)$ consists of single-element blocks only and by Definition 3.18 (p.33) is not greater than any other partition defined on the set of cubes $C(X)$. Hence, since $C(X)$ and $C(Y)$ are isomorphic, $P(X) \leq P(Y)$.*

For functions, if a subset of input cubes $\{c_i(X)\} \subseteq C(X)$ has a proper cube intersection then the corresponding output cubes are equal. This means that the conditions of Lemma 4.1 are always satisfied and for every function $Y = f(X)$ the condition $P(X) \leq P(Y)$ is true.

For relations, the set of input-output cubes defining a relation may not always satisfy the conditions of Lemma 4.1 but it can always be converted to a form which satisfies these conditions. For instance two input-output cubes $x_1^1 x_2^0 y_1^0$ and $x_1^1 x_2^0 y_1^1$

which input parts are equal and output parts do not have proper cube intersection can be combined into one cube $x_1^1 x_2^0 y_1^{0,1}$ that satisfies the conditions of Lemma 4.1. In general the transition to the form satisfying conditions of Lemma 4.1 can always be done by making the cubes disjoint (by using sharp operation for instance) and combining cubes with identical input parts.

The following theorem states the conditions for the existence of serial decomposition:

Theorem 4.1 (decomposition) *Let the relation $Y_0 = f(X_0)$ be defined by a set of input-output cubes $C(X_0 \cup Y_0)$ and bound and free sets X_1, X_2, $X_1 \cup X_2 = X_0$, be given. Then the set of cubes $C(X_0 \cup Y_0 \cup Y_1)$ defines a serial decomposition $Y_2 = f_2(f_1(X_1), X_2)$, $Y_2 = Y_0$, $Y_1 = f_1(X_1)$, of the relation $Y_0 = f_0(X_0)$ if the following conditions are satisfied:*

$$P(X_1) \leq P(Y_1), \textit{and}$$
$$P(X_2)P(Y_1) \leq P(Y_2)$$

PROOF *Condition $P(X_1) \leq P(Y_1)$ means that $Y_1 = f_1(X_1)$ is either a function or relation satisfying the conditions of Lemma 4.1. Condition $P(X_2)P(Y_1) \leq P(Y_2)$ means that $Y_2 = f_2(X_2 \cup Y_1)$ (Theorem 3.2) is either a function or relation satisfying the conditions of Lemma 4.1. Since relation $\leq$ on lr-partitions is transitive (if $P(X) \leq P(Y)$ and $P(Y) \leq P(Z)$ then $P(X) \leq P(Z)$) and the partition product is always less or equal to any of its components so from the first condition the second condition of the theorem can be rewritten as $P(X_2)P(X_1) \leq P(Y_2)$. By Theorem 3.2, $P(X_2)P(X_1) = P(X_1 \cup X_2) = P(X_0)$. Since $Y_2 = Y_0$ then the second condition can be finally reduced to $P(X_0) \leq P(Y_0)$ which means that relations $Y_1 = f_1(X_1)$ and $Y_2 = f_2(X_2 \cup Y_1)$ can be combined into a function or relation $Y_0 = f(X_0)$ satisfying the conditions of Lemma 4.1.*

According to Theorem 4.1 the decomposition problem can be reduced to the problem of finding an lr-partition $P(Y_1)$ satisfying the conditions of the theorem. It is easy to check however, that $P(Y_1) = P(X_1)$ always satisfies the conditions of the theorem. Is this the solution we are looking for? The answer is no. Our goal is to find a structure which complexity is smaller than the complexity of the original block. If $P(Y_1) = P(X_1)$ then the block f_1 of the decomposed structure represents a direct connection of variables from the set X_1 to the input of the block f_2. Therefore the block f_2 is equal to f_0 and the decomposed structure is identical to the original one. To evaluate the cost of decomposed structure and compare it to the original block an appropriate cost measure is needed. The discussion of various cost measures and their comparison will be provided in Section 4.5 (p.87). The problem of determination of $P(Y_1)$ satisfying conditions of Theorem 4.1 and minimizing the complexity will be discussed in Section 4.6.1 (p.104).

Algorithm 6 presents implementation details of the decomposition strategy. The program inputs are the relation to be decomposed R_0 and the minimum size of the block R_1 *user_min_size* to be extracted from R_0. The value of *user_min_size* characterizes the smallest decomposition unit and in the current version of the program is defined by the number of block inputs.

!t] Decomposition strategy

[1] ***Input:*** *$R : X \rightarrow Y$ relation to be decomposed user_min_size user specified minimal relation size* ***Output:***
S_{final} set of the relations after decomposition
$S_R = R$ S_R is a set of the relations to be decomposed $S_{final} = \emptyset$
S_R is not empty size = user_min_size R_0 = next_element(S_R) sizeof(R_0) > size (R_1, R_2) = decompose_optimized(R_0,
size) add R_1 and R_2 to S_R $R_1 = EMPTY$ and $R_2 = EMPTY$ increase size sizeof(R_0) = size add R_0 to
S_{final} sizeof(R_0) $\leq$ size or $R_1 \neq EMPTY$ or $R_2 \neq EMPTY$

end

The program keeps two sets of relations which are dynamically updated in the process of decomposition. The first set is a set of relations to be decomposed S_R and the second set is the set of relations which form the final solution S_{final}. They are initialized in lines 7 and 8. Set S_R contains initially only one relation R_0. At each decomposition step one relation is removed from S_R and decomposed (function decompose_optimized()). The result of decomposition is added to S_R

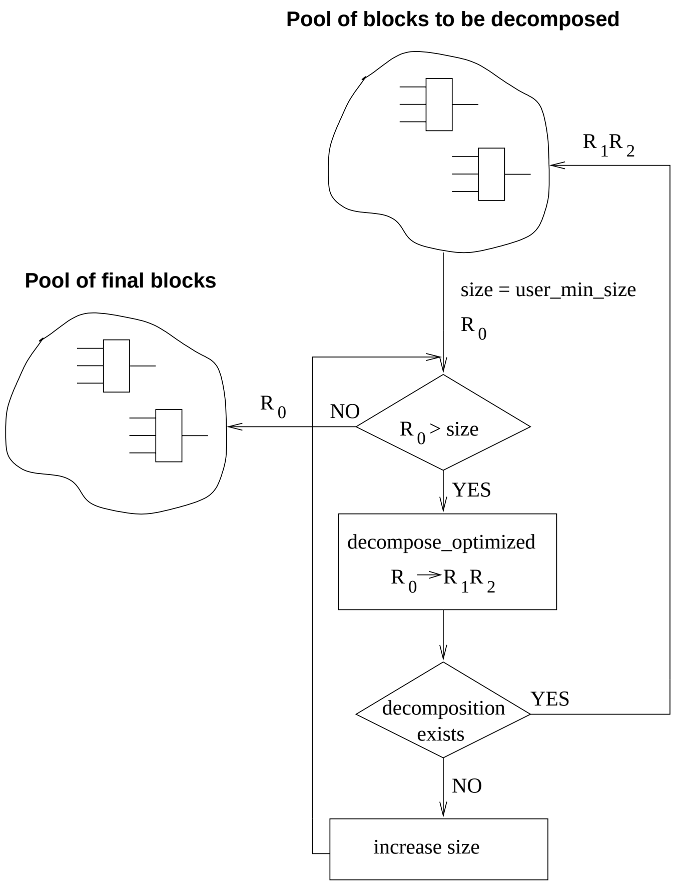


Figure 4.2: Algorithm 6: Decomposition strategy.

and the whole process repeated again (lines 13-15). If the size of the relation removed from S_R for decomposition is equal to the current minimal size $size$, then the relation is not decomposed but added to the set S_{final} (lines 16-17). The function decompose_optimized() (Algorithm 8 p.124) finds an optimum solution based on the criteria described in Section 4.6 p.104. However, if the function decompose_optimized() can not find a decomposition for a given minimum allowable block size $size$ then the relations R_1 and R_2 returned by decompose_optimized() are empty. The fact that a block of size $size$ can not be extracted from R_0 means that concepts hidden in the data are too complex to be described by a block of that size. In such a case the minimum allowable block size $size$ is increased (lines

16 and 17) and decomposition for the same R_0 attempted again. Increasing *size* before the next decomposition attempt allows for extraction of more complex concepts from data. Function decompose_optimized() is described by Algorithm 8 in Section 4.6 (p.124).

4.3 Decomposition of probabilistic neutral relations

An approach to decomposition of probabilistic relations presented in this section is significantly different from the approaches used by Ashby, Conant, Klir, and Krippendorff described earlier. In contrast to the methods developed by these researchers it doesn't use uncertainty as a main decomposition tool and introduces new variables in the decomposition process to decrease the complexity of final solution.

The main idea is to transform a probabilistic relation to a non-probabilistic function. A discrete probabilistic relation can be represented by a set of tuples with corresponding frequency or probability distribution. As such, it can be viewed as a probability density function and decomposed using approach developed for non-probabilistic directed relations. For the decomposition sake the frequency (or probability) distribution is considered to be a dependent variable and relation's variables to be independent variables. Then, the approach used for the decomposition of non-probabilistic relations presented in Section 4.2 (p.74) can be applied to perform the decomposition.

A slightly different decomposition algorithm will be presented in the following sections. It is a close derivative of the algorithm presented in Section 4.2 (p.74) but the interpretation of the extra variables created in the decomposition process is different. In the previous method the extra variables were interpreted as dependent variables by the decomposition procedures. An opposite approach is presented below where the extra variables created in the decomposition process are interpreted as being independent.

To simplify the description of the algorithm, the presentation of the decomposition procedure will be based on tabular representation of probabilistic relations. The software implementation, however, uses lr-partition representation [44],[43], which is more suitable for representation and manipulation of large multiple-valued relations.

Without lost of generalization we will reduce our analysis to binary relations (two variables) only. It is justified by the fact that any k-ary relation (Definition 3.13) can be reduced to a binary relation $R \subseteq S_a \times S_b$ where S_a and S_b are sets of n-ary and m-ary tuples respectively and $n + m = k$.

Data transformation

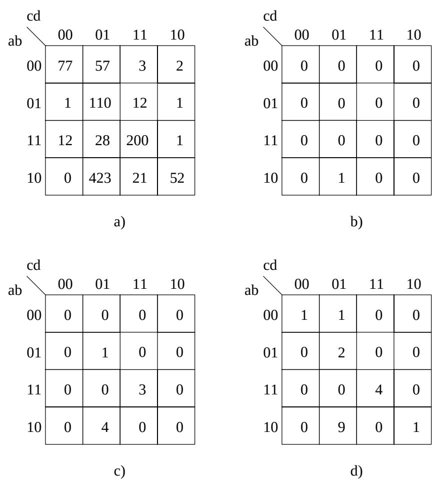
If the data table contains frequency values they can be directly used for the decomposition but doing so is often impractical. For instance if two frequency values differ by only 1% of the total range they can safely be treated as the same value. If the table contains probabilities their values have to be discretized. In both cases we assign one value to all values from a certain range to reduce the number of possible choices; we perform discretization of the dependent variable. The most often used discretization method, uniform binning, divides the space of each variable values into a number of equally sized bins. Another type of discretization methods is based on the entropy measure [20], [39] and use minimum entropy criterion to assign the values to different bins and often yields better results.

Example 4.1 shows an application of a uniform binning procedure for reduction of the total number of different frequency values in the data table.

Example 4.1
Let's take the relation from Figure 4.3a and assign the values in the table to two equally spaced bins labeled by 0 and 1. As a result we obtain the table in Figure 4.3b. The tables in Figures 4.3c and d correspond to assignments to 5 and 10 equally spaced bins respectively.

Figure 4.4 shows another example of transformation of data. Figure 4.4b was created from Figure 4.4a by setting up a threshold on cell frequencies. All the cells with frequencies greater or equal to 70 were assigned value 1, those with frequencies smaller than 70 were assigned value 0. If we designate variables a and b to be independent variables and variables c and d to be dependent variables then functions $c = f_1(a, b)$ and $d = f_2(a, b)$ are binary functions AND and OR respectively. If the threshold on the cell frequencies will be set up on 50 instead of 70, we will obtain the table from Figure 4.4c.

The Figure 4.4c doesn't represent a function anymore. It can be either in-



a)

ab \ cd	00	01	11	10
00	77	57	3	2
01	1	110	12	1
11	12	28	200	1
10	0	423	21	52

b)

ab \ cd	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	1	0	0

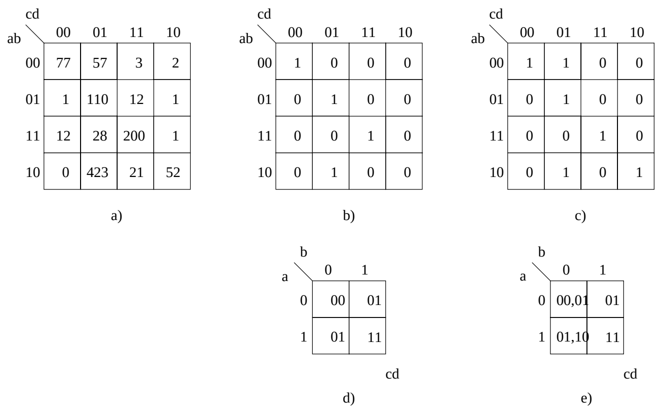
c)

ab \ cd	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	0	3	0
10	0	4	0	0

d)

ab \ cd	00	01	11	10
00	1	1	0	0
01	0	2	0	0
11	0	0	4	0
10	0	9	0	1

Figure 4.3: Transformation of data (a).



a)

ab \ cd	00	01	11	10
00	77	57	3	2
01	1	110	12	1
11	12	28	200	1
10	0	423	21	52

b)

ab \ cd	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	1	0	0

c)

ab \ cd	00	01	11	10
00	1	1	0	0
01	0	1	0	0
11	0	0	1	0
10	0	1	0	1

d)

a \ b	0	1
0	00	01
1	01	11

cd

e)

a \ b	0	1
0	00,01	01
1	01,10	11

cd

Figure 4.4: Transformation of data (b).

terpreted as a function with noise or a relation. Karnaugh maps for functions (relations) c and d corresponding to the tables in Figures 4.4b and c are shown in Figures 4.4d and e.

Other procedures that can be used for dealing with frequency (probability) values of dependent variables are the ones described in Section 5.2.1 (p.149). Lossy decomposition procedures described there can be used instead of procedures proposed in this section.

Disjoint decomposition

Let $X = \{x_i\}, i = 1, \ldots, n$, be a set of variables, X_1, X_2 be a partition of X, and Q_{x_i} be a set of values the variable x_i can take. If R is a relation based on the set of variables X then $R \subseteq Q_{x_1} \times \ldots \times Q_{x_n} = Q_{X_1} \times Q_{X_2}$, where $Q_{X_j} = \{q_{kX_j}\}$ is a set of combinations (tuples) q_{kX_1} variables in the set X_j can take. The table for the relation R has $|Q_{X_1}|$ columns and $|Q_{X_2}|$ rows, one column for every tuple $q_{kX_1} \in Q_{X_1}$.

Definition 4.1 (column multiplicity) *Column multiplicity μ is a number equal to the number of distinct columns in the data table.*

Definition 4.2 (row multiplicity) *Row multiplicity μ is a number equal to the number of distinct rows in the data table.*

Column multiplicity μ is greater or equal to 1 (it is equal to 1 if all the columns are identical) and less than or equal to $|Q_{X_1}|$ (all the columns are different). Let's create a set of new variables $A = \{a_i\}$ such that $|Q_A| \geq \mu$ (in particular $Q_A = \{a\}$ and $|a| = \mu$) and assign tuples $q_{jA} \in Q_A$ to distinct columns of the table (identical columns will have identical tuples assigned to them).

We want to decompose the original relation R into two sub-relations R_1 and R_2. Let $R_1 \subseteq Q_{X_1} \times Q_A$ be a relation created by extending every tuple $q_{iX_1} \in Q_{X_1}$ with a tuple $q_{jA} \in Q_A$ corresponding to the column q_{iX_1} so that $R_1 = \{q_{iX_1} q_{jA}\}$.

R_2 has to be now selected in such a way that the composition of R_1 and R_2 results in R. The selection of R_2 will be defined by the following theorem:

Theorem 4.2 (decomposition) *Relation $R_2 \subseteq Q_{X_2} \times Q_A$ meeting the above condition can be represented by a table created from the original data table for relation R by combining the identical columns of the table. The new columns will correspond to the tuples $q_{jA} \in Q_A$.*

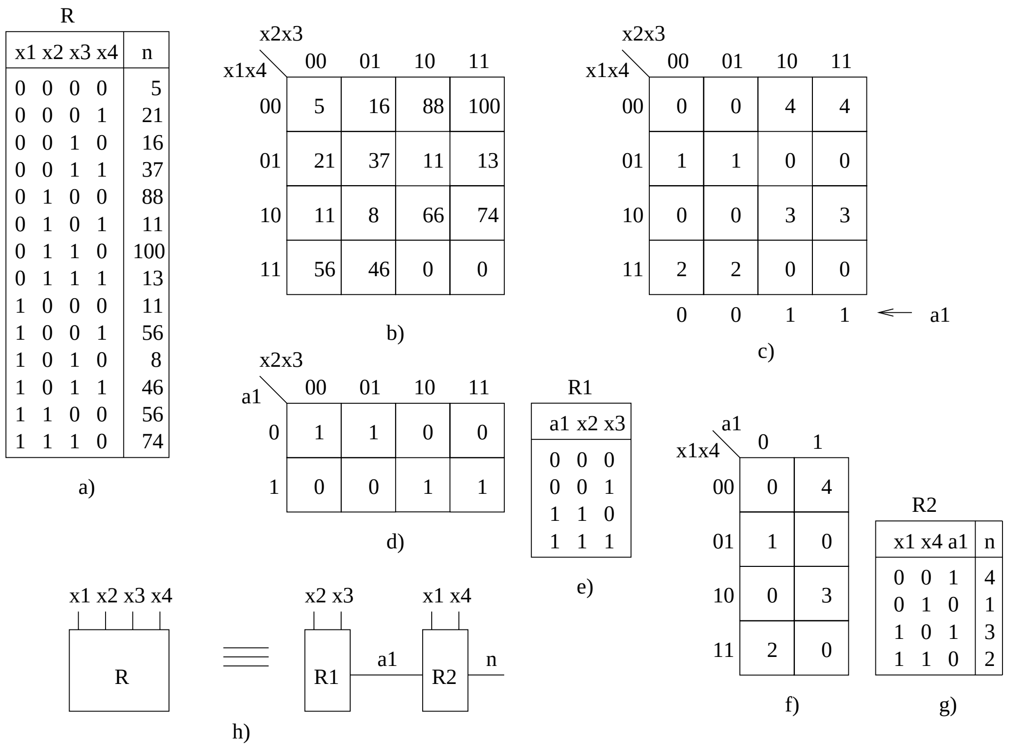
PROOF *It is enough to show that for every pair of tuples $q_{iX_1}q_{kA} \in R_1$ and $q_{jX_2}q_{kA} \in R_2$, it holds that the pair of tuples $q_{iX_1}q_{jX_2} \in R$.*

Let's assume that there exists a pair of tuples $q_{iX_1}q_{kA} \in R_1$ and $q_{jX_2}q_{kA} \in R_2$, such that $q_{iX_1}q_{jX_2} \notin R$. The condition $q_{iX_1}q_{jX_2} \notin R$ means that the intersection of column q_{iX_1} and row q_{jX_2} in the original table contains 0. The condition $q_{jX_2}q_{kA} \in R_2$ means that the intersection of row q_{X_2} and column q_{kA} in the data table corresponding to R_2 contains 1. By the construction of R_1, column q_{kA} corresponds to the set of identical columns containing q_{iX_1}. Hence, by the condition $q_{jX_2}q_{kA} \in R_2$, the intersection of row q_{jX_2} and column q_{iX_1} contains 1 which is in contradiction to the assumption that $q_{iX_1}q_{jX_2} \notin R$. This completes the proof.

Example 4.2 and Figure 4.5 describe a disjoint decomposition of probabilistic relations.

Example 4.2

The original relation R is defined by the set of tuples in the table in Figure 4.5*a* together with their frequencies n. Set $X = \{x_1, x_2, x_3, x_4\}$ is partitioned into two sets $X_1 = \{x_2, x_3\}$ and $X_2 = \{x_1, x_4\}$. Given that partition, the table representing R is shown in Figure 4.5*b*. The table in Figure 4.5*c* has been obtained from the table in Figure 4.5*b* by assigning the frequencies into five equally spaced bins labeled by numbers 0 to 4. Column multiplicity of this table is equal to 2. The identical columns correspond to tuples $q_{0X_1} = 00, q_{1X_1} = 01$, and $q_{2X_1} = 10, q_{3X_1} = 11$. A new variable a with cardinality equal to 2 (column multiplicity) was created and columns labeled with its values. Columns 00 and 01 with value 0, columns 10 and 11 with value 1.



x1 x2 x3 x4	n
0 0 0 0	5
0 0 0 1	21
0 0 1 0	16
0 0 1 1	37
0 1 0 0	88
0 1 0 1	11
0 1 1 0	100
0 1 1 1	13
1 0 0 0	11
1 0 0 1	56
1 0 1 0	8
1 0 1 1	46
1 1 0 0	56
1 1 1 0	74

x1x4 \ x2x3	00	01	10	11
00	5	16	88	100
01	21	37	11	13
10	11	8	66	74
11	56	46	0	0

x1x4 \ x2x3	00	01	10	11
00	0	0	4	4
01	1	1	0	0
10	0	0	3	3
11	2	2	0	0
a1	0	0	1	1

a1 \ x2x3	00	01	10	11
0	1	1	0	0
1	0	0	1	1

a1 x2 x3
0 0 0
0 0 1
1 1 0
1 1 1

x1x4 \ a1	0	1
00	0	4
01	1	0
10	0	3
11	2	0

x1 x4 a1	n
0 0 1	4
0 1 0	1
1 0 1	3
1 1 0	2

Figure 4.5: Disjoint decomposition.

The data table for relation R_1 is shown in Figure 4.5d and corresponding set of tuples in Figure 4.5e. The data table for relation R_2 is obtained by combining identical columns of the table for R. The result is shown in Figure 4.5f, g. A block diagram of the whole decomposition process is shown in Figure 4.5h.

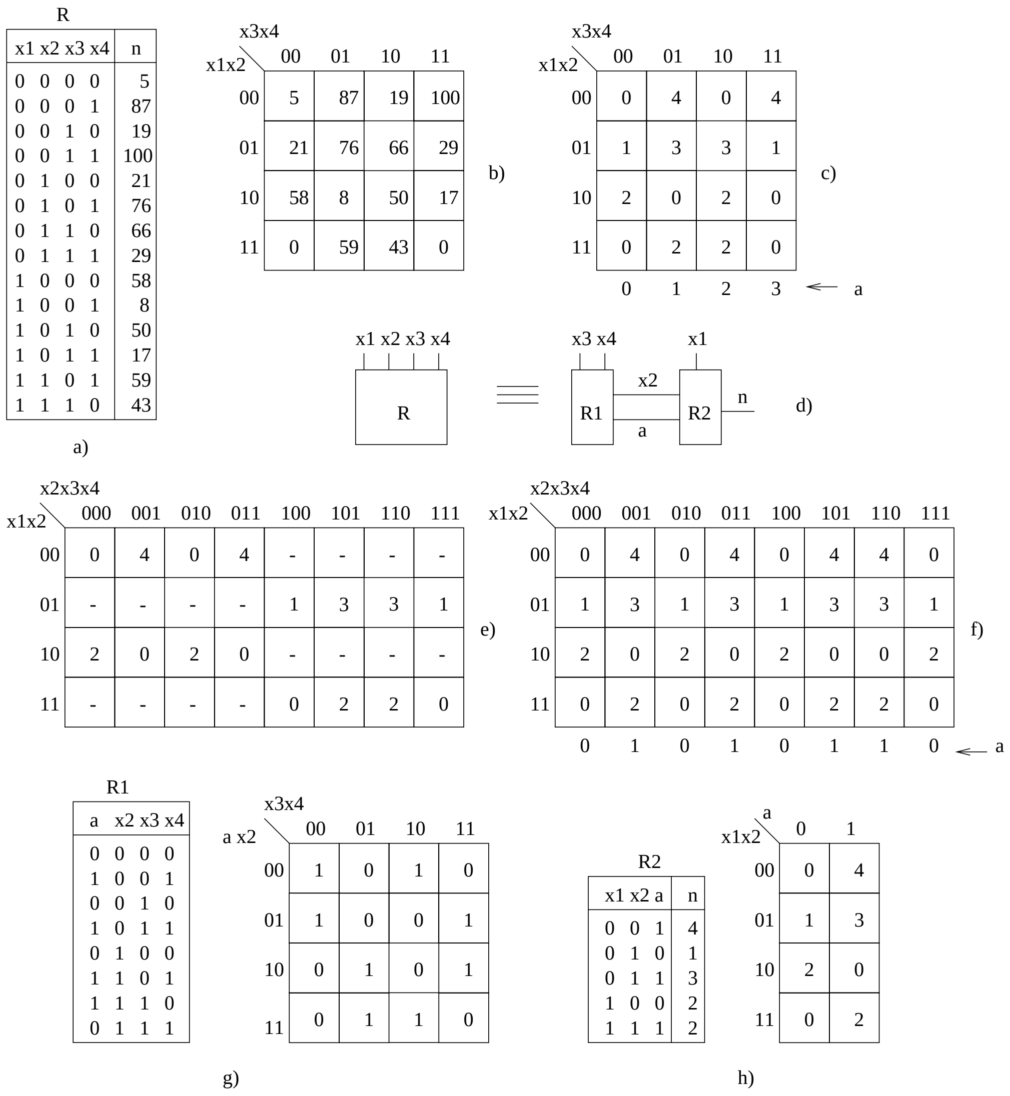
The example above shows the process of decomposition in respect to the columns of the table. The decomposition process is symmetric, however, and it could be performed in respect to the rows of the table as well.

