International Journal of Industrial Engineering and Management (IJIEM), Vol.10 No 2, June 2019, pp. 155-170
ISSN 2217-2661

DOI: http://doi.org/10.24867/IJIEM-2019-2-236

Further Empowering Variant Tables for Mass Customization

Albert Haag
CEO, Product Management GmbH, Bad Dürkheim, Germany, email: info@product-management-haag.de

Laura Haag
PhD Student, University of East Anglia, Norwich, UK, email: l.haag@uea.ac.uk

Received (10.11.2018.); Revised (25.03.2019); Accepted (1.04.2019.)

Abstract

Tables are a standard form of data representation in business. A variant table lists valid or excluded combinations of product features where each table column refers to a product property and each table row denotes a combination of product features. A table cell defines a feature, e.g. Color = Red, as an assignment of its value to the column's property. As technology and consumer demand drive ever increasing product choices, the number of feature combinations that can be offered for a product increases exponentially and can easily exceed the limits of a traditional table. However, variant tables can often be compressed in a way that scales both in size and query performance while retaining the tabular paradigm in a manner useful for a business. The basic idea is to partition the table rows into unconstrained slices, where each slice consists of all possible combinations of the product features it references. Such a slice can be represented as a c-tuple and readily stored in a spreadsheet. C-tuple representation is already supported in some product configurators. We give examples of products where it is feasible to efficiently represent all valid variants in one overall table using c-tuple compression. For cases where c-tuples do not suffice, the stronger compression to a variant decomposition diagram (VDD), a form of decision diagram, can be used. We propose complexity measures for a product based on the compressibility of its variants and discuss their usefulness to the business. We illustrate these ideas with examples and present some results on dealing with variant tables from real-world product models. We show that compression empowers variant tables by enabling enormous tables to be functionally used in a way like regular tables.

Keywords: *product configuration, product modeling, table compression, variant table*

1. INTRODUCTION

Mass customization (MC) combines product customization with mass production. Ways are sought to produce products that are customized to individual needs with the cost-efficiency attributed to mass production [1, 2]. In this paper, we consider MC products that can be defined by enumerating their variants. All variants are described using the same product properties (such as Size or Color for a T-shirt), but differ in the individual values that are assigned to the properties.
An increase in the number of choices typically causes an exponential explosion in the number of product variants. In a real context, the number of variants of a product can easily transcend the number of potential customers. This means that a business must be prepared to produce a given variant in a lot size of one.
Tools are needed to support the variability of MC products, which includes defining the valid product variants, guiding the user interaction when ordering a variant, and ensuring correct fulfillment of the order (e.g. manufacturing and invoicing) [3, 4]. Particularly product configurators, such as the SAP Variant Configurator (SAP VC) [5], play a central role in this [6, 7].
Every configurator has its own methods to model a product, i.e. to define its product variants. Tables are a natural element of product models and are universally understood. They are considered a preferred way of modeling [5], however the limiting factor with using tables is that they don't scale with an increasing number of choices. The limits of conventional databases and spreadsheets can be quickly reached.
To overcome this problem, different table compression techniques can be applied in a way that retains the functionality of a regular table. The basic technique focussed on in this paper is compression to *c-tuples* [8]. A c-tuple is a table row with multiple values in its cells. The other technique is compression to *variant decomposition diagrams,* a form of decision diagram [9]. C-tuples are not new and have been supported in the SAP VC [5] since the 1990's, however their potential has not yet been universally recognized, and their use is discouraged [10].

The objective of this paper is to demonstrate the utility of table compression in the context of MC. This enables handling larger tables more efficiently, and greatly empowers tables as a preferred element of MC product models. In addition, we define a complexity measure for product models based on compressibility and discuss its usefulness for the business.

To pursue this objective, the paper gradually introduces the terminology used to describe the table compression techniques. Two MC products are used as examples: a mass customizable T-shirt and a model of the Renault Megane, which was published by Renault as a benchmark for configurators [11]. The methods used for the different analyses are presented throughout the article as they occur as this makes it easier to follow.

2. BACKGROUND

2.1 The role of product variants in mass production

When Henry Ford originally mass produced his Model-T car, it was offered in only one color: black. Given that artisans of the time were able to produce vehicles in multiple colors to suit individual tastes of their clients, the innovation of mass production lay in eliminating the variability of a product in favor of the assembly line, which enabled a cheaper, stream-lined production process [12]. At the time, offering the Model-T in a second color might have meant providing a second assembly line, perhaps in a separate production plant. Indeed, the classical approach to mass production is to define and manage each product completely on its own. A *product key* or *material number* completely identifies a product and its specification for purposes of sales and manufacturing. Variability must be handled by introducing a unique product key for each variation.

On the other hand, commonalities in the design and production processes of similar items have been exploited to improve efficiency since antiquity. The Pont du Gard, a Roman aqueduct in southern France, is a complex masonry structure composed of multiple stone arches. The local museum there exhibits the Roman design for the construction of a generic stone arch, which served as a blue-print that could be annotated with the dimensions for each individual arch needed in the construction of the aqueduct (http://www.pontdugard.fr/en/espace-culturel/museum). This allowed for faster design and construction of the entire structure. Each arch is a variant of a common scheme. In MC terms, the arches are product variants of the generic product *aqueduct arch*.

The drive to exploit such similarities when offering a plurality of products is a basic economic force and may even become a necessity. The simple T-shirt, which we use as one example in Section 4, consists of eleven T-shirt variants. The T-shirt is offered in three sizes, four colors, and with two different imprints. The business may define eleven separate products that may be produced and stocked. However, it can also choose to define one product, *Simple T-shirt*, with eleven variants. The latter approach would collect most of the product business data, e.g. the material master data collected in the MARA table in SAP (see [5]), centrally with a single product definition. The possible variants are then defined separately as in Table 1. This approach becomes more attractive as variants increase and advances in production technology allow more and more flexibility in the production processes. It becomes a necessity as customer demand and technology cause the number of offered variants to become too large to handle individually. The modern T-shirt example in Section 4 offers T-shirts with more choices, which results in more than a quarter billion possible variants, which cannot be defined as individual products due to their sheer number.

2.2 Variant tables

The term *variant table* is used in SAP VC modeling [10] to refer to a table listing valid (or excluded) combinations of product features.

A variant table column refers to a *product property* (e.g. Imprint, Size or Color for a T-shirt). The value in each table cell denotes a value assignment to the column's property and defines a *product feature*, e.g. *Color* = *Red*. We adopt this term and use it extensively throughout this paper. Table 1 is an example of a variant table listing eleven valid combinations of a simple T-shirt.

Table 1 Simple T-shirt table of variants

Imprint	Size	Color
MIB	Small (S)	Black
MIB	Medium (M)	Black
MIB	Large (L)	Black
STW	Medium (M)	Black
STW	Medium (M)	Blue
STW	Medium (M)	Red
STW	Medium (M)	White
STW	Large (L)	Black
STW	Large (L)	Blue
STW	Large (L)	Red
STW	Large (L)	White

A variant table constrains valid combinations of product features. Therefore, it naturally functions as a constraint in a product model. In this paper, we assume that all product properties have a finite domain. In this case, any constraining relation between product features could be expressed as a variant table. However, size is often a limitation that prevents this, due to a combinatorial explosion of the way features may be combined as choices increase. In some cases, it is more efficient to list the invalid combinations (*exclusions*), when these are fewer than the valid ones. A table listing exclusions is termed a *negative variant table*. Negative variant tables are not definitive in themselves. They require additional definitions of the underlying domains of the product properties when used operationally [13, 14].

Subject to their size, variant tables are often the method of choice for capturing the valid combinations (or conversely the invalid ones) [10]. In this paper, we argue that the compression techniques in Sections 3 and 6 will largely mitigate the size limitation and make variant tables attractive as a primary means for modeling the variability of MC products. If a variant table pertaining to an MC product cannot be compressed, this is due to irregularities in the product definition, which should be reviewed to remain true to the aspect of being able to mass produce. Indeed, it should be possible to construct a single variant table that enumerates all variants of the product. Such a table is very wide: its columns encompass all product properties. It can be maintained directly, as for the T-shirt example in Section 4, or constructed by joining a set of individual variant table constraints. The latter aspect is beyond the scope of this paper.

