Generation of Rule-Based Variance Schemes Towards a Data-Driven Development of High-Variant Product Portfolios

Thorsten Schmidt[1], Steffen Marbach[1], Frank Mantwill[1]

[1]Helmut Schmidt University, Hamburg, Germany

Abstract: The management of product portfolios with high variance is complex due to the underlying constraints as well as prone to errors due to uncertainty and dynamics in the historical data. The documentation of these product portfolios is usually done in so called rule-based variance schemes, which are specified in Conjunctive Normal Form (CNF). For better planning, feasibility assessments and simulation of mentioned product portfolios as well as for the generation of a synthetic reference dataset for downstream analytic processes and training of Artificial Intelligence applications, this paper considers the development of a CNF-Generator for the creation of rule-based variance schemes as a contribution to data-driven product development. In this paper, particular consideration is given to satisfiability, reasoning, model counting and plausibility of the generated data. The implementation and evaluation are embedded in an use case of the mass customizable variant configuration being found in the automotive industry.

Keywords: Model Checking and Counting, Data-Driven Product Design, Rule-based Product Configuration, CNF-Generator, Automotive Industry

1 Introduction

The German automotive industry is a highly competitive environment in which manufacturers (and OEMs) obtain a considerable competitive advantage by offering configurable product variants to meet customer-specific requirements in mass customization (Tseng et al., 2017). The challenge is to achieve an optimum balance between variant-induced complexity costs and the customer's increased willingness to pay for a customized vehicle, which puts the car manufacturer under cost pressure to evaluate its product portfolio against the background of variant management. At the same time, the automotive industry is competing with international competitors to maintain quality standards and ensure safety. In addition, it is becoming increasingly necessary to guarantee shorter development cycles for more complex products, which is only possible through modelling dependencies and the collaboration of interdisciplinary teams. The German automotive industry is used as an exemplary use case because it relies on rule-based product encoding, is highly complex and at the same time based on a multi-generation product development concept (Albers et al., 2015). This context allows and encourages the use of data from predecessor vehicles as well as the use of decision support systems based on a generic product description. Due to stricter legal regulations and rules for ensuring functionalities and authorized homologation of variant-specific vehicles, transparency, governance and traceability of supply chains are also a factor in today's fast moving and increasingly sustainable environment.

Developing such complex products under dynamic processes, limited information leads to suboptimal decision-making, especially in the early stages of product planning and conceptual design. An algorithmic decision support system can provide hybrid assistance for these steps by supplementing experience with data-driven tools, leading to better decision-making on complex questions (Marbach et al., 2023). Data-driven decision support can be used along the entire product development process, for example according to VDI standard 2221 in the planning, conceptualization, design and elaboration phases (VDI 2221, 2019). Due to the high influence of frontloading in the early phases (Ehrlenspiel et al., 2014), product influencing is best suitable at an early stage when there is the greatest uncertainty yet the greatest flexibility in design decisions. For this reason, this paper focuses in particular on the early phase of product development when the system architecture is determined for example by modular design decisions.

The scope of this research includes two use cases: Firstly, the introduction of a new product or new product series for which there is no historical data yet, but which should be configurable according to a similar scheme as previous multi-generation products. Secondly, the generation of a synthetic reference dataset for simulation and evaluation purposes of downstream data-driven methods such as training of Artificial Intelligence applications like Association Rule Mining using the Apriori algorithm (Schmidt et al., 2022). For both cases of strategic product planning (especially new products), an approach will be developed that uses the existing syntax to generate a synthetic variance scheme due to the absence of and compensation for historical data (see Figure 1). A variance scheme is an implicit description of variant-rich products which are too numerous to be described explicitly. These variance schemes require to have the same notation, comparable complexity and behavior regarding manipulation and abstraction to used concepts. A synthetic variance scheme is not subject to operational dynamic changes and inconsistencies and is stable over time or can even be dynamically adapted to changing conditions for simulation, model building, testing and planning purposes. Synthesized variance schemes are comparable to existing schemes in the German automotive industry, which have been used and examined in the past (Demke et al., 2021).

2 Research Problem and Research Questions

The addressed research problem comprises the generation of synthetic variance schemes according to the requirements of product development as a contribution towards data-driven decision support. First, the challenges of design in the early phase of product development in dealing with uncertainty, predictability and complexity are considered. Furthermore, the resulting requirements for such a variance scheme are compiled in a common syntax. Subsequently, an algorithm is developed that meets the requirements and is able to generate variance schemes that fulfill satisfiability in individual instances and consistency overall. In addition, the considered variance schemes must be of comparable size and contain a comparable structure of constraints, which simulates the behavior of the use case as accurately as possible.

The goal of this contribution is a profound understanding of the use of algorithms in early-stage product development and a functional and powerful run-time optimized algorithm for downstream analysis, strategic planning, simulation and research. Higher-level research goals that are encouraged are independent data availability in the form of synthetic data, the possibility of targeted manipulation of this data to analyze dependencies in highly complex rule-based variance schemes and a reference dataset without the operational dynamic changes and inconsistencies of historical data. Figure 1 illustrates the possibilities of using a generic product description to develop, train and test algorithmic decision support as well as supporting designers and product engineers modelling variants in the early-stage system architecture. This assistance can be used for both new products and new product generations with insufficient information. It is just as indispensable for benchmarking different approaches against a reference of an entire product line including potential training of Artificial Intelligence applications.

