Aplicando programación lineal entera a la búsqueda de conjuntos de productos de prueba priorizados para líneas de productos software

Javier Ferrer[1], Francisco Chicano[1],
Roberto E. Lopez-Herrejon[2] y Enrique Alba[1]

[1] Universidad de Málaga, Málaga, Spain
{ferrer, chicano, eat}@lcc.uma.es
[2] Systems Engineering and Automation
Johannes Kepler University, Linz, Austria
roberto.lopez@jku.at

Resumen Las líneas de productos software son familias de productos que están íntimamente relacionados entre sí, normalmente formados por combinaciones de un conjunto de características software. Generalmente no es factible testar todos los productos de la familia, ya que el número de productos es muy elevado debido a la explosión combinatoria de características. Por este motivo, se han propuesto criterios de cobertura que pretenden probar al menos todas las interacciones entre características sin necesidad de probar todos los productos, por ejemplo todos los pares de características (*pairwise coverage*). Además, es deseable testar primero los productos compuestos por un conjunto de características prioritarias. Este problema es conocido como *Prioritized Pairwise Test Data Generation*. En este trabajo proponemos una técnica basada en programación lineal entera para generar este conjunto de pruebas priorizado. Nuestro estudio revela que la propuesta basada en programación lineal entera consigue mejores resultados estadísticamente tanto en calidad como en tiempo de computación con respecto a las técnicas existentes para este problema.

Palabras clave Pruebas de interacción combinatoria, líneas de productos software, prioridades, programación lineal entera

1. Introducción

Las líneas de productos software (*SPL*) son familias de sistemas software relacionados, las cuales poseen diferentes combinaciones de características [1]. La gestión efectiva de la variabilidad es crucial para obtener beneficios de las SPLs como el incremento en la reutilización, la personalización rápida, y la reducción del tiempo de llegada al mercado. Debido al gran número de combinaciones de características que es típico en las SPLs, los modelos con alta variabilidad suponen un desafío para el campo de las pruebas de programas. Recientemente se

han propuesto muchos enfoques de pruebas [2–5]), sin embargo, todavía hay potencial de mejora ya que la mayoría de enfoques ya propuestos son aproximados y no suelen conseguir la solución óptima. Por contra, nosotros proponemos un enfoque basado en Programación Lineal Entera (en inglés *Integer Linear Programming* - ILP) para la generación del conjunto mínimo de productos de prueba en SPLs. Aunque la resolución de programas lineales enteros es un problema NP-difícil en general, con un coste computacional exponencial en el peor caso, los resolutores actuales, como CPLEX[3] o Gurobi[4], incluyen sofisticadas estrategias de búsqueda que les permiten resolver una gran cantidad de instancias de ILP en pocos segundos. Hasta donde llega nuestro conocimiento, esta técnica no se ha aplicado anteriormente a este problema.

En este artículo presentamos un algoritmo Ávido basado en Programación Lineal Entera (APLE) que genera un conjunto priorizado de pruebas para SPLs usando el criterio de cobertura *pairwise*, en el cual se deben cubrir todos los pares de características existentes en la SPL. APLE recibe como entrada un modelo de características (en inglés *Feature Model* - FM) y un conjunto de productos ponderados. Su objetivo es generar un conjunto de productos que cubren los pares de características deseados siguiendo diferentes esquemas de asignación de prioridades, con los que se van a generar diferentes ponderaciones para las características. Este esquema ha sido propuesto en [3] y ha sido recientemente aplicado de forma satisfactoria en la industria.

En nuestra experimentación validamos nuestra propuesta frente a dos algoritmos del estado del arte, un algoritmo ávido que genera soluciones competitivas en poco tiempo, llamado prioritized-ICPL (pICPL) [3] y un algoritmo genético llamado *Prioritized Pairwise Genetic Solver* (PPGS) [6] que obtiene soluciones de mejor calidad que pICPL pero generalmente usando un tiempo mayor. Nuestra comparativa abarca un total de 235 FMs con un amplio rango de características y productos, usando tres métodos de asignación de prioridades a los productos y cinco estrategias de selección de productos.

Nuestro estudio ha revelado que APLE obtiene conjuntos de pruebas priorizadas más pequeños en un tiempo menor para diferentes porcentajes de cobertura ponderada, con respecto a los resultados obtenidos con PPGS y pICPL. Estos resultados muestran que el uso de técnicas exactas en combinación con algoritmos ávidos constituye una estrategia muy efectiva para la generación de conjuntos de productos de prueba para SPLs. Nuestras principales contribuciones en este artículo son las siguientes:

- Propuesta de un algoritmo constructivo ávido basado en ILP.
- Evaluación del rendimiento de APLE en comparación con PPGS e pICPL.

El resto del artículo se organiza de la siguiente manera. En la Sección 2 presentamos los modelos de características. En la Sección 3 se formaliza el problema de la generación de conjuntos de productos de prueba priorizados en SPL. La Sección 4 describe nuestra propuesta algorítmica. En la Sección 5 presentamos los demás algoritmos objeto de la comparación, los métodos de asignación de

[3] `http://www-03.ibm.com/software/products/es/ibmilogcpleoptistud`
[4] `https://www.gurobi.com`

prioridades y las instancias usadas en la experimentación. La Sección 6 está dedicada al análisis estadístico de los resultados y la Sección 7 a describir las amenazas a la validez del estudio. Por último, en la Sección 8 ofrecemos las conclusiones obtenidas y los siguientes pasos de nuestra investigación.