2.3 Product variants in mass customization

We take *mass customization* (MC) to refer to *mass production* coupled with *individualization.* By this we mean that the number of variants of a product exceeds the number of potential customers, allowing everyone to potentially own a unique product[1]. Offering a T-shirt with a quarter billion variants can be regarded as *mass customization* (MC) in this sense.

The production of individualized T-shirts and cars follows a continuous evolution of production technology, which allows increasing flexibility in dealing with product variants. The business processes of the value chain, i.e. ordering, production, invoicing and delivery of a product, all operate on the business data of a common underlying product, but additionally require the specification of an additional set of product features that define the specific variant. Different business processes may differ in the product features that need to be specified. The descriptive properties for sales often differ from those for manufacturing and both must be reconciled in delivery and invoicing [3]. Technological advances in production technology are also a prerequisite for MC: it is now becoming profitable to produce products in a lot size of one.

3. C-TUPLES - BASIC COMPRESSION

In the following, we assume a finite set of k product properties with finite domains[2] $D_1, \dots, D_k$. We denote the number of values in the j-th domain by $s_j := |D_j|$.

In the special case of an *unconstrained* MC product that allows arbitrary combinations of values from the property domains, the overall set of valid combinations is just the *Cartesian product* of the property domains $D_1 \times D_2 \times \dots \times D_k$. The number of value symbols needed to represent this Cartesian product is the sum of all the domain sizes (1). The number of combinations this represents is the product of the domain sizes (2). In the example of the "individualized T-shirt" in Section 4 we have $k = 9$, $N = 276{,}480{,}000$, while $S = 1044$. It is clearly infeasible to have an extensional representation of all these combinations in one table, but the Cartesian product can be represented in a very compact fashion as a tuple of the domains (3).

$$\mathrm{S} := \mathrm{s}_1 + \mathrm{s}_2 + \dots + \mathrm{s}_\mathrm{k} \quad (1)$$
$$\mathrm{N} := \mathrm{s}_1\,\mathrm{s}_2\,\dots\,\mathrm{s}_\mathrm{k} \quad (2)$$
$$\langle \mathrm{D}_1, \mathrm{D}_2, \dots, \mathrm{D}_\mathrm{k} \rangle \quad (3)$$

In Table 1, each row equals a tuple of values, which we refer to as an *r-tuple*, short for *relational tuple*. The concept of a *c-tuple* extends the concept of an *r-tuple* in allowing a table cell to hold a set of values from the column's property in comparison to only one value as in an r-tuple. The term *c-tuple* is short for *Cartesian tuple*. The tuple of domains (3) is an example of a *c-tuple*. We shall refer to the *size* of the c-tuple as S in (1).

A c-tuple represents an *unconstrained subset* of valid combinations, i.e. no constraints apply within this subset. The approach to simple compression of a variant table is to reorganize and partition a variant table into unconstrained sets of combinations and to then replace each such set by a c-tuple. For example, Table 1 can be reformulated using two c-tuples as in Table 2. Multiple values occurring in one cell are separated by a semi-colon. Similarly, the quarter of a billion variants of the unconstrained "individualized T-shirt" introduced above can be captured in a variant table with one row (see the first c-tuple in Table 4).

Table 2 Simple T-shirt represented in c-tuples

Imprint	Size	Color
MIB	S; M; L	Black
STW	M; L	Black; Blue; Red; White

It is important to note that the decomposition of a variant table into c-tuples is not unique, i.e. different decompositions may have different sizes. Heuristics are key to finding a good decomposition [8]. The compression results we present in this paper are not based on the work in [8], but rather on c-tuples derived from an advanced compression to a *variant decomposition diagram* (see Section 6).

4. EXAMPLE T-SHIRT SCENARIO

To better illustrate c-tuple compression and motivate the discussion on complexity we use the following exemplary scenario of a fictional business offering T-shirts:

At its early inception, the T-shirt business begins by selling a small number of T-shirt variants. They provide T-shirts with one of two possible imprints (*MIB – "Men In Black"*, *STW – "Save The Whales"*). They procure standard white T-shirts in three sizes (*S* (*Small*), *M* (*Medium*), *L (Large)*) from an outside source, which they optionally dye in one of three colors (*Black*, *Blue*, *Red*). Two silk-screens are used to make the imprint. One is set up for *MIB*, a print in heavy white done only on black T-shirts of all sizes. The other for *STW,* a larger blueish imprint done on T-shirts of any color (including *Blue*), but not for small sizes. Table 1 is the definitive list of all eleven T-shirt variants. The product is illustrated in Figure 1.

[1] We consider *personalization* to be more demanding: satisfying arbitrary customer requests such as producing a T-shirt in any color, as opposed to just offering many colors.

[2] The finiteness requirement may be relaxed (see [23]).

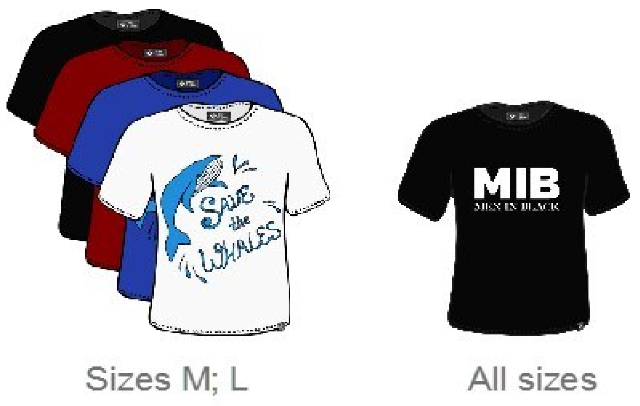

Figure 1 Simple T-shirts

The business expands over time and technology and customer expectations advance. A new T-shirt is designed. Five additional properties are added for individualization: *Style*, *Neck*, *Fabric*, *ImpColor* (color of imprint), and *ImpSize* (size of imprint). More T-shirt colors are added. The acquisition of a new textile printer allows a drastic increase in the 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 sleeve. Table 3 gives an overview over the T-shirt properties and their domains.

Table 3 Features of the individualized T-shirt

Property	Domain Size	Values
Style	4	NoSleeve; HalfSleeve; FullSleeve; Hoodie;
Neck	3	Round; VNeck; Collar
Fabric	3	Cotton; Synthetic; Mixed
Size	8	3T; 4T; XS; S; M; L; XL; XXL
Color	8	Black; Blue; Red; White; Green; Purple; Pink; Yellow
Imprint	1000	*1000 imprints from catalog*
Imp-Color	5	Green; Black; Blue; Red; White
ImpSize	8	Baby; Tiny; Cute; Small; Medium; Big; ExtraBig; Fill
CO_2-Offset	3	$0.00; $0.99; $1.99

Initially, the business does not define any constraints: any combination of product features from the domains is considered valid. Later, customer demand causes the reactivation of the two vintage silk-screen imprints: *MIB*, only on black shirts, and *STW*, not on small shirts, bringing the number of offered imprints to 1002. The T-shirt variants at this stage can be represented in the three c-tuples in Table 4.

The *wildcard symbol* "*" has been used as shorthand to refer to the entire domain of a product property. For space reasons, we omit any column that consists solely of wildcards in Table 4 and all the following variant tables. For any property that is not represented by a column in a variant table, one can always assume a column of wildcards. But these omitted columns are considered when assessing complexity (see Section 7). We use some additional shorthand in Table 4: the set of vintage prints {*MIB*}, {*STW*} will be tagged as ⟨*vintage*⟩, and we use the label ⟨*adult*⟩ for the set of sizes *{M, L, XL, XXL}*. The negation operator "¬" is used to negate a set, i.e. complement it with respect to the property domain. The negated form will be used when it is shorter for readability. The vintage imprint *MIB* will always be executed in imprint size *Fill* and the imprint *STW* in imprint size *Big*.

Table 4 Individualized T-shirt with vintage prints in c-tuples

Size	Color	Imprint	ImpColor	ImpSize
*	*	¬⟨vintage⟩	*	*
*	Black	MIB	White	Fill
⟨adult⟩	*	STW	Blue	Big

Still later, the sales department decides to enforce a constraint that the imprint color be distinguishable from the T-shirt color to avoid customer dissatisfaction. Table 5 lists the sales variants in seven c-tuples.

