Efficient Solutions for Anomaly Explanations in Feature Models

Cristian L. Vidal-Silva
Universidad de Viña del Mar, Viña del Mar, Chile
cvidal@uvm.cl

José A. Galindo
University of Seville, Seville, Spain
jagalindo@us.es

David Benavides
University of Seville, Seville, Spain
benavides@us.es

ABSTRACT

A feature model represents common and variant features at different levels of abstraction for the products of a Software Product Line (SPL). The Automated Analysis of Feature Models (AAFM) is a computer-assisted task for extracting information from features modes. Existing AAFM solutions such as FMDiag and FlexDiag (an FMDiag extension) efficiently diagnosis on normal-size feature models, but those solutions are inefficient for the analysis and diagnosis on existing large-scale feature models. We propose Parallel FMDiag (PFMDiag) as an FMDiag extension to apply parallelism on the divide section of the divide-and-conquer approach for getting more efficient diagnosis results. We provide an implementation of PFMDiag in the FaMa framework, a tool for the AAFM that already supports FMDiag, and utilize traditional FMDiag and PFMDiag on the product diagnosis over a set of normal-size and large-scale feature models, that is, feature models containing 10 to 100 and 1000-10000 features respectively, and compare their algorithmic performance to highlight the quality preservation and performance improvements of PFMDiag.

1. INTRODUCTION

0: Contextualizar el Tema - (P1): VIs y SPL = Product Family - FM y Product Family - (P2): Large-Scale FMs and AAFM

Variability modeling and management are core engineering activities in the development process of Variability Intensive Systems (VIS) and Software Product-Lines (SPL). Each VIS and SPL exemplifies systems which share and differ in their features setting (a product family). Kang [28] introduced Feature Model (FM), a hierarchical tree-like representation of common and varying properties for a software product family. The selection of features for the products definition circumscribes a family of valid configuration: a configuration space for the products definition according to hierarchical relations and cross-tree constraint among features in the FM [14]. FMs impulse the tailored-made, reduced cost, and quality product settings in a product family [11] [32] because, such as Trinidad et al. remark [40], FMs are bases for the systematic component reuse in product configurations.

Large-scale systems such as video-surveillance systems [25], Linux operating systems [37] [17], and distributions of Linux such as Debian [26] contain a high number of relations and constraints for the product configurations. The analysis of those FMs demand for computer-assisted tools. The Automated Analysis of Feature Models (AAFM) permits extracting information from FMs using computer-aided mechanisms [14].

3: Problem: (P1) Configuraciones de usuario y complejidad de AAFM

Users of evolving and large-scale SPL and VIS usually desire product configurations tailored to their individual requirements and unintentionally break configuration rules in the configuration process. Errors detection and diagnosis explanations are absolutely necessary in such situations, but, just as Pereira et al. [32] argue, the errors detection and diagnosis for non-valid features configuration on large-scale FMs are examples of computing complex and expensive tasks. A quick response solution for the analysis and diagnosis of the products validation on large-scale FMs would permit efficient configuration, testing, and systems improvements.

3: (P1) Soluciones existentes - (P2): nuestra propuesta - (P3): research line

White et al. [41], Felferning et al. [22], Benavides et al. [13] describe the transformation process of feature models into Constraint Satisfaction Problem (CSP) to look for the automated diagnosis and analysis solution in the detection and explanation process of inconsistent constraints for knowledge-based configuration. Felfernig et al. [22] and [19] present model-based diagnosis solutions for the feature model analysis and diagnosis (minimal explanation), FastDiag and FlexDiag algorithms respectively, divide-and-conquer tasks for the leading diagnosis. Liang et al. [30] give an explanation and experimental support for the scalability of SAT solvers on large real-world feature models.

In this article, we propose and validate Parallel FMDiag, a parallel-recursive and more efficient version of FastDiag [22]. We exemplify the Parallel FMDiag diagnosis process on the products of a small case study, and verify its applicability on the product configurations of FMs from three sources: the SPLOT repository [7], generated by the Betty tool [40], and the Debian Linux distribution. We define and apply metrics to compare and analyze the computing performance and quality results of our solution and traditional FMDiag. This article demonstrates that Parallel FMDiag represents an improvement for the AAFM on the analysis of product configurations of large-scale FMs.

This research work is in the line of the development of intelligent quality assurance mechanisms for configuration knowledge bases. Our contributions are the following:

- Proposing Parallel FMDiag, a more efficient version of FMDiag, for the diagnosis and explanations on product configurations of large-scale FMs.
- Demonstrating the Parallel FMDiag computing efficiency and quality preservation concerning its previous version for the diagnosis and explanations on product configurations of large-scale FMs.

Hereafter, this article organizes as follows: first, a motivating scenario for our AAFM solutions; second, a preliminary background about feature models, AAFM, and diagnosis solutions; third, a description of our solution Parallel FMDiag and an application example of a small feature model diagnosis; fourth, case studies definition and the application results of Parallel FMDiag; fifth, a set of related works; and finally, conclusions and future work.

2. MOTIVATING SCENARIO

A Feature Model (FM) represents the set of valid product configurations for a SPL, that is, the set of complete and valid features selection for the products definition [14]. Large-scale FMs contain a high number of packages able to select (features), hierarchical and cross-tree relations among packages, and also evolving changes in packages. For example, the Linux configuration process is a large-scale FM example of a VIS: a great number of installable packages (features) and a large number of feature relations (parent-child and cross-tree).

Product configurations of large-scale FMs are non-simple tasks mainly for their dynamism and complexity for the possible constraints among features. The installation process of Linux [31] and any of its distributions such as Debian [1] are instances of evolving, complex, time-demanding and error-prone tasks. Works about the analysis of variability models for Linux and related products already exist. For example, Galindo et al. [26] describe and use variability models for the Debian derivative, whereas Shen et al. [37] and Dintzner et al. [17] explain and utilize a variability model for the Linux kernel analysis.

The main goal of this research is to propose and apply an effective and more efficient AAFM solution for the diagnosis and explanation on large-scale FMs such as those for Linux derivatives. Figure 1 illustrate how Linux leads to many derivatives or 1st level products, and each derivative can guide to different versions or 2nd level products (D_1V_1, D_1V_2, ...). A high number of selectable packages (features) exist in the base system and each resulting derivative.

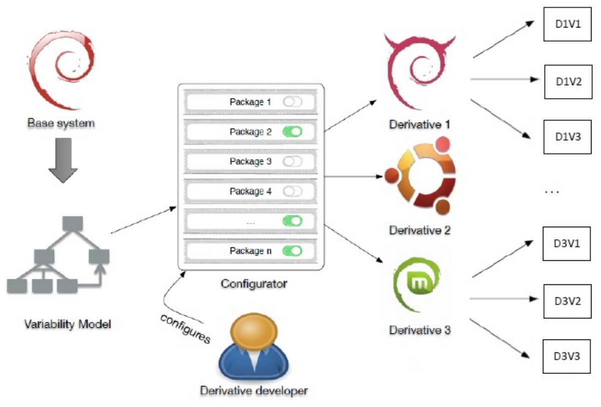


Figure 1: AAFM of Linux scalability challenge

The FM of Figure 2 illustrates a base feature model for an hypothetical Debian derivative that requires product configurations for installing processes. This model specifies a set of features relevant to configure the installation of a Debian operating system along with constraints between the features, some constraints are faulty. In the first two levels, this model indicates that a Debian configuration always must include texteditor, bash and gui features; and a Debian configuration does not necessarily include games. This model also indicates a cross-tree constraint: feature openoffice.org-gtk-gnome requires feature gnome, that is, the selection of openoffice.org-gtk-gnome needs the selection of gnome for a valid configuration. Next section gives more detail about feature models, feature relations and constraints in a CSP context.

A derivative generation process exists for Linux, just to make suitable any derivative on defined scenarios. For example, Ubuntu [10] is a derivative of the Debian derivative, Guadalinex [3] is a version of Ubuntu optimized to work for the local administration of Andalusia, Steam [8] is an Ubuntu version targeting videogamers, Qimo [6] is an Ubuntu version specialized for kids, and Ubuntu CE [9] is an Ubuntu version for religion purposes. The derivative generation process is like the production of large-scale software instances (products): a valid selection of installable packages (features) from the base Linux derivative (feature model).

All users of Linux are able to generate new derivatives from any existing base Linux derivative. Different tools currently exist to generate new versions for Ubuntu such as the Ubuntu Customization Kit (UCK) [5], the Linux Respin, the Systemback, and the Linux Live Kit tools [4].

The process of building a new derivative version after the packages selection in Ubuntu requires several hours (more than 200 minutes) [4] [5]. Figure 3 illustrates the derivative generation process cycle: users select packages for a derivative, spend time waiting for the building process, inconsistencies and warnings appear, users solve possible issues and repeat the whole process. The second goal of this research is to apply our solution proposal on the automated analysis of large products configuration of large-scale feature models such as Linux derivatives and their versions to verify their validity according to their base feature model.