2. Modelos de Características

Los modelos de características son un estándar *de facto* para modelar las características comunes y variables de un sistema (representadas por cajas etiquetadas) y sus relaciones (representadas con líneas) formando una estructura de tipo árbol, especificando así un conjunto de combinaciones de características que dan lugar a múltiples configuraciones diferentes [7]. Cada característica distinta de la raíz tiene una sola característica padre y puede tener un conjunto de características hijas. Nótese que una característica hija sólo puede ser incluida en una configuración si y sólo si, su padre es incluido también. Para ilustrar estos conceptos vamos a usar un FM (Figura 1) extraído del repositorio SPLOT [8].

Hay cuatro tipos de relaciones jerárquicas entre características:

- *Características opcionales:* son representadas con un círculo vacío e indica que esta característica puede o no ser seleccionada si su padre es seleccionado. En nuestra instancia de ejemplo sería la característica `Engine`.
- *Características obligatorias:* son representadas con un círculo relleno, y deben ser seleccionadas si su padre es seleccionado. En nuestra instancia de ejemplo serían obligatorias: `Wing` y `Materials`.
- *Relaciones Or-inclusivas:* son representadas como arcos independientes rellenos que abarcan un conjunto de líneas que conectan la característica padre con las características hijas. Indican que al menos una característica debe ser seleccionada si el padre es seleccionado. En nuestra instancia de ejemplo, la relación de `Wing` o `Materials` con sus hijos es de este tipo.
- *Relaciones Or-exclusivas:* son representadas como arcos independientes vacíos que abarcan un conjunto de líneas que conectan la característica padre con las características hijas. Indican que exactamente una característica debe ser seleccionada si el padre es seleccionado. En nuestra instancia de ejemplo la relaciones de `Engine` con sus respectivos hijos son Or-exclusivas.

Además de las relaciones padre-hijo, las características se pueden relacionar también con otras ramas del modelo de características, son las restricciones conocidas como *Cross-Tree Constraints (CTC)* [9]. Estas restricciones, así como las impuestas por las relaciones jerárquicas entre características, son expresadas y comprobadas usando lógica proposicional (para más detalles consultar [9]).

3. Formalización del Problema: *Prioritized Pairwise Test Data Generation*

En esta sección proporcionamos una descripción formal del problema de la generación de conjunto de pruebas priorizados por pares y del esquema de prioridades implementado por las propuestas algorítmicas estudiadas en este trabajo.

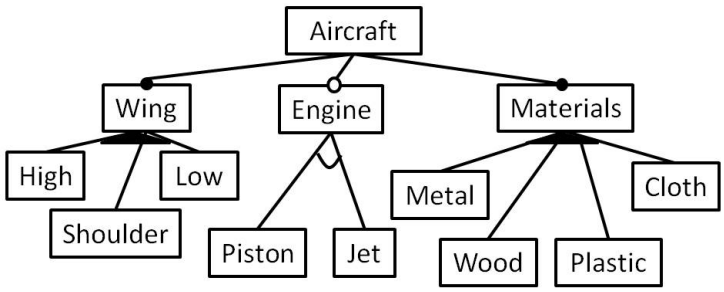

Figura 1. Modelo de características de Aircraft

Definición 1. *Lista de Características (LC) es la lista de característica del modelo de características.*

Definición 2. *Conjunto de Características (CC) es un par* $(sel, \overline{sel})$ *donde sel y* $\overline{sel}$ *son respectivamente los conjuntos de características seleccionadas y no seleccionadas de un producto. Sea LC una lista de características, entonces sel,* $\overline{sel} \subseteq LC$, $sel \cap \overline{sel} = \emptyset$, *y* $sel \cup \overline{sel} = LC$. *Si p es un producto, los términos p.sel y* $p.\overline{sel}$ *son el conjunto de características seleccionadas y no seleccionadas del producto p, respectivamente.*

Definición 3. *Un conjunto de características cc es válido en un modelo de características* fm, *es decir,* $valido(cc, fm)$ *es verdadero, si y solo si cc no contradice ninguna restricción introducida por* fm.

Las pruebas de interacción combinatorias son un enfoque que construye conjuntos de pruebas (*test suites*) que nos llevan a probar de forma sistemática todas las configuraciones de un sistema [10]. En este trabajo vamos a utilizar el enfoque *pairwise*, que es el más utilizado en pruebas combinatorias y está basado en la suposición de que la mayoría de errores originados en un parámetro es causado por la interacción de dos valores [11]. Este criterio se satisface si para cada par de características f_1 y f_2, podemos encontrar cuatro productos en el conjunto generado que contengan las cuatro posibles combinaciones de presencia/ausencia de dichas características: $(f_1, f_2), (f_1, \overline{f_2}), (\overline{f_1}, f_2)$ y $(\overline{f_1}, \overline{f_2})$. Cuando se aplica esta técnica a FMs, la idea es generar un conjunto de productos válidos, donde los errores posibles se manifiestan con alta probabilidad, sin tener que probar de forma exhaustiva todas las posibles configuraciones. Además, gracias a la priorización, vamos a testar en primer lugar los productos con aquellas características que son más frecuentes o importantes en nuestros sistemas.

