8th International Conference on Mass Customization and Personalization – Community of Europe (MCP-CE 2018)
Digital Customer Experience
September 19-21, 2018, Novi Sad, Serbia

# EMPOWERING THE USE OF VARIANT TABLES IN MASS CUSTOMIZATION

**Albert Haag, Laura Haag**
Product Management Haag GmbH, Germany

**Abstract:** *Due to regularities in the product data in mass customization (MC) products, variant tables can be substantially compressed. In this paper, we show that a table can be compressed in a way that scales with increasing number of feature combinations while retaining the core functionality of the original table that is useful to an MC business. Moreover, the suggested compressed form allows very fast table queries and may offer insights into the complexity of the underlying MC business as well. This compression technology could be integrated with existing MC implementations with little risk where the filtering queries important in configuration can be simply re-directed to the compressed form. We illustrate the concepts using personalized T-shirts as an example.*

**Key Words:** ***Variant Table, Mass Customization, Tabular Constraint, Table Compression, Product Configuration, Product Modeling***

## 1. INTRODUCTION

At heart, mass customization (MC) is mass production paired with individualization. We take this to be related to the production of product variants [1] with the emphasis on individualization, i.e. there will be many variants of a product and the business is prepared to produce only a single unit of each one on demand (lot size one). Accordingly, all variants of an MC product share a common finite set of properties, and their individualized features are defined by a value assignment to these properties. Valid combinations of product features for an MC product can be represented as rows of a variant table, where a value in a table cell defines an individual product feature while the associated table column refers to the respective product property. When the number of offered variants is finite and not too large, the variants can be modeled in one overall variant table. Alternatively, a product model could be constructed using rules and constraints[2].

Tables have the advantage that they are easy to understand and straightforward to communicate and exchange. Well-developed data management systems exist for processing them. They are the backbone of business data, both in sales and fulfillment. Variant tables are thus a method of choice when modeling variants in business. The downside of variant tables is that they may not scale: An increase in overall product features or properties can lead to an exponential increase in table size. Adding one property for a binary (yes/no) choice doubles the number of personalized combinations, adding two binary properties, quadruples it. Similarly, if the number of available features is doubled for a non-binary product property, a variant table referencing that property may double in size as well. This is analogous to the "Wheat and Chessboard" problem (see [1]) of placing a grain of wheat on a first square of a chessboard, and then doubling the number of grains placed on each subsequent square.

Currently, when a variant table becomes unmanageably large, it is broken up into smaller subtables or other means of product modeling are sought. However, these measures add complexity and cost to the business. We offer a simpler and perhaps more efficient alternative in this paper: enabling the representation and use of large, even huge, variant tables through the use of table compression.

Here, we show that very large variant tables can be extremely compressible due to regularities in the product variants (see Section 6). This is instrumental in several cases:

1. In situations where large variant tables are maintained in extensional form (all valid variants are listed in individual rows), but handling them is slow (e.g. a spreadsheet with several hundreds of thousands of rows), interaction with them can be sped up drastically by compressing them solely for purposes of better performance.
2. In situations where classical relational variant tables exceed plausible bounds on size, storing and operating on the compressed size offers an alternative by reducing the space and time required.
3. When a business wants to exploit the possibility of maintaining all variants of one product in a single large table.
4. Compression may identify basic structures inherent in a variant offering. The degree of

[1] This is the source of the name of the SAP Variant Configurator, which is covered in [8]. A brief sketch of its motivation and history is given in [11]. Note that the product properties are called characteristics, and the product features are called characteristic values in SAP terminology.
[2] Tabular constraints are themselves variant tables expressing sub-relations on the variants.

compressibility offers a key complexity measure for assessing the cost that variants bring to the business processes (see Section 8).

The main compression method used in this paper is to c-tuples (see Section 4). This basic form of compression is already very powerful and may be sufficient in many cases. Advanced compression using Variant Decomposition Diagrams (VDDs, see Section 7) may offer additional benefits if needed. We show in Section 7.2 that the tabular paradigm important to the handling of data in a business is preserved under this compression "for all practical purposes", i.e. a system managing compressed variant tables, which we propose to call a variant base, can provide the services of a database relevant to configuration.

Towards the goal of empowering the handling of huge variant tables, we have both suggestions for concrete measures that might be implemented now, and also a vision of what should become possible in the future. We also identify open issues, current development, and put up some conjectures for discussion. We address the issue of storing, processing, and exchanging tables in compressed form. We strive to minimize the cost and risk of deploying such methods by adhering to established standards as much as possible.

## 2. METHODS

### 2.1. C-tuples

The term c-tuple is taken from [2]. A c-tuple is a tuple of sets of features $\langle F_1, F_2, \cdots, F_k \rangle$. It represents the set of all combinations that can be formed by choosing one value from each set, i.e. the Cartesian product $F_1 \times F_2 \times \cdots \times F_k$. The rows of a variant table can be partitioned into a disjoint set of c-tuples (see [2]). C-tuples are described in more detail in Section 4 and [2].

### 2.2. Variant Decomposition Diagrams (VDDs)

A VDD is a type of decision diagram (DD), which was specifically designed with table compression in mind and is described in detail in [3]. VDDs offer better compression than c-tuples by making better use of redundancies in the c-tuples. See Section 7.1 for a brief introduction.

### 2.3. Size measures for a c-tuple/table

One can quantify the size of a c-tuple by two ways. First, one can determine the number of combinations a c-tuple represents. The number of combinations for each c-tuple is calculated by multiplying the number of values in each cell by one another. For a set of c-tuples that are a disjoint partitioning of a table, the sum of combinations across all c- tuples gives the total number of rows in table. In Table 10, we denote this by #rows. In a compressed table, the number of rows equals the number of c-tuples required, denoted by #c-tuples in Table 10. For example, Table 2 has 1 c-tuple which represents 276,480,000 combinations:

#rows = 4 * 3 * 3 * 8 * 8 * 5 * 1000 * 8 * 3

The second measure is the number of symbols in a c-tuple, where each symbol represents one value in each of its sets. The sum of symbols across all c-tuples gives the total number of symbols required for the compressed table and is denoted by #symbols in Table 10. Table 2 requires 1042 symbols.

