Un analizador de modelos de variabilidad basado en el árbol de características

José Miguel Horcas[1 [0000−0002−7771−0575]], Mónica Pinto[1 [0000−0002−5376−742X]], and Lidia Fuentes[1 [0000−0002−5677−7156]]

CAOSD Group, ITIS Software, Universidad de Málaga
{horcas,mpinto,lfuentes}@uma.es

Resumen Un árbol de características generalizado (GFT) es un modelo de variabilidad en el que las restricciones textuales han sido eliminadas manteniendo la semántica del modelo. La ventaja de un GFT es que se puede analizar directamente razonando sobre las relaciones jerárquicas del árbol de características, sin tener que transformar el modelo a SAT o construir un árbol de decisión binario (BDD). Las desventajas de un GFT son que puede contener características duplicadas y que su tamaño en número de características con respecto al modelo de variabilidad original es considerablemente mayor, lo que complica el análisis automático. En este artículo se propone un analizador de modelos GFT basado en las relaciones jerárquicas del árbol de características teniendo en cuenta la existencia de características duplicadas. Se definen un conjunto de operaciones de análisis sobre GFT y se compara su eficiencia con solvers SAT y BDD. El solver GFT mejora la eficiencia del análisis sobre solvers BDD para modelos de hasta diez mil características.

Keywords: Análisis automático · Árbol de característica · GFT · Modelo de variabilidad · SPL · Solver

1. Introducción

Los **modelos de características** (*feature models*) [1] se han convertido en el principal artefacto para modelar la variabilidad en una **línea de productos software** (*software product line*, SPL) [25]. Un modelo de características se compone por (1) un **árbol de características** (*feature tree*) que descompone jerárquicamente el conjunto de características mediante relaciones padre-hijo que dan lugar a características obligatorias, opcionales, o grupos de características con diferente multiplicidad (or = [1..*], ó xor = [1..1]); y (2) un conjunto de **restricciones** entre características (*cross-tree constraints*) expresadas mediante formulas textuales en lógica proposicional.

La complejidad de analizar los modelos de características viene dada por su tamaño (i.e., número de características) y por el conjunto de restricciones. Generalmente, el análisis de un modelo de características requiere su formalización a lógica proposicional (*boolean satisfability problem*, SAT), a programación con restricciones (*constraints satisfaction problem*, CSP), o su codificación a otra estructura que facilite el análisis como un diagrama de decisión binario (*binary*

decision diagram, BDD). Cuando el modelo de características no tiene restricciones textuales, su análisis se puede hacer directamente sobre su árbol de características razonando sobre las relaciones jerárquicas del árbol. A este respecto, existen varios artículos [7,20] enfocados en la refactorización de las restricciones textuales para representarlas en el árbol de características dando como resultado un **árbol de características generalizado** (*generalized feature tree*, GFT) [6]. Sin embargo, la eliminación de las restricciones tiene serias implicaciones en el modelo (GFT) resultante. Primero, el árbol de características duplica su tamaño con la eliminación de cada restricción, por lo que para modelos grandes con muchas restricciones puede ser inviable eliminar todas las restricciones. Segundo, las características aparecen en el GFT por duplicado en diferentes ramas, lo que dificulta el análisis y/o incluso imposibilita el análisis de un GFT con SAT. Esto se debe a que las características duplicadas aparecen siempre en ramas alternativas (i.e., debajo de un grupo xor), y aunque el modelo mantenga su semántica (los productos son los mismos), un resolutor (*solver* en adelante) SAT no tendrá en cuenta esas duplicidades, dando como resultado análisis erróneos. Por ejemplo, el GFT de la Figura 1 representa los tres productos listados, sin embargo un SAT solver nos devolverá que ese GFT no tiene productos válidos debido a que la feature `A` aparece duplicada en un grupo *xor*.

En este artículo se propone un solver de GFT basado en las relaciones jerárquicas del árbol de características y se define una serie de operaciones de análisis para comparar su eficiencia con los solvers tradicionales de modelos de características (i.e., SAT y BDD). El solver se ha integrado como un plugin dentro de Flama [11], una de las herramientas de análisis más conocidas en la comunidad SPL. Las contribuciones de este artículo son las siguientes:

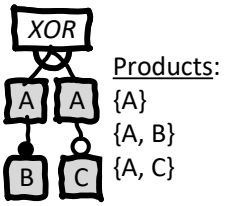

Figura 1. Semántica en un árbol de características generalizado (GFT).

- Se formalizan los modelos GFT usando los mismos conceptos usados por la comunidad SPL para definir un modelo de características genérico (Sección 2).
- Se define un conjunto de operaciones de análisis sobre modelos GFT siguiendo la formalización introducida (Sección 3).
- Se implementa un solver de GFT en Python y se integra como plugin dentro de Flama [11], una de las herramientas de análisis más conocidas en la comunidad SPL. Además, se evalúa la eficiencia de las operaciones de análisis del solver GFT comparándolas con los solvers SAT y BDD (Sección 4).

Esta contribución supone un primer paso en el estudio de la viabilidad y puesta en práctica de un solver basado en las relaciones del árbol de características de un modelo de variabilidad. Aunque la idea de los modelos GFT no es nueva [6], no había sido llevada a la práctica con anterioridad, por lo que esperamos que el solver de GFT forme parte del kit de herramientas de los investigadores de SPLs en el área del análisis automático.