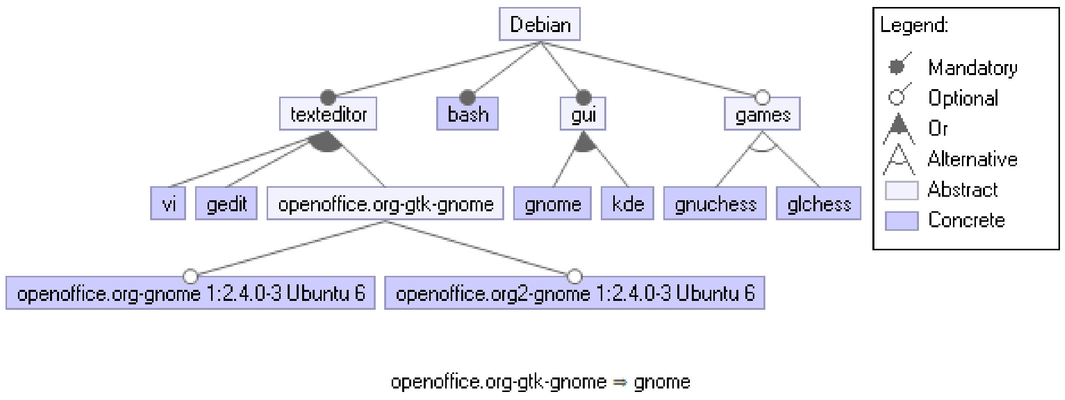


Figure 2: Feature model example of the Debian derivative of Linux

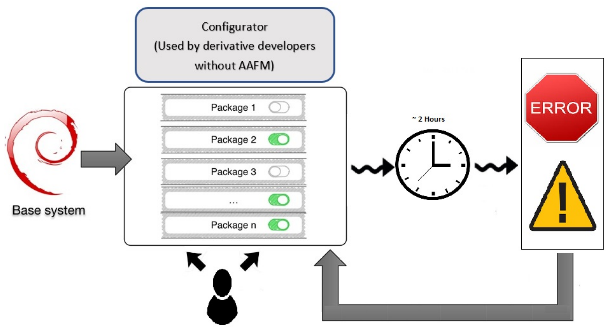


Figure 3: Automated analysis of a derivative scalability challenge

Non-valid configurations for the FM of figure 2 are the products: $p_1 = \{Debian, texteditor, bash, games, vi, gedit, gnuchess, glchess\}$ for selecting both exclusive games options and $p_2 = \{Debian, texteditor, gui, games, openoffice.org-gtk-gnome, kde, glchess\}$ for discarding the mandatory and required features *bash* and *gnome*, respectively; whereas $p_3 = \{Debian, texteditor, bash, gui, vi, gedit, gnome, kde\}$ is a valid-configuration example. Next section describes the semantic of FM relations.

Multi-core processors are widespread for computing architectures such as servers, desktops, laptops, and tablets [2], and parallel computing solutions can run in parallel using different cores. We propose and implement PFMDiag in the FAMA tool suite [15] [39] to work on large-scale FMs for a more efficient diagnosis about the validity of products configurations.

3. PRELIMINARIES

In this section, we present the central concepts of feature models and review some algorithmic solutions for the feature models diagnosis.

3.1 Feature Models

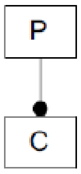
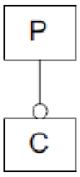
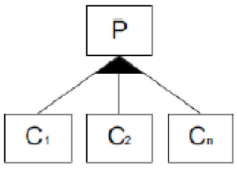
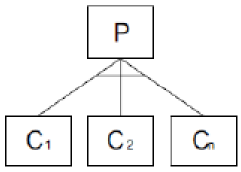
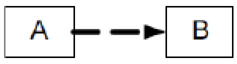
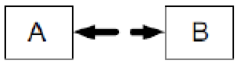
A FM [28] using a tree-like hierarchical structure represents and organizes distinctive characteristics (features) for configurable products concerning common and variable features along with their relations in a domain [18]. FMs can represent all products of instances of SPL and VIS [14] [23]. Such as Benavides et al. argue [14] different FMs dialects exist like basic FMs models, cardinality based FMs and extended FMs using feature attributes. This paper uses basic FMs.

Structurally, a FM starts with the root feature from which the remaining features emerge. A FM support two types of relationships between features: structural-parental associations and non-parental cross-tree constraints [14] [18]. Table 1 summarizes main relations in a traditional FM [14], and we present a brief description of those relations in the following lines:

- Parent-Child Relationships:
  - Mandatory: A mandatory relationship states that a parent feature requires its child.
  - Optional: An optional relationship states that a child feature may be or not present (it is does not required by its parent feature).
  - Set: A defined number of features of a set of children features (sub-features) are selectable for products when their parent is selected. This number of features is given by a cardinality relation [x, y], for x <= y and y <= number of child features in the set. Two cases are XOR (Alternative) and Or sets.
    - XOR: Only one child feature must be present. The associated cardinality relation is [1, 1] in this case, t.
    - Or: At least one child features must be present. The cardinality relation is [1, n] in this case (n corresponds to the number of child features).
- Non-parental Cross-Tree Constraints.
  - Requires: For two features A and B, if A requires B then the presence of A implies the presence of B in a product.
  - Excludes: For two features A and B, if A excludes B then A and B cannot be present in the same product.

Such as Felfernig et al. [22] describe, a FM is formally representable as a CSP FM configuration task, and each configuration as a FM configuration:

Table 1: Feature model relations

Unary relations	
mandatory	optional
	
Set relations	
optional (or)	alternative (xor)
	
Cross-tree constraints	
requires	excludes
	

- FM configuration task.
  1. A FM F is defined in function of all its features $F = \{f_1, ..., f_n\}$.
  2. Each feature f_i respects a domain $D = \{true, false\}$.
  3. A set of constraint $C = CF \cup CR$; CF is the set of constraint of the FM and CR represents a set of user requirements.
- feature model configuration.
  1. A FM configuration corresponds to a complete assignment of the variables f_i in F for a given FM configuration tasks.

Associations between features logically represent a configuration knowledge-base set [22]. Next, we present the CSP-based representation for the FM of figure 2 (the FM configuration task = $(F, D, C = CF \cup CR)$):

- $F = \{Debian, texteditor, bash, gui, games, vi, gedit, openoffice.org{-}gtk{-}gnome, gnome, kde, gnuchess, glchess, openoffice.org{-}gnome1:2.4.0{-}3Ubuntu6, openoffice.org2{-}gnome1:2.4.0-3Ubuntu6\}$
- $D = \{dom(Debian) = \{true, false\}, dom(texteditor) = \{true, false\}, dom(bash) = \{true, false\}, dom(gui) = \{true, false\}, dom(games) = \{true, false\}, dom(vi) = \{true, false\}, dom(gedit) = \{true, false\}$, dom(openoffice.org-gtk-gnome)= $\{true, false\}$, $dom(gnome) = \{true, false\}, dom(kde) = \{true, false\}, dom(gnuchess) = \{true, false\}, dom(glchess) = \{true, false\}, dom(openoffice.org - gnome1:2.4.0 - 3Ubuntu6) = \{true, false\}, dom(openoffice.org2{-}gnome1:2.4.0 - 3Ubuntu6) = \{true, false\}\}$
- $CR = \{c_0 : Debian = true\}$
- $CF = \{c_1 : Debian \leftrightarrow texteditor, c_2 : Debian \leftrightarrow bash, c_3 : Debian \leftrightarrow gui, c_4 : games \rightarrow Debian, c_5 : texteditor \leftrightarrow vi \vee gedit \vee$ openoffice.org-gtk-gnome, $c_6 : gui \leftrightarrow gnome \vee kde, c_7 : games \leftrightarrow gui, c_8 : (gnuchess \leftrightarrow \neg glchess \wedge games) \wedge (glchess \leftrightarrow \neg gnuchess \wedge games), c_9 : openoffice.org{-}gtk{-}gnome \leftrightarrow openoffice.org{-}gnome1:2.4.0{-}3Ubuntu6 \vee openoffice.org2{-}gnome1:2.4.0{-}3Ubuntu6, c_{10} : openoffice.org{-}gtk{-}gnome \rightarrow gnome$

3.2 Existing feature model diagnosis solutions

The following lines first mention existing solutions for the diagnosis and detection of errors over time, and then thoroughly describe the HSDiag, FMDiag, and FlexDiag solutions:

3.2.1 Errors-Detection and Diagnosis solutions in time

- Reiter [33] introduce the HSDAG algorithm, a Hitting Set Directed Acyclic Graph for diagnosis using a breadth-first search on conflict sets. Bakker [12] apply a model-based diagnosis to identify the set of relaxable constraint on conflict sets in a CSP context. Looking for a computing performance improvement regarding the previous conflict detection solutions, Junker [27] proposes QUICKXPLAIN, a divide-and-conquer approach to significantly accelerate the conflict detection on over-constrained problems. In that context, Felfernig et al. [20] propose FastDiag to look for an efficient diagnosis solution on customer requirements, and a consistent configuration knowledge base, by the use of the QUICKXPLAIN reasoning structure. Felfernig et al. [22] review and apply FastDiag (FMDiag) on the feature model diagnosis, whereas Felfernig et al. [19] [21] describe FlexDiag, a FastDiag extension, for anytime diagnosis scenarios and apply it for the FM diagnosis.