Definición 4. *Un producto priorizado pp es un par* (cc, w), *donde cc representa un conjunto de características válido en un modelo e características* fm *y* $w \in \mathbb{R}$ *representa su peso. Sean* pp_i *y* pp_j *dos productos priorizados. Decimos que* pp_i *tiene prioridad más alta que* pp_j *cuando el peso de* pp_i *es más alto que el peso de* pp_j, *es decir,* $pp_i.w > pp_j.w$.

Definición 5. *Una configuración por pares pc es un par* $(sel, \overline{sel})$ *que representa un producto parcialmente configurado, definidos por la selección de dos características de LC, es decir,* $pc.sel \cup pc.\overline{sel} \subseteq LC$, $pc.sel \cap pc.\overline{sel} = \emptyset$ *y* $|pc.sel \cup pc.\overline{sel}| = 2$. *Decimos que pc se cubre con un conjunto de características cc cuando* $pc.sel \subseteq cc.sel$ *y* $pc.\overline{sel} \subseteq cc.\overline{sel}$.

Definición 6. *Una configuración por pares ponderada wpc es un par* (pc, w) *donde pc es una configuración por pares y* $w \in \mathbb{R}$ *representa su peso calculado de la siguiente manera. Sea* PP *un conjunto de productos priorizados y* $PP_{pc} \subseteq PP$, *tal que* PP_{pc} *contiene todos los productos priorizados de* PP *que cubren wpc.pc, es decir,* $PP_{pc} = \{pp \in PP | pp.cc \text{ cubre } wpc.pc\}$. *Entonces* $w = \sum_{p \in PP_{pc}} p.w$.

Dada una colección de conjuntos de características $ppCA$ y un conjunto de configuraciones por pares ponderadas WPC, definimos la cobertura de $ppCA$, denotada por $cob(ppCA)$, como la suma de los pesos de las configuraciones por pares ponderadas de WPC cubiertas por alguna configuración de $ppCA$ dividida entre la suma de todos los pesos de las configuraciones en WPC, esto es:

$$cob(ppCA) = \frac{\sum_{\substack{wpc \in WPC \\ \exists cc \in ppCA, cc \text{ cubre } wpc.pc}} wpc.w}{\sum_{wpc \in WPC} wpc.w}. \quad (1)$$

El problema de optimización en el que estamos interesados consiste en encontrar una colección de conjuntos de características válidos, $ppCA$, que minimice el número de conjuntos de características $|ppCA|$ y maximice la cobertura $cob(ppCA)$. Este es un problema de optimización bi-objetivo con objetivos contrapuestos. Por tanto, la solución no será única, nuestro objetivo será encontrar un conjunto de soluciones eficientes (no dominadas).

4. Algoritmo Ávido basado en Programación Lineal Entera

Para resolver el problema proponemos usar un algoritmo ávido que, en cada iteración, busca un producto que maximice la cobertura con respecto a la actual. Una vez encontrado, añade dicho producto al conjunto solución, elimina los pares de características cubiertos por el producto y continúa su búsqueda de un nuevo producto. El algoritmo se detiene cuando no es posible aumentar la cobertura, lo cual sucede cuando todos los pares de características con peso mayor que cero han sido cubiertos.

Antes de presentar el algoritmo, describiremos el programa lineal entero que se utiliza como base en cada iteración de APLE.

Sea f el número de características de nuestro modelo fm. Usaremos las variables de decisión $x_j \in \{0, 1\}$ con $j \in \{1, 2, \ldots, f\}$ para indicar si debemos incluir la característica j en el siguiente producto ($x_j = 1$ o no ($x_j = 0$). No todas las combinaciones de características forman productos válidos. Siguiendo a Benavides et al. [9], podemos utilizar una fórmula de lógica proposicional para expresar la validez de un producto en un determinado FM. Para poder incluir

esas restricciones en nuestro programa lineal, esta fórmula se expresará en forma normal conjuntiva (CNF) y se transformarán en desigualdades que se añaden al programa lineal.

A continuación veremos cómo se transforma cada cláusula de la fórmula. Definamos los vectores binarios v y u como sigue:

$$v_j = \begin{cases} 1 \text{ si la característica } j \text{ aparece en la cláusula,} \\ 0 \text{ en otro caso,} \end{cases}$$

$$u_j = \begin{cases} 1 \text{ si la característica } j \text{ aparece negada en la cláusula,} \\ 0 \text{ en otro caso.} \end{cases}$$

