On the Use of Multi-valued Decision Diagrams to Count Valid Configurations of Feature Models

Andrea Bombarda
andrea.bombarda@unibg.it
University of Bergamo
Italy

Angelo Gargantini
angelo.gargantini@unibg.it
University of Bergamo
Italy

ABSTRACT

This paper addresses the challenge of efficiently counting valid configurations in Software Product Lines (SPLs). We propose a novel approach leveraging Multi-Valued Decision Diagrams (MDDs) for building the set of products. Building upon the MDD structure, we introduce several algorithmic optimizations to achieve a more compact and efficient representation of the product set compared to existing methods based on Binary Decision Diagrams.

The effectiveness of our approach is evaluated through experimentation on two datasets: a set of synthetic benchmarks and large-scale industrial feature models. The results demonstrate significant improvements in scalability for models of medium complexity, particularly those rich in alternative groups. However, challenges remain for other model types, highlighting areas for future research.

ACM Reference Format:
Andrea Bombarda and Angelo Gargantini. 2024. On the Use of Multi-valued Decision Diagrams to Count Valid Configurations of Feature Models. In *28th ACM International Systems and Software Product Line Conference (SPLC '24), September 2–6, 2024, Dommeldange, Luxembourg.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3646548.3672594

1 INTRODUCTION

Practitioners are increasingly turning to Software Product Lines (SPLs) to model highly configurable systems having common aspects and variability parameters. The various products that stem from a family of SPL are differentiated based on their features, which represent user-visible aspects or characteristics of the software system. These features within an SPL can exhibit diverse relationships, which can be captured in a compact notation by using Feature Models (FMs).

Given a FM, a common problem regards its knowledge compilation: the formula represented by the FM (i.e., the logical formula defining which products are valid and which are not) is translated into a target representation, which is then used to answer several queries in a more efficient way. A typical example which falls in the scope of knowledge compilation is the computation of the number of valid products in an SPL. Having such a number is extremely important for software engineers [1]. For example, it can support the testing phase by allowing uniform sampling of products (which requires the knowledge of how many valid configurations are available), detecting possible design errors, performing economical estimations, and it aids developers in reducing the variability of an SPL.

The problem of computing the number of valid configurations in an SPL is well-explored in the literature, and has been mainly tackled by using logical solvers, Binary Decision Diagrams (BDDs) [2], #SAT [3], or deterministic decomposable negation normal form (d-DNNF) [4]. However, the research on leveraging knowledge compilation for FM analysis is still rather limited in terms of considered analyses, considered knowledge-compilation target languages, and compiler scalability [5]. For this reason, in this paper, we present our attempt in using Multi-valued Decision Diagrams (MDDs) for this task, with the results we have obtained.

This paper introduces a fresh mapping approach that translates FMs into MDDs, aiming to address the limitations encountered in previous endeavors utilizing decision diagrams for knowledge compilation. In particular, by exploiting the possibility of having multiple edges from a single node, the approach we propose in this paper reduces the number of nodes required to represent a FM w.r.t. BDDs. Beside the straightforward mapping of `Alternative` groups, we propose three optimizations, namely, the merging of `And`/`Or` groups, the variable re-ordering, and the merging of the cross-tree constraints before their application to the MDD representing the FM under analysis.

We present the experiments we have conducted over benchmark models to analyze the impact of the proposed optimizations on the performance of the knowledge compilation process. In addition, we analyze our approach on large-scale industrial FMs to assess their applicability on real-world examples. Our experiments allowed us to claim that the proposed optimizations are effective for increasing the performance of our approach. Moreover, we have been able to compute the cardinality of 10 out of the 11 large-scale industrial FMs within the timeout of 24 hours, showing that the approach we propose in this paper can be applied to real-world examples and is particular useful for mid-scale case studies.

The paper is structured as follows. In Sect. 2, we describe the background on Software Product Lines, Feature Models, Configurations and Multi-valued Decision Diagrams. Sect. 3 presents our approach to count valid configurations in an SPL with MDDs, together with the optimizations we have adopted to push even further their performance, while in Sect. 4 we present the experiments designed to evaluate the proposed solution. Sect. 5 discusses potential threats to the validity of our findings. Finally, Sect. 6 reports the related work and Sect. 7 concludes the paper.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SPLC '24, September 2–6, 2024, Dommeldange, Luxembourg
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0593-9/24/09…$15.00
https://doi.org/10.1145/3646548.3672594

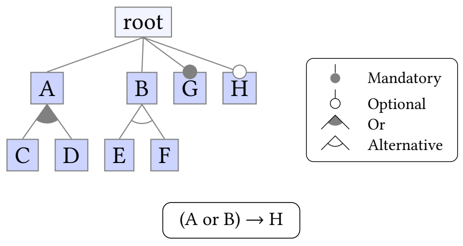


Figure 1: Example of a FM

2 BACKGROUND

This section provides an overview of basic concepts related to Feature Models, product configurations, and Multi-valued Decision Diagrams.

2.1 SPL and Feature Models

In the field of Software Product Line engineering, feature models [6, 7] serve as models delineating all potential products within a software product line (SPL) based on their features and interrelationships. To be precise, a feature model (denoted as *FM*) comprises a structured collection of features *F*, organized in a hierarchical manner. Each parent-child relationship within this hierarchy imposes a constraint falling into one of the subsequent categories: I) *Or*: at least one of the sub-features must be selected if the parent is selected. II) *Alternative/Xor*: exactly one of the children must be selected whenever the parent feature is selected. III) *And*: if the relation between a feature and its children is neither an *Or* nor an *Alternative*. Each child of an *And* must be either: - *Mandatory*: the child feature is selected whenever its respective parent feature is selected. - *Optional*: the child feature may or may not be selected if its parent feature is selected.

Only the `root` feature in *F* has no parent and it is selected in every product. In addition to hierarchical relations, FMs may have *cross-tree constraints*, i.e., relations that cross-cut hierarchy dependencies. The most common cross-tree constraints are:

- `A` *requires* `B`: the selection of a feature `A` in a product also implies the selection of the feature `B`. We indicate it as `A` ⟶ `B`.
- `A` *excludes* `B`: features `A` and `B` cannot be part of the same product. We indicate it as `A` ⟶ ¬`B`.

As common in the literature on FMs, in this work, we allow FMs to contain cross-tree constraints given by *general* propositional formulas. Furthermore, a feature can be *dead*, i.e., it can never be selected because of the constraints, or *core*, i.e., it is mandatory in every valid product, like the *root*.

An example of a *FM* is shown in Fig. 1. It has a `root` feature with an `And` group of sub-features, namely `A`, `B`, `G`, and `H`. The feature `A` represents an `Or` group between the features `C` and `D` while the feature `B` is an `Alt` between `E` and `F`. Finally, `G` is a *mandatory* feature while `H` is `optional`. Moreover, *FM* contains a cross-tree constraint requiring that if `A` or `B` are selected, then `H` must be selected as well.

2.2 SPL configurations

When working with a FM defined on the set of features *F*, each *configuration* specifies which of the features in *F* are selected and which are not. In other words, the set of all configurations represents all possible products that can be derived from an SPL designed by means of a FM. In this work, we consider a configuration as defined by [8, 9]. More specifically, a configuration *c* is a function giving the status of a feature $\mathtt{f} \in F$ in *c*

$$c(\mathtt{f}) = \begin{cases} \top(true) & \text{if } \mathtt{f} \in F \text{ is selected in c} \\ \bot(false) & \text{if } \mathtt{f} \in F \text{ is not selected in c} \end{cases}$$

A configuration is valid in *FM* if and only if it denotes a valid product, i.e., it does not violate any constraint in *FM* (both structural and cross-tree). In the example shown in Fig. 1, the configuration having only `root` and `G` selected is a valid one, as well as the one having `root`, `G`, `B`, `F`, and `H` selected. Instead, the configuration selecting `root`, `G`, `B`, and `F` is not valid, as it violates the cross-tree constraint (`B` is selected, but `H` is not).