Table 5 Variants of the individualized T-shirt offered for sale

Size	Color	ImpColor	Imprint	ImpSize
*	¬{Black}	Black	¬⟨vintage⟩	*
*	¬{Blue}	Blue	¬⟨vintage⟩	*
*	¬{Red}	Red	¬⟨vintage⟩	*
*	¬{White}	White	¬⟨vintage⟩	*
*	*	Green	¬⟨vintage⟩	*
*	Black	White	MIB	Fill
⟨adult⟩	*	Blue	STW	Big

The sales constraint affects what is offered by sales, but has no direct implication for production. Production, on the other hand, is constrained to use the correct dye for a given fabric and color combination. This necessitates adding a tenth product property *Dye* with a suitable domain. The manufacturing constraint is given in Table 6, and the overall table of variants from the manufacturing perspective is given in 19 c-tuples in Table 7. We introduce the label ⟨*std*⟩ as shorthand for the standard colors {*Black*, *Blue*, *Red*, *White*}.

Table 6 Fabric / Color / Dye constraint

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 7 Manufactured variants of the individualized T-shirt

Fabric	Size	Color	ImpColor	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⟩	Green	Blue	STW	GRSD#2
¬{Cotton}	⟨adult⟩	Purple	Blue	STW	PUSD#4
¬{Cotton}	⟨adult⟩	Pink	Blue	STW	PISD#6
¬{Cotton}	⟨adult⟩	Yellow	Blue	STW	YSD#8

5. DATABASE QUERIES

One major point we make in this paper is that tables compressed to c-tuples (as well those compressed to a VDD, see Section 6) support the database queries relevant to product configuration. A product configurator must be able to filter a variant table to obtain those combinations/variants[3] that match given external (e.g. user) selections or exclusions of product features. The *result set* (RS) of a filtering query is again conceptually a variant table that may be further filtered and can itself be represented in compressed form. The external selections/exclusions can be formulated as a c-tuple. We refer to this c-tuple as the *query condition* (QC). Looking for all cotton T-shirts in non-adult sizes in Table 7 corresponds to filtering with the c-tuple (4) as the QC.

$$\langle Cotton, \neg\langle adult\rangle, *, *, *, *\rangle \qquad (4)$$

This yields an RS of around 46 million[4] variants, which might be subsequently further queried. To check whether a given variant is valid can be done using the r-tuple representing the variant (a c-tuple comprised of singleton sets) as a QC. If this yields the empty RS, the variant is not valid. Such a look up can also be used to identify the manufacturing variant of a T-shirt for a given sales variant. For the T-shirt, the manufacturing variant in the RS will additionally include the value for the property *Dye*.

To identify the still available product features after filtering, it must be possible to determine the resulting *domain restrictions*, the sets of values for each product property that occur in the RS of the filtering. The domain restrictions are again representable as a c-tuple. For example, filtering Table 7 with the QC (4) yields domain restrictions that can be formulated as the c-tuple:

$$\langle Cotton, \neg\langle adult\ \rangle, *, *, \neg\{\ STW\ \}, *\ \rangle \qquad (5)$$

which differs from the QC (4) only in the exclusion of the imprint *STW*. This domain restriction on the property *Imprint* can be directly applied in query conditions for other tables that reference it. This allows a straightforward implementation of a *local propagation* algorithm [15]: restrictions obtained in filtering a table are used to further filter other tables, until no further restrictions are possible[5].

The above queries can be supported in compressed format [14]. We summarize these queries using a functional notation. The argument to the queries is $\langle vtab\rangle$, a compressed table. $\langle RS\rangle$ denotes a result set of a filtering query. This is again a compressed table and may be further queried. $\langle QC\rangle$ denotes a c-tuple used as a query condition, and $\langle DR\rangle$ denotes a c-tuple of domain restrictions obtained from a table or result set.

$$\langle \mathrm{RS}\ \rangle \leftarrow \mathrm{filter}(\langle \mathrm{vtab}\ \rangle, \langle \mathrm{QC}\ \rangle) \qquad (6)$$
$$\langle \mathrm{DR}\ \rangle \leftarrow \mathrm{restrict}(\langle \mathrm{vtab}\ \rangle, \langle \mathrm{QC}\ \rangle) \qquad (7)$$

Where tables are representable in uncompressed relational form in a database, this can be queried using the *Structured Query Language* (SQL) [16]. A formulation of the QC as an SQL `WHERE` clause is given in (8)[6].

```
WHERE [NOT] ⟨v₁⟩ IN ⟨R₁⟩ AND [NOT]
⟨v₂⟩ IN ⟨R₂⟩… [NOT] ⟨v₁⟩ IN ⟨Rₖ⟩ ;          (8)
```

The SQL equivalent of queries for filtering (6) and restricting (7), which are the most relevant for configuration (cf. [14]), are given in (9) and (10), which return an RS (9) and a domain restriction (10).

```
SELECT * FROM ⟨RS⟩ WHERE ⟨QC⟩ ;     (9)
```

[3] To facilitate formulation, in the sequel we shall use *variant* as the generic term to refer to either a complete specification of a product variant or to a combination of a subset of its features represented as a row in a variant table.
[4] To be precise, 46,081,152 out of the total of 276,514,560 variants (see Table 9).

[5] Local propagation is one basic *constraint processing* algorithm that has wide-spread use in product configurators [15]. It is the central constraint processing algorithm of the SAP VC [19]. A detailed discussion of constraint processing is beyond the scope of this paper.
[6] Any reference to a column with a wildcard can be omitted from the `WHERE` clause.

```
SELECT DISTINCT ⟨$v_j$⟩ FROM ⟨$RS$⟩
         WHERE ⟨$QC$⟩;                (10)
```

6. ADVANCED COMPRESSION

Compression of variant tables to c-tuples is powerful and is a method of choice, because c-tuples have a standard representation in a spreadsheet and are supported in other configurator tools. Sometimes a variant table has a column with an identifying key or number unique to each row. In this case, the only possible c-tuple decomposition is as the list of r-tuples making up the original rows, i.e. compression using c-tuples is then not possible. Advanced compression to a *decision diagram* (DD) can go further. Our choice is a *variant decomposition diagram* (VDD), which was designed with table compression in mind[7]. We consider compression to VDDs a side topic here and do not go into details, for which we refer to [14]. However, we briefly explain the basic idea using the VDD depicted in Figure 2, which encodes Table 4[8]. Each node is labeled with a set of feature values for a product property and has two emanating links, *HI* and *LO*[9]:

- The *HI*-link of a node points to a node for another product property or to the terminal sink $\top$ (*true*),
- The *LO*-link points to a node pertaining to the same product property or to the terminal sink $\perp$ (*false*).

The VDD construction, which we envision here, is determined uniquely by specifying an ordering of the product properties [14]. The VDD in Figure 2 is constructed for the property ordering *Imprint*, *ImpColor*, *ImpSize*, *Color*, and *Size*. Different orders result in VDDs with different compression. We believe this ordering of the product properties may also be important for organizing the business (see Section 8). In Table 8 we give the number of VDD symbols (see Footnote 9) that result from three different orderings of the product properties for the sales variants in Table 5, extended to all nine product properties *Style*, *Fabric*, *Neck*, *Imprint*, *ImprintColor*, *Size*, *Color*, *ImprintSize*, and *CO$_2$ Offset*, listed in this order.

Table 8 VDD sizes of T-shirt sales variants in different column orders

Column order	VDD size
1, 2, 3, 4, 5, 6, 7, 8, 9	1105
4, 5, 7, 6, 1, 3, 2, 8, 9	1071
3, 2, 9, 1, 5, 7, 6, 8, 4	3108

A VDD entails a decomposition into c-tuples. Each node is labeled with a set of values for an associated product property. All nodes that can be reached from the root node via *LO*-links are starting points. Each path following *HI*-links from a starting node to the sink $\top$ defines a c-tuple. The c-tuple compression encoded in a VDD can be extracted by iterating over all paths.
The three c-tuples of Table 4 are directly visible in the VDD in Figure 2. Notice that the node labeled "`(Size, *)`" is shared between the first two c-tuples. It is the strength of VDDs to exploit commonalities between c-tuples, when they exist.
Compression to a VDD is more powerful than a simple decomposition to c-tuples regarding both space and performance savings (see Section 9) and lends itself to a more fine-grained measure of *complexity* (see Section 7).
In the case of an MC product model containing several variant tables, the property ordering is per variant table. Different variant tables may use different orders of product properties.