3.2.2 Diagnosis Solutions HSDiag, FastDiag, FMDiag and FlexDiag

- HSDAG is a complete breadth-first algorithm to determine all minimal diagnosis in the analyzed set. HSDAG supports the determination of minimal diagnosis with a minimal cardinality (lowest number of constraints). Felfernig et al. [18] indicate that HSDAG functioning structures is like repeatedly activating a conflict detection algorithm to analyze the possibilities for resolving each obtained conflict, that is, HSDAG requires executing many times a conflict detection solution to work on each of its results. This approach, although complete, is an expensive computing solution example for the large-scale models' analysis.
- FastDiag is a direct-diagnosis algorithm to determine a minimal or preferred diagnosis for a given ranking of preferences on a solution set [20]. The main idea of FastDiag is to diagnosis on a set of inconsistent customer requirements S on a consistent configuration knowledge base AC that becomes inconsistent after combining with the customer requirements, $AC = AC \cup S$. FastDiag evaluates the consistency of the set AC in each iteration that depends of S. If the current AC is not

consistent, FastDiag determines if S, the inconsistency source, corresponds first to a unary set to return that set. If S is not unary yet, it is divided in the sets S_2 and S_1, just to evaluate what is the inconsistency source, in that order. If S_2 were directly the inconsistency cause, S_1 is not more considered; otherwise, FastDiag takes into account additional rules of S_1 looking for a minimal diagnosis.

Felfernig et al [22] highlight that FastDiag is a direct-diagnosis example to determine a minimal diagnosis without a preceding conflict detection because it works on a conflict-independent search strategy [18]. That property represents an advantageous quality concerning the previous diagnosis solutions.

In the feature model context, FastDiag (FMDiag) works on a set of base constraints AC (a knowledge-representation of the feature model) and S (a set of constraints to perform the diagnosis on it) [22] (S is a subset of AC).

Even though FMDiag is efficient for a leading diagnosis, applications of this algorithm for a full diagnosis on feature models are not always more efficient than applying HSDAG for the same goal [22].

- FlexDiag is a variant of FastDiag for anytime diagnosis scenarios [19] [21], that is, diagnosis within time limits. This algorithm allows for trade-offs between diagnosis quality and performance of the diagnostic search.

  Application examples of FlexDiag are diagnosis on interactive configuration and reconfiguration scenarios which demand for a quick-time response [19] such as the installation process for a derivative of Linux operating system.

  The algorithmic structure of FlexDiag is like FastDiag with only one main difference: FlexDiag considers the use of the variable m to limit the diagnosis quality concerning the diagnosis minimality (m is not 1 necessarily). Consequently, FlexDiag solutions are not necessarily minimal but of a great value because an usual trade-off exists between the diagnosis quality and the applied algorithm performance.

  FlexDiag like FastDiag assumes AC is consistent to determine a diagnosis. If AC is inconsistent and $AC-S$ continue being inconsistent, then the algorithm returns the empty set ($\emptyset$). In other respects, a recursive $FlexD$ function is active in charge of determining a minimal diagnosis Δ. Like FMDiag, FlexDiag determines one leading diagnosis at a time based on a strict lexicographical ordering of the constraints part of S.

Felferning et al. [19] [21] present the evaluation results of the FlexDiag performance and diagnosis quality for different values of the parameter m on feature models configurations in the SPLOT database [7] with a random number of inconsistent new requirements $|R_p|$.

Algorithm 1 is the base function of both described algorithms, FastDiag and FlexDiag, and that function uses the algorithm 2. Algorithm 2 emulates the described Diag and FDiag functions of FMDiag and FlexDiag algorithms respectively. Algorithm 2 sets $FlexActive = false$ for FastDiag and $FlexActive = true$ for FlexDiag.

Felfernig et al. [22] list anomalies in feature models, their property checks and explanation for FMDiag and FlexDiag.

Algorithm 1 FMDiag(S, AC): Δ

```
if isEmpty(S) or inconsistent(AC-S) then
    return $\emptyset$;
else
    return Diag($\emptyset$, S, AC);
end if
```

Algorithm 2 Diag(D, $S = \{s_1, ..., s_r\}$, AC): Δ

```
if D $\neq \emptyset$ and consistent(AC) then
    return $\emptyset$;
end if

if FlexActive then
    if size(S) <= m then
        return S;
    end if
else
    if size(S) = 1 then
        return S;
    end if
end if
$k = \left\lceil \frac{r}{2} \right\rceil$;
$S_1 = \{s_1, ..., s_k\}$; $S_2 = \{s_{k+1}, ..., s_r\}$;
$\Delta_1 = Diag(S_2, S_1, AC - S_2)$;
$\Delta_2 = Diag(\Delta_1, S_2, AC - \Delta_1)$;
return($\Delta_1 \cup \Delta_2$);
```

Felferning et al. [19] evaluate the performance and diagnosis quality of FlexDiag. Just, those aspects are evaluated in function of the parameter m on feature models configuration of the SPLOT database [7] for a random number of inconsistent new requirements $|R_p|$. A clear trade-off exists between the performance for calculating the diagnosis and its quality.

Table 2 presents usual anomalies in feature models along with their property checks and explanation usable by the diagnosis algorithm of any FlexDiag version.

Analysis operation	Property check	Explanation (Diagnosis)
Void feature model	inconsistent $CF \cup c_0$?	$FMDiag(CF, CF \cup c_0)$
Dead (f_i)	inconsistent $CF \cup c_0 \cup \{f_i = true\}$?	$FMDiag(CF, CF \cup c_0 \cup \{f_i = true\})$
Full mandatory (f_i)	inconsistent $CF \cup c_0 \cup \{f_i = false\}$?	$FMDiag(CF, CF \cup c_0 \cup \{f_i = false\})$
False optional (f_{opt})	inconsistent $CF \cup c_0 \cup \{f_{par} = true \wedge f_{opt} = false\}$?	$FMDiag(CF, CF \cup c_0 \cup \{f_{par} = true \wedge f_{opt} = false\})$

Table 2: Feature model analysis operations, property checks, and related explanations.

4. SOLUTION PROPOSAL

Algorithms 3 (function ParFlexDiag) and 4 (function ParFlexD) represent the two modules of our solution:

- Algorithms 3 structurally behaves like its sequential version *FlexDiag* [19], that is, ParFlexDiag receives a set AC for all the possible constraints (the configuration knowledge base) and a set S of candidate constraints (requirements). The underlying assumption is that AC and S is inconsistent.

  Algorithms 3, checks first either the constraints set S is empty or $AC - S$ is consistent to return the $\emptyset$. For a non-empty and inconsistent set S, this function defines and activate a parallel job of the algorithm (function) 4 and returns its diagnosis result.

Algorithm 3 ParFMDiag(S, AC): Δ

```
if isEmpty(S) or inconsistent(AC-S) then
    return $\emptyset$;
else
    defining a Parallel Job;
    return ParFMD($\emptyset$, S, AC);
end if
```

- The parallel-recursive function $ParFMD$ receives a set of constraints AC and two subsets of the solution set, D and S. In the first call, D is empty and S contains the set of constraints for diagnosis.

  Like FMDiag, PFMDiag looks for diagnosis on a consistent base knowledge AC that includes a set of inconsistent user requirements S. We consider the same base cases of FMDiag for PFMDiag:

  - To return the empty set when the current AC is consistent and there exist a non-empty set D to possibly diagnosis on it: this base case permits omitting the current set S in analysis (not further taken into account in the diagnosis process); an efficient way to get rid of constraints that do not participate in the diagnosis. If this case were not valid, possibly we need to consider additional elements of S.
  - To return the current S when the first base case is not valid (either D is empty or AC is inconsistent) and the size of the current S is minimal: Like for the function FMD of FMDiag [22], the second base case of $ParFMD$ reviews whether the number of constraints in S is equal to 1, that is, if the number of constraints in the solution set S represents a minimal diagnosis. If it is valid, the function returns S.

  If neither of previous base cases are valid then the current S contains a diagnosis. PFMDiag plans to divide the current S in *nSplits* subsets. For example, S_1, S_2, S_3 and S_4 for $nSplits = 4$. Like FMDiag, for each ($nSplits - 1$) subset of S, that is, for each subsets previous to the last subset, PFMDiag determines the rest and less sets (FMDiag determines rest = S_2 and less = AC - rest = AC - S_2 for the subset S_1) for getting diagnosis results $\{\Delta_1, ..., \Delta_{nSplits-1}\}$ setting aside each previous to the last partition of S, that is, $D = rest$, $S = rest_{1...(n-1)}$, $AC = less_{1..(n-1)}$ (FMDiag first diagnosis on $D = S_2$, $S = S_1$, and $AC = AC - S_2$). After getting diagnosis results for the ($nSplits - 1$) subsets of S, PFMDiag diagnosis on $S_{nSplits}$ and AC - $\{\Delta_1, ..., \Delta_{nSplits-1}\}$ to obtain $\Delta_{nSplits}$ to finally returns the set of all previous results $\{\Delta_1, ..., \Delta_{nSplits}\}$.

  PFMDiag as an FMDiag extension is also adequate for minimal reconfiguration scenarios. In this context, the set $AC = C \cup R_p \cup S$ represents the union of all constraints in a reconfiguration task for the set of base constraints C, a set of reconfiguration requirements R_p, and an existing configuration solution S.

  Figures 4 and **??** depict PFMDiag search trees on the feature model example of figure 2: the first one for a granularity setting of $m = 1$ and a $SplitsNumber = 2$, and the second one for $m = 1$ and $SplitsNumber = 4$ in a reconfiguration scenario, respectively. Note that the first example returns $\{C_7\}$ whereas the second one returns $\{C_6, C_7\}$. As next section highlights, for $m > 1$ Parallel FMDiag results usually diminish in accuracy.