*#symbols* = 4 + 3 + 3 + 8 + 8 + 5 + 1000 + 8 + 3

This measure gives the size needed to store a c-tuple and is comparable to the total number of cells (#cells) in an extensional table, which is calculated by multiplying the total number of rows by the total number of columns.

### 2.4. Complexity measures

Product complexity incurs cost both for the configuration task and the business processes. For the configuration task, there is a cost of modeling and a consideration of configurator performance, i.e. the time required for the configurator to respond to an external query. We propose a method of assessing complexity directly from the list of offered variants[3]. The method is based on counting the number of symbols needed in the compressed form. We argue that this relates directly to both the response time of configuration queries and is also relevant to assessing business complexity. Product and business complexity measures are described in more detail in Section 8.

## 3. EXAMPLE: PERSONALIZED T-SHIRTS

Let us now introduce the exemplary MC scenario which we will use throughout the paper to explain the various concepts.

An early MC business[4] begins by offering to personalize a standard T-shirt (Figure 1**)** by dyeing it in a particular color. They offer three colors (*Black*, *Blue*, *Red*) as well as the standard non-dyed *White* T-shirts in the three sizes *S* (*Small*), *M* (*Medium*), and *L* (*Large*). For this, they set up a shop that stores white T-shirts, takes them from stock as orders come in, dyes them as warranted and hands them over to their dealer who acquires the orders and handles invoicing and shipping. Two silk-screen printing shops are subcontracted to provide an imprint on the T-shirt. One is set up for "MEN IN BLACK" (MIB) a heavy white print performed only on black T-shirts of all sizes. The other for "Save the whales" (STW) a larger light-blue imprint performed on T-shirts of any color in sizes *M* and *L*.


Fig. 1. *Simple personalized T-shirts*

[3] This method also applies to variant tables expressing sub-relations.

[4] Any resemblance to an actual business is purely coincidental.

There are three properties and a total of nine features. The eleven combinations can be easily represented in a classic variant table in extended form, i.e. all variants are listed individually (Table 1).

The business expands over time and technology as well as customer expectations advance. A new T-shirt is designed. Five additional properties for personalization are added: Style, Neck, Fabric, ImpCol (color of imprint), and ImpSiz (size of imprint). More T-shirt colors are added. The acquisition of a new offset printer allows a drastic increase in number of offered imprints to 1000 that are presented in a catalog. The two "vintage" silk-screen prints are dropped. Lastly, customers have the choice of one of three amounts charged to offset the environmental impact of producing the T-shirt: \$0.00, \$0.99, and \$1.99, which is added to the sales price of their T-shirt. A certificate for this amount is stamped on the T-shirt.

Table 1. *Classic variant table for initial very simple T-shirts*

| *Imprint* | *Size* | *Color* |
|---|---|---|
| *MIB* | *S* | *Black* |
| *MIB* | *M* | *Black* |
| *MIB* | *L* | *Black* |
| *STW* | *M* | *Black* |
| *STW* | *M* | *White* |
| *STW* | *M* | *Red* |
| *STW* | *M* | *Blue* |
| *STW* | *L* | *Black* |
| *STW* | *L* | *White* |
| *STW* | *L* | *Red* |
| *STW* | *L* | *Blue* |

The new property domains, the sets of allowable values, are as follows:

- *Style* with values *NoSleeve*, *HalfSleeve*, *FullSleeve*, *Hoodie*;
- *Neck* with values *Round*, *VNeck*, *Collar*.
- *Fabric* with values *Cotton*, *Synthetic*, *Mixed*;
- *Size* with values *3T*, *4T*, *XS*, *S*, *M*, *L*, *XL*, *XXL*;
- *Color* with values *Black*, *Blue*, *Red*, *White*, *Green*, *Purple*, *Pink*, *Yellow*;
- *Imprint* with 1000 values collected in a catalog
- *ImpCol* with values *Green*, *Blue*, *Black*, *Red*, *White*;
- *ImpSiz* with values *Baby*, *Tiny*, *Cute*, *Small*, *Medium*, *Big*, *ExtraBig*, *Fill*;
- *$CO_2$-Offset* with values \$0.00, \$0.99, \$1.99

The product is now more complex from a business point of view, which is further discussed in Section 8. In Section 5 we will continue this example by adding constraints which disallow some feature combinations.

The 43 features of the new T-shirt can be combined to millions of different variants, more than are readily representable in an extensional form in a classic table. However, they can easily be represented in the compressed forms introduced in Sections 4 and 7.1.

## 4. C-TUPLES – BASIC BUT POWERFUL COMPRESSION

A *c-tuple* is a tuple of sets (one for each product property) which contain one or more values (=product features). A c-tuple represents the Cartesian product of its sets, i.e. the set of combinations of features that can be formed by choosing one value from each set. Used in a constraint, each c-tuple denotes an unconstrained subset, i.e. all its combinations are defined to be valid.

Consider the very simple T-shirt from before: A personalized T-shirt is sold with a particular size, color, and imprint. If the two imprints, three sizes, and four colors can be arbitrarily combined, this leads to an offering of 24 T-shirt variants, i.e. the respective classic variant table would have 24 rows and three columns for a total of 72 cells. However, the same content can be expressed as a single row in a spreadsheet by placing multiple values in each cell, thereby summarizing the valid combinations. This needs only 3 cells with a total of 9 symbols (2 imprints + 3 sizes + 4 colors). Each cell lists all values for the respective product properties. This is the *c-tuple* of the *property domains*, containing all features allowable for each property.

A classic extensional variant table for the above example with 24 rows is feasible, however, the situation looks very different for the new expanded T-shirt with nine product properties. Here, it is not feasible to list all possible combinations of features in a table due to its size (> 275 million possible combinations). Instead, they can be defined in a compact way as the c-tuple of the property domains. Table 2 shows this c-tuple substituting the shorthand *wildcard* symbol "*" to stand for an entire property domain.