7. PRODUCT VARIANT COMPLEXITY

We consider the complexity of an MC product to refer to the cost its variability adds to a business. This 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 variability.
Product complexity can be approached from various angles [17]. One approach is to measure the complexity of the underlying product model in terms of the number of product properties and features, the number of constraints and rules, and the required sophistication of the modeling language. In this paper, we focus on modeling MC products with variant tables only[10]. We take it as an axiom, grounded in experience, that using tables wherever possible is not only a transparent, easy-to-understand modeling technique, but also alleviates a central business problem: to maintain variant data in a way that is transparently accessible to the affected business processes.
The complexity of a product model is an indicator of the cost of maintaining the model, but it may also allow an estimate of expected configurator performance. When all variants of an MC product are enumerable in a single table, product configuration reduces mainly to making queries of the form (6) and (7) (or their SQL equivalents (9) and (10)). In this simple, single-table

[7] Various common forms of DD can be mapped to each other. See [14] on the relationship of VDDs to *zero-suppressed binary decision diagrams* (ZDDs) and *multi-valued decision diagrams* (MDDs). See [9] on the relationship between *binary decision diagrams* (BDDs) and ZDDs.
[8] For brevity, properties Style, Fabric, Neck, and ImpSize have been omitted.
[9] For ease of depiction here, a VDD node is labeled with a set of values. The actual VDD may be organized to have its nodes labeled with only one value [14], which is our currently preferred operational form. The *size* of a VDD is defined to be the overall number of value references in its nodes, regardless of how it is organized internally.
[10] Modeling solely with variant tables is possible given our assumptions on the finiteness of the properties and their domains. In [23] the finiteness assumptions are relaxed while sticking to the tabular paradigm. For a more general approach to assessing product model complexity, c.f. [24].

model, both the maintenance cost and the performance of queries are directly related to the size of the compressed table[11].

Hence, we propose a complexity measure for an MC product based on compressibility of the set of its variants. We motivate our approach by discussing the *unconstrained* case and then extending it.

In the unconstrained case, modeling is reduced to defining the variant properties and maintaining their domains. As defined in (1) there are a total of S possible features, each represented by a symbol that can be associated with additional data needed for the business, such as *surcharges* (see Section 8). The unconstrained set of variants can be represented as one c-tuple using S value symbols or as a VDD of size S (see Footnote 9). The performance of querying a VDD can be guaranteed based on its size (see [14]), We define the complexity of an unconstrained set of variants as:

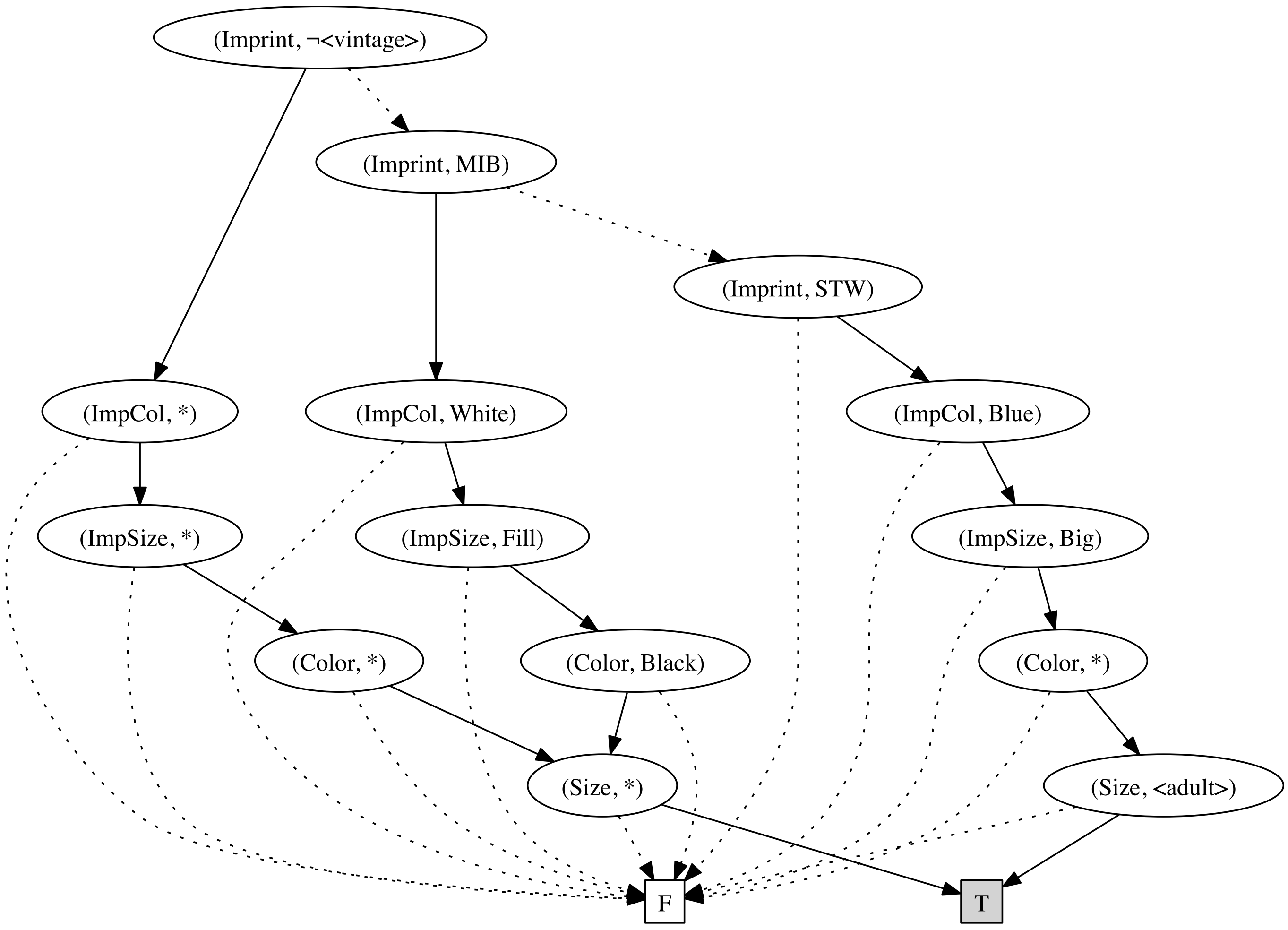

Figure 2 VDD with nodes labeled by value sets for Table 4
HI-links are represented by solid arrows, *LO*-links are represented by dotted arrows. T = true and F = false are terminal sinks

7.1 Variant complexity for an unconstrained set of variants

The simplest MC product model applies in the case that all product features are freely combinable without constraints. This case of an *unconstrained* product is rare in practice. It may be a feasible scenario when starting a business: In Section 4 we postulated that the T-shirt business first offered the individualized T-shirt in completely unconstrained form, without the *vintage* prints. The first c-tuple in Table 4 represents all the variants offered at that time.

$$u\_cmplx := c_0 S \qquad (11)$$

where c_0 is a constant that allows calibrating for a particular platform, so that a query to the VDD can be guaranteed to take at most u_cmplx milliseconds. We will set $c_0 = 1$ for simplicity of exposition and take $u\_cmplx = S$, except where otherwise noted in Section 9.3.

[11] We have already argued that there is no direct correlation between complexity and the total number of variants offered. The individualized T-shirt in Section 4 looks fairly simple, but has more than a quarter billion variants. Cf. [14] on the performance of queries for VDD compression and [8] on c-tuple compression.