Algorithm 4 ParFMDiag(D, $S = \{s_1, ..., s_r\}$, AC): Δ

```
if D $\neq \emptyset$ and consistent(AC) then
    return $\emptyset$;
end if

if size(S) = 1 then
    return S;
end if

nSplits $\leftarrow$ SplitsNumber;
nS $\leftarrow$ $Size(S)$;
splitsLTS $\leftarrow$ $splitListToSubLists(S, nS/nSplits)$;
for $i = 0; i < (nSplits - 1); i++$ do
    rest $\leftarrow$ getRest(splitsLTS(i), splitsLTS);
    less $\leftarrow$ getLess(rest, AC);
    Def Job $PJ_i$ $\leftarrow$ ParFMD(rest, splitsLTS(i), less);
    $\Delta(i) \leftarrow Exec\ in\ Parallel\ PJ_i$;
end for
PR = $\Delta_1 \cup ... \cup \Delta_{nSplits-1}$;
Def Job $PJ_{nSplits}$ $\leftarrow$ ParFMD(PR, splitsLTS(nSplits), PR);
$\Delta(nSplits) \leftarrow Exec\ in\ Parallel\ PJ_{nSplits}$;
$return(\Delta_1 \cup ... \cup \Delta_{nSplits})$
```

FMDiag and Parallel FMDiag are complete algorithms, that is, they would find out one diagnosis in S, a minimal diagnosis for $m = 1$, if such diagnosis exists. For example, if S = $\{s_1, s_2\}$, where $AC \cup \{s_1\}$ and $AC \cup \{s_2\}$ are consistent, but $AC \cup \{s_1, s_2\}$ are inconsistent, there should be at least one relation rule between s_1, s_2 for this inconsistency. Therefore, S does not include all its rules, that is, the solution S is no well specified.

Since Parallel FMDiag preserves the essence of FMDiag [19], it also returns one diagnosis at a time and always finds a diagnosis in S if such diagnosis exists.

The main differences between Parallel FMDiag and FMDiag are their computing approaches (parallel and sequential, respectively), their machine requirements for execution and computing performances: a parallel solution is more efficient in parallel computing architectures.

In a divide-and-conquer approach, sub-problem solutions at the same level in an execution tree run independently. Thus, each recursive invocation of Parallel FMDiag does not

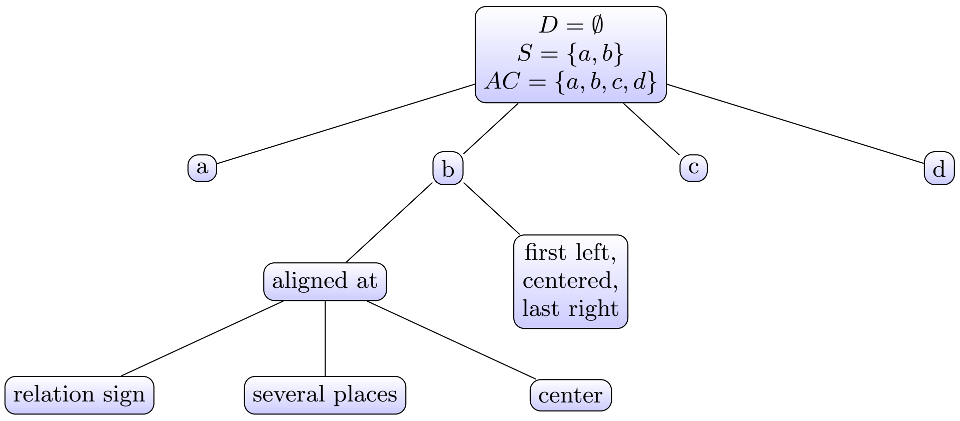


Figure 4: Parallel FMDiag in action: determining a minimal diagnosis for m = 1 and SplitsNumber = 2 ($\Delta = \{C_7\}$).

consider the results of previous function calls at the same level. This task independence and their parallel execution constitute the main characteristics of Parallel FMDiag.

5. EMPIRICAL EVALUATION

This section describes metrics to evaluate our solution and explains case studies for the metrics application. In particular, this section presents evaluation results for the validity and consistency of products configuration of generated feature models by Betty and feature models of the SPLOT repository, all the models of different size and structure to evaluate potential improvements and application advantages of Parallel FlexDiag diagnosis application regarding FlexDiag ones. This study only shows the analysis results of Parallel FlexDiag for 4 threads.

5.1 Metrics

We consider the execution time for the performance measurement as a main metric for diagnosis of FlexDiag and Parallel FlexDiag on the AAFM of feature models of products. Furthermore, we apply the FlexDiag metrics Minimality (*Min*) and Accuracy (*Ac*) in our analysis, just to evaluate and compare the diagnosis quality of FlexDiag and Parallel FlexDiag. We also describe the metric *Relevance* for future analysis work.

- $Min = \frac{|\Delta_{min}|}{|\Delta|}$

- $Ac = \frac{|\Delta \cap \Delta_{min}|}{|\Delta_{min}|}$

Like FlexDiag [19], Parallel FlexDiag allows reducing the number of consistency checks for determining preferred diagnosis, but it loses the minimality property of FastDiag. Hence, the size of Parallel FlexDiag solutions depends on the m value: If we increase the value of m, we decrease the number of consistency checks and vice-versa.

The *Relevance* metric [20] is useful to measure the rate of relevant constraints in FlexDiag diagnosis. Thus, we can apply it to measure the rate of relevant constraints of for FlexDiag and Parallel FlexDiag diagnosis concerning a minimal diagnosis Δ.

- $Relevance(C_D) = \frac{|\Delta|}{|C_D|}$

$C_D = \Delta \cup C_{offset}$, where C_{offset} is a set of constraints which are not part of δ.

We can define a *Relevance* ranking to classify the quality of diagnosis solutions to look for similarities in the results of different versions of Parallel FlexDiag concerning their number of threads T and values of m; that represents a future work of the authors.

5.2 Results

We evaluate FlexDiag and Parallel FlexDiag on the diagnosis of the validity of products configuration on feature models to analyze mainly their (1) *algorithm performance* and (2) *diagnosis quality*. Just, in the analysis of each feature model, we use 10 random generated products configuration and apply FlexDiag and Parallel FlexDiag on them to evaluate the consistency and diagnose potential issues of both solutions.

- First, we use a set of features models and random product configurations for feature models from the SPLOT repository [7]. Currently, we are working on the analysis of products validation on all SPLOT feature models. Since this work is still in progress, here we present a set of preliminary results.

- Second, we analyze a set of features models and random product configurations for X feature models generated by Betty [35].

All experiments ran on a 2.3 GHz i3 processor (Intel Corporation, Santa Clara, CA, USA) machine with 16 GB RAM. We present graphs usually with a log scale for the execution time, minimality and accuracy of some of the results for the FlexDiag and Parallel FlexDiag solutions.

Tables 3 and 4 show the hypothesis and use of variables to conduct the experiments on the SPLOT models and randomly generated feature models by Betty, for the validity of products configuration in their feature models. The null hypothesis of both experiments establishes that Parallel FlexDiag is lesser efficient and delivers poorer quality diagnosis results than FlexDiag. For this analysis, we implement and use FlexDiag and Parallel FlexDiag in the CHOCO constraint solver library, a reasoning engine part of the FAMA tool suite [15].

Hypotheses of Experiment 1	
Null Hypothesis (H_0)	Parallel FlexDiag is not more efficient than FlexDiag in the diagnosis of validity of products configuration for a set of chosen SPLOT feature models regardless the number of threads.
Alt. Hypothesis (H_1)	Parallel FlexDiag is more efficient than FlexDiag in the diagnosis of validity of products configuration for a set of chosen SPLOT feature models and more than one thread.
Models used as input	X chosen model from the SPLOT repository and generated products
Blocking variables	For each SPLOT model, we evaluate the product configuration for X, Y and Z products...
Constants	
FaMA Choco Reasoner	*FlexDiag and Parallel FlexDiag solutions built on 2015 and 2017, respectively*
Heuristic for variable selection in the SAT solver	*Default*

Table 3: Hypotheses and design of experiments for SPLOT models.

Hypotheses of Experiment 2	
Null Hypothesis (H_0)	Parallel FlexDiag does not perform more efficiently than FlexDiag in the diagnosis of validity of products configuration for feature models generated by Betty regardless the number of threads.
Working Hypothesis (H_1)	Parallel FlexDiag is more efficient than FlexDiag in the diagnosis of validity of products configuration for a set of random generated feature models by Betty and more than one thread.
Models used as input	X Feature models randomly generated by Betty tool suite.
Blocking variables	Established features number and cross-tree constraint percentage.
Constants	
FaMA Choco Reasoner	*FlexDiag and Parallel FlexDiag solutions built on 2015 and 2017, respectively*
Heuristic for variable selection in the SAT solver	*Default*

Table 4: Hypotheses and design of experiments for generated models by Betty.

5.2.1 Experiment 1: Working on SPLOT feature models

This experiment aims to show off improvements of our approach for the validity of products configuration on a set of feature models of small size from the SPLOT repository [7], overall to appreciate the execution time performance and quality results of Parallel FlexDiag diagnosis concerning the FlexDiag diagnosis results.