Table 2. *Property domains of new T-shirt as c-tuple*

| *Style* | *Neck* | *Fabric* | *Size* | *Color* | *ImpCol* | *Imprint* | *ImpSiz* | *CO2* |
|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | * | * | * | * |

A c-tuple represents a set of combinations that are not internally constrained in any way, i.e. any value from any of its sets of values can be freely combined with any value from any other set. For example, the c-tuple in Table 2 represents complete freedom of choice. Partitioning a variant table into c-tuples (*unconstrained* subsets) is a basic form of compression. It can be transparently represented in spreadsheets and is also supported by some MC tools[5]. We show by example in Section 5 the power and significance of compression to c-tuples. In Table 2, over a quarter of a billion variants are represented by a single c-tuple which requires only 1042 symbols.

## 5. MODELING THE T-SHIRT AS C-TUPLES

We now continue to evolve the T-shirt product by adding three constraints in an exemplary fashion. The examples are supposed to:

[5] Variant tables in the SAP Variant Configurator (SAP VC) support a c-tuple format, there referred to as "variant tables with multiple values in a cell" from the beginning of the 90's [12]. See [3] for a practical example.

- show that c-tuples can be used to efficiently represent individual constraints/tables
- convey a feeling for the potential and need of compression to c-tuples
- show the smooth interaction a tabular paradigm provides to the business processes

### 5.1. Constraint 1: Adding the vintage imprints

Customer demand may cause the reactivation of the two vintage silk-screen imprints. This increases the domain of imprints to 1002 imprints. The vintage imprint "MEN IN BLACK" (MIB) is still possible only on black shirts. This adds one c-tuple to the set of variants in Table 2. The other vintage imprint "Save the whales" (STW) is not possible on any shirts smaller than *M*. This also adds one c-tuple to the set of variants in Table 2. There are no other constraints. We now have the three c-tuples in Table 3. For space reasons, we omit the properties *Style*, *Neck*, *Fabric*, *ImpSiz*, and $CO_2$, which are still completely freely choosable.

We use the alias ⟨*adult*⟩ for the sizes {*M, L, XL, XXL*}[6]. The imprints *MIB*/*STW* are always executed in their special imprint sizes. We assume these imprints will service any specified imprint size.

For space reasons, we will omit any column that consists solely of wildcards in all of the following variant tables. For any table and any property not mentioned in the table one can always assume the wildcard "*" for the missing column. But they are considered part of the table when it comes to sizing.

Table 3. *New T-shirt with vintage imprints*

| *Size* | *Color* | *ImpCol* | *Imprint* |
|---|---|---|---|
| * | * | * | ¬<*vintage*> |
| * | *{Black}* | *{White}* | *{MIB}* |
| <*adult*> | * | *{Blue}* | *{STW}* |

Sizing of the variant table in Table 3 is now as follows. The c-tuple of five common properties not shown in the table has a combined total of 21 features, representing 864 combinations. The size of the three c-tuples including the four properties in Table 3 listed in the table is as follows:

- The first c-tuple has 1042 features, representing 276,480,000 combinations.
- The second c-tuple has 32 features, representing 6,912 combinations.
- The third c-tuple has 35 features, representing 27,648 combinations.

Combining these yields overall sizing of 1109 symbols for features needed in the three c-tuples. With merely three c-tuples we can represent 276,514,560 combinations which is the sum of combinations of the individual c-tuples above.

Although it is straightforward here to directly formulate the three c-tuples in Table 3, the two constraints can be formulated individually, as in Table 4 and Table 5.

[6] No discrimination towards adults preferring a smaller size is intended.

Table 4. *MIB constraint as c-tuples*

| Color | Imprint |
|---|---|
| * | ¬{MIB} |
| {Black} | {MIB} |

Table 5. *STW constraint as c-tuples*

| Size | Imprint |
|---|---|
| * | ¬{STW} |
| {M, L, XL, XXL} | {STW} |

Both tables are in a format a product manager might use informally. It is the task of proper tools to meld these two small tabular constraints into the overall set of variants of Table 3. Such tools will allow composing variant tables by combining blocks of variants (c-tuples of subsets of properties). This is a topic of future development.

### 5.2. Constraint 2: Adding a constraint on color and imprint color

To avoid customer dissatisfaction, the sales department decides to enforce a constraint that the imprint color be distinguishable from the T-shirt color7. This takes the form of Table 6[8].

Table 6. *Color / Imprint color relation (sales)*

| *Color* | *Imprint* | *ImpCol* |
|---|---|---|
| ¬*{Green}* | ¬*{STW}* | *{Green}* |
| ¬*{Blue}* | ¬*{STW}* | *{Blue}* |
| ¬*{Black}* | ¬*{STW}* | *{Black}* |
| ¬*{Red}* | ¬*{STW}* | *{Red}* |
| ¬*{White}* | ¬*{STW}* | *{White}* |
| * | *{STW}* | *{Blue}* |

These six c-tuples must be intersected with each of the previous c-tuples in Table 3. The result is shown in Table 7.

Adding this constraint affects the count of the valid variants offered. In contrast, T-shirt production is not affected, because the constraint is not technical. Thus Table 7 is relevant for sales and front-end validation. It can be interpreted as a product catalog of all offered T-shirt variants and could be published. Table 3 is still valid for production.

The addition of the color constraint led to an increase of ctuples and a reduction of valid variants. Table 7 now has seven ctuples representing about 30 million variants less than Table 3.

[7] They will allow the vintage light-blue silk-screen imprint *STW* on blue shirts, when desired.

[8] Note that using a formula involving symbolic values, such as $Color \neq ImpCol$, possible in some configuration tools, compares "apples" with "oranges". This is often not semantically correct, as the color of a T-shirt dye does not directly compare with the color of the imprint ink, e.g. the *blue* T-Shirt color is not the same as the *light-blue* imprint color although both are denoted by the symbol *blue*. Variant tables are a more correct and concise way of stating symbolic relations.

