On Two Kinds of Dataset Decomposition

Pavel Emelyanov[1,2(✉)]

[1] A.P. Ershov Institute of Informatics Systems,
Lavrentiev av. 6, 630090 Novosibirsk, Russia
[2] Novosibirsk State University, Pirogov st. 1, 630090 Novosibirsk, Russia
emelyanov@mmf.nsu.ru

Abstract. We consider a Cartesian decomposition of datasets, i.e. finding datasets such that their unordered Cartesian product yields the source set, and some natural generalization of this decomposition. In terms of relational databases, this means reversing the SQL `CROSS JOIN` and `INNER JOIN` operators (the last is equipped with a test verifying the equality of a tables attribute to another tables attribute). First we outline a polytime algorithm for computing the Cartesian decomposition. Then we describe a polytime algorithm for computing a generalized decomposition based on the Cartesian decomposition. Some applications and relating problems are discussed.

Keywords: Data analysis · Databases · Decision tables
Decomposition · Knowledge discovery · Functional dependency
Compactification · Optimization of boolean functions

1 Introduction

The analysis of datasets of different origins is a most topical problem. Decomposition methods are powerful analysis tools in data and knowledge mining as well in many others domains. Detecting the Cartesian property of a dataset, i.e. determining whether it can be given as an unordered Cartesian product of two (or several) datasets, as well as its generalizations, appears to be important in at least four out of the six classes of data analysis problems, as defined by the classics in the domain [9], namely in anomaly detection, dependency modeling, discovering hidden structures in datasets and constructing a more compact data representation. Algorithmic treatment this property has interesting applications, for example, for relational databases, decision tables, and some other table–based modeled domains, such as Boolean functions.

Let us consider the Cartesian product $\times$ of two relations given in the form of tables in Fig. 1. It corresponds to the SQL–operator `T1 CROSS JOIN T2`. In the first representation of the product result, where the "natural" order of rows and

This work is supported by the Ministry of Science and Education of the Russian Federation under the 5–100 Excellence Programme and the grant of Russian Foundation for Basic Research No. 17–51–45125.

Y. Shi et al. (Eds.): ICCS 2018, LNCS 10861, pp. 171–183, 2018.
https://doi.org/10.1007/978-3-319-93701-4_13

A	B
x	y
x	z

×

C	D	E
x	u	p
y	u	q
z	v	r

=

A	B	C	D	E
x	y	x	u	p
x	y	y	u	q
x	y	z	v	r
x	z	x	u	p
x	z	y	u	q
x	z	z	v	r

B	E	D	A	C
z	q	u	x	y
y	q	u	x	y
y	r	v	x	z
z	r	v	x	z
y	p	u	x	x
z	p	u	x	x

Fig. 1. Cartesian product of tables.

columns is preserved, a careful reader can easily recognize the Cartesian structure of the table. However, this is not so easy to do for the second representation, where the rows and columns are randomly shuffled, even though the table is small. In the sequel, we will only consider the relations having no key of any kind and assume that the tuples found in the relations are all different.

Only in the first twenty–five years after Codd had developed his relational data model, more than 100 types of dependencies were described in the literature [14]. Cartesian decomposition underlies the definitions of the major dependency types encompassed by the theory of relational databases. This is because the numerous concepts of dependency are based on the *join* operation, which is inverse to Cartesian decomposition. Recall that the *join dependency* is the most common kind of dependencies considered in the framework of the fifth normal form. A relation R satisfies the join dependency $\bowtie (A_1, \ldots, A_n)$ for a family of subsets of its attributes $\{A_1, \ldots, A_n\}$ if R is the union of the projections on the subsets A_i, $1 \leqslant i \leqslant n$. Thus, if A_i are disjoint, we have the Cartesian decomposition of the relation R into the corresponding components–projections.