Non-disjoint decomposition

We will call the decomposition non-disjoint if the sets X_1 and X_2 are non-disjoint, $X_1 \cap X_2 \neq \emptyset$. The decomposition process alone is very similar to the disjoint case. The input data, however, have to be appropriately rearranged in order to reduce this case to the previous one. The Example 4.3 and Figure 4.6 present the details of the decomposition procedure.

Example 4.3

The process of decomposition is shown in Figure 4.6. The relation itself is shown in Figures 4.6*a* and *b*.



R

x1 x2 x3 x4	n
0 0 0 0	5
0 0 0 1	87
0 0 1 0	19
0 0 1 1	100
0 1 0 0	21
0 1 0 1	76
0 1 1 0	66
0 1 1 1	29
1 0 0 0	58
1 0 0 1	8
1 0 1 0	50
1 0 1 1	17
1 1 0 1	59
1 1 1 0	43

a)

x1x2 \ x3x4	00	01	10	11
00	5	87	19	100
01	21	76	66	29
10	58	8	50	17
11	0	59	43	0

b)

x1x2 \ x3x4	00	01	10	11
00	0	4	0	4
01	1	3	3	1
10	2	0	2	0
11	0	2	2	0
a	0	1	2	3

c)

x1x2 \ x2x3x4	000	001	010	011	100	101	110	111
00	0	4	0	4	-	-	-	-
01	-	-	-	-	1	3	3	1
10	2	0	2	0	-	-	-	-
11	-	-	-	-	0	2	2	0

e)

x1x2 \ x2x3x4	000	001	010	011	100	101	110	111
00	0	4	0	4	0	4	4	0
01	1	3	1	3	1	3	3	1
10	2	0	2	0	2	0	0	2
11	0	2	0	2	0	2	2	0
a	0	1	0	1	0	1	1	0

f)

R1

a x2 x3 x4
0 0 0 0
1 0 0 1
0 0 1 0
1 0 1 1
0 1 0 0
1 1 0 1
1 1 1 0
0 1 1 1

a x2 \ x3x4	00	01	10	11
00	1	0	1	0
01	1	0	0	1
10	0	1	0	1
11	0	1	1	0

g)

R2

x1 x2 a	n
0 0 1	4
0 1 0	1
0 1 1	3
1 0 0	2
1 1 1	2

x1x2 \ a	0	1
00	0	4
01	1	3
10	2	0
11	0	2

h)

Figure 4.6: Non-disjoint decomposition.

The relation is incompletely specified, two tuples are missing from the table in Figure 4.6*a* and the corresponding entries in the table in Figure 4.6 are equal to zero. These are so called *observational zeroes*, meaning that a given data tuple was not observed so far but it would be if enough measurements were taken. The frequency values have been assigned to ten equally spaced bins and relabeled accordingly. The resulting data table is shown in Figure 4.6*c*. Let us select the following sets X_1 and X_2: $X_1 = \{x_1, x_2\}, X_2 = \{x_2, x_3, x_4\}$. They don't form a partition of X anymore, they are non-disjoint. The data table corresponding to selected X_1 and X_2 is shown in Figure 4.6*e*. Some of the cells of the table contain so called *structural zeros* (denoted by "-" in the table) and correspond to combinations of variable values for which observations are impossible [62]. In our case they correspond to situations where variable x_2 would take value 0 and 1 at the same time which never happens. Since structural zeroes correspond to impossible observations we can replace them with any values for column multiplicity computation. Replacing structural zeroes as it was done in Figure 4.6*f* results in column multiplicity equal to 2. This value is smaller than the value of column multiplicity of the table from Figure 4.6*c* (four) corresponding to the non-disjoint case. Relations R_1 and R_2 can be now determined the same way as for the disjoint case. The result of the decomposition is shown in Figures 4.6*g* and *h*.

4.4 Decomposition of non-probabilistic neutral relations

The approach presented in this section follows closely the approach developed for decomposition of probabilistic relations presented in Section 4.3. A non-probabilistic neutral relation is first transformed to a function and then one of the algorithms presented in Section 4.3 can be applied to perform decomposition. The transformation is based on the following theorem:

Theorem 4.3 (relation) *An k-ary relation based on a set of variables* $X = \{x_i\}$, $|X| = k$, *is equivalent to a function* $f(X)$ *which associates values 0 or 1 to every k-tuple* $x_1, x_2, \ldots, x_k$ *(*$f : X \longrightarrow \{0, 1\}$*).*

Proof *Let R be a relation defined by Definition 3.13 (p.30). Then the equivalence can be shown by selecting a function f as follows:*

$$f(k\text{-}tuple) = \begin{cases} 1 & \textit{if } f(k\text{-}tuple) \in R \\ 0 & \textit{otherwise} \end{cases}$$

The transformation described above can be applied to any relation. However, the algorithms presented in Section 4.2 (p. 74) are more efficient if a directed relation is not transformed to function. This is due to the fact that such a transformation increases the number of independent variables which may results in increased decomposition time. Also, this approach can only be used if we have a representative sets of both, tuples which are contained in R and those which are not. Or, if the relation is completely specified.

4.5 Complexity measures

An appropriate complexity measure together with variable partitioning algorithms are of crucial importance for the quality of the hypothesis selection process. Hypotheses are generated by different partitions X_1, X_2 of the input variables. The best solution is selected based on the cost function used. The complexity measure used as a starting point in this dissertation was the *normalized circuit complexity* proposed by Abu-Mostafa [2] for binary functions. He defined complexity C_x of a binary function $Y = f(X)$ as follows:

$$C_x(f) = \log_2 \min\{\textit{cost of } \Gamma : \Gamma \textit{ simulates } f\} \tag{4.1}$$

where Γ is a combinational circuit realizing function f and cost is equal to 2^n for n-inputs universal block and the cost of a collection of blocks is the sum of the costs of the blocks.

Following his definition the cost of a single $|X|$-inputs $|Y|$-outputs universal block is equal to:

$$cost(f) = 2^{|X|}|Y| \tag{4.2}$$

where: $|X|, |Y|$ are cardinalities of sets of input and output variables respectively.

From Abu-Mosatfa's definition the cost of a binary function realized by a single block is equal to the number of cells of the Karnaugh map representing the function which is equal to the total number of tuples defining the function. The larger that number is, the more variety the function can store, the more details can be described, and more difficult will be the physical realization of the function.

The cost of a single block defined above can also be related to its Kolmogorov complexity. Every relation (function in particular) can be represented by a binary vector of length n, n equal to the total number of tuples representing the function or relation, and can be considered to be a program describing that relation. The length n of the vector can be interpreted as the length of the program. This length is the upper bound of Kolmogorov complexity $K(\cdot)$ for that relation.

4.5.1 Cardinality

The first definition of cost that will be used in the dissertation is an extension of Abu-Mostafa's definition on multiple-valued variables. We will call this cost measure **cardinality** and define it as follows:

$$C_c = \prod_{x_i \in X} |x_i| \; \log_2 \prod_{y_j \in Y} |y_j| \tag{4.3}$$

where:	$\|x_i\|$	cardinality of an independent variable $x_i \in X$,
	$\|y_j\|$	cardinality of a dependent variable $y_j \in Y$.

If we define the cost as being equal to the maximum number of tuples then it is equal to the product $\prod_{x_i \in X} |x_i|$. If the number of dependent variables y_j is greater than one, then each y_j will correspond to a separate function. If all of them are of equal cardinality m then the maximum number of tuples (cost) will be equal to $\prod_{x_i \in X} |x_i|$ times the number of outputs. If we allow y_j to have different cardinalities $|y_j|$ then the number of equivalent m-valued outputs will be equal to $\log_m \prod_{y_j \in Y} |y_j|$ and the total cost equal to $\prod_{x_i \in X} |x_i| \log_m \prod_{y_j \in Y} |y_j|$. By selecting $m = 2$ we normalize the equation to the number of equivalent binary outputs obtaining Equation 4.3.

If the function or relation is represented by a composition of blocks the total cost is equal to the sum of costs of the blocks.

4.5.2 Functionality

Another cost measure can be defined by taking a cost as being equal to the total number of functions that can be realized by a given functional block. We will call this cost measure **functionality**. For a single output binary function $y = f(X)$ the functionality is equal to:

$$C_f = 2^{2^{|X|}} \tag{4.4}$$

The above formula can be extended to multiple-valued, multi-output functions and directed relations as follows:

$$C_f = \left(\prod_{y_j \in Y} |y_j| \right)^{\prod_{x_i \in X} |x_i|} \tag{4.5}$$

Notice that $C_c = \log_2 C_f$, hence, both cost measures provide the same information about a single functional block. In the text that follows the value of $\log_2 C_f$ will be called *log-functionality*.

If the function or relation is represented by a composition of blocks the total cost is equal to the product of the costs of blocks. If we take a logarithm of the total cost we will get again the value equal to the value of cardinality cost measure for the same circuit.

$$\log_2 C_f = \log_2 \prod_i C_{f_i} = \sum_i \log_2 C_{f_i} = \sum_i C_{c_i} \tag{4.6}$$

However, the total cost computed as a product of the costs of blocks is greater than the total number of distinct functions that can be realized by a given structure. It includes repeated functions count and can only be considered as an upper bound for the number of functions that can be realized by this structure. A derivation of the exact value for the functionality cost measure will be discussed in the following sections.

Functionality: disjoint structure

For the disjoint case of the structure shown in Figure 4.7 ($X_1 \cap X_2 = \emptyset$) the total number of different functions N that the structure can realize is given by Equations 4.7, 4.8, and 4.10.

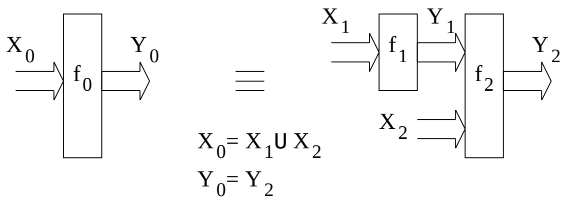


Figure 4.7: Serial decomposition.

Theorem 4.4 *The total number of different functions the structure from Figure 4.7 can realize for $X_1 \cap X_2 = \emptyset$ is equal to:*

$$C_f = \sum_{i=0}^{p_{Y_1}-1} \binom{p_{Y_2}^{p_{X_2}}}{p_{Y_1} - i} K(p_{X_1}, p_{Y_1} - i) \tag{4.7}$$

$$K(c, \mu) = \mu^c - \sum_{j=1}^{\mu-1} \binom{\mu}{\mu - j} K(c, \mu - j) \tag{4.8}$$

where:

$$p_{X_1} = \prod_{x_i \in X_1} |x_i| \qquad p_{Y_1} = \prod_{y_i \in Y_1} |y_i|$$
$$p_{X_2} = \prod_{x_i \in X_2} |x_i| \qquad p_{Y_2} = \prod_{y_i \in Y_2} |y_i|$$

PROOF *The structure from Figure 4.7 can in general be represented by a two dimensional table (Figure 4.8) where the number or rows is equal to the number of combinations the variables from the set X_2 can take and the number of columns is equal to the number of combinations the variables from the set X_1 can take. Each cell of the table stores values of the dependent variables corresponding to the combination of values of independent variables from X_1 and X_2.*

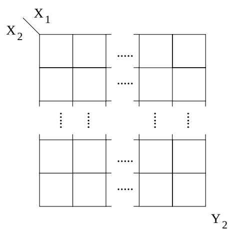


Figure 4.8: Karnaugh map for Figure 4.7.

Consequently, the numbers of rows and columns in the table in Figure 4.8 are equal to p_{X_2} and p_{X_1} respectively. The number of different values each cell of the table can take is equal to p_{Y_2}. The maximum value of the column multiplicity index μ is equal to p_{Y_1} (it is always less than or equal to the number of columns p_{X_1}). Given p_{X_2} - the number of elements in each column of the table, and p_{Y_2} - the number of possible values each element can take, the value $p_{Y_2}^{p_{X_2}}$ is equal to the total number of different columns than can possibly be generated. The total number of different μ-element subsets of that set is equal to $\binom{p_{Y_2}^{p_{X_2}}}{\mu}$. Let $K(p_{X_1}, \mu)$ be the total number of different ways the elements of each of these μ-element subsets can be arranged to form a table with p_{X_1} columns. This has to be done in such a way that at least one of each of μ elements is present in the table (this forms a μ elements partition of a set of p_{X_1} columns). Then, the number of functions with $p_{Y_1} = \mu$

that can be represented by the table is equal to $\binom{p_{Y_2}^{p_{X_2}}}{\mu}K(p_{X_1},\mu)$*. All the cases which result in column multiplicity index* $\mu < p_{Y_1}$ *however, can also be represented by this table. Hence, the total number of functions that can be realized, given the constraint* $\mu \leq p_{Y_1}$*, is equal to* $C_f = \sum_{i=0}^{p_{Y_1}-1}\binom{p_{Y_2}^{p_{X_2}}}{p_{Y_1}-i}K(p_{X_1},p_{Y_1}-i)$*.*
Let us now analyze the expression for $K(c,\mu)$*. The total number of different ways the elements of a* μ*-element subset of columns can be arranged to form a table with* $c, c \geq \mu$*, columns is equal to* μ^c*. This number however, includes not only the cases where all* μ *columns of the subset are used to form a table but also the cases where only subsets of the set of* μ *columns are used. For instance if* $\mu = 3$ *and* $c = 4$ *then not only sequences like* $abcc, acbb, \ldots$ *are counted but also sequences like aabb,cccc,etc. Since* $K(c,\mu)$ *corresponds to exactly* μ*-elements partitions, the later cases have to be subtracted from* μ^c*. The number of these cases is equal to* $\sum_{j=1}^{\mu-1}\binom{\mu}{\mu-j}K(c,\mu-j)$*.*

If all the variables are binary then $|Y_1| = |Y_2| = 1$ and $p_{Y_1} = p_{Y_2} = 2$. For this case Equation 4.7 reduces to:

$$
\begin{aligned}
C_f &= \sum_{i=0}^{1}\binom{2^{p_{X_2}}}{2-i}K(p_{X_1},2-i) \\
&= \binom{2^{p_{X_2}}}{2}K(p_{X_1},2) + \binom{2^{p_{X_2}}}{1}K(p_{X_1},1) \\
&= \binom{2^{p_{X_2}}}{2}K(p_{X_1},2) + 2^{p_{X_2}} \\
&= \binom{2^{p_{X_2}}}{2}(2^{p_{X_1}}-2) + 2^{p_{X_2}} \\
&= \frac{1}{2}2^{p_{X_2}}(2^{p_{X_2}}-1)(2^{p_{X_1}}-2) + 2^{p_{X_2}}
\end{aligned}
\tag{4.9}
$$

and this special case of Equation 4.7 was previously derived in [71].

Corollary 4.2 *The total number of different functions the structure from Figure 4.7 can realize for* $X_1 \cap X_2 = \emptyset$ *is equal to:*

$$C_f = \sum_{i=0}^{p_{Y_1}-1} P(p_{Y_2}^{p_{X_2}}, p_{Y_1}-i) S(p_{X_1}, p_{Y_1}-i) \tag{4.10}$$

where:

$P(n,r)$ *is an r-permutation of n distinct things*

$S(n,m)$ *is a Stirling number of the second kind defined as the number of ways of partitioning a set of n elements into m nonempty sets*

and

$$p_{X_1} = \prod_{x_i \in X_1} |x_i| \qquad p_{Y_1} = \prod_{y_i \in Y_1} |y_i|$$
$$p_{X_2} = \prod_{x_i \in X_2} |x_i| \qquad p_{Y_2} = \prod_{y_i \in Y_2} |y_i|$$

Proof *Expressing $K(c,\mu)$ from Theorem 4.4 in terms of Stirling numbers of the second kind we obtain $K(c,\mu) = \mu! S(c,\mu)$. Hence from Equation 4.7 we have:*

$$\begin{aligned}
C_f &= \sum_{i=0}^{p_{Y_1}-1} \binom{p_{Y_2}^{p_{X_2}}}{p_{Y_1}-i} K(p_{X_1}, p_{Y_1}-i) \\
&= \sum_{i=0}^{p_{Y_1}-1} \binom{p_{Y_2}^{p_{X_2}}}{p_{Y_1}-i} (p_{Y_1}-i)! S(p_{X_1}, p_{Y_1}-i) \\
&= \sum_{i=0}^{p_{Y_1}-1} \frac{p_{Y_2}^{p_{X_2}}!}{(p_{Y_1}-i)!(p_{Y_2}^{p_{X_2}} - p_{Y_1}+i)!} (p_{Y_1}-i)! S(p_{X_1}, p_{Y_1}-i) \\
&= \sum_{i=0}^{p_{Y_1}-1} \frac{p_{Y_2}^{p_{X_2}}!}{(p_{Y_2}^{p_{X_2}} - p_{Y_1}+i)!} S(p_{X_1}, p_{Y_1}-i) \\
&= \sum_{i=0}^{p_{Y_1}-1} P(p_{Y_2}^{p_{X_2}}, p_{Y_1}-i) S(p_{X_1}, p_{Y_1}-i)
\end{aligned}$$

Functionality: non-disjoint structure

The formula derived in [71] covered only disjoint decompositions of binary functions. Let us now analyze the non-disjoint case using a simple example of the binary function shown in Figure 4.9.

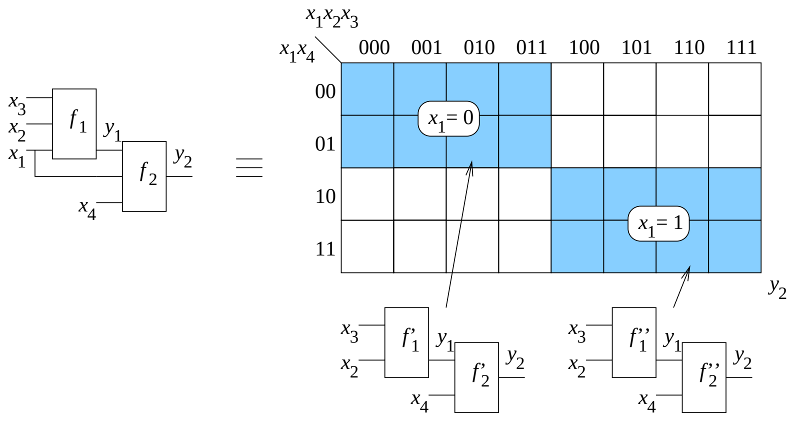


Figure 4.9: Functionality: non-disjoint case.

Non-disjoint structure consists of two subfunctions f_1 and f_2 and $X_1 = \{x_1, x_2, x_3\}$, $X_2 = \{x_1, x_4\}$, $X_3 = X_1 \cap X_2 = \{x_1\}$. The table corresponding to this structure consists of three separate regions:

- $x_1 = 0$: shaded area corresponds to variable x_1 set up to 0.
- $x_1 = 1$: shaded area corresponds to variable x_1 set up to 1.
- non-shaded area corresponds to forbidden combinations of input variables, it corresponds to the cases when the variable x_1 had to simultaneously take the values 0 and 1.

As we can see in Figure 4.9, first two cases correspond to two identical disjoint structures (each structure may realize a different function however) obtained from the non-disjoint structure by removing all the variables contained in set X_3. The number of the disjoint structures is equal to the number of different combinations of values variables from X_3 can take. In our case X_3 contains one binary variable so there are two disjoint structures in the table in Figure 4.9. If X_3 contained a binary and ternary variable we would have six disjoint structures, etc. The total number of functions that can be realized by the non-disjoint structure is therefore equal to the product of the numbers of functions that can be realized by each of the disjoint structures. Since all the disjoint structures are identical it is enough to compute the number of functions realized by one of the disjoint structures and raise it to the power n, where n is the number of disjoint structures.

Extension of the above reasoning to multiple-valued variables is straightforward and results in the following formulas (Equations 4.11, 4.12, and 4.13):

$$C_f = (C'_f)^{p_{X_3}} \tag{4.11}$$

$$C'_f = \sum_{i=0}^{p_{Y_1}-1} P(p_{Y_2}^{p'_{X_2}}, p_{Y_1} - i) S(p'_{X_1}, p_{Y_1} - i) \tag{4.12}$$

where:

$$p_{X_3} = \prod_{x_i \in X_3} |x_i|, \qquad \text{if } X_3 = \emptyset \text{ then } p_{X_3} = 1$$

$$p'_{X_1} = \prod_{x_i \in X_1 - X_3} |x_i|$$

$$p'_{X_2} = \prod_{x_i \in X_2 - X_3} |x_i|$$

Since $X_3 \subseteq X_1$ and $X_3 \subseteq X_2$ then $p'_{X_1} = \frac{p_{X_1}}{p_{X_3}}$, $p'_{X_2} = \frac{p_{X_2}}{p_{X_3}}$, and

$$C'_f = \sum_{i=0}^{p_{Y_1}-1} P(p_{Y_2}^{p_{X_2}/p_{X_3}}, p_{Y_1} - i) S(p_{X_1}/p_{X_3}, p_{Y_1} - i) \tag{4.13}$$

Notice that Equations 4.11, 4.12, and 4.13 (non-disjoint case) reduce to Equation 4.10 (disjoint case) for $X_3 = \emptyset$.

To compare cardinality and functionality measures let us analyze few simple examples. In Tables 4.3 and 4.2 the values of cardinality and log-functionality cost are provided for some simple decomposed structures along with the cost of the original data (it is the same for both cost measures). All the variables are for simplicity taken to be binary.

Three interesting observations can be derived from Tables 4.2 and 4.3:

1. Functionality cost measure (C_f) makes distinction between structures which are equal from the point of view of cardinality cost measure (C_c) (see Table 4.2(a) and 4.2(b)).

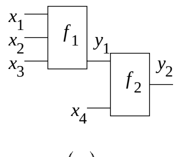
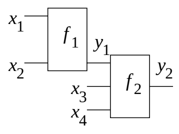
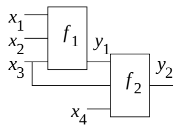
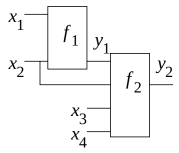
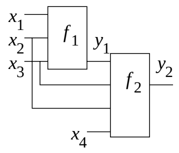
decomposed structure	cardinality C_c	log-functionality $\log_2 C_f$ (C_f)	original data cost
 x_1 x_2 x_3 f_1 y_1 f_2 y_2 x_4 (a)	12	10.6 (1528)	16
 x_1 f_1 y_1 x_2 f_2 y_2 x_3 x_4 (b)	12	10.7 (1696)	16
 x_1 x_2 x_3 f_1 y_1 f_2 y_2 x_4 (c)	16	12.9 (7744)	16
 x_1 f_1 y_1 x_2 f_2 y_2 x_3 x_4 (d)	20	16 (65536)	16
 x_1 x_2 x_3 f_1 y_1 f_2 y_2 x_4 (e)	24	16 (65536)	16

Table 4.2: Comparison of cardinality and log-functionality cost measures for 4-input binary functions.

2. In some cases cost of the decomposed structure is smaller than the cost of original data for functionality cost measure while cardinality cost measure doesn't change for the same structure (see Table 4.2(c) and Table 4.3(a)).

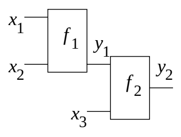
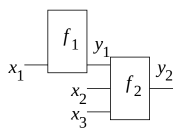
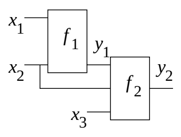
decomposed structure	cardinality C_c	log-functionality $\log_2 C_f$ (C_f)	original data cost
 (a)	8	6.5 (88)	8
 (b)	10	8 (256)	8
 (c)	12	8 (256)	8

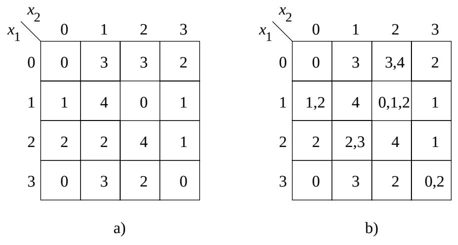
Table 4.3: Comparison of cardinality and log-functionality cost measures for 3-inputs binary functions.

3. If the cost of one of the blocks of the decomposed structure taken alone is equal to the cost of the original data then functionality cost measure is equal to the cost of the original data (and can not grow further). Cardinality cost measure however can become greater that the cost of the original data (see Table 4.2(d) and 4.2(e) and Table 4.3(b) and 4.3(c)).

From 1 and 2 we can conclude that the functionality cost measure provides finer distinction between decomposed structures than the cardinality cost measure. From 2 and 3 we can conclude that the cardinality cost measure for non-disjoint structures doesn't take into account certain repetitions due to the overlapping parts of the blocks of the structure.

Functionality: directed relations

Functionality equations derived in the previous sections for functions can be extended to the case of directed relations (see Definition 3.14). To show the way how this extension can be made, let us analyze the difference between function and directed relation on a Karnaugh map. Figures 4.10*a* and *b* show examples of Karnaugh maps for function and directed relation, respectively.



$x_1 \backslash x_2$	0	1	2	3
0	0	3	3	2
1	1	4	0	1
2	2	2	4	1
3	0	3	2	0

a)

$x_1 \backslash x_2$	0	1	2	3
0	0	3	3,4	2
1	1,2	4	0,1,2	1
2	2	2,3	4	1
3	0	3	2	0,2

b)

Figure 4.10: Function vs. relation.

As we can see in the figure the difference is that for a function each square of a map contains only one value and for a relation the square can contain a subset of a set of possible values the dependent variable can take. This is exactly what was defined as a *set-value* in Section 3.2. Cardinality of a variable allowed to take such values will be equal to 2^n (there are 2^n subsets of a set of n elements) where n is the cardinality of a variable in the sense used when deriving Equations 4.7, 4.8, 4.11, 4.12, and 4.13. Using this extended definition of cardinality for the variables in the sets Y_1 and Y_2 we can apply equations derived for functions to compute functionality of directed relations as well.