Counting the number of valid configurations is paramount important as it allows stakeholders to detect design errors, perform economical estimations, and aids developers by reducing the variability of the SPLs, to prioritize features and to perform uniform random sampling of configurations [10].

2.3 Multi-valued Decision Diagrams

The approach presented in this paper is based on using *Multi-valued Decision Diagrams* (MDDs), which are a particular type of *decision diagrams* (DDs). More formally, DDs are defined as follows.

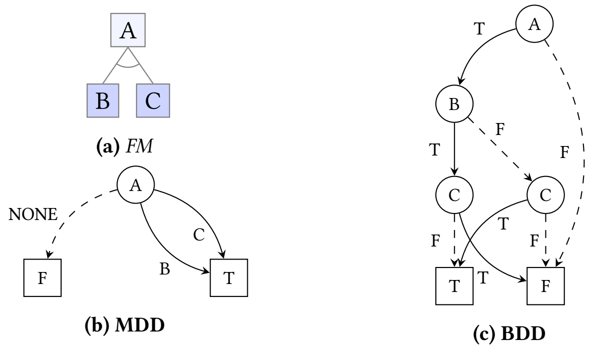
Definition 1 (Decision diagram). *A decision diagram is a rooted directed acyclic graph with two terminal nodes F and T and it represents a function* $g : D \longrightarrow B$ *where* $D = D_1 \times \cdots \times \cdots D_n$ *and B is the Boolean domain, i.e.,* $B = \{F, T\}$.

A DD can be used to evaluate the truth value of the function g when it is applied to the variables $x_1, \cdots, x_n$ in the following way. Every decision node is labeled by a variable x_i and every edge with its label represents the direction one must take depending on the value of x_i in D_i. The final node F or T represents the value of the function g when applied to the chosen values of x_i.

If all the domains D_i are binary, then we refer to the decision diagrams as Binary Decision Diagrams (BDDs). They are used to represent simple Boolean functions. BDDs are widely used within the domain of system design verification and for counting the number of configuration of SPLs. However, while operations on BDDs might scale well, their construction can be intractable [5] when representing FMs, as many constraints are needed to model the relationship between variables. An example of BDD modeling the FM in Fig. 2a is shown in Fig. 2c.

To partially solve the limitations of BDDs, Multi-Valued Decision Diagrams (MDDs) have been introduced and they extend BDDs by allowing every variable to have a different domain, with a different size. While many extensions of BDDs do exist [11], MDDs are suitable when the domain of the decision is Boolean (like the validity of a configuration) and the domain of the variables x_i can be enumerations (like a parent feature that can take values in the child feature domain). Each MDD has the following properties:

- only two terminal nodes are available, which are labeled as `F` and `T`;
- every non-terminal node is labeled by an input variable x_i and has $|D_i|$ outgoing labeled edges, i.e. one per each possible value of the domain;



(a) *FM*   (b) **MDD**   (c) **BDD**

Figure 2: Representation of a simple FM with DDs

- every variable appears only once in the MDD, in any path from the root to a terminal node;

Given these properties, an MDD can represent which values of the domain D are selected by the function g. In fact, if the values $x_1, \dots, x_n$ for the variables in D are selected by g, then $g(x_1, \dots, x_n) = T$, otherwise $g(x_1, \dots, x_n) = F$. An example of MDD modeling the FM in Fig. 2a is shown in Fig. 2b. It can be seen that the representation of the same FM is more compact in the case of the MDD w.r.t. the BDD. This means that the same information can be represented with fewer edges and nodes, making it easier to deal with more complex FMs.

Among decision diagrams, it is possible to perform unary operations such as *complement*, or the most classical binary operations like *union*, *intersection*, and *difference*. Additionally, in decision diagrams (both BDDs and MDDs) the computation of the *cardinality* - i.e. the number of assignments that make g true - is a simple arithmetic operation that can be done in linear time (w.r.t. the number of nodes [12, 13]) and it does not require the enumeration of all the valid assignments.

In this paper, we will exploit these operations to build the MDD corresponding to a *FM* and to compute the number of its valid configurations, i.e., its cardinality. The main challenge remains building the MDD for a given FM.

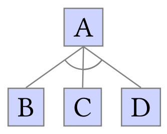
3 BUILDING AN MDD FOR A FM

In this section, we delve into how we use MDDs to map FMs and to count their valid configurations. Our approach is implemented in a tool written in C++, and available in our replication package at https://github.com/fmselab/FMConfigurationsCounter. It is based on the MDD implementation offered by MEDDLY [14]. Our tool supports FMs in the xml format of FeatureIDE [15]. Moreover, our approach exploits the GMP library [16] which allows us to handle arbitrary-long numbers.

The process implemented by our approach is reported in Algorithm 1. It works by taking as input the feature model *FM* of interest and returns as output the cardinality of the MDD, which corresponds to the number of valid configurations for *FM*. Initially, the variables representing features and feature groups are defined and the MDD M is created (line 1). This process is described in Sect. 3.1. Then, the constraints mapping the structure and semantic of *FM*, i.e., mandatory features, hierarchical relations, `Alternative`, `And` and `Or` groups, are added to M (line 2 and 3). This process is described in Sect. 3.2. Finally, in a similar way, the algorithm adds the cross-tree constraints (line 4). The way in which constraints are converted and merged with M is presented in Sect. 3.3. At the end of the process, M represents the MDD describing all the valid products, and the cardinality of M is the number of them (line 5). Additionally, the process building the MDD for a FM can be optimized as described later in Sect. 3.4.

Algorithm 1 Algorithm counting the number of valid configurations with an MDD

```
Require: FM the feature model of interest
Ensure: card the cardinality, i.e., the num. of valid configurations
        ▹ Define all the variables and the empty MDD
1: $M \leftarrow$ DEFINEMDD(FM)
        ▹ Add the constraints for the group features (from the root)
2: SETROOTMANDATORY(M,FM.getRoot())
3: ADDGROUPCONSTRAINTS(M,FM.getRoot())
        ▹ Add to the MDD the cross-tree constraints
4: ADDCROSSTREECONSTRAINTS(FM, M)
5: return $M.getCardinality()$
```



x_A	Selected F. A	B	C	D
NONE				
B	×	×		
C	×		×	
D	×			×

(a) **`Alt.` group**   (b) **Mapping of the `Alt.` group to the variable x_A**

Figure 3: `Alternative` group

3.1 Features and groups

Generally, every feature $f \in F$ is associated with a variable x_f within the MDD. The variables' domains and potential constraints between them vary based on the feature type. We decided to initially represent every feature f with a Boolean variable x_f except for the features that are parent or child of an *alternative* group. For an alternative group f with children $c_1 \dots c_n$, we introduce only *one* variable x_f with domain $\{\texttt{NONE}, c_1, \dots, c_n\}$. The value `NONE` is taken by x_f if and only if f is not selected, as formalized in the following. An example of how an alternative group is translated into a unique MDD variable is reported in Fig. 3. It shows the selected features depending on the value of the MDD variable x_A.