For the case $n = 2$ the join dependency is known in the context of the fourth normal form under the name *multivalued dependency*. A relation R for a family of subsets of its attributes $\{A_0, A_1, A_2\}$ satisfies the multivalued dependency $A_0 \mapsto A_1$ iff R satisfies the join dependency $\bowtie (A_0 \cup A_1,\ A_0 \cup A_2)$. Thus for each A_0-tuple of values, the projection of R onto $A_1 \cup A_2$ has a Cartesian decomposition. Historically, multivalued dependencies were introduced earlier than join dependencies [8] and attracted wide attention as a natural variant thereof.

An important task is the development of efficient algorithms for solving the computationally challenging problem of finding dependencies in data. A lot of research has been devoted to mining functional dependencies (see surveys [10,12]), while the detection of more general dependencies, like the multivalued ones, has been studied less. In [16], the authors propose a method based on directed enumeration of assumptions/conclusions of multivalued dependencies (exploring the properties of these dependencies to narrow the search space) with checking satisfaction of the generated dependencies on the relation of interest. In [13], the authors employ an enumeration procedure based on the refinement of assumptions/conclusions of the dependencies considered as hypotheses. Notice that when searching for functional dependencies $A \mapsto B$ on a relation R, once an assumption A is guessed, the conclusion B can be efficiently found. For multivalued dependencies, this property is not trivial and leads to the issue

of efficient recognition of Cartesian decomposition (of the projection of R on the attributes not contained in A). Thus, the algorithmic results presented in this paper can be viewed as a foundation for the development of new methods for detecting the general kind dependencies, in particular, multivalued and join dependencies.

In [7] we considered the problem of Cartesian decomposition for the relational data model. A conceptual implementation of the decomposition algorithm in Transact SQL was provided. Its time complexity is polynomial. This algorithm is based on an algorithm for the disjoint (no common variables between components) AND–decomposition of Boolean functions given in ANF, which, in fact is an algorithm of the factorization of polylinear polynomials over the finite field of the order 2 (Boolean polynomials), described by the authors in [5,6]. Notice that another algorithm invented by Bioch [1] also applied to this problem is more complex because it essentially depends on a number of different values of attributes.

The relationship between the problems of the Cartesian decomposition and factorization of Boolean polynomials can be easily established. Each tuple of the relation is a monomial of a polynomial, where the attribute values play the role of variables. Importantly, the attributes of the same type are considered different. Thus, if in a tuple different attributes of the same type have equal values, the corresponding variables are different. `NULL` is also typed and appears as a different variable. For example, for the relation above the corresponding polynomial is

$$\begin{aligned} z_B \cdot q \cdot u \cdot x_A \cdot y_C + y_B \cdot q \cdot u \cdot x_A \cdot y_C + \\ y_B \cdot r \cdot v \cdot x_A \cdot z_C + z_B \cdot r \cdot v \cdot x_A \cdot z_C + \\ y_B \cdot p \cdot u \cdot x_A \cdot x_C + z_B \cdot p \cdot u \cdot x_A \cdot x_C = \\ x_A \cdot (y_B + z_B) \cdot (q \cdot u \cdot y_C + r \cdot v \cdot z_C + p \cdot u \cdot x_C) \end{aligned}$$

Subsequently, we use this correspondence between relational tables and polynomials. This polynomial will also be referred as the table's polynomial.

Apparently, however, datasets with pure Cartesian product structure are rare. Cartesian decomposition has natural generalizations allowing us to solve more complex problems. For example, it is shown [4] that more polynomials can be decomposed if we admit that decomposition components can share variables from some prescribed set. We could use the same idea for the decomposition of datasets. Hopefully, the developed decomposition algorithm for datasets, in contrast to [4], does not depend on number of shared variables and therefore remains practical for large tables.

Fig. 2 is an adapted example from [17] extended by one table. This example comes from the decision support domain which is closely related to database management [15] and has numerous applications. From the mathematical point of view, a decision table is a map defined, sometimes partially, by explicit listing arguments and results (a set of rules or a set of implications "conditions–conclusions"). The well–known example is truth tables, which are widely used to represent Boolean functions. The decomposition of a decision table is finding the

representation of the map $F(X)$ in the form $G(X_1, H(X_2))$, $X = X_1 \cup X_2$, which may not be unique. The map H can be treated as a new, previously unknown concept. This explication leads to a new knowledge about the data of interest and its more compact presentation.