2. Árboles de características generalizados

Los árboles de características generalizados (*generalized feature trees*, GFT), fueron introducidos por Broek et al. [6] en su propuesta para eliminar las restricciones textuales de un modelo de características [7]. Sin embargo, Broek et al. [7] solo consideró restricciones simples (del tipo *requires* y *excludes*). Posteriormente, Knüppel et al. [20] propuso eliminar las restricciones complejas (fórmulas arbitrarias en lógica proposicional) dando lugar a modelos con un número enorme de características y de restricciones simples. Usando primero el enfoque de Knüppel et al. [20] para eliminar las restricciones complejas y a continuación aplicando el enfoque de Broek et al. [7] se pueden eliminar, en teoría, las restricciones de cualquier modelo de características. Sin embargo, en la práctica, los modelos resultantes son tan grandes que analizarlos, o incluso gestionarlos (leerlos y escribirlos en fichero) resulta imposible. Esto se debe a que los modelos duplican el tamaño con la eliminación de cada restricción textual. A pesar de ello, existen multitud de modelos de características reales usados en la comunidad de SPL que o bien no tienen restricciones, o bien se han podido refactorizar [19] a un modelo sin restricciones siguiendo el enfoque anterior (ver Sección 4).

A continuación, formalizamos un modelo GFT siguiendo los mismos conceptos que se usan en SPL para definir un modelo de características [17,26]:

Definición 1 (Árbol de características generalizado (GFT)). *Un árbol de características generalizado (GFT) es un modelo de variabilidad sin restricciones textuales y con un árbol de características donde las características pueden tener múltiples ocurrencias. Formalmente, un GFT es una tripleta $(\mathcal{F}, r, \mathcal{R})$, donde:*

- *$\mathcal{F}$ es un lista finita de características (puede haber duplicados).*
- *$r \in F$ es la característica raíz del árbol.*
- *$\mathcal{R} \subseteq \mathcal{F} \times \mathcal{F}^n \times \mathbb{N}^2$ es el conjunto finito de relaciones que descomponen las características en una jerarquía organizada con estructura de árbol. Cada relación $r \in \mathcal{R}$ se define como $r = \big(f, [g_1, g_2, \ldots, g_n], \langle a..b \rangle\big)$ donde la característica f es el padre de las sub-características $g_i, 1 \leq i \leq n$, con multiplicidad $\langle a..b \rangle$ [8]. Cuando la característica f se incluye en una configuración, al menos a y como mucho b de las sub-características g_i deben incluirse también en la configuración. De esta forma, las características del árbol se descomponen en **"grupos or"** ($\langle 1..n \rangle$ y $n > 1$), **"grupos alternativos"** a.k.a. **"grupos xor"** ($\langle 1..1 \rangle$ y $n > 1$), **"características opcionales"** ($\langle 0..1 \rangle$ y $n = 1$), y **"características obligatorias"** ($\langle 1..1 \rangle$ y $n = 1$).*

Esta definición nos va a permitir definir las operaciones de análisis siguiendo la misma formalización. Además, la herramienta Flama [11], implementa esta formalización para sus modelos de características, por lo que podremos reutilizar dichos conceptos en la implementación del solver GFT.

Para facilitar la escritura de los algoritmos en el artículo, introducimos las siguientes funciones de ayuda:

- `parent`: $\boldsymbol{\mathcal{F} \rightarrow \mathcal{F} \cup \emptyset}$. Devuelve el padre de la característica $f \in \mathcal{F}$ o $\emptyset$ si f es la raíz.
- `relation`: $\boldsymbol{\mathcal{F} \times \mathcal{F} \rightarrow \mathcal{R}}$. Dadas dos características $f, g \in \mathcal{F}$ devuelve la relación de f como padre conteniendo a g como hijo. Una relación $r \in \mathcal{R}$ tiene cuatro atributos:

`parent`: $\mathcal{F}$. El padre de la relación, que es f.
`children`: $\mathcal{F}^n$. Conjunto de sub-características.
`card_min`: $\mathbb{N}$. Cardinalidad (o multiplicidad) mínima.
`card_max`: $\mathbb{N}$. Cardinalidad (o multiplicidad) máxima.

- `is_optional`: $\mathcal{F} \rightarrow \mathbb{B}$. Devuelve verdadero si la característica $f \in \mathcal{F}$ es opcional.
- `is_mandatory`: $\mathcal{F} \rightarrow \mathbb{B}$. Devuelve verdadero si la característica $f \in \mathcal{F}$ es obligatoria.
- `is_or_group`: $\mathcal{F} \rightarrow \mathbb{B}$. Devuelve verdadero si la característica $f \in \mathcal{F}$ es un grupo "or".
- `is_alternative_group`: $\mathcal{F} \rightarrow \mathbb{B}$. Devuelve verdadero si $f \in \mathcal{F}$ es un grupo "xor".
- `is_group`: $\mathcal{F} \rightarrow \mathbb{B}$. Devuelve verdadero si $f \in \mathcal{F}$ es un grupo ("or" ó "xor").
- `is_leaf`: $\mathcal{F} \rightarrow \mathbb{B}$. Devuelve verdadero si la característica $f \in \mathcal{F}$ no tiene hijos.

3. Operaciones de análisis sobre GFTs

Nuestra propuesta propone analizar los GFTs mediante operaciones de análisis recursivas sobre la estructura jerárquica del árbol, sin necesidad de transformar los modelos de características a ningún formalismo o codificarlos utilizando otra estructura adicional (e.g., BDD). Para comparar nuestro solver de GFT con los solvers tradicionales como SAT y BDD, vamos a utilizar las siguientes operaciones habituales de análisis [4] (Algoritmos 1 a 4). La entrada a todas estas operaciones será un GFT, de acuerdo a la definición proporcionada en la sección anterior (Definición 1).