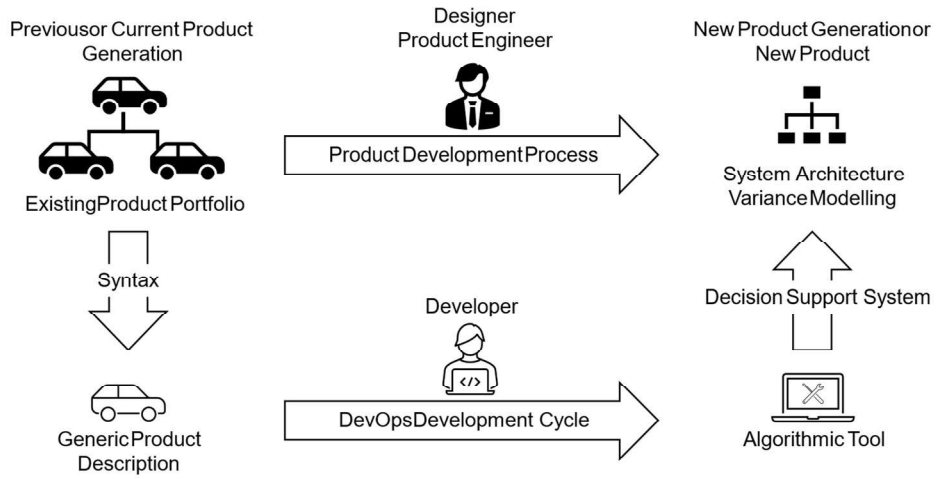

Figure 1. Systematic application of a generic product description for a developer and a decision support system for designer

The algorithmic decision support should be adjustable by entering parameters previously identified as important and thus cover a wide spectrum of applications. Such variance schemes are publicly available only to a very limited extent (Prokls, n.d.). Some of the few schemes that can be found have no or a very low number of solutions (FSU, n.d.) and are therefore not suitable for the described use case of highly variant product portfolios. A generator for this type of scheme could not be found at all and is provided open source as part of this research project via GitHub repository (CNF-Generator, n.d.).

The following three research questions are derived from this research problem:

RQ1) How can the product description of highly variant product portfolios be generalized in the form of a generic syntax?

RQ2) How to design an algorithm for generating a specific description of a rule-based variance scheme with a predefined syntax and adjustable parameters?

RQ3) To what extent can an algorithm-based decision support system assist in the early stages of data-driven product development?

To answer these research questions, the paper is organized into seven sections. After an introduction to the relevant background and related work in the following chapter, the used methodology is presented. The developed concept is then explained and evaluated with the help of the results obtained. Finally, the results are summarized and discussed and the need for further research is addressed in the outlook.

3 Background and Related Work

3.1 Product documentation and -encryption in the automotive industry

In order to understand data-driven product development, it is important to first understand the product documentation. In automotive manufacturing, a vehicle is uniquely described and encrypted based on its features. A customer selects the desired features from a catalog of feature options. Optional features are grouped together in the form of a feature family, from which exactly one option is allowed to be selected (this is referred to as “family concept”). In addition, the features are subject to constraints, sometimes referred to as "rules", which can be of technical, sales or marketing nature (Herlyn, 1990).

Feature encryption is performed using one-hot encoding, in which the typically three-digit feature codes (primary numbers) are converted into an efficient and compact binary matrix notation (allocation matrix) (Schmidt et al., 2022). An excerpt of such a feature catalog with two feature families and the associated one-hot encoding is presented in Table 1. In this example, the customer gets to choose one of two options for the partition wall (TRW), either without partition (3CA) or with a mesh partition (3CX). The chosen feature is encoded in the first or second position of binary allocation matrix.

Table 1. Excerpt of feature catalog according to Eisenhart (2002) and translated values in one-hot encoded allocation matrix

Options from the features catalog			One-hot encoding			
Feature Family	Feature Value	Description	3CA	3CX	3GA	3GN
TRW		Partition wall				
	3CA	Without partition	1	0	0	0
	3CX	Mesh partition	0	1	0	0
LBH		Loading floor				
	3GA	Without loading floor	0	0	1	0
	3GN	Variable loading floor	0	0	0	1

The aforementioned constraints are noted in the form of restrictions and prohibitions, likewise using their three-digit feature codes. For a machine-readable notation, the rules are then converted into a Boolean notation by linking them with the Boolean expressions AND (∧), OR (∨), NOT (¬). Later, the one-hot encoded notation of the features is combined with the Boolean notation of the constraints to form a unified notation in which the number of the respective feature is referenced which is then first OR'ed and then combined in an AND with all other ORs. This notation is known as Conjunctive Normal Form (CNF) and is presented in more detail in section 3.2. On the other hand, there is the Disjunctive Normal Form (DNF) which can be written as an OR of ANDs and is used as well in section 3.2. Table 2 lists an excerpt of the constraints with their corresponding Boolean notation. In this example, the choice of the body shape notchback (K8B) preselects the option without partition and therefore prohibit to configure the mesh partition from Table 1.

