TESIS DOCTORAL

2019

CODIFICACIÓN EFICIENTE DE MODELOS DE CONFIGURACIÓN UTILIZANDO BDD'S

ROBERTO BEÁN CASTELLÓ

PROGRAMA DE DOCTORADO EN INGENIERÍA DE SISTEMAS Y CONTROL

Director: Rubén Heradio Gil
Codirector: David Fernández Amorós

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

CODIFICACIÓN EFICIENTE DE MODELOS DE CONFIGURACIÓN UTILIZANDO BDD'S

Tesis doctoral presentada por Roberto Beán

dentro del Programa de Doctorado en Ingeniería de Sistemas y Control

Dirigida por Dr. Rubén Heradio

y Dr. David Fernández-Amorós

El doctorando El director El director

Madrid, marzo de 2019

Codificación eficiente de modelos de configuración utilizando BDD's

Autor: Roberto Beán Castelló
Director: Dr. Rubén Heradio Gil
Co-Director: Dr. David Fernández Amorós

Texto impreso en Madrid
Primera edición, marzo 2019

Para Lydia por su paciencia y apoyo incondicional

Resumen

La industria y los consumidores necesitan productos personalizados, que se adapten a ellos. La ingeniería de líneas de productos software aborda este problema mediante la creación de productos extremadamente configurables. Estos, se definen mediante modelos de configuración, posibilitando su análisis, mediante su transformación a lógica Booleana. Este análisis permite, por ejemplo, identificar errores del modelo antes de su producción. El análisis de lógica Booleana, a través de SAT *solvers* o BDD's, es un problema no soluble en tiempo polinomial. La representación de un producto con tan sólo 37 opciones, utilizando BDD's requiere en el peor de los casos, $2 \cdot 10^{11}$ nodos tantos, como galaxias hay en el universo conocido. En esta tesis se proponen soluciones que permiten la representaciones del modelo completo más escalable, y cuando esto no es posible, se ofrece: (i) dar soluciones a problemas parciales, y (ii) simplificar el problema completo resolviendo primero problemas parciales.

Reconocimientos

Rubén y David, gracias por vuestro apoyo y paciencia durante el desarrollo de mi trabajo. Con vosotros he aprendido y me habéis acompañado, en el largo camino que ha culminado con esta tesis.

Gracias

Roberto Beán Castelló

Índice general

Índice de figuras

Índice de tablas

Símbolos, Abreviaturas y Siglas

ANOVA	Analysis of VAriance
BDD	Binary Decision Diagram
CNF	Conjuntive Normal Form
CTA	Closed Timed Automata
DPLL	Davis-Putnam-Logemann-Loveland
DSL	Domain Specific Language
FSM	Finite State Machine
MCL	Markov Cluster Algorithm
NP	Nondeterministic Polynomial time
OBDD	Ordered Binary Decision Diagram
PSG	Possible Symmetric Group

ROBDD	Reduced Ordered Binary Decision Diagram
SAT	Problema de satisfacibilidad
SPL	Software Product Line
VLSI	Very Large Scale Integration Circuits

El principio es la parte más importe del trabajo.

Platón

CAPÍTULO

1

Introducción

Esta tesis facilita el razonamiento automático sobre modelos de configuración, los cuales son esenciales para la especificación de sistemas altamente configurables. La tecnología que subyace a dicho razonamiento son los Binary Decision Diagrams (BDD's). Precisamente, el presente capítulo introduce todos estos conceptos: sistemas configurables, modelos de configuración, su importancia, y necesidad de análisis. Posteriormente, se introducirán las herramientas de análisis, y se presentarán las ventajas de los BDD's con respecto a otras herramientas, y el objetivo de nuestra investigación.

1.1 Sistemas configurables

La línea de producción en masa introducida por Henry Ford en el siglo pasado estandarizó los productos y redujo los costes de producción. Pero, hoy en día, para incrementar la variedad, mejorar la satisfacción del cliente, reducir los tiempos de entrega y costes, las compañías han cambiado este paradigma a las líneas de producción customizada [113].

Los clientes desean su producto o sistema configurable adaptado a sus propias necesidades, y no un producto único. El Ford T sólo estaba disponible en un único color, el negro, tal y como indica la frase atribuida a Henry Ford:

> "Cualquier color siempre y cuando sea negro."

Hoy en día, en la compra de un coche configuramos habitualmente, como mínimo, el motor, color y nivel de acabado. Por ejemplo, para el Toyota Auris de 2017, supone 3.696 variaciones. El modelo más vendido de Ford, el Focus, tenía en 2004 hasta 366.901.933 variaciones, o la Serie 3 de BMW, $6,4 \cdot 10^{16}$[108]. Los ordenadores también son sistemas configurables, la configuración de portátiles DELL en 2011 tenía $7,3 \cdot 10^{17}$ variaciones [102].

El cambio de modelo de producción no sólo se aplica a ámbitos de fabricación, sino, también al desarrollo de software, dos ejemplos de ello son los sistemas operativos eCos y Linux, con 6320 y 1256 opciones configurables respectivamente [15]. Para hacerse una idea de la enorme variabilidad que esto supone, téngase en cuenta que un pequeño modelo con tan sólo 260 opciones configurables representa más variaciones que el número de átomos en el universo observable[52].

Tal y como indican Thüm et al., es imperativo tener métodos de análisis para la fiabilidad y éxito de estos sistemas [132]. Por ejemplo, el 31,3 % de los errores de configuración son debidos a parámetros incorrectos en el sistema que no son detectados [141]. Para el análisis de los sistemas configurables se utilizan modelos de configuración que los codifican.

1.2 Modelos de configuración

1.2.1 Definición

Un modelo de configuración (M_c) codifica los productos de una Línea de Productos o SPL (Software Product Line). Estos modelos se representan con diagramas de características [80] o Lenguajes Específicos de Dominio DSL (Domain Specific Language), como, por ejemplo, los lenguajes CDL y KConfig utilizados para codificar las opciones de configuración de los sistemas operativos eCos y Linux, respectivamente.

El análisis de diagramas de características se realiza habitualmente en la literatura mediante su transformación a lógica Booleana [61],[11], [94], [123], [118],[119], [120], [39],[12],[63], [123], [68], [109], [106], [116] y [67].

Los modelos de configuración se definen mediante una lista de opciones o características de configuración (*fearures*) $F = \{f_1, f_2, \ldots, f_n\}$, y una serie de restricciones (*constraints*) $C = \{c_1, c_2 \ldots, c_m\}$.

Un modelo de configuración ($M_c = (F, C)$) puede ser transformado a una función Booleana (ψ) mediante una función de transformación ($f_t : M_c \rightarrow \psi$).

En esta tesis utilizaremos los diagramas de características para codificar los modelos de configuración que transformaremos a lógica Booleana para su análisis A continuación, describiremos brevemente los diagramas de características y como transformarlos.

1.2.2 Diagramas de características

Los diagramas de características introducidos por Kang et al. en [80], modelan todas las posibles configuraciones de una SPL, en términos de opciones de configuración del sistema y de sus relaciones [12].

La Figura 1.1 representa un modelo de configuración de un automóvil, con restricciones jerárquicas como las ruedas, y no jerárquicas o *crosstree*, como la relación existente entre el color de la carrocería y el color del tapizado.

Existen diferentes tipos de relaciones entre las opciones jerárquicas de los diagramas, cada una de ellas con su correspondencia con lógica Booleana. Por claridad, en los ejemplos de esta tesis nos centraremos en las relaciones que Benavides

et al. proponen como básicas [12]. Esto no supone una pérdida de generalidad en absoluto, puesto que cualquier otro tipo de relaciones, como las identificadas por Schobbens et al. [118], pueden codificarse fácilmente en lógica Booleana, y las fórmulas Booleanas son precisamente el *input* de nuestro mecanismo de razonamiento automático.

Obligatorias(Mandatory): La opción hija es obligatoria al seleccionar el padre. Por ejemplo, para todos los automóviles es obligatorio tener ruedas.

Opcionales(Optional): La característica hija es opcional al seleccionar el padre. Por ejemplo, añadir lunas tintadas al coche.

Alternativa(Alternative): Sólo se debe seleccionar una de las opciones hijas, si el padre se ha seleccionado. Por ejemplo, el color de la carrocería, al seleccionar un determinado modelo de automóvil.

O:(Or) Se puede seleccionar una o más de una opción hija si el padre se ha seleccionado. Cómo, por ejemplo, las opciones de adicionales al configurar un automóvil.

Además de las relaciones jerárquicas, las restricciones *crosstree*, habitualmente se representan como fórmulas Booleanas que añaden restricciones adicionales al modelo. Cómo, por ejemplo, al seleccionar el color rojo del coche no podemos elegir las ruedas azules. Utilizando las *crosstree* se puede representar cualquier modelo de configuración.

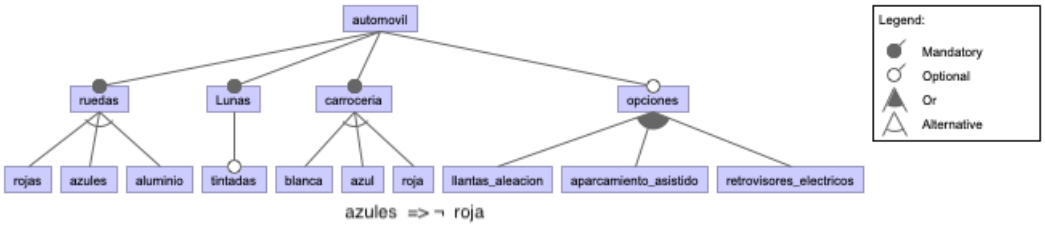


Figura 1.1: Diagrama de características

1.2.3 Definición formal de diagramas de características

Una característica $f = (n, p, h, r)$ es una tupla donde:

- n es el nombre de la característica. Por ejemplo, `ruedas`.
- p es el nombre de la característica padre. En el caso de la raíz su valor es $\emptyset$.
- $h = \{f_1, f_2, \ldots, f_m\}$ es una lista ordenada de características hijas.
- $r \in \{\textit{Mandatory}, \textit{Optional}, \textit{Or}, \textit{Alternative}\}$ es el tipo relación.

Un diagrama de características $dc = (root, features, ct)$ es una tupla donde:

- $root$ es el nombre de la característica raíz. Por ejemplo, `automovil`.
- $features = \{f_1, f_2, \ldots, f_n\}$ es una lista de características.
- $ct = \{ct_1, ct_2, \ldots, ct_n\}$ es una lista de restricciones *crosstree*.

1.2.4 Conversión a lógica Booleana

Los diagramas de características tienen una conversión directa a lógica Booleana para cada una de las relaciones definidas en la Sección 1.2.2, tal y como se muestra en la Tabla 1.1. Es también necesario que la característica raíz sea cierta. Por ejemplo, en el caso del automóvil la opción $automovil$ debe ser cierta.

Por ejemplo, la transformación del diagrama de características de la Figura 1.1 se corresponde con la Tabla 1.2, a la que se debería añadir la restricción *crosstree* $azules \implies \neg roja$.

1.2.5 Conclusiones

Los diagramas de características permiten codificar cualquier modelo de configuración, y son transformables a lógica Booleana. Utilizando esta transformación, podemos analizarlos automáticamente como se verá a continuación.

Relación	**Fórmula Booleana**	**#cláusulas**
Raíz $root$	$root$	1
p Optional h	$\neg h \vee p$	1
p Mandatory h	$(\neg h \vee p) \wedge (\neg p \vee h)$	2
p Or $(h_1, h_2, \ldots, h_n)$	$(\neg p \vee (h_1 \vee h_2 \vee \ldots \vee h_n))$	1
	$\wedge(\neg h_1 \vee p)$	2
	$\wedge(\neg h_2 \vee p\ )$	3
	$\ldots$	$\ldots$
	$\wedge(\neg h_n \vee p)$	$n+1$
p Alternative $(h_1, h_2, \ldots, h_n)$	$(\neg p \vee (h_1 \vee h_2 \vee \ldots \vee h_n))$	1
	$\wedge(\neg h_1 \vee p)$	2
	$\wedge(\neg h_2 \vee p\ )$	3
	$\ldots$	$\ldots$
	$\wedge(\neg h_n \vee p)$	$n+1$
	$\wedge((\neg h_1 \vee \neg h_2) \wedge (\neg h_1 \vee \neg h_3) \wedge \ldots$	$\ldots$
	$\wedge(\neg h_1 \vee \neg h_n) \wedge \ldots \wedge (\neg h_{n-1} \vee \neg h_n))$	$(n+1) + C_2(n)$

Tabla 1.1: Conversión de diagramas de características a lógica Booleana

Relación	**Fórmula Booleana**
Root $automovil$	$automovil$
$automovil$ Mandatory $ruedas$	$(automovil \vee \neg ruedas) \wedge (\neg automovil \vee ruedas)$
$automovil$ Mandatory $lunas$	$(automovil \vee \neg lunas) \wedge (\neg automovil \vee lunas)$
$automovil$ Mandatory $color$	$(automovil \vee \neg color) \wedge (\neg automovil \vee color)$
$automovil$ Optional $opciones$	$(\neg opciones \vee automovil)$
$ruedas$ Alternative $(rojas, azules, alumnio)$	$(\neg ruedas \vee (rojas \vee azules \vee aluminio))$
	$\wedge((\neg rojas \vee \neg azules) \wedge (\neg rojas \vee \neg alumnio)$
	$\wedge(\neg azules \vee \neg alumnio))$
	$\wedge(\neg rojas \vee ruedas)$
	$\wedge(\neg azules \vee ruedas)$
	$\wedge(\neg alumnio \vee ruedas)$
$carroceria$ Alternative $(blanca, azul, roja)$	$(\neg carroceria \vee (blanca \vee azul \vee roja))$
	$\wedge((\neg blanca \vee \neg azul) \wedge (\neg blanca \vee \neg roja)$
	$\wedge(\neg azul \vee \neg roja))$
	$\wedge(\neg blanca \vee carroceria)$
	$\wedge(\neg azul \vee carroceria)$
	$\wedge(\neg roja \vee carroceria)$
$opciones$ Or $(aleacion, asistido, electronicos)$	$(\neg opciones \vee (aleacion \vee asistido \vee electronicos))$
	$\wedge(\neg aleacion \vee opciones)$
	$\wedge(\neg asistido \vee opciones)$
	$\wedge(\neg electronicos \vee opciones)$
crosstree	$\neg azul \vee \neg roja$

Tabla 1.2: Fórmula Booleana del diagrama de características de la Figura 1.1

1.3 Análisis de modelos de configuración

1.3.1 Introducción

El análisis de los modelos de configuración se realiza a través de lógica Booleana, como se ha indicado anteriormente. Este análisis suele realizarse utilizando dos tecnologías los SAT *solvers* o los BDD's. A continuación, describiremos brevemente en que se basa cada una de las ellas, y sus ventajas e inconvenientes.

1.3.2 SAT solvers

Un SAT *solver*, es un programa informático, que recibe como entrada una Función Booleana $f(x_1, x_2, \ldots, x_n)$, y devuelve cierto, si la Función Booleana es satisfacible. Es decir,si existe una asignación $(a_1, a_2, \ldots, a_n)$ que la hace cierta.

$$f_{x_1=a_1, x_2=a_2, \ldots, x_n=a_n}(x_1, x_2, \ldots, x_n)$$

Los SAT *solvers* utilizan variantes del algoritmo DPLL (Davis-Putnam-Logemann-Loveland) [41] (Algoritmo 1). Este algoritmo precisa que las fórmulas Booleanas estén en formato CNF (Conjuntive Normal Form), es decir una fórmula Booleana $f(x_1, x_2, \ldots, x_n)$ se expresará como:

$$f = C_1 \wedge C_2 \wedge C_k$$

donde

$$C_i = a_{i1} \vee a_{i2} \vee \cdots \vee a_{il}$$

y, $a_{ij} \in \{x_j, \neg x_j\} | 1 \leq j \leq n$.

El Algoritmo 1 describe en pseudocódigo el funcionamiento de la búsqueda recursiva DPLL. Primero, se comprueba si ψ contiene una única variable, en cuyo caso el problema es satisfacible. Si el conjunto de cláusulas obtenidas es vacío, el resultado es falso (línea 4). Las líneas 8 a 10 buscan todas las tautologías en la fórmula, es decir aquellas cláusulas con un único literal y las eliminan de la fórmula. Las líneas 11 a 13 eliminan aquellos literales que sólo estén asignados como valor cierto o falso en las conjunciones. Finalmente se entra en el bucle recursivo seleccionado un literal y asignándole valor cierto o falso.

Algoritmo 1: Algoritmo DPLL

```
1  Function DPLL ( ψ = {C_1, C_2, ..., C_n} )
2      if ψ contiene una única variable then
3          return cierto;
4      end
5      if ψ = ∅ then
6          return falso;
7      end
8      foreach l ∈ tautologias(ψ) do
9          ψ = eliminar_tautologias(ψ, l);
10     end
11     foreach l ∈ literales_puros(ψ) do
12         ψ = eliminar_literales_puros(ψ, l);
13     end
14     l = seleccionar_un_literal(ψ);
15     return DPLL(ψ ∧ l) ∨ DPLL(ψ ∧ ¬l);
```

El algoritmo estándar finaliza cuando obtiene una asignación correcta, pero también puede ser extendido para obtener todas las asignaciones no finalizando el recorrido recursivo hasta que no se obtengan todas las soluciones. Es decir, modificando la línea 15 para que ejecuten ambas ramas de la asignación de l y no sólo la primera que tenga una asignación a cierto. Este algoritmo básico puede modificarse para construir un #SAT-*solver* capaz de enumerar todas las asignaciones satisfacibles de la fórmula Booleana.

Los algoritmos DPLL son eficientes para verificar la satisfacibilidad del problema, sin embargo, no son eficientes para la obtención de todas las soluciones, ya que en estos casos básicamente recorren todo el espacio de búsqueda aplicando sólo las reducciones de tautologías y literales puros.

Los SAT *solvers* funcionan únicamente si la fórmula está expresada en CNF. Pero eso no siempre es posible en los modelos de configuración y la transformación de una fórmula Booleana a CNF es un problema NP (Nondeterministic Polynomial time). Por ejemplo, la transformación de la fórmula Booleana $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$ con n disyunciones implica una fórmula con 2^n conjunciones tal y como muestra la Ecuación 1.1.

$$\begin{array}{c}(x_1 \vee x_2 \vee \ldots \vee x_n) \\ \wedge(y_1 \vee x_2 \vee \ldots \vee x_n) \\ \wedge(y_1 \vee y_2 \vee \ldots \vee x_n) \\ \vdots \\ \wedge(y_1 \vee y_2 \vee \ldots \vee y_n)\end{array} \tag{1.1}$$

1.3.3 BDD's

Los BDD's codifican eficientemente una fórmula Booleana, aplicando diversas reducciones para minimizar el espacio requerido en memoria. Los BDD's son grafos, y se construyen asignando valores cierto y falso, a cada una de las variables durante la construcción, de modo eficiente, de forma similar al Algoritmo 1.

Como se verá en el Capítulo 2, en el que se mostrarán las operaciones con modelos de configuración, un BDD puede dar respuesta a varias preguntas, y puede ser reutilizado. Es decir, se puede codificar una fórmula Booleana en el BDD A y combinarlo con el B para obtener el BDD $A \wedge B$.

Los BDD's no están restringidos a la codificación de fórmulas Booleanas en formato CNF. Lamentablemente, esto no siempre evita que la solución a problemas complejos tenga costes exponenciales, al igual que los SAT *solvers* en el peor de los casos, ya que recorren todo el espacio de búsqueda.

1.3.4 Conclusiones

Para soportar el razonamiento automático sobre modelos de configuración complejos se utilizan dos tipos de herramientas: los SAT *solvers* y los BDD's. La idea básica es codificar los modelos con lógica Booleana y posteriormente utilizar dichas herramientas para analizar la fórmula resultante.

Como se verá en esta tesis, los BDD's son más genéricos que los SAT *solvers*, en el sentido de que permiten realizar eficientemente más tipos de análisis. Esto se debe a que la tecnología SAT está orientada específicamente a encontrar una única asignación satisfacible para una fórmula lógica, mientras que los BDD's codifican la tabla de verdad completa de una fórmula lógica, y "recorriendo esa tabla" pueden realizarse muchos tipos de razonamientos.

Veremos que ambas herramientas presentan limitaciones de escalabilidad debido a que enfrentan problemas NP. Uno de los objetivos de esta tesis es precisamente minimizar dichas limitaciones.

1.4 Objetivo de la investigación

El objeto de nuestra investigación es la codificación de modelos de configuración utilizando BDD's de modo eficiente, reduciendo su tamaño, y permitiendo así, realizar el análisis de modelos grandes.

Primero, se realizará una investigación de las heurísticas de codificación existentes de modelos de configuración utilizando BDD's, y un análisis sistemático de dichas heurísticas. Posteriormente, se presentarán las heurísticas y técnicas de descomposición desarrolladas, junto con sus experimentos y resultados obtenidos. Finalmente se presentarán las conclusiones.

Personalmente, yo pienso que ayuda, que la convierte en una diferencia beneficiosa, pero la literatura científica sobre un tema es muy desordenada.

Jeanne Petrek

CAPÍTULO

2

Estado del arte

Los BDD's pueden considerarse una versión comprimida de la tabla de verdad de una fórmula Booleana. Para que esta compresión sea efectiva es fundamental encontrar una buena ordenación de variables y restricciones. Lamentablemente, la búsqueda del orden óptimo es un problema NP y, por ello, todas las aproximaciones existentes son de tipo heurístico.

En este capítulo se definirán formalmente los BDD's, posteriormente se describirán, clasificarán y analizarán las heurísticas de ordenación de BDD's existentes y finalmente se presentarán las operaciones sobre modelos de configuración que pueden realizarse utilizando BDD's.

2.1 BDD's

2.1.1 Definiciones

Definición 2.1.1. *Sea $m, n \in \mathbb{N}$, la siguiente transformación es denominada función Booleana.*

$$f : \mathbf{B}^n \rightarrow \mathbf{B}^m$$

En esta tesis denotamos las funciones Booleanas como

$$f(x_1, x_2, \ldots, x_n)$$

donde f es la función Booleana y $\{x_1, \ldots, x_n\}$ son las variables Booleanas de entrada.

$$f_{x_i=b} = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n\}$$

Denota una función Booleana con un valor $b \in \{0, 1\}$ asignado a la variable x_i, donde $1 \leq i \leq n$.

Definición 2.1.2. *Un BDD es un grafo acíclico con raíz, que contiene nodos de decisión y terminales [25]. Existen dos tipos de nodos terminales llamados 0-terminal y 1-terminal, etiquetados con 0 y 1 respectivamente. Cada nodo de decisión v está etiquetado con una variable Booleana x_i, para una función Booleana f, y tiene dos nodos hijos denominados $else(x_i)$ and $then(x_i)$, que usualmente, se visualizan con líneas discontinúas, y sólidas respectivamente. La arista desde el nodo x_i al nodo $else(x_i)$ ($then(x_i)$) es la asignación $f_{x_i=0}$ ($f_{x_i=1}$).*

La definición 2.1.2 es consecuencia de la descomposición de Shannon, y permite definir cualquier función Booleana f mediante la siguiente ecuación:

$$f = (x \wedge f_{x=1}) \vee (\neg x \wedge f_{x=0}) \tag{2.1}$$

Definición 2.1.3. *Un BDD es* ordenado *(OBDD (Ordered Binary Decision Diagram)) si dos variables diferentes aparecen siempre en el mismo orden ($\pi = \{a_1, a_2, \ldots, a_n\}$) en todos los caminos desde la raíz hasta los nodos terminales. En consecuencia, un OBDD ($B = (f, \pi)$) es definido por una función Booleana f y un orden π.*

Definición 2.1.4. *El nivel de una variable x_i ($l_{f,\pi}(x_i)$) en un OBDD $B = (f, \pi)$ es la posición que ocupa la variable variable x_i en el orden π.*

Definición 2.1.5. *La función $l^{-1}_{f,\pi} : \{1, \ldots, n\} \rightarrow \{x_1, \ldots, x_n\}$ devuelve la variable x_k correspondiente al nivel k.*

Definición 2.1.6. *Un OBDD es reducido (ROBDD) si las siguientes reglas son aplicadas al grafo [72]:*

R1 Eliminación de terminales duplicados. *Si un OBDD contiene más de un nodo 0-terminal, todas las aristas que apuntan a dicho nodo son reducidas a uno de ellos. Esta reducción también se aplica a nodos redundantes 1-terminal.*

R2 Eliminación de test redundantes. *Si todas las aristas de salida de un nodo interno v_i, apuntan al mismo nodo hijo v_j. El nodo padre v_i se elimina, y sus aristas entrantes se apuntan al nodo hijo v_j.*

R3 Eliminación de nodos no terminales duplicados. *Si dos nodos distintos v_i y v_j del ROBDD son raíces de subROBDD's estructuralmente idénticos, entonces se elimina uno de ellos, y se redireccionan todas aristas de entrada al otro nodo.*

Como se muestra en la Figura 2.1 el orden (π) condiciona el tamaño del ROBDD. Los dos BDD's definen la misma función Booleana $f = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ con dos diferentes órdenes $\pi_1 = \{x_1, x_2, x_3, x_4\}$, y $\pi_2 = \{x_1, x_3, x_2, x_4\}$, tienen un tamaño con 4 y 6 nodos respectivamente.

Definición 2.1.7. *Si $x_v \in \{x_1, x_2, \ldots, x_n\}$ y π es un orden sobre $\{x_1, x_2, \ldots, x_n\}$ entonces $cost_{x_v}(f, \pi)$ denota el número de nodos etiquetados x_v en el ROBDD.*

De acuerdo con la Definición 2.1.7, la minimización del tamaño del ROBDD es equivalente a minimizar la siguiente ecuación:

$$\sum_{i=1}^{n} cost_{x_i}(f, \pi) \tag{2.2}$$

Definición 2.1.8. *Definimos $\Pi(Q)$ como todas posibles permutaciones de un subconjunto variables $Q = \{x_1, x_2, \ldots, x_q\}$, con $q \leq n$.*

Por conveniencia, en esta tesis nos referiremos a los ROBDD's como BDD's de aquí en adelante.

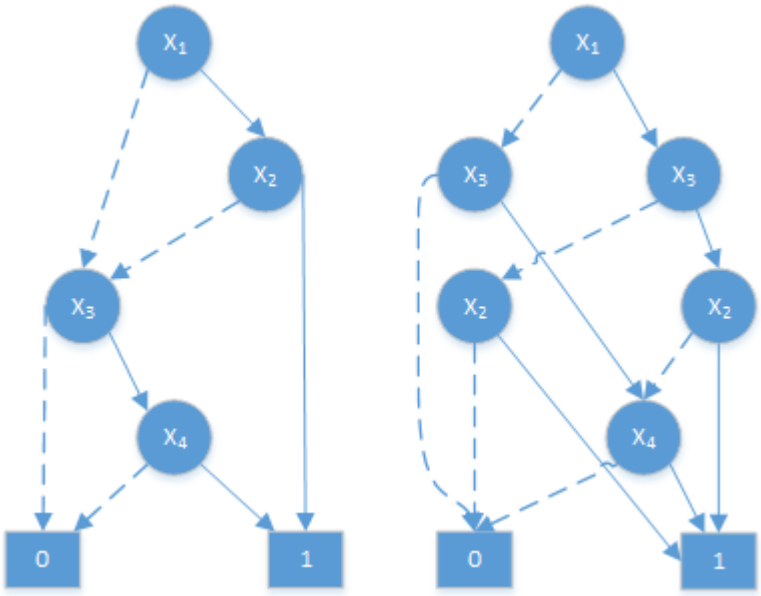


Figura 2.1: ROBDDs para la función $f = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$

2.1.2 Representación

Siguiendo las indicaciones dadas por Bryant [25], representaremos un BDD que tiene m nodos y que codifica una fórmula Booleana con n variables utilizando las siguientes estructuras.

- El orden de las variables usado para sintetizar el BDD es representado por un *array* declarado como: **Transcripción 1**

```
var_ordering: array [0..n-1] of string
```

- Cada nodo es representado como un registro (*record*) declarado como sigue: **Transcripción 2**

```
type node = record
index: 0..n
then,else : node
mark: boolean
value: (true,false)
end
```

Donde

1. `index` es el índice de las variables en el orden. Los nodos terminales del BDD (i.e., 0 y 1) tienen índice n.
2. `then` y `else` son los sucesores *true* y *false*.
3. `mark` se utiliza para marcar los nodos que son visitados en el recorrido de un camino.
4. `value` se utiliza en los nodos terminales, el nodo terminal 1 tiene valor cierto (*true*)y el 0 valor falso (*false*).

- El BDD es representado por un *array* declarado como sigue: **Transcripción 3**

```
bdd: array [0..m] of node
```

2.2 Heurísticas de ordenación de BDDs

2.2.1 Clasificación

La literatura diferencia las heurísticas ordenación de variables de los BDD's, según el momento en que se aplican: Las heurísticas estáticas, establecen un orden fijo $\pi = \{x_1, x_2, \ldots, x_n\}$ de las variables, antes su construcción; y las dinámicas, parten de un orden inicial, con el cual se construye el BDD, y una vez construido total o parcialmente, se reordenan las variables con un nuevo orden π', que minimizando su tamaño.

Las dos técnicas pueden ser complementarias, podemos utilizar las primeras antes de construir el BDD, y las segundas una vez construido total o parcialmente.

2.2.2 Heurísticas analizadas

En esta tesis se he realizado un análisis bibliográfico sistemático sobre heurísticas de ordenación de BDD's combinando en la cadena de búsqueda los diferentes sinónimos de BDD de la Tabla 2.1 junto con los sinónimos de ordenación de variables de la Tabla 2.2. La búsqueda se ha realizado en ISI Web of Science [112] y Scopus [50], ambos resultados se combinaron en una única base de datos [1] utilizando la herramienta Scimat [31].

Con los resultados obtenidos se realizó una revisión por expertos para descartar artículos no relevantes, y se seleccionaron los artículos más citados en todos los tiempos generando una base de datos [2] con criterios de clasificación de acuerdo con cuando se produce la heurística (estática y dinámica), y como se realiza.