Experiment 1 Results. Tables 5.2.1 and 5.2.1 show diagnosis results of FlexDiag and Parallel FlexDiag on different feature models from the SPLOT repository for different values of m (1, 2, and 4). The maximum number of features in a feature model was 172. For the set of analyzed models and number of associated experiments, both tables present mean values of product, relations, resultSize, time, minimality and accuracy metrics.

Taking into account the diagnosis results of FlexDiag and Parallel FlexDiag on the chosen SPLOT feature models, undoubtedly FlexDiag seems more efficient in average time to work on the diagnosis of the validity of products configuration on feature models of small size; Parallel FlexDiag performance faster than FlexDiag for the biggest number of features models.

Figure 5 illustrates the execution time performance for diagnosis of FlexDiag and Parallel FlexDiag on the validity of products on the analyzed SPLOT feature models. Clearly, FlexDiag is more efficient for diagnosis tasks on reduced in size (small) feature models meanwhile Parallel FlexDiag is more efficient for bigger feature models.

Concerning the quality results, note that FlexDiag and Parallel FlexDiag solutions present the same average values for $Minimality$ and $Accuracy$ for $m = 1$; these algorithms give different $Minimality$ and $Accuracy$ values for other values of m. As section 3 describe, Parallel FlexDiag captures the essence of FlexDiag, but for the independence of its parallel-recursive tasks, its $Accuracy$ and $Minimality$ results can differ regarding the equivalent FlexDiag results for $m > 1$. We can appreciate these differences in tables 5.2.1 and 5.2.1; sometimes Parallel FlexDiag gives the best solution meanwhile there are cases in which FlexDiag gives the best ones.

The main hypothesis of the first experiment is not valid, that is, given the experiment results we discard the main hypothesis since there exist cases on which Parallel FlexDiag performs more efficiently and delivers better quality results on diagnosis than FlexDiag in the same multi-core computing environment. Parallel FlexDiag functioning adds more parallel threads which imply a more efficient computation preserving the quality results in a multi-core computing environment; for the current availability of that computing architecture, we can omit to invest in advanced computing technology to look for more efficient and quality diagnosis results.

5.2.2 Experiment 2: Working on generated by Betty feature models

We practice experiments on feature models of a variety of size generated by Betty and we consider 10 randomly generated products for each of the models, just to analyze the Parallel FlexDiag and FlexDiag performance and quality diagnosis application on the validity of products.

Experiment 2 Results. Tables 5.2.2 and 5.2.2 show execution and performance quality details of FlexDiag and Parallel FlexDiag diagnosis solutions.

Figure 6 and 7 show the execution time for diagnosis of FlexDiag and Parallel FlexDiag solutions for the validity of products on generated feature models by Betty; figure 8 and 9 depict the Minimality results for these experiments; and figure 10 and 11 illustrate the Accuracy results of both solutions. There exist again cases for which Parallel FlexDiag

features	variables	m	product	relations	resultSize	time	minimality	accuracy
10	10	1	484.598	3.275	1.157	113.167	1	1
10	10	2	484.598	3.275	1.824	107.431	0.698	0.944
10	10	4	484.598	3.275	2.627	101	0.563	0.936
20	20	1	335.833	5 1.875	146.25	1	1	
20	20	2	335.833	5 3.083	133.313	0.582	0.942	
20	20	4	335.833	5 3.896	123.354	0.487	0.915	
30	30	1	494.5	5	1	106.5	1	1
30	30	2	494.5	5	1.75	113.75	0.625	1
30	30	4	494.5	5	2.75	95.75	0.479	1
40	40	1	360.5	18.5	3.625	351.25	1	1
40	40	2	360.5	18.5	5.75	305	0.596	0.852
40	40	4	360.5	18.5	7.875	266	0.411	0.840
51	51	1	358.5	11	1	143	1	1
51	51	2	358.5	11	1.75	133.5	0.625	1
51	51	4	358.5	11	3	125.25	0.458	1
60	60	1	262.5	9	1	209	1	1
60	60	2	262.5	9	2	198.25	0.5	1
60	60	4	262.5	9	3.5	172	0.292	1
70	70	1	710.5	9	2.25	1082.5	1	1
70	70	2	710.5	9	4	1036.25	0.639	0.825
70	70	4	710.5	9	7.25	926	0.340	0.825
77	77	1	534.5	12	3	340.3475	1	1
77	77	2	534.5	12	5.75	330.5	0.625	0.845
77	77	4	534.5	12	8	274.25	0.339	0.726
88	88	1	374.5	62	1.5	394.75	1	1
88	88	2	374.5	62	2.5	323.25	0.583	0.875
88	88	4	374.5	62	5	338.75	0.292	0.875
97	97	1	822.5	26	7.25	605.25	1	1
97	97	2	822.5	26	11.25	534.75	0.612	0.765
97	97	4	822.5	26	17.75	409.75	0.352	0.765
137	137	1	222.5	2	1	178.25	1	1
137	137	2	222.5	2	1.75	147	0.625	1
137	137	4	222.5	2	2.75	142.25	0.479	1
168	168	1	862.5	56	6.75	840.25	1	1
168	168	2	862.5	56	10.75	669	0.615	0.753
168	168	4	862.5	56	17.75	618	0.342	0.742
172	172	1	414.5	29	14.25	3570.5	1	1
172	172	2	414.5	29	22.75	3547.75	0.654	0.856
172	172	4	414.5	29	34	3107	0.405	0.730

Table 5: FlexDiag Diagnosis for the validity of Products configuration on feature models from SPLOT repository

is more efficient than FlexDiag in time for giving a diagnosis solution. Results of Minimality and Accuracy metrics are the same for $m = 1$, and they differ for other values of m mainly for the subtasks dependence and independence functioning of both solutions. Like in the previous experiment, concerning the quality metrics, there exist cases in which Parallel Flexdiag delivers the best results and cases for which FlexDiag results are the best. By these results, the main hypothesis of our second experiment is also discarded and replaced by the null hypothesis.

Similar to the previous experiments, Parallel FlexDiag is more efficient than FlexDiag to work on mainly products configuration of large-scale feature models. Nevertheless, as we appreciate in the first experiment, there exist exceptions, overall for cases with a well located and small number of diagnosis. As section 3 discuss, Parallel FlexDiag presents parallel-recursive processes and each one of them does not depend of results of other processes at the same level, that is, they will execute analysis work independently of other tasks; this is not the same in FlexDiag. A current research work represents to define the small number of threads for which we can obtain diagnosis results improvements concerning the performance and results quality.

5.3 Analysis of Results

According to this experiment results, there exist different cases for which Parallel FlexDiag is more efficient in performance and gives better quality results than FlexDiag overall for the diagnosis of the validity of products configuration in large-scale feature models.

6. RELATED WORK

Next, we mention work related to the feature models diagnosis and their representation as knowledge-based configuration.

- The work of [38] implement feature models into CSP di-

features	variables	m	threads	product	relations	resultSize	time	minimality	accuracy
10	10	1	4	484.598	3.275	1	191.196	1	1
10	10	2	4	484.598	3.275	1.122	191.745	0.941	1
10	10	4	4	121.529	3.275	1.833	175.902	0.715	1
20	20	1	4	335.833	5	1	204.854	1	1
20	20	2	4	335.833	5	1.458	195.479	0.802	1
20	20	4	4	335.833	5	1.792	191.792	0.726	1
30	30	1	4	494.5	5	1	157.75	1	1
30	30	2	4	494.5	5	1	155	1	1
30	30	4	4	494.5	5	1.5	155	0.833	1
40	40	1	4	360.5	18.5	1	332	1	1
40	40	2	4	360.5	18.5	1	333.875	1	1
40	40	4	4	360.5	18.5	3.625	306	0.514	1
51	51	1	4	358.5	11	1	176	1	1
51	51	2	4	358.5	11	1	179	1	1
51	51	4	4	358.5	11	1	180.5	1	1
60	60	1	4	262.5	9	1	251.5	1	1
60	60	2	4	262.5	9	1.25	242.75	0.875	1
60	60	4	4	262.5	9	1.25	252.75	0.875	1
70	70	1	4	710.5	9	1	985.75	1	1
70	70	2	4	710.5	9	1	922.75	1	1
70	70	4	4	710.5	9	1	965.25	1	1
77	77	1	4	534.5	12	1	363.5	1	1
77	77	2	4	534.5	12	1	368	1	1
77	77	4	4	534.5	12	1	366.5	1	1
88	88	1	4	374.5	62	1	359.75	1	1
88	88	2	4	374.5	62	1	364	1	1
88	88	4	4	374.5	62	1	354.75	1	1
97	97	1	4	822.5	26	1	516	1	1
97	97	2	4	822.5	26	1	472.5	1	1
97	97	4	4	822.5	26	1	472	1	1
137	137	1	4	222.5	2	1	163.75	1	1
137	137	2	4	222.5	2	1	165	1	1
137	137	4	4	222.5	2	1	163.75	1	1
168	168	1	4	862.5	56	1	588.5	1	1
168	168	2	4	862.5	56	1	587.5	1	1
168	168	4	4	862.5	56	1.5	572.25	0.833	1
172	172	1	4	414.5	29	1	1444.75	1	1
172	172	2	4	414.5	29	1.75	1465	0.813	1
172	172	4	4	414.5	29	1.75	1477	0.813	1

Table 6: Parallel FlexDiag Diagnosis for the validity of Products configuration on feature models from SPLOT repository working with 4 threads.

agnosis problems to identify void models, dead features and false optional features for the automated support of error analysis in feature model, by the use of Choco solver in the FaMa tool [15]. First, [38] proposes an algorithmic solution that relies on the Reiter's theory of diagnosis [33] (a theory mainly based on a strict breadth-first search regime). Second, the work of [38] evaluates and gives a result for five small feature models only (feature models of at most 86 features) without giving details about scalability and efficiency of the solution. Thus, the FastDiag and FlexDiag algorithms are computationally more efficient in the large-scale feature models analysis for their divide-and-conquer approach. Furthermore, since explanations are in an abbreviated form, a quick recognition of the cause of an anomaly may be very expensive.