Table 2. Excerpt of constraints translated and according to Eisenhart (2002)

Restriction	Description	Boolean notation
K8B Z 3CA	Notchback forces without partition	K8B → 3CA
3GN Z K8D/K8M	Variable loading floor forces Golf Variant or Bora Variant	3GN → K8D ∨ K8M
3GN Z 1X0	Variable loading floor forces front-wheel drive	3GN → 1X0

To derive a scope of components for an explicitly described vehicle, a 150% bill of material (BOM) is then broken down into a vehicle configuration-specific 100% BOM based on the feature encoding. This is done by documenting the installation condition in the form of a so-called "part validity" in the rule-based complex BOM used (Herlyn, 1990; Frischen et al., 2019). Table 3 shows an excerpt of a complex BOM with exemplary part numbers and their part validity. Part number 1J0.343.132.A for example will only be installed if the configuration has chosen features 3CA ∧ 1X0 ∧ 3GA ∧ 7B2 ∧ (K8D ∨ K8M).

Table 3. Excerpt of a complex bill-of-material according to Eisenhart (2002) and Herlyn (1990)

Part number	Description	Part validity	Quantity
1J0.343.132.A	Load compartment trim left	3CA+1X0+3GA+7B2+K8D/K8M	1
1J0.343.132.B	Load compartment trim left	3CA+1X0+3GN+7B2+K8D/K8M	1
1J0.343.132.J	Load compartment trim left	3CX+1X0+3GA+7B0+K8D	1
1J0.343.132.K	Load compartment trim left	3CX+1X0+3GN+7B0+K8D	1

The explained product description through feature encoding with consideration of constraints for the formation of highly variant product portfolios is also known as a rule-based variance scheme. Many of the premium German automotive industry employ this type of scheme from product planning all the way to production and after sales (Zagel, 2006).

In the field, a rule-based variance scheme of this type is often subject to operational dynamic changes, which makes it difficult to use as a basis for data-driven approaches. On the one hand, using such a variance scheme results in changes to both the feature catalog and the underlying constraints over the course of the life cycle, which is why the variance scheme reacts dynamically. These changes can relate to the addition or removal of a feature, an entire feature family, a constraint or a versioned part number, which means that the previous encoding must be extensively adapted. Such planned changes are assigned a change date by means of a revision key. From the perspective of product data evaluation as well as product planning, these dynamics prove to be extremely obstructive. For this reason, a synthetic variance scheme is to be used for the basic validation of data-driven approaches as well as for the targeted planning of changes, which do not require these changes in the considered period but can be added later in a targeted effort. However, two of the observed consequences are taken into account in the purpose of accurate modeling: Firstly, the family concept, which allows features to be combined into groups of different sizes, from which only one option can be selected at a time. Secondly, the use of features which are declared as always true and consequently features that are always false due to a boundary constraint. The concept of features that are either always true or always false corresponds to a redundancy, which for various reasons has been observed occasionally in the historical data and is considered characteristic in the distribution of installation rates of past configurations. The consideration of this type of variable supports the assessment of consistency in the evaluation process later on.

3.2 Automated Model Checking and Counting using SAT- and #SAT-Solvers

The application of rule-based variance schemes enables customer-oriented mass customization, but also requires appropriate tools to answer complex questions. With a large number of constraints that can be difficult to comprehend, the first question that must be considered is whether the given variance scheme has a solution at all, i.e. whether it can be fulfilled or not ((SAT)isfiability) (Biere et al., 2009). This problem is better known as Boolean satisfiability problem (Davis et al., 1962) or Constraint Solving Problem (CSP) (Petrie, 2012). The Boolean Satisfiability problem (SAT) is about determining a satisfying variable assignment for a Boolean function or determining that there is none (Biere et al., 2009). All Boolean functions can be reformulated and written in the CNF format. The advantage of CNF over other formulations is that in this form, each individual clause must be satisfied due to the conjunction to satisfy the function as a whole (Moskewicz et al., 2001). If there is more than one possible solution, the next question is how many solutions there are (#SAT) (Valiant, 1979). Furthermore, the question must be answered as to whether the variance scheme is plausible and consistent, i.e. does not contradict itself. The question of plausibility is answered for example by the naive consideration of identities. Identities are two variables in which one requires the existence of the other and conversely. The comparison of such simple plausibilities increases understanding, but the satisfiability is already ensured by the use of a respective solver. The question of consistency can only be answered by analyzing a whole set of configurations that are based on a given CNF. Answering these questions in information technology form is the subject of Automated Reasoning, Model Checking and Model Counting.

Efficient solvers are used to provide answers to the above questions, most of which expect an input in the form of a CNF. For this purpose, the previously one-hot encrypted features and Boolean constraints are first converted into a CNF. At this point it should be mentioned that in the field of information technology and computer science the terms literals and clauses are used instead of features and constraints. As mentioned above, the CNF consists of a conjunction of logical ANDs of one or more clauses, which consist of the logical OR of one or more literals (Moskewicz et al., 2001). Literals are variables, in this case the encrypted features, clauses are the now reformulated constraints of the variance scheme. The formulation of the CNF in general notation over the literals l_{ji} with $j = 1, \dots, r$ clauses and $i = 1, \dots, s$ literals in the clause j is shown in Equation 1. Each l_{ji} can be positive or negated ($\neg$):

$$CNF - Formulation: \quad \bigwedge_{j=1}^{r} \bigvee_{i=1}^{s} (\neg)\, l_{ji} \tag{1}$$

In the context of this paper, Sat4J has been used as SAT solver (SAT4J, n.d.). SAT4J is a JAVA implementation of Eén and Sörensson's MiniSAT (Eén and Sörensson, 2003). This solver also inputs a CNF and outputs whether there is a satisfiable assignment for this equation or not. In addition, it is possible to perform a simple check to see whether a satisfiable assignment still exists by adding a further clause. As #SAT solvers (also known as propositional Model Counting solvers), both the c2d solver under Windows (C2D, n.d.) and, for testing purposes, the sharpsat solver under Linux (Sharpsat, n.d.) have been used. The use of SMT solvers is not considered in the context of this elaboration as they do not offer any advantage against the background of questions raised.