4.5.3 Number of degrees of freedom

The complexity of the result of decomposition described in the previous sections depends of the selection of set X_1 defining relation R_1. The common cost measure for probabilistic relations discussed in Section 4.3 is the number of degrees of freedom defined in [62]. The number of degrees of freedom d_f of a relation R is equal to the number of probability values needed to specify that relation.

For single block structure defined by a set of variables X it is equal to $\prod_{x_i \in X} |x_i| - 1$ where $\prod_{x_i \in X} |x_i|$ is the number of cells in the data table and -1 component results from the fact that all probabilities must sum to 1. For decomposed structure let X_1, X_2 be sets of variables for relations R_1 and R_2 respectively, $X = X_1 \cup X_2$ be a set of variables of the relation R, $X_3 = X_1 \cap X_2$ be the non-disjoint part of sets X_1 and X_2, and a be the extra variable. Let $p_1 = \prod_{x_i \in X_1} |x_i|$, $p_2 = \prod_{x_i \in X_2} |x_i|$, and $p_3 = \prod_{x_i \in X_3} |x_i|$. By definition $p_j = 1$ if $X_j = \emptyset$.

Then the number of degrees of freedom of the decomposed structure is equal to (see [62] p. 49):

$$\begin{aligned} d_f(X_1 : X_2) &= d_f(X_1 \cup \{a\}) + d_f(X_2 \cup \{a\}) \\ &\quad - d_f(X_3 \cup \{a\}) \\ &= (p_1|a| - 1) + (p_2|a| - 1) - (p_3|a| - 1) \\ &= (p_1 + p_2 - p_3)|a| - 1 \end{aligned} \tag{4.14}$$

where $|a|$ is the cardinality of variable a.

For disjoint decomposition $X_3 = \emptyset$, $p_3 = 1$ and the above equation reduces to:

$$d_f(X_1 : X_2) = (p_1 + p_2 - 1)|a| - 1 \tag{4.15}$$

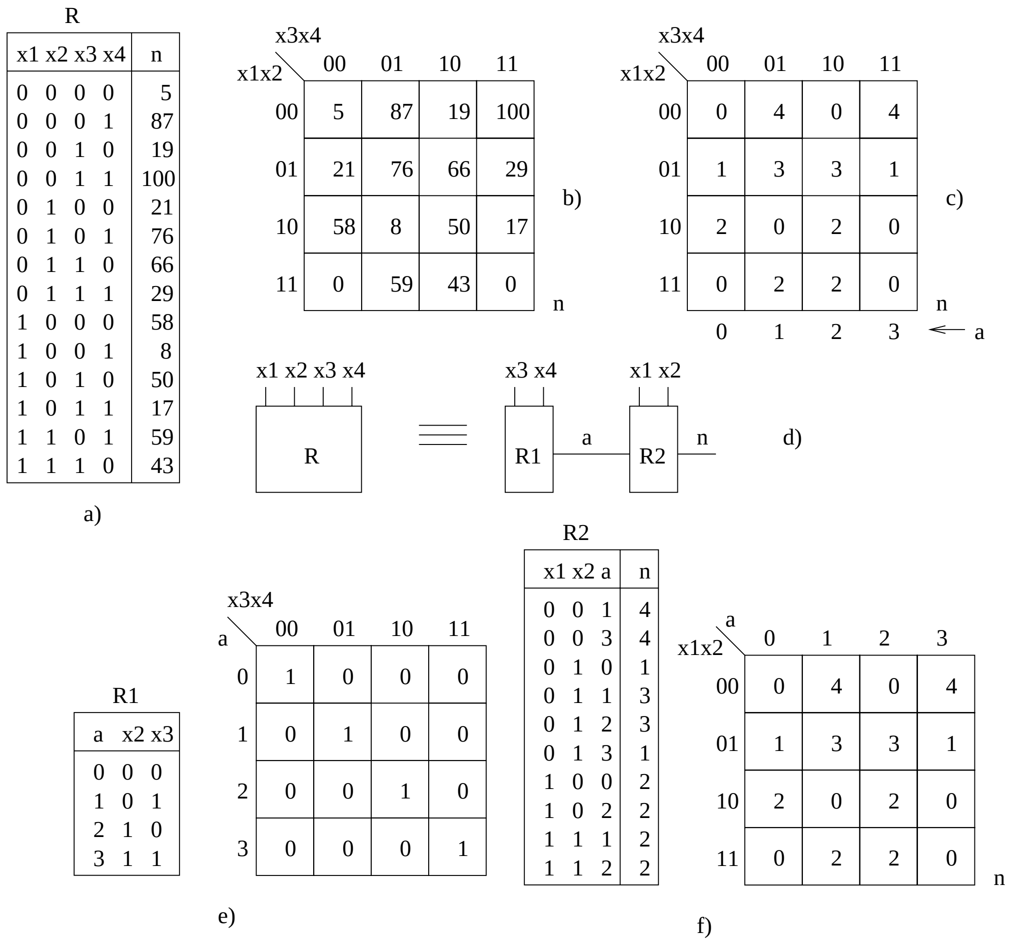
As we can see from these equations introducing an extra variable a always increases d_f of the decomposed structure. One may then ask a question why to introduce an extra variable? The answer is that without it decomposition of R into R_1 and R_2 may not be possible at all. If the decomposition of R into R_1 and R_2 is possible without an extra variable it is better to do so. If such a decomposition is not possible then introducing an additional variable(s) makes it always possible and the problem is how to select the sets $X_1(X_2)$ and X_3 as to minimize the complexity of the result (d_f in this case, but other complexity measures may be used as well).

Comparison: disjoint vs. non-disjoint decomposition

Let us start our analysis from an example.

Example 4.4

In Figure 4.11 the procedure of disjoint decomposition of the relation from Figure 4.6 is presented (disjoint decomposition is the one for which $X_1 \cap X_2 = \emptyset$).



R

x1	x2	x3	x4	n
0	0	0	0	5
0	0	0	1	87
0	0	1	0	19
0	0	1	1	100
0	1	0	0	21
0	1	0	1	76
0	1	1	0	66
0	1	1	1	29
1	0	0	0	58
1	0	0	1	8
1	0	1	0	50
1	0	1	1	17
1	1	0	1	59
1	1	1	0	43

a)

x1x2 \ x3x4	00	01	10	11
00	5	87	19	100
01	21	76	66	29
10	58	8	50	17
11	0	59	43	0

n b)

x1x2 \ x3x4	00	01	10	11
00	0	4	0	4
01	1	3	3	1
10	2	0	2	0
11	0	2	2	0
← a	0	1	2	3

n c)

R1

a	x2	x3
0	0	0
1	0	1
2	1	0
3	1	1

a \ x3x4	00	01	10	11
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

e)

R2

x1	x2	a	n
0	0	1	4
0	0	3	4
0	1	0	1
0	1	1	3
0	1	2	3
0	1	3	1
1	0	0	2
1	0	2	2
1	1	1	2
1	1	2	2

x1x2 \ a	0	1	2	3
00	0	4	0	4
01	1	3	3	1
10	2	0	2	0
11	0	2	2	0

n f)

Figure 4.11: Disjoint decomposition.

The number of degrees of freedom d_f of the structure resulting from disjoint decomposition is equal to 27. Non-disjoint decomposition of the same relation yields d_f equal to 19. This reduction was possible because the selection of non-disjoint sets X_1 and X_2 significantly reduced the value of column multiplicity. But this is not always the case and disjoint decomposition may provide simpler results as well.

The relation in Examples 4.3 and 4.4 was selected to demonstrate possibility of improvement when applying non-disjoint decomposition. The reduction in cost is possible only if the selection of non-disjoint sets X_1 and X_2 results in decrease of column multiplicity. For larger relations this reduction can be quite significant. The following paragraphs provide a formal analysis of the differences in d_f for disjoint an non-disjoint decompositions.

In order to determine the difference in the number of degrees of freedom for disjoint and non-disjoint cases let us follow the following procedure. First, we perform disjoint decomposition of relation $R = X_1 \cup X_2$ with resulting column multiplicity $\mu' = |a'|$. Then according to Equation 4.15 we have:

$$d'_f(X_1 : X_2) = (p_1 + p_2 - 1)|a'| - 1 \qquad (4.16)$$

For non-disjoint decomposition of the same relation R let us create X_3 by taking off some parts of sets X_1 and X_2 so that $X'_3 \subseteq X_1$, $X''_3 \subseteq X_2$ and $X_3 = X'_3 \cup X''_3$. Thus the sets of variables for relations R_1, R_2 for non-disjoint decomposition will be: $X'_1 = X_1 \cup X''_3$ and $X'_2 = X_2 \cup X'_3$ (see Figure 4.12).

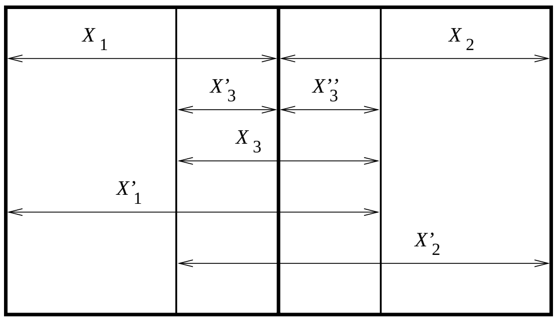


Figure 4.12: Determination of sets X_1, X_2, X_3, X'_1, X'_2, X'_3, and X''_3.

Let the column multiplicity index in this case be $\mu'' = |a''|$. Applying Equation 4.14 we have:

$$d''_f(X_1 : X_2) = (p'_1 + p'_2 - p_3)|a''| - 1 \tag{4.17}$$

where: $p'_1 = \prod_{x_i \in X'_1} |x_i| = p_1 p''_3$,
$p'_2 = \prod_{x_i \in X'_2} |x_i| = p_2 p'_3$,
$p_3 = \prod_{x_i \in X_3} |x_i| = p'_3 p''_3$
$p'_3 = \prod_{x_i \in X'_3} |x_i|$
$p''_3 = \prod_{x_i \in X''_3} |x_i|$.

The difference between d'_f and d''_f will be a measure of what we can gain when performing a non-disjoint decomposition.

$$\begin{aligned}\Delta d_f &= d'_f - d''_f \\ &= (p_1 + p_2 - 1)|a'| - (p'_1 + p'_2 - p_3)|a''| \\ &= p_1(|a'| - p''_3\,|a''|) + p_2(|a'| - p'_3\,|a''|) \\ &\quad -(|a'| - p_3\,|a''|)\end{aligned} \tag{4.18}$$

We will now prove that if $|a'| = |a''|$ then disjoint decomposition always results in lower d_f.

Theorem 4.5 *If $|a'| = |a''|$ then $\Delta d_f < 0$.*
PROOF *For $|a'| = |a''| = |a|$ the Equation 4.18 reduces to:*

$$\Delta d_f = [p_1(1 - p''_3) + p_2(1 - p'_3) - (1 - p_3)] \cdot |a|$$

$\Delta d_f < 0$ *iff*

$$p_1(1 - p''_3) + p_2(1 - p'_3) < (1 - p'_3 p''_3) \tag{4.19}$$

From the definition of $p_1, p_2, p_3, p'_3, p''_3$ we have three additional conditions:

$$p'_3 \le p_1 \qquad p''_3 \le p_2 \qquad p_1, p_2 \ge 2$$

This results in the following:

$$1 - p_3' \geq 1 - p_1 \qquad 1 - p_3'' \geq 1 - p_2 \qquad 1 - p_3'p_3'' \geq 1 - p_1p_2$$

Considering the above, inequality 4.19 can be rewritten as follows:

$$p_1(1 - p_2) + p_2(1 - p_1) < (1 - p_1p_2)$$

and then reduced to:

$$p_1 + p_2 < (1 + p_1p_2)$$

which always holds for $p_1, p_2 \geq 2$.

Let us now consider the situation when one of the sets X_3', X_3'' is empty. Let $X_3' = \emptyset$, then $p_3' = 1$ (by definition), $p_3 = p_3'', p_1' = p_1p_3$, and $p_2' = p_2$. The Equation 4.18 reduces then to:

$$\Delta d_f = p_2(|a'| - |a''|) + (p_1 - 1)(|a'| - p_3\,|a''|) \tag{4.20}$$

In the Table 4.4 the comparison of d_f of original and decomposed structure is presented. For simplicity, we assume that the cardinalities of all the variables $x_i \in X$ are equal to 2, and $X_3' = \emptyset$. $|X_3| = 0$ corresponds to disjoint decomposition and $|X_3| = 1$ to the non-disjoint one. We use Equation 4.17 to compute $d_f(X_1 : X_2)$ in the table.

As we can see from the table, non-disjoint decomposition may improve d_f if $|a''| < |a'|$, which is in agreement with Theorem 4.5. Further, the reduction of d_f resulting from decomposition can be quite significant when the number of variables x_i grows. For small relations (with number of variables equal to 3 or less) the decomposition of neutral relations presented in this chapter will never decrease d_f comparing to the original structure.

This suggests that the number of degrees of freedom cost measure may not be able to capture fine dissimilarities of different structures. This means that a decomposed structure can be rejected as not providing any improvement when in fact it does. For larger structures however, dissimilarities are coarser and possibility of rejecting a good structure is very small.

$\|X\|$	$\|X_1\|$	$\|X_2\|$	$\|X_3\|$	$\|a\|$	$d_f(X)$	$d_f(X_1 : X_2)$
3	2	1	0	2	7	9
4	2	2	0	2	15	13
4	2	2	0	3	15	20
4	3	2	1	2	15	19
5	3	2	0	2	31	21
5	3	2	0	3	31	32
5	3	3	1	2	31	27
6	3	3	0	2	63	29
6	3	3	0	3	63	44
6	4	3	1	2	63	43

Table 4.4: Disjoint vs. non-disjoint decomposition: comparison.

4.6 Complexity minimization

The optimization problem discussed in this section is the one of minimizing the cost of the structure $Y_2 = f_2(Y_1, X_2)$ resulting from the decomposition of a directed relation $Y_0 = f_0(X_0)$ where $Y_1 = f_1(X_1)$, $Y_2 = Y_0$, and $X_1 \cup X_2 = X_0$. The cost functions have been discussed in Section 4.5 (p. 87) and they all depend on cardinalities of variable in the sets X_1, X_2, Y_1, and Y_2. From the optimization point of view the set $Y_2 = Y_0$ is constant and cost of the decomposed structure depends only on the selection of the sets X_1, X_2, Y_1. Determination and selection of the set Y_1 ($P(Y_1)$) is discussed in Section 4.6.1. The selection of sets X_1 and X_2 is referred to as *variables partitioning* and is discussed in Section 4.6.2. Reduction of number of variables in the sets X_1 and X_2 is referred to as *vacuous variables removal* and is discussed in Section 4.6.3. The implementation details of the above algorithms are discussed in Section 4.6.5.

4.6.1 $P(Y_1)$ determination

Let us notice first that for given X_1 and X_2 the complexity reduction depends directly on the number of blocks of $P(Y_1)$ (number and cardinalities of variables in Y_1). The smaller the number of blocks, the greater the complexity reduction.

So given X_1 and X_2 we want to minimize $|P(Y_1)|$ under the constraint that $P(Y_1)$ satisfies the criteria of Theorem 4.1.

The above problem can be reduced to the problem of clique covering of a graph. Nodes of the graph correspond to the blocks of $P(X_1)$, there is an edge between two nodes if and only if the corresponding blocks are compatible. The set of cliques covering all the nodes of the graph corresponds to the blocks of $P(Y_1)$. To minimize the number of cliques we select maximum cliques for the cover.

For the following definitions we assume that the original directed relation is defined by $Y_0 = f_0(X_0)$ and the decomposed structure by $Y_2 = f_2(Y_1, X_2)$, where $Y_1 = f_1(X_1)$, $Y_2 = Y_0$, and $X_1 \cup X_2 = X_0$.

Definition 4.3 (block pair compatibility) *Blocks $B_i, B_j \in P(X_1)$ are compatible, denoted by $B_i \sim B_j$, iff $P(X_2)(B_i \bigcup B_j) \leq P(Y_0)$.*

Definition 4.4 (block set compatibility) *A set of lr-partition blocks $\mathcal{B} = \{B_i\}$, $\mathcal{B} \subseteq P(X_1)$ is a compatible set of blocks iff $P(X_2)(\bigcup_{B_i \in \mathcal{B}} B_i) \leq P(Y_0)$.*

Definition 4.5 (block compatibility graph) *Block Compatibility Graph (BCG) is a graph that has nodes corresponding to blocks of partition $P(X_1)$ and there is an edge between nodes i and j if corresponding blocks B_i and B_j are compatible.*

Definition 4.6 (compatibility clique) *Clique of nodes of BCG graph is called compatibility clique iff the set of blocks corresponding to the nodes of the clique is a compatible set of blocks.*

Definition 4.7 (multiplicity index) *Multiplicity index μ is a number equal to the number of compatibility cliques in BCG graph.*

Theorem 4.6 *Lr-partition $P(Y_1) = \{\mathcal{B}_j | \mathcal{B}_j = \bigcup_{B_i \in CC_j} B_i\}$ where B_i are blocks corresponding to the nodes of compatibility clique CC_j and cliques CC_j cover all the nodes of the graph, satisfies requirements of Theorem 4.1.*

PROOF *Since compatibility cliques CC_j cover all the nodes of the graph then $P(Y_1)$*

is a legitimate lr-partition on the set of cubes $C(X_0 \cup Y_0)$. By Definitions 4.4 and 3.20 (p.33) the second condition of Theorem 4.1, $P(X_2)P(Y_1) \leq P(Y_0)$, is true. By Definition 4.4, every block $B_i \in \mathcal{B}_j$ is also a block of $P(X_1)$. Hence the first condition of Theorem 4.1, $P(X_1) \leq P(Y_1)$, is true.

For functions, every clique forms a compatible set of blocks. For relations however, it is not always true and only cliques of compatible sets of blocks (Definition 4.4) may be used for graph covering.

Example 4.5

Given a relation with 4 binary variables and a 5-valued output variable from Table 4.5 the map from Fig.4.13a is created. The bound set $X_1 = \{x_3, x_4\}$, free set $X_2 = \{x_1, x_2\}$, and output set $Y = \{y_2\}$.

	x_1	x_2	x_3	x_4	y_2
a	0	0	0	0	0,1
b	0	1	0	0	1,2
c	1	1	0	-	0
d	1	0	0	0	0,3
e	0	0	0	1	0,3
f	1	0	0	1	0,4
g	0	0	1	1	1,3
h	0	1	1	1	0,1
i	0	0	1	0	2,3
j	0	1	1	0	2,3
k	1	0	1	0	1,4

Table 4.5: Multiple-valued relation from Example 4.5.

$P(X_1) = \{B'_{00}, B'_{01}, B'_{10}, B'_{11}\} = \{\{a,b,c,d\}_{00}, \{c,e,f\}_{01}, \{i,j,k\}_{10}, \{g,h\}_{11}\}_{x_3x_4}$,
$P(X_2) = \{B''_{00}, B''_{01}, B''_{10}, B''_{11}\} = \{\{a,e,g,i\}_{00}, \{b,h,j\}_{01}, \{d,f,k\}_{10}, \{c\}_{11}\}_{x_1x_2}$,
$P(Y) = \{B_0, B_1, B_2, B_3, B_4\} = \{\{a,c,d,e,f,h\}_0, \{a,b,g,h,k\}_1, \{b,i,j\}_2, \{d,e,g,i,j\}_3, \{f,k\}_4\}_{y_2}$.

The BCG graph is presented in Fig. 4.13b. The minimum number of cliques to cover the graph is 2. If we want the cliques to be disjoint we can select either

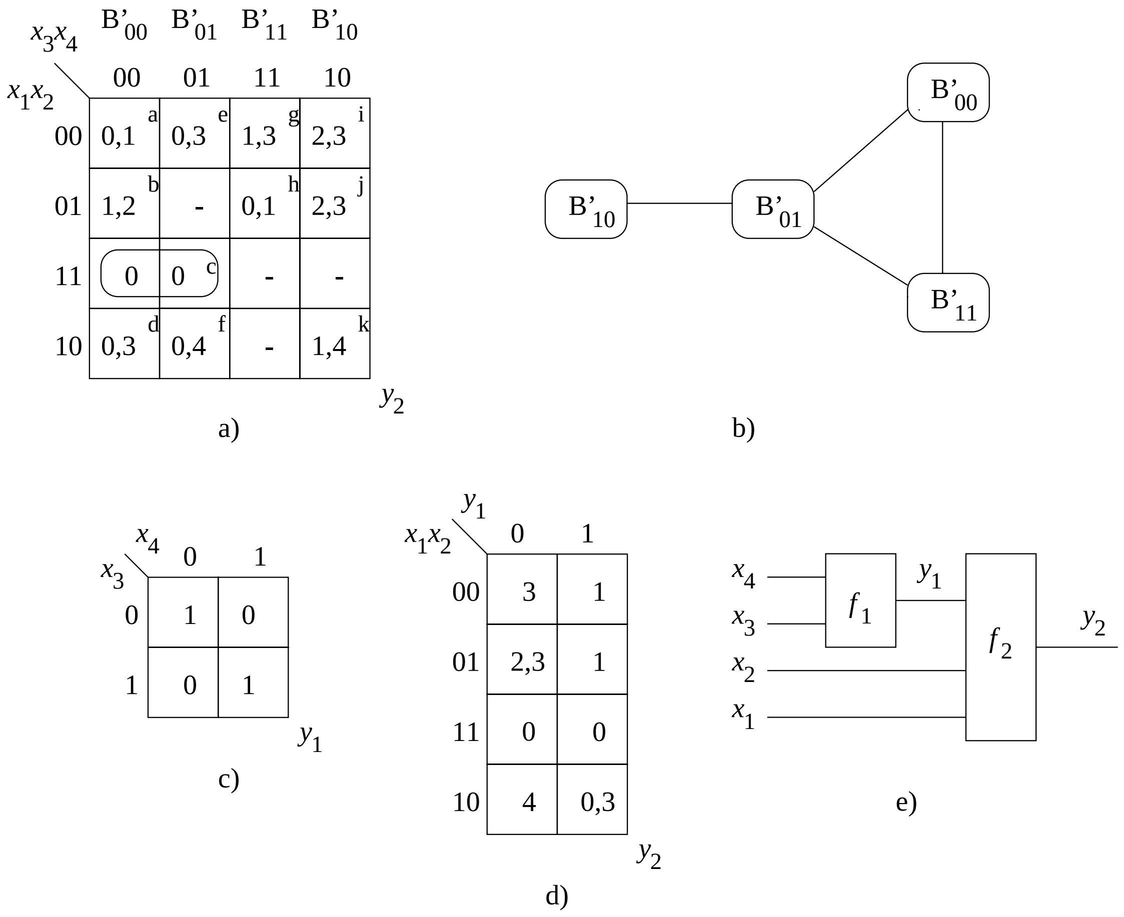


Figure 4.13: Decomposition of relation from Table 4.5.

B'_{10}, B'_{01} and B'_{00}, B'_{11} or B'_{10} and B'_{00}, B'_{01}, B'_{11}. In the clique B'_{00}, B'_{01}, B'_{11} all the blocks are pair compatible but the whole set of blocks is not compatible because (Definition 4.4): $P(X_2)(B'_{00} \bigcup B'_{01} \bigcup B'_{11}) = P(X_2)\{a,b,c,d,e,f,g,h\} = \{\{a,e,g\}$,$\{b,h\}$,$\{d,f\}$, $\{c\}\}$. Since the set $\{a,e,g\}$ is not included into any of the blocks B_0, B_1, B_2, B_3, B_4 of partition $P(Y)$, the conditions of Definition 4.4, 4.6 and Theorem 4.6 are not satisfied. The second selection of cliques satisfies the compatibility clique conditions and according to Theorems 4.1 and 4.6 it can be used to create partition $P(Y_1)$. Fig. 4.13*c* and 4.13*d* show the realization of f_1 and f_2.

Example 4.6

Given a relation with 4 binary variables and a 5-valued output variable from Table 4.6 the map from Figure 4.14*a* is created. The bound set $X_1 = \{x_3, x_4\}$, free set $X_2 = \{x_1, x_2\}$, and the output set $Y = \{y_2\}$. $P(X_1) = \{B'_{00}$, B'_{01}, B'_{10}, $B'_{11}\} = \{\{a,b,c,d\}_{00}$, $\{e,f,g\}_{01}$, $\{j,k,l\}_{10}$, $\{h,i\}_{11}\}_{x_3x_4}$, $P(X_2) = \{B''_{00}$, B''_{01}, B''_{10}, $B''_{11}\} = \{\{a,e,h,j\}_{00}$, $\{b,i,k\}_{01}$, $\{d,g,l\}_{10}$, $\{c,f\}_{11}\}_{x_1x_2}$, $P(Y) = \{B_0, B_1, B_2, B_3, B_4\} = \{\{a,c,d,e,f,g,i\}_0$, $\{b,h,i,k,l\}_1$, $\{b,j\}_2$, $\{a,d,e,f,h,j,k\}_3$, $\{g,l\}_4\}_f$.
The BCG graph is presented in Fig. 4.14*b*. The minimum number of cliques to cover the graph is 2. Let us select cliques $B'_{00}, B'_{01}, B'_{11}$ and $B'_{01}, B'_{10}, B'_{11}$ to cover the graph. Both cliques satisfy requirements of Definition 4.4 and according to Theorems 4.1 and 4.6 can be chosen to create partition $P(Y_1)$. Tables for relations f_1 and f_2 are shown in Fig. 4.14*c* and 4.14*d* respectively. Since the cliques selected to cover BCG graph were non-disjoint, f_1 is a relation. If the cliques were disjoint f_1 would be a function.