3.2 Constraints for groups

To define the constraints (both tree and cross-tree), it is imperative to represent the selection status of a feature within the MDD. For instance, this is necessary because we need to represent that a child feature may only be selectable when its parent is also selected. However, since various group types are translated differently (as explained below), there is not a singular method to determine how to assess a feature's status. Therefore, we introduce the `isSel`

Algorithm 2 Procedure adding group constraints to the MDD

```
Require: $M$ the MDD being built
Require: $f$ the feature being visited
1: procedure AddGroupConstraints($M, f$)
2:     switch $group(f)$ do
3:         case $And$: visitAnd($M, f$)
4:         case $Or$: visitOr($M, f$)
5:         case $Alternative$: visitAlternative($M, f$)
6:     for all $c_i$ in $children(f)$ do
7:         AddGroupConstraints($M, c_i$)
8:     end for
9: end procedure
```

function as follows:

$$\mathtt{isSel}(f) = \begin{cases} x_f \neq \mathtt{NONE} & \text{if } \mathtt{f} \text{ is the parent of an alternative group} \\ x_p = \mathtt{f} & \text{if } \mathtt{f} \text{ is a child of an alternative group with parent } \mathtt{p} \\ x_f = \mathtt{true} & \text{otherwise (or simply } x_f \text{ since it is Boolean)} \end{cases}$$

While for features translated to Boolean variables, the `isSel` function is trivial, for alternative groups `isSel` must take care of the child feature selected, if any.

Note that, if a feature f is *both* parent and child of an alternative group, we assume that the function $\mathtt{isSel}(f)$ considers f only as a parent, i.e., $x_f \neq \mathtt{NONE}$. However, this situation requires attention when translating `alternative` groups and we will deal with it later, when adding the constraints among parents and their children.

The `isSel` function is firstly used to set the root feature as mandatory (line 2 in Algorithm 1). More specifically, for each FM, the following constraint will always be added:

$$\mathtt{isSel(root)}$$

To build the MDD representing a FM, besides creating the variables as outlined before, we need to add the constraints among the features due to the structure of the FM, as reported in Algorithm 2. To do this, we recursively visit the model starting from the root (see line 3 in Algorithm 1). For every feature f, we consider the group type it represents as explained below, we add the tree constraints for f and its particular group, and we recursively visit all the children of f. In the following, we explain what the visiting algorithm does when called for a feature f.

Leaf. If f is a leaf, the visiting algorithm does not do anything - since all the constraints are added by considering the feature groups as explained below.

And groups. The translation of `And` groups is straightforward. Let f be the parent feature, c_i be its n optional children, and $\hat{c}_i$ be its m mandatory children, we add the following constraints:

$$\begin{cases} \mathtt{isSel}(c_1) \rightarrow \mathtt{isSel}(f) \\ \dots \\ \mathtt{isSel}(c_n) \rightarrow \mathtt{isSel}(f) \\ \mathtt{isSel}(\hat{c}_1) \leftrightarrow \mathtt{isSel}(f) \\ \dots \\ \mathtt{isSel}(\hat{c}_m) \leftrightarrow \mathtt{isSel}(f) \end{cases}$$

The constraints guarantee that if a child is selected, then, also the parent must be selected. For mandatory children also the vice-versa holds: if the parent is selected, then also the child must be selected.

Let us consider the `And` group in the FM in Fig. 1. The parent feature (i.e., `root`) as well as all the children (i.e., `A`, `B`, `G`, and `H`) are translated to MDD variables $(x_{root}, x_A, x_B, x_G, x_H)$. The constraints guaranteeing the correct parental relation between the children and the parent are:

$$\begin{cases} \mathtt{isSel}(A) \rightarrow \mathtt{isSel}(root) \\ \mathtt{isSel}(B) \rightarrow \mathtt{isSel}(root) \\ \mathtt{isSel}(G) \leftrightarrow \mathtt{isSel}(root) \\ \mathtt{isSel}(H) \rightarrow \mathtt{isSel}(root) \end{cases} \rightarrow \begin{cases} x_A \rightarrow x_{root} \\ x_B \neq \mathtt{NONE} \rightarrow x_{root} \\ x_G \leftrightarrow x_{root} \\ x_H \rightarrow x_{root} \end{cases}$$

Or groups. Representing `Or` groups in an MDD is very similar to `And` ones. Let f be the parent feature, c_i be its n or children, we add the following constraints:

$$\begin{cases} \mathtt{isSel}(c_1) \rightarrow \mathtt{isSel}(f) \\ \dots \\ \mathtt{isSel}(c_n) \rightarrow \mathtt{isSel}(f) \\ \mathtt{isSel}(f) \rightarrow \bigvee_{i=1}^{n} \mathtt{isSel}(c_i) \end{cases}$$

The constraints guarantee the correct parental relation between the children. Moreover, the last additional constraint forces at least one of the children features to be selected.

Let us consider the `Or` group in the FM in Fig. 1. The parent feature (i.e., `A`) as well all the children (i.e., `C` and `D`) are translated into the MDD Boolean variables x_A, x_C, and x_D. The following constraints are added:

$$\begin{cases} \mathtt{isSel}(C) \rightarrow \mathtt{isSel}(A) \\ \mathtt{isSel}(D) \rightarrow \mathtt{isSel}(A) \\ \mathtt{isSel}(A) \rightarrow (\mathtt{isSel}(C) \vee \mathtt{isSel}(D)) \end{cases} \rightarrow \begin{cases} x_C \rightarrow x_A \\ x_D \rightarrow x_A \\ x_A \rightarrow (x_C \vee x_D) \end{cases}$$

Alternative groups. When visiting a feature that is a parent of an alternative group, we introduce just one variable for the entire group, such as in Fig. 3 - and this allows us to take advantage of the multivalued nature of MDDs. Using a single variable already takes into account that if a child is selected then the parent is selected as well, and if the parent is selected then one of the children is selected as well. Indeed, when the parent is not selected (`NONE`) no child can be selected. Instead, if the parent is selected ($\neq$ `NONE`) one child must be selected as well. So, in general, no action is required in case of an alternative group. However, in case a feature f is both parent and child of an alternative group, the function $\mathtt{isSel}(f)$ considers f only as parent (i.e., $x_f \neq \mathtt{NONE}$), so we have to add a constraint to its parent x_p forcing the consistency between the two

Logical Op.	MDD Op.	Logical Op.	MDD Op.
A *and* B	$M_A \cap M_B$	A *or* B	$M_A \cup M_B$
not A	$\neg M_A$	A $\rightarrow$ B	$\neg M_A \cup M_B$
A = B	$M_A = M_B$		

Table 1: Mapping logical operations to MDD operations

variables:

$$x_f \neq \text{NONE} \leftrightarrow x_p = \text{f}$$

An example of this mapping procedure is reported in Fig. 3. The *FM* in Fig. 3a is an `alternative` group with three children. It is mapped to a single MDD variable x_A which can assume only 4 values: `NONE` indicates that no feature of the group is selected (not even the parent `A`), `B` indicates that the feature `B` is selected (and, consequently, also feature `A`), and so on.

More details on how these constraints are added to an MDD are given in Sect. 3.3.