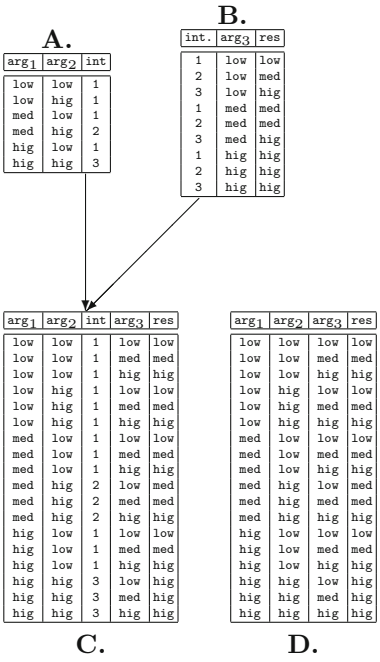

Fig. 2. Examples of decision tables.

Fig. 2 gives two examples of the interrelation between bigger and smaller decision tables. The rules of **Table C** explicitly repeat the "conclusion" for subrules. Thereby, we can detect the three dependencies

$$\mathtt{arg}_1, \mathtt{arg}_2 \mapsto \mathtt{int}, \quad \mathtt{int}, \mathtt{arg}_3 \mapsto \mathtt{res}, \quad \text{and} \quad \mathtt{arg}_1, \mathtt{arg}_2, \mathtt{arg}_3 \mapsto \mathtt{res}$$

The rules of **Table D** are more lapidary; they have no intermediate "conclusions" (the column `int`), and therefore this table has only the third dependency.

In the other words, **Table B** is a compacted version of **Table C** (and **D** as well) where compactification is based on a new concept described by **Table A**. In the map terms, informally

$$\mathbf{C}(\mathtt{arg}_1, \mathtt{arg}_2, \mathtt{int}, \mathtt{arg}_3) = \mathbf{D}(\mathtt{arg}_1, \mathtt{arg}_2, \mathtt{arg}_3) = \mathbf{B}(\mathbf{A}(\mathtt{arg}_1, \mathtt{arg}_2), \mathtt{arg}_3).$$

Table C may appear as a result of the routine design of decision tables (a set of business rules) by analysts. Yet another natural source of these tables is SQL queries. In SQL terms, the decompositions mentioned above are the reversing operators of the following kind:

```
SELECT T1.*, T2.* EXCEPT(Attr2)
  FROM T1 INNER JOIN T2
    ON T1.Attr1 = T2.Attr2
```

for **Table C** and

```
SELECT T1.* EXCEPT(Attr1), T2.* EXCEPT(Attr2)
  FROM T1 INNER JOIN T2
    ON T1.Attr1 = T2.Attr2
```

for **Table D**. Here, `EXCEPT(list)` is an informal extension of SQL used to exclude `list` from the resulting attributes. We will denote this operator as $\times_{A1=A2}$ also.

Among numerous approaches to the decomposition of decision tables via finding functional dependencies we would mention the approaches [2,11,17] having the same origins as our investigations: decomposition methods for logic circuits optimizations. These approaches perform the case exemplified by **Table D** which evidently occurs more frequently in the K&DM domain. They construct some auxiliary graphs and use the graph–coloring techniques to derive new concepts. Additional consideration are taken into account because the new concept derivation may be non–unique.

In this paper, we give a polynomial–time algorithm to solve **Table C** decomposition problem. It is based on Cartesian decomposition; therefore, we will briefly describe it. Also it explores the idea of taking into account shared variables. Namely, as it is easy to see, the values of an attribute assumed to be connector–attribute compose such a set of shared variables. They will be presented in both derived components of decomposition, appearing as conclusions and conditions, respectively.

Among possible applications of this algorithm we consider decomposition problems of Boolean tables. In particular, we demonstrate how it can be used to provide the disjunctive Shannon's decomposition of some special form and how it can be used in some generalized approach to designing decompositions for Boolean functions given in the form of truth tables with *don't care* values. In addition, some relating problems are discussed.