As we can see from this example f_1 and f_2 can be relations. If the final realization is required to be unambiguous however, we have to reduce relations to functions some time. We can do it after every decomposition step or we can postpone the reduction until the decomposition process is finished. The second solution allows for a global optimization of the final solution.

	x_1	x_2	x_3	x_4	y_2
a	0	0	0	0	0,3
b	0	1	0	0	1,2
c	1	1	0	0	0
d	1	0	0	0	0,3
e	0	0	0	1	0,3
f	1	1	0	1	0,3
g	1	0	0	1	0,4
h	0	0	1	1	1,3
i	0	1	1	1	0,1
j	0	0	1	0	2,3
k	0	1	1	0	1,3
l	1	0	1	0	1,4

Table 4.6: Multiple-valued relation from Example 4.6.

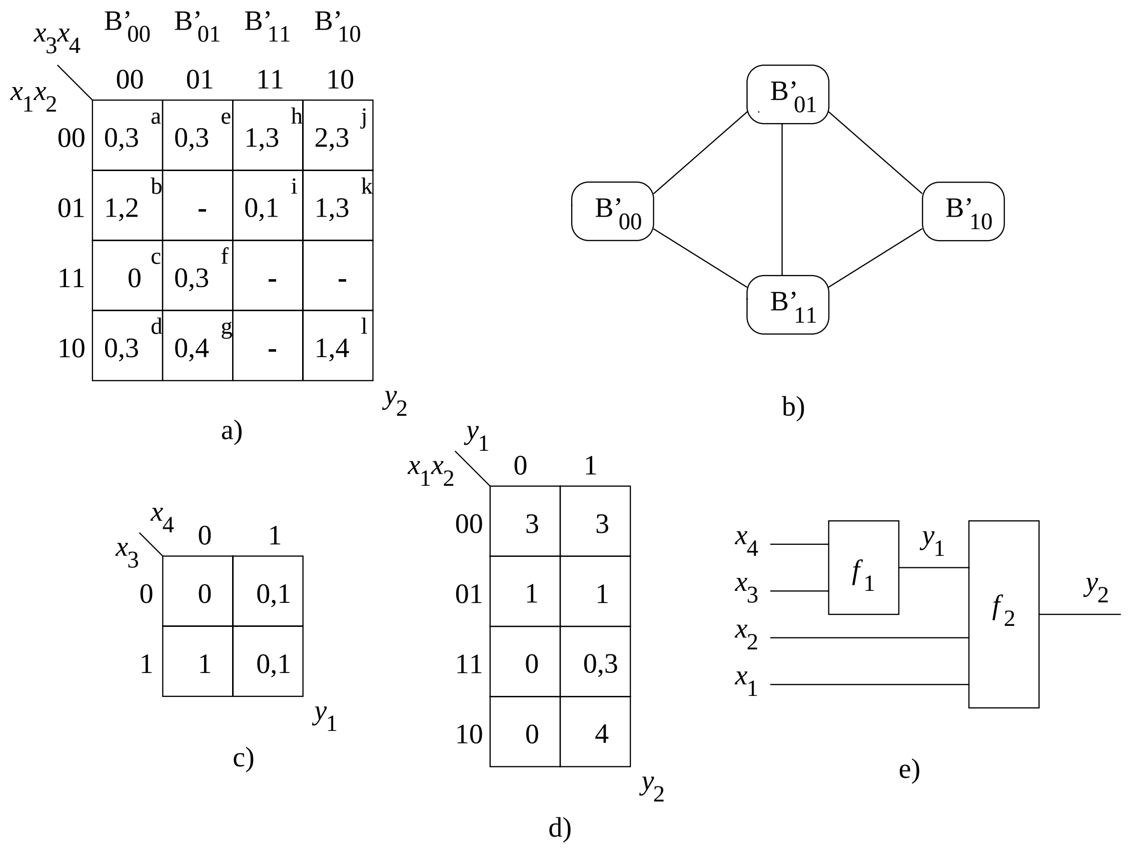


Figure 4.14: Decomposition of relation from Table 4.6.

4.6.2 Variable partitioning for decomposition

Variable partitioning for decomposition is a process of splitting up the set of relation's variables X into subsets X_1 and X_2 in such a way that the decomposed relation is less complex than the initial one. Even though the problem of variable partitioning is very important not much has been done in this domain so far. This is mainly because most of the published decomposition algorithms were tested on small functions only and exhaustive search for partitions was possible in reasonable amount of time. In general the problem of optimal variable partitioning is NP-complete and fast heuristic procedures are needed to perform this task effectively. One such procedure called "Pair Weighting Method" (PWM) was developed in [127] for binary functions, it was extended to the case of multiple-valued relations and used in the dissertation.

The method developed in this dissertation generates a limited set of pairs $\{X_1, X_2\}$ and selects the one which would minimize the cost of the decomposed relation. The idea is to have the set X_2 (inputs of the block f_2, see for instance Figure 4.16 on p.113) contain the variables which are the most relevant for the determination of the output variables Y_0 and the remaining variables being in the set X_1. The method uses entropy [112] or variety [27] measure to order input variables according to their relevancy for the output variable determination (see Figure 4.15).

To satisfy the condition $\sum_i p(x = x_i) = 1$ for computation of uncertainty the set of data cubes must be disjoint. So, if the original data cubes are non-disjoint they have to be made disjoint before uncertainty computations start.

In Figure 4.15 uncertainty $u(y)$ of the dependent variable y is computed first. Then, the conditional uncertainties of independent variables in respect to the dependent variable are computed and compared to $u(y)$. The variable which results in the highest uncertainty reduction (variable b in the Figure) is selected as the most relevant variable. In the next step conditional uncertainties of all the remaining independent variables in respect to the variables y and b are computed and the

- **Uncertainty:**

$$u(x) = -\sum_i p(x = x_i) \log_2 p(x = x_i)$$

$$\sum_i p(x = x_i) = 1$$

- **Conditional Uncertainty:**

$$u(x|y) = u(xy) - u(y)$$

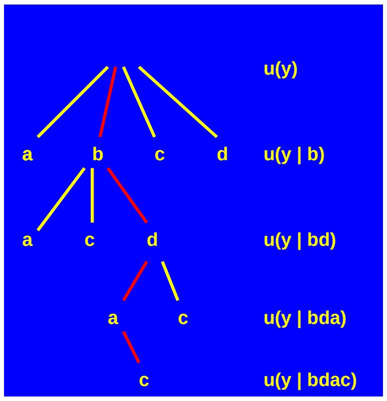


Figure 4.15: Variables ordering.

most relevant variable is appended to the ordered list. This process is repeated until all the variables are ordered. The probabilities for calculating uncertainties presented in Figure 4.15 can be computed directly for lr-partitions as follows:

$$p(x = x_i) = \frac{|B_{x_i}|}{|\bigcup_i B_{x_i}|} \tag{4.21}$$

where $|B_{x_i}|$ is a cardinality of block B_{x_i} of lr-partition $P(x)$ and x_i is a block label (value of variable x).

Ordered set of input variables is then partitioned into sets X_1 and X_2 for decomposition. The optimal partitioning criteria can be determined based on the cost function used in the decomposition process. Criteria for the cost measures discussed in Section 4.5 will be developed in the sections that follow. The details of the entropy based variable partitioning procedure are described by Algorithm 7.

!htbp] Variable partitioning: entropy method

[1] X set of independent variables Y set of dependent variables $final\_list$ empty final list of ordered variables
$s_{xy} = Y$
each $x_i \in X$ compute conditional entropy $u(x_i|s_{xy})$ $u(x_i|s_{xy}) < u_{min}$ $i_{min} = i$ $u_{min} = u(x_i|s_{xy})$ remove $x_{i_{min}}$ from X and add it to s_{xy} append $x_{i_{min}}$ to $final\_list$ $X \neq \emptyset$ or $u_{min} == 0$
partition $final\_list$ into sets X_1 and X_2

end

	entropy %
5xp1	52.9
9sym	100.0
9symml	100.0
add4	100.0
b12	76.0
bw	73.7
clip	11.8
ex5p	89.2
f51m	30.0
gpio	84.0
house	25.0
inc	57.9
misex1	70.6
parity	100.0
rd53	100.0
rd73	100.0
rd84	100.0
root	71.4
sao2	64.3
sqrt8	85.7
squar5	63.6
xor5	100.0
Total	75.28

	entropy %
balance	50.0
balloon1	100.0
balloon2	100.0
balloon3	100.0
balloon4	100.0
breastc	66.7
flag	64.3
irish	100.0
lensesmv	100.0
monk1te	71.4
monk1tr	71.4
monk2te	71.4
monk2tr	85.7
monk3te	75.0
monk3tr	33.3
sensory	50.0
ships	100.0
shuttlem	50.0
sponge	83.3
tic-tac-toe	33.3
trains	100.0
trains20	100.0
zoo	71.4
Total	77.27

Table 4.7: Effectiveness of entropy variable partitioning procedure.

The sets of partitions computed by the two variable partitioning procedures (entropy based, and a procedure based on PWM developed in [127]) are used to find minimum cost decomposition. Table 4.7 shows the result of comparison of effectiveness of the entropy based procedure and the one based on PWM method.

The left table contains MCNC benchmarks [81] and the right one UCI benchmarks [121]. The second column of each table contains the percentage of cases in which the best decomposition was produced from a variable partition generated by entropy partitioning method. As we can see from the table in more than 75% of cases entropy based method resulted in the lowest cost decomposition. It means that in more than 75% cases entropy based method could have been the only variable partitioning method used and result wouldn't be worse than obtained when both methods were used.

Optimal size of sets X_1 and X_2: cardinality cost measure

In this section we will be analyzing the cost measure developed in Section 4.5.1 for the structure shown in Figure 4.16.

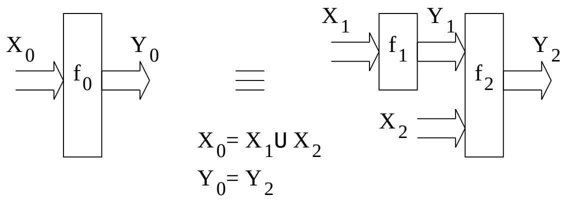


Figure 4.16: Serial decomposition.

$$cost(f_0) = p_{X_0} \log_2 p_{Y_0} \tag{4.22}$$

$$cost(f_1 : f_2) = p_{X_1} \log_2 p_{Y_1} + p_{X_2} p_{Y_1} \log_2 p_{Y_2} \tag{4.23}$$

where:

$$p_{X_0} = \prod_{x_i \in X_0} |x_i| \qquad p_{Y_0} = \prod_{y_i \in Y_0} |y_i| = p_{Y_2}$$
$$p_{X_1} = \prod_{x_i \in X_1} |x_i| \qquad p_{Y_1} = \prod_{y_i \in Y_1} |y_i|$$
$$p_{X_2} = \prod_{x_i \in X_2} |x_i| \qquad p_{Y_2} = \prod_{y_i \in Y_2} |y_i|$$

Disjoint structure For disjoint case of the structure shown in Figure 4.16, $X_1 \cap X_2 = \emptyset$ and $p_{X_0} = p_{X_1} p_{X_2}$. To determine the partition $X_1 : X_2$ for which the cost function $cost(f_1 : f_2)$ takes the minimum value we compute the following derivative:

$$\frac{\partial\, cost(f_1 : f_2)}{\partial\, p_{X_1}} = \log_2 p_{Y_1} - \frac{p_{X_0}}{p_{X_1}^2} p_{Y_1} \log_2 p_{Y_2} \tag{4.24}$$

where we substituted p_{X_2} for $\frac{p_{X_0}}{p_{X_1}}$ in Equation 4.23. To find the minimum of function $cost(f_1 : f_2)$ we compare the derivative to zero and obtain the following relation:

$$\begin{aligned}\frac{\partial\, cost(f_1 : f_2)}{\partial\, p_{X_1}} = 0 \;&\Leftrightarrow\; p_{X_1} \log_2 p_{Y_1} = p_{X_2} p_{Y_1} \log_2 p_{Y_2} \\ &\Leftrightarrow\; cost(f_1) = cost(f_2)\end{aligned} \tag{4.25}$$

which means that the minimum value of the cardinality cost measure of a decomposed structure is obtained for the blocks f_1 and f_2 having their costs equal. Of course achieving equality of costs may not always be possible but we try to get as close to it as we can. Table 4.8 shows examples of different structures where for simplification we assumed that all the variables were binary and $|Y_1| = |Y_2| = 1$.

$\lvert X\rvert$	$\lvert X_1\rvert$	$\lvert X_2\rvert$	$cost(f_1)$	$cost(f_2)$	$cost(f_1 : f_2)$
3	2	1	4	4	8
4	2	2	4	8	12
4	3	1	8	4	12
5	2	3	4	16	20
5	3	2	8	8	16
5	4	1	16	4	20
6	2	4	4	32	36
6	3	3	8	16	24
6	4	2	16	8	24
6	5	1	32	4	36
10	4	6	16	128	144
10	5	5	32	64	98
10	6	4	64	32	98
10	7	3	128	16	144
11	5	6	32	128	160
11	6	5	64	64	128
11	7	4	128	32	160

Table 4.8: Cost for different structures.

Non-disjoint structure The non-disjoint structure is the one for which $X_1 \cap X_2 = X_3 \neq \emptyset$ and $p_{X_3} = \prod_{x_i \in X_3} |x_i|$. In this case the relation $p_{X_2} = p_{X_0}/p_{X_1}$ is no longer true and must be replaced by $p_{X_2} = p_{X_0} p_{X_3}/p_{X_1}$ when computing the derivative in Equation 4.24. This results in the following relations:

$$\frac{\partial\, cost(f_1 : f_2)}{\partial\, p_{X_1}} = \log_2 p_{Y_1} - \frac{p_{X_0} p_{X_3}}{p_{X_1}^2} p_{Y_1} \log_2 p_{Y_2} \tag{4.26}$$

and

$$\begin{aligned} \frac{\partial\, cost(f_1 : f_2)}{\partial\, p_{X_1}} = 0 \;&\Leftrightarrow\; p_{X_1} \log_2 p_{Y_1} = \frac{p_{X_0} p_{X_3}}{p_{X_1}} p_{Y_1} \log_2 p_{Y_2} \\ &\Leftrightarrow\; p_{X_1} \log_2 p_{Y_1} = p_{X_2} p_{Y_1} \log_2 p_{Y_2} \\ &\Leftrightarrow\; cost(f_1) = cost(f_2) \end{aligned} \tag{4.27}$$

which is exactly the same as for the disjoint case.

Conclusions for cardinality cost measure

As we can see from Equations 4.25 and 4.27, in order to minimize the cost of the decomposed structure the variables have to be partitioned in such a way as to make the costs of decomposed blocks equal. The problem with these result however is that the set Y_1 (we need it to compute p_{Y_1} in Equations 4.25 and 4.27) is not known until decomposition is almost finished. One way to proceed is to perform decomposition for every two-subsets partition of the ordered set of input variables obtained in the variable partitioning process (see Section 4.6.2) and select the solution which is closest to the optimum. The number of decompositions to perform in such a case is equal to $n-3$ where n is the number of input variables (we exclude decompositions with single input variable blocks). This solution is acceptable if the number of input variables is small. If n is large we have to limit the number of decompositions. The way to address this problem is to start from decompositions (partitions) which are the most likely to provide solutions close to the optimum and stop if either the optimum was found or the assumed maximum number of decompositions performed. How to find such partitions? From the decomposition of MCNC benchmarks (30 binary, completely specified functions) [81] we found that p_{Y_1} takes most often the values 2 and 3 (88.9% of the total 1661 cases, see Table 4.9). Similar result was obtained when decomposing multiple-valued benchmarks see Table 4.10 (11 MV benchmarks, 55 cases, [121] and [17]).

p_{Y_1}	% of total
2	66.2
3	22.7
4	3.0
5	3.0
6	2.0
7	0.7
8	0.7
9	0.5
10 - 27	1.4

Table 4.9: Cardinality of variables in Y_1 for MCNC benchmarks.

p_{Y_1}	% of total
3	36.4
2	16.4
4	12.7
7	12.7
5	3.6
6	3.6
11	3.6
13	3.6
8	1.8
14	1.8
22	1.8
24	1.8

Table 4.10: Cardinality of variables in Y_1 for MV benchmarks.

Assuming a most probable value for p_{Y_1} we can reduce the optimality condition (Equation 4.25 or 4.27) to the following:

$$\frac{p_{X_2}}{p_{X_1}} = \frac{1}{p_{Y_1}} \frac{\log_2 p_{Y_1}}{\log_2 p_{Y_2}} = const \tag{4.28}$$

Assuming for instance $p_{Y_1} = 2$ (only binary variables) we have $\frac{p_{X_2}}{p_{X_1}} = \frac{1}{2}$ which results in $|X_1| = |X_2|$.

Using this simplified condition we can limit the number of decompositions performed to select the solution closest to the optimum.

We implemented two approaches to decomposition:

1. top-down: block F_0 is decomposed into blocks F_1 and F_2 of approximately equal number of inputs ($|X_1| = |X_2|$ condition).

2. bottom-up: block F_1 extracted from F_0 is of the minimal size specified by the user (usually 1 input).

The first approach used for the binary functions implements closely the optimality condition of Equations 4.25 and 4.27. For multiple-valued relations however this implementation can be quite far from optimality due to unequal cardinalities of the variables.

In the second approach we always extract the smallest possible unit from the block F_0. The reason for implementing it was that it is faster for the data sets which result in large compatibility/incompatibility graphs for top-down approach.

Table 4.11 shows the comparison of the two approaches in terms of cardinality cost measure (cost after decomposition), number of decomposition steps performed to complete the decomposition, and the decomposition time. The benchmarks listed in Table 4.11 are benchmarks for which compatibility graphs for the top-down approach are not too big.

As we can see from the comparison the top-down implementation which follows optimality conditions from Equations 4.25 and 4.27 is better than the bottom-up approach in all three compared aspects. The reason the top-down implementation decomposes faster is that at each decomposition step the cost reduction is near the

	i/o	top-down			bottom-up		
		cost	#steps	t [s]	cost	#steps	t [s]
5xp1	7/10	285.4	17	0.69	311.7	28	0.87
9sym	9/1	117.4	4	1.66	129.1	6	2.77
9symml	9/1	117.4	4	1.26	129.1	6	2.53
add4	8/1	4	2	0.17	4	6	0.08
adr2	4/3	28	2	0.04	28	3	0.04
b12	15/9	295.1	25	6.99	323.5	42	1.35
bw	5/28	612.5	38	1.07	629.1	45	1.12
clip	9/5	689.2	17	5.23	918.6	22	8.81
con1	7/2	76.68	3	0.17	79.02	5	0.14
ex5p	8/63	2321	194	27.1	2516	298	14.92
f51m	8/8	234.2	10	0.59	274.4	23	0.86
house	16/1	209.8	4	24.93	46.94	13	3.66
inc	7/9	411.6	19	0.92	409.3	26	1.18
misex1	8/7	305.3	17	0.79	325	24	0.58
parity	12/1	44	6	403.1	44	9	848.8
rd53	5/3	71.02	3	0.13	74.19	5	0.13
rd73	7/3	155.3	9	0.61	153.4	12	0.94
rd84	8/4	226	14	3.29	231.3	20	3.95
root	8/5	458.1	14	6.14	487.9	20	10.21
sao2	10/4	543.7	14	35.75	918.7	26	10.94
sqrt8	8/4	153	7	0.62	158.4	9	0.83
squar5	5/8	157.2	11	0.27	150.2	10	0.33
xor5	5/1	16	1	0.01	16	2	0.03
Total		7531.9	435	521.53	8357.85	660	915.07

Table 4.11: Top-down vs. bottom-up approach to decomposition: binary functions.

maximum value so the fewer steps are needed to reach the final result. And also, the final result is of lower cost in most cases.

The same kind of comparison was also performed for multiple-valued functions with varying cardinalities of the variables. In this case simple implementation of the top-down approach (blocks F_1 and F_2 have equal number of inputs) was often far from the optimality conditions and comparison appeared to be more favorable to the bottom-up approach (see Table 4.12). The number of steps needed to perform decomposition is still significantly lower but the time to perform them is

	i/o	top-down			bottom-up		
		cost	#steps	t [s]	cost	#steps	t [s]
balance	4/1	424.5	2	30.07	424.5	2	7.7
breastc	9/1	504.2	12	1821	310.8	11	37.79
flag	28/1	941.8	14	54.17	1034	32	81.67
irish	4/1	19.02	1	0.28	19.02	3	0.33
monk1te	6/1	23	7	2.26	17	5	1.07
monk1tr	6/1	23	7	1.23	17	5	0.16
monk2te	6/1	23	7	1.59	17	5	1.19
monk2tr	6/1	70.68	7	7.04	52.85	6	1.05
monk3te	6/1	24.34	4	1.89	22	5	0.6
monk3tr	6/1	95.79	3	6.03	80.9	3	0.36
sensory	11/1	2139	2	81.53	2050	7	196.2
ships	4/1	93.4	1	0.21	80	1	0.05
shuttlem	6/1	45.36	2	1.54	50.45	5	0.14
sponge	44/1	46.7	6	1.49	65.02	45	19.37
tic-tac-toe	9/1	863.5	3	1719	720.5	8	198.4
trains	32/1	6	5	0.1	10	25	0.21
trains20	29/1	25	5	0.29	55.85	28	0.96
zoo	16/1	96.14	8	0.29	89.95	15	0.52
Total		5522.96	107	3730.18	5169.37	219	547.86

Table 4.12: Top-down vs. bottom-up approach to decomposition: multiple-valued functions.

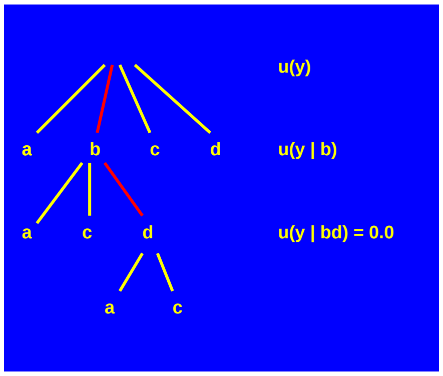
longer. This is because the compatibility graphs grow larger in many cases and creating them and finding clique covering takes more time. To improve the results for top-down approach in this case more sophisticated variables selection algorithm for F_1 and F_2 should be used (clearly the value for p_{Y_1} in Equation 4.28 should be greater than 2 in this case).

4.6.3 Data reduction

Real life data are often redundant, containing many vacuous variables (having no significant impact, if at all, on the classification result), which may obscure, otherwise obvious, relations and dependencies. Vacuous variables (attributes) and data samples may also significantly increase time complexity of the learning algorithms and lead to more complex solutions. The data and cardinality reduction becomes though an important step of the learning algorithm and may significantly decrease not only the time complexity but may also lead to simpler solutions, with better predictive accuracy.

In this section two different methods of reduction of vacuous variables are proposed, one incorporated into the variable partitioning process and the other, based on different principles, will be included in the decomposition algorithm. The reduction of redundant data will be performed at each level of multi-level decomposition process.

The first method, based on the *conditional uncertainty*, will be incorporated into the *variable partitioning procedure* (see Section 4.6.2 p.110). The procedure of variable ordering computes conditional uncertainties of input variables in respect to the output variables. If, after investigating a successive variable, uncertainty reduces to zero, it implies that all the remaining variables are irrelevant for the output variables determination and can be eliminated from further investigation (Figure 4.17).



- Variables b and d reduce uncertainty of y to 0 which means they provide all the information necessary for determination of the output y
- Variables a and c are vacuous

Figure 4.17: Vacuous variables removing.

The second method can be applied at *each step of decomposition procedure*. If in a given decomposition step (Figure 4.16 p.113) function $f_1(X_1)$ is a constant function then the function $f_2(f_1(X_1), X_2)$ doesn't depend on variables $x_i \in X_1$. Hence, all the variables $x_i \in X_1$ are vacuous and can be removed from further analysis.

4.6.4 Discretization of continuous variables

The inference method described earlier in this chapter was developed for discrete functions and relations. Continuous variables have to be discretized in order for the method to be applied. There exist many discretization methods, some of them mentioned in Section 4.3, but in this section we will focus on a new method which is an intrinsic part of our decomposition procedure.

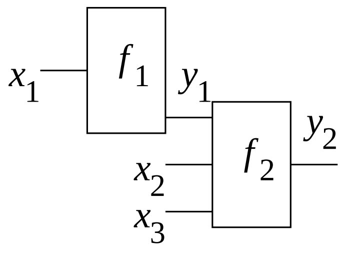


Figure 4.18: Discretization: general procedure.

As it was described earlier, in the decomposition process user specifies a minimal decomposition block which is the smallest size unit that can be extracted from the block being decomposed (block f_1 in Figure 4.18). If the smallest decomposition block has only one input variable x_1 then the decomposition procedure attempts finding a block f_1 with smallest possible cardinality of the output variable y_1. If y_1 can be found such that $|y_1| < |x_1|$ then the block f_1 presents a mapping from the set of values the variable x_1 can take to a smaller set of values the variable y_1 can take. One may say that what we've just described has nothing to do with continuous variables. This is true but if we discretize a continuous variable x_1 using a simple uniform binning method and the number of bins is large enough then the decomposition procedure provides a mean for optimizing both the number and the size of the bins. Moreover the function f_1 doesn't even have to be monotonic. It is discovered in the decomposition process in such a way as to minimize the cost of the decomposed structure. Therefore, the discretization scheme discovered is directly related to the way the dependent and independent variables relate to each other and, as such, can better fit the data than any other independent discretization procedure.

An example of application of this procedure is presented in Figure 4.19.

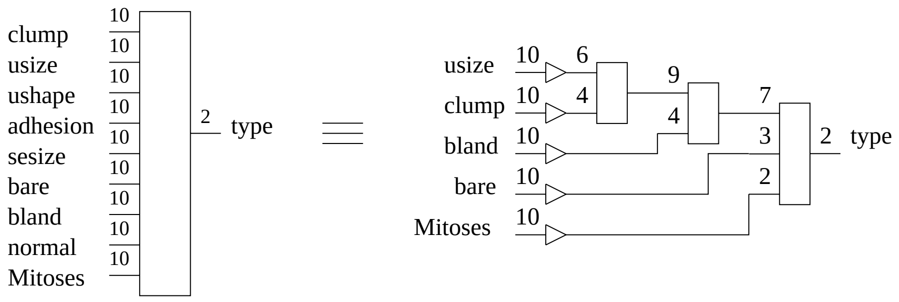


Figure 4.19: Discretization: Univ. of Wisconsin breast cancer data.