Operación "Valid" (Algorithm 1). Un GFT es válido si representa al menos un producto. Cuando no hay restricciones, un árbol de característica no vacío es siempre válido [4]. Por tanto, un GFT es válido si su lista de características ($\mathcal{T}.\mathcal{F}$) no está vacía, ya que al menos representará un producto.

Algorithm 1 Valid (Modelo válido).

Input: $\mathcal{T}$: El árbol de características generalizado (GFT).
Output: *True* si $\mathcal{T}$ representa al menos un producto; *False* en otro caso.
1: **function** VALID($\mathcal{T}$)
2: **return** $\mathcal{T}.\mathcal{F} \neq \emptyset$ ▷ Un GFT tiene productos si y solo si su árbol $\mathcal{T}.\mathcal{F}$ no está vacío.
3: **end function**

Operación "Products number" (Algorithm 2). El número de productos en un GFT se calcula recorriendo de forma recursiva la estructura del árbol $\mathcal{T}$ desde la raíz hasta las hojas [1], teniendo en cuenta para el cálculo el tipo de relación de cada característica (ver líneas 8 a 14).

Operación "Core features" (Algorithm 3). Las características básicas ("core") son aquellas que están presentes en todos los productos [4]. En un GFT se calculan también recorriendo la estructura del árbol $\mathcal{T}$ a partir de la raíz, y lo forman la raíz y todos sus hijos obligatorios calculados recursivamente. Puesto que un GFT puede contener características duplicadas y estas características

Algorithm 2 Number of products (Número de productos).

Input: $\mathcal{T}$: El árbol de características generalizado (GFT).
Output: El número de productos de $\mathcal{T}$.
1: **function** ProductsNumber(T)
2: **return** ProductsNumberRec($\mathcal{T}$, $\mathcal{T}.r$) ▷ Función recursiva.
3: **end function**
Input: $\mathcal{T}$: El modelo GFT, f: La característica raíz ($\mathcal{T}.r$).
Output: El número de productos de $\mathcal{T}$.
4: **function** ProductsNumberRec($\mathcal{T}$, f)
5: **if** $is\_leaf(f)$ **then return** 1 ▷ La característica f no tiene hijos.
6: **else**
7: $n \leftarrow 1$
8: **for all** $g \in \mathcal{T}.\mathcal{F} | \exists relation(f, g) \in \mathcal{T}.\mathcal{R}$ **do**
9: **if** `is_mandatory(`g`)` **then** $n \leftarrow n *$ ProductsNumberRec($\mathcal{T}$, g)
10: **else if** `is_optional(`g`)` **then** $n \leftarrow n * (1 +$ ProductsNumberRec($\mathcal{T}$, g))
11: **else if** `is_alternative(`f`)` **then** $n \leftarrow n * (\sum$ ProductsNumberRec($\mathcal{T}$, g))
12: **else if** `is_or_group(`f`)` **then** $n \leftarrow n * (\prod(1 +$ ProductsNumberRec($\mathcal{T}$, g))$-1)$
13: **end if**
14: **end for**
15: **end if**
16: **return** n
17: **end function**

siempre aparecen como hijas en un grupo "xor", si el grupo "xor" forma parte de las características "core", su característica hija también será una característica "core" (ver líneas 6 a 9). En el ejemplo del GFT de la Figura 1 las características "core" serían `XOR`[1] y `A`.

Algorithm 3 Core features (Características básicas).

Input: $\mathcal{T}$: El árbol de características generalizado (GFT).
Output: Las características básicas ("core") de $\mathcal{T}$.
1: **function** CoreFeatures(T)
2: **return** CoreFeaturesRec($\mathcal{T}$, $\mathcal{T}.r$) ▷ Función recursiva.
3: **end function**
Input: $\mathcal{T}$: El modelo GFT, f: La característica raíz ($\mathcal{T}.r$).
Output: Las características básicas ("core") de $\mathcal{T}$.
4: **function** CoreFeaturesRec($\mathcal{T}$, f)
5: $cores \leftarrow \{f\}$
6: $children \leftarrow \{g \in \mathcal{T}.\mathcal{F} | \exists relation(f, g) \in \mathcal{T}.\mathcal{R}\}$
7: **if** `is_alternative(`f`)` $\wedge$ $|children| = 1$ **then** ▷ Los hijos son características duplicadas.
8: $cores \leftarrow cores \bigcup$ (CoreFeaturesRec($\mathcal{T}$, g), $g \in children$)
9: **end if**
10: **for all** $child \in children$ **do**
11: **if** `is_mandatory(`$child$`)` **then**
12: $cores \leftarrow cores \bigcup$ CoreFeaturesRec($\mathcal{T}$, $child$)
13: **end if**
14: **end for**
15: **return** $cores$
16: **end function**

Operación "Dead features" (Algorithm 3). Las características muertas ("dead") son aquellas que no aparecen en ningún producto. Un GFT no tiene características muertas debido a que estas están causadas por las restricciones textuales y un GFT no tiene restricciones textuales.

[1] La característica `XOR` es abstracta y en los productos aparece filtrada.

Algorithm 4 Dead features (Características muertas).