2 Cartesian Decomposition

First, we give a description for the AND–decomposition of Boolean polynomials which serves as a basis for the Cartesian decomposition of datasets. Then we outline its SQL–implementation for relational databases.

2.1 Algorithm for Factorization of Boolean Polynomials

Let us briefly mention the factorization algorithm given in [5,6]. It is assumed that the input polynomial F has no trivial divisors and contains at least two variables.

1. Take an arbitrary variable x from F.
2. Let $\Sigma_{same} := \{x\}, \Sigma_{other} := \varnothing$, and $F_{same} := 0, F_{other} := 0$.
3. Compute $G := F_{x=0} \cdot F'_x$.
4. For each variable $y \in Var(F) \setminus \{x\}$:
   if $G'_y = 0$ then $\Sigma_{other} := \Sigma_{other} \cup \{y\}$
   else $\Sigma_{same} := \Sigma_{same} \cup \{y\}$.
5. If $\Sigma_{other} = \varnothing$, then output $F_{same} := F, F_{other} := 1$ and stop.
6. Restrict each monomial of F onto Σ_{same} and add every obtained monomial to F_{same}; the monomial is added once to F_{same}.
7. Restrict each monomial of F onto Σ_{other} and add every obtained monomial to F_{other}; the monomial is added once to F_{other}.

Remark 1. The decomposition components F_{same} and F_{other} possess the following property. The polynomial F_{same} is not further decomposable, while the polynomial F_{other} may be decomposed. Hence, we should apply the algorithm to F_{other} to derive a finer decomposition.

The worst–time complexity of the algorithm is $O(L^3)$, where L is the length of the polynomial F, i.e., for the polynomial over n variables having M monomials of lengths $m_1, \ldots, m_M$, $L = \sum_{i=1}^{M} m_i = O(nM)$. In [5] we also show that the algorithm can be implemented without computing the product $F_{x=0} \cdot F'_x$ explicitly.

2.2 SQL–Implementation of Decomposition Algorithm

A decomposition algorithm for relational tables implements the steps of the factorization algorithm described above. An implementation of this algorithm in Transact SQL is given in [3].

In terms of polynomials, it is easy to formulate and prove the following property: if two variables always appear in different monomials (i.e., there is no monomial in which they appear simultaneously) then these variables appear in different monomials of the same decomposition component if a decomposition exists. A direct consequence of this observation is that for each relation attribute it is enough to consider just one value of this attribute because the others must belong to the same decomposition component (if it exists).

Trivial Attribute Elimination. If some attribute of a relation has only one value, we have a case of trivial decomposition. In terms of polynomials, this condition can be written as $F = x \cdot F'_x$. This attribute can be extracted into a separate table. In what follows, we assume that there are no such trivial attributes.

Preliminary Manipulations. This creates auxiliary strings which are needed to form SQL queries. At the first step, we need to select a "variable" x, with respect to which decomposition will be constructed. We need to find two sets of attributes forming the tables as decomposition components. As mentioned above,

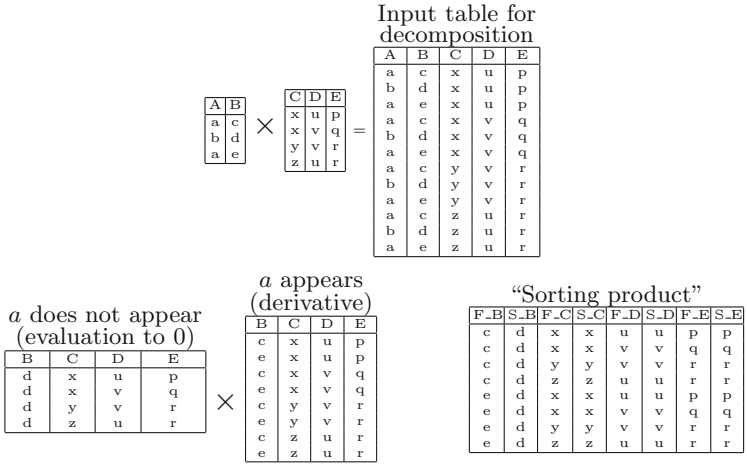

Fig. 3. Example of Cartesian Decomposition

we can take an arbitrary value of an arbitrary attribute of the table. Next, we create the string representing table attributes and their aliases corresponding to the product $F_{x=0} \cdot F'_x$ (in terms of polynomials). The prefixes `F_` and `S_` correspond to $F_{x=0}$ and F'_x.