The original data file had 9 continuous independent variables and one discrete dependent variable. Independent variables have been uniformly discretized into 10 bins each and in this form the data file is available in [121]. Specifying minimal decomposed block with only one input variable forces decomposer to investigate possibility of reducing cardinality of data variables. As a result, variables which were discretized into too many levels can be discovered and better discretization scheme determined for them. What is interesting here is that often the discovered discretization scheme is not monotonically dependent on the variable values. Commonly used discretization procedures assign consecutive discrete values to adjacent intervals of the values being discretized. In `concept`, the discretization scheme is determined in such a way as to minimize the cost of decomposed structure. Two examples of discretization scheme discovered in the decomposition process for variables `Mitoses` and `bare` are shown in Figure 4.19. For the first variable 10 original discretization levels are reduced non-monotonically to only two levels, for the second variable we have monotonic reduction from 10 to 3 discretization levels. For the remaining variables similar reduction in the number of discretization levels was possible. Similar procedure can be applied for any continuous variable. Initially

Mitoses	f_1(Mitoses)
9	0
8	1
7	0
6	0
5	0
4	0
3	0
2	0
1	0
0	1

bare	f_1(bare)
9	0
8	1
7	1
6	1
5	1
4	1
3	1
2	2
1	2
0	2

Figure 4.20: Discretization schemes discovered in breast cancer data.

they are to be uniformly discretized into larger number of bins without any optimization and then, by applying our inference method, the optimal discretization scheme can be discovered.

4.6.5 Implementation

In the preceding sections three optimization levels have been described:

1. determination of $P(Y_1)$
2. selection of the best partition of input variables
3. removing of vacuous variables

The first level of optimization is provided by function decompose() which performs a decomposition of relation R_0 into two relations R_1 and R_2 minimizing the cost in respect to Y_1 only. The second level of optimization is performed in function variable_partitioning() and in the main loop of the procedure. Selection of the best partition of input variables is performed by calling function decompose() for every partition from the set S_X and comparing costs of different solutions. The third level of optimization is performed in function decompose() and in the main loop of the procedure.

The general framework of the optimization procedure is described in Algorithm 8 (function decompose_optimized()) and illustrated in Figure 4.21. The sets of variables X_1 and X_2 that form a variables partition can in general be non-disjoint, i.e. $X_1 \cap X_2 = X_3 \neq \emptyset$ (non-disjoint decomposition). If $X_3 = \emptyset$ then the decomposition is disjoint. Function decompose() returns a result of decomposition R_1, R_2 and the cost of decomposed structure *cost*. If the relation R_1 returned by function decompose() is empty it means that the dependent variables Y_1 of the relation R_1 are constant and all the variables contained in the set X_1 are vacuous. In this case the decomposition loop (lines 11-19) is interrupted and function terminates returning relations R_1, R_2 (lines 14-15). This earlier termination can significantly speed up the decomposition process in the presence of vacuous variables. If none of the partitions from the set S_X results in the decomposition with the cost smaller than the cost of the relation R_0 then both R_1 and R_2 are returned empty. It means that usable decomposition does not exist. Variable partitioning procedure is described on page 111.

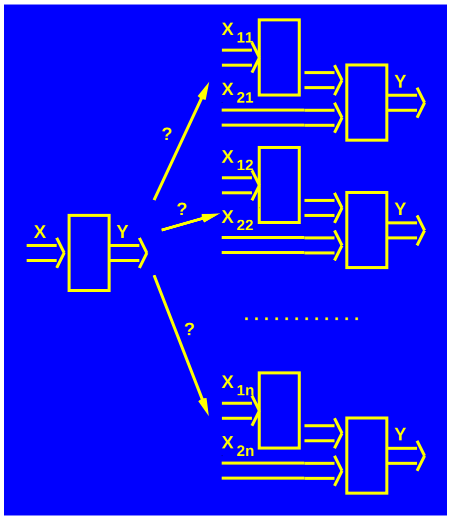


Figure 4.21: Algorithm 8: Illustration.

!htbp] Optimization strategy (function decompose_optimized())

[1] ***Input:*** *$R_0 : X_0 \rightarrow Y_0$ relation to be decomposed min_size cardinality of set X_1* ***Output:*** *R_{1min}, R_{2min} minimal decomposed structure*
$Y_2 = Y_0$ R_{1min} = EMPTY, R_{2min} = EMPTY $cost_{min}$ = cost_func(R_0) S_X = variable_partitioning(R_0, min_size)
$S_X = \{(X_1, X_2)_i\}$ set of partitions every $(X_1, X_2)_i \in S_X$ $X_3 = X_1 \cap X_2$ (R_1, R_2, cost) = decompose(R_0, X_1, X_2, X_3, Y_2)
R_1 = EMPTY return (R_1, R_2) cost < $cost_{min}$ ($R_{1min}, R_{2min}, cost_{min}$) = ($R_1, R_2$, cost) return ($R_{1min}$, R_{2min})

end

The details of implementation of function decompose() are described in Algorithm 9. The main part of the procedure is performed in lines 18-24 where compatibility graph is created and clique cover determined. Then, multiplicity index μ calculated and lr-partition $P(Y_1)$ is created. Based on $P(Y_1)$ the relations R_1, R_2 are determined and the cost of the decomposed structure is calculated.

To increase the speed of the decomposition procedure the code in lines 12-16 is used for a fast check for $\mu = 1$. Function lower_bound_μ() calculates lower bound value for μ (Algorithm 10). If the lower bound value is equal to 1 then also $\mu = 1$. Calculation of the lower bound value can be performed without creating compatibility graph and can significantly reduce decomposition time if the vacuous variables are present.

t] Serial decomposition (function decompose())

[1] ***Input:*** *$R_0 : X_0 \rightarrow Y_0$ relation to be decomposed $\{X_1, X_2, X_3, Y_2\}$* ***Output:*** *R_1, R_2, cost Create lr-partitions $P(X_1), P(X_2)$, and $P(Y_2)$ $P(X_1)$ = make_partition(R_0, X_1) $P(X_2)$ = make_partition(R_0, X_2) $P(Y_2)$ = make_partition(R_0, Y_2) μ_l = lower_bound_μ($P(X_2), P(Y_2)$) Compute lower bound for μ, it is faster than computing μ $\mu_l = 1$ R_1 = EMPTY R_2 = make_new_relation($P(X_2), P(Y_2)$) cost = cost_func(R_1, R_2) graph = create_compatibility_graph($P(X_1), P(X_2), P(Y_2)$) cliques = find_cliques(graph) μ = multiplicity_index(cliques)*
$P(Y_1)$ = make_partition_Y_1(R_0, cliques, μ) R_1 = make_new_relation($P(X_1), P(Y_1)$) R_2 = make_new_relation($P(X_2) \cdot P(Y_1), P(Y_2)$) cost = cost_func($R_1, R_2$)
return (R_1, R_2, cost)

end

Algorithm 10 describes a process of computing the lower bound value for multiplicity index μ. For every block B_i of lr-partition $P(X_2)$ the product $B_i \cdot P(Y_2)$ is computed and its cardinality c compared to the current lower bound value of μ,

μ_l. If $c > \mu_l$ then the value of μ_l is updated. At the end of the procedure μ_l is equal to the lower bound of μ and returned from the function.

!htbp] Computation of lower bound for μ (function lower_bound_μ())

```
[1]
$P(X_2), P(Y_2)$
$\mu_l = 1$ every block $B_i$ of lr-partition $P(X_2)$ $c$ = cardinality of the product $B_i \cdot P(Y_2)$ $c > \mu_l$ $\mu_l = c$
return $\mu_l$
```

end

4.7 Experimental results

4.7.1 Multiple-valued functions

We start our presentation from a well known in the machine learning community benchmark `trains`. The problem was proposed more than 20 years ago by Ryszard Michalski [66]. The set of 10 trains in question is shown in Figure 4.22. The task is to develop decision rules distinguishing trains traveling west from those traveling east.

The data file is available from [121] and contains 10 data tuples (each tuple corresponds to one train from Figure 4.22) with 33 attributes (including one class attribute). This data file has been transformed to the format accepted by concept and is listed below for illustration.

```
.type mv
.i 32
.o 1
.ilb size load w0 l0 s0 n0 ls0 w1 l1 s1 n1 ls1 w2 l2 s2 n2 ls2 w3
l3 s3 n3 ls3 a b c d e f g h i j
.ob direction
.imv 3 4 2 2 10 4 4 2 2 10 3 4 2 2 7 3 4 2 2 8 2 3 2 2 2 2 2 2 2
2 2 2
.omv 2
2 3 0 1 6 3 2 0 0 8 1 3 1 1 6 1 1 0 0 6 1 0 0 1 0 0 0 1 0 0 1 0 0
1 2 0 0 9 1 3 0 0 7 1 2 0 0 0 2 0 - - - - - 0 1 0 1 0 0 0 0 0 0 0
```

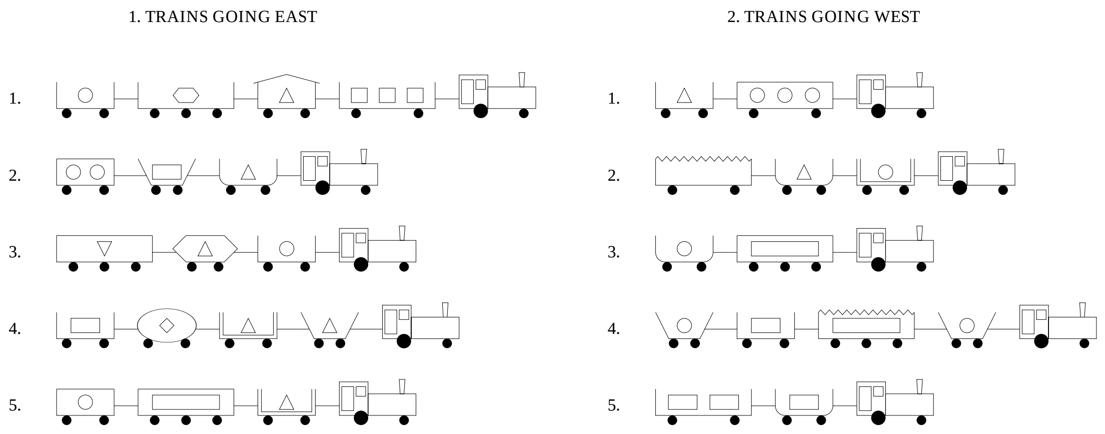


Figure 4.22: Michalski's trains.

```
1 1 0 0 6 1 0 0 0 4 1 3 1 1 0 1 3 - - - - - 0 0 0 0 1 0 1 0 0 0 0
2 1 0 0 7 1 3 0 0 1 1 3 0 0 2 1 2 0 0 6 1 2 1 1 0 0 1 0 0 0 0 0 0
1 2 0 0 1 1 3 1 1 0 1 2 0 0 0 1 0 - - - - - 0 1 0 1 0 0 0 0 0 0 0
0 1 0 1 0 3 0 0 0 6 1 3 - - - - - - - - - - 0 0 0 0 0 0 1 0 0 0 1
1 1 0 0 1 1 0 0 0 9 1 3 0 1 5 0 - - - - - - 0 0 0 0 0 0 1 0 0 0 1
0 1 1 1 0 1 2 0 0 9 1 0 - - - - - - - - - - 0 0 0 1 0 0 0 0 0 0 1
2 1 0 0 7 1 0 0 1 5 1 2 0 0 6 1 2 0 0 7 1 0 1 0 0 1 0 0 0 0 0 0 1
0 0 0 0 9 1 2 0 1 6 2 2 - - - - - - - - - - 1 0 0 0 0 0 0 0 0 0 1
.end
```

Where 33 attributes are:

`size`	1.	Number of cars (integer in [3-5]$\rightarrow$[0-2])
`load`	2.	Number of different loads (integer in [1-4]$\rightarrow$[0-3])
	3-22:	5 attributes for each of cars 2 through 5: (20 attributes total)
`w`	-	number of wheels (integer in [2-3]$\rightarrow$[0-1])
`l`	-	length (short or long)$\rightarrow$[0-1]
`s`	-	shape (closedrect, dblopnrect, ellipse, engine, hexagon, jaggedtop, openrect, opentrap, slopetop, ushaped)$\rightarrow$[0-9]
`n`	-	number of loads (integer in [0-3])
`ls`	-	load shape (circlelod, hexagonlod, rectanglod, trianglod)$\rightarrow$[0-3]

	23-32:	10 Boolean attributes describing whether 2 types of loads are on adjacent cars of the train
`a`	-	Rectangle next to rectangle (0 if false, 1 if true)
`b`	-	Rectangle next to triangle (0 if false, 1 if true)
`c`	-	Rectangle next to hexagon (0 if false, 1 if true)
`d`	-	Rectangle next to circle (0 if false, 1 if true)
`e`	-	Triangle next to triangle (0 if false, 1 if true)
`f`	-	Triangle next to hexagon (0 if false, 1 if true)
`g`	-	Triangle next to circle (0 if false, 1 if true)
`h`	-	Hexagon next to hexagon (0 if false, 1 if true)
`i`	-	Hexagon next to circle (0 if false, 1 if true)
`j`	-	Circle next to circle (0 if false, 1 if true)
	33.	Class attribute (east or west)
`direction`		(east = 0, west = 1)

The format we use is a derivative of the well known in the logic synthesis community espresso format for binary functions. The first four lines of the header have the same meaning as in the original format, namely:

`.type` - type of the data file,

`.i` - number of inputs (independent variables),

`.o` - number of outputs (dependent variables),

`.ilb` - list of input variables names,

`.ob` - list output variables names.

We extended the format to the multiple-valued cases by adding two description lines to the header:

`.imv` - list of input variables cardinalities,

`.omv` - list output variables cardinalities.

and allowing a variable to take any integer value, not only 0 or 1 as it was in the original espresso format. In a fact, since in general we want to be able to represent multiple-valued functions and relations and input data can be represented by cubes, we allow a variable to take a set of integers as its value, for instance 2,4,5. These set values have to be separated by spaces in the data file. One special case of a set value is present in the data, it is the so called "don't care" value represented by "-" (notation taken from the logic synthesis domain) and it corresponds to the set of all values the variable can take. It doesn't of course mean that the variable can take all the values from the set at the same time, it only means that any single value from the set can be assigned to it. For instance let's take the first "don't care" value in the second tuple in the data, it corresponds to variable `w3` which cardinality is 2 (can be read from the `.imv` header line), and means that variable `w3` in this tuple can be assigned either value 0 or 1.

In the context of the `trains` data file "don't cares" correspond to the variables which are not applicable for a given train (for instance the variable `w3` in the second tuple corresponds to the number of wheels of the fourth car of the second train going east; this train, however, is composed of only three cars so there is no data for description of the fourth car; see Figure 4.22 and variables description). This means that during decomposition we can assign to them any value without affecting the original description. In a fact, it is used by the decomposition procedure to simplify decomposed structure, "don't cares" are assigned the values which minimize the cost of the solution.

The result of the decomposition is written to files in the same format as described above, one file per block of the final solution. Running `concept` on the `trains` data file resulted in the data files shown in Figure 4.23.

The first data file represents a boolean function OR on the variables b and e, $s2.0 = b \vee e$. The second data file is a simple boolean negation of the binary variable $s2.0$, $direction = \overline{s2.0}$. So the classification of the trains can be described by a very simple boolean equation:

$$direction = \overline{b \vee e}$$

or equivalently:

$$\overline{direction} = b \vee e$$

```
.type mv
.i 2
.o 1
.ilb b e
.ob s2.0
.imv 2 2
.omv 2
.p 4
1 0 1
1 1 1
0 0 0
0 1 1
.end
```

```
.type mv
.i 1
.o 1
.ilb s2.0
.ob direction
.imv 2
.omv 2
.p 2
1 0
0 1
.end
```

Figure 4.23: Decomposition of trains benchmark.

Taking the meaning of the variables from the data file description we can write the following rule:

If a train has triangle next to triangle or rectangle next to triangle on adjacent cars, then it is Eastbound and otherwise Westbound.

The best rules discovered for the same problem and reported in literature are:

1. If a train has a short closed car, then it is Eastbound and otherwise Westbound.

2. If a train has two cars, or has a car with a jagged roof then it is Westbound and otherwise Eastbound.

As we can see the complexity of the solution obtained by concept is comparable to the best known solutions for this problem.

Analyzing the solution to the `trains` problem one thing particularly strikes a reader, out of 32 independent variables present in the initial description of the problem only two are relevant for determination of the dependent variable! These kind of problems are not rare in Machine Learning and effective detection and removal of such variables can significantly improve the learning process. Concept is able to detect and remove irrelevant (vacuous) variables "on fly" in a very effective manner as a part of the decomposition process (see Section 4.6.3).

Another data file we want to analyze here is `breastc`. This is a University of Wisconsin breast cancer data base that has been used for Machine Learning algorithms testing by various researchers [13][131]. It contains 699 data tuples which are classified into two classes: benign and malignant. There are 9 independent variables corresponding to different measurements, each variable takes 10 discrete values. The results of decomposition of this data file for two different options are shown in Figure 4.24.

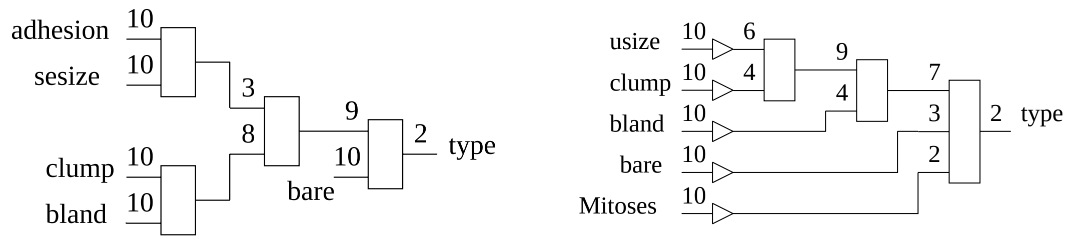
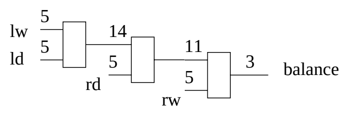
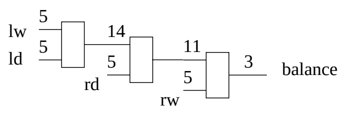
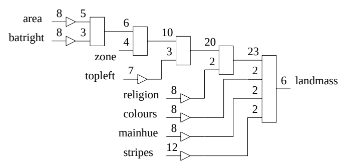
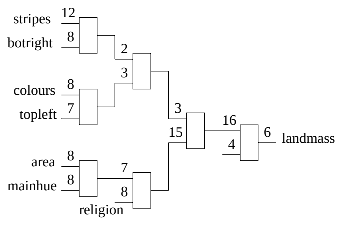


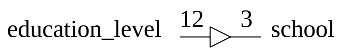
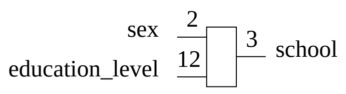
Figure 4.24: Decomposition of Univ. of Wisconsin breast cancer data.

The left-hand side structure has been obtained as a result of selecting a minimal decomposed block with two input variables. The cardinality cost measure of this structure is 624.6 and the number of relevant variables is reduced from 9 to 5. The second structure was obtained by selecting one input variable minimal block and its cardinality cost measure is equal to 310.8. As we can see selecting smaller minimal decomposition block reduces the cost of the final solution (the decomposition time increases however). More examples of comparison of the two decomposition approaches are presented in Tables 4.7.1 and 4.7.1 where columns 'i/o' and 'cost' specify the number of inputs/outputs and cost for the original data file, and columns 'final(1)' and 'final(2)' show the decomposed structures for minimal blocks with 1 and 2 inputs, respectively. The total cost value for the structures in column 'final(1)' includes the cost of blocks which optimize the discretization scheme of the variables.

Choosing minimum block with a smaller number of inputs usually results in a lower cost structure but decomposition time is longer than for larger blocks. With the increase in the size of the minimal block the final structure becomes coarser but the decomposition time shortens. Since each block is basically a set of human understandable rules the whole process can easily be made interactive. By starting decomposition from relatively large minimal block, a human expert can analyze the decomposition results and, if the results are not satisfactory, decide on further finer decomposition for selected blocks. This iterative refinement can be stopped at any time provided the result is satisfactory. Such interaction provides usually much better performance than "only machine" or "only human" approach.

Table 4.13: Decomposition of MV functions (a).

	initial		final (1)	final (2)
	i/o	cost		
balance	4/1	9.9e02	 lw ld 5 5 14 5 rd 11 5 rw 3 balance i = 4, cost = 424.5 t = 7.27 s	 lw ld 5 5 14 5 rd 11 5 rw 3 balance i = 4, cost = 424.5 t = 6.17 s
flag	28/1	2.9e16	 area batright zone topleft religion colours mainhue stripes landmass i = 8, cost = 983.1 t = 81.18 s	 stripes botright colours topleft area mainhue religion landmass i = 8, cost = 938.2 t = 45.28 s

irish	4/1	6.8e02	
			i = 1, cost = 19.02
			t = 0.38 s
				
				i = 2, cost = 38.04
				t = 0.27 s

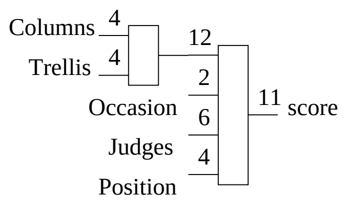
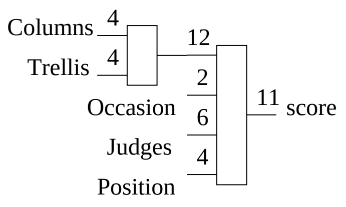
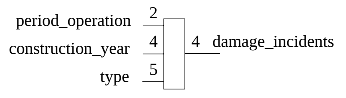
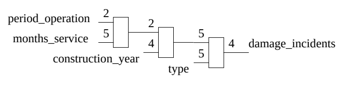
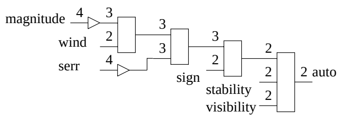
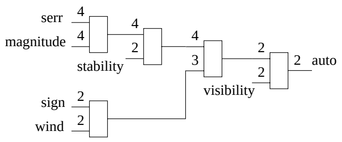
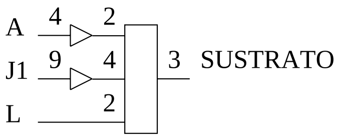
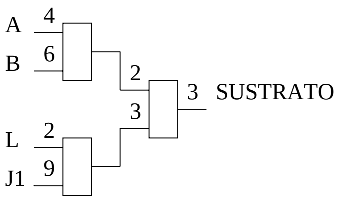
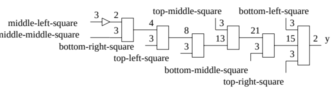
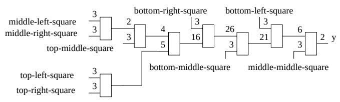
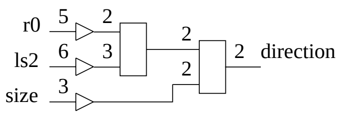
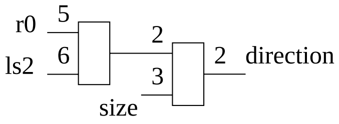
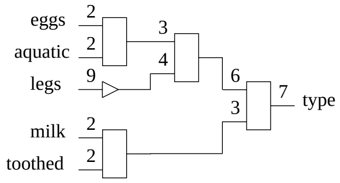
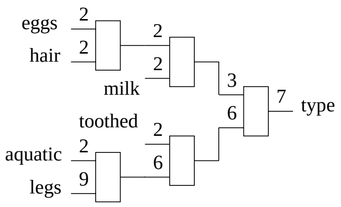
sensory	11/1	7.7e05	 Columns Trellis Occasion Judges Position 4 4 12 2 6 4 11 score i = 5, cost = 2050 t = 196.8 s	 Columns Trellis Occasion Judges Position 4 4 12 2 6 4 11 score i = 5, cost = 2050 t = 148.2 s
ships	4/1	4e02	 period_operation construction_year type 2 4 5 4 damage_incidents i = 3, cost = 80 t = 0.02 s	 period_operation months_service construction_year type damage_incidents i = 4, cost = 78.58 t = 0.16 s

Table 4.14: Decomposition of MV functions (b).

	initial		final (1)	final (2)
	i/o	cost		
shuttle	6/1	2.6e02	 i = 6, cost = 50.45 t = 0.12 s	 i = 6, cost = 70.34 t = 0.08 s
sponge	44/1	2.8e23	 i = 3, cost = 47.36 t = 15.4 s	 i = 4, cost = 62.04 t = 7.93 s
tic-tac-toe	9/1	2e04	 middle-left-square middle-middle-square bottom-right-square top-left-square top-middle-square bottom-middle-square bottom-left-square top-right-square y i = 8, cost = 692.2 t = 189 s	 middle-left-square middle-right-square top-middle-square top-left-square top-right-square bottom-right-square bottom-middle-square bottom-left-square middle-middle-square y i = 9, cost = 871 t = 117.6 s
trains20	29/1	1.6e15	 i = 3, cost = 27.51 t = 0.42 s	 i = 3, cost = 36 t = 0.22 s
zoo	16/1	8.3e05	 i = 5, cost = 112.2 t = 0.43 s	 i = 6, cost = 138.4 t = 0.35 s

4.7.2 Binary functions

In this section we present the results of decomposition of binary functions [81]. Table 4.15 presents comparison of costs before and after the decomposition. As we see from this comparison the cost reduction is sometimes very significant (`c8`, `cc`, `gpio`, `i1`, `b12` for instance).

	initial		final		time
	i/o	cost	i/o	cost	[s]
add4	8/1	256	2/1	4	0.11
adr2	4/3	48	4/3	32.34	0.05
c8	28/18	4.83184e+09	28/18	718.8	11.57
cc	21/20	4.1943e+07	21/20	282.9	0.83
f51m	8/8	2048	8/8	386.8	0.83
gpio	131/91	2.47726e+41	131/91	1480	87.6
house	16/1	65536	8/1	50.45	3.54
i1	25/16	5.36871e+08	25/16	188.3	6.63
parity	12/1	4096	12/1	44	740.8
root	8/5	1280	8/5	610	7.34
5xp1	7/10	1280	7/10	319.7	0.79
9sym	9/1	512	9/1	126.3	2.7
9symml	9/1	512	9/1	129.1	2.56
b12	15/9	294912	15/9	330.4	1.08
bw	5/28	896	5/28	629.1	1.15
clip	9/5	2560	9/5	918.6	8.87
con1	7/2	256	7/2	79.02	0.11
ex5p	8/63	16128	8/63	2523	13.26
inc	7/9	1152	7/9	409.3	1.02
misex1	8/7	1792	8/7	316.7	0.58
rd53	5/3	96	5/3	74.19	0.14
rd73	7/3	384	7/3	153.4	0.83
rd84	8/4	1024	8/4	231.3	3.65
sao2	10/4	4096	10/4	856.3	9.08
sqrt8	8/4	1024	8/4	158.4	0.75
squar5	5/8	256	5/8	150.2	0.3
xor5	5/1	32	5/1	16	0.03

Table 4.15: Decomposition of binary functions.