3.3 Cross-Tree constraints

After having built the MDD M representing all the features and groups of *FM*, cross-tree constraints are added to M (line 4 in Algorithm 1). In this phase, each single constraint c is converted into its corresponding MDD M_c. The procedure allowing the representation of a constraint with its corresponding MDD is recursive: the constraint is visited until a set of simple MDDs of each single atomic predicate is created. The atomic predicates are *features* which are represented in the constraints by using the `isSel` function. Then, the MDDs are merged by exploiting binary operations as reported in Tab. 1, where M_x is the MDD representation of the atomic predicate x. Let us consider the cross-tree constraint c reported in Fig. 1. The atomic predicates, corresponding to the features, are A, B, and H. These predicates are converted into their corresponding `isSel` function and three MDDs representing these functions are derived:

$$\begin{cases} \texttt{isSel}(A) = x_A \\ \texttt{isSel}(B) = (x_B \neq \text{NONE}) \\ \texttt{isSel}(H) = x_H \end{cases} \rightarrow \begin{cases} M_A \\ M_B \\ M_H \end{cases}$$

The final MDD M_c for the constraint c is obtained by combining M_A, M_B, and M_H using the operators from Tab. 1 corresponding to those in c:

$$M_c = \neg(M_A \cup M_B) \cup M_H$$

At the end of the process, M_c will be the exact MDD representation of the constraint c. Thus, it can be added to the MDD M created when defining groups and features by computing

$$M = M \cap M_c$$

After having repeated this recursive operation for all constraints, M will contain all valid products and its cardinality will be the number of valid configurations for the *FM* under analysis.

3.4 Optimizations

One of the main problems of decision diagrams is related to their potential scalability issues, especially in terms of memory occupancy. Every time a new variable is added to the decision diagram, this implies introducing new nodes and edges to the MDD whose number is multiplied at every operation (e.g., the addition of a constraint). For this reason, we have introduced several optimizations aiming to reduce the number of edges and nodes.

Leaf mandatory features. When encountering a *leaf* and *mandatory* feature $\text{f} \in F$, we avoid introducing it into the MDD. As a consequence, if the mandatory feature `f` is present in any of the constraints, it is substituted with its parent feature, if any, or with `true`. In this way, we reduce the number of variables and, thus, nodes, but we keep the MDD cardinality unaltered.

Dynamic variable re-ordering. The order of variables in an MDD strongly influences the performance of operations between MDDs. In our approach, we apply dynamic re-ordering which is a technique to re-order the MDD variables to reduce the size (in terms of number of edges and nodes) of the existing MDDs [17, 18]. Dynamic ordering can often reduce the size of the MDDs dramatically, but it may be time consuming. For this reason, we do not apply variable re-ordering after every operation (i.e., the application of a cross-tree constraint). Based on the results we obtained with our experiments, we have defined the following strategy: we re-order the variables when the number of nodes in the MDD exceeds 10^6 or when the number of nodes from one operation to the next one increases by 50%. In this way, we reduce the number of ordering calls and we save time, especially for simple models, which would be treatable even without re-ordering the variables. Instead, for more complex models, performing dynamic re-ordering may slow down the computation, but it can make the difference between completing and not completing the task.

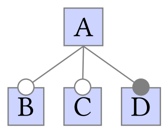
Merging constraints. Applying a cross-tree constraint c to the MDD M under construction is a complex operation requiring the execution of the *intersection* operation between M and the corresponding MDD representation of c. Given the computational cost of this operation, reducing the number of intersections and the complexity of each of them is advisable. For this reason, in our approach, we allow to merge n cross-tree constraints before applying them to M. More specifically, we sort all the constraints in terms of their cardinality using the *alternative sorting* algorithm[1]. Then, we merge n constraints into a single one, obtaining n time less cross-tree constraints.

The reason behind this decision is that when constraints with different cardinality are merged, those with higher cardinality are constrained by the lower cardinality ones. Consequently, as the procedure progresses, the resulting constraint becomes simpler compared to those initially involved. Furthermore, by decreasing the total number of constraints, we minimize the intersections with M. As a consequence, by reducing the number of intersections, we also decrease the need for variable re-ordering calls, which can be a computationally intensive process.

3.5 Merging `And`/`Or` groups

Another optimization that pushes the use of MDDs and tries to take full advantage of their use, consists in mimicking what has been done for the `Alternative` groups also for the other groups, and consequently reducing the number of variables in the MDD.

[1]With the alternative sorting, a list is modified in such a way that the first element is first maximum and second element is first minimum and so on.



(a) Example of an `And` group

x_A	Binary LSB → MSB	Invalid	Selected F. A	B	C	D
`NONE`						
`0`	000	×	×			
`1`	100	×	×	×		
`2`	010	×	×		×	
`3`	110	×	×	×	×	
`4`	001		×			×
`5`	101		×	×		×
`6`	011		×		×	×
`7`	111		×	×	×	×

(b) Mapping of the `And` group to the variable x_A

Figure 4: Merging `And` groups optimization

In the case of n features appertaining to an `And`/`Or` group with parent `p` and n children, we provide the user with the opportunity to merge the entire group into a single MDD variable x_p having 2^n integer values representing which feature combination is selected and a `NONE` value to represent that p is not selected. In this way, when x_p has the i-th bit set, it will mean having the i-th child feature selected (with the first feature corresponding to $i = 0$). We indicate it by using the bitwise operator $x_p \wedge (1 \ll i)$.

At this point, no feature will ever be translated as Boolean variable and the `isSel` function can be simplified and defined as:

$$\texttt{isSel}(f) = \begin{cases} x_f \neq \texttt{NONE} & f \text{ is not a leaf} \\ x_p \wedge (1 \ll i) & \text{if } f \text{ is a leaf and the i-th child of } p \end{cases}$$

Note that not all $2^n + 1$ values are feasible, since some of the n features may be mandatory in case of And group or at least a feature must be selected in case of Or groups. The visiting procedures for And/Or groups presented in Sect. 3.2 must be modified accordingly. As for the Alternative group, the fact that when a child is selected, then the parent is selected as well ($x_p \neq \texttt{NONE}$) is implicit. We have to add only the constraints:

$$\begin{cases} \texttt{isSel}(x_p) \rightarrow \texttt{isSel}(x_f) & \text{if f is mandatory in And group} \\ \bigvee_{i=1}^{n} \texttt{isSel}(c_i) & \text{for every Or group with parent } x_p \end{cases}$$

Moreover, since now every feature is both parent and child of a group (except for the root and leaves), and the function `isSel` is defined by considering the feature as parent, we have to add a new constraint for each Or and And group, if f is the i-th child of p

$$x_f \neq \texttt{NONE} \leftrightarrow (x_p \wedge (1 \ll i) > 0)$$

This additional constraint forces the parent variable to select exactly the value of the children, when it is selected as well.

An example of this optimization, in the case of an `And` group, is reported in Fig. 4. The FM shown in Fig. 4a is an `And` group with 3 children, and one of them (the feature `D`) is mandatory. While translating the `And` group into the MDD, a single variable x_A may be used. It has maximum 9 values: `NONE` is used when no feature (not even the parent one) is selected, `0` when only the parent feature is selected, `1` when only the parent and the first child feature is selected, and so on (see Fig. 4b). In this example, the feature `D` is mandatory so, among the combinations selecting `A`, only those having `D` selected are valid. At the end, the MDD variable x_A will have only 5 values, i.e., only the valid ones.

Type	×n	# F	And	Or	Alt.	Opt.	Mand.	# C	T/O
OnlyAND	10	200	×			×	×	80	0/840
OnlyOR	10	100		×				63	486/840
OnlyXOr	10	200			×			80	0/840
Mixed	10	200	×	×	×	×	×	80	0/840

Table 2: Summary of benchmarks used for the evaluation

This operation can be costly, since it requires also the rewriting of the constraints (i.e., all the checks on the features involved in an `And` group must be translated in checks on multiple values, depending on the bit of interest) and reducing the number of nodes may imply increasing the number of edges in the MDD. For this reason, we allow the user to set whether to enable or disable the optimization, and we set a threshold on the maximum number of children in an `And`/`Or` group eligible for this optimization.