Creation of Duplicates Filter. After that, we create a string of a logical expression allowing us to reduce the size of the table–product through the exclusion of duplicate rows; they appear exactly twice. In terms of polynomials, these are the monomials of the polynomial-product with the coefficient 2, which can be obviously omitted in the field of the order 2. In an experimental evaluation we observed that the share of such duplicates reached 80%. Since this table is used for bulk queries, its size significantly impacts the performance.

Retrieval of "Sorting Product". The table-product allowing for sorting attributes with respect to the component selected is created in the form `VIEW`. It is worth noting that it can be constructed in different ways. A "materialized" `VIEW` can significantly accelerate the next massively executed query to this table–product. It is easy to see that the table corresponding to the full product is bigger than the original table. In the example given above it would contain 32 rows. However, its size can be reduced substantially by applying the duplicates filter. The view `SortingProduct` contains only 8 rows.

Partition of Attributes. The membership of a variable y in a component containing the variable x selected at the first step is decided by checking whether the partial derivative of the polynomial $\frac{\partial}{\partial y}(F_{x=0} \cdot F'_x)$ is not equal to zero (in the finite field of order 2). They are from different components iff this derivative

vanishes. This corresponds to checking whether a variable appears in the monomials in the second degree (or is absent at all). In SQL terms, an attribute `A` belongs to another component (with respect to the attribute of x) if each row of the sorting table contains equal values at `F_A` and `S_A` columns.

Retrieval of Decomposition. At the previous steps, we find a partition of attributes and constructs strings representing it. If the cycle is completed and the string for the second component is empty, then the table is not decomposable. Otherwise, the resulting tables–components are produced by restricting the source table onto the corresponding component attributes and selecting unique tuples.

To verify the new concepts discovery algorithms, Jupan and Bohanec described an artificial dataset establishing characteristics of cars (see, for example, [17]). As it is pure Cartesian product of several attribute domains representing characteristics, the decomposition algorithm given above produces a set of linear factors.

At the same time, disjointly decomposable Boolean polynomials are rare:

Proposition 1. *If a random polynomial F has M monomials defined over $n > 2$ variables without trivial divisors, then*

$$\mathbb{P}[F \text{ is } \varnothing\text{--}\textit{undecomposable}] > 1 - \left(1 - \frac{\phi(M)}{M}\right)^n > 1 - \left(1 - \frac{1}{e^\gamma \ln\ln M + \frac{3}{\ln\ln M}}\right)^n,$$

where ϕ and γ are Euler's totient function and constant, respectively.

Remark 2. For database tables M is the relation's cardinality (number of the table's rows) and n is the number of different values in the table which can be estimated as $O(dM)$ where d is the relation's degree (number of the table's attributes). Notice that polynomials corresponding to database tables have a particular structure and, therefore, the bound can be improved.

3 One Generalization of Cartesian Decomposition

As "pure" Cartesian decomposition is rare, it is naturally to detect other tractable cases and to develop new kinds of decompositions for them. One way is to abandon the strict requirement on decomposition components to be disjoint on values. It is shown [4] that more Boolean polynomials can be decomposed if we admit that decomposition components can share variables from some prescribed set. We would use the same idea for decomposition of datasets. Arbitrariness of choice of variables results in an exponential growth of the algorithm complexity with respect to the number of variables. Hopefully, table–based datasets have a particular structure that can be taken into account. Namely, we can take as shared variables only those which corresponds to the same attribute. This attribute connects original datasets (items of them) on base of the equality of theirs values. In this case, the decomposition algorithm does not depend on the number of shared variables in contrast to the Boolean polynomials case and therefore appears practical for large tables.