Table 7. *New T-shirt with sales constraint on color*

| *Size* | *Color* | *ImpCol* | *Imprint* |
|---|---|---|---|
| * | ¬*{Black}* | *{Black}* | ¬*<vintage>* |
| * | ¬*{Blue}* | *{Blue}* | ¬*<vintage>* |
| * | ¬*{Red}* | *{Red}* | ¬*<vintage>* |
| * | ¬*{White}* | *{White}* | ¬*<vintage>* |
| * | ¬*{Green}* | *{Green}* | ¬*<vintage>* |
| * | *{Black}* | *{White}* | *{MIB}* |
| *<adult>* | * | *{Blue}* | *{STW}* |

### 5.3. Constraint 3: Adding a constraint for dye used with fabric

The constraint relation between the color of the T-shirt and the color of the imprint given in Table 7 affected sales, but not manufacturing. Vice versa, there may be constraints that pertain only to manufacturing, but not sales. As an example, consider the following: The dye used with cotton fabric is different from that used for synthetic or mixed materials. For this purpose, a tenth product property Dye is added for dyeing a shirt green, purple, pink, or yellow. The domain of the property Dye is the set of dyes: none, GRCD#1, GRSD#2, PUCD#3, PUSD#4, PICD#5, PISD#6, YCD#7, YSD#8. Also, we may assume that the T-shirts are now stocked (or procured) in the standard colors black, blue, red, and white, thus requiring no further dyeing. Table 8 gives the constraining relation, Table 9 gives the ensuing overall list of c-tuples. For sales, Table 7 still represents the catalog of offered T-shirts.

We abbreviate the standard colors {Black,Blue,Red,White} with the alias <std>.

The constraint can be formulated with nine c-tuples in Table 8 using 41 value symbols and represents 44 rows. The overall T-shirt model (Table 9) can be expressed with 19 c-tuples using 9224 value symbols that represent > 250 million producible variants.

Table 8. *Fabric / Color / Dye relation*

| *Fabric* | *Color* | *Dye* |
|---|---|---|
| * | *<std>* | *none* |
| *{Cotton}* | *{Green}* | *GRCD#1* |
| *{Cotton}* | *{Purple}* | *PUCD#3* |
| *{Cotton}* | *{Pink}* | *PICD#5* |
| *{Cotton}* | *{Yellow}* | *YCD#7* |
| ¬*{Cotton}* | *{Green}* | *GRSD#2* |
| ¬*{Cotton}* | *{Purple}* | *PUSD#4* |
| ¬*{Cotton}* | *{Pink}* | *PISD#6* |
| ¬*{Cotton}* | *{Yellow}* | *YSD#8* |

Table 9. *New T-shirt with manufacturing constraint on dyes*

| *Fabric* | *Size* | *Color* | *ImpCol* | *Imprint* | *Dye* |
|---|---|---|---|---|---|
| * | * | *<std>* | * | ¬*<vintage>* | *none* |
| *{Cotton}* | * | *{Green}* | * | ¬*<vintage>* | *GRCD#1* |
| *{Cotton}* | * | *{Purple}* | * | ¬*<vintage>* | *PUCD#3* |
| *{Cotton}* | * | *{Pink}* | * | ¬*<vintage>* | *PICD#5* |
| *{Cotton}* | * | *{Yellow}* | * | ¬*<vintage>* | *YCD#7* |
| ¬*{Cotton}* | * | *{Green}* | * | ¬*<vintage>* | *GRSD#2* |
| ¬*{Cotton}* | * | *{Purple}* | * | ¬*<vintage>* | *PUSD#4* |
| ¬*{Cotton}* | * | *{Pink}* | * | ¬*<vintage>* | *PISD#6* |
| ¬*{Cotton}* | * | *{Yellow}* | * | ¬*<vintage>* | *YSD#8* |
| * | * | *{Black}* | *{White}* | *{MIB}* | *none* |
| * | *<adult>* | *<std>* | *{Blue}* | *{STW}* | *none* |
| *{Cotton}* | *<adult>* | *{Green}* | *{Blue}* | *{STW}* | *GRCD#1* |
| *{Cotton}* | *<adult>* | *{Purple}* | *{Blue}* | *{STW}* | *PUCD#3* |
| *{Cotton}* | *<adult>* | *{Pink}* | *{Blue}* | *{STW}* | *PICD#5* |
| *{Cotton}* | *<adult>* | *{Yellow}* | *{Blue}* | *{STW}* | *YCD#7* |
| ¬*{Cotton}* | *<adult>* | *{Purple}* | *{Blue}* | *{STW}* | *PUSD#4* |
| ¬*{Cotton}* | *<adult>* | *{Green}* | *{Blue}* | *{STW}* | *GRSD#2* |
| ¬*{Cotton}* | *<adult>* | *{Pink}* | *{Blue}* | *{STW}* | *PISD#6* |
| ¬*{Cotton}* | *<adult>* | *{Yellow}* | *{Blue}* | *{STW}* | *YSD#8* |

### 5.4. Summary: T-shirt as c-tuples

The T-shirt example in this Section 5 aims to show several things:

- C-tuples can be used to efficiently represent variant tables for individual constraints, as well as the overall set of variants. The underlying property domains are defined in Section 3, summarized in Table 2. The four individual constraints are given in Table 4, Table 5, Table 6, and Table 8. The overall set of variants are shown in Table 3, Table 7, and Table 9. All tables are in c-tuple format.
- Compression is a significant factor in handling large variant tables. In extensional form, there are many millions of valid variants. In compressed form, they can be easily represented using only thousands of value symbols.
- It can be feasible to maintain an overall table of all variants in c-tuple form, as demonstrated by the T-shirt example. In practice this will depend on support by dedicated maintenance aids. This is a topic beyond the scope of this work, but a vision we offer to consider when starting or re-engineering an MC enterprise.
- Tables used in sales may differ from those used in manufacturing. It will be necessary to lookup a sales variant in a manufacturing table and vice versa. The interoperation between database tables is well-established in business practice. The filtering queries (2) and (3) described in Section 7.2 can be efficiently supported using the forms of compression which we discuss here.

## 6. OVERVIEW COMPRESSION RESULTS

There are two ways to measure the effect of compression on a variant table in extensional form: First, to look at a partition of the table into c-tuples and to compare the number of rows in the table with the number of c-tuples. Second, to compare the number of cells in

the table (each cell containing one value) with the number of value symbols used in the compressed form. The latter comparison is particularly interesting here, because it is related to performance of queries/filtering the table. In the worst case, all value symbols must be inspected to confirm validity of a particular value. Because the compression to c-tuples reduces the number of symbols needed, this process is accelerated accordingly. The number of symbols needed can be reduced further when using an advanced compression technique (see Section 7.1).