A continuación, se enumerarán las heurísticas estáticas, y posteriormente las dinámicas, ambas en orden cronológico. Para cada una de las heurísticas se incluirá una descripción de su funcionamiento y, se introducirá el contexto en donde se utiliza, y se discutirá su idoneidad para sintetizar BDD's sobre modelos de configuración. .

[1] https://github.com/robcbean/literature_review_tesis/blob/master/scimat_roberto
[2] https://github.com/robcbean/literature_review_tesis/blob/master/literature_review.accdb

Sinónimo BDD
Binary decision diagram
BDD
OBDD ROBDD
BDDs
OBDDs
ROBDDs
Zero-supressed BDD
ZDD
ZDDs
Branch program

Tabla 2.1: Sinónimos de BDD

Sinónimos ordenación de variables
variable order
variable-order
constraint order
minimization
optimization

Tabla 2.2: Sinónimos de ordenación de variables

2.2.3 Heurísticas estáticas

2.2.3.1 Topología

La primera aproximación es utilizar la topología del problema, incluyendo la descomposición para inferir el orden de las variables.

El primer uso de los BDD's, fue la optimización y análisis de VLSI (Very Large Scale Integration Circuits), los investigadores inicialmente utilizaban un orden manual para cada problema, pero esto impedía realizar un análisis automático. Fujita et al. [57] propusieron una heurística basada en la topología de los circuitos, recorriéndolos en postorden, empezando por la puerta de salida del circuito. En el caso de que una variable sólo esté incluida en una única puerta lógica, o función, se añade la variable al final del orden, en caso contrario, se seleccionan primero aquellas conectadas a más de una puerta lógica el resto se añaden posteriormente. Utilizando esta heurística pudieron procesar los circuitos del ISCAS benchmark [95], no procesables utilizando ordenes aleatorios.

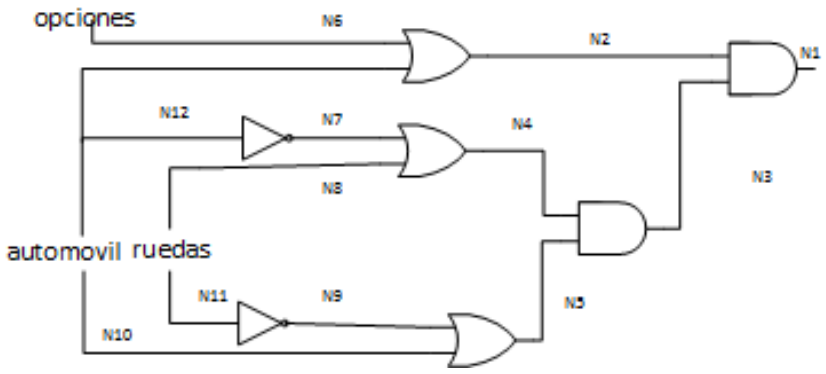


Figura 2.2: Ejemplo circuito diagrama características

El Algoritmo 2 parte de una puerta de salida de un circuito lógico $N1$ en el ejemplo de la Figura 2.2, posteriormente recorre todas las puertas conectadas a esta (línea 2), en este caso $N2$ y $N3$, verifica si cada uno de las puertas lógicas está conectado a una entrada del circuito o entrada primaria (línea 4), si no es así desciende por el circuito en postorden aplicando el algoritmo recursivamente (línea 14), hasta llegar a una puerta lógica conectada a una entrada primaria, por ejemplo, $N6$. Si la puerta $N6$ no está conectada a más entradas la añade a la lista elementos de salida $fanoutlist$ (línea 11), es decir la almacena para recorrerla

posteriormente. Si está conectada a una entrada la añade al orden existente (líneas 6 a 9).

Finalmente, añadiremos todas las entradas primarias no conectadas a ninguna puerta después de los nodos conectados a más de una puerta (línea 18 a 24) y marcaremos el nodo como visitado.

Algoritmo 2: Heurística Fujita et al. [57]

```
1  Action Fujita_et_al( N : Nodo de salida, visted : lista, order : lista ordenada )
2      foreach I ∈ puertas_conectadas(N) do
3          if I ∉ visited then
4              if conectado_entrada_primaria(I) then
5                  if conectado_a_mas_de_una_puerta(I) then
6                      fanout2up = I;
7                      if I ∉ order then
8                          order.push(I);
9                      end
10                 else
11                     fanoutlist.push(I);
12                 end
13             else
14                 Fujita_et_al(I, visited, order);
15             end
16         end
17     end
18     if fanout2up ≠ null then
19         index = buscar_elemento(fanout2up);
20         foreach variable ∈ fanoutlist do
21             order.push_at(index, variable);
22             index = index + 1;
23         end
24     end
25     visted = visited ∪ I;
26     return;
```

La heurística Fujita et al. realiza un recorrido en postorden basado en las puertas de salida, este criterio es equivalente a la utilización del preorden en un diagrama de características, seleccionando primero aquellas variables que estén incluidas en otras funciones, en el caso de los diagramas de características aquellas que forman parte de una estructura jerárquica.

Malik et al. extendió la heurística de Fujita et al., ordenando las variables de acuerdo con el número de puertas lógicas (*fanin*), en la que está incluida cada variable o el nivel en que se encuentra dentro del circuito, en ambos casos con orden decreciente.

Con la inclusión del número de puertas lógicas en el que se encuentra una variable se añade más información acerca del peso de dichas variables en la función y en consecuencia mejora el Algoritmo Fujita et al. [57], en la mayoría de los casos como se verá posteriormente.

Butler et al. introduce la propagación del peso de las variables. Es decir, añadir información de relevancia de las variables dentro del problema, para ordenar las variables en una misma puerta [27], realizando un recorrido en postorden, al igual que los algoritmos anteriores.

La dificultad de la heurística Butler et al. es identificar la función de peso adecuada de las variables. Es decir, es específica para cada problema, en cambio las heurísticas anteriores pueden utilizarse directamente para la codificación de diagramas de características .

Descomposición de problemas

Cuando el problema puede ser modularizado mediante varias funciones Booleanas sobre el conjunto de variables $\{x_1, x_2, \ldots, x_n\}$:

$$M_c = \{f_1(x_{1_1}, x_{1_2}, \ldots, x_{1_{j1}}), f_2(x_{2_1}, x_{2_2}, \ldots, x_{2_{j2}}), \ldots, f_k(x_{k_1}, x_{k_2}, \ldots, x_{k_{jk}})\}$$

donde las variables de cada función son disjuntas, es decir:

$$\{x_{1_1}, x_{1_2}, \ldots, x_{1_{j1}}\} \cap \{x_{2_1}, x_{2_2}, \ldots, x_{1_{j2}}\} \cap \ldots \cap \{x_{k_1}, x_{k_2}, \ldots, x_{k_{j1}}\} = \emptyset$$

En estos casos, es posible aplicar algoritmos individuales a cada una de las funciones, y posteriormente combinarlas para construir el BDD.

Malik et al. [88] aplicaron está técnica para circuitos sin variables compartidas, y mostró que optimizar el problema de cada uno de los subproblemas es equivalente a optimizar el problema globalmente.

Para combinar funciones múltiples, o módulos, con variables compartidas algunos autores utilizan el concepto de intercalación (*interleaving*). Por ejemplo, si tenemos dos funciones (o módulos) $\{f_1, f_2, \ldots, f_n\}$ sobre el conjunto de variables $\{x_1, \ldots, , x_6\}$, a la que se le asigna los siguientes pesos $\{6, 5, 3, 4, 2, 1\}$, las variables pueden ser añadidas para obtener el orden $\{x_1, x_2, x_3, x_4, x_5, x_6\}$, o intercaladas ordenándolas de acuerdo con su peso $\{x_1, x_2, x_4, x_3, x_5, x_6\}$.

Fujii et al. [56] aplica la técnica de intercalación para problemas de VLSI, Berndl et al. [16] la utilizan también para puntos de análisis de programas, y Wang [139] para autómatas temporales finitos, en modelado y verificación.

El Algoritmo 3 es la implementación de la heurística Fujii et al., el primer paso del algoritmo es recorrer todas las funciones, o lo que es equivalente las salidas de un circuito, ordenando cada función individualmente mediante el Algoritmo 4. Este algoritmo, al igual que los algoritmos anteriores, primero verifica si la puerta o variable que estamos tratando ha sido visitada (línea 2). En caso de haber sido visitada y que en la puerta anterior visitada no sea una puerta de salida (línea 3), guardamos como última puerta visita la puerta actual (línea 4) y de dónde venimos (línea 5). En caso de no haber visitado la puerta se marca como visitada (línea 9) y salvamos de qué función venimos (línea 10). Luego, se aplica el algoritmo recursivamente para cada una de las puertas de entrada enlazadas con la puerta que se trata (líneas 11 a 13). Una vez recorridas todas las puertas en postorden, se genera el orden: en el caso que estemos en la primera variable (línea 14) de la función, añadimos la puerta al orden; en caso contrario buscamos donde está la última puerta que hemos visitado y la añadimos justo a continuación (líneas 17 a 19) realizando la intercalación.

Algoritmo 3: Heurística Fujii et al.

```
1 Function Fujii_et_al( outputs : lista ordenada de salidas de un circuito )
2     foreach o ∈ output do
3         last = null;
4         order_output(o, o, visited, last, from);
5     end
```

Los algoritmos de intercalación son útiles cuando el problema se puede descomponer en múltiples funciones, y permiten aplicar por ejemplo en el caso de la heurística Fujii et al. la combinación del algoritmo de [57] junto con dicha técnica. Sin embargo, su aplicación, sólo es útil cuando el problema puede descomponerse.

Las heurísticas aplicadas a VLSI [57],[88] y [56], pueden aplicarse a diagramas de características cambiando los recorridos en postorden de los circuitos por recorridos en preorden de los diagramas. Por otra parte, las restricciones *crosstree*, pueden codificarse como una función adicional que se combina con el operador *and* con la estructura jerárquica.

Narodytska and Walsh [100] proponen una heurística para problemas de mode-

Algoritmo 4: Ordenación de salida de circuito de la heurística Fujii et al.

```
1  Action order_output(o : salida de un circuito, g : puerta, visited : lista ordenada de variables, last :
     variable, from : mapa)
2      if g ∈ visited then
3          if from[g] ≠ o then
4              last = g ;
5              from[g] = o;
6          end
7          return;
8      end
9      visited.push(g);
10     from[g] = o;
11     foreach f ∈ puertas_entrada(g) do
12         order_output(o, f, visited, last, from);
13     end
14     if last = null then
15         vl.push(g);
16     else
17         index = vl.find_index(last);
18         vl.insert_at(index + 1, g);
19     end
20     last = g;
21     return;
```

los configuración basada en la descomposición del problema. Esta heurística primero identifica las relaciones entre las variables de cláusulas que componen la formula Booleana que representa el modelo de configuración generando un grafo primario de restricciones. Dos variables están conectadas en dicho grafo si y sólo sí existe al menos una cláusula que contiene ambas variables. Se define también el peso de una arista como el número de cláusulas en las que las dos variables están juntas.

El grafo de restricciones se descompone en clústeres utilizando el algoritmo MCL (Markov Cluster Algorithm) [44] de Dongen. Posteriormente, para cada uno de los clústeres, se ordenan las variables de acuerdo con el siguiente algoritmo:

- Para cada variable dentro de un clúster, se computa el peso total de las aristas adyacentes. Nos referiremos a él como el peso de la variable.

- Para las variables dentro del clúster, seleccionar aquella con mayor peso. Esta variable es considerada como la variable central y se sitúa como variable inicial dentro del clúster.

- Las variables dentro del clúster son ordenadas de acuerdo con el peso de las aristas que las conectan a la variable central. Las que no están directamente conectadas se colocan al final del clúster.

- Las variables que no han sido ordenadas en el paso anterior se ordenan de acuerdo a su peso. Las variables más pesadas primero. Las variables con el mismo peso se ordenan de acuerdo con el peso total de sus variables vecinas.

Primero se ordenan aquellos clústeres que están fuertemente conectados con el resto de los clústeres, utilizando como referencia de la conexión la suma del peso de las aristas que conectan los clústeres.

Este algoritmo difiere de los anteriores en que utiliza la información topológica de la fórmula Booleana y no del modelo del problema, eso permite gracias algoritmo de MCL identificar clústeres de características, o variables, que no se observan en el modelo del problema, y en consecuencia puede ser aplicado directamente a la codificación Booleana de los diagramas de características.

El algoritmo MCL utiliza un parámetro que controla la granularidad de la descomposición i.e., lo fino que se descompondrá el problema, en esta tesis, se ha utilizado su valor recomendado 2,5 [44].

Técnicas complementarias

En esta sección, mostraremos algunas técnicas complementarias que se pueden utilizar, las heurísticas basadas en topología.

Estimación del tamaño del BDD Beyer et al. [19] utilizan para sistemas de tiempo real basados en n módulos de CTA (Closed Timed Automata), una ordenación basada en el preorden de los módulos. La heurística ordena los variables vecinas dentro del mismo módulo utilizando una función [18], que estima el tamaño del BDD en tiempo polinómico.

Aplicación de reducciones Jung et al. [78] utilizan la reducción en la codificación de sistemas de análisis de fallos (*fault tree*), con funciones coherentes. Un *fault tree* de un sistema se representa como un circuito, donde cada sub. sistema puede verse como un nodo o función de salida. La Figura 2.3 muestra un *fault tree*

que se define a través de la función $f_{sistema} = ((x_1 \vee x_2) \wedge (x_2 \vee x_3)) \vee x_4$. En la figura los nodos de entrada son representados con letras minúsculas y las puertas lógicas con letras mayúsculas.

La transformación de un *fault tree* a lógica Booleana se realiza habitualmente en postorden como en los circuitos VLSI.

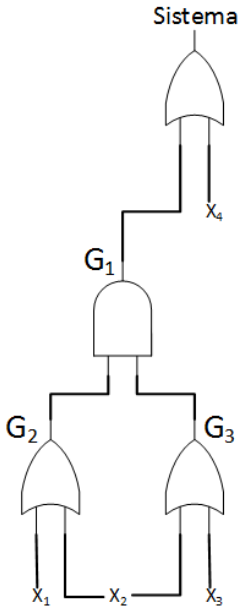


Figura 2.3: *Fault tree*

Una función f es coherente, si su estructura es creciente, las variables son introducidas en orden descendiente, y cada variable es relevante, i.e., para cada variable x de la función $f_{x=1} \neq f_{x=0}$. Es decir, no puede aplicarse la reducción **R2** a dicha variable, ver Definición 2.1.6, ya que si se cumpliera que $f_{x=1} = f_{x=0}$ el nodo del BDD que representaría esa función mostrado en la Figura 2.4

Si una función es coherente, la descomposición de Shannon para sistemas coherentes puede ser simplificada como:

$$f = x \wedge f_{x=1} \vee f_{x=0} \tag{2.3}$$

Utilizando esta descomposición, Jung et al., reduce el tamaño de los BDD's, ya que es posible realizar operaciones de reducción, durante la construcción del BDD, y no al final.

Figura 2.4: No reducible con la regla **R2**

Estas heurísticas sólo pueden aplicarse cuando el problema es coherente, pero los modelos de configuración con características opcionales no son coherentes, puesto que una característica opcional aislada no afecta al resto de opciones de configuración. Por lo que, en general, esta reducción no es útil en modelos de configuración.

Figura 2.5: Diagrama de características de un automóvil simple

Por ejemplo, la Figura 2.5 muestra un modelo de configuración simple que se codifica en lógica Booleana mediante la ecuación 2.4.

$$f_{automovil} = automovil \wedge (\neg climatizador \vee automovil) \wedge \\ \wedge (\neg carrceria \vee automovil) \wedge (carrocerial \vee \neg automovil) \tag{2.4}$$

La fórmula es satisfacible independientemente del valor de la característica climatizador, es decir, el valor de la función resultante es el mismo, y en consecuencia la función que lo representa no es coherente.

2.2.3.2 Span

Las heurísticas anteriores están relacionadas con la topología del problema, las basadas en el *span* se basan en las cláusulas $\{c_1, c_2, \ldots, c_m\}$) de la fórmula Booleana ($f(x_1, x_2, \ldots, x_n)$). Estas heurísticas buscan minimizar el *span* (la distancia entre la mayor y menor variable en las cláusulas respecto al orden π del BDD (f, π)), minimizando el *span* total ($TotSpan$) y el *span* promedio ($AvgSpan$) definidos como:

$$TotSpan = \sum_{i=1}^{m} span(c_i) \tag{2.5}$$

$$AvgSpan = \frac{TotSpan}{m} \tag{2.6}$$

La minimización del *span*, es equivalente a la minimización del *i-cut* o MINCUT, que es un problema NP Completo [60]. En consecuencia, no existe ninguna heurística que lo resuelva en tiempo polinomial. Por tanto, su es complejidad equivalente a la minimización de BDD's.

La minimización del *span*, ha sido utilizada por Aloul et al. en las heurísticas MINCE [3] y FORCE[4], que describiremos a continuación.

MINCE

Una formula CNF $f(x_1, x_2, \ldots, x_n) = c_1 \vee c_2 \ldots \vee c_m$ puede ser vista como un *hypergrafo* $H = (V, E)$, donde el conjunto de vértices $V = \{x_1, x_2, \ldots, x_n\}$, y una arista e_i para cada cláusula $c_i \in \{c_1, c_2, \ldots, c_n\}$ conectando todas las variables dentro de la cláusula [13].

Aloul et al. [2] demuestran que minimizar el *span* promedio para una formula f, y el *hypergraph* $G = (V, E)$, es equivalente a minimizar el *i-cut* ($AvgCut$) promedio (ecuación 2.7).

Definición 2.2.1. *El $i - cut(x_i)$ respecto a una variable x_i, en un orden $\pi = \{x_1, x_2, \ldots, x_n\}$, es el número de cláusulas con variables, menores que, y mayores, que $i + 0{,}5$.*

La Definición 2.2.1, dada una variable x_i, nos indica en cuantas clausulas dicha variable está contenida. En consecuencia, cuantos cortes se producen en el hypergrafo.

Cláusula	***span***
$c_1 = x_1 \vee \neg x_2$	$2 - 1 = 1$
$c_2 = x_2 \vee x_3 \vee \neg x_4$	$4 - 2 = 2$
$c_3 = x_1 \vee x_5$	$5 - 1 = 4$
TotSpan	7
AvgSpan	$2,3333$

Tabla 2.3: Cálculo del *span* para la Función f

Variable	**i-cut**
x_1	2
x_2	3
x_3	2
x_4	2
x_5	0
TotCut	9
AvgCut	$2,25$

Tabla 2.4: i-cut para la Función f

$$AvgCut = \frac{\sum_{i=1}^{n} i - cut(x_i)}{n} \tag{2.7}$$

Las Tabla 2.3 y 2.4, contienen el cálculo del *span* y i-cut, para la función Boleana de la Ecuación 2.8, con el orden $\pi = \{x_1, x_2, x_3, x_4, x_5\}$.

$$f = c_1 \wedge c_2 \wedge c_3 = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_5) \tag{2.8}$$

MINCE utiliza el algoritmo **CAPO** [29] para reducir el i-cut promedio y en consecuencia el *span*. La Figura 2.6 resume esquemáticamente el funcionamiento del algoritmo **MINCE**.

FORCE

La heurística **FORCE** [4] parte de un orden aleatorio π para una fórmula f, e itera sobre este orden inicial asignado un nuevo orden mientras el *span* total disminuya. El algoritmo calcula el centro de gravedad (COG) de cada arista (e) del *hypergrafo* $G = (\{x_1, x_2, \ldots, x_n\}, \{e_1, e_2, \ldots, e_m\})$ que representa la fórmula definido como.

$$COG(e) = \frac{\sum_{x_i \in e} l_{f,\pi}(x_i)}{|e|} \tag{2.9}$$

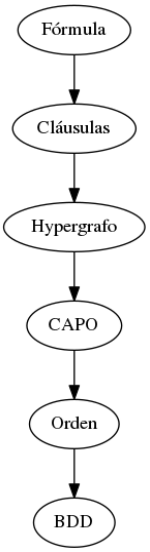


Figura 2.6: Algoritmo MINCE

donde $|e|$ es el número de vértices conectados a la arista e en el *hypergrafo*. La nueva localización de cada variable ($l'_{f,\pi}(x_i)$) se calcula en cada iteración como:

$$l'_{f,\pi}(x_i) = \frac{\sum_{e \in E_{x_i}} COG(e)}{|E_{x_i}|} \tag{2.10}$$

donde E_{x_i} son las aristas conectadas con el vértice x_i. El algoritmo también puede ser forzado a detenerse pasadas un número de iteraciones. La heurística **FORCE** es un 29,39 % más rápida que **MINCE**, en los ejemplos heterogéneos tal y como se muestra en la Tabla 2.5 de los resultados de Aloul [3].

Las heurísticas que utilizan el *span* como **FORCE** pueden ser aplicadas a cualquier fórmula boolena y no dependen del conocimiento del problema, y en consecuencia a la codificación Booleana de diagramas de características.

2.2.3.3 Combinación de heurísticas

En esta sección se muestran heurísticas construidas a partir de la combinación de otras heurísticas, en estos casos se utiliza una herramienta generalista combinada con una heurística diseñada específicamente para trabajar con modelos de configuración

Model	Nº de variables	MINCE	FORCE	%
hole10 [53]	110	82,50	85,50	-3,64
hole11 [53]	132	920,00	738,00	19,78
fpga10_8 [99]	120	47,50	68,90	-45,05
fpga10_9 [99]	135	345,00	5,12	98,52
chnl9_11 [135]	198	19,50	16,00	17,95
chnl9_12 [135]	216	37,00	30,40	17,84
xor1_32 [5]	94	175,00	116,00	33,71
xor1_36 [5]	106	1000,00	229,00	77,10
Urq3_1 [5]	43	664,00	485,00	26,96
Urq3_9 [5]	37	7,30	3,98	45,48
2pipe_1_ooo [137]	834	32,80	6,47	80,27
2pipe_2_ooo [137]	925	34,4	3,54	89,71
2pipe [137]	861	23,4	1,74	92,56
grout3.3-8 [5]	912	33,50	143,70	-328,96
grout3.3-10 [5]	1056	59,20	526,20	-788,85
Total		3481,10	2459,60	29,35

Tabla 2.5: FORCE versus MINCE

Mendonca et al. [94] proponen la combinación de heurísticas que utilizan la topología del problema (preorden y descomposición) combinadas con heurísticas de reducción de *span* (Pre-CL-Span) u ordenación por niveles (Pre-CL-Size) para problemas de modelos de configuración.

Lo primero que hace la heurística de Mendonca et al. es descomponer el problema en sub. problemas, esta descomposición está dirigida por las restricciones *crosstree*. La idea es mantener las características que tienen *crosstree* compartidas juntas. Antes de pasar a describir la descomposición, es necesario introducir las siguientes definiciones de [94].

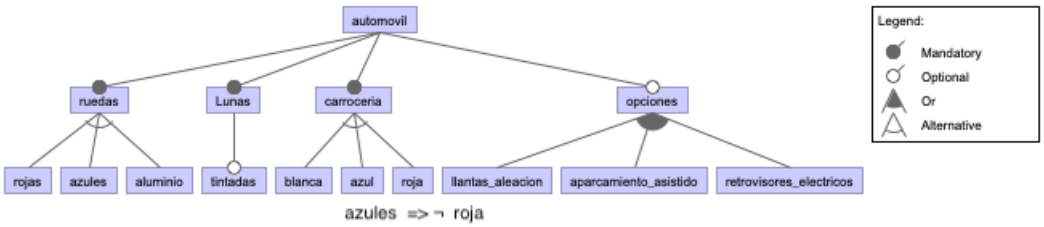


Figura 2.7: Diagrama de características

Definición 2.2.2. *Para un conjunto de características* $f_1, f_2, \ldots, f_n$*, su mínimo común antecesor, escrito como* $LCA(f_1, f_2, \ldots, f_n)$*, es la característica superior*

compartida, con un nivel del diagrama de características mayor. Teniendo en cuenta que el nodo raíz tiene nivel 1.

Definición 2.2.3. *Dada una característica* $f = LCA(f_1, f_2, \ldots, f_n)$*, las raíces de las características* $f_1, f_2, \ldots, f_n$ *escritas como* $Roots(f_1, f_2, \ldots, f_n)$*, es el conjunto* $\{f\}$*, si* f *no tiene hijos, o el sub.conjunto de hijos que son antecesores de* $f_1, f_2, \ldots, f_n$*.*

Para el diagrama de características 2.7:

$$LCA(\texttt{Ruedas azules}, \texttt{Carrocería roja}) = \texttt{Automóvil}$$

y

$$Roots(\texttt{Ruedas azules}, \texttt{Carrocería roja}) = \{\texttt{Ruedas}, \texttt{Color carrocería}\}$$

Algoritmo 5: Process FT-Cluster

```
1  Input F: Diagrama de características
2  Output hypergrafos : Mapa(variable,Hypergrafo)
3  var H: Hypergrafo
4  begin
5      C = crosstree(F);
6      foreach c ∈ C do
7          A = LCA(variables(c));
8          H = hypergrafos[A] ;
9          if EsNulo(H) then
10             H = newHypergrafo();
11             hypergrafos[A] = H;
12             Añadir como vértices los nodos hijos de A al hyper grafo H;
13             foreach node hijo N ∈ A do
14                 Añadir la hyper arista {N} a H;
15             end
16         end
17         R = Roots(variables(c));
18         hypergrafos[A] = Mezclar las hyper aristas de H con los elementos compartidos en R;
19     end
20     return hypergrafos;
21 end
```

El algoritmo de descomposición del problema 5, recorre todas las *crosstree* (línea 5), para cada una de ellas obtiene su LCA y el hyper grafo asociado a dicho

LCA (líneas 6 a 7). Si todavía no se ha construido el hyper grafo, se construye un hyper grafo H con los nodos igual a los hijos del LCA, y con una hyper arista para cada nodo (líneas 9 a 14).

Posteriormente, obtiene las raíces de las variables asociadas a la *crosstree* c y las mezcla con las hyperaristas asociadas del hypergrafo asociado a c (líneas 16 y 17). El resultado final es una serie de hypergrafos, cada uno de ellos representando un clúster.

Una vez descompuesto el problema, se hace un recorrido en preorden partiendo del nodo raíz, dirigido por los clústeres obtenidos por el Algoritmo 6, tal y como se muestra a continuación. Si el nodo que se está procesando no pertenece a un clúster (línea 7), se ordenan sus hijos de acuerdo con el número de descendientes, en orden ascendente (línea 8). Y si no es así, se ordenan las hyper aristas del clúster de acuerdo con el número de descendientes de sus nodos en orden descendiente (línea 10), y los elementos en una hyper arista de acuerdo con el número de descendientes o bien utilizando el algoritmo **FORCE** [4] (heurística generalista del algoritmo combinado).

En este tipo de heurísticas también podemos incluir la de Narodytska and Walsh [100], dado que utilizan la heurística generalista de Dongen combinando con la específica para modelos de configuración.

2.2.4 Heurísticas dinámicas

Fujita et al. [58] identificaron que intercambiar el orden de dos variables adyacentes (x_i y x_{i+1}) en un BDD sólo afecta a los niveles i y $i+1$. El resto permanece inalterado, haciendo posible la alteración de un orden predefinido en un BDD. La Figura 2.8 muestra un ejemplo de intercambio de variables, usando el algoritmo de Fujita et al. [58].

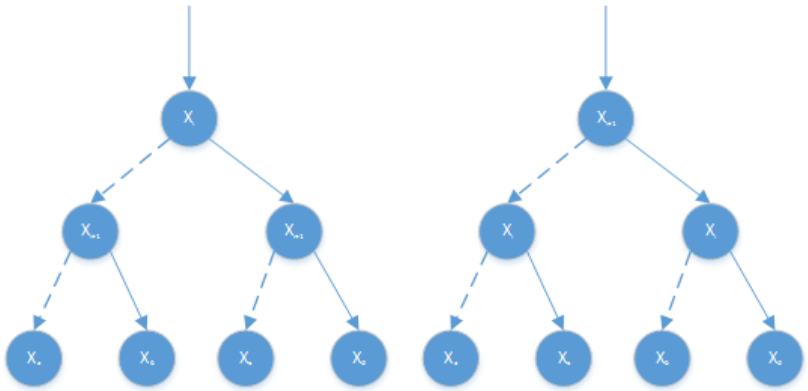
2.2.4.1 Intercambio de variables (*Swapping*)

Ishiura et al. [75] propusieron intercambiar aleatoriamente dos variables adyacentes, comparando el tamaño del BDD antes y después del intercambio, en caso de mejora, se mantiene el nuevo orden, y si no se restablece el orden anterior.

Los autores proponen como mejora utilizar una heurística *greedy* o *simulated ANNEALING*, para seleccionar las variables. Una heurística *greedy* tendría los si-

Algoritmo 6: Heurística Pre-CL-Rec

```
1  Input N : Característica, hypergrafos : Mapa(variable,Hypergrafo), O : lista, S: Estrategia
2  var H: Hypergrafo
3  begin
4      O = O ∪ {N};
5      H = hypergrafos[N];
6      L = {};
7      if EsNulo(H) then
8          L = nodos hijos de N en orden ascendete de acuerdo a el número de descendientes;
9      else
           // El tamaño es la suma de los descientes de sus nodos
10         Ordenar las hyper aristas de H en orden ascendiente;
11         if S= SIZE then
12             Ordenar las elementos de una hyper arista en orden ascendente respecto al número de descendientes;
13         else
14             Utilizar el algoritmo FORCE [4] para ordenar los elementos
15         end
16         foreach E ∈ HyperEdges(H) do
17             L = L ∪ {nodos del diagrama correspondientes a E};
18         end
19     end
20     foreach P ∈ L do
21         Pre-CL-Rec(P,hypergrafos,O,S);
22     end
23     return O;
24 end
```



(a) Orden $i, i+1$ (b) Orden $i+1, i$

Figura 2.8: Intercambio de variables

guientes elementos:

- Una solución candidata: el nuevo BDD con las variables intercambiadas.
- Una función de selección: los posibles ordenes de una variable, seleccionará aquel que minimiza el tamaño del BDD.
- Una función posible: el intercambio de k variables aleatorias.
- Una función objetivo: el cálculo del tamaño del BDD para cada uno de los intercambios de variables.
- Una función de solución, que devuelve valor cierto, si se obtiene la solución, o falso en caso contrario.