Table 4.16 shows the results of comparison of our decomposer to leading binary decomposers in terms of cost of the final results. The *cardinality cost measure* has been used for decomposition.

		cardinality cost measure				
File	i/o	TRADE	MISII	DSGN174	concept	[time]
5xp1	7/10	496	384	292	285	[11.0]
9sym	9/1	640	984	400	117	[26.4]
9symml	9/1	644	908	796	117	[23.7]
con1	7/2	80	68	60	77	[2.3]
duke2	22/29	6516	2428	2200	6664	[2562.0]
f51m	8/8	372	392	240	234	[10.1]
misex1	8/7	472	208	224	305	[8.6]
misex2	25/18	548	464	436	669	[2568.0]
misex3c	14/14	19816	4204	3028	5045	[1700.0]
rd53	5/3	120	96	84	71	[1.8]
rd73	7/3	320	352	256	155	[13.1]
rd84	8/4	508	672	320	226	[32.6]
sao2	10/4	1848	516	468	544	[47.2]
Total		32380	15396	10072	14509	

Table 4.16: Decomposition of binary benchmarks.

TRADE is a decomposer developed at Portland State University [127], MISII at University of California, Berkeley, and DSGN174 is a decomposer developed under supervision of Prof. Steinbach in Germany [119]. The binary functions for testing are selected from MCNC benchmarks [81] and are the same as in [119] where results of decomposition of MISII and DSGN174 are compared. The final cost value is computed as a sum of the costs of single blocks of the result of the decomposition. The cost of a single block is computed using Equation 4.2. For our program (concept) the execution time is given in the last column of the table (DECstation 5000/240, 64 MB of memory, user time in seconds). The lowest cost result for every benchmark is underlined in Table 4.16. As we can see from the table our decomposer provided the best result in 7 out of 13 cases, DSGN174 in 5 out of 13, and MISII in 1 out of 13 cases. Taking into account that the other three decomposers were optimized specifically for decomposition of binary function and our decomposer is a general purpose decomposer of multiple-valued functions and relations these results are promising.

4.8 Summary

In this section we formulated a research problem not yet tackled by previous researchers - decomposition of multiple-valued relations, and we proposed a method to solve it.

The method we proposed and described in this chapter is in its general framework based on the works of Ashenhurst [7], Curtis [29] and Karp [50]. They developed a theory of decomposition of boolean and multiple valued functions and proposed a set of decomposition algorithms based on the Karnaugh maps representation. The method consists of splitting a functional block into two blocks, interconnected by a new attribute(s), in such a way as to minimize a certain cost measure. The method described in this chapter is an extension of their work on multiple-valued directed (at least one variable is a dependent variable) and neutral (all the variables are independent) relations. Decomposition algorithms we developed in this chapter are based on a much more efficient data representation, lr-partitions (developed in Chapter 3). Decomposition of directed relations was described in Section 4.2 (p.74) and decomposition of neutral relations in Section 4.3 (p.79).

The whole decomposition process is optimized in respect to certain cost measures. Three such measures were analyzed in this chapter, cardinality, functionality, and number of degrees of freedom (Section 4.5). Cardinality cost measure developed in this chapter (Section 4.5.1, p. 88) is a straightforward extension of the measure proposed by Abu-Mostafa [2] for binary functions. The second measure, we call it functionality, is based on a formula developed by Lendaris and Stanley for disjoint decomposition of binary functions [73]. We developed a more general formula (Section 4.5.2, p.89) that covers not only binary and disjoint but also multiple-valued and non-disjoint cases. The third cost measure used in this work, number of degrees of freedom [62], was analyzed in this chapter in the context of disjoint and non-disjoint decomposition of neutral relations (Section 4.5.3, p.98).

One decomposition step consists of splitting a functional or relational block into two blocks in such a way as to minimize one of the cost measures. The number of possible splits grows exponentially with the number of independent variables so in order to make this problem tractable heuristic procedures have to be used. The procedure developed in this chapter is based on conditional entropy calculation (Section 4.6.2, p.110). The second procedure used in the implementation was borrowed from the work of Wei Wan [127]. It was experimentally shown that splits based on the entropy based procedure resulted in the minimal cost measure in more than 75% of cases.

The heuristic procedures of variable partitioning generate few candidate solutions which are then used for decomposition. Given a variable partition the decomposition consists primarily in finding the lr-partition $P(Y_1)$. This is done by mapping data interrelations into a graph and performing clique covering or node coloring on the graph. The idea of using graph coloring for decomposition has previously been used by Wan Wei and Perkowski [127][126], but the transformation algorithms they developed were related to binary functions represented by sets of cubes. We developed new transformation algorithms for lr-partitions data structure, and extended their clique covering and node covering algorithms onto multiple-valued relations case (see Section 4.6.1, p.104). The search for a low cost solution consists of two-level heuristic optimization: selection of variable partitions and finding minimum clique covering (node coloring) of a graph. The cost of each decomposed structure is evaluated and the best of them is selected as a final solution.

Real life data often contain significant number of irrelevant (vacuous) variables, and their detection and removal may significantly speed up the decomposition process and simplify the resulting structure. Therefore, effective procedures to perform this task are highly desirable. We propose two such procedures (Section 4.6.3, p.119) as an intrinsic part of the decomposition process, which makes them very time effective.

The algorithms presented in this chapter were developed for a very general case of nominal (symbolic) variable values. The assumption therefore is that the variables are discrete. If they happen to be continuous they have to be discretized before our algorithms can be applied to them. As a part of the decomposition procedure we developed an algorithm for analysis of existing variable discretization schemes and, if possible, developing better ones (fewer discretization levels, non-monotonic discretization) (see discussion of decomposition of `breastc` data set in Section 4.6.4, p.121).

It is quite common for Machine Learning data that some of the variable values are missing or irrelevant. Different techniques have been proposed in the past to deal with this problem. The decomposition approach used in this work deals with this problem in a very natural way. The unknown values are either replaced with "don't cares" in the decomposition process and replaced with values which minimize the cost of decomposed structure or kept as unknown if possible. This is again an intrinsic part of the decomposition procedure and is performed for no extra computational cost (see discussion of decomposition of `trains` data file on p.127 for an example).

The decomposition process is iterative; in one decomposition step, a block is decomposed into two smaller blocks and the process is repeated automatically until a certain stopping criterion is satisfied. The stopping criterion used by the implementation is a minimal block size specified by the number of independent variables. Therefore, the granularity of decomposition can be specified by a user. Interactive decomposition process can be controlled by a human expert by starting with a low granularity decomposition, investigating the results, and proceeding with higher granularity decomposition of selected blocks if needed. The opposite approach would start from the lowest possible granularity decomposition, investigating possible meanings of the discovered concepts and composing blocks which don't provide any additional explanation to the data. The advantage of the first approach is speed, the disadvantage is that large data blocks may be difficult for analysis. In the second approach the analysis of small data block is much easier but the decomposition time may be significantly larger.

Chapter 5

LEARNING

5.1 Introduction

What is learning? What is a learning system? Learning is a very broad notion and is usually defined as acquiring knowledge or skill by study, experience, instruction, etc. This may mean, in the simplest case, memorizing and directly retrieving memorized knowledge without any additional transformation, or, in more complex cases, understanding and creating concepts from existing knowledge. Understanding how things work often implies predictive ability, being able to predict the future. Using other terminology, it means creating new theories (concepts) which would explain the future behavior of a system. Learning implies increase in knowledge and improved performance.

From the point of view of the type of inference strategy or methods used in the learning process we can distinguish the following learning methods:

- **Memorization** (rote learning): the simplest form of learning, the knowledge is simply stored in the same form as it will be used. Memorizing multiplication tables is an example.

- **Direct instruction** (by being told): more complex form of learning which require transformation of knowledge before storing. Learning from a teacher presenting facts is an example.

- **Analogy**: is a process of learning new concepts based on similar concepts or solutions.

- **Deduction**: or inference, is a process of logical derivation of new facts from other, already known, facts. Enough information has to be available to permit this kind of learning. For instance, if we know that an animal is a bird and that all birds have wings we can deduce that that animal has wings.

- **Induction**: consists of generating a general concept after seeing a limited number of instances or examples of the concept. This is almost always the case in real problems; there is only partial information available and based on this partial information we have to derive general concepts. For instance, we learn a concept of horse by seeing several (but not all) horses, possibly of different color, size, age, etc., so that we could grasp what they have in common. And these features, common for all horses, define the general concept of horse.

Many learning tasks in real life can not be defined well except by examples. The relationship between different features (variables) is often unclear and difficult to discover and translate into rules understandable by humans. Decision making, which is one of the most fundamental processes of human activity, is a process of deciding what to do if presented with a variety of evidence (also referred to as *features, attributes* or *variables*). In other words, it is a process of establishing a correspondence between input values and a decision to be made, or the process of selecting a class (decision) for a given combination of attribute values, and is commonly referred to as a **classification** task. In statistics, the classification problem is also known as **prediction** problem, in machine learning, as **concept learning**.

The quality of the decision making process heavily depends on the decision maker's experience and knowledge in the area of interest. Since a need for accurate decisions is common and expertise in many fields is scarce, computers are used to facilitate these, often difficult, tasks. Decision-making computer programs are called **classifiers** or **predictors**.

Learning System will be defined here as a computer program whose fundamental goal is to build a classifier in the inductive learning process (which is always biased). Learning system is presented with a finite number of samples of solved cases, and building a classifier is usually based on adjusting parameters of a selected classifier model (linear discriminator, nearest neighbor classifier, decision tree, neural network, etc.) according to certain performance criteria. Using different terminology, building a classifier is also referred to as finding the best hypothesis or theory which explains the data.

Constructive Induction (concept proposed first by Michalski [83]) is a two level learning method which consists of searching first for new concepts in data (search for an adequate representation space), and then searching for the best hypothesis within the space of new concepts. The process of discovering new concepts is equivalent to specifying a vocabulary (variables) to be used while writing a program (hypothesis) using certain programming language (functional architecture).

Constructive induction is based on a number of ideas and assumptions [5]:

- It is based on the idea that the quality of the knowledge representation space is the most important factor in concept learning. If the representation space is of high quality (i.e. chosen attributes or descriptive terms are of high relevance to the problem at hand), learning process will be relatively easy and will likely produce hypotheses with high predictive accuracy. If the quality of the representation space is low (i.e. attributes are of little relevance to the problem), a learning process will be complex and no method may be able to produce good hypotheses.

- It searches for patterns in data and/or learned hypotheses, and uses them for proposing knowledge space transformations (that may expand or/and contract the space).

- It creates new descriptors (attributes or terms) that may be very complex, multilevel functions or transformations of the original descriptors).

- It postulates that produced concept descriptions should be comprehensible to human experts, so that they are relatively easy to interpret and express in terms and forms used by experts.

Many constructive induction methods have been developed and they are usually classified based on the strategy employed for generating new representation spaces. They may be classified into the following categories:

- **Data Driven Constructive Induction**
  The input data are analyzed and, based on the relationships between the input variables, changes are made in the representation space. [104], [14], [64], [65], [36], [42], [82], [41], [111].

- **Hypothesis Driven Constructive Induction**
  Hypotheses generated in the second step are used for selection of the representation space. The whole process (both steps) is iterated until the satisfying solution is found [35], [90], [132], [93], [80], [129], [130].

- **Knowledge Driven Constructive Induction**
  The new representation space is generated based on expert-provided domain knowledge [66], [68], [69], [33], [56].

- **Multistrategy Constructive Induction** is a combination of any of the above types [66], [83], [84], [89], [122], [123], [91], [92], [53], [54], [103].

The method of induction presented in this work, decomposition, extracts from data new variables (concepts) to reduce complexity of the solution. The idea of using new variables to represent concepts that can not be measured directly has been used in *latent class analysis* technique. Latent class analysis is a technique of analyzing data that has been very popular in the social and behavioral sciences for a very long time [67],[12]. This technique introduces new variables, not observed directly in the data (*latent variables*) to represent concepts that can not be directly measured. For instance there is no way to directly measure intelligence of a person, intelligence can only be identified based on the results of certain tests, observable variables. Latent class analysis is a technique that determines relationships between observable and latent variables within a predefined stochastic model. Different models have been proposed for different applications and the selection of the right model is usually based on the experience and knowledge of a researcher.

The idea of using decomposition in order to construct a network of functional blocks (concepts) matching given data was presented first by Lendaris and Stanley [73], [72], [71]. They use the theory of decomposition of binary functions developed in [7], [29], and [50] as a tool for the development of self-organizing systems, networks of adaptive logic elements in particular. The structure of the network (hypothesis) is modified according to the constraints in the environment pertinent to the task (function to be learned known by the teacher). Structure of the network analyzed in [73] and [72] is a disjunctive cascade of universal logic elements similar to Maitra cascades [76]. Subfunctions relevant to the task are discovered in the process of adjusting parameters of universal logic elements until they match the learning data. The approach was applied to completely and incompletely specified binary functions.

The standard machine learning approach is focused on learning a single concept from data. In this dissertation Ashenhurst-Curtis type decomposition is used to decompose data into an organized multi-level structure of primitive (non-decomposable) functional blocks (concepts). As an example of this approach let us consider a decomposition process shown in Fig. 5.1.

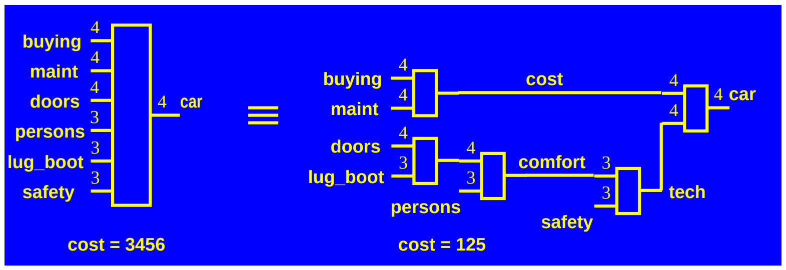


Figure 5.1: Discovering new concepts.

The benchmark car described in [15] was developed for evaluating cars based on their price and technical characteristics. Three new concepts (variables) have been discovered in the decomposition process: *cost*, *comfort*, and *tech*. Concept *cost* depends on independent (input) variables *buying* (buying price), and *maint* (maintenance cost). Concept *comfort* depends on independent variables *doors* (number of doors), *lug_boot* (luggage boot), and *persons* (number of persons). Variables *safety* and *comfort* define concept *tech* (technical characteristics), and the original concept *car* can be now expressed in terms of the concepts *tech* and *cost*. The complexity of the new structure is much smaller than the original one and according to the Occam Razor principle should have better generalization properties (more detailed treatment of generalization issue is provided in [70][73][72]).

5.2 Learning from noisy or incomplete data

Real life data sets used for building classifiers are hardly ever perfect and have often incorrect or uncertain values of variables (attributes). The way to deal with the problem usually depends on whether it is an input or output variable.

The most common types of errors are:

- Missing value.
  The most common technique to handle missing value is to either neglect the whole data sample or substitute the value with a value selected according to certain criteria, for instance select the value the variable takes most often in the same class. If the value of the output variable is unknown the whole data sample is neglected.

- Uncertain value.
  In some situations it is more appropriate, or even necessary, to allow a variable to take multiple values. It doesn't mean that the variable can take different values at the same time however, it only means that we are not sure which of the set of values is the correct one and we want to postpone

the decision until more information will be available. For instance dependent (output) variables, their values, in the case of supervised learning, have to be determined by a teacher, expert in the area (class assignment). However, since experts do not always agree on what class a given set of input values has to be assigned to, it is better to postpone the final decision in the hope that additional information will be available to help us select the right value. In cases where ambiguity can not be resolved based on available knowledge, we can apply some general criteria, like Occam razor principle, to select the value for a variable. The decision can be postponed until the learning process is finished and then a value can be selected to minimize overall complexity of the classifier. The situations described above can not be described by using functions; the more general notion of relation is needed to capture the problem.

In this dissertation missing values and uncertainties are represented by set-values presented in Section 3.

The presence of noise in the data adds a new dimension to the problem of hypothesis selection. If we want to avoid overfitting problem the procedure of construction of the compatibility graph (Example 4.6 p.108) has to be modified (a hypothesis which perfectly fits the data with noise has usually poor generalization properties). For instance the curve in Figure 5.2*b* follows all the small details (or noise) in data but may work poorly in extrapolation or interpolation.

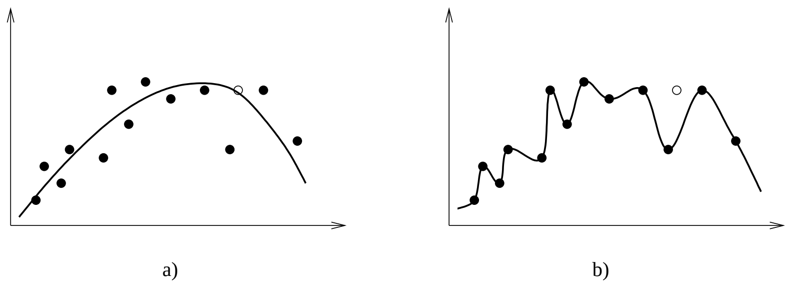


Figure 5.2: a) A good fit to a noisy data b) Overfitting on the same data

In the presence of noise, complexity measures discussed earlier are not sufficient anymore. In the absence of noise, the least complex solution perfectly fitting the data would be selected. If noise is involved however, this procedure could result in selecting solutions with poor generalization properties (overfitting problem). In order to select the best solution both the complexity and error (losses) have to be taken into account. One of the most often used approaches is to asses Kolmogorov complexity of these two factors and minimize their sum (minimum description length principle described in Section 1.1.2). In practice, encoding schemes tend to overestimate the number of bits used for hypothesis description compared to the description of errors. To compensate for this inaccuracy, the minimum description length equation is modified by introducing an empirical factor k:

$$MDL = k \cdot \mathit{hypothesis\ description\ length} + \mathit{error\ description\ length} \tag{5.1}$$

which was by default set up to 0.5. In our implementation we use cardinality (or log-functionality) cost measure as *hypothesis description length* and the approach presented in [102] to calculate *error description length.*

The number of errors when classifying a set of n objects can be in general encoded by an n-bit string of 0's and 1's, 0 corresponding to the correct classification, 1 to an error. The string of length n will always work fine but for some combinations of 0's and 1's we can encode the same information with smaller number of bits. The encoding procedure proposed in [102] is the following:

1. First encode the number of 1's, this requires $\log_2(n+1)$ bits (the number of ones can be in the range $0 \ldots n$)

2. Let the number of 1's be k. Then the number of strings of length n containing k 1's is equal to $\binom{n}{k}$ and we need $\log_2\left(\binom{n}{k}\right)$ bits to describe which one it is.

Hence, the total number of bits needed for the error encoding is equal to:

$$\log_2(n+1) + \log_2\left(\binom{n}{k}\right) \tag{5.2}$$

This equation can be approximated using Stirling's formula to obtain:

$$nH(k/n) + \log_2(n) - \frac{1}{2}(\log_2(k) + \log_2(n-k) + \log_2(2\pi)) + O(1/n) \tag{5.3}$$

where $H(k/n) = -k/n \log_2(k/n) - (1-k/n)\log_2(1-k/n)$ is the familiar Shannon's entropy function.

5.2.1 Lossy decomposition

The decomposition procedure described in Chapter 4 creates solutions which perfectly fit training data. In the presence of noise such solutions would very likely do poorly on the data not present in the training set. In order to address the overfitting problem we have to modify the decomposition procedure in such a way as to be able to create solutions with error level and complexity related to each other. We should be able to create simpler solutions at a cost of higher error rate on the training data and vice versa. In other words, we should be able to perform lossy decomposition.

A good place for modification of the decomposition procedure is where the compatibility graph is constructed (see Section 4.6.1 p.104). The graph in Figure 5.3*b* is constructed for data represented by the Karnaugh map in Figure 5.3*a*. It is a full graph with a weight assigned to every edge. Each weight takes values between 0.0 and 1.0 and is equal to the minimum number of mismatches between the columns connected by that edge, normalized to the maximum possible number of mismatches (number of rows). YES or NO selection criterion removes all the edges from the graph which weights are greater or equal to certain value d. Figure 5.3*c* shows the graph obtained for $d = 0.5$. Of course if we know the threshold value d before we start creating the graph, then the full graph doesn't have to be created at all.

The algorithm described above was developed for the general case of nominal data (as were the decomposition algorithms described in Chapter 4). For such data the variable values are mere symbols and no distance measure can be defined in such spaces. This is the most general approach for discrete data spaces. If the system

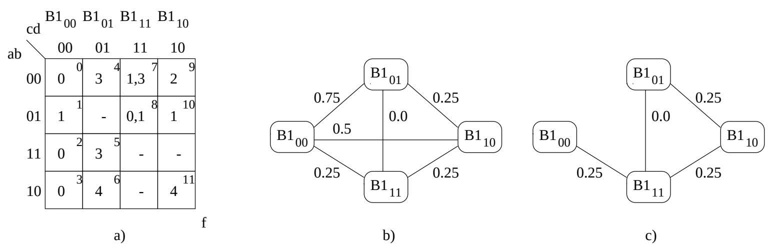


Figure 5.3: Compatibility graph for lossy decomposition in nominal spaces.

variables are metric and a distance can be computed, then the decomposition algorithms can be improved. For instance in the decomposition of probabilistic relations described in Section 4.3 (p.79), the frequency (probability) distribution variable is metric. The lossy decomposition algorithm for metric spaces is very similar to the one developed for the nominal spaces. The only difference is that the column compatibility is determined based not on the number of mismatches but on a distance metric defined for columns.

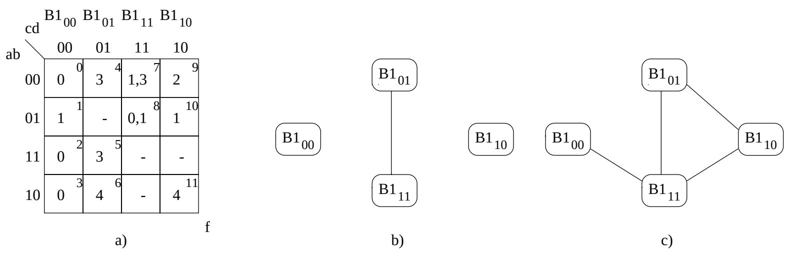


Figure 5.4: Compatibility graph for lossy decomposition in metric spaces.

Figure 5.4 shows a construction of compatibility graphs in two cases: nominal data without loss and metric data with loss. The graph in Figure 5.4*b* was constructed assuming nominal data space and no loss. Two columns are compatible only if they have a common value in every row of the table. The graph in Figure 5.4*c* was created assuming metric data space. In the example in Figure 5.4 two columns are considered to be compatible if the difference (distance) between values of the corresponding row cells is not greater than 1. The resulting graph (5.4*c*) can be covered by fewer cliques (2 vs. 3) than the graph for lossless case (Figure 5.4*b*) and results in simpler decomposed structure. Other distance metrics that could be used here are Euclidean distance and correlation coefficient.

5.3 Estimation of performance

In chapter 4 we created our new decomposer, ran it on several data files and got beautifully decomposed and simplified structures. Here, we want to evaluate it as a learning method. We want to know how good is it in learning underlying structures from data, how accurately the concepts discovered by decomposition describe the data, or how closely the network of decomposed blocks models the true structure of the system described by the data file. In other words we want to determine how much our model is in error when used to model the true system. The most common definition of an **error rate** is:

$$error\ rate = \frac{number\ of\ test\ cases\ in\ error}{total\ number\ of\ test\ cases} \tag{5.4}$$

This equation is very simple but there is one problem with it. The problem is how to select the test cases. If we use the whole data file to discover new concepts and use the same data file for testing them, then the error rate is called **apparent error rate** and it is very likely to be close to zero. This is perfect, one may say. Not really so. It is perfect if the data file contains a complete description (or close to it) of the underlying problem. But for real world problems it is almost never so. The common situation is that we only have a very small subset of the full description. We hope it captures all the essential features of the underlying system and try to build a model which would describe that system as closely as possible. Building a model which perfectly describes this limited sample does not necessarily mean that it will correctly model the whole system. What we really want is to be able to estimate a **true error rate**, i.e. the error rate obtained if the whole system were available for testing. So the question really is: How to evaluate a true error rate when only a limited sample of data is available? Below we will provide a short review of the most commonly used methods together with their range of application.