Con la ayuda de u y v podemos escribir transformar la cláusula en la siguiente desigualdad para nuestro programa lineal:

$$\sum_{j=1}^{f} v_j(u_j(1-x_j) + (1-u_j)x_j) \geq 1. \tag{2}$$

Por otro lado, necesitaremos variables de decisión para modelar las configuraciones por pares que se cubren con un producto. Las variables las denotaremos con $c_{j,k}$, $c_{j,\overline{k}}$, $c_{\overline{j},k}$ o $c_{\overline{j},\overline{k}}$, dependiendo de la combinación de presencia/ausencia de características en la configuración, y tomarán valor 1 si el producto cubre la configuración y 0 en caso contrario. Los valores de las variables c dependen de los valores de las variables x. Para reflejar esta dependencia en nuestro programa lineal, necesitamos añadir las siguientes restricciones para todos los pares de características $1 \leq j < k \leq f$:

$$2c_{\overline{j},\overline{k}} \leq (1-x_j) + (1-x_k), \tag{3}$$

$$2c_{\overline{j},k} \leq (1-x_j) + x_k, \tag{4}$$

$$2c_{j,\overline{k}} \leq x_j + (1-x_k), \tag{5}$$

$$2c_{j,k} \leq x_j + x_k. \tag{6}$$

En realidad no es necesario añadir todas las variables c posibles, sino solo aquellas que se correspondan con una configuración que no haya sido cubierta anteriormente. Llamemos al conjunto de configuraciones no cubiertas U. Entonces, la función objetivo (a maximizar) de nuestro programa lineal, que es la cobertura conseguida con el producto, tiene como expresión:

$$f(c) = \sum_{(j,k)\in U} w_{j,k}c_{j,k}, \tag{7}$$

donde, abusando de notación, usado j y k para representar características del modelo y su presencia/ausencia, y hemos expresado el peso de las configuración (j,k) con $w_{j,k}$.

En el Algoritmo 1 presentamos nuestra propuesta APLE. En la línea 1 inicializa la lista de productos *ppCA*, que por el momento está vacía. A continuación entra en un bucle en el que busca el producto que maximiza la cobertura con

Algoritmo 1. Algoritmo Ávido basado en Programación Lineal Entera (APLE)

Entrada: U //conjunto de configuraciones con peso mayor que cero
Salida: $ppCA$ // lista de productos
1: $ppCA \leftarrow []$
2: **while** $U \neq \emptyset$ **do**
3: $z \leftarrow$ resolver (mín $f(x)$ sujeto a (2)-(6))
4: $ppCA \leftarrow ppCA + z$
5: $U \leftarrow U/cob(z)$ // Elimina las configuraciones cubiertas por z
6: **end while**

respecto a las configuraciones que quedan por cubrir, U (línea 2). El nuevo producto es añadido a la lista de productos y las configuraciones cubiertas por el producto son eliminadas de U.

5. Evaluación

Esta sección describe como fue llevada a cabo nuestra evaluación. Empezamos describiendo los algoritmos PPGS y pICPL objeto de la comparación, seguido de los métodos usados para asignar prioridades, los modelos de características usados como instancias y la configuración de los experimentos.

5.1. Algoritmo PPGS

El algoritmo llamado *Prioritized Pairwise Genetic Solver* (PPGS) es un algoritmo genético constructivo que sigue un modelo maestro-esclavo para paralelizar la evaluación de los individuos. En cada iteración, el algoritmo añade un nuevo producto al conjunto de pruebas en construcción hasta que todas las combinaciones de características se hayan cubierto. Este algoritmo considera como mejor producto para ser añadido aquel que cubra un conjunto de pares de características (aún por cubrir) que aporten mayor cobertura al conjunto de productos priorizados.

La configuración de parámetros utilizada para PPGS es la siguiente: torneo binario como operador de selección, cruce de un punto con probabilidad $0,8$, mutación con probabilidad $0,1$, población de 10 individuos y condición de parada 1000 evaluaciones de fitness para la generación del siguiente mejor producto. La condición de parada del algoritmo es conseguir cobertura total. Para más detalles consultar [6].

5.2. Algoritmo pICPL

pICPL es un algoritmo ávido que genera matrices de cobertura con fortaleza n (*n-wise covering arrays*) desarrollado por Johansen et al. [3]. Este algoritmo no genera covering arrays con cobertura total sino que cubre sólo aquellas combinaciones que aparecen en al menos un producto priorizado. El resultado obtenido

con este enfoque es equivalente al que se persigue resolviendo este problema, ya que el conjunto de pares por cubrir va a ser el mismo en todos los algoritmos utilizados en este artículo. Debemos destacar que pICPL usa ejecución paralela a nivel de datos. Este paralelismo viene de las operaciones realizadas sobre un conjunto amplio de datos. Para más detalles consultar [3]. Queremos remarcar que existe una versión muy conocida de este algoritmo para testar SPLs, desarrollado por los mismos autores, llamado ICPL [12]. Sin embargo, esa versión no contempla prioridades en la computación del conjunto de pruebas.

5.3. Métodos de Asignación de Prioridades

Consideraremos tres métodos de asignación de pesos a productos:

Valores Medidos Los pesos están derivados de propiedades no funcionales obtenidas de 16 sistemas SPL reales. Estos modelos pertenecen a dominios de problema diferentes, estando implementados usando diferentes tecnologías, y fueron medidos utilizando SPL Conqueror [13]. El resultado son estimaciones reales de propiedades medibles no funcionales como consumo de memoria o rendimiento. Se calculan mediciones para un conjunto de productos, usualmente un subconjunto de aquellos que denota el FM. Esta opción de asignación de pesos nos permite emular escenarios de pruebas más realistas donde los ingenieros de pruebas deben realizar mayor esfuerzo en los productos que demuestran más rendimiento, por ejemplo. Para nuestro trabajo tomamos los valores reales de los productos considerando las interacciones de pares de características. La Tabla 1 resume los sistemas SPL evaluados, su propiedad medida (`Prop`), el número de características o *features* (`NF`), el número de productos (`NP`), número de configuraciones medidas (`NC`), y el porcentaje de productos priorizados (`PP` %) usados en nuestra comparativa, como explicaremos en breve.