La función de solución, debería devolver valor cierto, al encontrar el orden π que construya el BDD con menor número de nodos. Esto, obviamente es equivalente a resolver el problema. En consecuencia, para los BDD's la función de solución devuelve falso mientras el tamaño del BDD disminuya respecto al anterior.

La heurística *simulated ANNEALING* define una función que selecciona el nuevo orden utilizando una función de energía, como la que propone, por ejemplo Bollig et al. [22]. Ishiura et al. [75] no indican cual es la función que se utiliza en sus resultados.

La heurística de Ishiura et al. no sólo puede ser aplicada al intercambio de 2 variables, si no generalizarse, al intercambio de k variables, probando todas las permutaciones posibles de los k elementos seleccionados. Es decir, $k!$. El uso de un valor de k mayor que dos mejora el resultado del número de nodos, pero penaliza en exceso el rendimiento del algoritmo como se demostrará en el Capítulo 3. Por ejemplo, para un valor de $k = 4$ se son necesarias 24 permutaciones por cada selección de tres variables aleatorias.

El algoritmo estándar, intercambia cada una de las variables del BDD, en consecuencia, el coste del algoritmo es $\mathcal{O}(n \cdot k)$, donde n es el número de variables del BDD, y k las variables adyacentes que se intercambian. El algoritmo puede ser implementado también utilizando una versión iterativa que intercambia variables aleatoriamente, mientras se mejora el tamaño del BDD.

2.2.4.2 **Desplazamiento (*Shifting*)**

Rudell [114] propuso una variación del intercambio de variables, su algoritmo sólo mueve una variable arriba y abajo para probar las n posiciones posibles, el resto de las variables permanecen fijas. La Figura 2.9 muestra cómo se desplaza la variable x_3. Si el mejor orden obtenido es el tercero, el algoritmo vuelve atrás desde la última posición a la tercera posición. Aplicando este algoritmo iterativamente, a todos los niveles, almacenando el mejor orden.

inicial	x_1,x_2,x_3,x_4,x_5,x_6	
$order - 1$	x_1,x_2,x_4,x_3,x_5,x_6	intercambiar(x_3,x_4)
$orden - 2$	x_1,x_2,x_4,x_5,x_3,x_6	intercambiar(x_3,x_5)
$orden - 3$	x_1,x_2,x_4,x_5,x_6,x_3	intercambiar(x_3,x_6)
$order - 4$	x_1,x_3,x_2,x_4,x_5,x_6	intercambiar(x_2,x_3)
$order - 5$	x_3,x_1,x_2,x_4,x_5,x_6	intercambiar(x_1,x_3)

Figura 2.9: Ejemplo del algoritmo de *shifting*

La heurística Rudell mejora en un 30 % a la Ishiura et al., como se mostrará en el Capítulo 3, en contraposición, esta heurística tiene un coste mayor, en el peor de los casos es $\mathcal{O}(n^2)$ donde n es el número de variables, la diferencia de coste es $\frac{n}{k}$. Para valores de k cercanos a n el coste de ambos algoritmos se iguala.

Panda et al. [105] observaron que algunas variables necesitan estar cerca y propusieron bloquear estas variables en el algoritmo de *sifting* haciendo que se muevan conjuntamente. Las variables que se muevan juntas deben ser obviamente variables consecutivas en el orden seleccionado, y una variable no puede estar tampoco en más de un grupo.

2.2.4.3 **Simetría**

La Definición 2.2.4 [104] permite identificar cuando dos variables en un BDD son simétricas.

Definición 2.2.4. *Sea $f(x_1, x_2, \ldots, x_n)$ una función Booleana, representada por el BDD B, con el orden $\pi = \{x_1, x_2, \ldots, x_n\}$. Dos variables x_i y x_{i+1} son simétricas en f, si y sólo sí:*

1. *Para todos los nodos de B etiquetados con x_i, se verifica la siguiente condición $g_{x_i \neg x_{i+1}} = g_{\neg x_i x_{i+1}}$. Donde g es la función representada por el nodo etiquetado con x_i. Es decir, la negación la variable x_i en la función, es igual la negación de la variable x_{i+1}.*

2. *Todos los arcos de los nodos etiquetados como x_{i+1} provienen de x_i.*

En un BDD, si dos variables son continuas y simétricas (x_i, x_{1+1}), podemos intercambiar dichas variables sin afectar al BDD. Por tanto, podemos descartarlas en el algoritmo de *sifting*, como proponen Panda et al. [104].

Moller et al. [98] y Panda et al. [105] demostraron que agrupar las variables simétricas reduce el número de nodos en algunas funciones. En consecuencia, bloquear las variables simétricas en el *sifting* en estas funciones reduce el número de nodos.

Para evitar realizar un test de simetría para todos los pares de variables Scholl et al. [121] demostraron los siguientes lemas:

Lema 1. *La función f es asimétrica en $\{x_i, x_j\}$ si $|f_{x_i}| \neq |f_{x_j}|$. Donde $|f_{x_i}|$ y $|f_{x_j}|$ es el número de soluciones de las funciones que representan los nodos etiquetados con x_i y x_j respectivamente.*

Lema 2. *La función f es asimétrica en $\{x_i, x_j\}$, si un nodo en el BDD de f con etiqueta x_i no tiene ningún acceso a un sucesor con etiqueta x_j.*

Lema 3. *La función f es asimétrica en $\{x_i, x_j\}$, si un nodo en BDD de f con etiqueta x_j puede ser accedido desde el nodo raíz sin pasar por ningún nodo etiquetado con x_j.*

Lema 4. *La función f es asimétrica en $\{x_i, x_j\}$ en el BDD de f un nodo v etiquetado con x_i existe un conjunto de los hijos izquierdos del nodo x_j que es diferente del conjunto de hijos derechos del nodo x_j.*

La identificación de partición mínima simétrica en un BDD es un problema NP, como prueban Scholl et al. en [121], transformado el problema de k-coloracion.

Tal y como indicaron Mishchenko [97], los test de simetría requieren recorrer el BDD en múltiples ocasiones, realizando la cuenta de *minterms* del lema 1, lo que lleva a un coste alto de cálculo, y no detecta otros tipos de simetrías como:

- Simetrías de una sola variable [47], en una función $f(x_1, x_2, \ldots, x_i, x_j, \ldots, x_n)$, una variable x_i es simétrica si

  $$f(x_1, x_2, \ldots, 0, 0, \ldots, x_n) = f(x_1, x_2, \ldots, 1, 0, \ldots, x_n)$$

  o

  $$f(x_1, x_2, \ldots, 0, 1, \ldots, x_n) = f(x_1, x_2, \ldots, 1, 1, \ldots, x_n)$$

- Simetrías *skew*. Una función f es *skew-nonquivalente symmetric* respecto x_i y x_j si $f_{x_i x_j} = \neg f_{x_i x_j}$ [51]. Una función f es *skew-equivalent-symmetric* respecto x_i y x_j si $f_{\neg x_i \neg x_j} = \neg f_{x_i x_j}$ [51].

Mishchenko propone en [97] un algoritmo recursivo basado en recorrido del BDD que reduce el tiempo de cálculo con la siguiente estructura, obteniendo todos los resultados a partir de un recorrido del BDD.

La idea básica del algoritmo es recorrer el BDD, calculando las soluciones de los nodos hijos del nodo raíz de BDD, y construir la solución sumando soluciones de subproblemas. La complejidad del algoritmo completo respecto a un BDD es $\mathcal{O}(|B|^3)$, donde $|B|$ es el número de nodos del BDD.

Los algoritmos de detección de simetrías son costosos y dependen del tamaño del BDD. Por tanto, en estos algoritmos suelen utilizar un buen orden inicial que reduzca el tamaño del BDD.

Algoritmo 7: Procedimiento recursivo

```
1  ProcedimientoRecursivo( P: Problema (BDD))
2  begin
3      if P es el caso trivial then
4          devolver la solución
5          f
6      end
7      Dividir el problema P en dos subproblemas, P_0 (Hijo falso del nodo raiz), y P_1 (Hijo cierto nodo raíz)
8      Obtener las soluciones parciales S_0 y S_1, para los subproblemas P_0 y P_1
9      Dervivar S, la solución de P, de las soluciones parciales S_0 y S_1
10     return S
11 end
```

Si es posible identificar las variables simétricas antes de la construcción del BDD y bloquearlas como propone Moller et al. [98] y Panda et al. [105], los algoritmos de *sifting* reducen el número de nodos como se mostrará en el Capítulo 3.

2.2.4.4 **Minimización exacta**

Las heurísticas anteriores no obtienen el mejor orden posible, para generarlo debemos usar heurísticas de minimización exacta. La primera aproximación es probar todas las posibles permutaciones. Para una función f con n variables, probar todas las combinaciones tiene un coste de $\mathcal{O}(n!2^n)$, lo que hace imposible su aplicación a problemas reales.

Friedman and Supowit [55], demostraron que dado un BDD con n variables, las k primeras posiciones del mismo, con $1 \leq k \leq n$, no dependen de los niveles superiores. El algoritmo parte del valor $k = 1$, y busca que variable se puede asignar como nodo raíz minimizando el tamaño del BDD. Una vez fijado el valor de la variable k, busca el mejor valor para la variable $k + 1$, y así sucesivamente. Este algoritmo redujo el coste computacional a $\mathcal{O}(n^2 3^n)$.

Drechsler, Rolf and Drechsler and Günher [45], incluyen también información de las cotas superiores al nivel k como cota inferior, utilizando la Ecuación 2.11.

$$lower\_bound = min\_cost_I + max\{c + m_R, n - k\} + 1 \tag{2.11}$$

Donde:

- min_cost_I es el mejor de los órdenes posibles con k nodos.

- c, son los nodos de niveles $k+1, \ldots, n$ referenciados directamente por nodos de los niveles $1 \ldots, k$.
- m_R es el número de nodos para los niveles $k+1, \ldots, n$.
- Y $n-k$, son las variables no procesadas todavía

Con esta ecuación, de los posibles ordenes en el nivel k, además descarta aquellos que hacen que la cota inferior, sea mayor o igual al coste estimado de las cotas superior del nivel $k-1$. .

El algoritmo también evita el intercambio de aquellas variables que son simétricas, porque no afecta al tamaño del BDD.

El problema principal de este algoritmo es el tiempo de proceso tal y como muestra la figura 2.10, que hace inviable para CUDD [130] procesar modelos de configuración de más de 25 opciones.

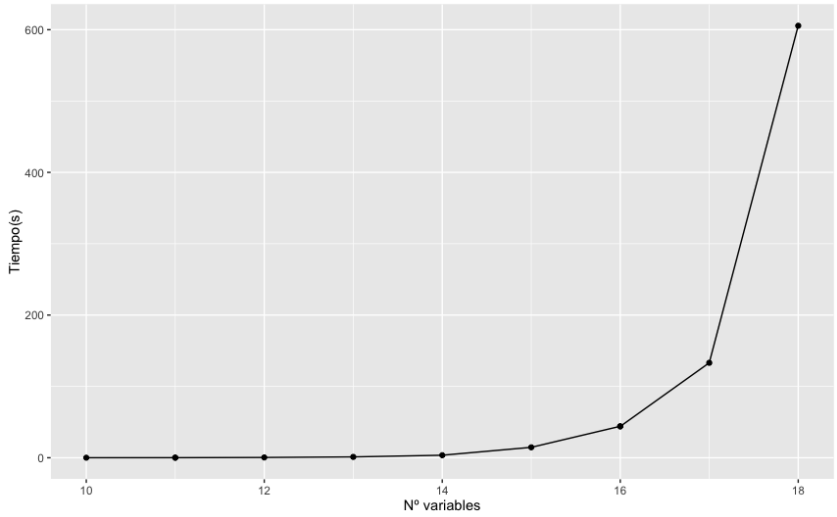


Figura 2.10: Crecimiento del tiempo proceso algoritmo [45]

Ebendt et al. [46] mejoraron el algoritmo de Drechsler, Rolf and Drechsler and Günher, utilizando la Ecuación 2.12 como cota inferior.

$$|ref(f, |K|)| - r\_upper \tag{2.12}$$

Donde $ref(f, |K|)$ denota el conjunto de nodos en los niveles $k+1, k+2, \ldots, n$ referenciados directamente en los niveles $1, 2, \ldots, k$, y r_upper el número de nodos

de salida de los niveles $1, 2, \ldots, k$, que llevan nodos de niveles mayor que k. Esta cota, previene el sobre crecimiento (debido al intercambio local de BDD), testeando si el BDD parcial es mayor en $0,7$ del BDD anterior, y lo introduce en la cache del BDD.

2.2.4.5 Heurísticas generalistas

En general, la ordenación de variables de un BDD puede considerarse un problema de optimización. En consecuencia, pueden utilizarse variaciones de algoritmos de optimización, como veremos a continuación.

Branch and bound

Las cotas superiores e inferiores del número de nodos de un BDD pueden calcularse más eficientemente si es conocida la topología. Aziz et al. [7] proponen una cota superior para FSM (Finite State Machine), que combinan con algoritmo *greedy* para obtener el mejor orden para las k primeras variables con información de la topología del problema. Una vez seleccionado el orden, el algoritmo continúa con una selección *branch and bound* o *greedy*, con la construcción del BDD.

Heurística de inserción arbitraria

Beyer [18] propuso un algoritmo específico para autómatas paralelos de tiempo, compuestos por n autómatas $A_1, A_2, \ldots A_n$. El algoritmo construye un grafo de comunicación, donde se añaden aristas entre los autómatas que comparten variables en las transacciones, y utilizan el resultado de Aziz et al. [7], que indica que el tamaño del BDD es proporcional a la complejidad de la comunicación.

El algoritmo de Beyer parte de dos variables, y utilizando técnicas de inserción arbitraria de variables [85] (derivadas del algoritmo del viajante), selecciona una nueva variable que minimiza la cota superior.

Simulated ANNEALING

Bollig et al. proponen en [22] un algoritmo que partiendo de un BDD B, genera ordenes aleatorios para el intercambio de variables, de los BDD's generados, selecciona aquel que minimiza el número de nodos, que denomina B_{new}. En el caso,

que no encuentre un BDD que minimice el número de nodos, usa una heurística de *simulated ANNEALING*. Esta heurística acepta ordenes aleatorios, generando un valor entre 0 y 1, y lo selecciona si el resultado es menor que:

$$\frac{e^{|B|-|B_{new}|}}{T}$$

Dónde T es una constante decreciente de temperatura, inicializada con el tamaño inicial del BDD.

Scatter search

Hung et al. [71] utilizan un algoritmo clásico de *scatter search* [72] para obtener un buen orden. El algoritmo parte de un orden inicial (π_0) y genera $\{\pi_1, \pi_2, \ldots, \pi_n\}$ permutaciones sobre el orden inicial. Entonces aplica *sifting* [114] para optimizar cada solución encontrada, y genera el conjunto $RefSet$ de resultados.

El algoritmo separa los b_1 mejores de resultados $RefSet_1$ y los b_2 con mayor diversidad $RefSet_2$, y menor tamaño. La diversidad, se calcula en términos de distancia. La distancia entre dos soluciones $d(\pi_1, pi_2)$, se calcula como la suma de las diferencias de las posiciones de cada variable, en los dos ordenes. La diversidad $div(\pi_i)$, de una solución π_i se calcula mediante la Ecuación 2.13.

$$div(\pi_i) = min\{d(\pi_i, \pi_j) | \forall \pi_j \in (RefSet_1 \cup RefSet_2) \wedge \pi_i \neq \pi_j\} \qquad (2.13)$$

Con el conjunto completo $RefSet = RefSet_1 \cup RefSet_2$, combina todos los posibles elementos cogidos de dos en dos, e incluye en cada uno de ellos, el elemento no seleccionado con menor tamaño de BDD, luego se aumenta hasta tres elementos con el mismo procedimiento, y luego hasta cuatro, y finalmente los seleccione los $5 \leq i \leq b_1 + b_2$, mejores.

Para cada subconjunto generado S_j con ordenes $\{\pi_1^{'}, \pi_1^{'}, \ldots, \pi_t^{'}\}$ el algoritmo genera una nueva posición para cada variable x_i, mediante la siguiente ecuación:

$$new\_pos(x_i) = \frac{l_{f,\pi_1^{'}}(x) + l_{f,\pi_2^{'}}(x) + \ldots + l_{f,\pi_t^{'}}(x)}{t} \qquad (2.14)$$

Después del cálculo de los nuevos ordenes el algoritmo los optimiza aplicando *sifting*, y vuelve aplicar el algoritmo de *scatter*, hasta que no se mejora o bien se supera el máximo número de iteraciones.

Hung et al. [71], reducen el número de nodos respecto a Rudell [114], y tanto los nodos, como el tiempo, con Bollig et al. [22].

2.2.4.6 **Selección de heurística dinámica**

Una vez construido el BDD, este puede mejorarse aplicando técnicas de reordenación. Pero, dependiendo del problema y su codificación como BDD puede ser mejor aplicar una u otra reordenación. En esta sección presentaremos dos heurísticas, aplicadas máquinas finitas de estado y *fault tree's* que selecciona la mejor reordenación.

La selección de una heurística dependiendo del problema, no sólo es aplicable a las heurísticas dinámicas. Pero, en el análisis sistemático realizado, sólo hemos identificado su aplicación en heurísticas dinámicas.

Máquinas de aprendizaje y artefactos de decisión

Grumberg et al. [65] utilizan árboles de decisión generados por máquinas de aprendizaje, para obtener un orden bueno para la codificación de una FSM como BDD.

El algoritmo genera un orden basado en *pares de precedencia* de dos variables. Un par de precedencia $x_i \prec x_j$ denota que la variable x_i precede a la variable x_j en el orden. Para un orden $\pi = \{x_a, x_b, x_c, x_d\}$, tenemos los siguientes pares de precedencia $\{x_a \prec x_b, x_a \prec x_c, x_a \prec x_d, x_b \prec x_c, x_b \prec x_d, x_c \prec x_d\}$, en consecuencia encontrar los pares de precedencia es equivalente a encontrar el orden de las variables.

El algoritmo de Grumberg et al. genera un vector etiquetado con atributos y valores, a los posibles pares de precedencia $x_i \prec x_j$ o $x_j \prec x_j$. Las etiquetas se definen a partir de atributos extraídos del problema. Por ejemplo, si dos variables son simétricas, se utiliza dicho vector para seleccionar el orden. Para generar el vector indexado se sigue el siguiente algoritmo:

1. Crear ejemplos de posibles órdenes para problemas reales o aleatorios.

2. Transformar el problema a BDD generando todos los órdenes posibles.

3. Encontrar todas las variables que interactúan con otras en cada BDD generado.

4. Etiquetar cada par de variables dependientes con los atributos predefinidos, seleccionando el mejor orden.

5. Transformar cada etiqueta y orden en un vector etiquetado con atributos.

6. Crear un clasificador basado en el vector etiquetado con atributos y valores.

El procedimiento genera múltiples vectores etiquetados con atributos y valores para todos los ejemplos. El algoritmo mezcla los valores seleccionado aquellas posiciones de variables que aparecen más veces (las más votadas).

Lam et al. [84], utilizan una máquina de aprendizaje automático (*machine learning*), para realizar un Análisis Sensitivo de Contexto (Context-Sensitive Analysis) de código fuente, con BDD's. El algoritmo asigna una puntuación a cada ordenación, que combina información del tiempo de construcción del BDD, y el orden de las variables dentro de las cláusulas, y construye un predictor con dicha información.

El BDD se construye de forma incremental, añadiendo cláusulas, si el tiempo es mayor que un determinado valor, se inicia el proceso de aprendizaje, que utiliza el predictor para obtener un nuevo orden.

Redes neuronales

El problema de codificación de *fault trees* utilizando BDD's, es un problema con amplia producción durante los últimos 20 años [115]. Como, por ejemplo, Coudert et al. [36], Bouissou [23] o Nusbaumer et al. [103]. Bartlett and Andrews [10] proponen el uso de una red neuronal, para seleccionar una heurística, de seis posibles.

Bartlett and Andrews usan las características del problema como nodos de entrada de la red neuronal, y la heurística seleccionada como salida, y el tamaño del BDD como selector de calidad. Los autores utilizan dos tipos de redes neuronales *multil-layer* y *radial basic functions*.

Las heurísticas de selección son posibles cuando el problema es conocido y podemos utilizar atributos de este, como selectores. Esta heurística tiene muy buenos resultados, obteniendo en todos los casos, la mejor o segunda mejor heurística.

2.3 Técnicas complementarias

2.3.1 Introducción

En la sección anterior hemos descrito heurísticas de ordenación de variables para reducir el tamaño del BDD. Además, se pueden utilizar técnicas complementarias, como la descomposición del problema en BDD más pequeños, o realizar reducciones en las funciones Booleanas.

2.3.2 Descomposición de BDD's

Cuando una función Booleana es representada por un BDD, puede ser descompuesta, y utilizar alguna de las siguientes técnicas [23], [28], [26] y [76].

Reducción de fórmulas Booleanas

Es posible disminuir la complejidad de las funciones utilizando técnicas de reducción antes de construir el BDD [35], [40], [128],y [134]. Otra aproximación, es construir el BDD y posteriormente reducir la fórmula directamente en el BDD, [126] y [70].

2.4 Operaciones con modelos configuración

2.4.1 Introducción

Existen múltiples operaciones de análisis sobre modelos de configuración que son de interés. En Benavides et al. [12], puede consultarse una enumeración exhaustiva. A continuación, se describen algunas operaciones cuyo cómputo con SAT *solvers* es muy costoso, y por tanto, el uso de BDD's es más aconsejable:

1. Configuración vacía (*Void model*).
2. Configuración valida (*Valid product*).
3. Configuración parcial válida (*Valid partial configuration*).
4. Todas las configuraciones (*All products*).
5. Número de configuraciones (*Number of products*).
6. Características no validas (*Dead features*).
7. Características siempre válidas (*Core features*).
8. Características requeridas de una opción (*Impact set*).
9. Características excluidas de una opción (*Exclusion set*).

En esta sección se identificará la función Booleana $\psi(x_1, x_2, \ldots, x_n)$ como la que codifica el modelo configuración $M_c = \{F, C\}$, y como B el BDD que codifica ψ.

2.4.2 *Void model*

Un *void model* es un modelo de configuración (M_c) para el que no puede obtenerse ninguna instancia válida, es decir, la función Booleana ψ que lo representa no es satisfacible.

Su verificación con un *SAT solver* implica el cálculo de la satisfacibilidad del problema, en consecuencia, su coste es $\mathcal{O}(2^n)$.

Alternativamente, el BDD B, será vacío con un sólo nodo el 0-terminal, tal y como se muestra en Figura 2.11. Su verificación, es simple, comprobar si las aristas del nodo raíz, nos llevan al nodo 0-terminal. Por tanto, su coste es $\mathcal{O}(1)$.

Figura 2.11: *Void model*

2.4.3 *Valid product*

Un *valid product* o producto válido, es una asignación $\psi_{x_1=b_1,x_2=b_2,\ldots,x_n=b_n}$ satisfacible. Por tanto, su coste con SAT es $\mathcal{O}(2^n)$.

La ventaja de los BDD's reside en que una vez construido, puede evaluarse si una asignación es válida con coste $\mathcal{O}(n)$, y puede reutilizarse para comprobar cualquier asignación. En cambio, en un SAT *solver*, implica resolver un problema nuevo cada vez.

2.4.4 *Valid partial configuration*

Una *Valid partial configuration* o asignación parcial, implica que $\psi_{x_1=b_1,v_2=x_2,\ldots,x_k=b_k}$ es satisfacible, es decir, que la siguiente fórmula Booleana es satisfacible.

$$\psi_{x_1=b_1,x_2=b_2,\ldots x_k=b_k} \equiv \psi \wedge (x_1 \iff b_1 \wedge x_2 \iff b_2 \wedge \ldots \wedge x_k \iff b_k)$$

Comprobar la satisfabilidad de una *valid partial configuration*, es equivalente a comprobar la satisfibalidad de un nuevo problema, en consecuencia, su coste es $\mathcal{O}(2^n)$ con un SAT *solver*.

En cambio, con un BDD el coste es la suma de:

- Recorrer el camino de la asignación parcial $\mathcal{O}(k)$, que nos llevará al nodo x_{k+1}, que repesenta la función $\psi_{x_1=b_1,x_2=b_2,\ldots x_k=b_k}$.
- Y verificar si el BDD, que tiene como nodo de partida x_{k+1}, es satisficable. Por tanto, aplicar a la operación `Satisfy-one` que definió Bryant en [25], $\mathcal{O}(n-k)$.

En consecuencia, el coste de verificar una asignación parcial en un BDD es $\mathcal{O}(n)$. Además, el BDD puede ser reutilizado, para verificar otras asignaciones parciales.

2.4.5 *All products*

All products son todas las opciones de configuración válidas para un determinado modelo.

Obtener todas las opciones de configuración con un SAT *solver* es equivalente a verificar la satisfabilidad de todas sus posibles soluciones, en consecuencia, su coste es $\mathcal{O}(2^n) \cdot \mathcal{O}(2^n) = \mathcal{O}(2^{2n})$.

Para su obtención en un BDD se debe aplicar la operación *Satisfy-all* ($\mathcal{O}(|B|)$)[25], que obtiene todas soluciones de la fórmula Booleana del modelo de configuración.

2.4.6 *Number of products*

Esta operación cuenta el número de posibles configuraciones de un modelo. Esta operación se calcula de igual modo a *All products* tanto para SAT *solvers* como BDD's.

2.4.7 *Dead features*

Una característica f es *dead* en un modelo de configuración representado por la fórmula Booleana ψ si y sólo sí $f \wedge \psi$ no es satisfacible. En el caso de un SAT *solver* es equivalente a verificar la satisfisfabilidad de $f \wedge \psi$, es decir, $\mathcal{O}(2^n)$.

El coste para determinar si una característica es *dead* c_{dead}, es igual al coste de realizar la operación $f \wedge \psi$, en consecuencia $c_{dead} = \mathcal{O}(|1| \cdot |B|) = \mathcal{O}(|B|)$.

2.4.8 *Core features*

Una característica f es *core* en un modelo de configuración representado por la fórmula Booleana ψ (codificada en el BDD B) si y sólo sí $f \wedge \psi$ es una tautología. Por tanto, el coste es idéntico al coste de las características *dead*, ya que debemos responder a las mismas cuestiones.

2.4.9 *Impact set*

Dadas dos características f y $f^{'}$, $f^{'}$ es *impact set*, si y sólo sí $f \wedge \psi \wedge f^{'}$ es una tautología. Es decir, su coste tanto para SAT *solvers* como BDD's es $\mathcal{O}(2^n)$.

El coste de la obtención de todos los *impact set* de un modelo de configuración M_c con n opciones de configuración. Se deben probar todas las combinaciones de dos elementos sin repetición es decir $\frac{n^2-n}{2}$, por tanto, su coste es $\mathcal{O}(\frac{n^2-n}{2}) \cdot \mathcal{O}(2^n) = \mathcal{O}(n^2 \cdot 2^n)$.

2.4.10 *Exclusion set*

Dadas dos características f y $f^{'}$, $f^{'}$ es *exclusion set*, si y sólo sí $f \wedge \psi \wedge f^{'}$ no es satisfacible. Por tanto, el coste de es el mismo que la obtención de todos *impact set*.

2.4.11 **Conclusiones**

Los BDD's, son mejores que los SAT *solvers*, cuando se realizan múltiples operaciones, ya que el BDD codifica la fórmula completa, y se pueden añadir restricciones partiendo de dicho BDD, en cambio en el caso de los SAT *solvers* es necesario volver a ejecutar el proceso.

El coste de las operaciones con un BDD depende del tamaño de éste. En consecuencia, es necesario tener un buen orden para que las operaciones sean eficientes.

Un experimento es una pregunta que la ciencia plantea a la naturaleza, y una medición es el registro de la respuesta de la naturaleza.

Max Planck

CAPÍTULO

3

Análisis sistemático de las heurísticas existentes

3.1 Introducción

Antes del desarrollo de nuevas heurísticas es necesario analizar los resultados de las existentes, para así, identificar las adecuadas para modelos de configuración. Si bien es cierto, que Mendonca et al. [94], realizaron un análisis, este fue limitado y no sistemático.

Para la realización del análisis, se ha definido una metodología compuesta por un *benchmark*, la herramienta **test_bdd_heuristics**, y un procedimiento de análisis, que se detallarán a continuación.

Primero se han analizado las heurísticas estáticas, mediante una revisión por expertos. Con los resultados obtenidos, se han seleccionado las mejores, que se han combinado con las heurísticas dinámicas, con el objetivo de seleccionar la mejor combinación posible de ambas heurísticas, realizando un análisis conjunto.

3.2 Metodología

3.2.1 Benchmark

El *benchmark* incluye 29 modelos de configuración de sistemas reales compuestos por:

- Los 12 modelos de configuración reales con mayor número de opciones del repositorio de SPLOT (http://www.splot-research.org/), procedentes de artículos científicos.
- 10 sub. sistemas de eCos.
- 7 sub. sistemas de Linux.

Los modelos se detallan en la Tabla 3.1, y están accesibles en el repositorio GitHub de la tesis[1], en formato SXFM [92].

3.2.2 Heurísticas estáticas para analizar

Se han analizado las heurísticas estáticas descritas en la sección 2.2 del Capítulo 2 de las cuales sus autores describían el algoritmo con suficiente detalle para su implementación y pueden ser aplicadas a modelos de configuración.