5.3.1 Trial-and-test error rate estimation

The simplest method of evaluation of a true error rate is to partition the data set into a training set and a testing set, develop a model using only data from the training set and estimate the true error rate based on evaluation of the model performance on the testing set. How close this estimation will be depends on the size of the testing set. If the size of the testing set is 1000 then the estimation is off the true error rate by no more than 1%. If the test sample size is 5000 then the estimate is virtually equal to the true error rate. Traditionally, a fixed percentage of data available is used for training and the remaining data used for testing. The usual split ratio is 2/3 and 1/3. However, if 1/3 of the data is greater than 1000 (or 5000) more of the data can be used for training.

5.3.2 Resampling techniques

If large amounts of data are available, a single trial-and-test method can be used with good accuracy. If we are short of data however, multiple trial-and-test experiments will do better.

Random subsampling

This method consists of repeating a single trial-and-test experiment many times and averaging error rates to get a true error rate estimate. Partitions for each trial-and-test experiment have to be determined independently and randomly. By averaging, this method can provide better true error estimate than a single trial-and-test experiment but it is more time consuming.

K-fold cross-validation

In k-fold cross-validation method the data is randomly partitioned into k subsets. A single trial-and-test experiment is repeated k times where each time different subset is selected as a testing set and the remaining data as a training set.

A special case of this method, for k equal to the sample size n, is called *leave-one-out method.* In each of n experiments $n - 1$ samples are used for training and 1 sample for testing. The method is known to be an almost unbiased estimator of the true error rate but is computationally expensive and is usually used only for small sample sizes.

Another special case of this method which is used when the data sets are large is the so called *inverse k-fold cross-validation method.* The procedure of data partitioning is the same as for k-fold cross-validation but the selected subset is used not for testing but for training. Since it is much smaller than the remaining data the training time can be significantly reduced for large data sets.

Which method to use?

Selection of the true error rate estimation method depends of the size of the available data set. Here are the guidelines [128]:

- For sample sizes greater than 100 use k-fold cross-validation method. The most often used are 10-fold cross-validation which is reliable for sample sizes of couple of hundreds and greater or leave-one-out (may be computationally expensive).
- For sample sizes less than 100 (but greater than 50) use leave-one-out method.

5.4 Experimental results

In this section we will present results of running concept on various benchmarks and compare it to the results obtained from C4.5, one of the best known learning programs in the machine learning domain. C4.5 is a program which constructs a decision diagram that best fits the data. Three different versions of the program will be used for comparison: *before pruning*, *after pruning*, and *rules*. The first version is the decision tree constructed directly from the data and may often overfit the data. The second version is a decision tree constructed from the first version by removing parts that contribute very little to the classification of unseen cases. The third version is constructed from the second version by extracting human comprehensible rules (*disjunctive normal form*) from the decision tree which may even further simplify the description of data.

In Table 5.1 we compare concept and the three versions of C4.5 on selected machine learning benchmarks from [121]. Only discrete variable benchmarks were selected for comparison (concept doesn't have a discretizer build in, C4.5 does). For each benchmark the number of inputs and data tuples is listed in the table. All the benchmarks in Table 5.1 have between 200 and 1000 data tuples, so according to the rules specified in Section 5.3 we used 10-fold cross-validation method for true error rate estimation. Underscore indicates the best result for a given benchmark. From this comparison we can see that concept outperforms both the first and the second versions of C4.5 in 6 out of 10 cases. It is outperformed by the third version of C4.5 in 5 out of 9 cases.

The results of comparison on four benchmarks `balance`, `monk2`, `parity`, and `tic-tac-toe` deserve an extra comment here. These are the cases where one of the programs was significantly better than the others. The benchmark `balance` models a balance scale, it can tip to the left, right, or be balanced. The four attributes are '`the left weight`', '`the left distance`', '`the right weight`', '`the right distance`'. Class assignment is determined based on comparison of the products '`the left weight * the left distance`' and '`the right weight * the right distance`'. This problem is characterized by a significant level of symmetry between variables and apparently concept is doing much better here than the other programs. The second benchmark where concept is significantly better is monk2. This problem was created to make it difficult to describe in *disjunctive normal form* or *conjunctive normal form*, it is similar to parity problems (symmetry again!) and concept can learn it without error whereas the other programs report significant ($\approx$ 30%) error. To fully confirm our observation that concept is significantly better on symmetrical problems we added a binary parity function parity to the set of benchmarks. For every combination of the train and test sets used in 10-fold cross-validation method concept was able to learn this function without errors. Other programs did poorly on this data set. The fourth data set, tic-tac-toe, encodes the complete set of possible board configurations at the end of tic-tac-toe games (3×3 board) and a set of simple rules can be extracted from C4.5 decision tree which describes this problem very accurately (0.6% vs. more than 10% error for the other methods).

	inputs/size	concept [%]	c4.5 [%]		
			unpruned	pruned	rules
audiology	69/200	35.0	28.0	$\underline{26.0}$	$\underline{26.0}$
balance	4/625	$\underline{3.3}$	31.5	35.2	22.7
breastc	9/699	9.1	6.0	5.4	$\underline{4.6}$
flag	28/194	56.8	28.4	$\underline{27.4}$	36.0
house-vote-84	16/435	6.7	5.0	$\underline{4.6}$	5.7
irish	4/500	$\underline{0.0}$	$\underline{0.0}$	$\underline{0.0}$	$\underline{0.0}$
monk1	6/556	$\underline{0.0}$	4.5	0.7	$\underline{0.0}$
monk2	6/601	$\underline{0.0}$	37.1	35.1	26.5
sensory	11/576	$\underline{75.7}$	82.6	80.9	76.5
tic-tac-toe	9/958	11.0	14.9	13.8	$\underline{0.6}$
parity	12/4096	$\underline{0.0}$	97.1	57.3	65.4

Table 5.1: 10-fold cross-validation true error estimation: machine learning benchmarks.

For benchmarks presented in Table 5.2 fewer cases were available so we used leave-one-out method for the true error estimation. The results for different programs are comparable but surprisingly the unpruned decision tree of C4.5 is the best and C4.5 rules the worst here.

For benchmarks presented in Table 5.3 more data were available so we used trial-and-test method for the true error estimation. The training and testing sets

	inputs/size	concept [%]	c4.5 [%]		
			unpruned	pruned	rules
lung-cancer	56/32	62.5	56.3	<u>53.1</u>	68.8
sleep	9/62	37.1	<u>30.6</u>	38.7	35.5
sponge	44/76	7.9	<u>6.6</u>	7.9	10.5
zoo	16/101	<u>4.9</u>	7.9	8.9	8.9

Table 5.2: leave-one-out true error estimation: machine learning benchmarks.

were generated randomly, 1/3 of the data for the testing set and remaining data for the training set. The results for different programs are comparable but concept is slightly better than the other programs here.

	inputs/size	concept [%]	c4.5 [%]		
			unpruned	pruned	rules
chess2	36/3196	2.3	1.1	<u>0.9</u>	<u>0.9</u>
mushroom	22/8124	<u>0.0</u>	<u>0.0</u>	<u>0.0</u>	0.2
nursery	8/12960	<u>0.0</u>	2.7	4.1	2.2

Table 5.3: test-and-error true error estimation: machine learning benchmarks.

To show the learning process in more detail, Figures 5.5, 5.6, and 5.7 show examples of learning curves of concept and C4.5 on the benchmarks we used for true error estimation. Learning curve is a plot of accuracy versus size of the learning set. Ten learning sets are drawn randomly for each data set in such a way that larger sets contain all the smaller ones. The plots are drawn after averaging ten learning curves for each program. On the plots solid line corresponds to concept, dotted line to C4.5 before pruning, dot-dashed line to C4.5 after pruning, and short-dashed line to C4.5 rules. Interestingly, all the learning curves have in most cases similar character. In most cases concept is doing slightly better than the three C4.5 algorithms. The exception is the benchmark `tic-tac-toe` where C4.5 rules is the best of four programs compared. For `balance` and `monk1` benchmarks concept learns much faster than C4.5. Notice that for `monk1` benchmark learning curve for concept is missing. It is not because we were not able to get it but because it is reduced to one point. For each of the ten learning curves concept accuracy was equal to 100% for the smallest learning set, the first point of the curve. The same situation is for the benchmark parity where accuracy of all C4.5 versions were about 50% and concept learned without error in the first step (10% subset of the whole data set).

Comparing results of the true error estimation and learning curves for the different programs we can notice certain discrepancy of the results. In many cases one program appears to be more accurate on the learning curve and yet another program has lower true error estimate for the same benchmark. These are all the cases where the differences in error and accuracy on the learning curve are not very large. For some benchmarks however the results remain consistent. These are all the cases where a given program is significantly better on both true error estimate and accuracy on the learning curve. The above observations may suggest that for the cases where the results are inconsistent the differences between the programs are insignificant and that they are doing equally well on these benchmarks.

The largest benchmarks used for experiments in this section, expressed in terms of the number of equivalent binary variables (independent), were: lung-cancer (112), sponge (78), audiology (77), flag (54), mushroom (48), chess2 (37), and breastc (30). The largest binary benchmark used for testing the decomposition procedure in Chapter 4 (Section 4.7.2) was gpio (131).

5.5 Summary

In this chapter we presented various issues related to learning systems (Section 5.1). We discussed the problem of noise in data and how it affects the learning process (Section 5.2). We also presented the most often used methods of the true error estimation of the learning systems (Section 5.3). And finally we performed various experiments on a set of commonly accepted benchmarks to asses the performance of the program concept we developed based on the theoretical considerations discussed in Chapters 3 and 4 (Section 5.4). We also ran the same experiments using

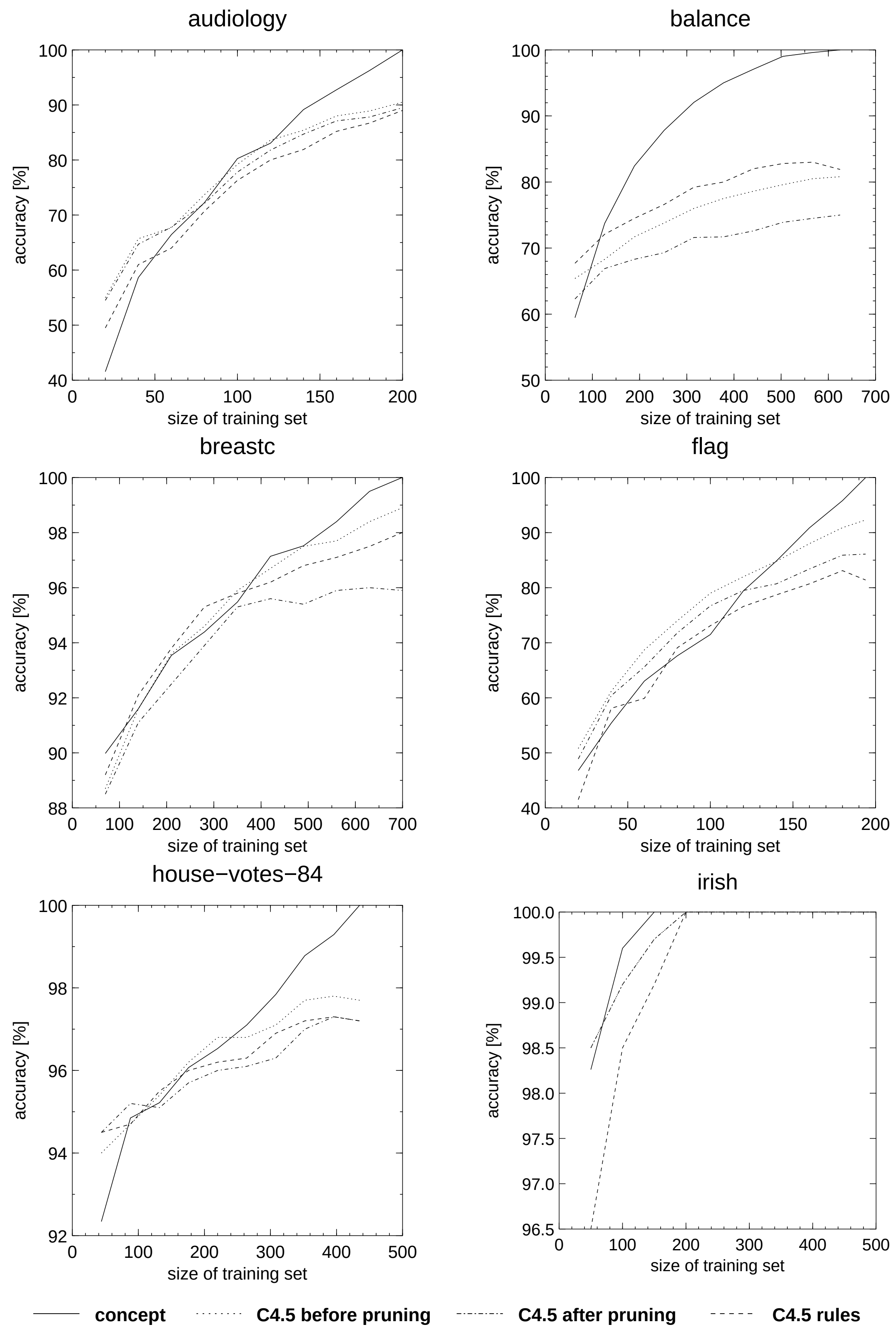


Figure 5.5: Learning curves for machine learning benchmarks.

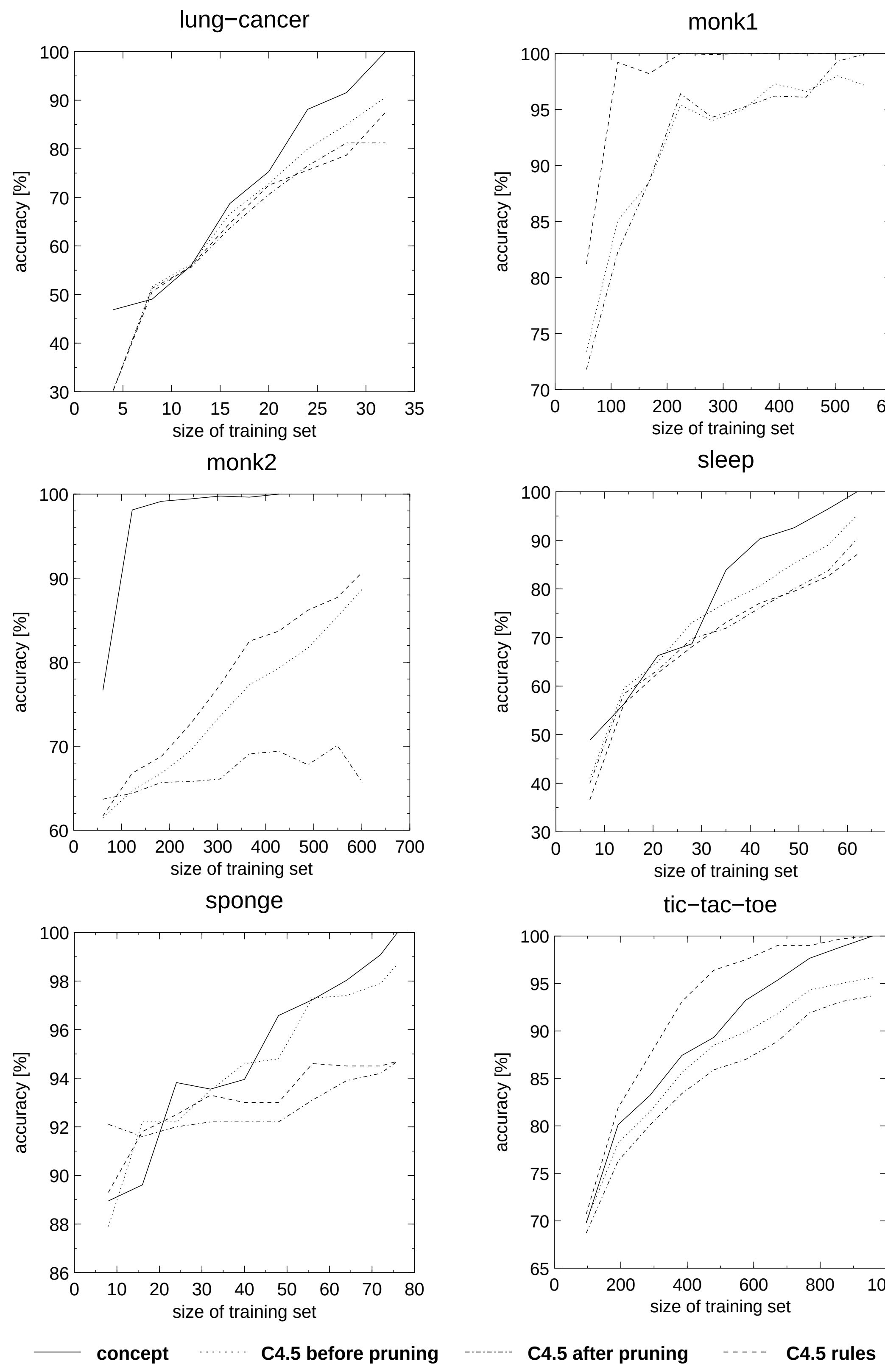


Figure 5.6: Learning curves for machine learning benchmarks (cont.).

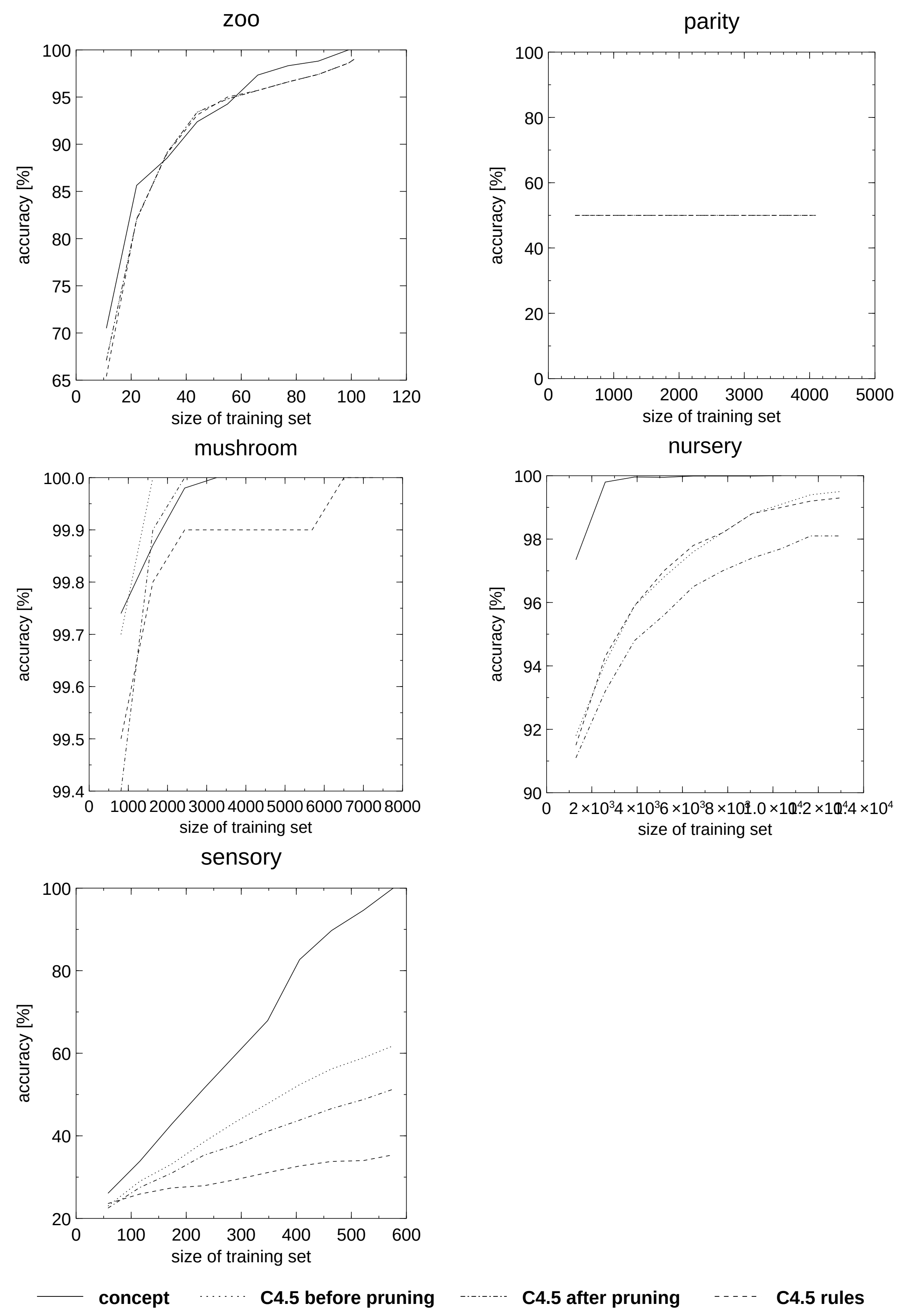


Figure 5.7: Learning curves for machine learning benchmarks (cont.).

one of the best known machine learning programs C4.5 and compared the results. The results of comparison were quite interesting. While C4.5 rules was significantly better on the `tic-tac-toe` benchmark (C4.5 rules is known for doing very well on this benchmark), concept turned out to be significantly better than C4.5 on the benchmarks with certain symmetry of variables (`balance`, `monk2`, `parity`) and comparable on the others. Also, concept's learning was very fast on these benchmarks, and only a small, randomly selected fraction of data was enough to learn the underlying data patterns. This shows the ability of concept to discover regular patterns from even a small fraction of data (`parity` function represented as a Karnaugh map shows a chess-board like pattern for instance).

Chapter 6

CONCLUSIONS

6.1 Conclusions

The two main goals of this dissertation were:

1. Study the applicability of decomposition for machine learning and data mining.

2. Develop a memory efficient data structure which would facilitate extracting knowledge from large data bases.

Since decomposition is a fairly general concept, we decided to focus our attention on one specific type of decomposition: Ashenhurst-Curtis type serial decomposition. The reason for selecting this particular type of decomposition was that theoretical settings for this method were already developed for functions represented by Karnaugh maps and could serve as a good starting point for adapting the theory to a different data structure. The decomposition was used as a tool for extracting new concepts from data and combine them into a simpler but equivalent description. The general optimization framework for this procedure was established based on two general principles: Occam razor's principle and minimum description length principle.

The second goal was motivated by the fact that data mining is a process of extracting knowledge from (large) databases, most of which are relational type and distributed databases. A good representation matching this type of data and facilitating the knowledge extraction process seemed to be a crucial part of the whole system. The data structure developed in this dissertation, lr-partitions, seems to naturally match the structure of relational distributed databases. Lr-partitions are conceptually simple and almost all the operations on them can be reduced to set operations on integer numbers. It was experimentally shown that lr-partitions can outperform decision diagram type representations in terms of memory requirements and speed for many applications.

Decomposition algorithms were developed using lr-partitions data structure and implemented in program concept. The decomposition process is controlled by a cost function which is used as a complexity measure. Three different cost functions were analyzed in the dissertation: cardinality, functionality, and the number of degrees of freedom. They are all conceptually very close and provide similar results in most cases. For some percentage of cases however, functionality cost measure seems to provide more accurate assessment of complexity than the other two. In the decomposition process new variables are created that serve as links between decomposed blocks. They correspond to new concepts extracted from data and the process of extraction can be reduced to the process of graph coloring of incompatibility graph or clique covering of compatibility graph. These two approaches are complementary. The current implementation uses the clique covering method, which is more efficient than graph coloring for lower percentage of edges in the compatibility graph. For higher percentage of edges however, the graph coloring method would be faster and a combination of both would definitely speed up the concept extraction process.

We implemented two approaches for decomposition of discrete directed relations: bottom-up and top-down. The first one performs decomposition by iteratively extracting the smallest possible block from larger blocks. The second one splits a block into two smaller blocks of approximately equal size. It was shown that this approach usually leads to better solutions in a smaller number of iterations. However, as the number of variables grows, the concept extraction graph may become very large and its coloring or covering be very time consuming. In the bottom-up approach the graphs are usually much smaller but the process of their creation is a time bottleneck here. Overall, the top-down approach usually provides better results for smaller data sets and bottom-up approach had to be used for larger ones due to the inefficiency of graph covering algorithms used in the current implementation.

Two new cost measures were developed in this dissertation to control the decomposition process: cardinality and functionality. They are both based on similar measures developed in the past for binary functions. The cardinality cost measure is closely related to the maximum number of tuples that can be realized by a given structure to describe the data. The functionality cost measure is equal to the total number of functions that can be realized by a given structure. The functionality cost measure provides finer than cardinality distinction between decomposed structures but is more computationally expensive. There exists an interesting relationship between both measures: $\log_2$ of functionality cost measure (log-functionality) is equal to the cardinality cost measure for a single block structure. This relationship doesn't hold for decomposed structures. The values of both measures are close but not equal. The value of log-functionality is smaller than the value of cardinality cost measure for multi-block decomposed structures.

A new variable partitioning algorithm was developed to generate a limited set of decomposed structures at each decomposition step. The algorithm is based on uncertainty measure and is used to heuristically select the most promising (in terms of the cost) structures for decomposition. Limiting the number of structures is necessary because the number of all possible structures is exponential in the number of variables. The new algorithm was compared to another variable partitioning procedure and proved to be more efficient on most of the test cases used. To increase the quality of the selection, both algorithms are used in the software implementation.

As a part of the decomposition procedure a novel algorithm for removing vacuous variables was implemented. It proved to be very useful; it significantly simplifies the decomposition process, especially for real life Machine Learning data.

Usefulness of this procedure for MCNC benchmarks (binary functions) was not as significant as in the Machine Learning area. This can be explained by the fact that most of these binary functions were descriptions of already highly optimized combinational circuits.