4 EXPERIMENTS

In this section, we evaluate the proposed approach against synthetic benchmarks and models available in the literature. Tab. 2 reports the characteristics of synthetic benchmarks. They have been generated with the SPLOT Feature Model Generator proposed by [19] and have been converted in the xml format supported by FeatureIDE. In particular, in order to analyze the performance depending on the type of features, we have generated models having only `And` groups, only `Or`, only `Alternative` groups, and mixed groups. For each category, we have generated 10 benchmarks. Models in the *OnlyAND* category have been generated by using 50% probabilities both for mandatory and optional features; those in the *OnlyOr* category with 100% of `Or` features; those in the *OnlyXOr* category with 100% of `Alternative` features and, finally, those in the *Mixed* category have been generated by using 25% probability for mandatory, optional, `Or`, and `Alternative` features. Note that, in the case of models appertaining to the OnlyOR category, we have reduced the number of features to 100 and, consequently, also the number of constraints which result to be 63. This choice is made because of the higher cardinality and complexity of those models, potentially leading to a significant amount of timeouts.

Additionally, in Tab. 3 we report the FMs we have gathered from the literature and we have used to evaluate whether our approach was able to complete the computation of the number of configurations. We refer to those models as *industrial* FMs. We report the percentage of different group types in each industrial FM.

With these FMs, we aim to investigate the following RQs:

- RQ1 Does merging `And`/`Or` groups impact on the performance of the process?
- RQ2 Does the variable re-ordering impact on the performance of the process?
- RQ3 Does merging the constraints impact on the performance of the process?
- RQ4 Is our approach suitable for large-scale industrial FMs? How does it compare with a more classical BDD-based approach?

In the following, we present the experimental methodology and the answer to each research question.

System	# F	# C	Source	% Alt	% Or	% And
BerkeleyDB	76	20	[20]	34.78	17.39	47.83
axTLS	96	14	[21]	27.78	0.00	72.22
uClibc	313	56	[21]	47.83	0.00	52.17
uClinux-base	380	3, 455	[21]	98.61	0.00	1.39
BusyBox	631	681	[22]	4.68	0.00	95.32
FinancialServices	771	1, 080	[23]	67.93	3.27	28.80
Embtoolkit	1, 179	323	[21]	52.11	0.00	47.89
uClinux-distribution	1, 580	197	[21]	4.72	0.00	95.28
Automotive01	2, 513	2, 833	[20]	57.49	6.07	36.44
Linuxv2.6.33.3	6, 467	3, 545	[21]	4.23	0.21	95.56
Automotive02	18, 616	1, 369	[21]	69.18	6.49	24.33

Table 3: FMs from the literature, ordered by # F

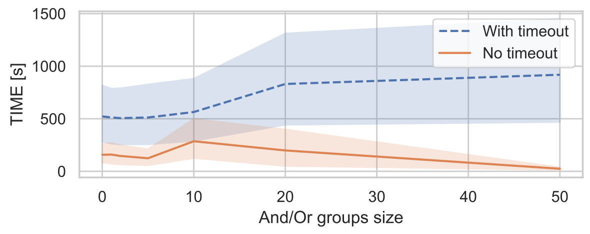
Experimental methodology

We here explain the experimental methodology we have adopted to evaluate the proposed approach through the 4 RQs previously introduced. The first three RQs aim at investigating whether the optimizations we have introduced do impact or not on the performance of the knowledge compilation process. For these RQs, we exploit the benchmarks introduced in Tab. 2. On the one hand, RQ1 investigates the impact of the And/Or merging optimization. For this reason, for its evaluation, we limit to the models having And/Or groups, namely those appertaining to the OnlyAND, OnlyOR, and Mixed categories. On the other hand, RQ2 and RQ3 evaluate optimizations on variable re-ordering and cross-tree constraints, which apply to all 40 synthetic benchmarks. For this reason, we use all benchmarks for their evaluation. We emphasize that we do not mix synthetic models with industrial ones. In the first three research questions, we only use synthetic benchmarks, while for RQ4 we use solely industrial ones. In this way, we are able to set the configuration parameters to be used in the case of industrial (and most critical) models by avoiding the bias introduced when tuning a method on the actual evaluation set.

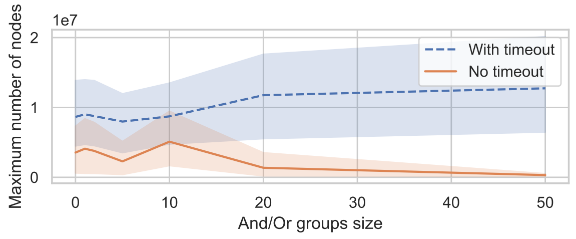
For every instance of test execution, we record the experimental setup, including the count of configurations, duration of computation, activation of the And/Or merging optimization, threshold for merging values within an And/Or group (the values 0, 2, 5, 10, 20, 50 are tested[2]), activation of variable re-ordering optimization, quantity of cross-tree constraints merged prior to their application to the MDDs (the values 1, 2, 5, 10, 20, 50 are tested), and the peak number of edges and nodes attained during the knowledge compilation phase. Note that the number of edges and nodes is used to measure the memory utilization, which is reported to be a critical aspect in the literature on using decision diagrams for knowledge compilation. In this way, for each of the 40 benchmarks we gather data from 84 executions, with different configurations. For each category of the synthetic benchmarks, Tab. 2 reports in the T/O column how many executions timed out in our experiments.

After having evaluated the impact of the three optimizations, in RQ4, we investigate the applicability of our approach on large-scale industrial FMs. Additionally, in RQ4 we compare the performance of the proposed approach with that of a more classical BDD-based knowledge compilation technique. The BDD-based tool, developed in Java, is based on the primitives offered by FeatureIDE and is available in our replication package.

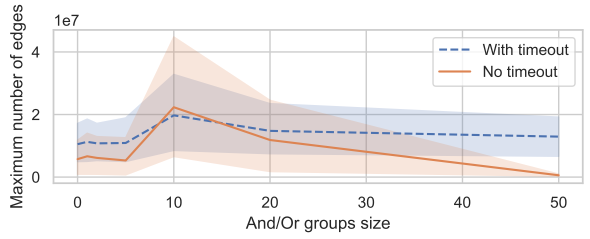
[2]The value 0 is only used when the And/Or merging optimization is disabled.



(a) Time



(b) Number of nodes



(c) Number of edges

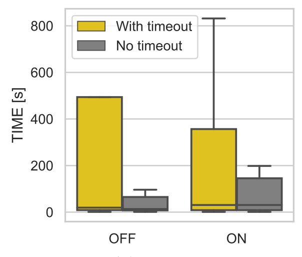
Figure 5: Merging And/Or groups optimization

We have performed the experiments on a Linux server running Ubuntu 18.04.6 LTS, with 2 CPUs Intel®Xeon®E5-2620 v4 @ 2.10 GHz, and 128 GB RAM. When working with industrial large-scale FMs, for which requiring a huge amount of time the timeout is likely, we have set a timeout of 24 hours ($8.64 \cdot 10^4$ seconds), while for synthetic benchmarks a 1-hour timeout ($3.6 \cdot 10^3$ seconds) has been set. In case of timeout, we set the time to a value higher than the maximum possible (3601 seconds), and the number of nodes and edges to $5 \cdot 10^7$ which is higher than the number of nodes and edges measured in the cases in which the computation did not time out. All scripts and data used in the presented experiments are available in our replication package at https://github.com/fmselab/FMConfigurationsCounter.