- fujita (Fujita et al. [57]).
- malik_fanin (Malik et al. [88]).
- malik_level (Malik et al. [88]).
- fujii (Fujii et al. [56]).
- narodyska (Narodytska and Walsh [100])
- FORCE (Aloul et al. [4]).
- MINCE (Aloul [3]).
- mendonca (Mendonca et al. [94]).

[1] https://github.com/robcbean/thesis_data/tree/master/xml

Se ha incluido también la ordenación en preorden, partiendo del nodo raíz del modelo de configuración y descendiendo en preorden. La inclusión del preorden, que según indican Mendonca et al. [94] produce buenos órdenes, nos permite tener una heurística base con la que comparar.

Origen	Modelo de configuración	Nº de características	Referencia
SPLOT	Graph	30	Schobbens et al. [118]
SPLOT	Smart Home v2	38	Alferez et al. [1]
SPLOT	Assess4me	40	Chaudy et al. [30]
SPLOT	Web_Portal	43	Mendonca and Cowan [93]
SPLOT	Documentation_Generation	44	van Deursen and Klint [136]
SPLOT	Thread	44	Beuche [17]
SPLOT	HIS	67	Kang et al. [81]
SPLOT	Model_Transformation	88	Czarnecki and Helsen [38]
SPLOT	SmartTV	96	Gebizli and Sözer [62]
SPLOT	Dell_XML	118	Nohrer and Egyed [102]
SPLOT	BankingSoftware	176	Millo et al. [96]
SPLOT	Electronic Shopping	290	Quan Lau [110]
eCos	fs	22	
eCos	infra	29	
eCos	compat	30	
eCos	services	55	
eCos	kernel	59	
eCos	redboot	65	
eCos	language	83	
eCos	net	108	
eCos	io	132	
eCos	isoinfra	157	
Linux	block	54	
Linux	security	59	
Linux	mm	64	
Linux	arch	113	
Linux	lib	115	
Linux	crypto	266	
Linux	fs	318	

Tabla 3.1: Benchmark

3.2.3 Herramienta desarrollada: bdd_heur_analyisis

bdd_heur_analyisis utiliza el paquete CUDD [130], ya que es el más citado en los artículos analizados en el Capítulo 2, además, es el más actualizado, como muestra la Tabla 3.2.

La herramienta **bdd_heur_analysis** ha sido implementada en Ruby por ser un

Paquete	Nº de artículos	Año última actualización
CUDD [130]	109	2016
BuDDy [87]	19	2014
CAL [117]	7	1998
JavaBDD [140]	4	2013

Tabla 3.2: Paquetes BDD

entorno multiplataforma, y está disponible en el repositorio GitHUB [1] bajo licencia GNU V3.

bdd_heur_analysis implementa las heurísticas estáticas que se detallarán a continuación, para modelos de configuración en formato SXFM [92], o circuitos en formato ISCAS 89 netlist [24], generando su función Booleana, y transformando dicha función a BDD. Esta herramienta permite, una vez construido el BDD, aplicar una heurística dinámica implementada en CUDD.

La herramienta ha sido testada en los entornos Linux (Ubuntu 16.04 LTS), Apple Mac OS X(Sierra y High Sierra) y Microsoft Windows 10 con Cygwin con Ruby 2.X.

La herramienta se ejecuta mediante la siguiente línea de comando,

Transcripción 4

```
Usage: test_bdd_heuristics.rb [options]
-f, --file file                  File or directory
-e, --heuristics file            Heuristics to analyze
-t, --wait_time wait_time        Wait time
-a, --attemps attemps            Attemps
-m, --minisat                    Launch minisat++
-b, --best_span                  Select best span
-d, --calc_des                   Calculate descomposition
-w, --write_dot                  Write dot file
-c, --mcl_range from-step-to     Mcl range
-o random_factor ,               Random factor
--random_factor
-s, --count_symmetric            Count symmetric continuous before and after
-y, --enable-autodyn             Enable AutoDyn reorder
-h, --help                       Display Help
```

donde:

- `-f` es un fichero en formato SXFM o ISCAS 89 netlist, o bien un directorio que contiene ficheros con dichos formatos.

[1] https://github.com/robcbean/bdd_heur_analysis

- `-e`, es el fichero que contiene las heurísticas a analizar. En caso de no indicarlo, la herramienta busca el fichero `heuristics.txt`.
- `-t`, si la construcción del BDD y su reordenación superan el tiempo en segundos indicado en este parámetro, se finaliza la construcción del BDD, por defecto, 10 minutos.
- `-r`, es el número de ordenaciones aleatorias que se aplicará al modelo. Con el objetivo de comparar los resultados de las heurísticas con órdenes aleatorios.
- `-a`, es el número de ejecuciones para una misma heurística. Utilizado para realizar promedios de tiempo, y para las heurísticas no deterministas, como, por ejemplo, **mendonca** [94].
- `-m`, permite indicar si se ejecuta **minisat+** o no para la optimización de los grupos[1].
- `-b`, selecciona la heurística con menor *span*[2].
- `-d`, descompone el modelo de configuración en dos BDD's uno para las restricciones *crosstree* y otra para las jerárquicas, calculando el tiempo de identificación de las características *core* y *dead* con el modelo completo o bien con los modelos separados.
- `-w`, genera el un fichero en el lenguaje DOT (graphviz) que representa el BDD del modelo de configuración.
- `-c`, permite indicar el rango del parámetro I del algoritmo identificación de clústeres de Dongen [44].
- `-s`, calcula el número posible de pares de variables simétricas, el porcentaje que hay en el BDD, y el porcentaje de ellas que son contiguas.
- `-y`, habilita la reordenación dinámica mientras se construye el BDD. Por defecto, esta desactivada y se aplica una vez construido.
- `-h`, muestra la ayuda de la herramienta.

[1] Se describirá en el Capítulo 4
[2] Se describirá en el Capítulo 4

El fichero de las heurísticas a analizar tiene el siguiente formato:

Transcripción 5

```
"heuristic",reorder,grouping
```

donde `heuristic` es el nombre de la heurística, pudiendo ser, uno de los siguientes valores, que se muestran en la Tabla 3.3.

Heurística	identificador
Fujita et al. [57]	fujita
Malik et al. [88] fanin.	malik_fanin
Malik et al. [88] level	malik_level
Fujii et al. [56]	fuji
Narodytska and Walsh [100]	narodyska
Aloul et al. [4]	FORCE
Aloul et al. [3]	MINCE
Mendonca et al. [94]	mendonca

Tabla 3.3: Identificadores de las heurísticas

`reorder` indica la heurística de ordenación dinámica de CUDD, y `grouping` puede tener valor Y (Sí) o N(No) en caso de que se desee que se agrupen las variables para la ordenación dinámica. Este parámetro sólo debe tener valor cierto en heurísticas que agrupen variables.

Se pueden añadir comentarios anteponiendo el carácter # a la línea, y se pueden incluir múltiples heurísticas, tal y como muestra el siguiente ejemplo, en el que se realiza heurística estática Fujita et al. (línea 1) y aplicando posteriormente *sifting* Fujii et al. (línea 2), variables simétricas agrupadas con grupos Panda et al. (línea 3) y sin grupos Rudell (línea 4).

Transcripción 6

```
"fujita88"
"fujita88",CUDD_REORDER_WINDOW2
"FORCE",CUDD_REORDER_GROUP_SIFT,Y
"FORCE",CUDD_REORDER_SIFT,N
```

La herramienta genera como salida la siguiente información:

- `file`, nombre del modelo de configuración.
- `order`, heurística utilizada de la Tabla 3.3.
- `reorder`, heurística de ordenación dinámica.

- `algo_time`, tiempo en segundos de proceso de la heurística.
- `bdd_time`, tiempo en segundos de construcción del BDD completo.
- `minterms`, número de minterms (soluciones) del BDD.
- `span`, *span* del orden utilizado.
- `inputs`, número de opciones de configuración/variables.
- `clauses`, número de clausulas.
- `functions`, número de funciones del modelo.

Las siguientes salidas sólo se generan si se activa la opción `-d` (descomposición) en cuyo caso se añade la siguiente información:

- `and_time`, tiempo en segundos de la realización del AND entre las restricciones jerárquicas y *crosstree*.
- `reach_one_decomposed`, tiempo en segundos del cálculo del Algoritmo 12, calculado con dos BDD's, una para las restricciones jerárquicas y *crosstree*.
- `reach_one_total`, tiempo en segundos del cálculo del Algoritmo 12, calculado con el BDD completo.

Si se añade la opción `-s`, se añade la siguiente información de simetría:

- `total_possible_symmetric` , número total de pares de variables posibles.
- `symmetric_percent`, el número de pares de variables simétricas respecto al total de posibles.
- `symmetric_continuos`, número de pares de variables simétricas que son contiguas respecto al total de posibles.

3.2.4 Análisis estadístico

Para el análisis estadístico primero definiremos el objetivo del análisis, a continuación, el modelo, y finalmente la herramienta que utilizaremos: Análisis de medidas repetidas no paramétrico.

El objetivo del análisis es realizar un estudio comparativo de las heurísticas de ordenación de BDD's existentes para el *benchmark* de modelos de configuración. Es decir, se estudiará el resultado de n heurísticas diferentes para un conjunto de k modelos de configuración, utilizando para ello un análisis de varianza o ANOVA (Analysis of VAriance).

La variable explicativa será la heurística, y las de respuesta serán:

1. El nº de nodos del BDD.
2. El tiempo de las heurísticas de ordenación.
3. El tiempo de construcción del BDD.

El objetivo es identificar cual son las mejores heurísticas, tanto en tamaño del BDD, como en tiempo de proceso.

Modelo

Se utilizará el modelo de medidas repetidas (*repetead measures*), ya que para la misma población (*benchmark*), se analiza su variación respecto a una variable explicativa (heurísticas).

En el modelo *repetead measures* se pueden utilizar test paramétricos o no paramétricos. Los primeros precisan que las variables de estudio tengan una distribución normal como la que muestra la Figura 3.1, en cambio los segundos no precisan que las variables se ciñan a estos parámetros.

Las variables de respuesta, el número de nodos del BDD y el tiempo de construcción tiene un crecimiento exponencial respecto al número de variables n, $\mathcal{O}(2^n)$. Para utilizar modelos lineales, se ha aplicado una transformación logarítmica en base 2 a las variables dependientes. En las Figuras 3.2(a), 3.2(b), 3.3(a) y 3.3(b), podemos observar como las diferencias se pueden apreciar mejor de este modo. En el caso del análisis de tiempo se ha eliminado el preorden, ya que no existe ninguna heurística en este caso, y por tanto no hay tiempo.

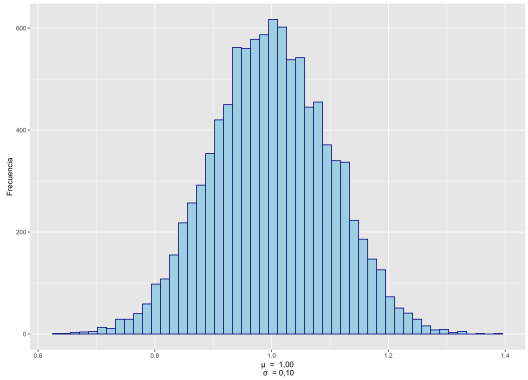


Figura 3.1: Distribución normal con media 1 y desviación típica 0,1

El ANOVA de *repetead measures* paramétrico precisa la condición de esfericidad (*sphericity*), es decir que las diferencias de varianzas entre todas las combinaciones de los grupos, y dentro de los grupos, sean iguales. El test Mauchly's [90], permite verificar esta condición, cuya hipótesis nula H_0, es que todas las varianzas son iguales.

Si aplicamos el test para el logaritmo del número nodos y tiempo obtenemos valores de p de $8,0997e-42$ y $4,3412e-33$ respectivamente, tal y como muestran los resultados de **R**[1], en consecuencia, descartamos la hipótesis nula, lo que indica que las varianzas son diferentes, y no se cumple el test de esfericidad, tanto a nivel de nodos como de tiempo.

Transcripción 7

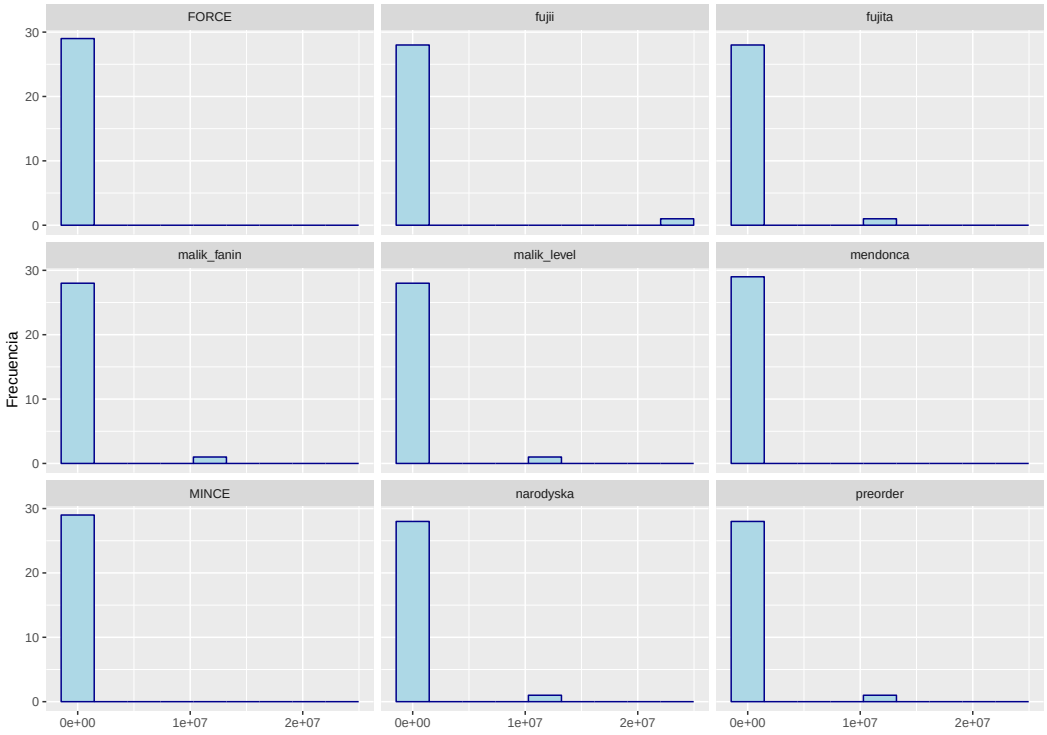
```
Mauchly Tests for Sphericity (Nodos)
Test statistic    p-value
heurFactor.input     2,2449e-05 8,0997e-42
```

Transcripción 8

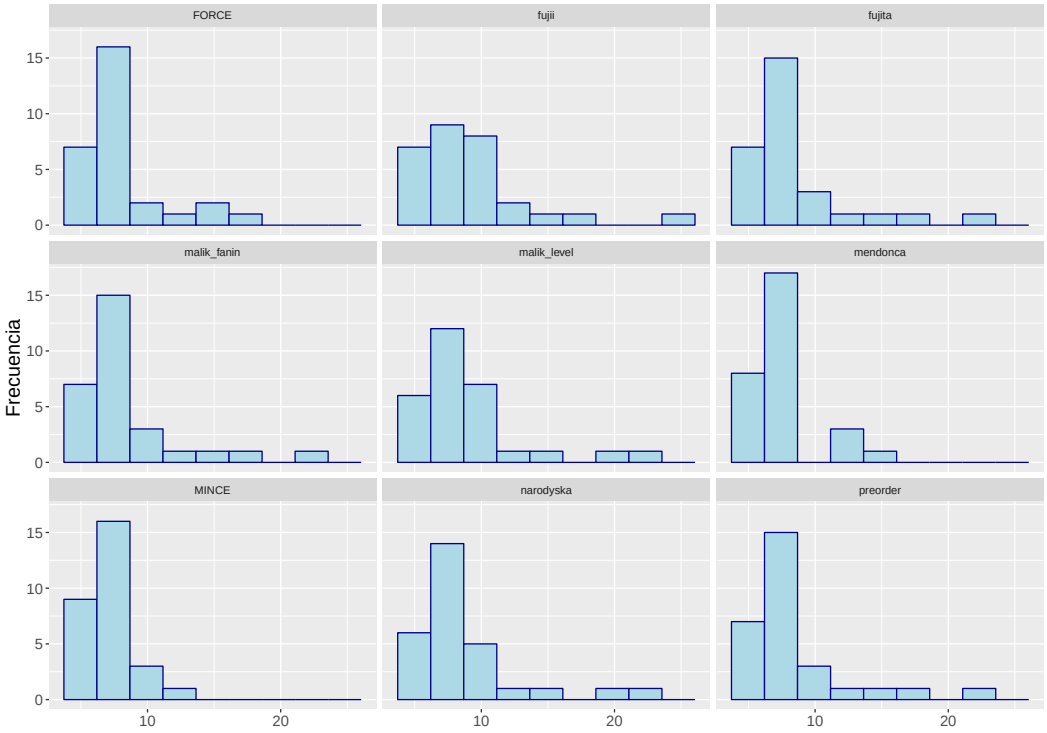
```
Mauchly Tests for Sphericity (Tiempo)
Test statistic    p-value
heurFactor.input      0,0001326 4,3412e-33
```

El grado con que la esfericidad está presente o no, se representa a través del valor estadístico epsilon (ϵ), un valor ϵ igual a uno, indica que la condición de esfericidad es exacta, utilizando este valor las correcciones Greenhouse-Geisser [64] y Huynd-Feld [73], corrigen el valor de p de acuerdo con los grados de libertad,

[1] https://www.r-project.org/

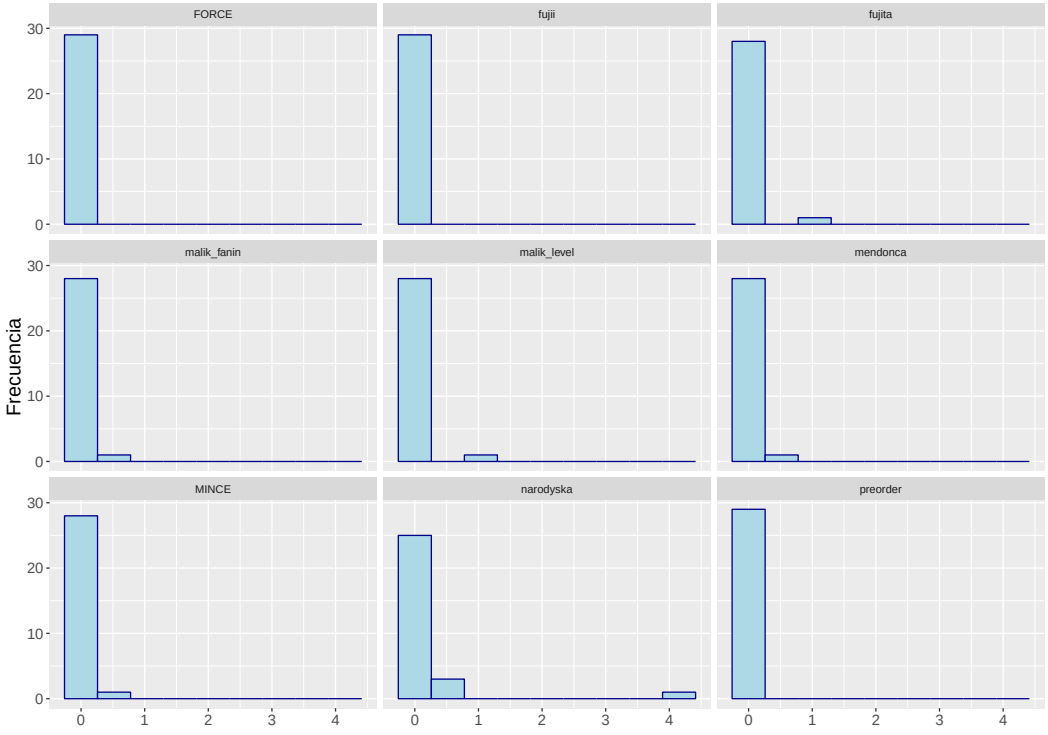


(a) Nº de nodos

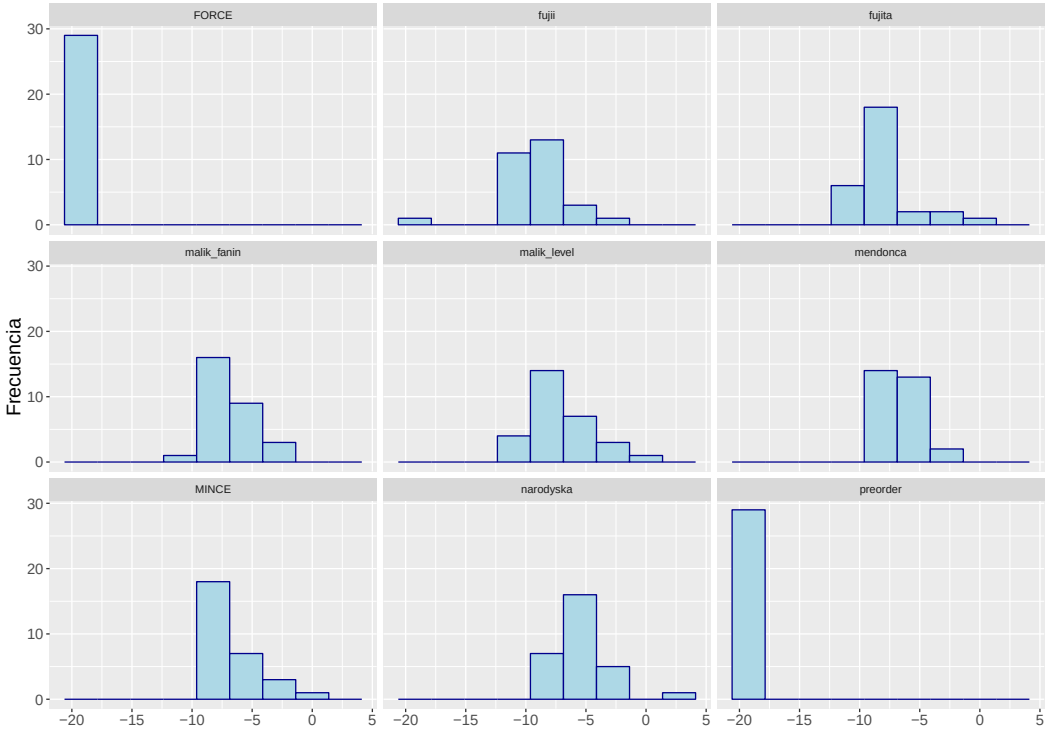


(b) Logaritmo en base 2 del nº de nodos

Figura 3.2: Nº de nodos del BDD por heurística



(a) Tiempo (s)



(b) Logaritmo en base 2 del tiempo (s)

Figura 3.3: Tiempo en segundos por heurística

pero en ambos casos para nuestros valores la corrección es insuficiente, tal y como se muestran los resultados de **R**, tanto para nodos como para el tiempo.

Transcripción 9

```
Greenhouse-Geisser and Huynh-Feldt Corrections
for Departure from Sphericity

GG eps Pr(>F[GG])
heurFactor.input 0.21858    0.00645 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

HF eps Pr(>F[HF])
heurFactor.input 0.2286836 0.00569944
```

Transcripción 10

```
Greenhouse-Geisser and Huynh-Feldt Corrections
for Departure from Sphericity

GG eps Pr(>F[GG])
heurFactor.input 0.36723  < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

HF eps  Pr(>F[HF])
heurFactor.input 0.4075602 1.74429e-31
```

Es evidente que no se cumple la condición necesaria de esfericidad para el ANOVA paramétrico, en consecuencia, no podemos utilizar este análisis y usaremos el no paramétrico, que describiremos a continuación.

Análisis de medidas repetidas no paramétrico

El análisis que utiliza la bibliografía para medidas no paramétricas ([127], [125], [34] y [82]) es el test de Friedman [54]. Este test tiene como hipótesis nula H_0 que la suma de del total de rangos de cada uno de los grupos son iguales. Es decir, que son idénticos. Para nuestro *benchmark* el valor de p, es en ambos casos, menor que $0,05$. Por tanto, existen diferencias entre las heurísticas analizadas, tanto a nivel de nodos como a nivel de tiempos.

Transcripción 11

```
Friedman rank sum test
data:  Análisis del número de nodos
Friedman chi-squared = 92.387, df = 8, p-value < 2.2e-16

Friedman rank sum test
```

```
data:  Análisis de la construcción del BDD
Friedman chi-squared = 137.4, df = 8, p-value < 2.2e-16
```

La potencia de este test ha sido cuestionada por Conover and Iman [33] [74], Zimmerman and Zumbo [142], Conover [32] y Baguley [8]. Pero Iman et al. [74] concluyen que este test es adecuado para grupos mayores que 6, como es nuestro caso, que analizamos 9 heurísticas, que detallaremos posteriormente.

El resultado del test de Friedman, indica que existen diferencias sustanciales entre las heurísticas analizadas. Nuestro objetivo se centra en comparar las heurísticas dos a dos. Para ello, aplicaremos el test de Nemenyi [101], que es el test no paramétrico equivalente al test paramétrico Tukey [79]. Este es el procedimiento de análisis que propone Demšar [42] una vez aplicado el test de Friedman.

Análisis de tiempo de las heurísticas estáticas

La heurística estática **MINCE** utiliza el algoritmo **Capo** [6] del cual sólo se dispone del fichero ejecutable, y no hay suficiente descripción para su implementación, por lo que se ha recurrido a su ejecución directa. El resto de las heurísticas han sido implementadas en Ruby, un lenguaje interpretado, por lo que no pueden ser comparables a nivel de tiempos, dado que el tiempo de proceso en Ruby es superior al de un fichero binario. En consecuencia, no podemos comparar tiempos a nivel de heurísticas estáticas.

También se ha utilizado la implementación de la heurística **mendonca** [94] realizada por los autores en **Java**, en consecuencia, tampoco es comparable, es por ello, que no se ha realizado un análisis de tiempos en las heurísticas estáticas.

Consideraciones de los test de Friedman y Nemenyi

Tanto el test de Friedman como el test de Nemenyi, al ser test no paramétricos están basados en clasificaciones de los valores. Dado un experimento con k bloques o heurísticas y n individuos o modelos de configuración, que se representa habitualmente como la siguiente matriz.

$$\begin{matrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k1} & x_{k2} & \dots & x_{kn} \end{matrix}$$

Se asigna un valor de rango a cada uno de los valores de la cada fila R_j. Si, por ejemplo, tenemos la siguiente lista de valores:

$$R_j = \{R_{j1} = 31; R_{j2} = 23, R_{j3} = 5; R_{j4} = 15; R_{j5} = 10; R_{j6} = 33; R_{j7} = 5\}$$

Se ordenan los valores en orden ascendiente (S_j) y se asigna a cada elemento la posición que ocupan dentro del orden (R_j). En el caso que un valor se repite varias veces, se hace la media de las posiciones que ocupan, los valores. Por ejemplo, en este caso el orden sería. Y en el caso del valor 5 que ocupa las posiciones 1 y 2 su valor de rango $R\#$, sería $\frac{1+2}{2} = 1,5$.

$$R_j = \{R_{j1} = 31; R_{j2} = 23, R_{j3} = 5; R_{j4} = 15; R_{j5} = 10; R_{j6} = 33; R_{j7} = 5\}$$
$$S_j = \{S_{j1} = 5; S_{j2} = 5; S_{j3} = 10; S_{j4} = 15; S_{j5} = 23; S_{j6} = 31; S_{j7} = 33\}$$
$$R\#(R_j) = \{R\#_{j1} = 6,0; R\#_{j2} = 5,0; R\#_{j3} = 1,5; R\#_{j4} = 4,0; R\#_{j5} = 3,0; R\#_{j6} = 7,0; R\#_{j7} = 1{,}5\}$$

Luego, se comparan los valores de rango de cada uno de los diferentes modelos de configuración para las diferentes heurísticas. En este caso no se están comparando los valores si no su rango. Por tanto, para los análisis no paramétricos, los resultados de aplicar el logaritmo serán idénticos que no aplicar los logaritmos.

3.3 Heurísticas estáticas

Como se ha indicado anteriormente existen diferencias entre heurísticas. Para realizar la comparativa, se ha realizado el Test Nemenyi [101], junto con un análisis descriptivo.

3.3.1 Resultados del Test Nemenyi y análisis descriptivo

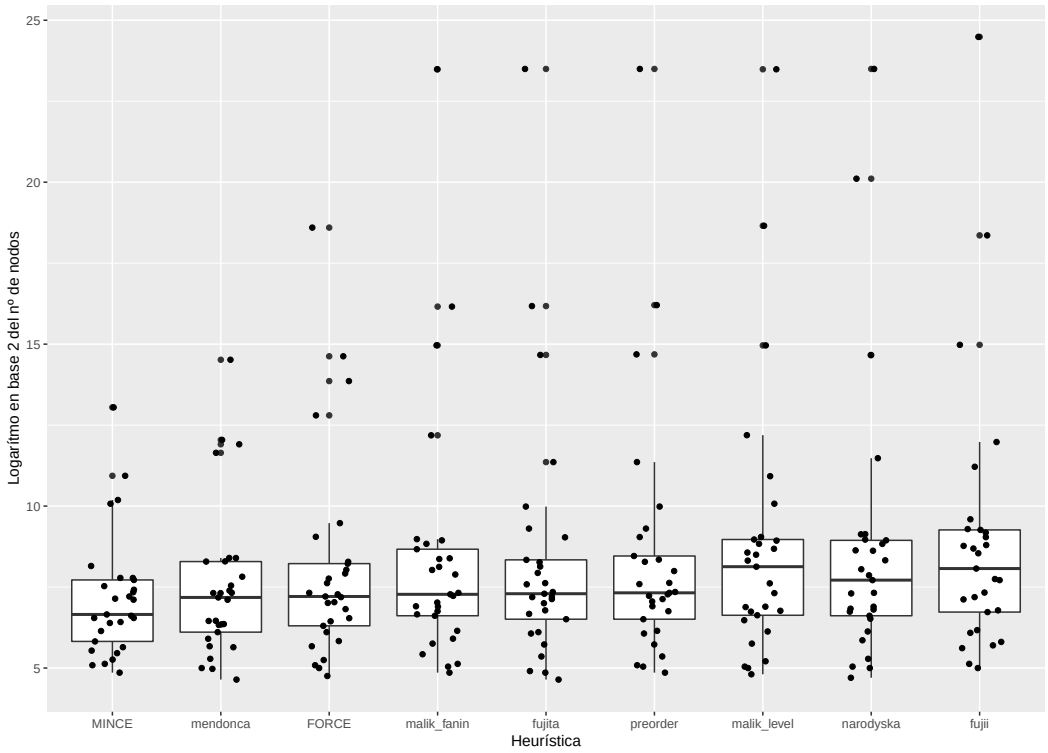
La Tabla 3.4 muestra los resultados del valor de p para el *benchmark*. Los valores en azul indican aquellos cuyo valor es inferior el límite de referencia marcado $0,05$. El detalle de las diferencias se visualiza en el diagrama de caja de la Figura 3.4(a) y los resultados de estadística descriptiva en la Tabla 3.4, junto con el gráfico de barras de la media por heurística de la Figura 3.4(b).