The decomposition strategy used in this dissertation provided means for a development of a new discretization algorithm. The discretization procedure is automatically performed when a single input smallest decomposition block is selected by the user. Each variable is analyzed and a function is developed which provides a mapping from existing to a new, better, discretization scheme for that variable. The discretization procedure takes advantage of the existing relationship between the independent variable being discretized and the dependent variables. Our procedure adapts itself to the problem in question; it tries to optimize the solution within the framework of existing relationships (constraints) between the system variables. To perform a discretization of continuous variables we have to perform a preliminary, rough discretization, using for instance the uniform binning method. The number of bins selected for the primary discretization has to be large enough for the secondary algorithm to discover an optimal discretization scheme.

The main decomposition procedure proposed in this dissertation was developed for non-probabilistic discrete directed relations and it constitutes an essential part of all other decomposition procedures developed in this dissertation. The decomposition algorithms developed for probabilistic and non-probabilistic discrete neutral relations are based on this procedure. By using our new optimizing discretization scheme the above algorithms can also be applied to continuous systems.

If available data are noisy or incomplete, then a theory which perfectly fits the training data may not perform well on unseen data samples. In such cases a better solution can often be developed by selecting a theory that takes into account not only an error rate but also the complexity of the solution (minimum description length principle). For this scheme to be implemented, the means have to be provided to control the decomposition process in terms of complexity and error rate. Two such algorithms were developed, one for nominal and the other for metric data.

Implementation of these ideas resulted in the program concept that uses lr-partitions as a data structure and decomposition as an inference method. Classifiers developed by concept were compared to the results obtained from one of the best Machine Learning programs, C4.5. In many cases the results obtained from both programs were comparable, but concept appeared to significantly outperform C4.5 on the problems with regular, repeating patterns. As a result of decomposition, a less complex structure of blocks is being discovered by concept. Each block corresponds to a concept extracted from the original data. These new concepts can facilitate the description of the data and provide for better understanding of both the structure and the meaning of underlying ideas.

6.2 Future work

While we attempted to perform as complete a job as possible many questions still remain open for further investigation and research.

Efficiency of decomposition algorithms proposed in this dissertation depends on the efficiency of clique covering (graph coloring) algorithms. This is especially important for the top-down approach to decomposition where compatibility (incompatibility) graphs may become large and efficient algorithms and their implementation is crucial for the overall decomposition performance.

Lossy decomposition algorithms were developed to provide a means for dealing with incomplete and noisy data. In the current implementation, the trade-off between complexity and error rate is determined by a user specified parameter. A good automatic optimization procedure would definitely help in finding solutions with better predictive accuracy.

Two algorithms for decomposition of probabilistic relations were proposed in this dissertation. One of them directly uses the algorithm developed for directed relations for decomposition of probability density functions. The other, although based on the same principles, is different and was not thoroughly tested on real life data sets yet.

The variable partitioning algorithm developed in this dissertation is very simple and quite effective. However, when many variables appear to be equally good at the consecutive ordering step, then the choice of one of them is basically random. Some sort of simple look-ahead algorithm would provide better ordering in these cases. Also, the current algorithm doesn't take into account any higher level relations between variables. Exploring this way of thought might lead to further improvement.

We experimented with two set representations for lr-partitions blocks: bit-sets and binary decision diagrams. Any set representation could be used however. Which one to choose and for which application may still be further explored. Also, integrating a direct interface to commonly used data base format(s) would definitely make the whole system better suited for exploration of real life data sets. Also, it is not clear yet how to effectively select the subsets of variables for lr-partitions. Two possibilities were investigated so far: lr-partitions based on single variables and lr-partitions based on all variables. It is not clear, however, how to optimize the size and the content of the sets of variables lr-partitions are based on. This is still an open research question.

The decomposed structure is an Acyclic Directed Graph structure with nodes corresponding to functional or relational blocks. In this dissertation we didn't explore the possibility of using different representations for these blocks. Each block however can be represented according to the kind of concept it represents. For some blocks decision diagrams would be a good choice, for some others neural networks or set of cubes or lr-partitions, etc. This kind of adjustment performed as a final step following the decomposition process could probably further improve classification accuracy of the whole structure.

Bibliography

[1] Reconstructibility analysis bibliography. *Int. J. General Systems*, 24:225–229, 1996.

[2] Y.S. Abu-Mostafa. *Complexity in Information Theory.* Springer-Verlag, New York, 1988.

[3] S.B. Abugharbieh and S.C. Lee. A fast algorithm for disjunctive decomposition of m-valued functions. Part I: The decomposition algorithm. In *Proc. 23th ISMVL*, pages 118–125, 1993.

[4] S.B. Abugharbieh and S.C. Lee. A fast algorithm for disjunctive decomposition of m-valued functions. Part II: Time complexity analysis. In *Proc. 23th ISMVL*, pages 126–131, 1993.

[5] T. Arciszewski, R.S. Michalski, and J. Wnek. Constructive induction: the key to design creativity. Reports on machine learning and inference laboratory, mli 95-6, George Mason University, April 1995.

[6] W.R. Ashby. Measuring the internal informational exchange in a system. *Cybernetica*, 1(8):5–22, 1965.

[7] R. L. Ashenhurst. The decomposition of switching functions. In *Proc. Int. Symp. Theory of Switching, Part I*, pages 74–116, Ann. Comput. Lab. Harvard Univ., 1959.

[8] L. Nguyen at al. Palmini - fast boolean minimizer for personal computers. In *DAC*, 1987.

[9] M. Ciesielski at al. Multiple-valued minimization based on graph coloring. In *International Conference on Computer Design*, 1989.

[10] M. Perkowski at al. Kuai-exact: a new approach for multi-valued logic minimization in vlsi synthesis. In *IEEE International Symposium on Circuits and Systems*, 1989.

[11] R. K. Brayton at al. A new exact minimizer for two-level logic synthesis. In T. Sasao, editor, *Logic Synthesis and Optimization*, pages 1–31. Kluwer Academic Publishers, 1993.

[12] D.J. Bartholomew. *Latent variable models and factor anaysis.* Charles Griffin, London, 1968.

[13] K.P. Bennett and O.L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

[14] E. Bloedorn and R.S. Michalski. Data driven constructive induction in AQ17-PRE: A method and experiments. In *Proceedings of the Third International Conference on Tools for AI*, San Jose, CA, 1991.

[15] M. Bohanec and V. Rajkovic. Knowledge acquisition and explanation for multi-attribute decision making. In *Proc. of the 8th Int. Workshop on Expert Systems and their Applications*, pages 59–78, Avignon, France, 1988.

[16] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Proc. of 27th Design Automatiom Conference*, pages 40–45, June 1990.

[17] I. Bratko. Private communication, 1996.

[18] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *Trans. on Comput.*, C-35(8):667–691, 1986.

[19] R.E. Bryant. On the complexity of VLSI implementation and graph representations of boolean functions with application to integer multiplication. *IEEE Trans. on Computers*, 40:205–213, 1991.

[20] J. Catlett. On changing continuous attributes into ordered discrete attributes. In Y. Kodratoff, editor, *Proc. of the European Working Session on Learning*, pages 164–178, Berlin, Germany, 1991. Springer-Verlag.

[21] G. J. Chaitin. On the length of programs for computing finite binary sequences. *J. ACM*, 13:547, 1966.

[22] G. J. Chaitin. On the difficulty of computations. *IEEE Trans. Info. Theor.*, IT-16:5, 1970.

[23] G. J. Chaitin. Information-theoretic limitations of formal systems. *J. ACM*, 21:403, 1974.

[24] G. J. Chaitin. Algorithmic entropy of sets. *Comput and Math. Appls.*, 2:233, 1976.

[25] G. J. Chaitin. Algorithmic information theory. *IBM J. Res. Develop.*, pages 350–359, July 1976.

[26] R.C. Conant. Detecting subsystems of a complex system. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 350–353, 1972.

[27] R.C. Conant. Set-theoretic structure modeling. *International Journal of General Systems*, 7(38):93–107, 1981.

[28] R. Cummings. *The nature of psychological explanation.* MIT Press, Cambridge, MA, 1983.

[29] H. A. Curtis. *A New Approach to the Design of Switching Circuits.* Van Nostrand, Princeton, 1962.

[30] A.L. de Oliveira. *Inductive Learning by Selection of Minimal Complexity Representations.* PhD thesis, University of California at Berkeley, 1994.

[31] M. DesJardins and D. F. Gordon. Evaluation and selection of biases in machine learning. *Machine Learning Journal*, 20:5–21, 95.

[32] S. Devadas. Comparing two-level and ordered binary decision diagram representations of logic functions. *IEEE Trans. on CAD*, 12(5):722–723, May 1993.

[33] G. Drastal, G. Czako, and S. Raatz. Induction in an abstraction space: A form of constructive induction. In *Proceedings of the IJCAI-89*, pages 708–712, Detroit, MI, 1989. Morgan Kaufmann.

[34] E.V. Dubrova, J.C. Muzio, and B. von Stengel. Finding composition trees for multiple-valued functions. In *Proc. 27th ISMVL*, 1997.

[35] W. Emde, C.U. Habel, and C.R. Rollinger. The discovery of the equator or concept driven learning. In *Proceedings of IJCAI-83*, pages 455–458, Karlsruhe, Germany, 1983. Morgan Kaufmann.

[36] B.C. Falkenhainer and R.S. Michalski. Integrating quantitative and qualitative discovery in the ABACUS system. In Y. Kodratoff and R.S. Michalski, editors, *Machine Learning: An Artificial Inteligence Approach*, volume 3. Morgan Kaufmann, Palo Alto, CA, 1990.

[37] K.Y. Fang and A.S. Wojcik. Modular decomposition of combinational multiple-valued circuits. *IEEE Transactions on Computers*, 37(10):1293–1301, 1988.

[38] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overwiev. In Fayaad at al., editor, *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1996.

[39] U.M. Fayyad, P. Smyth, N. Weir, and S. Djorgovski. Automated analysis and exploration of image databases: Results, progress, and challenges. *Journal of Intelligent Information Systems*, (4):1–19, 1993.

[40] C. Files, R. Drechsler, and M. Perkowski. Functional decomposition of mvl

functions using multi-valued decision diagrams. In *Proc. of ISMVL'97*, pages 27–32, Halifax, Nova Scotia, Canada, May 28-30 1997.

[41] N.S. Flann. Improving problem solving performance by example guided reformulation of knowledge. In D.P. Benjamin, editor, *Change of Representation and Iductive Bias*. Kluwer Academic, Boston, MA, 1990.

[42] G.H. Greene. The Abacus 2 system for quantitative discovery: Using dependencies to discover non-linear terms. Reports of machine learning and inference laboratory, mli 88-4, Center for Artificial Intelligence, George Mason University, Fairfax, VA, 1988.

[43] S. Grygiel and M. Perkowski. New compact representation of multiple-valued functions, relations, and non-deterministic state machines. In *ICCD-98*, pages 168–174, Austin, Texas, October 1998.

[44] S. Grygiel, M. Perkowski, M. Marek-Sadowska, T. Luba, and L. Jozwiak. Cube diagram bundles: a new representation of strongly unspecified multiple-valued functions and relations. In *Proc. of ISMVL'97*, pages 287–292, Halifax, Nova Scotia, Canada, May 28-30 1997.

[45] R.V.L. Hartley. Transmission of information. *The Bell Systems Technical Journal*, 7(3), 1928.

[46] J. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975.

[47] T. Kalganova. Combinational multiple-valued circuit design by generalized disjunctive decomposition. In *Proc. of the Europ. Conf. on Circuit Theory and Design, ECCTD'97*, Budapest, Hungary, 1996.

[48] T. Kalganova. Functional decomposition methods for multiple-valued logic functions and its system. In *Proc. of the 3rd Int. Conf. on Applications of Computer Systems*, pages 75–82, Szczecin, Poland, 1996.

[49] T. Kalganova. The studing of the finctional decomposition methods for r-valued logic functions in the logic design courses. In *Proc. of the 3nd Int. Conf. on the New Information Technologies in Education*, volume 2, pages 150–158, Minsk, Bielarus, 1996.

[50] R.M. Karp. Functional decomposition and switching circuit design. *SIAM Journal on Applied Mathematics*, 11(2):291–335, June 1963.

[51] G.J. Klir. Identification of generative structures in empirical data. *International Journal of General Systems*, (3):89–104, 1976.

[52] G.J. Klir. *Architecture of Systems Problem Solving.* Plenum Press, New York, 1985.

[53] C.A. Knoblock. A theory of abstraction for hierarchical planning. In D.P. Benjamin, editor, *Change of Representation and Inductive Bias.* Kluwer Academic, Boston, MA, 1990.

[54] C.A. Knoblock, S. Minton, and O. Etzioni. Integrating abstraction and explanation based learning in PRODIGY. In *Proceedings of AAAI-91*, pages 541–546. AAAI Press/MIT Press, 1991.

[55] R. Kohavi. Bottom-up induction of oblivious read-once decision graphs. In *Europeean Conference on Machine Learning*, 1994.

[56] M.M. Kokar. Discovering functional formulas through changing representation base. In *Proceedings of the AAAI-86*, pages 455–459, Philadelphia, PA, 1986.

[57] A. N. Kolmogorov. On tables of random numbers. *Sankhya*, A25:369, 1963.

[58] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Prob. Info. Transmission*, 1(1):1, 1965.

[59] A. N. Kolmogorov. Logical basis for information theory and probability theory. *IEEE Trans. Info. Theor.*, IT-14:662, 1968.

[60] I. Kononenko. Inductive and bayesian learning in medical diagnosis. *Applied Artificial Intelligence*, (7):317–337, 1993.

[61] K. Krippendorff. On the identification of structures in multivariate data by the spectral analysis of relations. In *Proc. 23th Annual Meeting of SGSiL*, Houston, Texas, 1979.

[62] K. Krippendorff. *Information Theory: Structural Models for Qualitative Data.* Sage Publications, Inc., 1986.

[63] Y. T. Lai, K.R. Pan, M. Pedram, and S. Vrudhula. FGMap: A technology mapping algorithm for look-up table type FPGA synthesis. In *Proc. 30-th DAC*, pages 642–647, 1993.

[64] P. Langley, G.L. Bradshaw, and H.A. Simon. Rediscovering chemistry with the BACON system. In R.S. Michalski, J.G. Carbonell, and T.M Mitchell, editors, *Machine Learning: An Artificial Inteligence Approach.* Morgan Kaufmann, Los Altos, CA, 1983.

[65] P. Langley, H.A. Simon, G.L. Bradshaw, and J.M. Zytkow. *Scientific Discovery: Computational Explorations of the Creative Process.* MIT Press, Cambridge, MA, 1987.

[66] J.B. Larson and R.S. Michalski. Inductive inference of VL decision rules. *ACM SIGART Newsletter*, (63):38–44, 1977.

[67] P.F. Lazrsfeld and N.W. Henry. *Latent structure anaysis.* Houghton Mifflin, Boston, MA, 1968.

[68] D.B. Lenat. On automated scientific theory formation: A case study using AM program. In *Machine Intellience*, volume 9. Halsted Press, New York, 1977.

[69] D.B. Lenat. Learning from observation and discovery. In *Machine Learning: An Aritficial Intellience Approach*. Morgan Kaufmann, Los Altos, CA, 1983.

[70] G.G. Lendaris. On the definiton of self-organizing systems. *Proceedings IEEE*, 52:324–325, March 1964. `http://www.sysc.pdx.edu/res_nnets.html`

[71] G.G. Lendaris and G.L. Stanley. On the structure-dependant properties of adaptive logic networks. Technical report, GM Defense Research Laboratories, Santa Barbara, California, July 1963.

[72] G.G. Lendaris and G.L. Stanley. Self-oranization: meaning and means. In J. Spiegel and D. Walker, editors, *Proceedings of the Second Congress*, Information System Sciences. Spartan Books, Baltimore, 1965.

[73] G.G. Lendaris and G.L. Stanley. Structure and constraints in discrete adaptive networks. In *National Electronics Conference*, volume XXI, 1965.

[74] M. Li and P. Vitanyi. *An Introduction to Kolmogorow Complexity and its Applications*. Springer-Verlag, 1997.

[75] T. Luba. Decomposition of multiple-valued functions. In *Proc. 25th ISMVL*, pages 256–261, 1995.

[76] K.K. Maitra. Cascaded switching networks of two-input flexible cells. *IRE Transactions on Electronics Computers*, April 1962.

[77] P. Martin-Löf. The definition of random sequences. *Info. Control*, 9:602, 1966.

[78] P. Martin-Löf. Algorithms and randomness. *Intl. Stat. Rev.*, 37:265, 1969.

[79] P. Martin-Löf. On the notion of randomness. In A. Kino J. Myhill and R. E. Vesley, editors, *Intuitionism and Proof Theory*, page 73. North-HollandPublishing Co., Amsterdam, 1970.

[80] C. Matheus. *Feature Construction: An Analytic Framework and Application to Decision Trees.* PhD dissertation, University of Illinois, Urbana-Champaign, 1989.

[81] Logic Synthesis and Optimization Benchmarks. `ftp://ftp.mcnc.org/pub/benchmark/Benchmark_dirs/LGSynth91/twolexamples`

[82] J. Michael. Validation, verification, and experimentation with Abacus 2. Reports of machine learning and inference laboratory, mli 91-8, Center for Artificial Intelligence, George Mason University, Fairfax, VA, 1988.

[83] R.S. Michalski. Pattern recognition as knowledge-guided computer induction. Technical report no. 927, Department of Computer Science, University of Illinois, Urbana-Champaign, 1978.

[84] R.S. Michalski. A theory and methodology of inductive learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann, Los Altos, CA, 1983.

[85] R.S. Michalski and J.B. Larson. Inductive inference of VL decision rules. In *Workshop in Pattern-Directed Inference Systems*, Hawaii, May 1977. Also published in SIGART Newsletter, ACM No. 63, pp. 38-44, June 1977.

[86] G.A. Miller and W.G. Madow. On the maximum likelihood estimate of the Shannon-Wiener measure of information. Technical Report TR-54-75, Air Force Cambridge Research Center, Washington, DC, 1954.

[87] S. Minato. Graph-based representations of discrete functions. In *Proc. Reed-Muller'95 Workshop*, pages 1–10, Chiba, Japan, August 1995.

[88] M.L. Minsky. Problems of formulation for artificial intelligence. In R. E. Belman, editor, *Proc. of Symposia in Applied Mathematics XIV*, page 35, Providence, RI, 1962.

[89] T.M. Mitchell, P.E. Utgoff, and R. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann, Los Altos, CA, 1983.

[90] K. Morik. Sloppy modeling. In K. Morik, editor, *Knowledge Representation and Organization in Machine Learning.* Springer-Verlag, Berlin Heidelberg, 1989.

[91] S. Muggleton. Duce, and oracle-based approach to constructive induction. In *Proceedings of IJCAI-87*, pages 287–292, Milan, Italy, 1987. Morgan Kaufmann.

[92] S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352, Ann Arbor, MI, 1988. Morgan Kaufmann.

[93] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, (5):71–99, 90.

[94] M. Perkowski and S. Grygiel at. al. A survey of literature on function decomposition. Technical report, Portland State University, Electrical Engineering Department, Portland, OR, September 1995.

[95] M. Perkowski, M. Burns, T. Luba, S. Grygiel, C. Stanley, R. Price, Z. Wang, J. Lu, P. Burkey, D. Manoharan, and S. Mohammad. Development of search strategies for MULTIS. Technical report, Portland State University, Electrical Engineering Department, Portland, OR, December 1995.

[96] M. Perkowski, T. Luba, S. Grygiel, P. Burkey, M. Burns, N. Iliev, M. Kolsteren, R. Lisanke, R. Malvi, Z. Wang, H. Wu, F. Yang, S. Zhou, and J.S. Zhang. Unified approach to functional decompositions of switching functions. Technical report, Portland State University, Electrical Engineering Department, Portland, OR, June 1995.

[97] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Zhi Wang, and Jin S. Zhang. Decomposition of multiple-valued relations. In *Proc. of ISMVL'97*, pages 13–18, Halifax, Nova Scotia, Canada, May 28-30 1997.

[98] M. Perkowski, T. Ross, D. Gadd, J. A. Goldman, and N. Song. Application of ESOP minimization in machine learning and knowledge discovery. In *Proc. Reed-Muller'95 Workshop*, pages 102–109, Chiba, Japan, August 1995.

[99] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, Los Altos, California, 1992.

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

[101] J.R. Quinlan. Comparing connectionist and symbolic learning methods. In S.J. Hanson, G.A. Drastal, and R.L. Rivest, editors, *Volume I: Constraints and Prospects*, Computational Learning Theory, chapter 15, pages 445–456. MIT Press, 1994.

[102] J.R. Quinlan and R.L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[103] L. De Raedt and M. Bruynooghe. Constructive induction by analogy. In *Proceedings of EWSL-89*, pages 189–200, Montpellier, France, 1989. Pitman.

[104] L. Rendell. Substantial constructive induction using layered information compression: Tractable feature formation in search. In *Proceedings of IJCAI-85*, pages 650–658, 1985.

[105] John Riordan. *An Introduction to Combinatorial Analysis.* John Wiley & Sons, Inc., New York, 1958.

[106] J. Rissanen. Modeling by the shortest data description. *Automatica-J.IFAC*, (14):465–471, 1978.

[107] R.L. Rudell and A. Sangiovanni-Vincentelli. Multiple-Valued Minimization for PLA Optimization. *IEEE Transactions on CAD*, CAD-6(5):727–750, September 1987.

[108] T. Sasao. An application of multiple-valued logic to a design of programmable logic arrays. In *Proc. 18th Int. Symp. Mult. Valued Logic*, 1978.

[109] T. Sasao. Multiple-valued decomposition of generalized boolean functions and the complexity of programmable logic arrays. *IEEE Transactions on Computers*, C-30:635–643, September 1981.

[110] T. Sasao. FPGA design by generalized functional decomposition. In T. Sasao, editor, *Logic Synthesis and Optimization*, pages 233–258. Kluwer Academic Publishers, 1993.

[111] J.C. Schlimmer. Learning and representation change. In *Proceedings of AAAI-87*, pages 511–515. Morgan Kaufmann, 1987.

[112] C.E. Shannon and W. Weaver. *The Mathematical Theory of Communication.* University of Illinois Press, 1975 (first published in 1949).

[113] V.Y. Shen, A.C. McKellar, and P. Weiner. A fast algorithm for the disjunctive decompositon of switching functions. *IEEE Transactions on Computers*, C-20(3), March 1971.

[114] V. Shmerko, L. Jozwiak, and industry. Private communication, 1996.

[115] R. J. Solomonoff. Complexity based induction systems: Comparisons and convergence theorems. Report rr-329, Rockford Research, Cambridge, MA, August 1976.

[116] R.J. Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Inform. Contr.*, 7:1–22, 224–254, 1964.

[117] A. Srinivasan, T. Kam, S. Malik, and R. Brayton. Algorithms for discrete function manipulation. In *IEEE International Conference on CAD*, pages 92–95, 1990.

[118] Steinbach, Hesse, Kempe, Rhode, and Barthel. Papers and discussions at the 2nd workshop boolesche probleme. Freiberg, Germany, 19-20 September 1996.

[119] B. Steinbach and A. Wereszczynski. Synthesis of multi-level circuits using exor-gates. In *Proc. Reed-Muller'95 Workshop*, Chiba, Japan, August 1995.

[120] Y.H. Su and P.T. Cheung. Computer minimization of multiple-valued switching functions. *IEEE Transactions on Computers*, C-21:995–1003, 1972.

[121] U.C. Irvine, Repository of Machine Learning Databases and Domain Theories. `ftp://ftp.ics.uci.edu/pub/machine-learning-databases/`

[122] P.E. Utgoff. *Shift of Bias for Inductive Concept Learning.* PhD dissertation, Rutgers University, 1984.

[123] P.E. Utgoff. Shift of bias for inductive concept learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2. Morgan Kaufmann, Los Altos, CA, 1986.

[124] L.G. Valiant. A theory of the learnable. *Comm. ACM*, (27):1134–1142, 1984.

[125] K.M. Walliuzzaman and Z.G. Vranesic. On decomposition of multiple-valued switching functions. *Computer Journal*, 13:359–362, 1970.

[126] W. Wan and M. Perkowski. A new approach to the decomposition of incompletely specified multi-output function based on graph coloring and local transformations and its application to FPGA mapping. In *Proc. Euro-DAC*, pages 230–235, 1992.

[127] Wei Wan. A new approach to the decomposition of incompletely specified functions based on graph coloring and local transformation. Master's thesis, Portland State University, May 1992.

[128] S. M. Weiss and C. A. Kulikowski. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neral Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1991.

[129] J. Wnek and R.S. Michalski. Hypothesis-driven constructive induction in AQ17: A method and experiments. In *Proceedings of IJCAI-91 Workshop on Evaluating and Changing Representation in Machine Learning*, pages 13–22, Sydney, Australia, 1991.

[130] J. Wnek and R.S. Michalski. Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning*, (14):139–168, 1994.

[131] W.H. Wolberg and O.L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. In *Proceedings of the National Academy of Sciences*, volume 87, pages 9193–9196, U.S.A., December 1990.

[132] S. Wrobel. Demand-driven concept formation. In K. Morik, editor, *Knowledge Representation and Organization in Machine Learning*. Springer-Verlag, Berlin Heidelberg, 1989.

[133] B. Zupan. *Machine Learning based on function decomposition*. PhD thesis, University of Ljubljana, 1997.

[134] B. Zupan and M. Bohanec. Learning concept hierarchies from examples by function decomposition. Technical report, J. Stefan Institute, Ljubljana, 1996.

[135] M. Zwick. Control uniqueness in reconstructability analysis. *International Journal of General Systems*, 23(2), 1995.

[136] M. Zwick. Set-theoretic reconstructability of elementary cellular automata. *Advances in System Science and Application, Special Issue I*, pages 31–36, 1995.

[137] M. Zwick. Whole and parts in general systems methodology. In G. Wagner, editor, *Evolutionary Biology and Characterictics.* Academic Press, 1999.