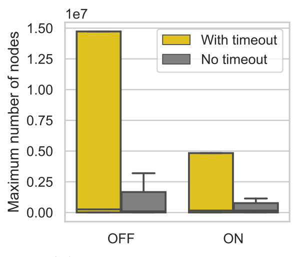
4.1 RQ1: And/Or groups optimization

In this RQ, we are interested in investigating whether merging And/Or groups impacts on the performance of the knowledge compilation process. Moreover, we analyze which threshold (i.e., the maximum size of And/Or groups to be merged in a single MDD variable) for the groups to be considered is the best fit.

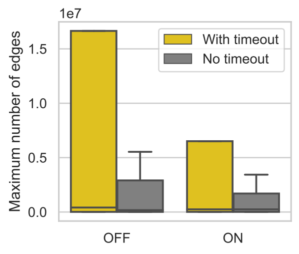
The results of our experiments are reported in Fig. 5. These results are average results, computed on 30 benchmarks having at least one And/Or group (those appertaining to the OnlyAND, OnlyOR, and Mixed categories). In general, we can see that introducing



(a) Time



(b) Number of nodes



(c) Number of edges

Figure 6: Dynamic variable re-ordering optimization

the And/Or groups merging optimization can be advantageous. Indeed, the time (Fig. 5a), number of nodes (Fig. 5b), and edges (Fig. 5c) when using the value 0 as threshold (meaning that the optimization is not enabled) are higher than that obtained when merging at least a small amount of children features. This consideration holds both when considering executions timing out or those finishing the computation. As expected, merging children features in And/Or groups leads to a reduction of the number of nodes (see the orange line in Fig. 5b) and this contributes in decreasing the time required by the knowledge compilation process (see the orange line in Fig. 5a).

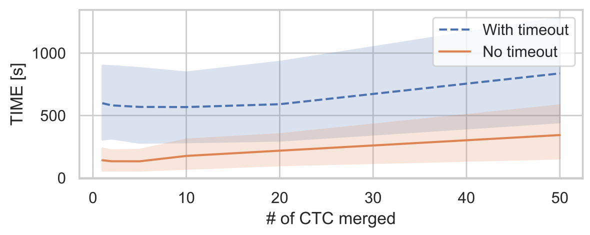
Moreover, when the optimization is enabled, values included in the range between 2 and 5 are the best ones. In fact, when the threshold is higher than 5, we have a consistent increase in time due to timeouts (see Fig. 5a).

Merging feature in And/Or groups when mapping them into their MDD representation can be advantageous. However, on the one hand, increasing the size of the groups to be considered reduces the number of nodes but, on the other hand, the complexity of the MDD increases. This implies higher memory utilization and time required for knowledge compilation, leading to the increase of the number of timeouts. For this reason, as confirmed by our experiments, we believe that this optimization is better used only for "small" groups and we suggest enabling the optimization by setting a threshold between 2 and 5.

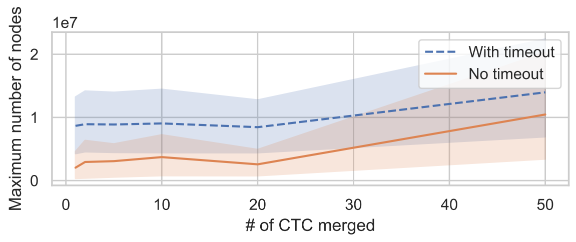
RQ1: And/Or groups optimization

4.2 RQ2: Variable re-ordering

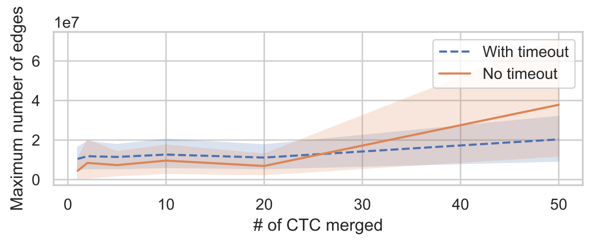
In this RQ, we are interested in investigating whether enabling the dynamic variable re-ordering optimization leads to enhanced performance w.r.t. always leaving the same variable order. To do this, we analyze the results of our experiments on all the benchmarks reported in Tab. 2.



(a) Time



(b) Number of nodes



(c) Number of edges

Figure 7: Cross-tree Constraint Merging optimization

Our experimental findings, detailed in Fig. 6 and excluding outliers for a better visualization, encompass analyses that both include and exclude instances resulting in timeouts. Concentrating on the results that incorporate timeouts (shown in yellow in Fig. 6), it becomes apparent that activating the optimization reduces the occurrence of timeouts (see the decrease in time in Fig. 6a), as well as reductions in the number of nodes (Fig. 6b) and edges (Fig. 6c). Nevertheless, as mentioned previously in Sect. 3.4, dynamically reordering variables within an MDD is a resource-intensive task. Consequently, when the assessment is restricted to simpler models and excludes timeouts (depicted in gray in Fig. 6), the time required for knowledge compilation with the optimization enabled is higher (Fig. 6a). However, the number of nodes and edges (Fig. 6b and Fig. 6c) remains lower with the optimization in place.

Dynamic re-ordering of the MDD variables is an effective way to reduce the number of nodes and edges, potentially leading to lower knowledge compilation time, especially for more complex FMs. For this reason, as confirmed by our experiments, we believe that this optimization should be always used, except for very simple models.

RQ2: Variable re-ordering

4.3 RQ3: Cross-tree constraint merging

In this RQ, we investigate whether merging the cross-tree constraints before their intersection with the MDD representing the

System	# Valid Conf.	Time [s] MDD	Time [s] BDD
BerkeleyDB	$4.03 \cdot 10^{9}$	0.01	0.07
axTLS	$8.26 \cdot 10^{11}$	0.01	0.09
uClibc	$1.66 \cdot 10^{40}$	6.13	17.15
uClinux-base	$2.62 \cdot 10^{22}$	0.39	0.59
BusyBox	$2.06 \cdot 10^{201}$	205.60	–
FinancialServices	$9.74 \cdot 10^{13}$	3.09	235.56
Embtoolkit	$5.13 \cdot 10^{96}$	2.18	21.70
uClinux-distribution	$4.07 \cdot 10^{409}$	157.18	–
Automotive01	$5.42 \cdot 10^{217}$	$8.29 \cdot 10^{4}$	–
Linuxv2.6.33.3	–	–	–
Automotive02	$1.78 \cdot 10^{1534}$	$4.68 \cdot 10^{4}$	–

Table 4: Cardinality and results on industrial FMs

FM is advantageous. To do this, we analyze the results of our experiments on all the benchmarks reported in Tab. 2.

As in the previous RQs, our experimental results, detailed in Fig. 7, includes analyses that both takes into account (dashed blue line) and exclude (orange line) instances resulting in timeouts. Interestingly, merging a large amount of cross-tree constraints (higher than 20) seems to be counterproductive as the number of nodes and edges increases significantly, as well as the time. By looking to the results including timeouts, it can be noticed that for a reduction factor up to 20, there is a slight decrease in time (see Fig. 7a) and number of nodes (see Fig. 7b), which implies a reduction of timeouts. The reduction in terms of number of nodes (also in the case of experiments excluding the timeouts) is the most important one, as it is directly correlated with the increase in the probability of the knowledge compilation to complete.