Input: $\mathcal{T}$:El árbol de características generalizado (GFT).
Output: Las características muertas ("dead") de $\mathcal{T}$.
1: **function** DEADFEATURES($\mathcal{T}$)
2: **return** $\emptyset$ ▷ Un GFT nunca tiene características muertas.
3: **end function**

4. Evaluación

En esta sección evaluamos cuantitativamente las operaciones de análisis sobre modelos GFT. Para ello hemos implementado las operaciones presentadas en la Sección 3 en Python dentro del ecosistema de Flama [11]. Hemos elegido esta herramienta porque ya dispone de las utilidades necesarias para leer, transformar, y gestionar modelos de características, así como para analizarlos con diferentes solvers. En concreto, en este artículo comparamos la eficiencia de nuestras operaciones con las mismas operaciones realizadas con un solver SAT y un solver BDD. El artefacto software está disponible online[2]

4.1. Configuración de la experimentación

La Tabla 1 muestra los 10 modelos que hemos considerado para la evaluación. Se trata de modelos de diferente naturaleza usados habitualmente en la evaluación de propuestas en la comunidad de SPL y cuyas versiones GFT están disponibles. Aunque existen modelos de mayor tamaño en cuanto al número de características, no existe una versión del GFT para esos modelos tan grandes. Esto se debe a que el GFT duplica el tamaño del model original con cada restricción textual. Los modelos GFT considerados contienen hasta 28.897 características (véase el modelo `BerkeleyDB`). A partir de ese tamaño, los modelos GFT no son factibles pues la gestión del propio modelo (lectura, serialización, almacenamiento en memoria, recorrido del árbol, etc.) llegando a requerir más tiempo que el propio análisis. El formato de los modelos en todos los casos es UVL (*Universal Variability Language*) [27], el nuevo lenguaje universal para especificar la variabilidad que está teniendo éxito en la comunidad SPL [3].

Para el análisis con SAT se ha usado el solver Glucose3 [2], mientras que para BDD se ha usado el solver *BDD4va*[3] propuesto en [14], ambos disponibles en Flama [11]. Para las operaciones de SAT y BDD se han usado los modelos en su versión original con restricciones debido a que los modelos GFT no pueden ser analizados directamente con SAT o BDD como se explicó en la Sección 1.

Los experimentos se han realizado en un ordenador de sobremesa Intel Core i5-10400 CPU @ 3.6GHz x 6, 16 GB RAM, Linux Mint 21.1, y Python 3.10. Para cada operación de análisis y modelo se han realizado 30 ejecuciones y calculado las medianas, medias y desviaciones típicas del tiempo de ejecución. Los tiempos (en segundos) solo incluyen el análisis, y en ningún caso se tiene en cuenta el tiempo de lectura o transformación de los modelos a la formalización necesaria

[2] *Solver GFT*: https://doi.org/10.5281/zenodo.7892907
[3] BDD4va: https://github.com/rheradio/bdd4va

Tabla 1. Modelos de características usados en la evaluación. Se muestra el número de características ($\#\mathcal{F}$), restricciones ($\#\mathcal{CTC}$), y productos (#Productos) del modelo original; y el número de características ($\#\mathcal{F}$), el incremento en tamaño (%) y el número de productos (#Productos) del modelo GFT correspondiente.

Nombre	Modelo original			GFT		
	$\#\mathcal{F}$	$\#\mathcal{CTC}$	#Productos	$\#\mathcal{F}$	Incremento (%)	#Productos
Tank War	37	0	1,74e6	37	0 %	1,74e6
Pizzas	12	2	25	131	992 %	25
Mobile Media	43	3	2,12e6	303	605 %	2,12e6
Apo-Games	63	14	4,30e8	328	421 %	4,30e8
BerkeleyDB Cache	19	29	3.840	522	2,6e3 %	3.840
Truck	33	10	234	745	2.158 %	234
JHipster	45	13	26.256	2.027	4,4e3 %	26.256
Graphs VIZ	86	19	37.706	5.134	5.870 %	37.706
WeaFQAs	179	7	2,93e24	19.713	1,1e4 %	2,93e24
BerkeleyDB	76	20	4,08e9	28.897	3,8e4 %	4,08e9

(SAT, BDD, o GFT), ya que el tamaño de los modelos varía de versión original a su versión codificada o a su versión GFT.

4.2. Resultados y discusión

En primer lugar, en la Tabla 1 se puede observar que la semántica de los modelos (i.e., los productos) se mantiene siempre en el modelo GFT con respecto al modelo original. Se puede observar también el incremento exponencial en el tamaño de los modelos GFT con respecto a su versión original, llegando a un incremento de 3,8e4 % en el caso del modelo `BerkeleyDB`. De hecho, no hemos encontrado ningún modelo GFT mayor que `BerkeleyDB`, y esto se debe principalmente a la imposibilidad de serializar modelos mayores donde el GFT puede superar fácilmente el millón de características. Las Figuras 2 a 5 muestran los resultados de la evaluación para cada operación de análisis.

Operación "Valid" (Figura 2). Esta operación en GFT es constante e instantánea, ya que un GFT es válido siempre y cuando tenga al menos una característica presente en el árbol. En SAT, la operación "Valid" es también muy eficiente para los modelos evaluados, pero para modelos más grandes esta operación puede llegar a ser un problema NP [24]. En BDD, el tiempo es constante [23], aunque un poco mayor en comparación con GFT porque se debe recorrer la estructura al menos una vez para encontrar una asignación válida. En cualquier caso, estamos hablando de tiempos muy pequeños del orden de 10 milisegundos.