3.1 Decomposition with Explicit Attribute–Connector

For the decomposition of tables with an explicit connector–attribute, the Cartesian decomposition is a crucial step. In general, this decomposition consists of the following steps:

A	B	C	D	E
a	p	u	x	1
b	q	u	x	1
a	p	u	y	2
a	q	v	y	2
b	p	u	x	3
b	p	v	y	3

$$P = [\{\{A, B\}, \{C\}, \{D\}\}, \{\{A\}, \{B, C\}, \{D\}\}, \{\{A\}, \{B\}, \{C, D\}\}]$$

Fig. 4. An undecomposable table with decomposable sub–tables for the connector–attribute E.

1. Subdivide the original table into k sub–tables such that all sub–table rows contain the same value at the connector–attribute (this attribute should be excluded for further manipulations).
2. For each sub–table perform the full Cartesian decomposition (i.e. all components are undecomposable), skipping the last step (projection on partition of attributes). Notice that all trivial components appear in the partition of attributes as singleton sets. Then we have a set of partitions $P = [p_1, \ldots, p_k]$ of table attributes $\mathcal{A}$, where one partition corresponds to the Cartesian decomposition of one sub–table.
3. We cannot use a simple projection on partition of attributes because it is possible that all sub–tables are decomposable while the entire table is not (an example at Fig. 4). The table of interest is decomposable if there exists a minimal closure of the parts of attribute partitions across all sub–tables (if parts of different partitions have a common attribute, then both parts are joined with the resulting closure) such that this closure does not coincide with the entire set of the table attributes. This simple procedure can be done in $O(|P| \cdot |\mathcal{A}|^2)$ steps.

1. Select any attribute set π of any partition from P.
2. Initialize the result set R by π.
//when the algorithm stops then R contains component attributes
3. Initialize the active set A by π.
//it contains attributes that will be treated at the next closure steps
4. While $A \neq \varnothing$ do:
5. Take any attribute a from A; remove it from A.
6. For each $p \in P$ do:
7. Select from p the attribute set π containing a.
8. $A := A \cup (\pi \setminus R)$.

9. $R := R \cup \pi$.
10. If $R = \mathcal{A}$ then the table is not decomposable; otherwise, it is.
11. If decomposable then R and $\mathcal{A} \backslash R$ are the attribute sets of the components of decomposition.
12. For each sub–table perform projections on these attribute sets.

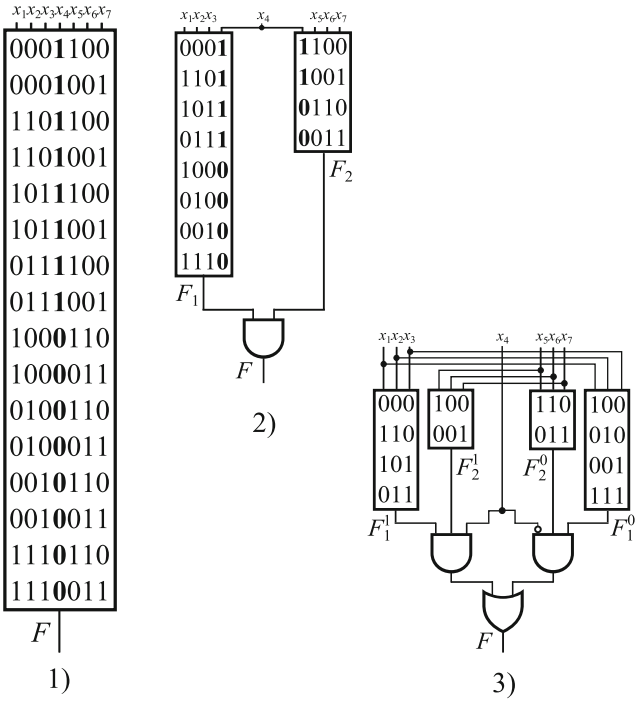

Fig. 5. Circuit decomposition example.