V.p / V.p	malik_fanin	FORCE	narodyska	malik_level	fujii	fujita	mendonca	MINCE	preorder
malik_fanin	-	0,401870	0,999322	0,999999	0,136262	0,433812	3,991e-05	0,000486	0,999997
FORCE	0,401870	-	0,827417	0,584387	2,775e-05	1,000000	0,161945	0,450093	0,223460
narodyska	0,999322	0,827417	-	0,999987	0,020741	0,851160	0,000823	0,006894	0,989357
malik_level	0,999999	0,584387	0,999987	-	0,067840	0,618029	0,000144	0,001516	0,999717
fujii	0,136262	2,775e-05	0,020741	0,067840	-	3,538e-05	1,125e-12	5,499e-11	0,272105
fujita	0,433812	1,000000	0,851160	0,618029	3,538e-05	-	0,144459	0,417730	0,247028
mendonca	3,991e-05	0,161945	0,000823	0,000144	1,125e-12	0,144459	-	0,999853	7,955e-06
MINCE	0,000486	0,450093	0,006894	0,001516	5,499e-11	0,417730	0,999853	-	0,000115
preorder	0,999997	0,223460	0,989357	0,999717	0,272105	0,247028	7,955e-06	0,000115	-

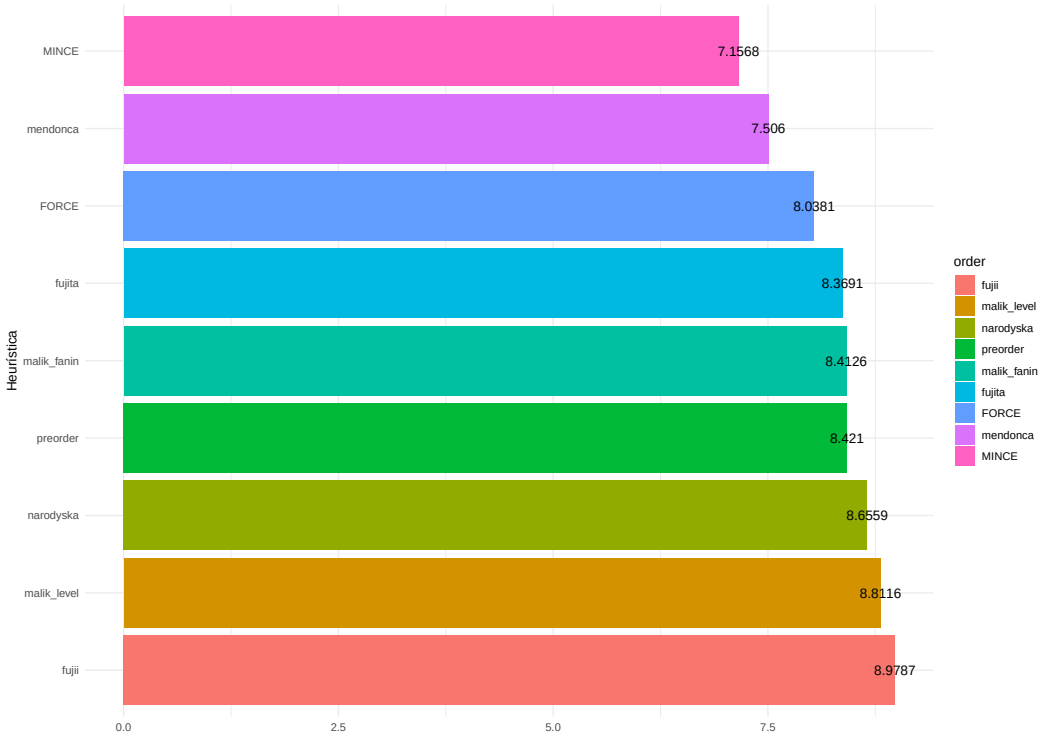
Tabla 3.4: Valor de p del test Nemenyi del *benchmark*

Heurística	Media(log_2)	Mediana(log_2)	Desv. std.(log_2)
MINCE	7,156786	6,654285	1,883329
mendonca	7,506047	7,179909	2,337459
FORCE	8,038065	7,209453	3,159272
fujita	8,369092	7,294621	3,898896
malik_fanin	8,41262	7,276124	3,906621
preorder	8,420962	7,321928	3,876609
narodyska	8,655852	7,714246	4,190039
malik_level	8,811614	8,129283	4,108071
fujii	8,978659	8,071462	4,147429

Tabla 3.5: Media, mediana y desviación estándar nº de nodos por heurística



(a) Diagrama de caja del logaritmo en base 2 del número de nodos



(b) Media del logaritmo en base 2 del número de nodos

Figura 3.4: Análisis gráfico del número de nodos por heurística

3.3.2 Conclusiones heurísticas estáticas

MINCE y **mendonca** son las heurísticas que el test identifica como diferentes a nivel de mediana con el mayor número de heurísticas. Este dato es avalado por la estadística descriptiva, ya que ambas heurísticas son las que obtiene los resultados medios, de mediana y desviación típica menores. Podemos concluir que son las mejores heurísticas para el análisis de modelos de configuración. Ambas intentan minimizar las conexiones entre clausulas, **mendonca** agrupando por clústeres, y **MINCE** agrupando sentencias con menos conexiones.

El test Nemenyi no detecta diferencias entre **FORCE** y **fujita** respecto a **MINCE** y **mendonca**. Si bien es cierto que el análisis descriptivo obtiene unos peores resultados, el test nos indica que los resultados son equivalentes a nivel de varianza. Es decir, los resultados a nivel de valores son peores, pero el comportamiento respecto a los diferentes modelos de configuración es equivalente. La heurística **FORCE** busca reducir el *span*, que es equivalente a reducir el i-cut [3], por tanto, aunque la heurística es peor, el comportamiento con los diferentes modelos es equivalente, a nivel de diferencias.

Las heurísticas **preorder**, **malik_fanin**, **malik_level**, **fujii** y **narodyska** tiene diferencias con las heurísticas **MINCE** y **mendonca**, el análisis descriptivo también nos muestra que son las heurísticas que tienen los peores resultados, en consecuencia, no son adecuadas para modelos de configuración. La heurística **malik_level**, está dirigida por niveles, lo que no es adecuado habitualmente, en los modelos de configuración los niveles son homogéneos. La heurística **malik_fanin**, añade información propia de circuitos (puertas de entrada), y no se comporta mejor que la heurística **fujita**, que es más genérica. La heurística **narodyska**, es adecuada, para modelos no jerárquicos, ya que busca clústeres organizando estructuras, pero los modelos de configuración tienen una estructura jerárquica habitualmente, por lo que no es adecuada. La heurística **preorden**, es la más sencilla, y tiene un resultado aceptable, pero obviamente únicamente tiene información de la estructura jerárquica, y no recoge información de las *crosstree*.

La heurística **fujii** es la heurística que presenta más diferencias con las del bloque anterior. El análisis de los datos nos indica que es la peor, el uso del *interleaving* no es adecuado para la codificación de modelos de configuración. Ya que, en ellos, sólo existen dos funciones, en el peor de los casos.

3.4 Heurísticas dinámicas

3.4.1 Heurísticas analizadas

En la sección anterior se ha identificado que las heurísticas **MINCE**, **mendonca**, **FORCE** y **fujita**, son heurísticas adecuadas para el análisis de modelos de configuración. Las ordenaciones dinámicas se aplicarán después de estas heurísticas. Se ha incluido también la heurística **fujii** [58] que será tomada como referencia de heurística no adecuada.

El uso de una heurística estática combinada con una dinámica es el procedimiento que utilizan las heurísticas específicas de modelos de configuración Narodytska and Walsh [100] y Mendonca et al. [94].

Las heurísticas dinámicas analizadas son:

- *swapping* (Ishiura et al. [75]) con ventanas de 2 a 4.
- *sifting* (Rudell [114]).
- *sifting* simétrico (Panda et al. [105]).
- *simulated ANNEALING* (Bollig et al. [22])

Se ha descartado la heurística Drechsler, Rolf and Drechsler and Günher [45] dado que, como se ha indicado en el capítulo anterior, no es viable para modelos de configuración con más de 25 características, y en nuestro *benchmark* sólo el modelo de **eCos fs** tiene menos de 25 opciones de configuración. El *sifting* agrupado [105] se analizará en el Capítulo 4 en el que se propone un método de agrupación de variables.

Una heurística dinámica puede aplicar sobre un BDD completo o parcial. Es decir, puede aplicarse al finalizar la construcción del BDD para compactarlo, o a medida que se sintetiza el BDD, consiguiendo así contener su tamaño.

Tras aplicar una heurística estática, examinaremos el efecto de una dinámica de tres modos:

1. Utilizándola sólo cuando el BDD ha sido totalmente sintetizado.
2. Utilizándola exclusivamente a medida que se sintetiza el BDD.

3. Utilizándola durante la síntesis y al final.

En todos los casos se analizarán las siguientes variables explicativas:

1. La heurística de ordenación estática.
2. La heurística de ordenación dinámica individualmente.
3. La combinación de las heurísticas de ordenación dinámicas y estáticas.
4. Cuando se realiza la reordenación dinámica: después, mientras se construye y después de construirlo, o únicamente durante la construcción el BDD.

En este caso se analizará también el tiempo de construcción del BDD, dado que la tecnología que se utiliza en la aplicación de construcción del BDD es la misma (CUDD, compilado con **g++**).

3.4.2 Análisis del número de nodos una vez construido el BDD

En esta sección analizaremos los resultados, construyendo primero el BDD completo, y una vez construido, aplicando la heurística de ordenación dinámica, analizaremos también el tiempo de construcción.

Heurística de ordenación estática

La Tabla 3.6 muestra los resultados del test Nemenyi [101], tomando como variable explicativa el orden estático, y como variable de respuesta el logaritmo en base 2 del nº de nodos del BDD. Los resultados obtenidos verifican los resultados obtenidos con la heurística de control **fujii**, aun aplicando heurísticas de ordenación dinámica después de aplicar la heurística estática sus resultados son diferentes, peores en media, mediana, y desviación típica del logaritmo. Es decir, si partimos de una heurística estática *mala*, las heurísticas de ordenación dinámica producen resultados malos.

La heurística **MINCE** obtiene resultados diferentes con todas las heurísticas, a excepción de **mendonca**, si analizamos los datos descriptivos de mediana y desviación estándar, sus resultados son similares. Las heurísticas **MINCE** y **mendonca** obtiene buenos resultados, si bien es cierto que **MINCE** aventaja en un poco a nivel

de media aritmética. Podemos concluir, que la heurística **MINCE** es la que mejores resultados obtiene, seguida de la heurística **mendonca**.

La heurística **FORCE** y **fujita** no presentan diferencias según indica el test Nemenyi [101], y a nivel descriptivo obtienen unos resultados similares de media, mediana, y desviación típica.

V.p / V.p	**MINCE**	**mendonca**	**FORCE**	**fujita**	**fujii**
MINCE	-	0,952887	0,046115	0,001388	3,73e-14
mendonca	0,952887	-	0,246418	0,017704	6,573e-14
FORCE	0,046115	0,246418	-	0,837556	1,15e-08
fujita	0,001388	0,017704	0,837556	-	4,416e-06
fujii	3,73e-14	6,573e-14	1,15e-08	4,416e-06	-

Tabla 3.6: Valor de p del test Nemenyi del *benchmark*

Heurística de ordenación dinámica

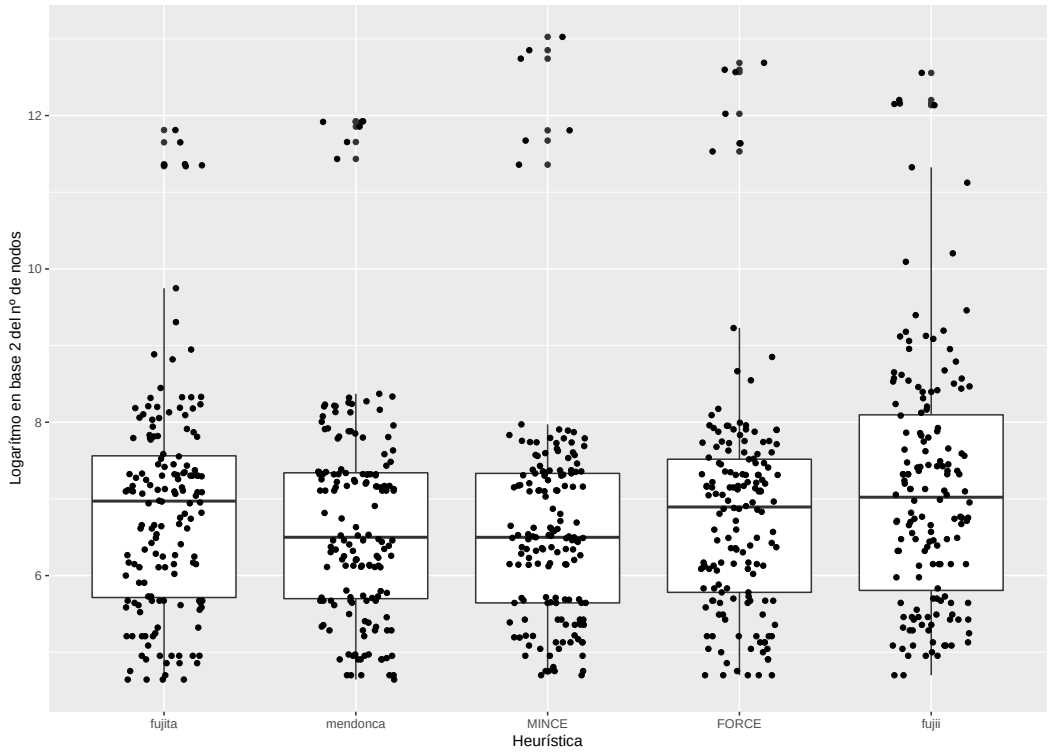
La Tabla 3.8 muestra los resultados de aplicar el test Nemenyi [101] tomando como variable explicativa la heurística de ordenación dinámica, y como variable respuesta el logaritmo en base 2 del número de nodos del BDD.

El test no identifica diferencias en las heurísticas ***sifting*** [114], ***sifting* simétrico** [105] y **ANNEALING** [22]. Si bien es cierto que, a nivel de estadística descriptiva **ANNEALING** [22], presenta un número de nodos menor que el sifting, con una desviación estándar también menor. El análisis descriptivo muestra que el ***sifting* simétrico** es mejor, pero con una diferencia muy pequeña, tan pequeña que el test Nemenyi, le asigna un valor de p igual a 1, lo que indica que son equivalente.

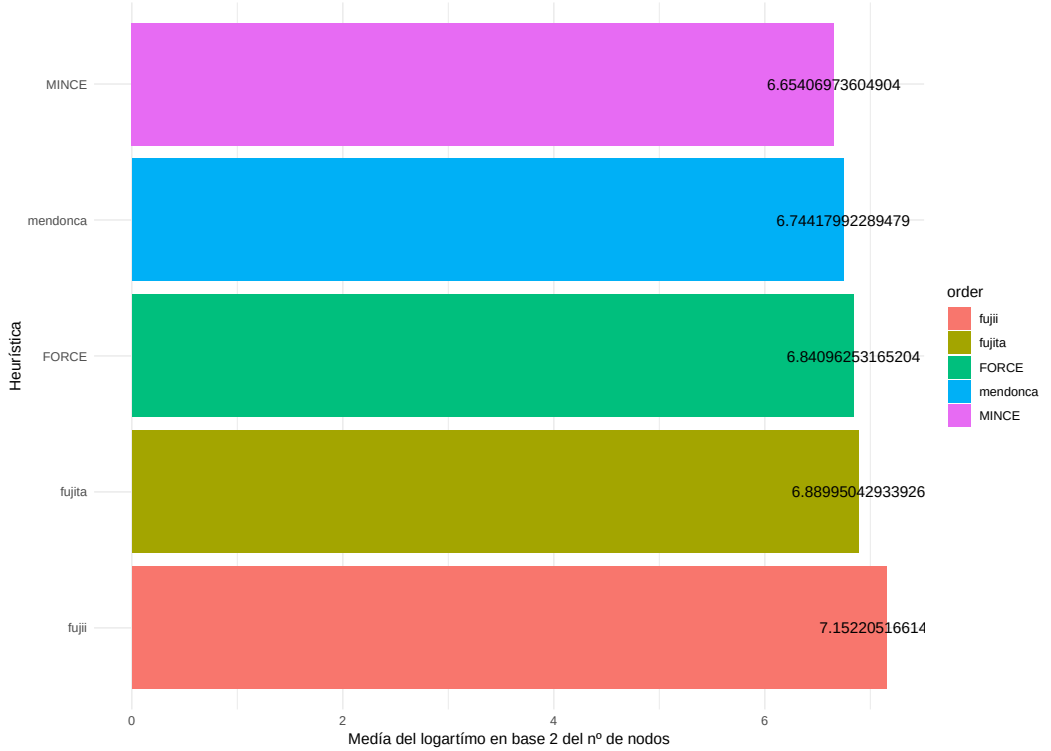
Las heurísticas de intercambio con ventanas [75], de 2 a 3 son similares de acuerdo con lo que indica el test, y sus valores descriptivos que son muy similares. En cambio, la heurística con una ventana de 4 es diferente, a las de 2, y 3 y sus resultados son mejores. Si bien no llegan los resultados del *sifting* simétrico, sí que son una mejora respecto al uso de ventanas de 2 o 3.

Heurística	**Media**(log_2)	**Mediana**(log_2)	**Desv. std.**(log_2)
MINCE	6,65407	6,499121	1,455021
mendonca	6,74418	6,499648	1,446657
FORCE	6,840963	6,894767	1,484673
fujita	6,88995	6,971532	1,452271
fujii	7,152205	7,021984	1,641283

Tabla 3.7: Media, mediana y desviación estándar nº de nodos por heurística estática

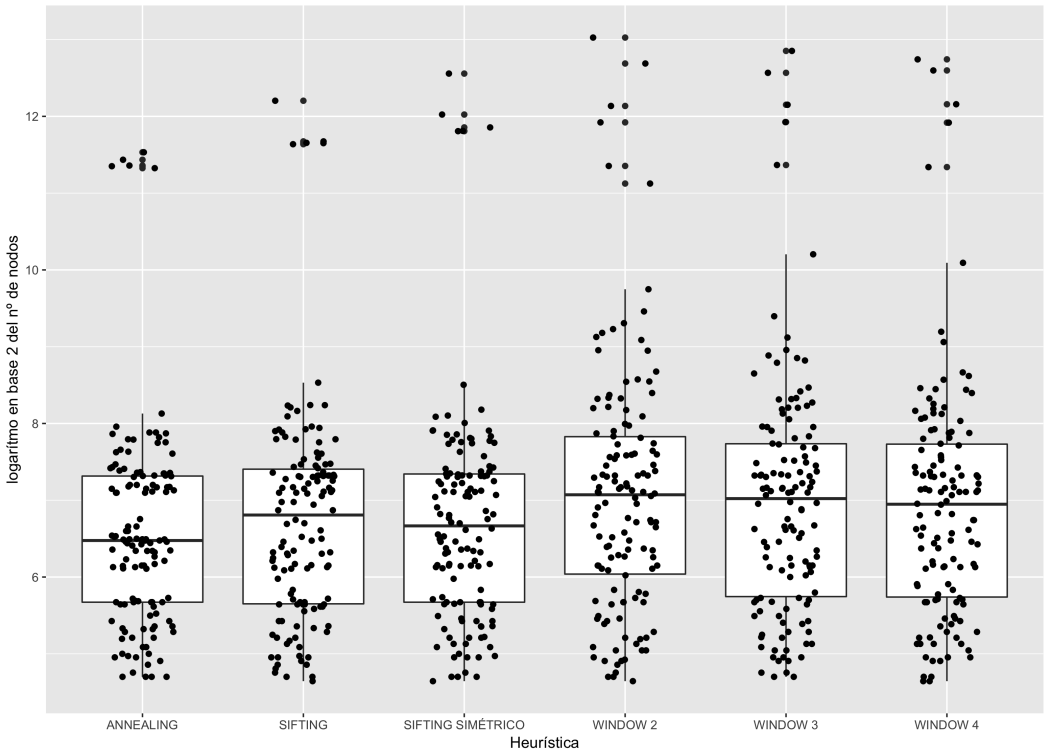


(a) Diagrama de caja del logaritmo en base 2 del número de nodos

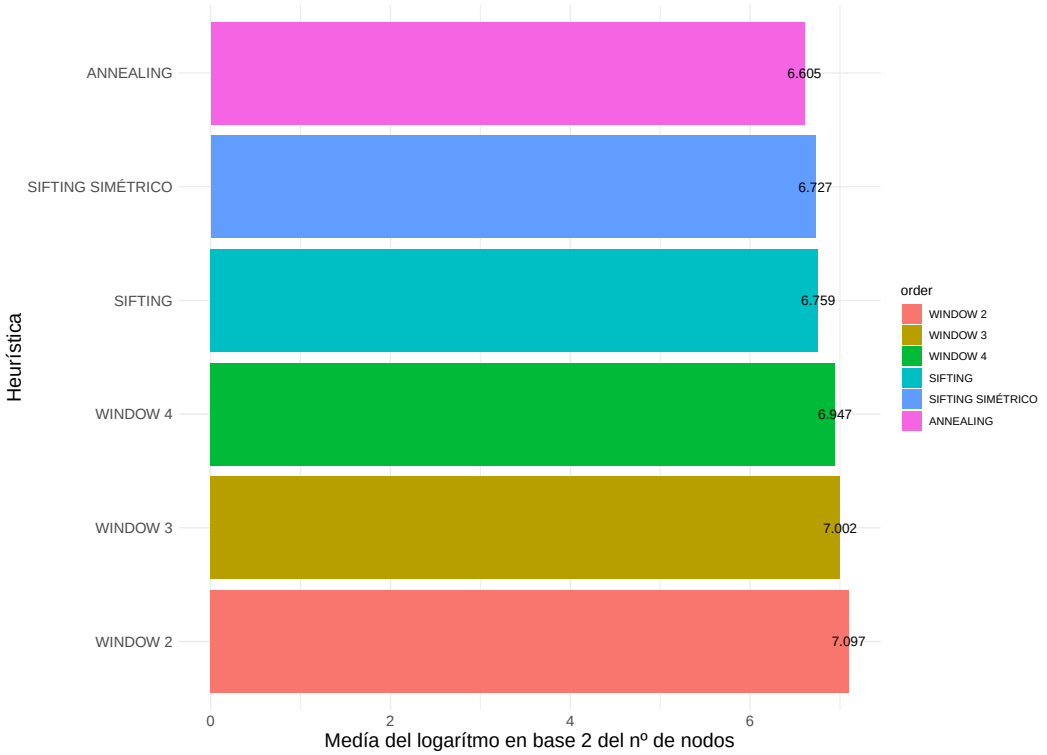


(b) Media del logaritmo en base 2 del número de nodos

Figura 3.5: Análisis gráfico del número de nodos por heurística estática



(a) Diagrama de caja del logaritmo en base 2 del número de nodos



(b) Media del logaritmo en base 2 del número de nodos

Figura 3.6: Análisis gráfico del número de nodos por heurística estática

V.p / V.p	SIFTING	SIFTING SIMÉTRICO	WINDOW 2	WINDOW 3	WINDOW 4	ANNEALING
SIFTING	-	1,000000	4,874e-14	1,527e-09	0,008889	0,368077
SIFTING SIMÉTRICO	1,000000	-	4,863e-14	1,368e-09	0,008390	0,378023
WINDOW 2	4,874e-14	4,863e-14	-	0,213188	1,543e-06	5,984e-14
WINDOW 3	1,527e-09	1,368e-09	0,213188	-	0,026374	6,106e-14
WINDOW 4	0,008889	0,008390	1,543e-06	0,026374	-	1,285e-06
ANNEALING	0,368077	0,378023	5,984e-14	6,106e-14	1,285e-06	-

Tabla 3.8: Valor de p del test Nemenyi del *benchmark*

Heurística	**Media**	**Mediana**	**Desv. std.**	**Media**(log_2)	**Mediana**(log_2)	**Desv. std.**(log_2)
ANNEALING	203,693	89	1,328172	6,605042	6,475733	1,328172
SIFTING SIMÉTRICO	271,4804	101,5909	1,439708	6,727412	6,666222	1,439708
SIFTING	249,7748	112,1364	1,42424	6,758785	6,807752	1,42424
WINDOW 4	327,6238	123,5	1,556228	6,946617	6,948355	1,556228
WINDOW 3	340,8266	130	1,5809	7,002366	7,021685	1,5809
WINDOW 2	376,0301	134,5	1,638735	7,097419	7,071373	1,638735

Tabla 3.9: Media, mediana y desviación estándar nº de nodos por heurística estática

Combinación heurística de ordenación estática y dinámica

La combinación de las heurísticas estáticas y dinámicas genera 30 posibles combinaciones, y debemos realizar comparaciones de 2 elementos, sin repetición lo que implica realizar 870 comparaciones, lo que hace que no sea viable realizar un análisis con el test Nemenyi [101], ni mediante un análisis gráfico. Por lo que debemos realizar otro tipo de análisis.

El objetivo del análisis será identificar que heurísticas y que combinaciones son mejores, es por lo que se ha decido optar por un modelo de regresión lineal, tomando como variables explicativas las heurísticas estáticas y dinámicas, y como variable resultante el número de nodos. Un análisis de regresión lineal nos dará una función con la siguiente forma:

$$y = \alpha + \sum_{i=1}^{n} x_i b_i$$

Donde α será el parámetro de intercepción o constante, x_i cada una de las variables explicativas, y b_i, el coeficiente multiplicador de la variable explicativa.

Dado, que nuestras variables explicativas son valores no numéricos, se sustituye cada una de ellos por una variable que toma valor 1 cuando el suceso es de esa variable y 0 en caso contrario. En nuestro caso tendremos una variable para cada heurística estática, y otra para cada heurística dinámica, lo que hará un total de 9

variables explicativas.

El modelo de datos que estamos trabajando no es paramétrico. En consecuencia, hay que utilizar regresión no paramétrica. Hemos utilizado el modelo Jaeckel [77], que se analiza de igual modo al paramétrico: para cada uno de los coeficientes calcula el valor p, que, en caso de ser menor al punto de corte fijado, $0,05$ en nuestro caso indica que la variable explicativa es relevante en el modelo.

El resultado obtenido para nuestros datos es:

Transcripción 12

```
Coefficients:
Estimate Std. Error t.value   p.value
(Intercept)                                6.545294   0.148885 43.9620 < 2.2e-16 ***
datos.dinamicos$orderfujii                 0.280085   0.140914  1.9876  0.047206 *
datos.dinamicos$orderfujita                0.071048   0.140914  0.5042  0.614269
datos.dinamicos$ordermendonca             -0.083305   0.140914 -0.5912  0.554576
datos.dinamicos$orderMINCE                -0.199225   0.140914 -1.4138  0.157821
datos.dinamicos$reorderSIFTING             0.155874   0.154363  1.0098  0.312915
datos.dinamicos$reorderSIFTING SIMÉTRICO   0.115226   0.154363  0.7465  0.455619
datos.dinamicos$reorderWINDOW 2            0.450586   0.154363  2.9190  0.003614 **
datos.dinamicos$reorderWINDOW 3            0.374128   0.154363  2.4237  0.015593 *
datos.dinamicos$reorderWINDOW 4            0.326417   0.154363  2.1146  0.034785 *

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Overall Wald Test: 25.60177 p-value: 0
```

El análisis del valor de p, nos indica que sólo las heurísticas Fujii et al. [56] y la Ishiura et al. [75], tienen relevancia. Este dato, cuadra con los análisis anteriores, la heurística de control Fujii et al. es la que es diferente en las heurísticas estáticas, y en las dinámicas la Ishiura et al..

Los coeficientes en este caso nos indican también que peso tiene la heurística en el número de nodos del BDD. Las heurísticas **fujita** [57] y **fujii** [56] tienen coeficientes positivos, mientras que las heurísticas, que habíamos identificado como las mejores tiene coeficientes negativos **MINCE** [3] y **mendonca** [94].

Si analizamos las heurísticas dinámicas se reafirma lo que habíamos observado anteriormente, las heurísticas ***sifting*** [114], ***sifting simétrico*** [105], y **ANNEALING** [22], tienen coeficientes menores a la heurística **window** [75].

Del análisis también se desprende que no existe mucha diferencia entre las heurísticas de ***sifting*** [114] y ***sifting simétrico*** [105], ya que los factores multiplicadores son muy similares.

Heurística	Media (s)	Mediana(s)	Desv. std.
WINDOW 3	0,4254601	0,004000000	2,316692
WINDOW 2	0,4283329	0,003454545	2,329419
WINDOW 4	0,4308951	0,004727273	2,336067
SIFTING	0,4642490	0,006181818	2,47193
SIFTING SIMÉTRICO	0,4935916	0,00800000	2,61102
ANNEALING	20,380870	0,91236360	91,96322

Tabla 3.10: Media, mediana y desviación estándar del tiempo de construcción del BDD por heurística estática

Análisis de tiempo de construcción del BDD

En el análisis del tiempo de construcción del BDD, tendremos en cuenta la heurística de ordenación estática y dinámica. En consecuencia, realizaremos el análisis mediante un modelo de regresión.