Operación "Products Number" (Figura 3). En el contexto de SPL esta operación suele realizarse con solvers BDD ya que los BDD estan pensados para contar el número de productos de forma muy eficiente, siendo su complejidad lineal. Es más, esta operación con BDD sí se puede aplicar sobre un modelo GFT ya que se tiene en cuenta el número de asignaciones posibles y no intervienen las duplicidades de las características. Como se observa en la Figura 3, el rendimiento con BDD empeora cuando se analiza el GFT en comparación con el modelo original debido al incremento en tamaño del GFT. Nuestra operación sobre el GFT es incluso más eficiente que el BDD para modelos de hasta 10.000 características.

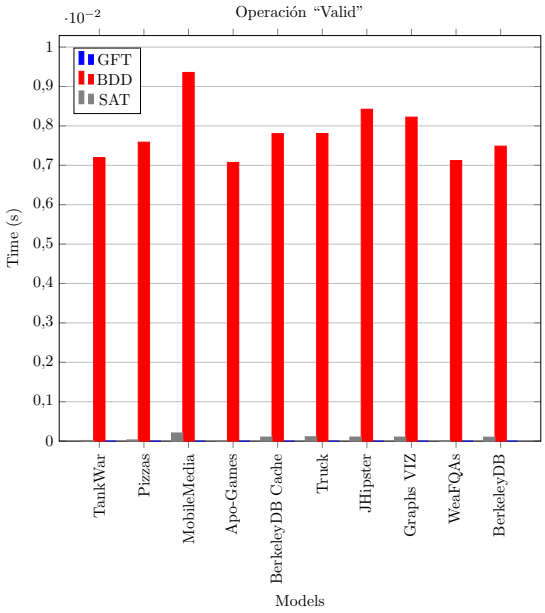

Figura 2. Comparativa de la operación "Valid" con el solver GFT, y un solver SAT y BDD sobre el modelo original.

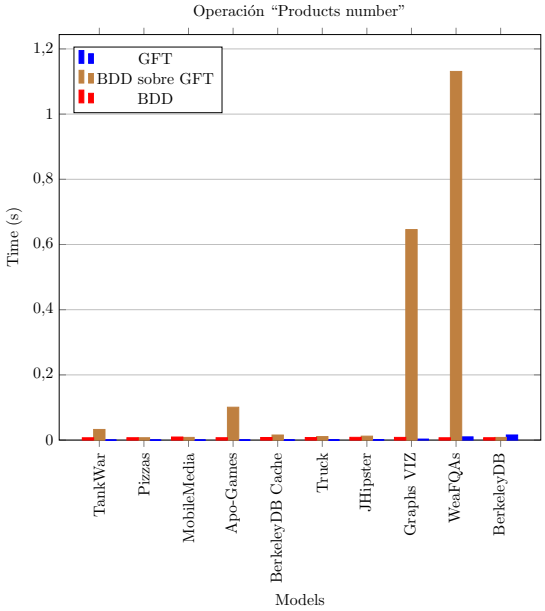

Figura 3. Comparativa de la operación "Products number" con el solver GFT, un solver BDD sobre el modelo GFT, y un solver BDD sobre el modelo original.

Por ejemplo, contar el número de productos del modelo `Graphs VIZ` con el solver GFT tarda 3 milisegundos, mientras que el solver BDD necesita 9 milisegundos. A partir de ese tamaño, el BDD supera al solver GFT (e.g., modelos `WeaFQAs`

y `BerkeleyDB`). Con SAT esta operación no tiene sentido, pues SAT enumera todas los productos para poder contarlos, lo que es inviable cuando el número de productos es colosal, independientemente del tamaño de los modelos.

Operación "Core features" (Figura 4). Esta operación es muy eficiente usando SAT solvers [24] consiguiendo mejores resultados que con otros solvers. En BDD esta operación se implementa a partir de la distribución de probabilidad de las características [15] donde las "core features" son aquellas características que están presentes en el 100 % de los productos. Usando el solver GFT, esta operación es más eficiente que el BDD para modelos pequeños, ya que solo hay que analizar las características obligatorias desde la raíz del árbol. A partir de 10.000 características, el rendimiento empeora debido al gran número de ramas en el árbol.

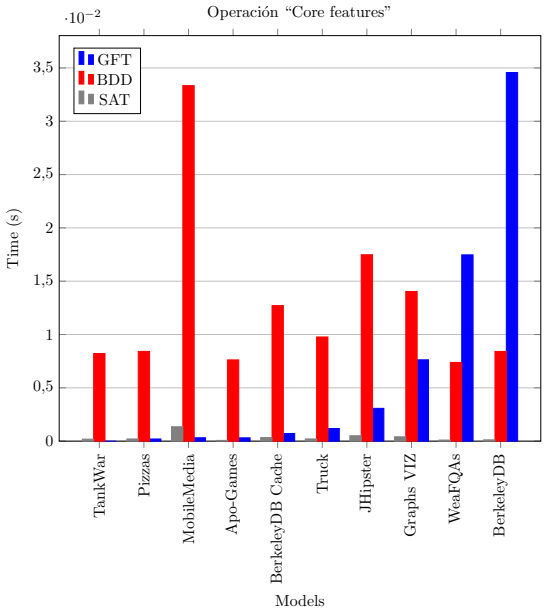

Figura 4. Comparativa de la operación "Core Features" con el solver GFT, y un solver SAT y BDD sobre el modelo original.

Operación "Dead features" (Figura 5). Al igual que la operación de "Core features", las "Dead features" se calculan de forma muy eficiente usando SAT solvers puesto que se implementan de la misma forma [24]. En BDD ocurre lo mismo, se calculan a partir de la distribución de probabilidad de las características [15] donde las "dead features" son aquellas características con un 0 % de probabilidad (i.e., no están presentes en ningún producto). Con el solver GFT, esta operación es constante e instantánea, ya que un GFT no tiene características muertas, siendo el tiempo de la operación despreciable (cercano a cero).