7.2 Complexity of a variant decomposition

A decomposition of a set of variants to c-tuples effectively subdivides or *slices* the variants into unconstrained subsets, each with a complexity defined by the number of features that it mentions as in (11). If this decomposition is explicitly represented in simple compression as a list of c-tuples, the arguments about performance and space for the unconstrained case in Section 7.1 apply to each c-tuple and the complexity of the decomposition is just the sum of the complexities of the individual c-tuples making up the composition. This measure can be applied to any set of variants of an MC product, e.g. also to the variants in the result set of a filtering query. If there are M c-tuples with sizes $S_1, S_2, \dots, S_M$ in the decomposition of a set of variants, then the complexity of this decomposition can be expressed as:

$$c\_cmplx := c_0\,(S_1 + S_2 + \dots + S_M) \qquad (12)$$

Advanced compression is potentially more powerful than simple compression to c-tuples. The size of a VDD is defined as the number of value references made by its nodes (see footnote 9) and will generally be less than the sum of the sizes of all its c-tuples. Query performance can be guaranteed based on the VDD size [14]. Similar to c-tuple compression, we define the complexity of advanced compression as the VDD size, again adjusted by the calibrating factor c_0 as in Section 7.1. For a VDD of size n, we define the its complexity v_cmplx as:

$$v\_cmplx := c_0\, n \qquad (13)$$

We set $c_0 = 1$, except where otherwise noted in Section 9.3.

7.3 Complexity of a negative representation

We can think of a value assignment to k product properties as a point in a k-dimensional space. **Error! Reference source not found.** suggests a variant space of two dimensions. Each point in the plane represents a combination of *Color* and *Imprint*. The left part of the figure represents a *positive variant table*: points that represent *valid* combinations are *dark*; points for *invalid* combinations are white. In this figure, there is only one combination of imprint and color that is not valid, which is represented by the white square in the middle. The right part of the figure represents a *negative variant table*. Here, the depiction is the other way around: points that represent *invalid* combinations are *dark*, points for *valid* combinations are white. The invalid combination is represented by the dark square. The white outlined rectangles correspond to combinations that are valid and lie in the solution space of all possible combinations.

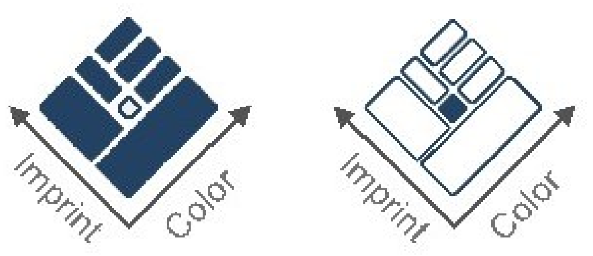

Figure 3 Positive (left) and negative (right) depiction of a variant relation

In the general k-dimensional space, the variants in a c-tuple form a cuboid. In the two-dimensional case this is a rectangle. The positive variant table on the left side of Figure 3 consists of six c-tuples: the six rectangles with a filled *dark* background. The negative variant table on the right side of Figure 3 consists of one c-tuple: the one point in the middle. The positive variant table implies the domains for the properties: only those values for a property that occur in a valid variant need to be considered in practice. This is not true for a negative variant table. However, when finite domains for all the product properties are known, then a list of all excluded variants is an alternate definition of the variants, because it is then possible to calculate the complement of the negative table with respect to the *solution space* of all combinations possible with the given domains. Moving from the positive representation on the left of Figure 3 to the negative one on the right, then means inverting the background fill of all the rectangles.

In the example, it is evident that the negative representation is more compact. However, investigations in [18] indicate that it does not make much difference for complexity in practice, whether the positive set of valid variants or the negative set of excluded variants is compressed to a VDD, and even in the extreme cases, where it does make a difference, both forms can be readily calculated and represented.

7.4 Complexity for a model comprised of several variant tables

In the absence of a manageable, easy-to-use tabular representation of the overall set of variants, a preferred way of modeling an MC products is as a set of constraints over the product properties and their domains. For the purposes of interactive configuration, local constraint propagation [15] is used to determine the domain restrictions that result from a query by the user to the combined set of variant tables. Local propagation is one basic *constraint processing* algorithm that has widespread use in product configurators. It is the central constraint processing algorithm of the SAP VC [19]. Performance of local propagation depends on the exact implementation, but is generally good in practice.

In the finite case, all constraints can be conceptually thought of as tables. The Renault Megane benchmark model we refer to in Section 9 is made up of 113 variant tables, some of them large. The underlying basic restriction operation (7) is very efficient on compressed tables. The performance of each query to a table can be guaranteed based on the complexity c_cmplx (12) or v_cmplx (13) applied to the table itself or to the reduced RS of previous filter queries. Thus, the simple compression in Section 3 and more specifically the advanced compression in Section 6 directly facilitate local propagation between tables.

Negative tables require special consideration for local propagation (see [13]). A straightforward approach would be to negate a negative table to obtain a positive one for processing.

8. BUSINESS UTILITY OF COMPRESSION

8.1 Business complexity for unconstrained T-shirt

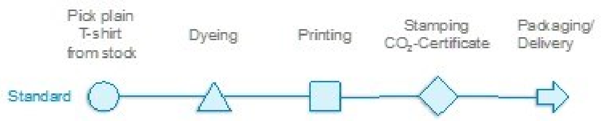

Figure 4 Production process of the simple personalized T-shirt

Figure 4 shows a possible production line for the *unconstrained* modern T-shirt used as a motivating example in Section 7.1. The business must set up for procuring/stocking T-shirts, dyeing them, imprinting them, stamping the appropriate *CO₂-Offset* label on T-shirts, as well as for the shipping and invoicing processes.

Some of these processes depend on the number of features offered and their business attributes. We may imagine that dyeing may have to be set up for each offered color individually, each at a different cost. And we may imagine that each of the 1000 non-vintage imprints may have its own different licensing cost and image size. The cost of stamping the CO₂-*Offset* certificate carries with it a fixed cost that may depend on the number of different labels offered. On the other hand, the cost associated with the processes of procuring T-shirts, of maintaining the textile printing itself, and the invoicing and delivery processes is fixed and does not depend on the number of features offered[12].

The unconstrained T-shirt example suggests that the effect of variability on the cost of setting up the business can be modeled as *fixed costs* on product properties as a whole and/or *surcharges* on individual product features. The cost for procurement/stocking is fixed and does not depend on variability, dyeing costs can be directly attributed to the individual available colors, and each imprint has its own individual cost (which might be zero). In addition, the property *Imprint* is further associated with the fixed cost of the printer. The stamping of *CO₂-Offset* certificates would also have a fixed cost attributable to the process.

So, for this example, business cost can be defined in a very similar manner to the unconstrained complexity c_cmplx (12), except that summing the surcharges and fixed costs replaces simply counting the occurring value symbols:

$$business\ cost = \sum_{\text{Fixed property costs}} F_j + \sum_{\text{Value surcharges}} w_{ij} \tag{14}$$

where F_j is the fixed costs associated with the j-th product property, and the w_{ij} is the surcharge associated with the i-th value for the j-th product property.

The effect on cost of adding/removing a feature is given by its surcharge w_{ij}. The effect on cost of adding/removing an entire product property is given by its fixed cost F_j, as well as any surcharges for its values.

8.2 Business complexity for the individualized T-shirt

Figure 5 shows a possible business setup for the modern individualized T-shirt with vintage prints. Table 4 lists all variants in three c-tuples. These c-tuples are mirrored in the VDD structure shown in Figure 2. The VDD uses the property ordering *Imprint*, *ImpColor*, *ImpSize*, *Color*, and *Size*, which suggests that the first decision on the imprint determines the production line 'Standard', 'Special 1' or 'Special 2' in Figure 5.

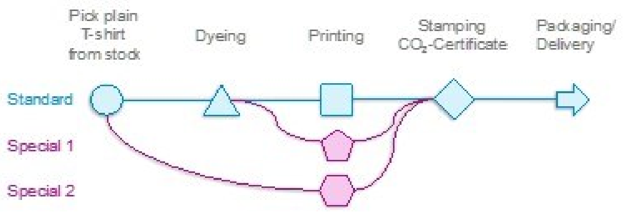

Figure 5 Three production lines of the personalized T-shirt

This example suggests that the business cost of variability can be assessed based on a VDD as in Figure 2. The cost of setting up the business is then given by summing the fixed costs and surcharges for all value references in the node labels (15):

$$business\ cost = \sum_{\text{Fixed property costs}} F_j + \sum_{\substack{\text{Surcharges for} \\ \text{VDD value references}}} w_{ij} \tag{15}$$

Adding or removing non-vintage prints will greatly affect the number of variants offered, but not the cost of the business. Removing a vintage print, on the other hand, will remove only a few thousand of the hundreds of millions of variants but will have a great effect on cost.

The investigation of how compression relates to business complexity is ongoing, but we think this example motivates following up on this idea.