Transcripción 13

```
Call:
rfit.default(formula = datos.dinamicos$total.time ~ datos.dinamicos$order +
datos.dinamicos$reorder)

Coefficients:
Estimate  Std. Error   t.value p.value
(Intercept)                                 0.87626160  0.00293760  298.2920  <2e-16 ***
datos.dinamicos$orderfujii                  0.00170764  0.00307092    0.5561  0.5783
datos.dinamicos$orderfujita                -0.00041757  0.00307092   -0.1360  0.8919
datos.dinamicos$ordermendonca              -0.00069572  0.00307092   -0.2266  0.8208
datos.dinamicos$orderMINCE                 -0.00051123  0.00307092   -0.1665  0.8678
datos.dinamicos$reorderSIFTING             -0.86994565  0.00336402 -258.6029  <2e-16 ***
datos.dinamicos$reorderSIFTING SIMÉTRICO -0.86876245  0.00336402 -258.2512  <2e-16 ***
datos.dinamicos$reorderWINDOW 2            -0.87259489  0.00336402 -259.3904  <2e-16 ***
datos.dinamicos$reorderWINDOW 3            -0.87241002  0.00336402 -259.3355  <2e-16 ***
datos.dinamicos$reorderWINDOW 4            -0.87169486  0.00336402 -259.1229  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Overall Wald Test: 111753.1 p-value: 0
```

El análisis nos muestra que las heurísticas estáticas no afectan al tiempo de construcción del BDD, únicamente las heurísticas de ordenación dinámicas. El termino independiente, es debido a la heurística de ordenación dinámica **ANNEALING** [22], que es la más lenta como muestra la Figura 3.7 y la Tabla 3.10, el reste de heurísticas tienen factores similares que restan tiempo.

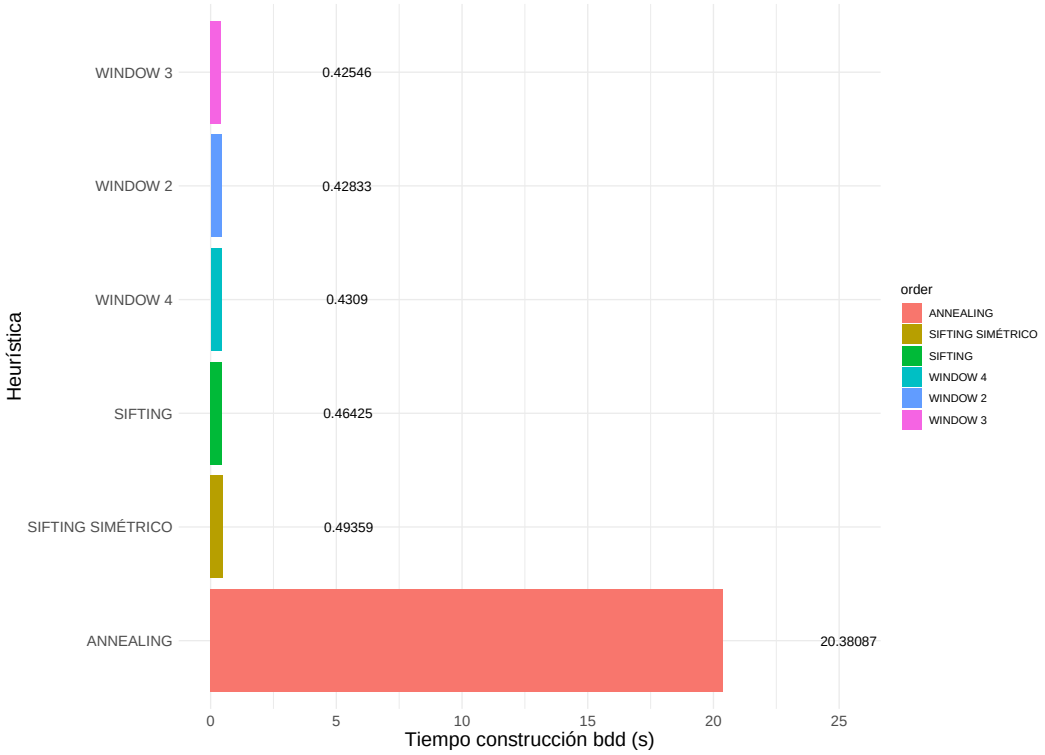


Figura 3.7: Promedio del tiempo de construcción del BDD por heurística (s)

3.4.3 Conclusiones heurísticas dinámicas

Utilizar una heurística estática aceptable (**FORCE** [4], **MINCE** [3], **mendonca** [94]) combinada con las dinámicas Rudell [114] o **ANNEALING** [22], es la opción óptima para la reducción nodos de los BDD's con modelos de configuración.

Hay que tener en cuenta que la heurística **ANNEALING** [22] es la que produce un resultado mejor a nivel del número de nodos, pero en contraprestación, es la más lenta. Para modelos grandes, no compensa la mejora que se obtiene a nivel de número de nodos con el tiempo que requiere, por lo que es preferible utilizar heurísticas de ***sifting*** ([114] y [105]) o de ventana [75] con $k = 4$.

3.4.4 Análisis del momento de aplicación de la heurística

En esta sección analizaremos la influencia que tiene el momento de aplicación de las heurísticas de reordenación: al final de la construcción, durante la construcción o durante y al final de la construcción, analizando tanto el tiempo como el nº de

nodos.

Este análisis debe realizarse teniendo en cuenta todas las variables explicativas del modelo, para aplicarlo en condiciones equivalentes, realizaremos un modelo de regresión lineal no paramétrico de Jaeckel [77].

Análisis del número de nodos

La aplicación de la ordenación después está incluida en la intercepción, y tiene valor de p menor que $0,05$, en consecuencia, es una variable que afecta al modelo, al igual que aplicar la ordenación durante la construcción, influyó en el modelo. No es así, con la aplicación de la ordenación durante y después. Esto es debido a, como se verá a continuación, el número de nodos que se obtiene es similar al que se obtiene en la aplicación de la heurística después de la construcción del BDD.

Transcripción 14

```
Coefficients:
Estimate  Std. Error t.value   p.value
(Intercept)                            6.5657e+00  9.0884e-02 72.2427 < 2.2e-16 ***
datos.total$cuandoDurante              3.2434e-01  6.0926e-02  5.3235 1.120e-07 ***
datos.total$cuandoDurante y después    1.9005e-05  6.0926e-02  0.0003 0.9997511
datos.total$orderfujii                 4.0625e-01  7.8655e-02  5.1650 2.617e-07 ***
datos.total$orderfujita                8.4821e-02  7.8655e-02  1.0784 0.2809747
datos.total$ordermendonca             -1.0695e-01  7.8655e-02 -1.3597 0.1740621
datos.total$orderMINCE                -2.5647e-01  7.8655e-02 -3.2607 0.0011281 **
datos.total$reorderSIFTING             1.0920e-01  8.6162e-02  1.2674 0.2051478
datos.total$reorderSIFTING SIMÉTRICO   7.9774e-02  8.6162e-02  0.9259 0.3546152
datos.total$reorderWINDOW 2            3.0048e-01  8.6162e-02  3.4874 0.0004971 ***
datos.total$reorderWINDOW 3            2.4860e-01  8.6162e-02  2.8853 0.0039482 **
datos.total$reorderWINDOW 4            2.1585e-01  8.6162e-02  2.5051 0.0123114 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Overall Wald Test: 134.9674 p-value: 0
```

El análisis de las heurísticas de ordenación, y reordenación tiene resultados similares. Los que realmente influyen a nivel de heurística estática, son las heurísticas **fujii** [56] y **MINCE** [3], que son respectivamente los que obtienen peor y mejor resultado.

A nivel de heurística dinámica, tomando como base que la heurística de **ANNEALING** [22] forma parte de la intercepción, se aprecia relevancia con las heurísticas de **ventana** [75], que hacen que el número de nodos aumente.

La Figura 3.8, nos muestra que el promedio de los resultados obtenidos en la aplicación después, y durante y después son similares, el peor resultado se obtiene

en sólo aplicar la heurística de ordenación dinámica durante la construcción del BDD.

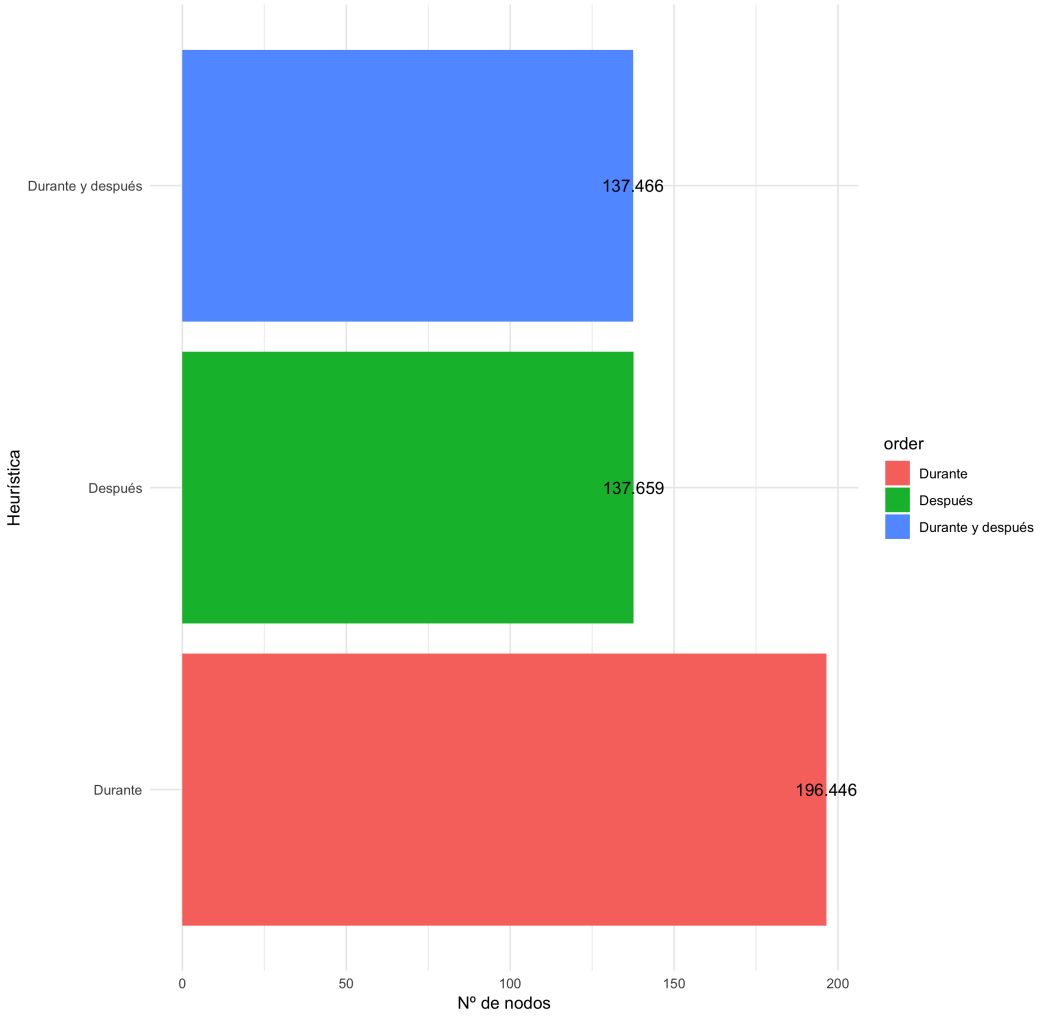


Figura 3.8: Promedio del nº de nodos dependiendo de cómo se aplica la heurística dinámica

Análisis del tiempo de construcción del BDD

El análisis del tiempo de construcción del BDD tiene influencia cuando se aplica la heurística de ordenación, dado que es relevante para los tres casos, con todos con valores de p menores de $0,05$. Si realizamos el análisis de datos descriptivos vemos

Sistemas comparados	Proporción Geekbench 4
Intel Core i5-5575R / Intel Core i5-3230M	1,7924
Intel Core i5-5575R / Intel Xeon E5-2660	1,0648
Intel Xeon E5-2660 / Intel Core i5-3230M	1,6833

Tabla 3.11: Comparación de puntuación entre sistemas

que el tiempo de construcción de la aplicación del algoritmo antes y después de la construcción del BDD es el doble a los otros dos, que son similares.

Transcripción 15

```
                                    Estimate  Std. Error   t.value   p.value
(Intercept)                       0.53384149  0.00093646  570.0629 < 2.2e-16 ***
datos.total$cuandoDurante        -0.00383914  0.00067292   -5.7052 1.316e-08 ***
datos.total$cuandoDurante y después  0.00160804  0.00067292    2.3896   0.01695 *
datos.total$orderfujii            0.00065243  0.00086874    0.7510   0.45273
datos.total$orderfujita          -0.00067425  0.00086874   -0.7761   0.43776
datos.total$ordermendonca        -0.00078145  0.00086874   -0.8995   0.36847
datos.total$orderMINCE           -0.00036177  0.00086874   -0.4164   0.67714
datos.total$reorderSIFTING       -0.52632721  0.00095166 -553.0645 < 2.2e-16 ***
datos.total$reorderSIFTING SIMÉTRICO -0.52564058  0.00095166 -552.3430 < 2.2e-16 ***
datos.total$reorderWINDOW 2      -0.52887163  0.00095166 -555.7382 < 2.2e-16 ***
datos.total$reorderWINDOW 3      -0.52857244  0.00095166 -555.4238 < 2.2e-16 ***
datos.total$reorderWINDOW 4      -0.52784098  0.00095166 -554.6552 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Overall Wald Test: 512069.5 p-value: 0
```

Análisis del tiempo en diferentes sistemas

El objetivo de esta sección es realizar un análisis de tiempo en diferentes sistemas con el objetivo de identificar la influencia que tiene la velocidad del sistema en la construcción de los BDD's. Para ello, se ha realizado el siguiente experimento: calcular el tiempo promedio de construcción, y reordenación de un BDD para tres sistemas diferentes con una puntuación **Geekbench 4** [83] para operaciones de un solo core de 2264 (Intel Core i5-3230M) , 3811 (Intel Xeon E5-2660) y 4058 (Intel Core i5-5575R)[1]. La Tabla 3.11 muestra la relación entre las diferentes puntuaciones de los sistemas.

La Tabla 3.12, muestra los resultados obtenidos y la Tabla 3.13 el ratio comparativo de ambos sistemas, como vemos las proporciones son similares a la proporción del *benchmark*, dato que corrobora conclusión.

[1]Benchmark disponibles https://github.com/robcbean/thesis_data/tree/master/system_comparative/bench

Procesador	Construcción (s)	Reordenación (s)	Suma (s)
Intel Core i5-3230M	241,230	26,6066	267,8374
Intel Core i5-5575R	132,649	15,0211	147,6701
Intel Xeon E5-2660	124,353	17,1021	141,4554

Tabla 3.12: Tiempo de construcción y reordenación de BDD's con sistemas diferentes

Sistemas comparados	Construcción	Reordenación	Total
Intel Core i5-3230M / Intel Core i5-5575R	1,81856	1,77128	1,81375
Intel Xeon E5-2660 / Intel Core i5-5575R	0,93746	1,13854	0,95791
Intel Core i5-3230M / Intel Xeon E5-2660	1,93988	1,55575	1,89344

Tabla 3.13: Comparación de tiempos entre sistemas

3.5 Conclusiones finales

La aplicación de una heurística dinámica, a un modelo construido con una heurística estática no adecuada, como por ejemplo **fujii** [56], no hace que esta se acerque al resultado que se obtiene con una heurística estática adecuada, como **MINCE**.

La combinación de las heurísticas estáticas **MINCE** [3] o **mendonca** [94], junto con la heurística dinámica de ***sifting*** [114], obtiene los mejores resultados para modelos de configuración.

El menor número de nodos se obtiene aplicando heurísticas dinámicas durante y al final de las síntesis del BDD. Si bien es cierto, que no se aprecian diferencias estadísticas significativas con el valor, una vez sintetizado el BDD. En cambio, a nivel de tiempo esta es la peor opción.

Las heurísticas estáticas analizadas dan el orden para el modelo completo. Pero, no garantizan el mejor orden durante la síntesis del BDD. Por eso, es conveniente combinar una heurística estática con una dinámica durante la síntesis del BDD. Así, evitamos que su tamaño, aumente en exceso durante su construcción.

Una vez sintetizado el BDD, aplicamos de nuevo la heurística dinámica, para así reducir al máximo el BDD. Dado, que el objetivo es construir el BDD con el menor número de nodos para que reducir el tiempo de las operaciones con BDD's, que es proporcional al tamaño del mismo.

Para el uso de las heurísticas de reordenación de BDD's se recomienda disponer de un procesador y memoria rápidas, ya que el proceso de síntesis, y reordenación, es intensivo a nivel de procesador y de memoria, tal y como se ha verificado en las secciones anteriores.

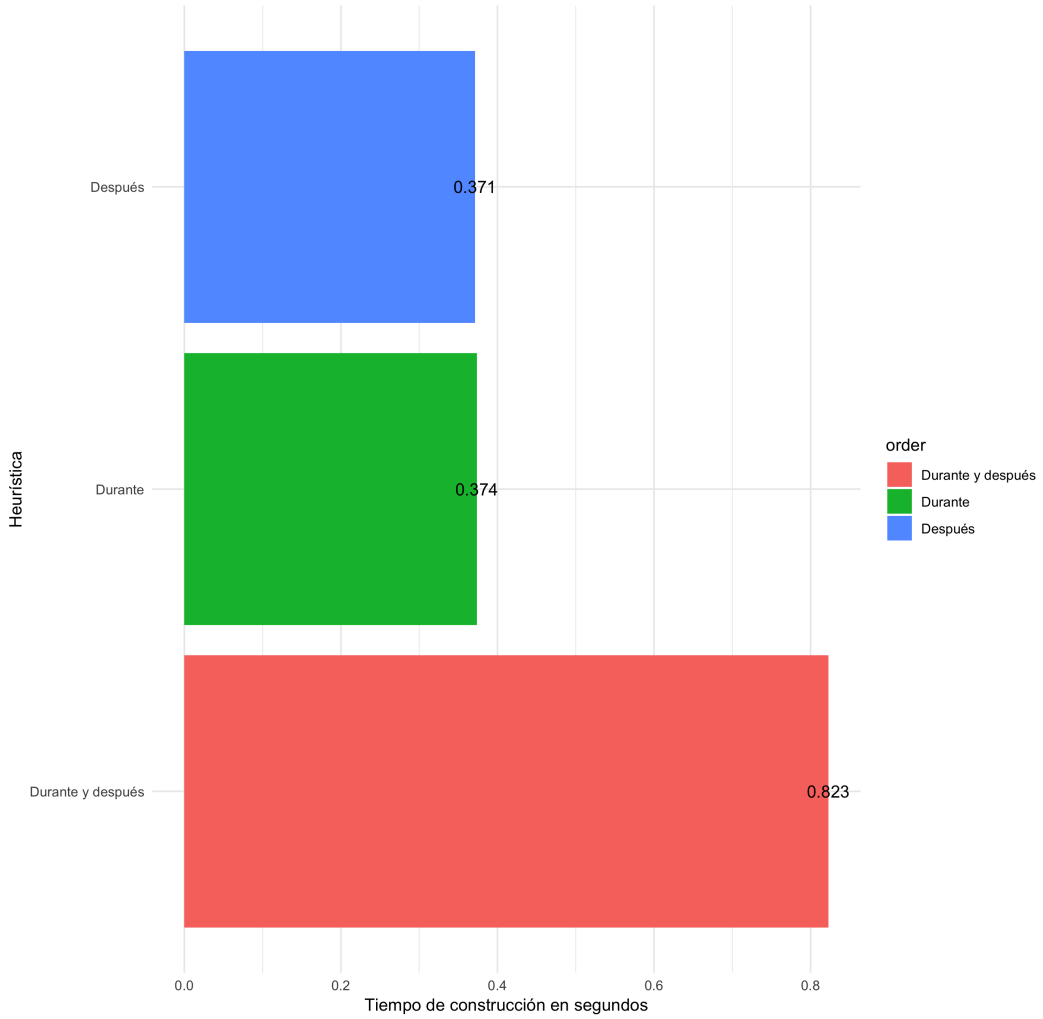


Figura 3.9: Promedio del tiempo de construcción del BDD dependiendo de cómo se aplica la heurística dinámica

No basta tener buen ingenio; lo principal es aplicarlo bien.

René Descartes

CAPÍTULO

4

Heurísticas de ordenación desarrolladas

4.1 Introducción

Las heurísticas de ordenación de variables en la codificación de modelos de configuración utilizando BDD's es un campo de investigación con poca producción, limitada a **narodyska** [100], y **mendonca** [94]. En este capítulo presentaremos los resultados y las heurísticas desarrolladas en esta tesis.

El primer resultado es que el *span* está relacionado con el tamaño del BDD. Posteriormente presentaremos, una heurística que utiliza los resultados de simetría, y de agrupación de variables del Capítulo 3. A continuación, se proponen, algoritmos de cálculo de características *core*/*dead* [107], e *impact*/*exclusion set* [67] con BDD's de modo eficiente, con el modelo completo o descompuesto. Finalmente se describe, una heurística que reduce el problema, eliminando características *core* y *dead*.

Para cada uno de los resultados y heurísticas desarrolladas se presentarán los fundamentos teóricos, se detallarán los algoritmos, se mostrarán y analizarán los resultados experimentales, y finalmente se expondrán las conclusiones.

4.2 *Span*

4.2.1 Objetivo

En esta sección se demostrará que el tamaño de los BDD's es proporcional al *span*. Para ello, primero se presentarán los fundamentos teóricos, y posteriormente un análisis estadístico de la correspondencia entre el *span* y el tamaño del BDD.

4.2.2 Fundamentos teóricos

La Figura 4.1 muestra la relación entre el logaritmo en base 2 del *span* y el logaritmo en base 2 del número de nodos del BDD, para los 29 modelos de configuración del *benchmark*, aplicando las heurísticas estáticas analizadas en el Capítulo 3.

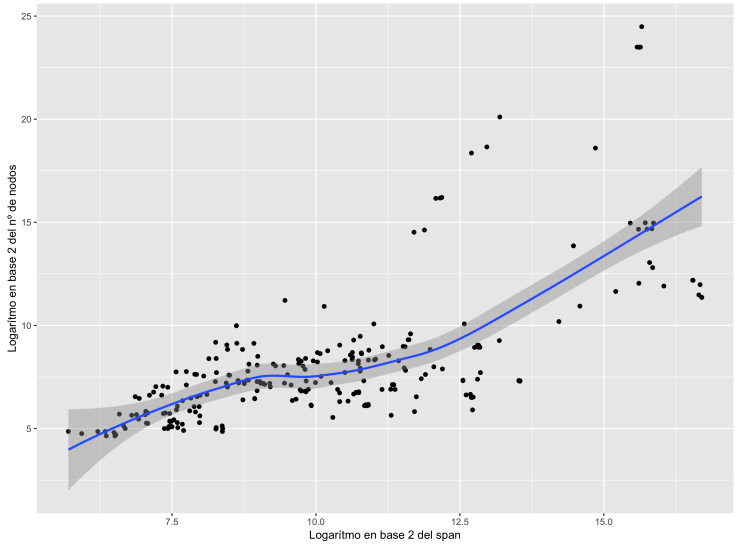


Figura 4.1: Logaritmo del *span* versus nº de nodos del BDD

Como muestra la figura, el número de nodos de los BDD's para los diferentes modelos de configuración parece aumentar a medida que aumenta el *span*. Este resultado es compatible con los resultados del capítulo anterior en que se había identificado a la heurística **MINCE** [3], basada en la minimización del *span*, como la mejor heurística. El *span* combina información acerca de la complejidad del problema (nº de variables y clausulas), y de la ordenación de las variables.

Sin embargo, en el capítulo anterior vimos que **MINCE** [3] no siempre obtiene

el mejor resultado, ver Tabla 4.1. Ello nos lleva a dudar sobre la eficacia del *span* y, por ello, hemos realizado la validación experimental que se describe a continuación.

Modelo de configuración	Nº de características	Mejor heurística
LinuxKernel_fs	339	MINCE
Electronic Shopping	290	MINCE
LinuxKernel_crypto	268	MINCE
BankingSoftware	176	MINCE
eCos_isoinfra	155	fujita
eCos_io	130	MINCE
LinuxKernel_lib	121	mendonca
LinuxKernel_arch	121	MINCE
Dell_XML	118	fujita
SmartTV	96	mendonca
Model_Transformation	88	mendonca
eCos_net	82	mendonca
eCos_language	82	MINCE
LinuxKernel_security	73	MINCE
LinuxKernel_mm	70	MINCE
HIS	67	MINCE
eCos_redboot	63	FORCE
LinuxKernel_block	60	mendonca
eCos_kernel	57	mendonca
eCos_services	53	mendonca
Documentation_Generation	44	MINCE
Thread	44	mendonca
Web_Portal	43	MINCE
Assess4me	40	MINCE
Smart Home v2	38	malik_level
Graph	30	MINCE
eCos_compat	28	fujita
eCos_infra	27	fujita
eCos_fs	22	fujita

Tabla 4.1: Mejor heurística para el *benchmark*

4.2.3 Procedimiento de análisis

El objetivo del análisis es evaluar si existe efectivamente una relación entre el *span* y el tamaño de un BDD. El modelo de configuración influye en la complejidad del problema, y por tanto debe formar parte del modelo de análisis, y se ha definido, el siguiente modelo de regresión lineal:

$$log_2(\text{número\_nodos}) = \alpha \cdot log_2(span) \cdot \text{modelo\_configuracion}$$

En el modelo se ha eliminado el término constante, dado que con *span* 0 el tamaño del BDD debe ser 0. Para verificar el modelo, se ha utilizado el modelo lineal no paramétrico de Jaeckel [77]. El resultado de este modelo nos da para cada variable explicativa (la combinación del *span* junto con el modelo de configuración), un valor de p. Si el valor de p está por debajo del valor de referencia, 0,05 en nuestro caso, indicará que las variables explicativas son estadísticamente significativas. El modelo nos estimará también el factor multiplicador α, que nos dará la relación entre el *span* y el número de nodos del BDD para cada modelo de configuración.

4.2.4 Resultados experimentales

La Tabla 4.2 muestra el resultado del modelo de regresión lineal de Jaeckel [77], en ella observamos que, para todas las variables explicativas, el valor de p es menor a $0,05$, lo que indica que es relevante. Por tanto, el *span* está relacionado con el número de todos para todos los modelos, reafirmando el resultado que este influye en el tamaño del BDD.

El coeficiente multiplicador α es mayor que 0 para todos los modelos, lo que indica que cuando aumenta el *span*, entonces aumenta el número de nodos de un modelo de configuración.

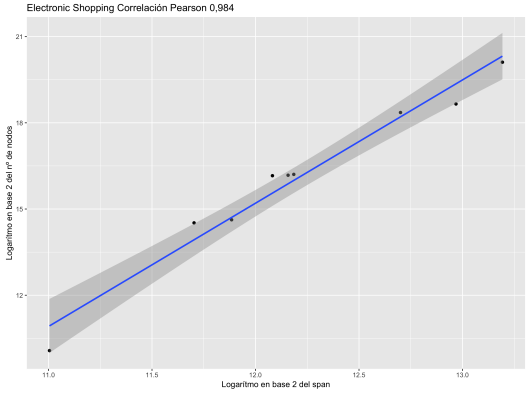
Si analizamos, por ejemplo, el modelo de configuración **Electronic Shopping** o **LinuxKernel_crypto**, vemos claramente esa tendencia tal y como muestra las Figuras 4.2(a) y 4.2(b). En ambos casos además la correlación de Pearson es superior 0,80.

4.2.5 Conclusiones

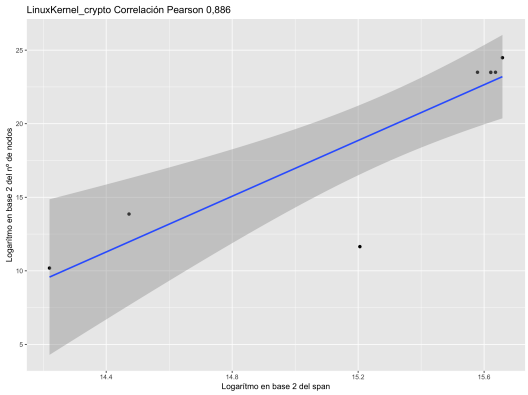
Reducir el *span* disminuye el tamaño del BDD para modelos de configuración, lo que reafirma los resultados obtenidos en el análisis del Capítulo 3, en el que la heurística **MINCE** [3] fue la que mejores resultados obtuvo, y la heurística **FORCE** [4], también obtuvo buenos resultados. Por tanto, podemos utilizar el *span* como medida de la mejora que obtiene una heurística, sin la necesidad de construir el BDD, y descartar/seleccionar las peores/mejores heurísticas.