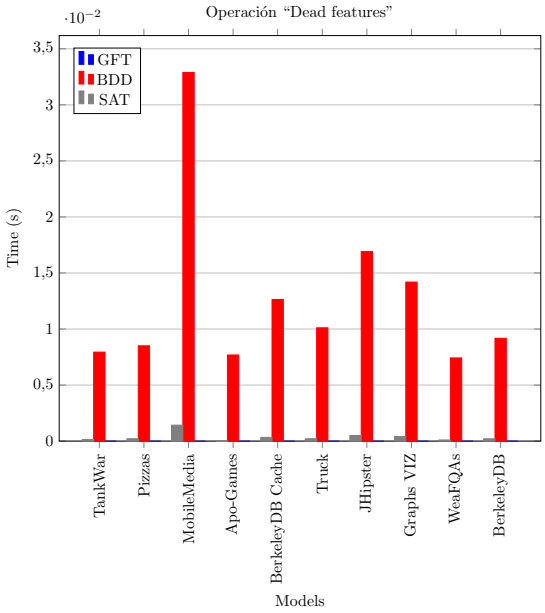

Figura 5. Comparativa de la operación "Dead Features" con el solver GFT, y un solver SAT y BDD sobre el modelo original.

5. Trabajo relacionado

Para realizar el análisis sobre un GFT, hay que partir de la base de que o bien el modelo de características original no tiene restricciones, o bien debe ser posible eliminar primero cualquier tipo de restricciones, tanto las simples como las complejas. Hay cinco trabajos principales centrados en la eliminación de restricciones [7,6,13,20,21]. Cronológicamente, Broek et al. se centró en eliminar primero las restricciones simples [7] para después analizar los resultados del árbol resultante definiendo las operaciones directamente sobre el árbol [6]. Los principales problemas de este trabajo es que no tenía en cuenta restricciones complejas y que no proporcionaba una evaluación práctica de este enfoque en comparación con otros enfoques de análisis existentes. Otro trabajo es el de Gil et al. [13], que abordó el problema de la eliminación de restricciones desde una perspectiva teórica, siguiendo el enfoque de Czarnecki y Wasowski [9] de trasladar los modelos de características a lógica y luego a un árbol de características, probando que era posible eliminar todas las restricciones al precio de introducir un nuevo conjunto de características. En este trabajo se demostró que el incremento exponencial en el tamaño del árbol era inevitable, aunque como se ha dicho era un trabajo teórico donde no se discutieron las limitaciones prácticas de la propuesta. Por último, el trabajo de Knüppel et al. [20] y su tesis doctoral [21] se centraron en la eliminación de las restricciones complejas utilizando características abstractas y restricciones simples. En este trabajo se incrementa significativamente el número de características abstractas auxiliares, lo que

podría provocar problemas de escalabilidad respecto al tamaño en memoria del árbol. Además, al ser la salida un modelo de características con restricciones simples, el modelo resultante no podría utilizarse directamente en nuestra propuesta, aunque podría combinarse con otros trabajos que eliminarían en un paso posterior las restricciones simples.

Respecto al análisis automático de modelos de características haciendo uso de la estructura jerárquica del árbol, como alternativa a utilizar un solver SAT [22], #SAT [28], CSP [4] o BDD [16], también Broek et al. [6] propuso algunas operaciones de análisis similares a nuestros Algoritmos 1-4, pero implementados en el lenguaje funcional Miranda [31], un lenguaje que se ha quedado obsoleto hoy en día. Esto hace que sea difícil aplicar sus algoritmos a los entornos para el modelado y análisis de características actuales [18], como FeatureIDE [30], pure::variants [5], o Flama [11]. Hay otras propuestas híbridas [23,10] que se basan en un motor de razonamiento para resolver las restricciones textuales, mientras otros módulos trabajan en el árbol de características. For ejemplo, Fernandez-Amoros et al. [10] propone *treecount*, un algoritmo para contar los productos en un modelo que recorre el árbol de característica al mismo tiempo que razona sobre las restricciones utilizando un motor de razonamiento, para evitar repetir dos veces los mismos cálculos. De forma similar, Mendonça et al. [23] define el *Feature Model Reasoning System* (FMRS) que propaga las restricciones como parte del árbol. Estas propuestas [23,10] soportan solo operaciones muy concretas, como contar el número de productos o calcular las características comunes [10].

6. Conclusiones y trabajo futuro

En este artículo hemos presentado un analizador de modelos de variabilidad basado en el árbol de características que pone en práctica el concepto de árbol de características generalizado (GFT) y sus correspondientes operaciones de análisis. Por un lado, se ha demostrado que es posible analizar un modelo de características sin tener que recurrir a un solver SAT o BDD siempre que el modelo no tenga restricciones o se haya transformado previamente a un GFT. Además, el análisis directo sobre la estructura en árbol del GFT es más eficiente que usando SAT o BDD para modelos GFT de hasta 10.000 características. Por el contrario, la principal desventaja de los modelos GFT es su tamaño, ya que un modelo de características de tamaño medio (100 características) puede convertirse en un modelo GFT de tamaño colosal (20.000 características) que puede resultar difícil de gestionar en la práctica.