Table 10 gives an overview of the relevant compression results for all the tables we have discussed as well as some tables resembling actual product data. Table 10 is split into three sections. The first section shows the compression of the individual T-shirt constraints of Table 4, Table 5, Table 6, and Table 8. The second section shows the compression results for Table 2 ("New T-Shirt all"), Table 3 ("New T-Shirt basic"), Table 7 ("New T-Shirt sales"), and Table 9 ("New T-Shirt mfg" (manufacturing)) that each contain an entire set of variants, i.e. represent an entire product model. The third section shows the compression of two tables from an older published benchmark for a Renault Megane (see [4]) using advanced compression introduced in Section 7.

The variant tables for the constraints in the first section are not huge. Compression is not that significant. However, as colors and imprints are added, the extensional size of the tables will grow exponentially, whereas the number of symbols needed in the compression (column #symbols), grows only linearly (also see Section 8.1).

"New T-shirt mfg" (Table 9) represents the example with the most c-tuples and is the most realistic of the three. It is compressed from a quarter of a billion rows to merely 19 c-tuples and needs around 10,000 symbols. The compression of the similarly large "New T-shirt sales" table (Table 7) needs only approximately one third as many c-tuples and half as many symbols. These compressed sizes are very reasonable and compare with results from prior work [1], [3].

For the two tables in the third section, it is interesting to note that compression for the larger table "Megane table #71" is significant. In compressed form, "Megane table #71" is actually smaller than the compression of the smaller "Megane table #1". In other words, the large table #71 with

almost 300,000 cells reduces to only 150 symbols, while the smaller table #1, which has only 1,476 cells, needs 288 symbols. The compression of table #71 measurably benefits performance in practice. This also demonstrates that the sheer number of rows of a table or its size are not suitable measures of complexity.

If we define the compression ratio $\rho$ of a variant table as:

$$\rho = 1 - \frac{\#symbols}{\#cells}$$

then we have $\rho$ = 99.9% for the tables "Megane table #71", "New T-Shirt sales" (Table 7), and "New T-Shirt mfg" (Table 9). Our experiences so far indicate that a compression ratio $\rho$ > 85% can be expected for variant tables of a size where compression becomes useful. As the examples show, compression gets more extreme for larger tables. Indeed, when variant tables do not compress, it may be sensible to review the product to verify that the irregularities causing this are intentional and unavoidable.

## 7. ADVANCED COMPRESSION

Compression of variant tables to c-tuples is powerful. It is the first choice because c-tuples have a transparent representation in a spreadsheet and are supported in other tools. However, when each variant is associated with unique data such as a variant ID, or a dynamically changing value for, say, the amount on stock, c-tuples may degenerate to tuples of singleton sets, i.e. the tables cannot be compressed using c-tuples. Advanced compression to a *decision diagram* can go further and deal with these issues. Our choice is a *variant decomposition diagram* (VDD), which was designed with this purpose in mind.[9] A VDD directly encodes a partition of the table it represents into c-tuples, while making use of further commonalities the c-tuples may share. In this sense, an approach at compression using a VDD is compatible with an approach with c-tuples in mind.

We refer to a VDD as a *variant decomposition diagram*, rather than a *variant decision diagram*, because we emphasize its relationship to table decomposition. Table 11 identifies blocks of partial c-tuple that multiple c-tuples have in common and that can be exploited when compressing to a VDD (see the construction of VDDs in [3], [5]). This further step in compression resembles the step compressing individual rows to c-tuples.

Table 10. *Utility of compression overview*

| *Variant table* | *#rows* | *#c-tuples* | *#cells* | *#symbols* | *ρ (%)* |
|---|---|---|---|---|---|
| *Constraint "MIB" (Table 4)* | 8,009 | 2 | 16,018 | 1,011 | 94 |
| *Constraint "STW" (Table 5)* | 8,012 | 2 | 16,024 | 1,014 | 94 |
| *Constraint "Color ≠ImpCol" (Table 6)* | 35,043 | 6 | 105,129 | 5,055 | 95 |
| *Constraint "Dye" (Table 8)* | 24 | 9 | 72 | 36 | 39 |
| *New T-Shirt all (Table. 2)* | 276,480,000 | 1 | 2,488,320,000 | 1,042 | 99.9 |
| *New T-Shirt basic (Table 3)* | 276,514,560 | 3 | 2,488,631,040 | 1,100 | 99.9 |
| *New T-Shirt sales (Table 7)* | 241,954,560 | 7 | 2,177,591,040 | 5,252 | 99.9 |
| *New T-Shirt mfg (Table 9)* | 276,514,560 | 19 | 2,765,145,600 | 9,930 | 99.9 |
| *Megane table #1* | 164 | 56 | 1,476 | 288 | 80 |
| *Megane table #71* | 48,722 | 45 | 292,332 | 150 | 99.9 |

[9] Various common forms of DD can be mapped to each other. See [3] on the relationship of VDDs to *multi-valued decision diagrams* (MDDs). See [13] on the relationship between *binary decision diagrams* (BDDs) and *zero-suppressed binary decision diagrams* ZDDs.

### 7.1 Variant decomposition diagrams (VDDs)

Figure 2 depicts a VDD for Table 2. Each node is labeled with a set of feature values for a product property and has two emanating links, *HI* and *LO*: It is beyond the scope of this paper to explain VDDs in detail, for which we refer to [3]. Briefly, however,

The *HI*-link of a node points to a node for another product property or to the terminal sink ⊤ (*true*),

The *LO*-link points to a node pertaining to the same product property or to the terminal sink ⊥ (*false*),

The *HI*-links from a chain of nodes from the root to ⊤ define a c-tuple.

If several c-tuples have a partial c-tuple in common, then the VDD will try to represent this common part of the c-tuples only once.

The compression to a VDD, which we envision here, is determined uniquely by a given ordering of the product properties. Different orders result in VDDs with different compression efficiency.