3.2 Applications to Boolean Tables

The interplay of K&DM and logic circuit optimization is quite important and fruitful. An interesting application of this decomposition algorithm is logic circuit optimization. Indeed, every Boolean table (with different rows) is the true/false part of the truth table of some Boolean function (the set of satisfying/unsatisfying vectors). This algorithm allows us to find tables corresponding to Boolean functions of the following Shannon's OR–decomposition, where $F_{x=0}$ and $F_{x=1}$ components have finer disjoint Cartesian decomposition

$$F(U, V, x) = \overline{x}F(U, V, 0) \vee xF(U, V, 1) = \overline{x}F_1^0(U)F_2^0(V) \vee xF_1^1(U)F_2^1(V).$$

A number of function that are decomposable in this way can be easily counted. For simplicity's sake they are $n^2 2^{n-2} - O(n2^n)$.

An example is shown at Fig. 5. The original circuit **(1)** is given in the form of the satisfying vectors table (on missing inputs the output is false). The connector–attribute corresponding to the input x_4 is given in **bold**. The composition **(2)** is the simplest result of decomposition as

$$F(x_1, \ldots, x_7) = F_1(x_1, x_2, x_3, x_4) \wedge F_2(x_4, x_5, x_6, x_7).$$

But evidently, the connector–attribute can be replaced by a simpler controlling wire. F_k^v is a part of the function $F_k, k = 0, 1$, with the value $v = 0, 1$, at x_4. The result is the composition **(3)**. Notice that the derived Boolean functions given by the tables have a specific structure and can be specifically optimized.

Table 1. (a) Decomposition example. (b) Function–combinator.

x_1	x_2	x_3	x_4	F
1	0	0	0	0
0	0	1	0	1
0	0	0	1	1
0	1	1	0	1
0	1	0	1	1
1	1	1	0	1
1	1	0	1	1
1	0	1	1	0

$=$

x_1	x_2	F_1
0	0	1
1	0	0
0	1	1
1	1	1

$\times_{F_1=F_2}$

x_3	x_4	F_2
0	0	0
1	0	1
0	1	1
1	1	0

a) Decomposition example.

x	y	H
0	0	0
1	1	1
DC		DC

b) Function–combinator.

Yet another application of this decomposition emerges when we consider the decomposition of a truth table with *don't care* (DC) inputs and outputs with respect to the resulting column. The following example at Table 1 plainly explains this idea. The decomposition components defines the *not–DC* part of the truth table.

The complete form of the original Boolean function can be defined by the function–combinator H

$$F(x_1, x_2, x_3, x_4) = H(F(x_1, x_2), F(x_3, x_4)).$$

Note that by extending definition on DCs we can deduce different kinds of decompositions (eliminating DC). For example, if we extend H to the definition of the disjunction (OR) then we establish the disjoint OR–decomposition of Boolean functions given in the form of truth tables with DC.

4 Further Work

To achieve deeper optimization we asked [5,6] how to find a representation of a Boolean function in the ANF–form $F(X, Y) = G(X)H(Y) + D(X, Y)$, i.e.

the relatively small "defect" $D(X,Y)$ extends or shrinks the pure "Cartesian product".

In the scope of decomposition of Boolean functions given in the form of truth tables with DC finding small extensions (redefinition of several DCs) may cause more compact representations.

Clearly, finding representation of the table's polynomial in the form

$$F(X,Y) = \sum_k G_k(X)H_k(Y), \qquad X \cap Y = \varnothing,$$

i.e. complete decomposition without any "defect", solves **Table D** decomposition problem. Here, valuation of k corresponds to a new concept (an implicit connector–attribute), which will serve as a result of the compacting table and an argument of the compacted table. Although, apparently, such decompositions (for example, this one, is trivial, where each monomial is treated separately) always exist, not all of them are meaningful from the K&DM point of view. Formulating additional constraints targeting decomposition algorithms is an interesting problem.

Finding a "defect" $D(X,Y)$ can be considered as completing the original "dataset" $F(X,Y)$ to derive some "conceptual" decompositions. In other words, $D(X,Y)$ represents incompleteness or noise/artifacts of the original dataset if we need to add or to remove data, respectively. It is relative because divers completions are possible. It can be Cartesian or involve explicit/implicit connectors. For example, there always exists a trivial completion ensuring Cartesian decomposition into linear factors

$$F(X) + D(X) = \prod_{i=1}^{n} \sum_{x_i^j \in A_i} x_i^j$$

where x_i^j are variables representing different values of a A_i domain (the i^{th}–column of the table) as for the mentioned above CARS–example of Bohanec and Jupan.