Valores Basados en Rango Para esta asignación de pesos seleccionamos los productos a priorizar basados en cómo de diferentes son comparados con otros productos de la SPL y se les asigna un peso basado en su rango de valores. La intuición bajo esta estrategia de asignación es dar un peso similar a dos productos muy diferentes. De esta manera los pesos altos están esparcidos entre un número alto de pares de características haciendo que el conjunto de pruebas priorizado sea más difícil de computar. Además, esto nos da la posibilidad de seleccionar diferentes porcentajes de productos usados en el cálculo final de los pesos de las características, como explicaremos a continuación.

Valores Aleatorios Esta asignación de pesos se consigue generando valores aleatorios dentro del rango obtenido en el enfoque anteriormente explicado (Valores Basados en Rango).

Vamos a seleccionar los productos para priorizar basándonos en los métodos de asignación. Para el método de *Valores Medidos*, todos los productos medidos

SPL Name	Prop	NF	NP	NC	PP %
Prevayler	F	6	32	24	75.0
LinkedList	F	26	1440	204	14.1
ZipMe	F	8	64	64	100.0
PKJab	F	12	72	72	100.0
SensorNetwork	F	27	16704	3240	19.4
BerkeleyDBF	F	9	256	256	100.0
Violet	F	101	≈ 1E20	101	≈ 0.0
Linux subset	F	25	≈ 3E24	100	≈ 0.0
LLVM	M	12	1024	53	5.1
Curl	M	14	1024	68	6.6
x264	M	17	2048	77	3.7
Wget	M	17	8192	94	1.15
BerkeleyDBM	M	19	3840	1280	33.3
SQLite	M	40	≈ 5E7	418	≈ 0.0
BerkeleyDBP	P	27	1440	180	12.50
Apache	P	10	256	192	75.0

Tabla 1. Resumen de los Valores Medidos para los Casos de Estudio

fueron usados como productos priorizados. Para los métodos *Basados en Rango* y *Aleatorio*, sólo un porcentaje de los productos denotados por cada FM fue usado como producto priorizado. Los porcentajes usados fueron: 5 %, 10 %, 20 %, 30 % y 50 %.

5.4. Casos de Estudio

Vamos a usar tres grupos de instancias basados en el número de productos denotados por el FM y como fueron asignadas las prioridades, como se muestra en la Tabla 2. El grupo G1 está compuesto por 160 FMs, cuyo número de productos oscila entre 16 y 1000 productos, y fueron evaluados usando los métodos *Basados en Rango* y *Aleatorio*. El grupo G2 está compuesto por 59 FMs con rango de productos entre 1000 y 80000 productos, y fueron evaluados usando estos mismos dos métodos de asignación. El grupo G3 está formado por 16 FMs reales, con un número de productos entre 16 y ≈3E24, que fueron evaluados usando el método de valores medidos. En este estudio analizamos un total de 235 FMs que fueron extraídos de varias fuentes: SPL Conqueror, Johansen et al. [3], y el repositorio SPLOT [8]. Para G1 y G2 las instancias son calculadas usando para cada FM dos métodos de asignación de prioridades a los productos y tres porcentajes diferentes de selección de productos priorizados. Mientras que en G3, sólo usamos el método de asignación de valores medidos. Esto hace un total de 1330 instancias analizadas para cada uno de los tres algoritmos de la comparación.

	G1	G2	G3	Resumen
NFM	160	59	16	235
NP	16-1K	1K-80K	32- ≈3E24	16- ≈3E24
NC	10-56	14-67	6-101	6-101
MA	R,A	R, A	M	
PP %	20,30,50	5,10,20	≈0.0 – 100	
IP	960	354	16	1330

NFM: Número de FMs, **NP**: Numero de Productos, **NC**: Numero de Características, **MA**: Método Asignación, **R**: Rango, **A**: Aleatorio, **M**: Medido, **PP** %: Porcentaje de Productos Priorizados, **IP**: Instancias del Problema

Tabla 2. Resumen de los Casos de Estudio

5.5. Configuración de los Experimentos

PPGS y pICPL son algoritmos no deterministas, por tanto hemos ejecutado 30 veces estos algoritmos para realizar una comparación justa entre ellos. Por otra parte, nuestro enfoque APLE es un algoritmo determinista y fue ejecutado una sola vez por cada instancia. Como medida de rendimiento de los algoritmos hemos analizado el número de productos requerido para alcanzar un porcentaje de cobertura ponderada dada y el tiempo de computación requerido. En ambos casos queremos minimizar los objetivos, ya que queremos conjuntos de prueba más pequeños obtenidos en el menor tiempo posible.