Table 11. *Common partial c-tuples in Table 9*

| *Color* | *Fabric* | *Dye* | *Imprint* | *ImpCol* | *Size* |
|---|---|---|---|---|---|
| *{Black}* | * | *none* | *{MIB}* | *{White}* | * |
| *<std>* | * | *none* | *{STW}* | *{Blue}* | *<adult>* |
| *<std>* | * | *none* | *¬<vintage>* | * | * |
| *{Green}* | *{Cotton}* | *GRCD#1* | *¬<vintage>* | * | * |
| *{Purple}* | *{Cotton}* | *PUCD#3* | *¬<vintage>* | * | * |
| *{Pink}* | *{Cotton}* | *PICD#5* | *¬<vintage>* | * | * |
| *{Yellow}* | *{Cotton}* | *YCD#7* | *¬<vintage>* | * | * |
| *{Green}* | *¬{Cotton}* | *GRSD#2* | *¬<vintage>* | * | * |
| *{Purple}* | *¬{Cotton}* | *PUSD#4* | *¬<vintage>* | * | * |
| *{Pink}* | *¬{Cotton}* | *PISD#6* | *¬<vintage>* | * | * |
| *{Yellow}* | *¬{Cotton}* | *YSD#8* | *¬<vintage>* | * | * |
| *{Green}* | *{Cotton}* | *GRCD#1* | *{STW}* | *{Blue}* | *<adult>* |
| *{Purple}* | *{Cotton}* | *PUCD#3* | *{STW}* | *{Blue}* | *<adult>* |
| *{Pink}* | *{Cotton}* | *PICD#5* | *{STW}* | *{Blue}* | *<adult>* |
| *{Yellow}* | *{Cotton}* | *YCD#7* | *{STW}* | *{Blue}* | *<adult>* |
| *{Green}* | *¬{Cotton}* | *GRSD#2* | *{STW}* | *{Blue}* | *<adult>* |
| *{Purple}* | *¬{Cotton}* | *PUSD#4* | *{STW}* | *{Blue}* | *<adult>* |
| *{Pink}* | *¬{Cotton}* | *PISD#6* | *{STW}* | *{Blue}* | *<adult>* |
| *{Yellow}* | *¬{Cotton}* | *YSD#8* | *{STW}* | *{Blue}* | *<adult>* |

We are currently investigating, whether the order of the product properties may also be important for the business logistics (see Section 8.2).

In a VDD the indicator *#symbols* for compression is the sum of the sizes of all node labels.

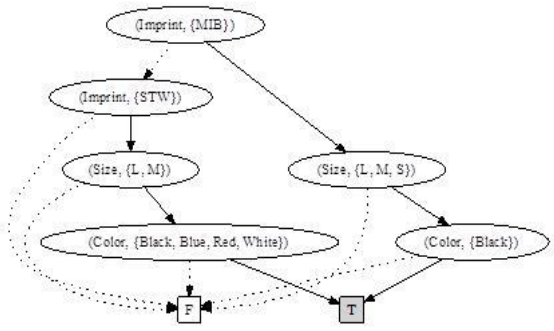



Fig. 2. *VDD of a T-shirt with set-labeled nodes*

### 7.2 Queries to compressed tables

In the context of MC, a configurator must be able to find the set of all rows in a variant table that match given user selections or exclusions of product features. Also, it must be possible to determine the *domain restrictions*, i.e. exclude any feature that can no longer be chosen according to user criteria. To this end, the configurator will need to query/filter variant tables in the product model.

A variant table in a relational database can be queried using SQL (Structured Query Language) [6]. We consider SQL queries of the form (2) and (3) to be the most relevant for configuration (see [3]). They allow both the filtering of a given set of variants to match given criteria, and provide the resulting *domain restrictions* for each of the properties.

The query in (2) returns the set of variant table rows that match the criteria of the *WHERE* clause as its *result set*. Importantly, the result set of a query to a compressed table will also be in the form of a compressed table and can be queried further, i.e. huge result sets and complex queries can be handled efficiently.

```
SELECT * FROM ⟨vtab⟩
WHERE ⟨v1⟩ IN ⟨R1⟩ AND … ⟨vk⟩ IN ⟨Rk⟩;
```

In contrast, the query in (3) returns the domain restriction for property $v_j$ under the same *WHERE* clause.

```
SELECT DISTINCT ⟨vj⟩ FROM ⟨vtab⟩
WHERE ⟨v1⟩ IN ⟨R1⟩ AND … ⟨vk⟩ IN ⟨Rk⟩;
```

To summarize, for efficient handling of variant tables, we must be able to filter to the set of all rows that match a given query condition, and obtain the domain restrictions expressed in a query result set. These queries can be supported on a compressed format using either c-tuples or VDDs.

## 8. PRODUCT COMPLEXITY

A business is faced with the following challenges when dealing with product variants:

- A *product model* must be maintained, perhaps differently for sales and production.
- *Sales* must ensure that only valid and producible variants are sold; this is usually the task of a sales configurator using a product model for sales.
- The customization of variant as sold must often be augmented for production; this is usually also the task of a configurator using a product model for production.
- The affected business process must be able to access the customized features of the variants and act on them.
- The product model and the production data and set-up must be kept in synch to ensure that what is offered can be built, and what is being produced is actually offered for sale.

The degree of difficulty this adds over dealing in plain non-configurable products is what we consider to be the *complexity* of the MC product. Complexity should be kept as low as possible while retaining the competitive edge individualization offers. If complexity

is higher than expected, then it may be advisable to review the product definition. This section focusses on a proposal on how to assess variant complexity.

We do not address the issues around product modeling here (see [7], [8]). Instead, we take it for granted that using tables wherever possible alleviates the problems of sharing variant and business data, as well as that of accessing variant data in the business processes. In an ideal scenario, the business data of an MC product would be maintained in the business system just as the data of any other non-configurable product. This data consists of a generic part common to all its variants as well as data for individualized product features defined as a variant table. Generally, product model data in the form of tables offers the possibility of re-use as business data and vice versa.

We have already established that there is no direct correlation of complexity and total number of variants offered. Section 8.1 deals with complexity from the perspective of configuration; Section 8.2 with the perspective of product logistics. These proposals apply to all variant tables, not just ones representing overall sets of variants.