Con esta contribución esperamos que se vuelva a despertar el interés por este tipo de analizadores y se vuelva a retomar la investigación en los problemas abiertos que supone este tipo de análisis. Por ejemplo, usando *técnicas de paralelismo* [12] sobre el GFT que permitan explotar el echo de tener características duplicadas en grupos "xor", o usando *programación dinámica* [7] para evitar analizar ramas duplicadas. Además, un solver GFT que permita contar el número de producto sin tener que recurrir a un solver BDD abre la puerta a la definición de análisis estadísticos de los modelos de características [15] como

operaciones de *sampling* [14], de *distribución de productos* [10], o de *probabilidades de las características* [15] que hasta ahora solo eran posibles usando un solver BDD [29].

Como trabajo futuro se plantea incorporar otras operaciones de análisis más complejas sobre el GFT como la enumeración de productos, o las operaciones estadísticas mencionadas [14] que se basan en el conteo de productos, una operación muy eficiente en GFT. También planteamos el uso de los conceptos de *importación de modelos* en UVL [27] o *interfaces y composición de modelos de características* [32] que permiten modularizar los modelos grandes para lidiar con el problema del tamaño de los GFT.

Artefacto software *Solver GFT*: https://doi.org/10.5281/zenodo.7892907

Agradecimientos Trabajo financiado por los proyectos *IRIS* PID2021-122812OB-I00 (con fondos FEDER), *LEIA* UMA18-FEDERJA-157, y *DAEMON* H2020-101017109; y por la Universidad de Málaga.

Referencias

1. Apel, S., Batory, D.S., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines - Concepts and Implementation. Springer (2013). https://doi.org/10.1007/978-3-642-37521-7, https://doi.org/10.1007/978-3-642-37521-7
2. Audemard, G., Simon, L.: On the glucose SAT solver. Int. J. Artif. Intell. Tools **27**(1), 1840001:1–1840001:25 (2018). https://doi.org/10.1142/S0218213018400018, https://doi.org/10.1142/S0218213018400018
3. Benavides, D., Rabiser, R., Batory, D.S., Acher, M.: First international workshop on languages for modelling variability (MODEVAR). In: 23rd International Systems and Software Product Line Conference (SPLC) , Volume A. p. 46:1 (2019). https://doi.org/10.1145/3336294.3342364
4. Benavides, D., Segura, S., Cortés, A.R.: Automated analysis of feature models 20 years later: A literature review. Inf. Syst. **35**(6), 615–636 (2010). https://doi.org/10.1016/j.is.2010.01.001, https://doi.org/10.1016/j.is.2010.01.001
5. Beuche, D.: Industrial variant management with pure: : variants. In: 23rd International Systems and Software Product Line Conference (SPLC). vol. B, pp. 64:1–64:3 (2019). https://doi.org/10.1145/3307630.3342391, https://doi.org/10.1145/3307630.3342391
6. van den Broek, P., Galvão, I.: Analysis of feature models using generalised feature trees. In: 3rd International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS). vol. 29, pp. 29–35 (2009), http://www.vamos-workshop.net/proceedings/VaMoS_2009_Proceedings.pdf
7. van den Broek, P., Galvão, I., Noppen, J.: Elimination of constraints from feature trees. In: Workshop on Analyses of Software Product Lines (ASPL) @ SPLC'08. vol. 2, pp. 227–232 (2008)
8. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Formalizing cardinality-based feature models and their specialization. Softw. Process. Improv. Pract. **10**(1), 7–29 (2005). https://doi.org/10.1002/spip.213, https://doi.org/10.1002/spip.213
9. Czarnecki, K., Wasowski, A.: Feature diagrams and logics: There and back again. In: 11th International Software Product Lines Conference (SPLC). pp. 23–34 (2007). https://doi.org/10.1109/SPLINE.2007.24, https://doi.org/10.1109/SPLINE.2007.24