Merging cross-tree constrains has a very small impact on the performance of the counting algorithm. Using a reduction factor greater than 20 is discouraged, while our experiments show slight reduction in terms of nodes when the reduction factor is set to 20. Thus, we suggest to carefully evaluate whether to enable the optimization depending on the complexity of the constraints and, in any case, with a reduction factor lower or equal to 20.

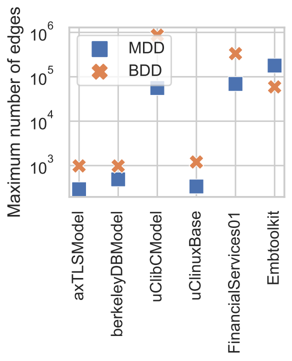
RQ3: Cross-tree Constraint merging

4.4 RQ4: Analysis on large-scale industrial FMs

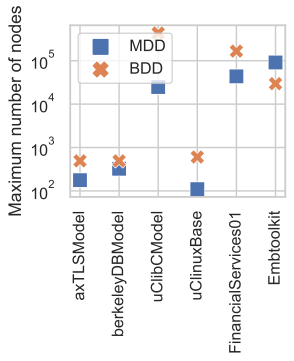
In this research question, we are interested in investigating whether the proposed solution is suitable for industrial SPLs, i.e., if it can be applied to compute their number of valid configurations. The experiments have been carried by merging 20 cross-tree constraints per time, enabling the variable re-ordering, and optimizing `And`/`Or` groups with maximum 5 children (see Sect. 3.4). The obtained results are reported in Tab. 4.

In general, from the obtained results we can see that the proposed solution, based on MDDs, allowed to compute the number of valid products for all the analyzed industrial FMs except for Linux, on which no technique in the literature has succeeded yet though, within the set timeout of 24 hours.

Interestingly, our results show that the performance of the approach does not only depend on the complexity (in terms of number of features or constraints) of the FM under analysis, but more on its structure. For example, The BusyBox model has fewer features and fewer constraints than the FinancialServices one. However, the time required to complete the computation of the number of valid configurations for the former is significantly higher than that required for the latter. As reported in Sect. 3.1, the `alternative` groups are those giving the most advantage to the MDD representation. Indeed, by inspecting the two FMs, as also reported in Tab. 3, we have discovered that the percentage of `alternative` groups in the BusyBox model is notably lower than that of the FinancialServices and this justifies the difference in performance in the two examples. The same consideration holds between Automotive01 and Automotive02, or between Linuxv2.6.33.3 and Automotive02.



(a) Number of edges



(b) Number of nodes

Figure 8: Number of nodes and edges produced by BDDs and MDDs

The superiority of MDDs over BDDs is evident from the comparison of completion times between the two methods, as detailed in Table 4. Across all industrial FMs, the MDD-based approach consistently outperforms its BDD-based counterpart. Moreover, the former experienced timeouts in only one instance, whereas the latter encountered timeouts in 5 out of 11 models and crashed due to the higher memory demands. This consideration is confirmed by Fig. 8, where the number of nodes and edges produced by the two approaches, on the FMs where computation completed by both of them, are compared. Indeed, we can see that for almost all models (except for Embtoolkit), the number of nodes and edges created within the MDDs is consistently lower than that within the BDDs.

The proposed MDD solution is suitable for medium-scale models as it allows the computation of the FM cardinality within few minutes. In the other cases, when the number of features and constraints exceeds the thousands, its suitability is limited and the time required to complete the computation rapidly grows. However, we have observed a strong influence of the FM structure on the performance of the approach: the higher the percentage of `alternative` features (see Tab. 3), the higher the advantage of using MDDs. Additionally, comparing the MDD-based technique with the BDD-based one,

RQ4: Large-scale models

our experiments have shown that the former consistently produces fewer nodes and edges (indicating a lower memory consumption) and completes the task in a shorter time.

RQ4: Large-scale models

5 THREATS TO VALIDITY

In this section, we discuss the threats to validity [24] and all the strategies we have undertaken to mitigate them.

Internal validity refers to the fact that different outcomes obtained with the analyzed approach and optimizations are actually caused by the different tool configuration themselves and by the way the experiments were carried out, and not by methodological errors. To mitigate this risk, we have carefully checked the code of our tool and experiments to see if there could be other factors that have caused the outcome, such as implementation errors. A possible threat to the *construct validity* comes from the assumption that our optimization and mapping strategies are suitable to maintain unaltered the number of configurations of FMs. To mitigate this risk, we have carefully checked our results against the literature in order to find other work dealing with the same FMs [10].

External validity is concerned with whether we can generalize the results outside the scope of the presented study. In particular, one threat to external validity refers to the case studies we have used in the experiments. For what concerns the synthetic benchmarks, we have tried to generate generic benchmarks, with varying features and characteristics. Instead, for what concerns the industrial FMs, we have tried to collect the same examples used by other research works investigating on the same problem, i.e., the knowledge compilation for SPLs. However, this work is just a preliminary attempt of introducing MDDs and suitable mapping strategies in the context of knowledge compilation. We believe that further experiments are needed to generalize the conclusions and to better investigate the use of MDDs on a greater set of examples.

One of the biggest threat to validity, refers to whether using MDDs instead of BDDs is truly beneficial. Some theoretical considerations (like what is shown in Fig. 2) and our experiments (see RQ4) provide evidence that MDDs are better than BDDs. Our results show that MDDs are more compact than BDDs, especially while dealing with `Alternative` groups or while compacting small groups in fewer MDD variables. However, we use a library for BDDs which may not implement them most efficiently. Since BDDs are far more well-studied than MDD, their use could take advantage of years of applied research. Thus, we plan to better investigate on how the two approaches compare in future work. Similarly, most works counting valid configurations in FMs use #SAT or d-DNNFs. We plan to compare our approach with them in future work.

6 RELATED WORK

The problem of knowledge compilation for Software Product Lines is a well-known and investigated issue in SPL engineering, being np-hard. Especially for large-scale systems computing the number of valid configurations is difficult and this is the reason why a challenge on knowledge compilation has been proposed at SPLC [5].

From the analysis of the literature, the problem of defining the number of valid configurations for SPLs have been tackled by using decision diagrams, mostly Binary Decision Diagrams (BDDs), or solvers. In [25], the authors presented the FAMA model analyzer integrating BDDs, SAT Solvers, and CSP solvers for the computation of the FM cardinality. Concerning the use of BDDs, their performance has been evaluated in [26]. The authors analyzed javaBDD (the same libary that we employ in the implementation used in RQ4-Sect. 4.4 when comparing with the MDD-based implementation), Logic2BDD, BuDDy, and CUDD against a set of FMs including those we have used in Sect. 4. They reported to be able to build BDDs for three of the 18 considered real-world FMs, many of them below the threshold of 2,000 features, indicating scalability issues for this approach. Thus, the authors of [26] claimed that the performance of BDD libraries had been significantly overestimated by previous research. Similarly, #SAT solvers have been used in [10, 27]. The results obtained by the authors, on the same large-scale industrial models we have used in our evaluation, report the same number of configurations but, in general, the cardinality is computed in a shorter time. Other approaches, such as [28] have been developed by exploiting the DPLL procedure, but require the model to have constraints written in CNF and this may influence their complexity.

To the best of our knowledge, no attempt of using MDDs, with the same optimizations we have proposed in our paper, has been previously published.