Las ejecuciones de PPGS y pICPL se ejecutaron en un clúster con máquinas Intel Core2 Quad processors Q9400 de 4 núcleos por procesador, por tanto su nivel de paralelismo es de 4 hebras para cada ejecución independiente. APLE fue ejecutado en un sólo núcleo ya que no es un algoritmo paralelo, esto debe ser teniendo en cuenta en la interpretación de los tiempos de ejecución obtenidos.

6. Análisis de los Resultados

En este trabajo usamos dos técnicas estadísticas para medir diferentes aspectos de la comparativa.

6.1. Test de Kruskal-Wallis

Para comprobar si las diferencias entre los algoritmos son estadísticamente significativas hemos aplicado el test no paramétrico Kruskal-Wallis usando la corrección de Bonferroni. En las tablas destacamos con un gris oscuro que un valor es estadísticamente significativo con respecto a los otros dos algoritmos y con gris claro cuando existen diferencias con sólo uno. El nivel de confianza utilizado es el habitual 95 % (p-value menor que $0,05$).

En la Tabla 3 resumimos los resultados obtenidos para el grupo 1 de FMs con hasta 1000 productos. Cada columna corresponde a un algoritmo y en las filas mostramos el número de productos necesarios para alcanzar desde 50 % a 100 %

de cobertura ponderada. Los datos de cada celda son la media y la desviación típica de todas las ejecuciones de las instancias de G1. Podemos observar que los resultados de APLE son estadísticamente diferentes para todos los valores de cobertura y tiempo de ejecución con respecto a al menos uno de los otros algoritmos. Concretamente en 7 ocasiones es estadísticamente diferente con respecto a los otros dos algoritmos. Aparentemente los resultados son estadísticamente mejores cuando las diferencias existen, hecho que confirmaremos en la siguiente subsección aplicando otro test estadístico. Queremos destacar también la gran ventaja con respecto al tiempo de computación de PPGS.

Cobertura	APLE	PPGS	pICPL	Cobertura	APLE	PPGS	pICPL
50 %	$1,19_{0,39}$	$1,20_{0,40}$	$1,20_{0,40}$	96 %	$3,98_{1,21}$	$4,00_{1,23}$	$4,37_{1,42}$
75 %	$1,91_{0,51}$	$1,92_{0,51}$	$1,98_{0,58}$	97 %	$4,35_{1,31}$	$4,38_{1,32}$	$4,71_{1,54}$
80 %	$2,13_{0,58}$	$2,15_{0,59}$	$2,25_{0,68}$	98 %	$4,80_{1,42}$	$4,83_{1,46}$	$5,18_{1,74}$
85 %	$2,44_{0,71}$	$2,47_{0,72}$	$2,58_{0,81}$	99 %	$5,53_{1,64}$	$5,58_{1,71}$	$5,87_{1,99}$
90 %	$2,86_{0,85}$	$2,88_{0,86}$	$3,13_{1,03}$	100 %	$7,49_{2,79}$	$7,56_{2,85}$	$7,56_{3,03}$
95 %	$3,70_{1,13}$	$3,72_{1,14}$	$4,06_{1,33}$	Tiempo	1562_{2026}	23897_{28669}	10116_{18842}

Tabla 3. Media y desviación estándar para las instancias del grupo G1. El tiempo está medido en milisegundos.

La Tabla 4 muestra los resultados para el grupo G2 de FMs de entre 1000 y 80000 productos. En este caso las diferencias significativas siempre existen entre APLE y PPGS con respecto a pICPL. En otras palabras, ambos algoritmos son mejores que pICPL en la calidad del conjunto de pruebas generado. Nuevamente APLE es significativamente mejor que los otros dos algoritmos en 7 ocasiones.

Con respecto al tiempo de computación, APLE es más rápido que los otros algoritmos con una amplia diferencia. En este caso su media de tiempo empleado es de 5 segundos, frente a los alrededor de 6 minutos para pICPL y los más de 10 minutos empleados por PPGS.

Cobertura	APLE	PPGS	pICPL	Cobertura	APLE	PPGS	pICPL
50 %	$1,16_{0,37}$	$1,16_{0,36}$	$1,36_{0,83}$	96 %	$4,96_{0,97}$	$4,98_{0,97}$	$5,83_{3,14}$
75 %	$2,08_{0,41}$	$2,09_{0,42}$	$2,47_{1,65}$	97 %	$5,50_{1,08}$	$5,55_{1,10}$	$6,43_{3,27}$
80 %	$2,36_{0,54}$	$2,39_{0,52}$	$2,86_{1,79}$	98 %	$6,27_{1,26}$	$6,34_{1,34}$	$7,23_{3,48}$
85 %	$2,73_{0,62}$	$2,73_{0,59}$	$3,27_{2,08}$	99 %	$7,48_{1,74}$	$7,66_{1,88}$	$8,59_{4,11}$
90 %	$3,32_{0,80}$	$3,36_{0,76}$	$3,98_{2,38}$	100 %	$13,89_{9,85}$	$14,57_{10,65}$	$13,79_{9,98}$
95 %	$4,55_{0,92}$	$4,59_{0,90}$	$5,42_{3,12}$	Tiempo	$4900_{6,7E+3}$	$273728_{7,2E+5}$	$638164_{2,1E+6}$

Tabla 4. Media y desviación estándar para las instancias del grupo G2. El tiempo está medido en milisegundos.