[12] Realistically, the number of T-shirts held on stock may need to become larger as choices of *Style*, *Size*, *Fabric*, and *Neck* increase.

8.3 Business with indeterminate variants

Due to wide-spread use of relational representation in business, a product variant is classically defined as a value assignment to each of its product properties. This is neither ideal nor sufficient in practice. Some degree of indeterminism in a variant is needed when a variant is to be further specialized in a later business process (e.g., at the customer's site). For example, a pump may be sold with a connection that fits several different sizes of hoses. The end customer may have to make a manual adjustment for the particular hose they want to attach by cutting off a part of the provided connector. The pump being sold to the customer by the business is a variant of their MC "pump" product that allows further personalization at the customer's site. We observe that in general, variants that result as output of one business process may be refined or specialized in subsequent business processes.
Allowing a variant to be defined by a c-tuple is an approach to dealing with this. However, it means moving from a purely relational data representation to one allowing c-tuples.

9. RESULTS

The RM benchmark, which we make use of here, was published by Renault as a *constraint satisfaction problem* (CSP) some years ago as a benchmark for configurators and constraint solvers [10, 19]. It consists of 99 product properties and 113 constraints that can be represented as variant tables. We have constructed an overall variant table in compressed form by joining these 113 tables using the SALADD system [21] as a tool. This was done purely for purposes of comparing complexity of the Renault Megane as a product with our exemplary T-shirt products and for experiments with performance in Section 9.3. Assessing the utility of making such joins available as part of productive variant table management system is a topic of future work.

9.1 Compression of overall set of MC variants

In this section, we summarize the compression and complexity results for the case that all variants are represented in a single compressed table. Table 2, Table 4, Table 5, and Table 7 each represent such a case and pertain to fairly simple products with a multitude of variants. Additionally, we also include a table representing all the variants for the Renault Megane (RM) benchmark. Table 9 gives an overview of the relevant compression results for these tables:

Table 9 Complexity comparison between product variants

Variant Table	#variants	#c-tuples	c_cmplx	v_cmplx
Simple T-shirt	11	2	12	12
T-Shirt all	276,514,560	3	1100	1058
T-Shirt sales	241,954,560	7	5252	1105
T-shirt mfg	276,514,560	19	9230	1092
Renault Megane	2,673,852,735,568	895,972	14,417,790	120,217

c_cmplx: complexity after c-tuple compression
v_cmplx: complexity after VDD compression
mfg = manufacturing
T-shirt examples from top to bottom: Table 2,Table 4,Table 5, and Table 7

If we define the compression ratio of the variants separately for the representation as c-tuples and as a VDD as in (16) and (17)

$$\rho_c = 1 - \frac{\#c\_cmplx}{\#cells} \tag{16}$$

$$\rho_v = 1 - \frac{\#v\_cmplx}{\#cells} \tag{17}$$

then the compression of all the above tables is more than 99.99 %, except for the "simple T-shirt" in Table 2. The other T-shirts are compressed from a quarter of a billion rows to a small number of c-tuples. The complexity measures c_cmplx for c-tuples grows with the number of required c-tuples to around $10{,}000$ symbols. In advanced compression, the complexity measure v_cmplx is only about 1000 for these tables. The compression of the "T-shirt sales" (Table 5) has approximately one third as many c-tuples and half as many symbols as the larger table "T-shirt manufacturing" (Table 7), but the VDD size is about the same. These compressed sizes are very reasonable and compare with results from prior work [14, 18]. The Renault Megane has $2 * 10^{13}$ variants, compressible to almost one million c-tuples. Its c_cmplx measure is more than 14 million, the v_cmplx is only around 120 thousand. Interestingly, its v_cmplx measure is only about 100 times larger than the v_cmplx of the much simpler T-shirt variants.
These MC products span the spectrum of products reaching from fairly simple T-shirts to cars. For all of them it is seems feasible to deal with an overall table of variants, if desired.

9.2 Compression of individual variant tables

In this section, we present some empirical data we have collected for three sets of variant tables.
The three sets of variant tables are:

- The 113 individual constraints of the RM model as variant tables – labeled RM.

- The set of 238 variant tables which we used in our workshop paper on VDD compression [18][13] – labelled CWS15.
- A “random sample” of $20{,}576$ variant tables from 678 actual SAP product models[14] – labeled “Random Sample”.

Interestingly, 1407 of the tables in the “random sample” were already maintained in c-tuple format by customers, as opposed to $19{,}169$ relational variant tables. This shows that the c-tuple format is in use despite being discouraged as a “best practice” in [10], albeit only for a small percentage. Table 10 summarizes the span of table sizes for the three sets. The maximum and averages are given for the number of rows (*relational tuples*), the number of columns (*arity*), and the load/compression times from relational tables (*Time*). The statistics for “Random Sample” include more extreme values. The average number of columns is uniformly between four and five. RM has by far the largest average table size.

Table 10 Summary of compression of different variant table sets

	#Rows		**#Columns**		**Compression Time (ms)**	
Table set	Max	Avg	Max	Avg	Max	Avg
RM	48271	1724.0	10	4.91	600	16.7
CWS15	21,372	238.5	16	4.29	598	9.8
Random Sample	1,152,832	429.9	71	4.15	516	2.9

For space reasons, we limit the presentation mainly to the relationship between the uncompressed table size, embodied by the *number of cells*, and the VDD size, embodied by the VDD complexity measure v_cmplx. Figure 6 (A – C) depicts this for the three data sets, respectively. Values have been logarithmically transformed, and the 45-degree line through the origin (dashed) and the linear regression line (solid) have been drawn in all figures[15].
The 45-degree line through the origin marks the line where no compression takes place, i.e. any tables below this line are compressed. The flatter the slope of the linear regression line, the better the overall compression of the set. The slope is very similar for CWS15 and “Random Sample”. The RM set behaves noticeably better. RM was probably the only model formulated entirely using tabular constraints and we might conjecture that these variant tables may have received more thought and attention.
Lastly, Figure 6D compares the results for VDD compression against c-tuple compression of the “Random Sample”. It shows that VDD compression is stronger than c-tuple compression over this sample. In particular, there are some large tables where the additional compression is significant. These are the cases where VDD compression shows a real benefit.

9.3 Selected performance results

We make use of the Renault Megane (RM) benchmark to present performance results. As published, there are 99 product properties and 113 variant tables. We have also constructed a join of the 113 tables to a single overall variant table of all RM variants [21]. In both cases, we have added a table for the entire solution space – the set of all possible combinations given the finite property domains – which does not add a constraint or affect the solution. The purpose of the solution space table is that it allows tracking the overall effect of the filtering query together with local propagation between the tables (see Footnote 5). We have identified a sequence of 30 feature selections that together result in a unique solution.
Two test sequences are performed separately with 100 repetitions of the same configuration for each of the two test cases:

- The 113 individual variant tables with the solution space table.
- The one joined overall table with the solution space table.

A configuration consists of selecting 30 features from 99 product properties[16] and is identical in both cases. The test set-up mimics what would happen in an interactive configuration process:

- The variant tables are loaded once for repeated execution of the test configuration.
- An initial local propagation is performed on loading to eliminate any choices that can never be chosen. This is the *initial state*.