- The work of [41] describes how to represent feature models into CSP tools for their analysis, and presents practical results to highlight the viability for analyzing models of 5000 features.

- The work of [40] highlight the advantages of CSP over different automated reasoning techniques mainly for its declarative essence, that is, systems are representable by a verifiable set of rules, and the availability of off-the-shelf solvers for implementing systems as rules. This article also defines the Automated Analysis of Feature Models (AAFM) and remarks the tendency of using declarative approaches in the AAFM. Thus, our FastDiag family solutions are examples of AAFM.

- The work of [34] transform feature models into a CSP representation and applies a linear algorithm to detect inconsistencies in feature models as Minimal Constraint Sets (MCSs) along with their possible corrections. This work analyzes and gives experiment results of 78 feature models of a small size.

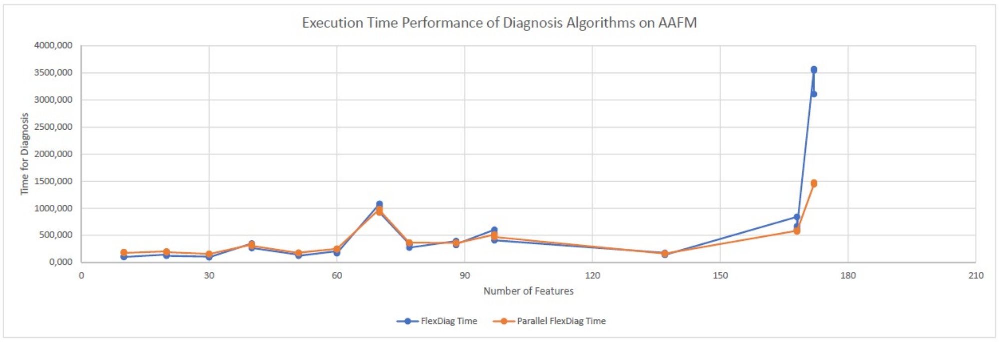


Figure 5: FlexDiag in action - Time on the validity of products configuration of feature models generated by Betty for diverse values of m and 4 threads

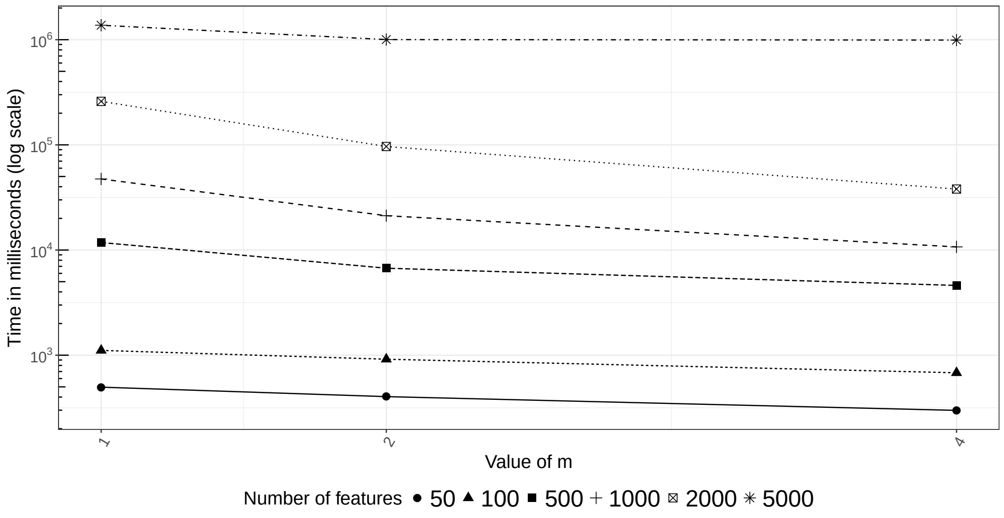


Figure 6: FlexDiag in action - Time on the validity of products configuration of feature models generated by Betty for diverse values of m and 4 threads

- For the anomalies explanation in feature models, [29] present a generic algorithm for explaining various possible inconsistencies in feature models considering their encoding as CNF rules along with an initial set of truth values assumptions. Even though the results are computationally non-efficient, they highlight the explanations length which remains acceptable. This article also mentions the base algorithms of the FastDiag family remarking that FastDiag explains all type of inconsistencies in feature models and it is independent of the reasoning techniques for its implementation, and previous results are only of a reduced number of small feature models.

- For extended feature models, that is, feature models with feature attributes (augmented features with additional non-Boolean configuration information), the work of [16] present a conflict detection approach based on symbolic graph transformation. They implement this proposal combining a graph transformation tool (eMoflon) with the Z3 SMT solver and evaluate the implementation concerning the applicability. They remark a reduction of false positives compared to a conventional conflict detection approach.

- Considering current computation approaches, [24] applies MapReduce big data techniques for enumerating all feature model configurations in a pre-compile process, to apply on those results reasoning operations in a polytime. Since the divide-and-conquer thinking is a base for the MapReduce computation approach [42], our current efforts are to design, implement and test a MapReduce version of FastDiag and FlexDiag in Hadoop. Likewise, for the feature models graph nature, we are looking for implementing a Giraph [36] version of FastDiag and FlexDiag.

As this article describe, the base for the FastDiag family algorithms are:

- First, the work of [20] which propose the FastDiag algorithm for an efficient identification of a preferred diagnosis in a set of inconsistent constraint.

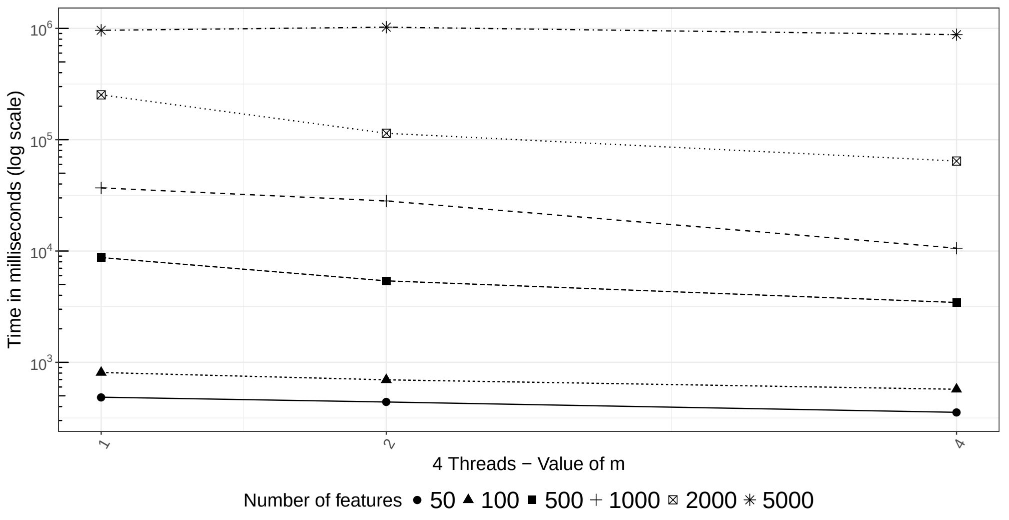


Figure 7: Parallel FlexDiag in action - Time on the validity of products configuration of feature models generated by Betty for diverse values of m and 4 threads

- Second, the work of [22] which apply and present results of the FastDiag algorithm for the AAFM.
- Third, the work of [19] for the FlexDiag proposal and its application on the AAFM.

Thus, this article describes the parallel version FlexDiag to highlights a result improvements in the AAFM. Table 9 presents a list of AAFM solutions and details about if they posses Scalability, their number of diagnosis, the accuracy of their solutions, and if they work on CSP. We appreciate that Parallel FlexDiag is the only scalable solution, it delivers one leading diagnosis, it partially ( ) permits accuracy solutions ($m = 1$), and it runs on an intelligent CSP scenario.

	Scalability	Diagnosis	Accurate Solution	CSP
Parallel FlexDiag	X	1 (leading)	∼X	X
FlexDiag		1 (leading)	∼X	X
FastDiag		1 (leading)	X	X
HSDAG		All	X	X

Table 9: AAFM Solutions

7. CONCLUDING REMARKS & LESSONS LEARNED

For the current availability of parallel computing by multi-core architectures; in this paper, we presented Parallel Flex-Diag, an efficient and scalable FlexDiag extension for the explanation of anomalies in feature models analysis overall for large-scale models. We motivated and gave the background for our proposal, and presented results to validate our main hypothesis: Parallel FlexDiag is an efficient FlexDiag extension for the diagnosis analysis on large-scale feature models.