All solvers in use process the CNF in a specific DIMACS format. The DIMACS format follows a strict syntax. Equation 2 shows the initialization of a CNF in DIMACS format by the following header declaration where n is the number of CNF variables or literals and m is the number of CNF clauses (Darwiche, 2005). This declaration must appear before any clause declaration:

Header declaration of CNF in DIMACS: $p\ cnf\ n\ m$ (2)

Clauses are specified in the CNF on a separate line each following the format in Equation 3. Each variable l_x corresponds to a numerical index from the one hot encryption. The index x of the used variables start from 1, therefore the set of all variable indices corresponding to encrypted features in this use case is $x = 1, \dots, n$. The variable l_k represents the last literal in the current clause from $1, \dots, m$. $[-]$ is DIMACS notation of a negation of the following literal ($\neg$). If the $[-]$ is not set, it is a positive literal. The line of the clause specification always terminates with a *0*, which is therefore the separator for a new line and consequently the next clause (Darwiche, 2005):

Clause specification in DIMACS: $[-]l_1\ [-]l_2 \dots [-]l_k\ 0$ (3)

There is a comment command for noting and commenting on parameters used, project-specific identifiers or any need for further clarification. In the CNF, a comment is initialized with a preceding c. Equation 4 shows the format of an arbitrary comment (Darwiche, 2005). Comments lines may appear anywhere in the file and are ignored by solvers when evaluating a CNF:

Comment line in DIMACS: *c this is a comment line* (4)

To emphasize the DIMACS syntax, an example of a CNF declaration is shown in Figure 2. The shown CNF contains 3 literals and 4 clauses. The clauses represent the family concept already described as an example for the configuration within a feature family in section 3.1 where exactly one feature, in this case either A (1), B (2) or C (3), must be selected in the configuration process in order to satisfy the CNF. Solving the satisfiability problem can be depicted as a conversion into the Disjunctive Normal Form (DNF) by applying Boolean algebra. The number and assignment of the satisfiable solutions can be deduced directly from the DNF. Asking #SAT for how many possible solutions to this very simple variance scheme exist, results in the three terms of the DNF $(1, \neg 2, \neg 3),\ (\neg 1, 2, \neg 3),\ (\neg 1, \neg 2, 3)$.

```
p cnf 3 4
1 2 3 0
-1 -2 0
-1 -3 0
-2 -3 0
```

Conjunctive Normal Form (CNF):
$(A \vee B \vee C) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee \neg C) \wedge (\neg B \vee \neg C)$

Disjunctive Normal Form (DNF):
$(\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge B \wedge \neg C) \vee (A \wedge \neg B \wedge \neg C)$

Figure 2. Example of a simple constraint with 3 literals and 4 clauses in DIMACS format, CNF formulation and DNF formulation

This already identifies the initial relevant dimensions of the rule-based variance scheme, namely the number of literals, the number of clauses, the size of the variance space and the application of the family concept.

The application of the family concept is initially determined as binary (true or false). The size of the variance space of a CNF describes the number of possible satisfiable assignments and is known as model counting or model counting problem (Thurley, 2006). Many problems and task in current Artificial Intelligence applications such as reasoning in Bayesian networks are computationally equivalent to the model counting problem (Wei and Selman, 2005).

3.3 Interim Conclusion

The German automotive industry is a perfect example of a rule-based variance scheme for representing high-variance product portfolios. The product description is based on a Boolean notation and binary one-hot encoding. The scientific field on the management of variant-rich product portfolios through a rule-based formulation of a Boolean satisfiability problem is vast and has been conducted separately in the form of a systematic literature review (Schmidt and Mantwill, 2024). Due to the complexity of the questions, designers and product engineers are dependent on the use of data-driven methods and propositional logic solvers in order to be able to provide a satisfiable solution and consider the entire variance space. Especially when developing new first products of a multi-generation product line, there might be no historical data from predecessors available and only existing experience can be used. On the other side is high quality data essential for training and testing in Artificial Intelligence applications, which cannot be guaranteed with historical data due to numerous dynamic dependencies. For those reasons, a data-driven support system is needed and intended for the early phase of product development to supports designers and product engineers in their decisions. For this purpose, a generator for synthetic variance schemes is developed and tested in this research to specifically generate variance schemes in the form of a CNF in DIMACS format. The CNF can be adapted according to predefined parameters like the number of features,

the number of constraints and the size of the variant space and therefore be used in both described cases. To answer RQ1, product portfolios which are too numerous to be described explicitly are implicitly described in variance schemes. This description can be generalized in the form of a CNF which can efficiently be processed by SAT and #SAT solvers. Therefore, the used syntax for a generic product description follows the DIMACS formulation presented in depth. Providing a generic and high complex product description contributes to examine existing variance schemes for their quality and to support the product development process by using a synthetic reference dataset for planning as well as to enable the future use of further data-driven methods which have to be trained and validated on high-quality data in the first place. This gives designers the opportunity to fully incorporate variance spaces and organize them efficiently. In particular, the efficiency of the calculation due to the consideration of the constraints is a crucial aspect, which is essential under the consideration of the size of the variance space. All these efforts are being made to ensure shorter development cycles due to fast assessments, fewer quality issues due to managing increasingly complex dependencies and to ensure the functionalities of derived homologations without modeling them all individually. In the context of frontloading, the early use of computer-aided assistants reduces effort and failure-proneness of a product and supports the management of variant-induced complexity in a highly interdependent system architecture.