En la Tabla 5 se muestran resultados del grupo de instancias G3, donde las diferencias significativas que existen son algo menores, pero las conclusiones son similares a las obtenidas con los anteriores grupos de instancias, en la mayoría de los casos existen diferencias entre APLE y pICPL. Con respecto a PPGS, los resultados en calidad de la solución esta vez son similares, pero el tiempo de computación sigue siendo mucho menor para APLE.

Cobertura	APLE	PPGS	pICPL	Cobertura	APLE	PPGS	pICPL
50 %	$1,56_{0,51}$	$1,58_{0,49}$	$1,56_{0,50}$	96 %	$5,69_{1,19}$	$5,86_{1,18}$	$6,38_{1,58}$
75 %	$2,63_{0,81}$	$2,66_{0,77}$	$2,75_{0,75}$	97 %	$6,13_{1,26}$	$6,24_{1,38}$	$6,75_{1,39}$
80 %	$2,81_{0,83}$	$2,81_{0,73}$	$3,25_{0,97}$	98 %	$6,81_{1,47}$	$6,98_{1,55}$	$7,44_{1,66}$
85 %	$3,44_{0,89}$	$3,46_{0,87}$	$3,81_{0,95}$	99 %	$7,75_{1,69}$	$7,92_{1,87}$	$8,75_{2,08}$
90 %	$4,06_{1,06}$	$4,12_{1,04}$	$4,56_{1,27}$	100 %	$11,69_{5,69}$	$12,08_{6,50}$	$12,19_{5,68}$
95 %	$5,38_{1,09}$	$5,45_{1,14}$	$6,06_{1,44}$	Tiempo	$18262_{1,0E+4}$	$2048719_{7,6E+6}$	$24330_{5,9E+4}$

Tabla 5. Media y desviación estándar para las instancias del grupo G3. El tiempo está medido en milisegundos.

Como conclusión general de este primer análisis podemos decir que APLE nunca es estadísticamente peor que ninguno de los otros algoritmos de la comparativa, siendo claramente mejor en muchos de los casos estudiados y en tiempo de computación. Para los grupos de instancias estudiados siempre es aconsejable el uso de APLE para la computación de conjuntos de pruebas priorizadas.

6.2. Estadístico $\hat{A}_{12}$

La interpretación de los test estadísticos puede ser engañosa en alguno de los casos, por eso es siempre aconsejable reportar una medida del tamaño del efecto. Para este propósito hemos usado el estadístico $\hat{A}_{12}$ como recomiendan Arcuri y Briand [14]. Dada una medida de rendimiento M, $\hat{A}_{12}$ mide la probabilidad de que un algoritmo A consiga valores más altos para M que otro algoritmo B. Si los dos algoritmos son equivalentes, entonces $\hat{A}_{12} = 0,5$. Si $\hat{A}_{12} = 0,3$ entonces vamos a obtener valores más pequeños en el 70 % de las ocasiones con el algoritmo A.

A continuación mostramos los valores del estadístico $\hat{A}_{12}$ para medir la significancia práctica de los resultados. Las tablas comparan los resultados de los algoritmos dos a dos para los tres grupos de instancias. En estas tablas un valor menor a $0,5$ indica mejores resultados para el primer algoritmo de la comparación, siendo las tablas las comparaciones de APLE-pICPL y APLE-PPGS.

En la Tabla 6 podemos observar como en ninguna ocasión pICPL tiene una probabilidad mayor que APLE de obtener un mejor conjunto de pruebas. En el peor caso, para el grupo G3 y 50 % de cobertura tenemos un empate en probabilidad. En 32 combinaciones de grupo y cobertura es más probable que APLE consiga una mejor solución, siendo el porcentaje más favorable un 79,7 %. Este estadístico no deja duda de que las diferencias significativas son en favor de APLE.

Grupo	50 %	75 %	80 %	85 %	90 %	95 %	96 %	97 %	98 %	99 %	100 %
G1	0,4953	0,4693	0,4380	0,4370	0,3672	0,3417	0,3286	0,3526	0,3526	0,3865	0,4870
G2	0,4492	0,4096	0,3588	0,3573	0,3121	0,2641	0,2669	0,2387	0,2288	0,2133	0,3446
G3	0,5000	0,4375	0,3125	0,3438	0,3125	0,2812	0,3125	0,2812	0,3438	0,2812	0,3750

Tabla 6. Resultado del estadístico $\hat{A}_{12}$ entre APLE-pICPL para todos los grupos

La Tabla 7 muestra nuevamente que la probabilidad de obtener conjuntos de pruebas mínimos es mayor para APLE que para PPGS. Tan sólo en dos ocasiones la probabilidad es mayor para PPGS, son las celdas G2-50 % y G3-80 %, aunque las diferencias son muy escasas, 50,26 % y 50,42 % respectivamente. En el resto de las 31 combinaciones de grupo y cobertura las probabilidades son favorables a APLE. Sin duda, esta prueba estadística también confirma que las diferencias significativas existentes entre APLE y PPGS son favorables a APLE.