[13] This set collects the variant tables of four working product models for different products, each not modeled exclusively using tables. Switching the table filtering to use VDDs when configuring these products with the *SAP Internet Pricing & Configurator* (SAP IPC) product configurator (see [5]) resulted in a noticeable performance gain [18].
[14] Since product data is highly confidential, it is not easy to obtain real data to work with. The $20{,}576$ tables were compressed by *eSpline* (http://espline.com), a company specializing in software tools to supplement and improve the modeling experience with configurator applications, particularly for SAP VC and SAP IPC customers. eSpline compressed all tables from its own site by calling our compression API over the web (https://www.vbase18.com). eSpline obfuscated the data and only returned the analysis results to us. We never saw the tables themselves. Only the compression analysis was returned to us. We do not know their purpose, quality, or state.
[15] The statistical analyses presented here are derived using the R package. The $\log()$ function in R calculates the natural logarithm.
[16] The sequence of 30 features from the product properties (vp) is: $vp_1 = D64$, $vp_3 = MD$, $vp_5 = ALLE$, $vp_{66} = JANTOL$, $vp_{80} = CRIT4X15KI$, $vp_{90} = ANTID$, $vp_{40} = EMBPIL$, $vp_{15} = PBCH$, $vp_{29} = DRA$, $vp_{79} = SSABCO$, $vp_{85} = CDCOF$, $vp_{12} = GALERI$, $vp_{13} = CHAUFO$, $vp_{11} = ABS$, $vp_{49} = Autre310$, $vp_{47} = LVAVEL$, $vp_{57} = SAILAR$, $vp_{23} = PROJAB$, $vp_{14} = TO$, $vp_{21} = RETROE$, $vp_{78} = FIPOU$, $vp_{52} = CORHLO$, $vp_{51} = SGACHA$, $vp_{34} = Autre167$, $vp_{60} = SLAVPH$, $vp_{22} = REGSIT$, $vp_{25} = CUSFIX$, $vp_{65} = PLAFT$, $vp_{63} = VOLRH$, $vp_{41} = PNERFL$

- A selection of the first feature is performed yielding a filtered result set and domain restrictions for the product properties.
- The number of variants left in the solution space after local propagation of the selection is printed to the terminal (*stdout*) to simulate user interaction.
- The next feature in the list is selected as a filtering query on the result set of the previous query yielding a further restriction. The number of variants left in the solution space after local propagation is again printed to the terminal (*stdout*).
- Selections are made until either a unique remaining variant is left, or an empty result set signaling *inconsistency* is encountered. The test sequence is set up in a way that a unique solution is found after making all 30 selections.
- The configuration state is reset to the *initial state*.

The overall time of each test sequence consisting of 100 configurations is measured using the Java timer which outputs nanoseconds. The time for loading and compressing the table is measured separately. We choose the same hardware platform[17] as was used to obtain the results in the workshop contribution [18] to have the results comparable. The measurements in [18] were based on a prototype written entirely in Java. Our current implementation is largely based on native code, called from Java using a JNI interface. The results are given for 100 repetitions in milliseconds (ms) in Table 11. The load times are given in Table 12.

[17] An Apple Mac mini with 2.5 GHz Intel Core i5 and 8GB memory.

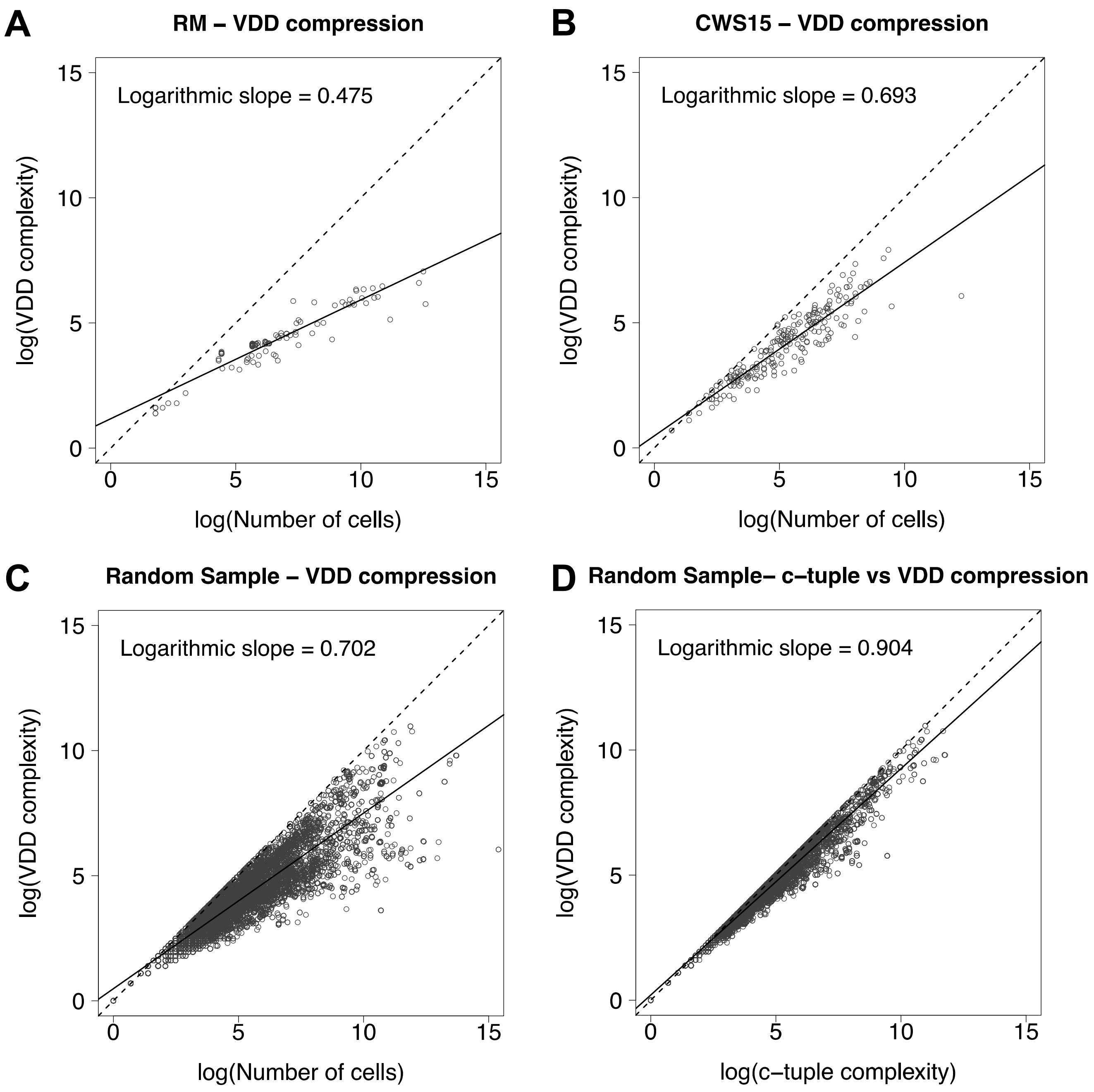

Figure 6 The effect of compression
Each dot represents a table which was compressed using VDD or c-tuple compression. The effect of compression is illustrated by comparing the complexity measure v_cmplx to the original size (number of cells) or to c_cmplx. The solid line represents the linear regression line. The 45-degree dashed line through the origin shows the points of no compression.
(A) Effect of VDD compression on the Renault Megane model (RM).
(B) Effect of VDD compression on a set of variant tables from four working product models (CWS15).
(C) Effect of VDD compression on over 20,000 variant tables from 680 SAP product models (random sample).
(D) Comparison between c-tuple and VDD compression on the same tables used in (C) (random sample).

Table 11 Comparing the configuration performance for the RM benchmark using individual tabular constraints (with local propagation) versus one joined tabular constraint

Test case	Execution time (ms)
113 tabular constraints	1727
Joined tabular constraint	8221

It was surprising to us that the local propagation between the 113 tables and the solution space was about four times faster than querying the joined table and propagating only to the solution space, which cannot produce any additional restrictions and does not itself cause further propagation.

Table 12 Loading times for the RM benchmark

Source format	Load time (ms)
113 relational tables	2905
113 tables from c-tuple format	466
113 tables from internal format	286
One joined table (internal format)	103

10. DISCUSSION

Variant tables that record valid combinations of product features are an established popular element of MC product models. The compelling advantages of using tables are:

- They are universally understood.
- They are an accepted best modeling practice.
- Content can be maintained and versioned independently of the product model.

However, their use has been severely limited by their lack of scalability. In this article, we showed that table compression techniques can overcome this limitation. We argued that variant tables should compress well due to regularities in the underlying products. Our results corroborated this (Section 9). We showed by examples ranging from T-shirts to cars that it is even often possible to represent the overall enormous number of variants as a single table in compressed form.

Nevertheless, the random sample of SAP VC variant tables presented in Section 9 shows that c-tuple compression has not yet been used as much as would be expected. Only 7% of the tables in the sample were in c-tuple format (rows with multiple values in the table cells). We conjecture that this has two causes:

- Where compression becomes essential, it is no longer possible to expand the table to uncompressed relational form due to the resulting size. However, a relational format is often required for compatibility with other existing processes.
- Suitable tools for managing tables in compressed form do not yet exist, i.e. c-tuples are not easy to read for humans without additional tools.