4 Methodology

In the following section, the methodological approach used to systematically answer the raised questions is presented in more detail. An empirical approach to software development in the form of a study has been used in this paper. According to Kitchenham et al. possible steps of this empirical method should include Definition, Planning, Implementation and Analysis (Kitchenham et al., 2002). This approach has been chosen because it has proven to be expedient to proceed iteratively and incrementally in software development and to evaluate the results directly within the scope of an empirical study before moving on to the next increment. This approach proves effective as many requirements in the development of software and algorithms are formulated during the course of its development and can therefore be quickly adapted to changing requirements. The declared goal is to reach a Minimum Viable Product (MVP) as quickly as possible and enhance it with functionalities after.

- **Definition:** Based on requirements engineering, the first step is the gathering of relevant requirements. These include for example the identified dimensions of a CNF from the description in the previous section 3. The requirements are specified and in some cases adjusted to the specific implementation as software. This procedure is repeated iteratively after each increment has been reached. As an example, only in a later iteration did the size of the families used in the family concept prove to be relevant, which has then been included as a requirement and adjustable parameter. Another example is the introduction of literals which are always true or always false, this behavior has also been found to be important in a later iteration and implemented in a subsequent release.

- **Planning:** In the second step, again based on requirements engineering, the planning and execution of requirements management is accompanied iteratively. For this purpose, logics are built up on the notation used and initial processing procedures are planned, which are then to be implemented algorithmically. In terms of planning, a concept is first created in the form of pseudo code, which represents all relevant functionalities but cannot be compiled. This will subsequently form the basis for implementation in executable software.

- **Implementation:** The third step is the actual development of the software. For this purpose, a programming language and an Integrated Development Environment (IDE) are first selected. Since the focus of this paper lies on the use of the SAT4J (SAT4J, n.d.) solver, the Java language and the IntelliJ development environment from JetBrains are used. Furthermore, Git is used for tracking changes and version control. Input settings for the adjustable parameters can be entered using a dialog in the console when the program is executed. To ensure the reproducibility of the experiment and the consistency of the results, the input of a seed is supported for handling pseudo-random number generators. To facilitate benchmarking, the elapsed time for generating a variance scheme from execution to completion of the code is counted and printed. The export format of the CNF is set to be a .txt file for ease of use and further processing.

- **Analysis:** In the fourth step, the result is analyzed and compared with the gathered requirements. Deviations either lead to the specification of existing requirements, are included as new requirements or, in the case of a malfunction, are eliminated in the code as part of a bug fix. Finally, the result achieved is evaluated, which is discussed in more detail in section 6. In order to finally assess the validity of the solution found, an empirical study is typically validated in an experiment.

5 Concept

This section describes the conceptual approach in the form of the developed pseudo code and therefore describes the structure, used variables and processing instructions through functions of the algorithm in a readable but non-compilable form. Pseudo code is used to describe algorithms and can also be used to describe requirements in software development. The developed algorithm is presented as "CNF-Generator" and shown in pseudo code in Table 4.

Input of the algorithm are the number of literals n (*numVars),* the number of literals that are always false n_F *(numFalseVars),* the number of literals that are always true n_T *(numTrueVars),* the desired number of solutions in the variance space v_{goal} *(goalVariance)* plus the accepted deviation from v_{goal} *(acceptedDeviation),* the application of the family concept *(isFamily),* as well as the length of the families *(lengthFamilies)* and the length of the clauses *(lengthRules).* To achieve reproducibility of the results, a *seed* is used when calling pseudo-random numbers. Deviation from the desired number of solutions is accepted within the scope of the *acceptedDeviation*, as it takes an enormous amount of time to achieve the exact desired variance in large variance spaces. Output of the algorithm is the reached variance v_{reach} *(reachedVariance)*, the set of clauses ϕ *(ruleset)* as well as the elapsed time for executing the code in seconds $t_{elapsed}$. Important variable for temporary memory is the respective candidate for the next clause ψ *(nextRule).*

The used functions in the pseudo code are the call of the #SAT solver *#sat()* and the call of the SAT solver *sat()*, the call of the SAT solver to calculate the always false literals *getFalseVars()*, the call of the SAT solver to calculate the always true literals *getTrueVars()* and the addition of the clause ψ to the set of previous clauses ϕ *addRule*(ϕ, ψ). All other undeclared functions are performed accordingly.