Modelo de configuración	Coeficiente multiplicadorα	Desviación estándar	Valor de t	Valor de p
Assess4me:span	0,80339	0,01376	58,37023	2,8194E-140
BankingSoftware:span	0,70484	0,00856	82,34632	3,3878E-173
Dell_XML:span	0,72918	0,0062	117,65994	3,7594E-208
Documentation_Generation:span	1,04871	0,01178	89,05492	8,1637E-181
eCos_compat:span	0,68101	0,01387	49,10109	3,1980E-124
eCos_fs:span	0,76461	0,01613	47,41609	4,9338E-121
eCos_infra:span	0,6113	0,0125	48,91738	7,0482E-124
eCos_io:span	0,80415	0,00967	83,18056	3,5607E-174
eCos_isoinfra:span	0,54431	0,00763	71,3587	2,2632E-159
eCos_kernel:span	0,63281	0,0097	65,23861	8,4420E-151
eCos_language:span	0,85593	0,01037	82,50957	2,1764E-173
eCos_net:span	0,75866	0,00945	80,24296	1,0925E-170
eCos_redboot:span	0,56848	0,00952	59,71048	2,0591E-142
eCos_services:span	0,792	0,01137	69,66056	4,6146E-157
Electronic Shopping:span	1,3336	0,00831	160,47144	6,6670E-239
Graph:span	0,78212	0,01346	58,1053	7,5435E-140
HIS:span	0,87854	0,01166	75,36765	1,2479E-164
LinuxKernel_arch:span	0,51702	0,00813	63,62536	2,0257E-148
LinuxKernel_block:span	0,6907	0,01046	66,022	6,1608E-152
LinuxKernel_crypto:span	1,497	0,00664	225,47002	8,4508E-273
LinuxKernel_fs:span	0,93811	0,00656	142,94718	2,0409E-227
LinuxKernel_lib:span	0,69072	0,00806	85,71851	4,2713E-177
LinuxKernel_mm:span	0,60935	0,00915	66,58935	9,4167E-153
LinuxKernel_security:span	0,79285	0,00898	88,30844	5,3996E-180
Model_Transformation:span	0,84288	0,011	76,60324	3,3676E-166
Smart Home v2:span	0,7078	0,01347	52,5636	1,6911E-130
SmartTV:span	0,76026	0,01037	73,30962	5,7907E-162
Thread:span	0,94794	0,01385	68,42555	2,3773E-155
Web_Portal:span	0,99944	0,01256	79,54929	7,5706E-170

Tabla 4.2: Modelo de regresión lineal del modelo y *span* $log2total\ modelo : span-1$



(a) Electronic Shopping



(b) LinuxKernel_crypto

Figura 4.2: Logaritmo del *span* versus nº de nodos del BDD por modelo

4.3 Agrupación de posibles variables simétricas

4.3.1 Objetivo

En esta sección se presentará una nueva heurística basada en simetría de variables. Para ello, primero se demostrará la complejidad del cálculo de las variables simétricas, en las funciones Booleanas, y en los BDD's. Posteriormente, se definirá el concepto de posibles variables simétricas, que se utiliza para realizar un algoritmo de ordenación agrupando variables, conjuntamente con una heurística de ordenación dinámica, para de este modo obtener una mejora de los algoritmos de *sifting* [114] y *sifting* simétrico [105].

4.3.2 Fundamentos teóricos

El análisis de las heurísticas dinámicas realizado en el Capítulo 3, el ***sifting* simétrico** reduce el número de nodos respecto al no simétrico, si bien es cierto que no es mucho. Pero, como indican Panda et al. [105] sus resultados mejoran cuando las variables simétricas aumentan. Esta heurística reduce el nº de nodos del BDD, si las variables simétricas están juntas [105].

El algoritmo de ***sifting* simétrico** no prueba todas las posibles variables simétricas. Como mucho sólo prueba un subconjunto de posibles variables simétricas $m \cdot (m-1)^2$, siendo m el número de variables.

Los conjuntos de posibles variables simétricas de un modelo de configuración, P_s, son todas las combinaciones sin repetición de m variables, cogidas de 2 a m variables.

$$P_s = \sum_{i=2}^{m} C_{m,i}$$

$$C_{m,n} = \frac{m!}{n! \cdot (m-n)!}$$

La Tabla 4.3 muestra el número de pruebas que hay que realizar para comprobar los pares, tríos, y cuartetos, junto con su suma. Es inviable su comprobación. En consecuencia, cuando aumenta el número de variables, el posible número de variables simétricas que la heurística de ***sifting* dinámico** es menor en términos relativos.

Nº de variables	Pares	Tríos	Cuartetos	Total
50	1225	19600	230300	251125
100	4950	161700	3921225	4087875
150	11175	551300	20260275	20822750
200	19900	1313400	64684950	66018250
250	31125	2573000	158882750	161486875
300	44850	4455100	330791175	335291125
350	61075	7084700	614597725	621743500
400	79800	10586800	1050739900	1061406500
450	101025	15086400	1685905200	1701092625
500	124750	20708500	2573031125	2593864375

Tabla 4.3: Test de variables simétricas

El decrecimiento del número de pares simétricos continuos versus el número total de opciones de configuración se puede observar en la Figura 4.3, que muestra el número de opciones de configuración y el porcentaje de los pares de variables respecto al total de posibles pares de variables. El porcentaje disminuye respecto al número de nodos. En cambio, el porcentaje de pares simétricos existentes respecto al total no decrece como se observa en la Figura 4.4.

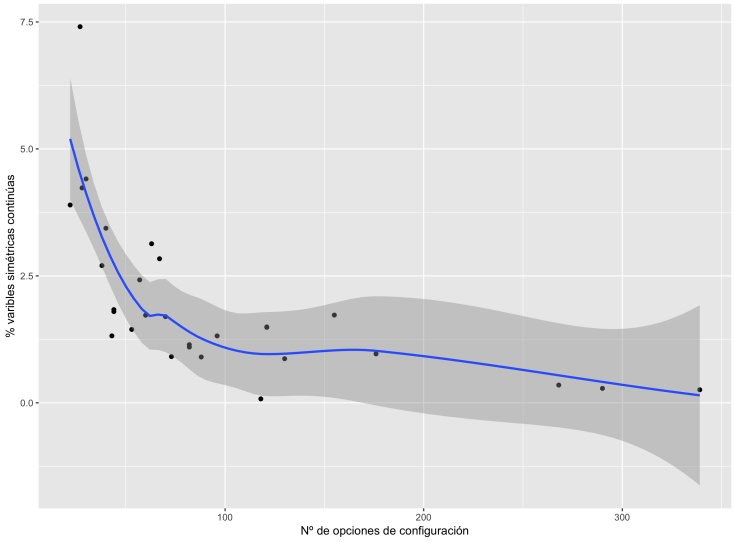


Figura 4.3: Pares continúos simétricos

Es posible identificar la simetría de modo eficiente con un BDD del modelo de configuración completo, una vez construido éste, utilizando las heurísticas de Scholl et al. [121] y Mishchenko [97]. Pero, no podemos utilizar estos resultados para que el BDD tenga un tamaño menor antes de la construcción de este.

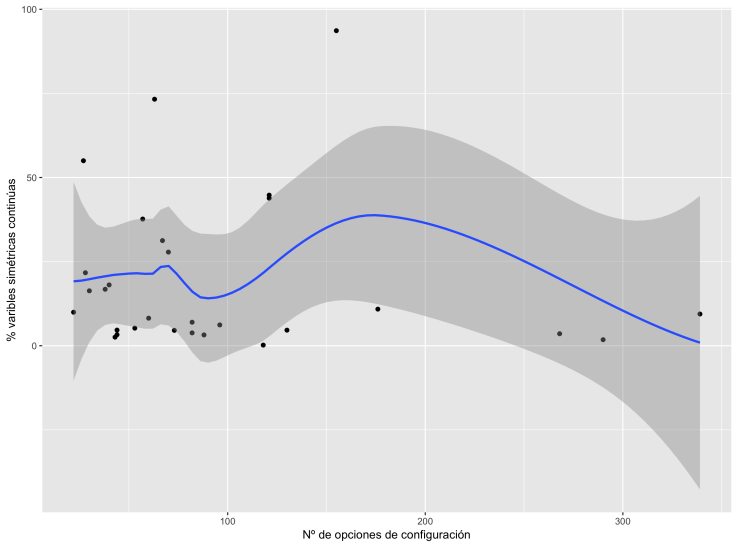


Figura 4.4: Pares simétricos

Identificar las variables simétricas y agruparlas reduce en la mayoría de los casos el nº de nodos del BDD [104], pero identificarlas es costoso, como se ha indicado anteriormente, en este trabajo se propone una nueva heurística para la identificación de posibles variables simétricas, para modelos de configuración. Esta heurística, se basa en la codificación de los modelos de configuración a lógica proposicional, identificando en la propia fórmula, aquellas variables que son candidatas a ser simétricas, y agrupándolas con la heurística de ordenación dinámica de Panda et al. [104]. Para así, obtener un mejor orden.

4.3.3 Heurística desarrollada

Simetría en los modelos de configuración

De acuerdo con la definición de simetría dada en el Capítulo 2 (Estado del arte), las relaciones de diagramas de características *Mandatory*, *Or* y *Alternative* son simétricas.

La conversión de la relación p *Mandatory* h es $(\neg h \vee p) \wedge (\neg p \vee h)$, por lo que podemos intercambiar los valores de p y h y la fórmula inalterada como se muestra a continuación.

$$(\neg h \vee p) \wedge (\neg p \vee h) \xrightarrow{p=h} (\neg p \vee h) \wedge (\neg h \vee p) = (\neg h \vee p) \wedge (\neg p \vee h)$$

En la conversión de la relación p *Or* $(h_1, h_2, \ldots, h_n)$ son simétricas las variables $\{h_1, h_2, \ldots, h_n\}$ dado que, en su conversión a lógica Booleana,

$$\begin{aligned} p \text{ Or } (h_1, h_2, \ldots, h_n) =& (\neg p \vee (h_1 \vee h_2 \vee \ldots \vee h_n)) \\ & \wedge (\neg h_1 \vee p) \\ & \wedge (\neg h_2 \vee p) \\ & \ldots \\ & \wedge (\neg h_n \vee p) \end{aligned} \tag{4.1}$$

podemos intercambiar cualquier par $h_i, h_j \in \{h_1, h_2, \ldots, h_n\}$ tq $i \neq j$, y la fórmula permanece inalterada. Lo mismo sucede con la relación *Alternative*

$$\begin{aligned} p \text{ Alternative } (h_1, h_2, \ldots, h_n) =& (\neg p \vee (h_1 \vee h_2 \vee \ldots \vee h_n)) \\ & \wedge (\neg h_1 \vee p) \\ & \wedge (\neg h_2 \vee p) \\ & \ldots \\ & \wedge (\neg h_n \vee p) \\ & \wedge ((\neg h_1 \vee \neg h_2) \wedge (\neg h_1 \vee \neg h_3) \wedge \ldots \\ & \wedge (\neg h_1 \vee \neg h_n) \wedge \ldots \wedge (\neg h_{n-1} \vee \neg h_n)) \end{aligned} \tag{4.2}$$

En ambas relaciones existen n variables simétricas donde n es el número de hijos definido en la relación. Es posible identificar estas simetrías antes de la construcción del BDD. En los modelos de configuración existe una restricción *crosstree* habitual que también es simétrica que es la relación de exclusión (*Excludes*) que se define como:

$$p \text{ Excludes } h = \neg p \vee \neg h \tag{4.3}$$

Es evidente que la simetría de las relaciones puede romperse cuando el padre (p) o los hijos de la relación ($\{h_1, h_2, \ldots, h_n\}$) se encuentran en otra relación o bien en una restricción *crosstree*. Para la identificación de la rotura de estas relaciones es necesario construir toda la función f que representa al modelo de configuración.

Identificación de posibles grupos simétricos

Un posible grupo simétrico o PSG, de un modelo de configuración $M_c = (F, C)$, es un par (t, sg), donde:

- t es el tipo de grupo simétrico, $t \in \{\textit{Mandatory}, \text{Or}, \textit{Alternative}, \textit{Excludes}\}$.
- $sg = \{f_1, f_2, \ldots, f_n\}$ es una lista de características que forma el PSG.

El Algoritmo 9 identifica los PSG de un diagrama de características D_c, retornando una lista de ellas. El algoritmo parte de la característica raíz $root$ y recorre recursivamente el modelo para obtener todos los PSG.

$$D_c = (root, features, ct)$$

Algoritmo 8: Obtención de las PSG del diagrama de características

```
1  Function GetPSG( f = (n, p, h, r) )
2      symmetric_groups = ∅;
3      if childs ≠ ∅ then /* No es un característica hoja */
4          foreach r_i ∈ r do
5              switch r_i do
6                  case Mandatory do
7                      symmetric_group = (t, p ∪ f);
8                  end
9                  case Or do
10                     symmetric_group = (t, c);
11                 end
12                 case Alternative do
13                     symmetric_group = (t, c);
14                 end
15             end
16         end
17         symmetric_groups = symmetric_groups ∪ symmetric_group;
18         foreach child ∈ childs do
19             symmetric_groups = symmetric_groups ∪ GetPSG(child);
20         end
21     end
22     return symmetric_groups;
```

Para identificar todos los PSG es necesario recorrer también las restricciones *crosstree* del modelo, para ello se utiliza el Algoritmo 9.

Si aplicamos estos algoritmos a nuestro ejemplo del automóvil obtendremos los PSG que muestra la Tabla 4.4.

Algoritmo 9: Obtención de las PSG del las *crosstree*

```
1 Function GetCTPSG( dc = (root, features, ct) )
2     symmetric_groups = ∅;
3     foreach c_i ∈ ct do
4         if IsExcluded(c_i) then
5             symmetric_groups = symmetric_groups ∪ {(Excluded, variables(c_i))};
6         end
7     end
8     return symmetric_groups;
```

Cláusula	PSG	#ID
$(automovil \vee \neg ruedas) \wedge (\neg automovil \vee ruedas)$	(*Mandatory*,$\{automovil, ruedas\}$)	g_0
$(\neg rojas \vee \neg azules) \wedge (\neg rojas \vee \neg aluminio) \wedge (\neg azules \vee \neg aluminio)$	(*Alternative*,$\{rojas, azules, aluminio\}$)	g_1
$(automovil \vee \neg lunas) \wedge (\neg automovil \vee lunas)$	(*Mandatory*,$\{automovil, lunas\}$)	g_2
$(automovil \vee \neg color) \wedge (\neg automovil \vee color)$	(*Mandatory*,$\{automovil, color\}$	g_3
$(\neg blanca \vee \neg azul) \wedge (\neg blanca \vee \neg roja) \wedge (\neg azul \vee \neg roja)$	(*Alternative*,$\{blanca, azul, roja\}$)	g_4
$(automovil \vee \neg opciones) \wedge (\neg automovil \vee opciones)$	(*Mandatory*,$\{automovil, opciones\}$)	g_5
$(aleacion \vee asistido \vee electronicos)$	(*Or*,$\{aleacion, asistido, electronicos\}$)	g_6
$\neg azul \vee \neg roja$	(*Excludes*,$\{azul, roja\}$)	g_7

Tabla 4.4: PSG de automóvil

Selección de los PSG

El algoritmo de *sifting* agrupado de Panda et al. [105], requiere que, una misma variable no puede estar incluida en varios grupos. Estas condiciones no se cumplen en los grupos identificados por el Algoritmo 9. Por ejemplo, las características $automovil$ o $azul$ forman parte de varios grupos. Por lo tanto, se debe realizar una selección de los grupos.

La selección de los grupos puede expresarse mediante fórmulas pseudo Booleanas [37]. Por ejemplo, en el caso de los grupos que contiene la característica $automovil$ se expresa mediante la ecuación 4.4. En ella se indica que podemos seleccionar sólo uno de los grupos.

$$1 \cdot g_0 + 1 \cdot g_2 \leq 1 \tag{4.4}$$

Por otra parte, también nos interesa maximizar la longitud de los grupos seleccionados, para agrupar un mayor número de variables, esto en nuestro ejemplo se traduciría en la maximización de la ecuación 4.5.

$$2 \cdot g_0 + 3 \cdot g_1 + 2 \cdot g_2 + 2 \cdot g_3 + 3 \cdot g_4 + 2 \cdot g_5 + 3 \cdot g_6 + 2 \cdot g_7 \tag{4.5}$$

Se ha utilizado la herramienta **minisat+** [49], que traslada las clausulas pseudo Booleana a un problema SAT. Esta herramienta precisa que se traslade a forma

normal para clausulas pseudo Booleanas. Para ello, es necesario cambiar la restricción $\leq$ por la restricción $\geq$ cambiando el signo de los coeficientes, quedando la ecuación 4.4 como:

$$-1 \cdot g_0 + -1 \cdot g_2 >= -1 \tag{4.6}$$

Y se debe minimizar la ecuación 4.5 en vez de maximizar. Por tanto, también deberemos cambiar el signo de la ecuación como:

$$-2 \cdot g_0 - 3 \cdot g_1 - 2 \cdot g_2 - 2 \cdot g_3 - 3 \cdot g_4 - 2 \cdot g_5 - 3 \cdot g_6 - 2 \cdot g_7 \tag{4.7}$$

El resultado sobre los PSG de la Tabla 4.4 selecciona los grupos $\{g_0, g_1, g_4, g_6\}$.

Ordenación de las características

Las variables que formen parte de los PSG inducen un posible orden para la construcción del BDD, pero es necesario un orden para todas las variables que garantice que los grupos sean contiguos y optimice el tamaño del BDD. Para ello se ha desarrollado el Algoritmo 10, que partiendo de un orden estático inicial π, reajusta el orden manteniendo los PSG.

El algoritmo obtiene la primera variable c_f en el orden de partida π (línea 5), lo elimina de la lista de pendientes de procesar (línea 6), y hace que sea la siguiente variable en el nuevo orden ret (línea 7), si la variable procesada c_f no está incluida en ningún grupo (línea 9) obtenemos el siguiente elemento (línea 5), en caso contrario buscamos el elemento c_f dentro del grupo encontrado c_{psg}, y selecciona la siguiente variable c_f dentro de la lista de variables pendientes de procesar f (línea 12 y 13) que esté incluida en las variables del grupo c_g, eliminamos la variable tanto de la lista de variables pendientes de procesar f (línea 14) como del grupo c_g (línea 15), esta acción se realiza mientras resten variables por procesar en el grupo (línea 11). Mientras queden variables por procesar (línea 4) se repite el proceso.

Al final de todo el proceso obtenemos un orden inducido por la ordenación estática combinado con la agrupación de variables simétricas.

La aplicación del Algoritmo 10 partiendo del orden π resultante de aplicar la heurística FORCE [3]:

$\pi =$ $\{rojas, aluminio, ruedas, azules, lunas, tintadas, roja, color\_carroceria,$
$blanca, azul, opciones, aleacion, asistido, electricos\}$

Algoritmo 10: Intercalación de heurística estática y PSG

```
1  Function CombinePSGAndStaticOrder( π : lista ordenada, psg : lista de PSG )
2      ret = ∅;
3      f = copia_de(π);
4      while f ≠ ∅ do
5          c_f = primer_elemento(f);
6          eliminar_primer_elemento(f);
7          ret = ret ∪ c_f;
8          c_psg = buscar_psg_que_contega_caracteristica(c_f, psg);
9          if c_psg ≠ ∅ then
10             c_g = caracteristicas_del_grupo(c_g);
11             while c_g ≠ ∅ do
12                 c_f = primer_elemento(c_g ∩ f);
13                 ret = ret ∪ c_f;
14                 eliminar_elemento(f, c_f);
15                 eliminar_elemento(c_g, c_f);
16             end
17         end
18     end
19     return ret;
```

y los PSG

$$
\begin{aligned}
g_0 &= \{automovil, ruedas\} \\
g_1 &= \{rojas, azules, alumnio\} \\
g_5 &= \{blanca, azul, roja\} \\
g_6 &= \{aleacion, asistido, electricos\}
\end{aligned}
$$

genera el orden π' manteniendo juntas las variables de los grupos PSG.

$$
\begin{aligned}
\pi' = \{&rojas, aluminio, azules, ruedas, automovil, lunas, tintadas, roja, blanca, \\
&azul, color\_carroceria, opciones, aleacion, asistido, electricos\}
\end{aligned}
$$

4.3.4 Resultados experimentales

La Tabla 4.5 muestra el número de nodos que se obtiene después de utilizar la heurística estática **MINCE** [4], obtener sus PSG, y posteriormente aplicar una la heurística dinámica agrupada utilizando como grupos los PSG, comparados con la aplicación de la heurística **MINCE** y después aplicar ***sifting*** **simétrico**. En ella se observa que la agrupación reduce el total del número de nodos, y el promedio, y es

menor para la mayoría de los modelos de configuración.

Modelo de configuración	Nodos *sifting* simétrico	Nodos *sifting* agrupado PSG
Thread	106	90
eCos_isoinfra	158	160
BankingSoftware	225	220
LinuxKernel_block	71	74
Electronic Shopping	930	807
LinuxKernel_mm	42	41
Dell_XML	3606	2748
LinuxKernel_lib	162	163
Web_Portal	93	95
Model_Transformation	149	139
eCos_infra	35	34
Assess4me	52	50
Documentation_Generation	88	116
LinuxKernel_arch	52	52
eCos_net	144	143
eCos_fs	27	26
eCos_language	196	194
Graph	43	45
eCos_compat	31	33
eCos_redboot	71	72
LinuxKernel_fs	1370	1368
LinuxKernel_crypto	966	955
eCos_services	81	82
eCos_kernel	80	80
Smart Home v2	37	36
HIS	89	95
LinuxKernel_security	172	167
SmartTV	138	139
eCos_io	221	231
Total	**9434**	**8454**
Promedio	**325**	**292**

Tabla 4.5: *Sifting* simétrico versus ***Sifting* agrupado**

La agrupación y bloqueo en el *sifting* de las PSG, mejora las heurísticas estáticas después de la aplicación del *sifting* simétrico.

Para validar los resultados utilizando estadística deberíamos aplicar el test de Friedman [54], pero como indica Conover [32] este test es adecuado para valores de k mayores de 5, y en nuestro caso el valor es de k es 2.

La ejecución del test tiene un valor de p de $0,046$, por debajo de $0,05$ en consecuencia es significativa la diferencia, y por tanto podemos confirmar que la agrupación de las PSG en el ***sifting* simétrico** mejora los resultados respecto al simétrico.
Transcripción 16

```
Pairwise comparisons using Conover's test for a two-way
balanced complete block design

SIFTING SIMÉTRICO
SIFTING AGRUPADO 0.046
P value adjustment method: holm
```

4.3.5 Conclusiones

El conocimiento del problema, los modelos de configuración en nuestro caso, ha permitido realizar un algoritmo que utiliza este conocimiento, para identificar las posibles variables simétricas. La agrupación de variables simétricas reduce el crecimiento del BDD. Además, está agrupación de variables, puede aplicarse independientemente de la heurística de ordenación estática inicial.

Esta heurística no sólo puede aplicarse a los modelos de configuración, si no a cualquier problema que, partiendo de un modelo, y unas reglas de transformación a lógica proposicional, como por ejemplo de los circuitos electrónicos, o los diagramas de Fault Tree (Análisis de fallos de sistemas) , que incluyen puertas XOR, que es una relación PSG.

4.4 *Core* y *dead* e *impact* y *exclusion set*

4.4.1 Objetivo

Identificar las características *core* y *dead*, y los conjuntos *impact* y *exclusion* de un modelo de configuración puede realizarse mediante SAT solvers [61] o BDD's [91]. Pero los algoritmos actuales tienen un rendimiento temporal pobre, y esto genera tiempos de respuesta altos, impidiendo la interactividad de los usuarios[66].

A continuación se describirá cómo de realizar las operaciones de *dead features*, *core features*, *impact set* y *exclusion set* de modo eficiente, sobre un modelo de configuración (M_c) modelado por la función Booleana $\psi = \psi_j \wedge \psi_c$ mediante BDD, donde ψ_j son las restricciones debidas a la jerarquía del modelo, y ψ_c las no jerárquicas o *crosstree*.

Primero se presentará el diagrama de características que se utilizará como ejemplo, a continuación, se describirá el nuevo algoritmo de cálculo de características *core* y *dead* [107] y posteriormente el algoritmo de cálculo de los *impact* y *exclusion set* [67], y finalmente el coste computacional de ambos algoritmos comparado con los algoritmos existentes de Tartler [131] y Lesta et al. [86].

Se presentarán también los resultados experimentales de los nuevos algoritmos de cálculo de características *dead* y *core*, así como los *exclusion* e *inclusion set*.

4.4.2 Fundamentos teóricos

Diagrama de características de ejemplo

Los ejemplos se realizarán utilizando el diagrama de características de la Figura 4.5, siendo su transformación a lógica Booleana la que muestra la Tabla 4.6.

Relación	Fórmula Booleana
f_1 optional f_3	$\neg f_3 \vee f_1$
f_3 optional f_2	$\neg f_3 \vee f_2$
f_3 optional f_4	$\neg f_3 \vee f_4$
f_3 optional f_5	$\neg f_3 \vee f_5$
f_3 optional f_6	$\neg f_3 \vee f_6$
crosstree	$\neg f_1 \vee \neg f_2$
crosstree	$\neg f_4 \vee \neg f_5$
crosstree	$\neg f_4 \vee \neg f_6$

Tabla 4.6: Transformación a lógica Booleana

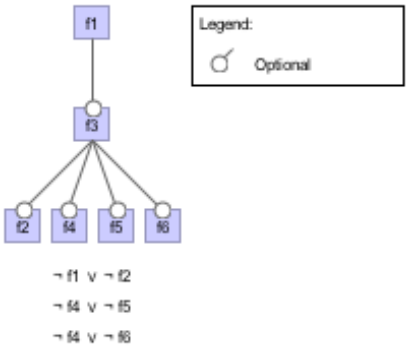


Figura 4.5: Diagrama de característica de ejemplo

Dead* y *core

Una evaluación de una formula ψ es una asignación en ψ en una tabla de verdad. Por ejemplo, una posible evaluación para la transformación a lógica Booleana (ψ) del modelo del ejemplo es $\{f_1 = 1, f_2 = 0, f_3 = 0, f_4 = 0, f_5 = 0, f_6 = 0\}$, que evalúa ψ a cierto, como vimos en la sección de operaciones la codificación de la fórmula ψ en un BDD codifica todas las posibles configuraciones del modelo como aquellas que hacen la fórmula ψ cierta.

En un BDD las evaluaciones se representan como caminos desde el nodo raíz a los nodos terminales. Por ejemplo, el BDD de la Figura 4.6 almacena la evaluación anteriormente mencionada $v_8 \rightarrow v_7 \dashrightarrow v_6 \dashrightarrow v_4 \dashrightarrow v_3 \dashrightarrow v_2 \dashrightarrow 1$, una evaluación es cierta si su correspondiente camino llega al nodo 1-terminal.

Sean $f \rightarrow_+ 1$ y $f \dashrightarrow_+ 1$ dos predicados. $f \rightarrow_+ 1$ es cierto si el nodo 1-terminal es accesible desde la raíz atravesando una arista continúa ($then$) de algún nodo etiquetado como f; $f \dashrightarrow_+ 1$ es cierto si el nodo 1-terminal es accesible desde la raíz atravesando una arista discontinúa ($right$) de algún nodo etiquetado como f. Por ejemplo, en la Figura 4.6 $f_3 \rightarrow_+ 1$ es cierto debido al camino $v_8 \rightarrow v_7 \dashrightarrow v_6 \rightarrow v_5 \dashrightarrow 1$. No obstante, $f_1 \dashrightarrow_+ 1$ es falso porque v_8 es el único nodo etiquetado con f_1 y, una vez se atraviesa la arista discontinúa ($false$), es imposible llegar al nodo 1-terminal.

Lema 5. *Una característica f es* core *si y sólo si $f \rightarrow_+ 1$ es cierto y $f \dashrightarrow_+ 1$ es falso.*

Lema 6. *Una característica f es* dead *si y sólo si $f \rightarrow_+ 1$ es falso y $f \dashrightarrow_+ 1$ es cierto.*

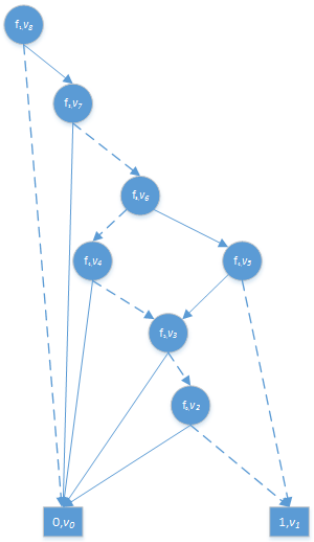


Figura 4.6: BDD para la ecuación de la Tabla 4.6 de acuerdo con el orden $f_1 \prec f_2, \prec f_3 \prec f_4 \prec f_5 \prec f_6$

Demostración. Por definición una característica f es *core* si (i) si y sólo si todas las configuraciones válidas incluyen f, y (ii) es *dead* si ninguna configuración valida incluye f. Es decir, cuando se determina si f es *core* o *dead*, sólo las configuraciones válidas son de interés, o en términos de BDD, los caminos de la raíz al nodo 1-terminal. Por lo tanto, debemos centrarnos en los casos donde se atraviesa f y el nodo 1-terminal es alcanzado, i.e. cuando $f \rightarrow_+ 1$ es cierto o $f \dashrightarrow_+ 1$ es cierto:

1. Si $f \rightarrow_+ 1 =$ cierto y $f \dashrightarrow_+ 1 =$ cierto, el nodo 1-terminal puede ser accesible a través de las aristas continúas ($then$) y discontinuas ($else$). Es decir, f es está incluido en algunas configuraciones, pero no en otras configuraciones validas. Por lo tanto, f no es ni *core* ni *dead*.

2. Si $f \rightarrow_+ 1 =$ cierto y $f \dashrightarrow_+ 1 =$ falso, el nodo 1-terminal no puede ser accesible a través de aristas discontinúas ($else$) con nodos etiquetados como f (debido a que $f \dashrightarrow_+ 1 =$ falso). En otras palabras, todas las configuraciones válidas incluyen f, por lo tanto f es *core*.

3. Si $f \rightarrow_+ 1 =$ false y $f \dashrightarrow_+ 1 =$ true, el nodo 1-terminal no puede ser accedido a través de nodos continuos ($then$). Todos los caminos pasan por f a través del nodo discontinúo ($else$). Por lo que f está deshabilitado en todas

las configuraciones, en consecuencia, f es *dead*.