Concerning other diagnosis solutions, Parallel FlexDiag preserves the benefits and properties of previous algorithms of its family and adds the scalability property. For the recursive and divide-and-conquer nature of FastDiag family algorithms, each of these solutions is expressible as a recursive-parallel solution if we grant their subtasks independence. FlexDiag uses m as a parameter to improve the computing performance for diagnosis. Parallel FlexDiag uses the value of m and adds the number of threads as a new parameter. We appreciated that for more threads, 4 threads for Parallel FlexDiag versus 1 thread for FlexDiag in these experiments, performance can be improved regarding execution time and diagnosis quality.

We classify future research work in those directly linked to Parallel FlexDiag and others:

- As a future work directly linked to Parallel FlexDiag, based on results of execution experiments, we want to try to define a general improvement-relation between the number of threads and m value for getting either execution time or quality metrics improvement on Parallel FlexDiag diagnosis solutions. In the same context, we want to apply the *Relevance* metric and consider these values for the same improvement-relation.
- As an additional future work directly linked to Parallel FlexDiag, we plan to analyze the applicability of Parallel FlexDiag on detecting anomaly patterns in other models representation such as UML and business process model notation which demand the definition of further anomaly patterns.
- As a future work no directly linked to Parallel FlexDiag, we plan to extend FMCore [22] to support parallel computing approach, just for more efficient detection of minimal redundancies in model analysis.

Acknowledgements

First author acknowledge the FAMA team from the University of Seville for the support and confidence to perform these experiments. Cristian also acknowledges his family in

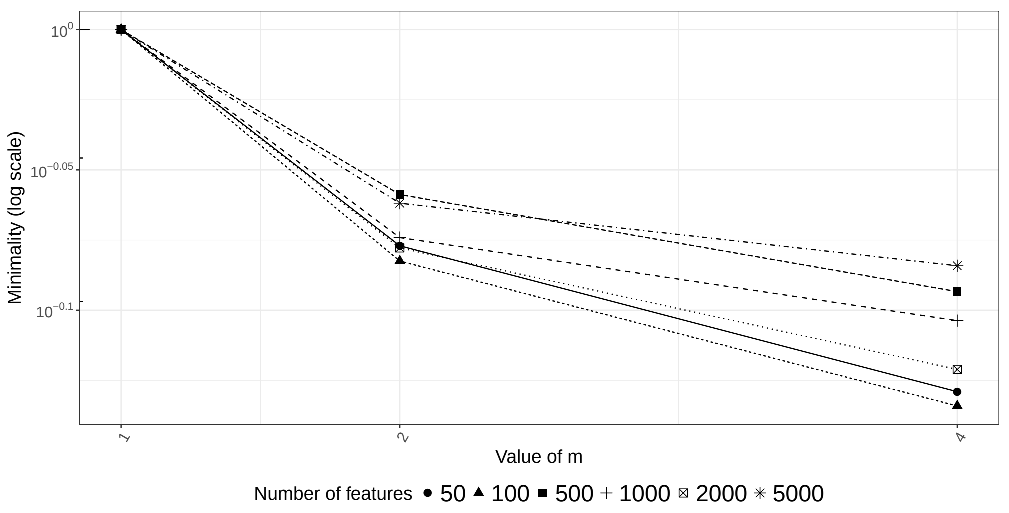


Figure 8: FlexDiag in action - Minimality results on the validity of products configuration of feature models generated by Betty for diverse values of m and 4 threads

Chile, specially for his son and wife, his main inspiration to perform and complete this research work.

8. REFERENCES

[1] Debian – el sistema operativo universal. `https://www.debian.org/`. Accessed: 2016-14-02.

[2] Fork and join: Java can excel at painless parallel programming too! `http://www.oracle.com/technetwork/articles/java/fork-join-422606.html`. Accessed: 2016-14-02.

[3] Guadalinex – software libre. `http://www.guadalinex.org/`. Accessed: 2017-30-09.

[4] Linux△ubuntu – make your very own customized linux distro from your current installation. `http://www.linuxandubuntu.com/`. Accessed: 2017-30-09.

[5] Linuxvoice – build your own linux distro. `https://www.linuxvoice.com/build-your-own-linux-distro/`. Accessed: 2017-30-09.

[6] Qimo for kids – goodbye friends. `http://www.qimo4kids.com/`. Accessed: 2017-30-09.

[7] S.p.l.o.t. software product lines online tools. `http://www.splot-research.org/`. Accessed: 2016-14-02.

[8] Steam – steamos is here, freely give. `http://store.steampowered.com/steamos/`. Accessed: 2017-30-09.

[9] Ubuntu christian edition – freely ye have received, freely give. `http://ubuntuce.com/`. Accessed: 2017-30-09.

[10] Ubuntu: The leading operating system for pcs, tablets, phones, iot devices, servers and the cloud. `https://www.ubuntu.com/`. Accessed: 2016-14-02.

[11] S. Apel, D. Batory, C. Kstner, and G. Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer Publishing Company, Incorporated, 2013.

[12] R. R. Bakker, F. Dikker, F. Tempelman, and P. M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. pages 276–281, 1993.

[13] D. Benavides, A. Felfernig, J. A. Galindo, and F. Reinfrank. Automated analysis in feature modelling and product configuration. In *ICSR*, pages 160–175, Pisa, Italy, 06/2013 2013.

[14] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, Sept. 2010.

[15] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz–Cortés. Fama: Tooling a framework for the automated analysis of feature models. In *Proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, page 129–134, 2007.

[16] F. Deckwerth, G. Kulcsar, M. Lochau, G. Varró, and A. Schürr. Conflict detection for edits on extended feature models using symbolic graph transformation. 206:17–31, 2016.

[17] N. Dintzner, A. Deursen, and M. Pinzger. Analysing the linux kernel feature model changes using fmdiff. *Softw. Syst. Model.*, 16(1):55–76, Feb. 2017.

[18] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen. *Knowledge-based Configuration: From Research to Business Cases.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edition, 2014.

[19] A. Felfernig, rouven Walter, and S. Reiterer. Flexdiag: Anytime diagnosis for reconfiguration. September 2015.

[20] A. Felfernig, M. Schubert, and C. Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *Artif. Intell. Eng. Des. Anal. Manuf.*, 26(1):53–62, Feb. 2012.

[21] A. Felfernig, R. Walter, J. A. Galindo, D. Benavides, S. P. Erdeniz, M. Atas, and S. Reiterer. Anytime

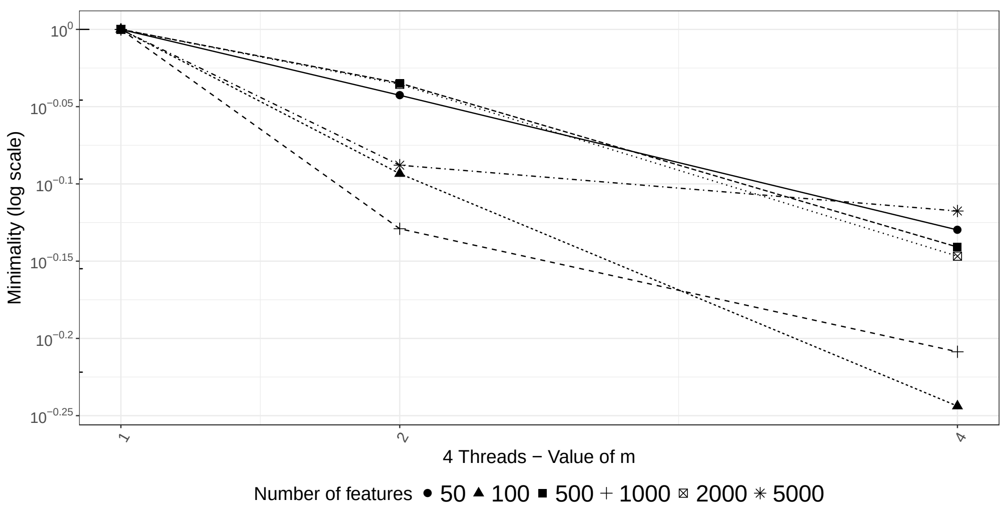


Figure 9: Parallel FlexDiag in action -Minimality results on the validity of products configuration of feature models generated by Betty for diverse values of m and 4 threads

diagnosis for reconfiguration. *J. Intell. Inf. Syst.*, 51(1):161–182, 2018.

[22] E. Felfernig, D. Benavides, J. Galindo, F. Reinfrank, A. Felfernig, D. Benavides, J. Galindo, and F. Reinfrank. Towards anomaly explanation in feature models, August 2013.

[23] J. A. Galindo, M. Acher, J. M. Tirado, C. Vidal, B. Baudry, and D. Benavides. Exploiting the enumeration of all feature model configurations: A new perspective with distributed computing. pages 74–78, 2016.

[24] J. A. Galindo, M. Acher, J. M. Tirado, C. Vidal, B. Baudry, and D. Benavides. Exploiting the enumeration of all feature model configurations: A new perspective with distributed computing. pages 74–78, 2016.

[25] J. A. Galindo, M. Alférez, M. Acher, B. Baudry, and D. Benavides. A variability-based testing approach for synthesizing video sequences. pages 293–303, 2014.

[26] J. A. Galindo, D. Benavides, and S. Segura. Debian packages repositories as software product line models. towards automated analysis. pages 29–34, 2010.