Table 4. Pseudocode of the CNF-Generator

Algorithm 1: CNF-Generator

Input: n, n_F, n_T, v_{goal}, *acceptedDeviation, isFamily, lengthFamilies, lengthRules, seed*
Output: ϕ, $v_{reached}$, $t_{elapsed}$
function *cnf-generator(n: int,* n_F*: int,* n_T*: int,* v_{goal}*: bigint, acceptedDeviation: double, isFamily: boolean, lengthFamilies: int[min, max], lengthRules: int[min, max], seed: long)*
    $\phi \leftarrow []$ *//initialize empty ruleset*
    if *isFamily* ***is*** *True*
        $\phi \leftarrow$ *getFamilyrules(lengthFamilies) //create family rules*
    end
    $v_{reached} \leftarrow$ *#sat(*ϕ*)*
    while abs($v_{reached}$ - v_{goal}) > v_{goal} * *acceptedDeviation*
        if n_F > *getFalseVars(*ϕ*)*
            $\psi \leftarrow$ *[-1 * **random**(1, n, seed)] //create rule that is always false*
        elseif n_T > *getTrueVars(*ϕ*)*
            $\psi \leftarrow$ *[**random**(1, n, seed)] //create rule that is always true*
        else
            ruleSize $\leftarrow$ ***random****(lengthRules[1], lengthRules[2], seed)*
            $\psi \leftarrow []$
            for *i* ***in*** *range (1, ruleSize)*
                ψ *[i]* $\leftarrow$ *-1* ***random****(1, n, seed) //create "normal" rule*
            end
        end
        addRule(ϕ*,* ψ*)*
        if *sat(*ϕ*)* ***is*** *False* $\vee$ *getFalseVars(*ϕ*)* > n_F $\vee$ *getTrueVars(*ϕ*)* > n_T
            removeRule(ϕ*,* ψ*) //ruleset check failed*
            continue
        end
        varianceCurrentRuleset $\leftarrow$ *#sat(*ϕ*)*
        if *varianceCurrentRuleset* < v_{goal} − (v_{goal} * *acceptedDeviation*)
            removeRule(ϕ*,* ψ*) //variance check failed*
            continue
        end
        $v_{reached} \leftarrow$ *varianceCurrentRuleset*
    end
    return ϕ, $v_{reached}$
end

6 Results and Concept Evaluation

As part of this research, executable software has been developed which is presented in this section in the form of the results of this work on the basis of a generated CNF. The software is fully available as a GitHub repository and made publicly accessible (CNF-Generator, n.d.).

Figure 3 shows an excerpt from an exemplarily CNF, which has been generated as a .txt file and can be viewed and downloaded from the output directory (CNF-Generator, n. d.). In the first 14 lines, input parameters and actual achieved parameters are listed as comments for a comprehensive summary including calculation time and used seed. The CNF fulfills the requirement of satisfiability and has more than one solution. The consistency of the variance scheme cannot be adequately answered without the subsequent derivation of probability-based configurations (Demke et al., 2021), but any contradiction in the CNF can be excluded. At approx. 0.04%, the deviation in the size of the variance space obtained is within the specified range in the order of magnitude of the variance space of approx. 10^{10} possible solutions. The generated variance scheme comprises a total of 100 literals, whereby 609 clauses had to be added in order to achieve the desired variance. In the example shown, the family concept has been applied with a maximum family size of 6 literals and a maximum size of 6 literals for inter-family constraints. The number of always true and always false literals have been reached. Line 15-624 specify the constraints in DIMACS syntax. For simplification reasons, only the first two feature families (line 16-23), two constraints for always true variables (line 215-216) and the last two of the inter-family constraints (line 623-624) are shown. The total time required to generate this specific CNF equals to approx. 1 hour and 12 minutes. When generating these CNFs, the majority of the time required is spent on the #SAT calculation. The used seed for traceability and reproducibility is depicted in shown example.

```
1   c
2   c Input Variance: 10000000000
3   c Actual Variance: 9995700077
4   c Input number of vars: 100
5   c Input use Fam rules: true
6   c Input Fam size: 1-6
7   c Input Rule size: 2-6
8   c Input False Variables: 4
9   c Actual False Variables: 4 Vars
10  c Input True Variables: 8
11  c Actual True Variables: 8 Vars
12  c Calculation time: 4331 seconds
13  c Used Seed: -1062837989883594406
14  c
15    p cnf 100 609
16    1 0
17    2 3 4 5 0
18    -2 -3 0
19    -2 -4 0
20    -2 -5 0
21    -3 -4 0
22    -3 -5 0
23    -4 -5 0
…     …
215 98 0
216 6 0
… …
623   -64 -90 -25 0
624   -51 -18 0
```

Figure 3. Example of a CNF generated by the CNF-Generator (CNF-Generator, n.d.) with 100 literals and 609 clauses

In addition to the technical generation of CNFs, the possibility has been created to examine and compare methods of data-driven product development more precisely without having to suppress the operational dynamic changes and inconsistencies. The Boolean logic and overall functionality of the generator could be verified and the requirements are achieved. The functionalities of the generator are currently limited to the described requirements of a CNF in DIMACS format. Furthermore, the results are not limited to the German automotive industry but are suitable for any use case of highly variant product portfolios described by a rule-based variance scheme. The distinctive advantage of the generated solution can be seen in the pseudo-random nature of the added clauses while maintaining satisfiability. CNFs of variable size can be generated and compared using model counting. The bottleneck in the runtime is up to now the model counting process, which is highly computationally intensive. The presented solution is runtime-optimized due to the use of efficient solvers and optimized calls and thus applicable in industry. The concept described in pseudo code (Table 4) shows how an algorithm must be conceptually designed in order to meet the requirements for the generation of a rule-based variance scheme. To cover a more extensive spectrum of possible applications and to enable comparisons, it is possible to create personalized variance schemes by inputting individual parameters into the generator. By using a seed, these are fully reproducible and directly usable. A conceptual evaluation has been carried out in initial tests, but an objective comparison under comparable conditions to a rule-based variance scheme from the automotive industry is still pending.