### 8.1 Variant complexity in configuration/queries

We can relate the performance of the queries/filtering in (2) and (3) to the number of symbols that need to be tested against the *query condition* formulated in the *WHERE*-clause (see [3]). The worst case is when all symbols in all c-tuples or VDD nodes (see Section 7.1) need to be inspected. The *complexity measure* we propose for configuration is the number of symbols needed for the compressed form, i.e. the column *#symbols* in Table 10. As established before, we see that complexity is not related to the size of the table. In general, the compressed form grows linearly in the number of symbols, whereas table size grows exponentially in number of cells - subject to added constraints, which increase complexity. In conclusion, complexity affects performance time. A table with high complexity requires more time to process than a table with a low complexity

Note that instead of maintaining the list of all valid variants, it is also conceivable to maintain the list of all invalid variants (the exclusions) in a table. Figure 3 suggests a two-column variant table. Each point in the plane is a combination of *Color* and *Imprint*. Combinations that are valid are represented with a *filled* background; invalid ones with a *transparent* background. Figure 3 shows the extreme cases of a single valid variant, versus a single invalid variant. The complexity should be the same. Since compression is not unique and depends on heuristics, it is not clear that the same complexity will actually be obtained in practice. But investigations in [5] indicate that indeed it does not make much difference in complexity, whether the positive set of valid variants or the negative set of excluded variants is used at the outset when compressing to a VDD.

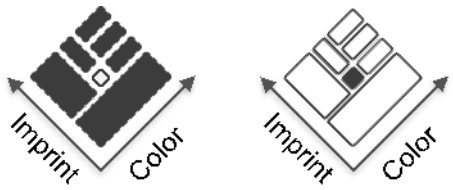



Fig. 3. *Positive and negative depiction of a variant relation*

In case the product model is made up of several variant tables, the complexity observation given here applies to each one individually. Additional processing is needed to ensure queries are correct across the tables. Local constraint propagation is one popular choice in interactive configuration to propagate domain restrictions that result from a query condition to other tables to obtain further restrictions. Local constraint propagation can be supported efficiently on the compressed formats we discuss here (see [3]).

The question remains, what can be expected as reasonable complexity of a variant table. We have limited access to models that have been compiled into a single variant table. Our own, still fairly simple T-shirt example has a complexity of around 5,000 for Table 7 and around 10,000 for Table 9. Advanced compression to a VDD can reduce the 9224 value symbols in Table 9 to 1098 by counting only the distinct symbols in the blocks in Table 11. This translates into a performance relevant gain of almost a factor of ten.[10]

We have more data for individual variant tables. As Table 10 shows for the two Megane tables, complexity is not linked to the size of the table. The majority of variant tables we have encountered has a complexity measure of less than or equal to 1000.

### 8.2 Variant complexity in a business process

A c-tuple represents an *unconstrained* subset of variants. From the business point of view this should mean that only values (features), but not combinations of values affect the business decisions or rules that need to be applied when producing variants within this set. A complexity measure for an unconstrained subset would be based only on the occurring values, similar to the variant complexity introduced in Section 8.1. We illustrate this with an assumed production setup depicted in Figure 4.

Figure 4 shows the production line of the new T-shirt. The steps in a production line will be organized to process features in a particular order. First, a T-shirt that matches the desired personalization of the properties *Style*, *Neck*, *Fabric*, *Size*, and *Color* is taken from stock. Then the T-shirt is dyed (*Dye* - in case of a standard color no dyeing is required), followed by making the desired imprint. For non-vintage imprints, the new offset printer is used with the proper color and size (*Imprint*, *ImpCol*, *ImpSiz*). In case of a vintage imprint, processing

[10] We can put complexity into perspective of performance time. Results published in Table 11 in [3] suggest a conversion factor of $c_p = 0.02$ ms (milliseconds) per VDD symbol. Under the same hardware and software setting, any query to Table 7 can be guaranteed in less than 100 ms and less than 200 ms for Table 9. Our experiences indicate that real response times are much faster.

is routed to separate dedicated silk-screen printing facilities. Here the imprint size and the imprint color are determined by the production process and need not be specially regarded. Finally, a certificate for the *$CO_2$-Offset* is stamped on the T-shirt, the T-shirt is packed, shipped, and invoiced. Packaging and invoicing may need access to the personalization features in order to identify them on the package and the invoice.

As illustrated in Figure 4, the business needs work stations for pulling T-shirts from stock, for dyeing them, for printing, and for stamping the certificate, packaging, invoicing, and delivery.

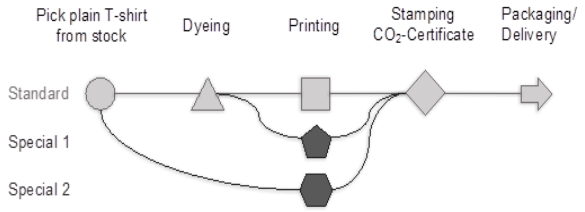



Fig. 4. *Production process of personalized T-shirt*

Figure 4 consists of three parallel production lines that correspond to the three c-tuples in Table 3. It suggests a correlation between compression and the business setup. Similar to the measure for variant complexity defined in Section 8.1, we suggest to define a measure for business complexity based on the associated compression. We have published first ideas on this in [9]. There, a cost coefficient $\omega$ was assigned to each property and a weight *w* to each feature. Business complexity was then defined as the weighted sum of all symbols occurring in the compression, the weight of a symbol being $\omega_{\text{property}} w_{feature}$.

For example, in Figure 4, the first production line 'Standard' represents the vast majority of variants, all with non-vintage imprints. Removing a number of imprints from this production line would drastically reduce the overall number of variants, but the sum of their weights would not affect the business cost much. However, two additional production lines (Specials 1 and 2) are required for the two vintage prints MIB and STW. The weight of MIB and STW can be taken to be very high. Therefore the decision to include vintage prints has a noticeable effect on cost although the total number of variants does not change much. In conclusion, complexity seems related to compression and the assigned coefficients/weights but is not reflected by the number of variants.

We think that it is worthwhile to further investigate this approach, however this goes beyond the scope of this paper.

## 9. CONCLUSION AND OUTLOOK

Tables are the pillar of business data for non-configurable (*mass producible*) products. Mass customization adds personalization / individualization / customization to such products. The individualized products are *variants* of an underlying generic *MC product*. An MC product has the same requirements for basic business data as a non-configurable product. In addition, it requires data for the variants of which each is described by a list of individualized features. In MC, this list will follow a regular scheme, defined by the MC product for all its variants. The MC product defines both a finite set of customizable product properties and a *domain* of the choices offered for that product property. The assignment of a product property to an element in its domain defines a *product feature*.