A misperception about the potential of compression in combination with the above problems may have led to the fact that c-tuple compression is discouraged as a best modeling practice for the SAP VC [10]. A major aim of our work was to set right this misperception: we showed that conventional database functionality can be smoothly extended to the compression formats we have discussed (c-tuple and variant decomposition diagram (VDD) (Section 6)). Operationally, queries on substantially compressed tables are much faster than queries on their relational uncompressed counterparts. What remains is to enhance tools for table maintenance directly in compressed format, which is ongoing work.

A great benefit of c-tuples is that they can be stored and exchanged by using established, transparent, and non-proprietary formats that are readily understood, e.g. csv files. An import and export of SAP VC variant tables in c-tuple format to csv files is described in [22]. C-tuples in csv files come close to being an accepted standard. Standards are important for business success. For this reason, we consider the c-tuple format to be the current compression format of choice. Although advanced compression to a variant decomposition diagram (VDD) is stronger, there is not yet a public exchange format for this. While we have been able to load the MDD (see Footnote 7) compiled from the 113 RM tables compiled from the SALADD system [21], there is to our knowledge no currently published format for storing and exchanging decision diagrams.

A variant table functions naturally as a constraint in a product model. Enabling larger tables by utilizing compression techniques allows more constraints to be expressed in tabular form with all the advantages associated with that.

Another key observation we made is that compression size can serve as a measure of product complexity, useful both to assess configurator performance and the business cost incurred by the variability offered with a product. We argued in Section 8 that our approach may offer a more concise approach to correlating the cost of adding or removing product features than simply trying to curb the overall number of variants.

Using the example of the T-shirt, we discussed in Section 8 that VDD compression may also be useful in analyzing business costs. Our construction of a VDD depends on the order in which the product properties are considered. This ordering might have a relationship with the order in which decisions needed to be made in producing the underlying product. In other words, the business set-up may prescribe an ordering of the variant properties for constructing an associated VDD. Vice versa, efficient compression may provide insight on how production steps might be ordered. These ideas are speculative and require verification in the field. This is a topic for further investigation.

The examples we used throughout this paper suggest that it is often possible to consider an overall table of all variants for an MC product. The advantages and disadvantages of such a table versus modeling with a plurality of constraints are not yet clear. It can be seen as a clear advantage that it is easy for any business process to query the overall table with normal SQL-like queries. In contrast, if the product model is made up of a set of constraints (like the 113 RM variant tables), a tool such as a configurator must be used to facilitate queries to the model. We have remarked that local propagation between tabular constraints can be supported very efficiently using the compressed format.

Contrary to our expectation, the performance measurements in Section 9.3 indicate that local propagation among the 113 RM variant tables turned out to be faster than queries to the overall large table (although that was also very fast). Since we cannot assess the quality of the overall table used in the measurement in Section 9.3 (it may be possible to achieve better compression), we cannot draw general conclusions from this, except to observe that local propagation should not be dismissed as an inferior method out of hand.

The problem of joining individual variant tables to an overall variant table is beyond the scope of this paper. It is a topic of systems that compile suitable product models to some form of decision diagram. It would be desirable to have a published format to exchange such compilation results, i.e. it would be interesting to see if other decision diagrams beside SALDD MDDs can be imported as a VDD.

11. CONCLUSIONS AND FUTURE RESEARCH

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 techniques discussed are available now for practical use in business.

We propose several levels of empowerment of variant tables:

1. Obtaining the complexity measure associated with compression size for a set of variants or an individual variant table may have a value of its own for a business.
2. Using tools that use compression internally can provide a noticeably faster interaction for anyone working with existing large variant tables.
3. Using variant tables in a c-tuple format where this is supported allows incorporating much larger tables in existing product modeling projects.
4. Filtering queries to variant tables in existing configurators can be modified to use a compressed format where available. A noticeable performance improvement for the SAP IPC (SAP sales configurator, see [5]) achieved by this is reported in [18]. Configurators that already make use of c-tuples, such as the SAP VC, benefit directly by compressing tables to c-tuples[18].

The complexity measure we introduce for variant tables is based on their compressibility and may provide important insights for an MC business.

We identify three main areas for future research:

- Developing tools for maintaining tables in compressed format.
- Assessing the utility of having an overall joined table versus multiple variant tables as constraints for a productive variant table management system.
- Continuing with the business complexity approach introduced in this paper.

[18] We have recently demonstrated enhanced performance with the SAP VC by utilizing a c-tuple format [22].

12. REFERENCES

[1] Pine II, J. B. (1999), *Mass Customization: New Frontiers in Business Competition*, Reprint. Harvard Business Press.

[2] Piller, F. (2006), *Mass Customization*, 4th ed. Wiesbaden: Deutscher Universitäts-Verlag.

[3] Forza, C. and Salvador, F. (2006), *Product Information Management for Mass Customization*, 1st ed. Palgrave Macmillan UK.

[4] Forza, C. and Salvador, F. (2008), "*Application support to product variety managemen*", International Journal of Production Research, Vol. 46, No. 3, pp. 817–836.

[5] Blumöhr, U., Münch, M. and Ukalovic, M. (2012), *Variant Configuration with SAP*, Second Ed. SAP Press, Galileo Press.

[6] Haug, A., Hvam, L. and Mortensen, N. H. (2012), "*Definition and evaluation of product configurator development strategies*", Computers in Industry, Vol. 63, No. 5, pp. 471–481.

[7] Shafiee, S., Felfernig, A., Hvam, L., Piroozfar, P. and Forza, C. (2018), "*Cost Benefit Analysis in Product Configuration Systems*", in *Proceedings of the 20th Configuration Workshop*, Vol. 2220, pp. 37–40.

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

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

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

[11] Amilhastre, J., Fargièr, H. and Marquis, P. (2002), "*Consistency restoration and explanations in dynamic CSPs - Application to configuration,*" Artificial Intelligence, Vol. 135, No. 1–2, pp. 199–234.

[12] Hounshell, D. (1984), *From the American System to Mass Production, 1800-1932: The Development of Manufacturing Technology in the United States*, Baltimore, Maryland: Johns Hopkins University Press.

[13] Haag, A. (2015), "*Arc consistency with negative variant tables,*" in Proceedings of the 17th International Configuration Workshop, Vienna, Austria, September 10-11, 2015, pp. 81–87.

[14] Haag, A. (2017), "*Managing variants of a personalized product: Practical compression and fast evaluation of variant tables,*" Journal of Intelligent Information Systems, Vol. 49, No. 1, pp. 59–86.

[15] Bessiere, C. (2006), "*Constraint Propagation,*" in Handbook of Constraint Programming, 1st ed., Rossi, F., van Beek, P. and Walsh, T. Eds., Elsevier.

[16] Wikipedia "*SQL.*" [Online]. Available: https://en.wikipedia.org/wiki/SQL. [Accessed: 07-Jun-2018].

[17] Trattner, A., Hvam, L., Forza, C. and Herbert-Hansen, Z.

N. L. (2019), "*Product complexity and operational performance: A systematic literature review*", CIRP Journal of Manufacturing Science and Technology, in press.

[18] Haag, A. (2015), "*Column Oriented Compilation of Variant Tables,*" in Proceedings of the 17th International Configuration Workshop, Vienna, Austria, September 10-11, 2015, pp. 89–96.

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

[20] Subbarayan, S. (2004), "*CLib: configuration benchmarks library.*" [Online]. Available: https://itu.dk/research/cla/externals/clib/. [Accessed: 03-Jun-2019].

[21] Schmidt, N. (2015), "*SALADD, compilateur SLDD.*" [Online]. Available: https://github.com/SchmidtNicolas/SALADD. [Accessed: 12-Feb-2018].

[22] Haag, A. and Faure, C. (2018), "*Polishing a raw diamond,*" Configuration Workgroup, Berlin, Germany, April 2018. [Online]. Available: https://www.configuration-workgroup.com/files/Doc%20Share/Public%20Docs/01%20-%20Conference%20Docs/2018%20April%20Berlin%20Conference/Day%20-%202%20Tuesday%20Multi%20Track%20Sessions/Track%203/02%20polishing%20a%20raw%20diamond%20-%20Haag;%20Faure.pdf. [Accessed: 09-Jun-2019].

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

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