This result can be used for strategic product planning of new products in the early phase and to generate synthetic reference data for data-driven design. The ability to influence the result in a targeted manner allows complex dependencies to be simulated and the effects of changes to the variance scheme to be verified. On the one hand, this enables developers to model, train and test applications on a benchmark with predefined syntax. On the other hand, designer and product engineers have the possibility of data-driven decision support in order to make assisted decisions for complex decisions like variant management and system architecture as early as possible.

7 Conclusion and Outlook

In conclusion, this research confirms that the generation of rule-based variance schemes to support the development of highly variant product portfolios is generally possible. The approach has been conceptually described in the form of detailed pseudo code and has been implemented in the form of publicly available software (CNF-Generator, n.d.). The generation of CNFs is scalable, however, the scaling is not linear, but approximately exponential to the number of literals used considering the runtime. The presented variance schemes are customized for the use case of variant configuration in the German automotive industry but can be applied to other industries as well. The examined parameters are relevant for the present use case and can be specified during the input dialog in the console. Other parameters may be of interest for other use cases, which results in limited generalizability. Reproducibility is available due to the possibility to use a seed. Compared to the examined CNFs, the generated variance schemes are still comparatively small samples. The runtime of the #SAT calculation is currently still challenging for the generation of solution spaces of the order of 10^{60} and bigger as they occur in automotive engineering.

Research Question RQ1 has already been answered in the interim conclusion of section 3 by the CNF as product description of choice for highly variant product portfolios and the used DIMACS formulation as the generic syntax. Research Question RQ2 has been answered methodological by applying the research approach developed by Kitchenham et al. (2002), conceptually in the form of pseudo-code and compilable in the form of software implementation. As a result, it is possible to create a specific but generic product model that complies with the predefined syntax but is adjustable to input parameters. Research Question RQ3 has only been answered to a limited extent. Synthetic reference data assist in the planning of variants and are a suitable method for simulating the effects of changed constraints on a rule-based variance scheme. They also help in the early phase to quantify the effects of decisions regarding system architectures and the impact of merging features in the form of packaging. They are particularly suitable for enabling developers of data-driven decision support systems to evaluate their algorithmic approaches and compare them against a scalable benchmark. A profound answer to RQ3 requires further research with more extensive data and more comprehensive analysis.

For developers, an investigation into the runtimes and potentially an estimated calculation to accelerate the approach to a good approximation is still pending. Initial investigations of downstream analyses on the reference dataset, such as the Apriori algorithm on synthetic vehicle orders from a generated CNF, suggest good comparability with real historical data and thus suitability as generic product description. An objective comparison with comparable data sets against a benchmark and an investigation of the installation rate intervals for a large number of vehicle orders generated from the CNF is a matter of future research.

For designers and product engineers, the application in the form of assistances has not yet been proven and is difficult to evaluate. The use of data-driven decision support is mandatory due to the prevailing complexity in variant management. This decision support includes the highlighted aspects of the satisfiability of logical expressions in Boolean logic, consistency analyses and the breakdown of a configuration-specific BOM. However, the decision support is not yet exhaustive. For this reason, this work contributes to demand-oriented decision support and enables the successive addition of new capabilities to support the product development process in the future.

References

Albers, A., Bursac, N., Wintergerst, E., 2015. Produktgenerationsentwicklung – Bedeutung und Herausforderungen aus einer entwicklungsmethodischen Perspektive. In: Binz, Hansgeorg; Bertsche, Bernd; Bauer, Wilhelm; Roth, Daniel (Eds.): Stuttgarter Symposium für Produktentwicklung, Fraunhofer IAO, Stuttgart, pp. 1-10.

Biere, A., Heule, M., Van Maaren, H., Walsh, T., 2009. Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, ISBN: 978-1-58603-929-5.

CNF-Generator, n.d. Marbach, S., CNF-Generator, GitHub repository, URL: https://github.com/SteffenHub/CNF-Generator, last accessed on July, 19th, 2024.

C2D, n.d. Darwiche, A., #SAT Solver for Windows, URL: http://reasoning.cs.ucla.edu/c2d/, last accessed on July, 19th, 2024.

Darwiche, A., 2005. The C2D Compiler User Manual, Computer Science Department, University of California, Los Angeles.

Davis, M., Logeman, G., Loveland, D., 1962. A machine program for theorem-proving. In: Communications of the ACM 5 (7), pp. 394–397, https://doi.org/10.1145/368273.368557.

Demke, N., Tichla, F., Zühlke, J., Schmidt, T., Mantwill, F., 2021. Derivation of probability-based configurations for early feasibility statements in the development of products with high variance, IJCAI Workshop 2021.

Eén, N., Sörensson, N., 2003. An extensible SAT solver. In: Giunchiglia, E., Tacchella, A. (Eds.): Theory and Applications of Satisfiability Testing, SAT 2003, Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing, LNCS 2919, pp 502-518, Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-540-24605-3_37.