Here, we consider the classic case that each product property domain is finite. Extending the ideas presented here to non-finite domains is ongoing work [10]. Each variant is described by its features in the form of a *value assignment* to all product assignments. Such a value assignment can be stored as a row in a table. We generally call any table of combinations of product features a *variant table*. The tabular paradigm of variant tables fit smoothly to the tables a business already deploys, except that they do not scale. Variant tables grow exponentially as the domains and/or properties grow.

While variant tables are an established popular element of product models, their use has been severely limited by their lack of scalability so far. Particularly, the plurality of small the tables embodying one constraint each, while readily processed by a configurator, are not as easy to query from a business perspective, necessitating special considerations.

Despite clear benefits, c-tuples are only little used. This is likely related to the fact that although c-tuples are generally straightforward to read, they are hard to work with. A current focus of our research and development is on tools to overcome this and to provide an interactive maintenance environment.

Our main contribution in this work is to show that utilizing table compression will greatly empower the use of huge variant tables, overcoming the limitations perceived so far. The compression results we give in Table 10 speak for themselves. Leaving an increase of constraints aside, the compressed size of a variant table grows only linearly with the offered customization choices, while the uncompressed size grows exponentially.

We propose several levels of empowerment of variant tables:

1. The complexity of a set of variants or of a single variant can be easily analyzed by compressing it, even without any intention of further use of the compressed result. This may also offer interesting insights for the business as we discussed in Section 8.2.
2. Without any change to an existing legacy MC environment, proper tools would provide a much better interaction with variant tables. This would greatly enhance working with existing large tables in product models. This will apply to anyone working with variant tables.
3. Using the currently existing possibilities of storing variant tables in a c-tuple format in spreadsheets and some configuration environments, much larger tables can be used in legacy configuration than hitherto practical,

again without changes to the legacy system11. This will apply to legacy systems that make use of c-tuples.

4. Where legacy systems use database queries of the form (2) and (3), it could be analysed whether the queries can be rerouted to a compressed format to facilitate handling of larger tables and increase efficiency.
5. When starting or re-engineering an MC enterprise, maintaining a single table of all variants could be envisioned. We used the simple setting of our T-shirt as an example for this. Compiling such a table is a problem of its own that would need to be solved.

There are several further issues under current investigation or development:

- Further data needs to be collected on what "good" compression is and may be expected, and when models should be considered overly complex.
- We limited our discussion to the classic case of finite property domains. Extending the ideas presented here to non-finite domains is ongoing work [10].
- The ideas we set forth on the link between compression and the business set-up are speculative and require verification in the field.
- We believe that the queries discussed in Section 7.2 are the relevant ones for MC. Collecting requirements for extending these is a topic of future work.

Our current research and development focus is to make the maintenance of huge tables feasible and attractive.

## 10. REFERENCES


[1] Wikipedia, "The wheat and chessboard problem." .

[2] G. Katsirelos and T. Walsh, "A Compression Algorithm for Large Arity Extensional Constraints," in *Principles and Practice of Constraint Programming – CP 2007. Lecture Notes in Computer Science, vol 4741.*, 2007, pp. 379–393.

[3] A. Haag, "Managing variants of a personalized product: Practical compression and fast evaluation of variant tables," *J. Intell. Inf. Syst.*, vol. 49, no. 1, pp. 59–86, 2017.

[4] J. Amilhastre, H. Fargièr, and P. Marquis, "Consistency restoration and explanations in dynamic CSPs -- Application to configuration," *Artif. Intell.*, vol. 135, no. 1–2, pp. 199–234, 2002.

[5] A. Haag, "Column Oriented Compilation of Variant Tables," *Proc. 17th Int. Config. Work. Vienna, Austria, Sept. 10-11, 2015*, vol. 1453 of CE, pp. 89–96.

[6] Wikipedia, "SQL." .

[7] K. Kristjansdottir, S. Shafiee, L. Hvam, L. Battistello, and C. Forza, "Complexity of Configurators Relative to Integrations and Field of Application," in *Proceedings of the 19th International Configuration Workshop, Paris, France, October 2017*.

[8] U. Blumöhr, M. Münch, and M. Ukalovic, *Variant Configuration in SAP*, Second Ed. SAP Press, Galileo Press, 2012.

[9] A. Haag, "Assessing the complexity expressed in a variant table," in *Proceedings of the 19th International Configuration Workshop, Paris, France, September*, 2017, pp. 20–27.

[10] A. Haag, "Quasi-finite domains: dealing with the infinite in mass customization," in *Proceedings of the 20th International Configuration Workshop, Graz, Austria, September 2018*, p. submitted.

[11] A. Haag, "Chapter 27 - Product Configuration in SAP: A Retrospective," in *Knowledge-Based Configuration*, A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen, Eds. Morgan Kaufmann, Boston, 2014, pp. 319–337.

[12] U. Blumöhr, M. Münch, and M. Ukalovic, "Chapter 2.6.3 - Variant Tables in Detail," in *Variant Configuration in SAP*, Second Ed., SAP Press, Galileo Press, 2012, pp. 152–158.

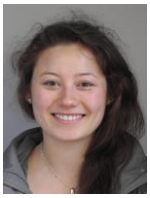
[13] D. E. Knuth, "Chapter 7.1.4. - Binary Decision Diagrams," in *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms*, Boston: Pearson Education, 2011, pp. 202–280.

[14] A. Haag and C. Faure, "Polishing a raw diamond," in *Configuration Workgroup, Berlin, Germany, April 2018*.


[11] We have recently demonstrated the performance possible in utilizing a c-tuple format with the SAP VC [14]

## CORRESPONDENCE


Dr Albert Haag, CEO
Product Management Haag GmbH
Sonnenwendstrasse 50
67098 Bad Dürkheim, Germany
albert@product-management-haag.de


Laura Haag, MSc.
University of East Anglia
Department of Nutrition and Preventive Medicine
Bob Champion Research and Education Building
James Watson Road
Norwich NR4 7UQ, England
l.haag@uea.ac.uk