7 CONCLUSIONS

Counting the valid configurations in an SPL is a challenging process that can be particularly useful for developers and stakeholders to allow economical estimation, uniform testing, and to detect possible design errors. In this paper, we have proposed an approach, based on Multi-Valued Decision Diagrams (MDDs), which implements several optimizations to reduce the memory consumption and make the computation more efficient. We have evaluated the effectiveness of our approach against two datasets: one composed by synthetic benchmarks and one by large-scale industrial models. Our experiments have shown that MDDs allows for building the set of products in a more efficient way than the classical binary decision diagram-based approaches, and that using the proposed optimizations positively impacts on the performance of the knowledge compilation process. More specifically, using MDDs is very effective when dealing with medium-high complex FMs, while some shortcoming has been revealed for more complex models.

As future work, we are investigating possible algorithmic improvements, such as avoid representing all mandatory features and change accordingly the cross-tree constraints to reduce the number of MDD variables. Moreover, we are trying to structure the MDD construction process in such a way that multi-thread approaches can be applied. We believe that working in these two directions will make the proposed approach suitable to compute the number of configurations of larger models, such as for Linux, or to speed up the counting operation for other large-scale models.

Acknowledgment. The work of Andrea Bombarda was supported by PNRR - ANTHEM (AdvaNced Technologies for Human-centrEd Medicine) - Grant PNC0000003 – CUP: B53C22006700001 - Spoke 1 - Pilot 1.4. The work of Angelo Gargantini was supported by project SERICS (PE00000014), under the MUR NRRP funded by the EU - NextGenerationEU.

REFERENCES

[1] C. Sundermann, E. Kuiter, T. Heß, H. Raab, S. Krieter, and T. Thüm, "On the benefits of knowledge compilation for feature-model analyses," *Annals of Mathematics and Artificial Intelligence*, Nov. 2023. [Online]. Available: http://dx.doi.org/10.1007/s10472-023-09906-6

[2] M. Mendonca, A. Wasowski, K. Czarnecki, and D. Cowan, "Efficient compilation techniques for large scale feature models," in *Proceedings of the 7th International Conference on Generative Programming and Component Engineering*, ser. GPCE '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 13–22. [Online]. Available: https://doi.org/10.1145/1449913.1449918

[3] C. Sundermann, T. Thüm, and I. Schaefer, "Evaluating #sat solvers on industrial feature models," in *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*, ser. VaMoS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3377024.3377025

[4] P. Bourhis, L. Duchien, J. Dusart, E. Lonca, P. Marquis, and C. Quinton, "Reasoning on feature models: Compilation-based vs. direct approaches," 2023. [Online]. Available: https://arxiv.org/abs/2302.06867

[5] T. Thüm, "A bdd for linux?: the knowledge compilation challenge for variability," in *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*, ser. SPLC '20. ACM, Oct. 2020. [Online]. Available: http://dx.doi.org/10.1145/3382025.3414943

[6] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-021, 1990. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231

[7] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake, "A classification and survey of analysis strategies for software product lines," *ACM Computing Surveys*, vol. 47, no. 1, pp. 1–45, Jun. 2014. [Online]. Available: https://doi.org/10.1145/2580950

[8] A. Bombarda, S. Bonfanti, and A. Gargantini, "Testing the evolution of feature models with specific combinatorial tests," in *2024 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2024. [Online]. Available: https://doi.org/10.1109/icstw60967.2024.00025

[9] ——, "On the reuse of existing configurations for testing evolving feature models," in *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume B*. ACM, Aug. 2023. [Online]. Available: https://doi.org/10.1145/3579028.3609017

[10] C. Sundermann, T. Heß, M. Nieke, P. M. Bittner, J. M. Young, T. Thüm, and I. Schaefer, "Evaluating state-of-the-art #sat solvers on industrial configuration spaces," *Empirical Software Engineering*, vol. 28, no. 2, Jan. 2023. [Online]. Available: http://dx.doi.org/10.1007/s10664-022-10265-9

[11] S. V. Amari and L. Xing, *Binary Decision Diagrams and Extensions for System Reliability Analysis*. wiley, 2015.

[12] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*, 12th ed. Addison-Wesley Professional, 2009.

[13] R. E. Bryant, *Binary Decision Diagrams*. Cham: Springer International Publishing, 2018, pp. 191–217. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8_7

[14] J. Babar and A. Miner, "Meddly: Multi-terminal and edge-valued decision diagram library," in *2010 Seventh International Conference on the Quantitative Evaluation of Systems*. IEEE, Sep. 2010. [Online]. Available: http://dx.doi.org/10.1109/QEST.2010.34

[15] J. Meinicke, T. Thm, R. Schrter, F. Benduhn, T. Leich, and G. Saake, *Mastering Software Variability with FeatureIDE*, 1st ed. Springer Publishing Company, Incorporated, 2017.

[16] B. I. Tuleuov and A. B. Ospanova, *GMP (GNU Multiprecision Library)*. Apress, 2024, p. 131–148. [Online]. Available: http://dx.doi.org/10.1007/978-1-4842-9563-2_12

[17] R. Rudell, *Dynamic Variable Ordering for Ordered Binary Decision Diagrams*. Springer US, 2003, p. 51–63. [Online]. Available: http://dx.doi.org/10.1007/978-1-4615-0292-0_5

[18] F. Somenzi, "Efficient manipulation of decision diagrams," *International Journal on Software Tools for Technology Transfer*, vol. 3, no. 2, p. 171–181, May 2001. [Online]. Available: http://dx.doi.org/10.1007/s100090100042

[19] M. Mendonca, A. Wąsowski, and K. Czarnecki, "Sat-based analysis of feature models is easy," in *Proceedings of the 13th International Software Product Line Conference*, ser. SPLC '09. USA: Carnegie Mellon University, 2009, p. 231–240.

[20] FeatureIDE, "Official featureide github repository," [Online; accessed 20-03-2024]. [Online]. Available: https://github.com/FeatureIDE/FeatureIDE

[21] A. Knüppel, T. Thüm, S. Mennicke, J. Meinicke, and I. Schaefer, "Is there a mismatch between real-world feature models and product-line research?" in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE'17. ACM, Aug. 2017. [Online]. Available: http://dx.doi.org/10.1145/3106237.3106252

[22] T. Pett, S. Krieter, T. Runge, T. Thüm, M. Lochau, and I. Schaefer, "Stability of product-line samplingin continuous integration," in *Proceedings of the 15th International Working Conference on Variability Modelling of Software-Intensive Systems*, ser. VaMoS '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3442391.3442410

[23] T. Pett, T. Thüm, T. Runge, S. Krieter, M. Lochau, and I. Schaefer, "Product sampling for product lines: The scalability challenge," in *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, ser. SPLC 2019. ACM, Sep. 2019. [Online]. Available: http://dx.doi.org/10.1145/3336294.3336322

[24] R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research - an initial survey," in *SEKE*, 2010.

[25] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés, "Fama: Tooling a framework for the automated analysis of feature models," in *First International Workshop on Variability Modelling of Software-intensive Systems*, p. 129.

[26] T. Heß, C. Sundermann, and T. Thüm, "On the scalability of building binary decision diagrams for current feature models," in *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume A*, ser. SPLC '21. ACM, Sep. 2021.

[27] A. Kübler, C. Zengler, and W. Küchlin, "Model counting in product configuration," *Electronic Proceedings in Theoretical Computer Science*, vol. 29, p. 44–53, Jul. 2010. [Online]. Available: http://dx.doi.org/10.4204/EPTCS.29.5

[28] D. Fernandez-Amoros, R. Heradio, J. A. Cerrada, and C. Cerrada, "A scalable approach to exact model and commonality counting for extended feature models," *IEEE Transactions on Software Engineering*, vol. 40, no. 9, p. 895–910, Sep. 2014. [Online]. Available: http://dx.doi.org/10.1109/tse.2014.2331073