Ehrlenspiel, K., Kiewert, A., Lindemann, U., Mörtl, M., 2014. Kostengünstig Entwickeln und Konstruieren. Kostenmanagement bei der integrierten Produktentwicklung. 7. Aufl., Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-642-41959-1.

Eisenhart Rothe, M. von, 2002. Konzeption und Einführung eines IT-gestützten Produkt-Konfigurationsmanagements für die technische Information in der Automobilentwicklung und -herstellung. Shaker Verlag, Aachen, ISBN: 978-3-8322-0419-9.

Frischen, C., Marbach, A., Tichla, F., Mantwill, F., 2019. Consistent controlling of variants with the aid of the rule-based complex bill of materials. Proceedings of the 30th Symposium Design for X, Symposium Design for X, DFX 2019, Jesteburg, pp. 13 - 24, https://doi.org/10.35199/dfx2019.2.

FSU, n.d. Florida State University (FSU), CNF Files, URL: https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html, last accessed on July, 19th, 2024.

Herlyn, W., 1990. Zur Problematik der Abbildung variantenreicher Erzeugnisse in der Automobilindustrie. VDI Verlag, Düsseldorf, ISBN: 978-3-18-145216-5.

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El-Emam, K., 2002. Preliminary guidelines for empirical research in software engineering; IEEE-Transactions-on-Software-Engineering, 28 (8), pp. 721-734, doi: 10.1109/TSE.2002.1027796.

Marbach, A., Tichla, F., Schmidt, T., Mantwill, F., 2023. Data-Driven Optimization of the Product Portfolio for Highly Variant End Products in the Automotive Industry. In: Hölzle, Katharina; Kreimeyer, Matthias; Roth, Daniel; Maier, Thomas; Riedel, Oliver (Eds.): Stuttgarter Symposium für Produktentwicklung. Fraunhofer IAO, Stuttgart, URL: http://dx.doi.org/10.18419/opus-13131.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., Malik, S., 2001. Chaff: Engineering an efficient SAT Solver. In: Jan Rabaey (Ed.): Proceedings of the 38th conference on Design automation - DAC '01. Las Vegas, Nevada, United States. New York, New York, USA: ACM Press, pp. 530–535, URL: https://dl.acm.org/doi/pdf/10.1145/378239.379017.

Petrie, C. J., 2012. Automated Configuration Problem Solving, Springer New York, New York, URL: https://doi.org/10.1007/978-1-4614-4532-6.

Prokls, n.d.. Prokls, Public CNF benchmark files with zsh scripts, GitHub repository, URL: https://github.com/prokls/cnf-files-download, last accessed on July, 19th, 2024.

Sat4J, n.d. Artois University and CNRS, SAT Solver for Java, URL: http://www.sat4j.org/, last accessed on July, 19th, 2024.

Schmidt, T., Marbach, A., Mantwill, F., 2022. Recommender Systems for Variant Management in the Automotive Industry. In DfX 2022: Proceedings of the 33th Symposium Design for X, Hamburg, Germany, https://doi.org/10.35199/dfx2022.13.

Schmidt, T., Mantwill, F., 2024. Management of rule-based product-portfolio with high variance: A systematic literature review. Proceedings of the Design Society, Cambridge University Press, pp. 755-764, doi:10.1017/pds.2024.78.

SharpSAT, n.d. Marc Thurley, #SAT Solver for Linux, GitHub repository, URL: https://github.com/marcthurley/sharpSAT, last accessed on July, 19th, 2024.

Thurley, M., 2006. sharpSAT – Counting Models with Advanced Component Caching and Implicit BCP. In: Biere, A., Gomes, C.P. (Eds.) Theory and Applications of Satisfiability Testing - SAT 2006. SAT 2006. Lecture Notes in Computer Science, vol 4121. Springer, Berlin, Heidelberg, URL: https://doi.org/10.1007/11814948_38.

Tseng, M. M., Yue W., Roger J. J., 2017. Mass Customization. In: The International Academy for Production Engineering, Laperrière L., Reinhart G. (eds) CIRP Encyclopedia of Production Engineering. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35950-7_16701-3

Valiant, L.G., 1979. The complexity of computing the permanent. In: Theoretical Computer Science 8 (2), pp. 189-201, ISSN 0304-3975, https://doi.org/10.1016/0304-3975(79)90044-6.

VDI 2221. 2019. VDI-Standard VDI 2221 Part 1, 2019, Design of technical products and systems.

Wei, W., Selman, B., 2005. A New Approach to Model Counting. In: Bacchus, F., Walsh, T. (Eds.) Theory and Applications of Satisfiability Testing. SAT 2005. Lecture Notes in Computer Science, vol 3569. Springer, Berlin, Heidelberg, URL: https://doi.org/10.1007/11499107_24.

Zagel, M., 2006. Übergreifendes Konzept zur Strukturierung variantenreicher Produkte und Vorgehensweise zur iterativen Produktstruktur-Optimierung, Dissertation, Schriftenreihe VPE, Band 1, ISBN: 978-3-939432-26-5.

Contact: Thorsten Schmidt, Helmut Schmidt University, Department of Mechanical Engineering and Computer-assisted Product Development, Holstenhofweg 85, 22043 Hamburg, Germany, +49 40 6541-3794, thorsten.schmidt@hsu-hh.de, www.hsu-hh.de/mrp