[27] U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. pages 167–172, 2004.

[28] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study, 1990.

[29] M. Kowal, S. Ananieva, and T. Thüm. Explaining anomalies in feature models. pages 132–143, 2016.

[30] J. H. Liang, V. Ganesh, K. Czarnecki, and V. Raman. Sat-based analysis of large real-world feature models is easy. pages 91–100, 2015.

[31] R. Lotufo, S. She, T. Berger, K. Czarnecki, and A. Wąsowski. Evolution of the linux kernel variability model. In *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*, SPLC'10, pages 136–150, Jeju Island, South Korea, 2010. Springer-Verlag.

[32] J. A. Pereira, S. Krieter, J. Meinicke, R. Schröter, G. Saake, and T. Leich. Featureide: Scalable product configuration of variable systems. pages 397–401, 2016.

[33] R. Reiter. A theory of diagnosis from first principles. *AI Journal*, 23(1):57—95, 1987.

[34] L. Rincón, G. Giraldo, R. Mazo, C. Salinesi, and D. Diaz. Method to identify corrections of defects on product line models. *Electron. Notes Theor. Comput. Sci.*, 314(C):61–81, June 2015.

[35] S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, and A. Ruiz-Cortés. Betty: Benchmarking and testing on the automated analysis of feature models. pages 63–71, 2012.

[36] R. Shaposhnik, C. Martella, and D. Logothetis. *Practical Graph Analytics with Apache Giraph.* Apress, Berkely, CA, USA, 1st edition, 2015.

[37] S. She, L. R., T. Berger, A. Wasowski, and K. Czarnecki. The variability model of the linux kernel, 2010.

[38] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro. Automated error analysis for the agilization of feature modeling. *J. Syst. Softw.*, 81(6):883–896, June 2008.

[39] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. Fama framework. pages 359–, 2008.

[40] P. Trinidad, A. Ruiz-Cortés, and D. Benavides. Automated analysis of stateful feature models. pages 375–380, 2013.

[41] J. White, D. Benavides, D. C. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. Automated diagnosis of feature model configurations. *J. Syst. Softw.*, 83(7):1094–1107, July 2010.

[42] T. White. *Hadoop: The Definitive Guide.* O'Reilly

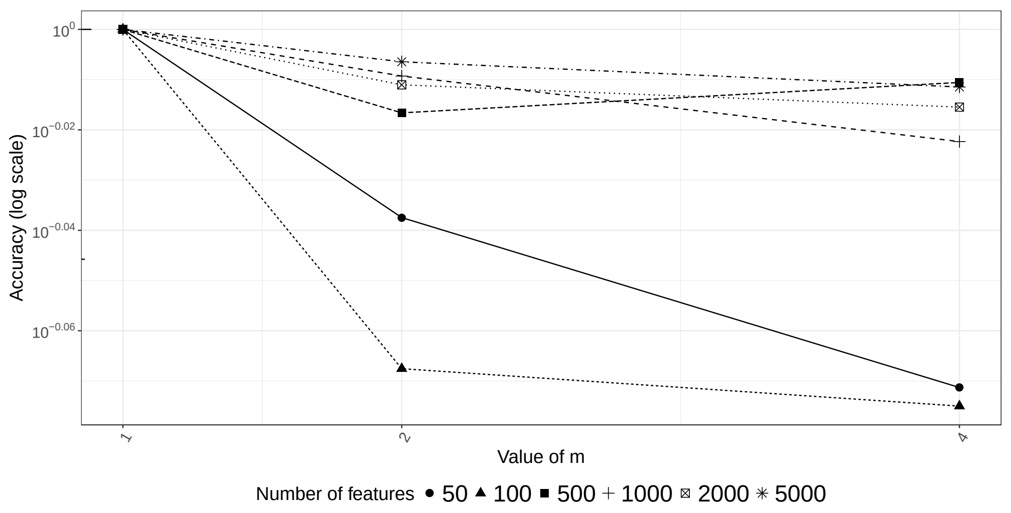


Figure 10: FlexDiag in action - Accuracy results on the validity of products configuration of feature models generated by Betty for diverse values of m and 4 threads

Media, Inc., 2012.

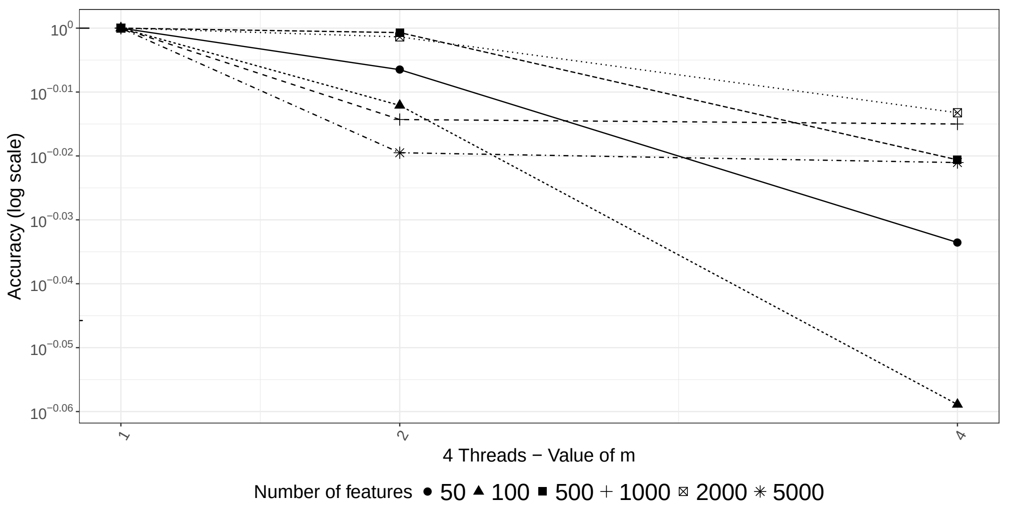


Figure 11: Parallel FlexDiag in action - Accuracy results on the validity of products configuration of feature models generated by Betty for diverse values of m and 4 threads

features	variables	m	product	relations	resultSize.mean	time.mean	minimality.mean	accuracy.mean
50	50	1	70.50	58.40	14.65	496.80	1.00	1.00
50	50	2	70.50	58.40	15.90	403.85	0.84	0.92
50	50	4	70.50	58.40	17.45	298.85	0.74	0.85
100	100	1	10.50	109.80	27.35	1110.70	1.00	1.00
100	100	2	10.50	109.80	32.40	916.00	0.83	0.86
100	100	4	10.50	109.80	36.40	680.60	0.73	0.84
500	500	1	90.50	566.40	179.55	11789.45	1.00	1.00
500	500	2	90.50	566.40	199.45	6717.60	0.87	0.96
500	500	4	90.50	566.40	217.80	4606.25	0.81	0.98
1000	1000	1	30.50	1141.00	346.55	47408.30	1.00	1.00
1000	1000	2	30.50	1141.00	397.65	21206.65	0.84	0.98
1000	1000	4	30.50	1141.00	434.55	10704.70	0.79	0.95
2000	2000	1	50.50	2274.80	665.70	259164.60	1.00	1.00
2000	2000	2	50.50	2274.80	748.75	96463.20	0.84	0.97
2000	2000	4	50.50	2274.80	822.85	38010.05	0.76	0.97
5000	5000	1	109.25	5464.42	813.08	1372034.25	1.00	1.00
5000	5000	2	110.20	5406.87	1242.27	1002577.80	0.87	0.99
5000	5000	4	109.50	5523.83	1804.89	990962.56	0.82	0.97

Table 7: Diagnosis of FlexDiag for the validity of Products configuration on Betty generated feature models.

features	variables	m	threads	product	relations	resultSize	time	minimality	accuracy
50	50	1	4	70.50	58.40	13.65	485.90	1.00	1.00
50	50	2	4	70.50	58.40	14.05	439.95	0.91	0.99
50	50	4	4	70.50	58.40	15.60	355.35	0.74	0.93
100	100	1	4	10.50	109.80	19.90	809.95	1.00	1.00
100	100	2	4	10.50	109.80	24.55	695.95	0.81	0.97
100	100	4	4	10.50	109.80	30.65	572.55	0.57	0.87
500	500	1	4	90.50	566.40	156.10	8734.85	1.00	1.00
500	500	2	4	90.50	566.40	168.35	5395.20	0.92	1.00
500	500	4	4	90.50	566.40	187.80	3448.30	0.72	0.95
1000	1000	1	4	30.50	1141.00	270.40	36902.40	1.00	1.00
1000	1000	2	4	30.50	1141.00	285.25	28166.75	0.74	0.97
1000	1000	4	4	30.50	1141.00	338.80	10601.25	0.62	0.97
2000	2000	1	4	50.50	2274.80	574.45	253001.65	1.00	1.00
2000	2000	2	4	50.50	2274.80	599.35	114190.00	0.92	1.00
2000	2000	4	4	50.50	2274.80	638.70	64297.00	0.71	0.97
5000	5000	1	4	109.25	5464.42	656.92	961444.50	1.00	1.00
5000	5000	2	4	110.20	5406.87	898.20	1025588.40	0.82	0.96
5000	5000	4	4	109.50	5523.83	1421.00	878325.61	0.76	0.95

Table 8: Diagnosis of Parallel FlexDiag for the validity of Products configuration on Betty generated feature models