10. Fernández-Amorós, D., Heradio, R., Cerrada, J.A., Cerrada, C.: A scalable approach to exact model and commonality counting for extended feature models. IEEE Trans. Software Eng. **40**(9), 895–910 (2014). https://doi.org/10.1109/TSE.2014.2331073, https://doi.org/10.1109/TSE.2014.2331073
11. Galindo, J.A., Benavides, D.: A python framework for the automated analysis of feature models: A first step to integrate community efforts. In: 24th ACM International Systems and Software Product Line Conference (SPLC). vol. B, pp. 52–55 (2020). https://doi.org/10.1145/3382026.3425773, https://doi.org/10.1145/3382026.3425773
12. Galindo, J.A., Acher, M., Tirado, J.M., Vidal, C., Baudry, B., Benavides, D.: Exploiting the enumeration of all feature model configurations: a new perspective with distributed computing. In: 20th International Systems and Software Product Line Conference (SPLC). pp. 74–78 (2016). https://doi.org/10.1145/2934466.2934478, https://doi.org/10.1145/2934466.2934478
13. Gil, Y., Kremer-Davidson, S., Maman, I.: Sans constraints? feature diagrams vs. feature models. In: 14th International Software Product Lines Conference (SPLC): Going Beyond. vol. 6287, pp. 271–285 (2010). https://doi.org/10.1007/978-3-642-15579-6_19, https://doi.org/10.1007/978-3-642-15579-6_19
14. Heradio, R., Fernández-Amorós, D., Galindo, J.A., Benavides, D., Batory, D.S.: Uniform and scalable sampling of highly configurable systems. Empir. Softw. Eng. **27**(2), 44 (2022). https://doi.org/10.1007/s10664-021-10102-5, https://doi.org/10.1007/s10664-021-10102-5
15. Heradio, R., Fernández-Amorós, D., Mayr-Dorn, C., Egyed, A.: Supporting the statistical analysis of variability models. In: 41st International Conference on Software Engineering (ICSE). pp. 843–853 (2019). https://doi.org/10.1109/ICSE.2019.00091, https://doi.org/10.1109/ICSE.2019.00091
16. Heradio, R., Perez-Morago, H., Fernández-Amorós, D., Bean, R., Cabrerizo, F.J., Cerrada, C., Herrera-Viedma, E.: Binary decision diagram algorithms to perform hard analysis operations on variability models. In: 15th New Trends in Software Methodologies, Tools and Techniques (SoMeT). vol. 286, pp. 139–154 (2016). https://doi.org/10.3233/978-1-61499-674-3-139, https://doi.org/10.3233/978-1-61499-674-3-139
17. Horcas, J.M., Galindo, J.A., Heradio, R., Fernández-Amorós, D., Benavides, D.: A monte carlo tree search conceptual framework for feature model analyses. J. Syst. Softw. **195**, 111551 (2023). https://doi.org/10.1016/j.jss.2022.111551, https://doi.org/10.1016/j.jss.2022.111551
18. Horcas, J.M., Pinto, M., Fuentes, L.: Empirical analysis of the tool support for software product lines. Softw. Syst. Model. **22**(1), 377–414 (2023). https://doi.org/10.1007/s10270-022-01011-2, https://doi.org/10.1007/s10270-022-01011-2
19. Horcas, J.M., Pinto, M., Fuentes, L.: A modular metamodel and refactoring rules to achieve software product line interoperability. J. Syst. Softw. **197**, 111579 (2023). https://doi.org/10.1016/j.jss.2022.111579, https://doi.org/10.1016/j.jss.2022.111579
20. Knüppel, A., Thüm, T., Mennicke, S., Meinicke, J., Schaefer, I.: Is there a mismatch between real-world feature models and product-line research? In: 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE). pp. 291–302 (2017). https://doi.org/10.1145/3106237.3106252, https://doi.org/10.1145/3106237.3106252
21. Knüppel, A.: The Role of Complex Constraints in Feature Modeling. Master's thesis, Technische Universität Braunschweig (2016), https://www.isf.cs.tu-bs.de/cms/team/knueppel/downloads/thesisKnueppel16.pdf

22. Liang, J.H., Ganesh, V., Czarnecki, K., Raman, V.: Sat-based analysis of large real-world feature models is easy. In: 19th International Conference on Software Product Line (SPLC). p. 91–100 (2015). https://doi.org/10.1145/2791060.2791070, https://doi.org/10.1145/2791060.2791070
23. Mendonça, M.: Efficient Reasoning Techniques for Large Scale Feature Models. Ph.D. thesis, University of Waterloo (2009), https://hdl.handle.net/10012/4201
24. Mendonça, M., Wasowski, A., Czarnecki, K.: SAT-based analysis of feature models is easy. In: 13th International Software Product Lines Conference (SPLC). vol. 446, pp. 231–240 (2009), https://dl.acm.org/citation.cfm?id=1753267
25. Raatikainen, M., Tiihonen, J., Männistö, T.: Software product lines and variability modeling: A tertiary study. Journal of Systems and Software **149**, 485–510 (2019). https://doi.org/https://doi.org/10.1016/j.jss.2018.12.027, https://www.sciencedirect.com/science/article/pii/S016412121830284X
26. Schobbens, P., Heymans, P., Trigaux, J., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Networks **51**(2), 456–479 (2007). https://doi.org/10.1016/j.comnet.2006.08.008, https://doi.org/10.1016/j.comnet.2006.08.008
27. Sundermann, C., Feichtinger, K., Engelhardt, D., Rabiser, R., Thüm, T.: Yet another textual variability language?: a community effort towards a unified language. In: 25th ACM International Systems and Software Product Line Conference (SPLC). vol. A, pp. 136–147 (2021). https://doi.org/10.1145/3461001.3471145, https://doi.org/10.1145/3461001.3471145
28. Sundermann, C., Heß, T., Nieke, M., Bittner, P.M., Young, J.M., Thüm, T., Schaefer, I.: Evaluating state-of-the-art # SAT solvers on industrial configuration spaces. Empir. Softw. Eng. **28**(2), 29 (2023). https://doi.org/10.1007/s10664-022-10265-9, https://doi.org/10.1007/s10664-022-10265-9
29. Sundermann, C., Nieke, M., Bittner, P.M., Heß, T., Thüm, T., Schaefer, I.: Applications of #sat solvers on feature models. In: 15th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS). pp. 12:1–12:10 (2021). https://doi.org/10.1145/3442391.3442404, https://doi.org/10.1145/3442391.3442404
30. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Sci. Comput. Program. **79**, 70–85 (2014). https://doi.org/10.1016/j.scico.2012.06.002, https://doi.org/10.1016/j.scico.2012.06.002
31. Turner, D.A.: Miranda: A non-strict functional language with polymorphic types. In: Functional Programming Languages and Computer Architecture (FPCA). vol. 201, pp. 1–16 (1985). https://doi.org/10.1007/3-540-15975-4_26, https://doi.org/10.1007/3-540-15975-4_26
32. Urli, S., Blay-Fornarino, M., Collet, P., Mosser, S.: Using composite feature models to support agile software product line evolution. In: Proceedings of the 6th International Workshop on Models and Evolution (ME@MoDELS). pp. 21–26 (2012). https://doi.org/10.1145/2523599.2523604, https://doi.org/10.1145/2523599.2523604