A simple observation is inspired by considering non–linear factors that can appear under some completions. For example, if A and B domains belong to the same non–decomposable factor then all the factor's monomials $a_i b_j$ form values of a new concept that is a subconcept of $A \times B$. It can serve for the reduction of dataset dimension (degree of a relation) and space requirements to represent domain values.

References

1. Bioch, J.C.: The complexity of modular decomposition of boolean functions. Discrete Appl. Math. **149**(1–3), 1–13 (2005)
2. Bohanec, M., Zupan, B.: A function-decomposition method for development of hierarchical multi-attribute decision models. Decis. Support Syst. **36**(3), 215–233 (2004)

3. Emelyanov, P.: Cartesian decomposition of tables. Transact SQL. http://algo.nsu.ru/CartesianDecomposition.sql
4. Emelyanov, P.: AND–decomposition of boolean polynomials with prescribed shared variables. In: Govindarajan, S., Maheshwari, A. (eds.) CALDAM 2016. LNCS, vol. 9602, pp. 164–175. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29221-2_14
5. Emelyanov, P., Ponomaryov, D.: Algorithmic issues of conjunctive decomposition of boolean formulas. Program. Comput. Softw. **41**(3), 162–169 (2015)
6. Emelyanov, P., Ponomaryov, D.: On tractability of disjoint AND-decomposition of boolean formulas. In: Voronkov, A., Virbitskaite, I. (eds.) PSI 2014. LNCS, vol. 8974, pp. 92–101. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46823-4_8
7. Emelyanov, P., Ponomaryov, D.: Cartesian decomposition in data analysis. In: Proceedings of the Siberian Symposium on Data Science and Engineering (SSDSE 2017), pp. 55–60 (2017)
8. Fagin, R., Vardi, M.: The theory of data dependencies: a survey. In: Mathematics of Information Processing: Proceedings of Symposia in Applied Mathematics, vol. 34, pp. 19–71. AMS, Providence (1986)
9. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery in databases. AI Mag. **17**(3), 37–54 (1996)
10. Liu, J., Li, J., Liu, C., Chen, Y.: Discover dependencies from data - a review. IEEE Trans. Knowl. Data Eng. **24**(2), 251–264 (2012)
11. Mankowski, M., Łuba, T., Jankowski, C.: Evaluation of decision table decomposition using dynamic programming classifiers. In: Suraj, Z., Czaja, L. (eds.) Proceedings of the 24^{th} International Workshop on Concurrency, Specification and Programming (CS&P 2015), pp. 34–43 (2015)
12. Papenbrock, T., Ehrlich, J., Marten, J., Neubert, T., Rudolph, J., Schoenberg, M., Zwiener, J., Naumann, F.: Functional dependency discovery: an experimental evaluation of seven algorithms. Proc. VLDB Endowment **8**(10), 1082–1093 (2015)
13. Savnik, I., Flach, P.: Discovery of multivalued dependencies from relations. Intell. Data Anal. **4**(3–4), 195–211 (2000)
14. Thalheim, B.: An overview on semantical constraints for database models. In: Proceedings of the 6th International Conference on Intellectual Systems and Computer Science, pp. 81–102 (1996)
15. Vanthienen, J.: Rules as data: decision tables and relational databases. Bus. Rules J. **11**(1) (2010). http://www.brcommunity.com/a2010/b516.html
16. Yan, M., Fu, A.W.: Algorithm for discovering multivalued dependencies. In: Proceedings of the 10^{th} International Conference on Information and Knowledge Management (CIKM 2001), pp. 556–558. ACM, New York (2001)
17. Zupan, B., Bohanec, M.: Experimental evaluation of three partition selection criteria for decision table decomposition. Informatica **22**, 207–217 (1998)