Grupo	50 %	75 %	80 %	85 %	90 %	95 %	96 %	97 %	98 %	99 %	100 %
G1	0,4970	0,4967	0,4881	0,4912	0,4921	0,4936	0,4924	0,4889	0,4918	0,4910	0,4810
G2	0,5026	0,4943	0,4868	0,4936	0,4750	0,4745	0,4832	0,4738	0,4653	0,4288	0,3693
G3	0,4938	0,4854	0,5042	0,4917	0,4750	0,4708	0,4188	0,4479	0,4354	0,4417	0,4146

Tabla 7. Resultado del estadístico $\hat{A}_{12}$ entre APLE-PPGS para todos los grupos

7. Amenazas a la Validez

Hemos identificado dos amenazas a la validez de este trabajo. La primera es que la selección de los modelos de características que forman parte de nuestro experimento puede afectar a los resultados obtenidos. Para combatir esta amenaza hemos usado tres fuentes diferentes para obtener un gran número de FMs, con un amplio rango tanto de número de características como de número de productos que denotan. La segunda es la selección de los métodos de asignación de prioridades. Para aliviar este problema hemos utilizado tres métodos diferentes de asignación de prioridades y diferentes porcentajes de productos tenidos en cuenta. Además, para el grupo G3 estas prioridades están calculadas sobre propiedades no funcionales del propio código fuente, por lo que la distribución de pesos utilizada para ponderar las características es realista.

8. Conclusiones y Trabajo Futuro

En este artículo hemos presentado un enfoque novedoso que hibrida un algoritmo ávido con un resolutor ILP. Nuestra propuesta algorítmica se ha evaluado con 235 FMs con diferentes características y usando diferentes criterios de selección para la ponderación de los productos. Hemos comparado APLE con PPGS y pICPL, siendo APLE siempre superior manifiestamente en rendimiento, tanto en tiempo de ejecución como en calidad de la solución, especialmente frente a pICPL. Por tanto, nuestra recomendación es usar APLE para el cálculo de conjunto de pruebas priorizadas para SPLs, ya que consigue mantener la tasa de detección de fallos usando menos productos de prueba.

Nuestra propuesta, APLE, no asegura que vaya a obtener el frente de Pareto del problema bi-objetivo que pretendemos resolver, ya que en cada iteración solo decide cuál será el siguiente producto a añadir. Para obtener el frente de Pareto sería necesario buscar un conjunto de productos en cada iteración. Algunos resultados preliminares que hemos obtenido en esta línea sugieren que el

número de variables y restricciones de las instancias ILP necesarias para resolver el problema crecen muy rápido con el número de productos, y esto repercute negativamente en el tiempo de requerido por el resolutor ILP para resolver el problema. Sería interesante analizar este incremento del número de variables para buscar alternativas para obtener el frente de Pareto en un tiempo razonable.

Acknowledgements

Esta investigación ha sido parcialmente financiada por el Ministerio de Economía y Competitividad y fondos FEDER (proyecto TIN2014-57341-R), la beca BES-2012-055967, la Universidad de Málaga y Andalucía Tech.

Referencias

1. Pohl, K., Bockle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer (2005)
2. Cichos, H., Oster, S., Lochau, M., Schürr, A.: Model-based coverage-driven test suite generation for software product lines. In: MoDELS. (2011) 425–439
3. Johansen, M.F., Haugen, Ø., Fleurey, F., Eldegard, A.G., Syversen, T.: Generating better partial covering arrays by modeling weights on sub-product lines. In France, R.B., Kazmeier, J., Breu, R., Atkinson, C., eds.: MoDELS. Volume 7590 of Lecture Notes in Computer Science., Springer (2012) 269–284
4. Engström, E., Runeson, P.: Software product line testing - a systematic mapping study. Information & Software Technology **53**(1) (2011) 2–13
5. da Mota Silveira Neto, P.A., do Carmo Machado, I., McGregor, J.D., de Almeida, E.S., de Lemos Meira, S.R.: A systematic mapping study of software product lines testing. Information & Software Technology **53**(5) (2011) 407–423
6. Lopez-Herrejon, R.E., Javier Ferrer, J., Chicano, F., Haslinger, E.N., Egyed, A., Alba, E.: A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. GECCO '14, New York, NY, USA, ACM (2014) 1255–1262
7. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)
8. Mendonca, M.: Software Product Line Online Tools(SPLOT) (2013) http://www.splot-research.org/.
9. Benavides, D., Segura, S., Cortés, A.R.: Automated analysis of feature models 20 years later: A literature review. Inf. Syst. **35**(6) (2010) 615–636
10. Cohen, M.B., Dwyer, M.B., Shi, J.: Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. IEEE Trans. Software Eng. **34**(5) (2008) 633–650
11. Stevens, B., Mendelsohn, E.: Efficient software testing protocols. In: Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research. CASCON '98, IBM Press (1998) 22–
12. Johansen, M.F., Haugen, Ø., Fleurey, F.: An algorithm for generating t-wise covering arrays from large feature models. In: SPLC (1). (2012) 46–55
13. Siegmund, N., Rosenmüller, M., Kästner, C., Giarrusso, P., Apel, S., Kolesnikov, S.: Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption. Information & Soft. Technology **55**(3) (2013) 491–507
14. Arcuri, A., Briand, L.: A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. Softw. Test. Verif. Reliab (2012)