□

4.4.3 Heurísticas desarrolladas

El Algoritmo 11 calcula las características *core* y *dead* para una fórmula de entrada ψ, que representa un modelo de configuración (M_c) codificado mediante el BDD `bdd`, de acuerdo con la ordenación de variables $var\_ordering$. Se utilizan dos *arrays* denominados $through\_then$ y $through\_else$ para almacenar en la posición $i-1$ los valores $f \rightarrow_+ 1$ y $f \dashrightarrow_+ 1$ respectivamente. El cálculo de estos *arrays* es realizado por el Algoritmo 12 (línea 8). Por ejemplo, después de la ejecución del Algoritmo 12 para el BDD de la Figura 4.5 obtenemos los siguientes valores:

$$through\_then = \{cierto, falso, cierto, cierto, cierto\}$$
$$through\_else = \{falso, cierto, cierto, cierto, cierto\}$$

Por lo tanto,

$$through\_then[1] = f_2 \rightarrow_+ 1 = falso$$
$$through\_else[1] = f_2 \dashrightarrow_+ 1 = cierto$$

Entonces el Algoritmo 11 aplica los lemas 8 y 7 para identificar las características *core* y *dead* de acuerdo con los *arrays* $through\_then$ y $through\_else$ (líneas 10-16). Por ejemplo, dado que $f_2 \rightarrow_+ 1 = falso$ y $f_2 \dashrightarrow_+ 1 = cierto$, el Algoritmo 11 determina que f_2 es *dead*.

Para calcular $f \rightarrow_+ 1$ y $f \dashrightarrow_+$, el Algoritmo 12 sigue el procedimiento propuesto por Bryant [25] para atravesar un BDD y realizar una operación con sus vértices. El algoritmo se llama con el vértice raíz (líneas 7 y 8), y marcando primero todos los nodos como visitados a falso. Este procedimiento visita cada vértice en el grafo recursivamente visitando los dos nodos hijos del nodo raíz. Cuando se visita un vértice, se marca (línea 3), con el objetivo de identificar los nodos que ya han sido visitados. Para cada nodo, se testea si este llega al nodo 1-terminal a través de la arista $then$ (línea 4) o $else$ (línea 17), si es así se llama al Algoritmo 13 para actualizar los *arrays* $through\_then$ y $through\_else$, para marcar los nodos que han sido eliminados de acuerdo con la reducción R2 (Sección 2.1 del Capítulo 2). Por ejemplo, como muestra la Figura 4.7, la arista $v_5 \dashrightarrow 1$ en la Figura 4.5 es el resultado de eliminar los vértices v_x y v_y del BDD, porque sus aristas *then* y

Algoritmo 11: Cálculo de las características *core* y *dead*

```
1  Function get_core_and_dead_features( bdd: BDD, var_ordering: array)
       Output: core_features: list; dead_features : list
       var: i:int;through_then, through_else: array[0..n-1] of boolean; it_reaches_the_1_terminal :
            array[0..m-1] of boolean
2      core_features = {};
3      dead_feature = {};
4      through_then = {falso, falso, ..., falso};
5      through_else = {falso, falso, ..., falso};
6      it_reaches_the_1_terminal = {falso, falso, ..., falso};
7      v = root(bdd);
8      does_it_reach_the_1_terminal?(v, through_then, through_else, it_reaches_the_1_terminal);
9      for i=0; i ¡length(var_ordering);i++ do
10         if through_then[i] ∧ ¬through_else[i] then
11             core_features.insert(var_ordering[i]);
12         else
13             if through_then[i] ∧ ¬through_else[i] then
14                 dead_features.insert(var_ordering[i]);
15             end
16         end
17     end
18     return (core_features, dead_features);
```

else apuntan al mismo nodo. Teniendo en cuenta la eliminación y que $f_4 \dashrightarrow_+ 1$ es cierto, entonces $f_5 \rightarrow_+ 1$, $f_5 \dashrightarrow_+ 1$, $f_6 \rightarrow_+ 1$ y $f_6 \dashrightarrow_+ 1$ también son cierto.

Impact y *exclusion set*

Los *impact* y *exclusion set* pueden calcularse llamado repetidamente a un SAT-solver. Dado que dadas dos características f y f', f' es *impact*(*exclusion*) *set* de f si y sólo si $f' \wedge \psi$ es una tautología ($f \wedge \psi \wedge f'$ es unsatisfacible). Utilizando un razonamiento análogo a [20],[43] y [21] como sigue:

- en una asignación satisfacible de $f \wedge \psi \wedge f'$:
  - todas las características que son ciertas no se incluyen en el *exclusion set* de f ni en el *impact set* f'.
  - todas las características que son falsas no se incluyen el *impact set* de f ni el *exclusion set* de f'.
- en una asignación satisfacible de $\neg f \wedge \psi \wedge f'$.
  - todas las características que son ciertas no se incluyen en el *impact set* de f ni el *exclusion set* de f'.

Algoritmo 12: Cálculo del *array* $t\_reaches\_the\_1\_terminal$

```
1  Function does_it_reach_the_1_terminal?( bdd : BDD, v: Node, through_then, through_else: array[0..n-1] of
     boolean,it_reaches_the_1_terminal : array[0..m-1] of boolean)
2      if ¬bdd[v].mark then
3          bdd[v].mark = cierto;
           // v alcanza el nodo 1-terminal a través de then?
4          if bdd.then = 1 then
5              through_then[bdd[v].index] = true;
6              it_reaches_the_1_terminal[v] = true;
7              update_reduced_nodes(v, "then", through_then, through_else);
8          else
9              if bdd[v].then ≠ 0 then
                   // Continuar buscando
10                 it_reaches_the_1_terminal[v] = does_it_reach_the_1_terminal?(bdd[v].then,
                     through_then, through_else, it_reaches_the_1_terminal);
11                 if it_reaches_the_1_terminal[v] then
12                     through_then[bdd[v].index] = true;
13                     update_reduced_nodes(v, "then", through_then, through_else);
14                 end
15             end
16         end
           // v alcanza el nodo 1-terminal a través de else?
17         if bdd.else = 1 then
18             through_else[bdd[v].index] = true;
19             it_reaches_the_1_terminal[v] = true;
20             update_reduced_nodes(v, "else", through_then, through_else);
21         else
22             if bdd[v].else ≠ 0 then
                   // Continuar buscando
23                 it_reaches_the_1_terminal[v] = does_it_reach_the_1_terminal?(bdd[v].else,
                     through_then, through_else, it_reaches_the_1_terminal);
24                 if it_reaches_the_1_terminal[v] then
25                     through_else[bdd[v].index] = true;
26                     update_reduced_nodes(v, "then", through_then, through_else);
27                 end
28             end
29         end
30     end
31     return it_reaches_the_1_terminal[v];
```

Algoritmo 13: Actualización de nodos de reducidos

```
1  Action update_reduced_nodes( direction : string ∈ {"then","else"}, bdd : BDD, v: Node,
     through_then, through_else: array[0..n-1] of boolean)
       var: i:int
2      ;
3      if direction == "then" then
4          for i = bdd[v].index + 1; i < bdd[bdd[v].then].index; i++ do
5              through_then[i] = cierto;
6              through_else[i] = falso;
7          end
8      else
9          for i = bdd[v].index + 1; i < bdd[bdd[v].else].index; i++ do
10             through_then[i] = cierto;
11             through_else[i] = falso;
12         end
13     end
```

– todas las características que son falsas no se incluyen en el *exclusion set* de f ni el *impact set* de $f^{'}$.

El Algoritmo 14 calcula el *impact* y *exclusion* set de llamando repetitivamente al Algoritmo 11.

Coste computacional

La Figura 4.8 resume las dependencias y costes de los algoritmos. En relación con las dependencias, el Algoritmo 11 detecta las características *core* y *dead* llamando al Algoritmo 12, que posteriormente llama al Algoritmo 13. Finalmente, el Algoritmo 14 calcula el *impact* y *exclusion set* llamando al Algoritmo 11.

Con relación al coste computacional, siendo m el número de nodos del BDD, y n el número de variables de fórmula Booleana. El Algoritmo 12 sigue el procedimiento propuesto por Bryant para recorrer BDDs, en consecuencia, visita cada nodo únicamente una vez. Adicionalmente, hay que contar los nodos eliminados del BDD mediante la reducción R2 realizando $n^{'}$ operaciones, donde $n^{'}$ es estrictamente menor que n: si $n^{'}$ fuera igual a n, entonces todos los nodos en el BDD deberían ir a los nodos terminales 1 y 0 pasando por todas las variables, esto es imposible , es decir, los nodos que codifican una variable con posición k sólo podrían saltar a los nodos con posiciones $k+1\ldots$. Por lo tanto, el Algoritmo 13 es $\mathcal{O}(n^{'})$ y por lo tanto el Algoritmo 12 es $\mathcal{O}(mn^{'})$. Como consecuencia los algoritmos 11 y 14 son $\mathcal{O}(mn^{'})$ y $\mathcal{O}(mnn^{'})$ respectivamente, hay que tener en cuenta que m es

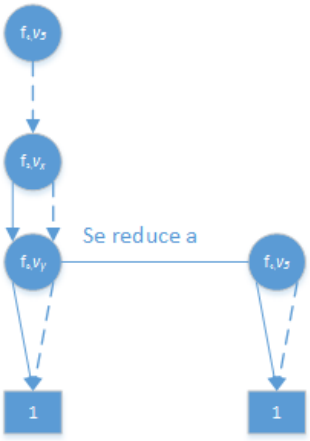


Figura 4.7: El Algoritmo 13 actualiza el acceso al nodo 1-terminal de los vértices que han sido eliminados del BDD debido a la reducción R2 , i.e. las aristas $then$ y $else$ van a parar al mismo nodo

mucho mayor que n, y domina el comportamiento asintótico del algoritmo, y por tanto, reducir el tamaño del BDD mejora estos algoritmos, ambos tiempos mejoran los tiempos existentes $\mathcal{O}(mn)$ (*core* y *dead*) y $\mathcal{O}(mn^2)$ (*impact* y *exclusion*).

Como se mostrará en los resultados experimentales, en la práctica esta aproximación es más eficiente que los trabajos existentes, puesto que m y n están positivamente correlados. En contraste m y $n^{'}$ están negativamente correlados porqué si $n^{'}$ es grande se aplica la reducción R2 intensivamente, en consecuencia, el tamaño del BDD decrece. De lo anterior se deduce que $mn^{'} \ll mn$, y por lo tanto $mnn^{'} \ll mn^2$.

4.4.4 Resultados experimentales

La Tabla 4.9 resume los resultados experimentales con el *benchmark* utilizado para validar nuestros algoritmos. Los modelos de configuración $m_1 - m_{10}$ han sido obtenidos del repositorio SPLOT [1], los modelos $m_{11} - m_{14}$ [124], [14], y los modelos $m_{15} - m_{19}$ en Bak, Kacper [9], siendo los modelos $m_{16} - m_{19}$ aleatorios. Los test han sido realizados con un Intel Core i7-3537 v 2.00 GHZ con 8 GB de RAM, realizando un promedio de de 50 experimentos.

Los resultados del Algoritmo 11 de cálculo *core* y *dead* son mejores que los de

[1] http://www.splot-research.org

Algoritmo 14: Cálculo de *imapact* y *exclusion set*

```
1  Function does_it_reach_the_1_terminal?( bdd : BDD, var_ordering : array)
2      impact_set = {};
3      dead, core = get_dead_and_core_features(bdd, var_ordering);
4      for i = 0; i < length(var_ordering); i++ do
5          if var_ordering[i] ∈ dead then
6              impact_set[i] = {};
7              exclusion_set[i] = var_ordering;
8          else
9              if var_ordering[i] ∈ core then
10                 impact_set[i] = var_ordering;
11                 exclusion_set[i] = dead;
12             else
13                 core', dead' = get_dead_and_core_features(bdd, var_ordering);
14                 for j = 0; j < length(core'); j++ do
15                     k = index of core envar_ordering;
16                     impact_set[k].insert(var_ordering[i]);
17                     exclusion_set[i] = dead';
18                 end
19             end
20         end
21     end
```

Lesta et al. [86] como Tartler [131], utilizando o bien BDD's (implementados con la librearía BuDDy [87]) o con SAT (implementados con minisat[48]).

Los resultados del Algoritmo 14 de cálculo del *impact* y *exclusion set* son mejores a los resultados de Boender [20], tanto en su implementación con BDD's como con SAT.

La heurística de ordenación que se utilizó para obtener estos resultados fue la Narodytska and Walsh [100], ha de hacerse notar que aun no siendo, la mejor heurística de ordenación de variables de BDD's para modelos de configuración como se ha mostrado en el Capítulo 3, los resultados son mejores que la implementación con SAT.

Con el objetivo de verificar las diferencias estadísticas de los datos obtenidos, se realizó también el test de Conover [33], tanto para el Algoritmo 11 como el 14, y como se muestra en las tablas 4.7 y 4.8,y en todos ellos existen diferencias significativas.

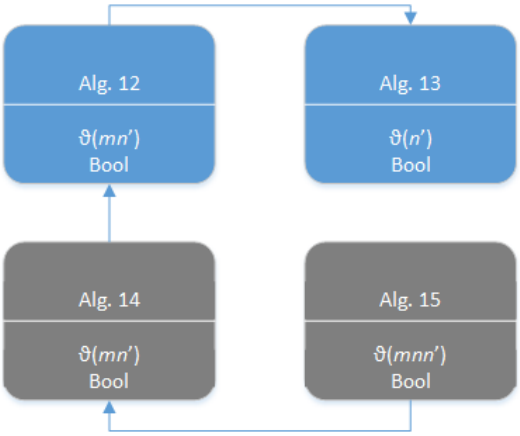


Figura 4.8: Interdependencias de los algoritmos y costes computacionales

V.*p* / V.*p*	**Lesta SAT**	**Lesta BDD**	**Tartler SAT**	**Tartler BDD**	**Perez-Morago**
Lesta SAT	-	1,391e-28	1,661e-13	1,268e-39	5,327e-48
Lesta BDD	1,391e-28	-	1,661e-13	1,661e-13	1,391e-28
Tartler SAT	1,661e-13	1,661e-13	-	1,391e-28	1,268e-39
Tartler BDD	1,268e-39	1,661e-13	1,391e-28	-	1,661e-13
Perez-Morago	5,327e-48	1,391e-28	1,268e-39	1,661e-13	-

Tabla 4.7: Valor de p Test Conover [32] para algoritmos cálculo *dead/core*

V.*p* / V.*p*	**Boender SAT**	**Boender BDD**	**Perez-Morago**
Boender SAT	-	6,737e-11	2,389e-18
Boender BDD	6,737e-11	-	3,469e-10
Perez-Morago	2,389e-18	3,469e-10	-

Tabla 4.8: Valor de p Test Conover [32] para algoritmos *impact/exclusion*

Id	Modelo	#caract.	Lesta [86]		Tartler [131]		Perez-Morago et al. [107] (s)	Boender [20]		Heradio et al. [67] (s)
			SAT (s)	BDD (s)	SAT (s)	BDD (s)		SAT (s)	BDD (s)	
m1	Billing	88	0,022	0,001	0,0101	0,0007	0,00001	1,0239	0,0555	0,0012
m2	Coche ecológico	94	0,091	0,005	0,05008	0,002	0,00002	1,55376	0,1318	0,0023
m3	UP estructural	97	0,084	0,003	0,0502	0,0017	0,00002	3,30642	0,1893	0,0052
m4	Xtext	137	0,076	0,005	0,04062	0,0027	0,00003	16,6447	0,4054	0,00702
m5	Battle of tanks	144	0,188	0,008	0,09803	0,0039	0,00003	16,8988	0,8784	0,0375
m6	FM Test	168	0,175	0,018	0,10501	0,0085	0,00009	21,3226	0,9692	0,0701
m7	Printers	172	0,128	0,012	0,068	0,0051	0,00006	15,3983	0,6942	0,0645
m8	Banking software	176	0,194	0,009	0,09802	0,0053	0,00005	23,9632	1,41058	0,0952
m9	eShop	290	0,48	0,041	0,234	0,02013	0,00031	173,126	33,9121	0,69801
m10	EIS	366	0,742	0,065	0,44803	0,03404	0,0021	477,41604	91,1755	9,31114
m11	axTLS	108	0,071	0,002	0,0384	0,00121	0,00015	3,3516	0,0776	0,0096
m12	Fiasco	171	0,149	0,005	0,07864	0,00252	0,00024	4,0653	0,21343	0,01856
m13	uClibc	369	3,844	0,42	1,83082	0,1874	0,01362	147,65444	20,7488	2,02692
m14	BusyBox	881	10,05	0,997	4,93475	0,5346	0,04274	401,12872	74,4468	6,74952
m15	Android	88	0,792	0,088	0,3805	0,04031	0,00232	1,0836	0,17446	0,017322
m16	FM-500-50-1	500	27,7	5,238	14,42221	2,24335	0,18439	944,63451	212,45127	18,6329
m17	FM-1000-100-2	1000	70,12	16,68	37,82924	9,02403	0,29285	-1	2986,45786	189,0357
m18	FM-2000-200-3	2000	325,7	78,73	175,905	42,35382	2,80637	-1	-1	772,07279
m19	FM-5000-500-4	5000	1976	406,6	874,46547	226,93796	7,73044	-1	-1	3385,9763

Tabla 4.9: Resumen de los resultados experimentales Algoritmo 11

4.4.5 Conclusiones

Los BDD's a parte de ser utilizados para la representación y cálculo de funciones Booleanas, permiten obtener de forma eficiente, información de las funciones que representan como las características *core*/*dead* o los *imapact*/*exclusion sets*. Como se ha demostrado, tanto a nivel teórico, como a nivel experimental.

Las otras heurísticas ([131], [86] y [20]) utilizaban los BDD's o SAT *solvers* como cajas negras, nuestros algoritmos en cambio utilizan su representación en BDD's de forma eficiente para obtener una mejora del rendimiento.

4.5 Cálculo *reach_one* múltiples BDD's

4.5.1 Objetivo

En esta sección veremos como descomponer el Algoritmo 12, en varios BDD's. Esta descomposición, permite además la distribución del modelo y el cálculo en diferentes hilos de ejecución, aprovechando la concurrencia de los sistemas computacionales actuales.

Una vez presentados los fundamentos teóricos de la heurística, compararemos sus resultados experimentales con los obtenidos por el Algoritmo 12, utilizando el *benchmark* del Capítulo 3.

4.5.2 Fundamentos teóricos

Algoritmo reach-one sobre varios BDD's

Un modelo de configuración M_c, incluye clausulas derivadas de las restricciones jerárquicas del modelo (ψ_j), y las no jerárquicas (ψ_c). En consecuencia, podemos dividir la función que codifica el modelo $\psi = \psi_j \wedge \psi_c$, en dos funciones. Obviamente, es posible construir un BDD para cada una de estas funciones, B_j y B_c.

Independientemente de la función Booleana, que represente el modelo de configuración, tenga o no restricciones *crosstree*. Una función Booleana ψ siempre va a poder descomponerse como una conjunción de k sub. funciones $\psi = \psi_1 \wedge \psi_2 \wedge \ldots \psi_k$ [133].

El BDD B_t que codifica la función ψ, se obtiene aplicando la operación *AND* a los BDD B_j, y B_c. Esta operación se realiza utilizando el Algoritmo 15 *Apply* de Bryant [25].

La operación *AND*, sobre los BDD's B_1 y B_2 de la Figura 4.9, siguiendo el Algoritmo 15, obtiene el BDD B, descendiendo en paralelo por ambos BDD's, como describe la siguiente secuencia:

1. Partiremos de $v_1 = x_1$ y $v_2 = x_2$, dado que ambos nodos no son terminales, seleccionaremos el nodo con índice más pequeño, v_1 en este caso.

2. Si descendemos por el nodo x_1 del BDD B_1 del por la arista *else* nos llevaría al nodo terminal 0, en el BDD B_1, y al nodo x_2 del BDD B_2. En consecuencia $v_1 = 0$ y $v_2 = x_2$.

3. Luego descendemos para la arista *else* del nodo x_2 lo que nos lleva al nodo x_4 del BDD B_2. En consecuencia $v_1 = 0$ y $v_2 = x_4$.

4. Al descender por la arista *else* del nodo x_4 nos lleva al nodo terminal 0. Por tanto, $v_1 = 0$ y $v_2 = 0$.

5. Al ser tanto v_1 como v_2 constantes realizamos la operación, and en nuestro caso, y por tanto el resultado, al ser dos nodos terminales 0, y una operación and, el nodo terminal final es el 0.

La secuencia seguida genera un camino que es la construcción del BDD resultante. Hay que tener en cuenta que una vez realizada la operación *Apply*, es necesario aplicar las reducciones **R1**, **R2** y **R3**. Pero si no lo aplicamos, obtenemos un OBDD que representa la función, pero no reducido. Sólo tiene las reducciones de los BDD's de las sub. funciones.

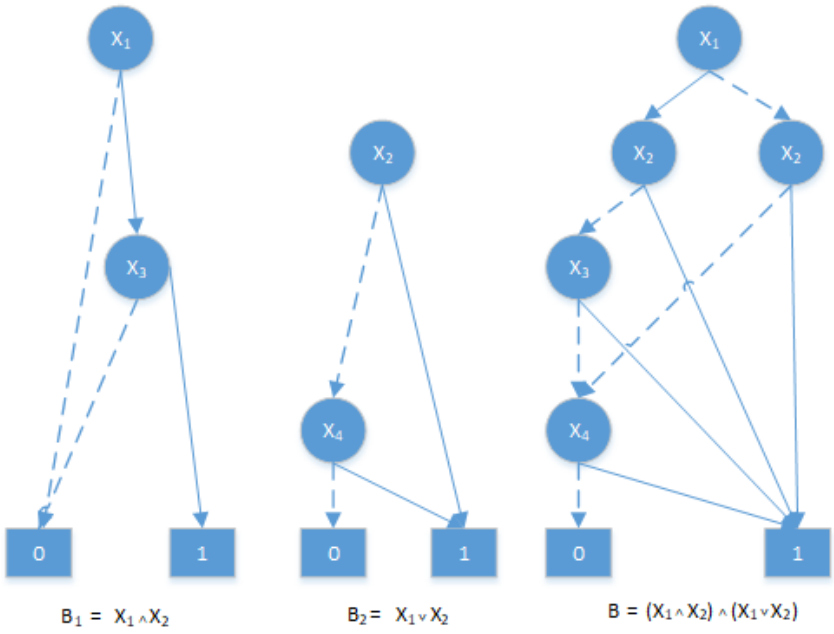


Figura 4.9: BDD ejemplo operación *Apply*

El Algoritmo 12 recorre los nodos del BDD completo (B_t) de la función ψ, de igual que el Algoritmo 15, lo hace con los nodos de los BDD's B_j y B_c, desciende por el grafo hasta encontrar un nodo terminal.

Algoritmo 15: Algoritmo *Apply* para la operación *AND* [25]

```
1  Function Apply_And( v1 : Node, v2 : Node)
       // Nodos terminales
2      if v1.index == n ∧ v2.index == n then
3          if v1 == 1_terminal ∧ v2 == 1_terminal then
4              return 1_terminal;
5          else
6              return 0_terminal;
7          end
8      else
           // Nodos no terminales
9          u = crear nuevo Node;
10         u.index = min(v1.index, v2.index);
11         if v1.index == u.index then
12             v1_else = v1.else;
13             v1_then = v1.then;
14         else
15             v1_else = v1;
16             v1_then = v1;
17         end
18         if v2.index == u.index then
19             v2_else = v2.else;
20             v2_then = v2.then;
21         else
22             v2_else2 = v2;
23             v2_then = v2;
24         end
25         u.else = Apply_And(v1_else, v2_else);
26         u.then = Apply_And(v1_then, v2_then);
27         return u;
28     end
```

Algoritmo 16: Combinación de los algoritmos 12 y 15

```
1  Function Combine( v_j: Node, v_c: Node, reaches_one: array, through_then, through_else: array, visted : hash)
2      id = concatenar(v_j.id, v_c.id);
3      index = min(v_j.index, v_c.index);
4      if id ∉ visited then
5          visited.insertar(id);
6          if v_j.then.index == n ∧ v_c.then.index == n then
7              if v_j.then == 1_terminal ∧ v_c.then == 1_terminal then
8                  through_then[index] = cierto;
9                  ireaches_one[id] = cierto;
10                 if min(v_j.then.index, v_c.then.index) > index then
11                     update_reduced_nodes(index, min(v_j.then.index, v_c.then.index),
                         through_then, through_else);
12                 end
13             end
14             if v_j.else == 1_terminal ∧ v_c.else == 1_terminal then
15                 through_else[index] = cierto;
16                 reaches_one[id] = cierto;
17                 if min(v_j.else.index, v_c.else.index) > index then
18                     update_reduced_nodes(index, min(v_j.else.index, v_c.else.index),
                         through_then, through_else);
19                 end
20             end
21         else
22             if v_j.index == index then v_j_else = v_j.else; v_j_the = v_j.then;
23             else v_j_else = v_j; v_j_then = v_j ;
24             if v_c.index == index then v_c_else = v_c.elsev_c_then = v_c.then;
25             else v_c_else = v_c; v_c_then = v_c ;
26             if Combine(v_j_then, c_j_then, reaches_one, through_then, through_else, visted) then
27                 through_then[index] = cierto;
28                 reaches_one[id] = cierto;
29                 if min(v_j.then.index, v_c.then.index) > index then
30                     update_reduced_nodes(index, min(v_j.then.index, v_c.then.index),
                         through_then, through_else);
31                 end
32             end
33             if Combine(v_j_then, c_j_then, reaches_one, through_then, through_else, visted) then
34                 through_else[index] = cierto;
35                 reaches_one[id] = cierto;
36                 if min(v_j.else.index, v_c.else.index) > index then
37                     update_reduced_nodes(index, min(v_j.else.index, v_c.else.index),
                         through_then, through_else);
38                 end
39             end
40         end
41     end
42     return it_reaches_the_one_terminal[id];
```

4.5.3 Heurística desarrollada

El Algoritmo 16 es la combinación de los Algoritmos 15 y 12. Éste, realiza la operación *AND* y a su vez calcula las variables mediante las cuales se llega a un nodo 1-terminal. Este algoritmo genera un identificador de nodo combinando los identificadores de los dos BDD's sobre los cuales se está realizando la operación *AND* (línea 2), y marca este nodo como visitado insertándolo en un diccionario que contiene todos los nodos visitados.

El caso trivial de este algoritmo ocurre cuando, los dos nodos son terminales (línea 2), si ambos nodos son el 1-terminal, su resultado, en consecuencia, el nodo 1-terminal (línea 4), en caso contrario el nodo 0-terminal (línea 6). En otras palabras, se está realizando la operación *AND* dos nodos, y sólo tomará valor cierto si ambos nodos son ciertos. El caso recursivo crea un nuevo nodo (línea 9) y desciende en ambos BDD's por las aristas *else* (línea 25) y *true* (línea 26), de este modo se identifican los nodos visitados. El caso trivial del algoritmo se produce cuando o bien los nodos accesibles desde las aristas $then$ o $else$ llevan ambas al nodo 1-terminal (líneas 7 y 8), en cuyo caso se realiza el mismo proceso que el Algoritmo 12. Para el caso recursivo primero se obtienen los nodos descendientes de ambos BDD's (líneas 22 a 35). Una vez obtenidos estos nodos se realiza la llamada recursiva.

El algoritmo ha requerido de las siguientes modificaciones:

1. Identificador concatenando los nodos de todos los BDD's (línea 2).

2. Índice mínimo de todos los BDD's (línea 3).

3. Test 1-terminal en todos los nodos (líneas 6,7 y 14).

4. Obtención de los siguientes nodos de todos los BDD's (líneas 22 a 25).

Una vez aplicado el Algoritmo 16, los algoritmos que se utilizan para calcular las características *dead* y *core* (Algoritmo 11) y los *impact* y *exclusion set*(Algoritmo 14) pueden ser aplicados de igual modo, ya que el algoritmo devuelve también como valores de salidas los *arrays* $through\_then$ y $through\_else$ que utilizan estos algoritmos.

Descomposición para más de 2 BDD's

El Algoritmo 16 puede ser aplicado también no sólo para la descomposición de dos BDD's si no también para k BDD. Para ello, únicamente es necesario modificar el algoritmo, utilizando un *array* de longitud k en el que almacenar cada uno de los nodos de los BDD's, este *array* se inicializaría con los nodos raíz, de cada BDD, y obtendríamos el menor índice, e iríamos descendiendo por todos los BDD's.

Las validaciones de nodos terminales se realizarán en todos los nodos terminales de los k BDD's, una vez llegados al final de camino.

Coste computacional

Para analizar el coste computacional del Algoritmo 16, con el Algoritmo 12 debemos tener en cuenta el coste de la operación AND. Tal y como indica Bryant [25] es $\mathcal{O}(m_j m_c)$, donde m_j y m_c es el número de nodos de los BDD's B_j y B_c respectivamente. Por tanto, el coste total del Algoritmo 12 es:

$$\mathcal{O}(m_j m_c) + \mathcal{O}(n' m_j m_c)$$

El coste de la operación AND tiene un coste asintótico, igual a la multiplicación del tamaño de los BDD's, al igual que el coste del Algoritmo 12. Por tanto, aplicar las dos operaciones en una operación conjunta como realiza el Algoritmo 16, debería reducir el coste del problema a la mitad, aun teniendo en cuenta que el Algoritmo 16 no aplica las reducciones **R1**, **R2** y **R3**. Pero, como se verá, en los resultados experimentales, esto no es así.

El coste de la operación AND, como indica Somenzi [129] en raras ocasiones es exponencial. En cambio, el coste del Algoritmo 12 implica el recorrido completo del BDD, que en el caso del Algoritmo 16 no tiene aplicadas las reducciones. Por tanto, si bien el coste nos da una orientación del coste real, es necesario la realización de una validación experimental.

Aplicación de heurísticas dinámicas

En el Capítulo 3, se demostró que aplicar una heurística dinámica, mejora el resultado a nivel de nodos. La heurística dinámica, puede aplicarse durante, o una vez

construido el BDD. El análisis experimental que se ha realizado combina, el momento de aplicación de la heurística dinámica, y los Algoritmos 12 y 16. La Tabla 4.10, muestra las combinaciones posibles, indicando si la construcción del BDD se realiza de forma concurrente (**Concurrente**) en dos hilos de ejecución, si utiliza el Algoritmo 12, o el 16 (**Agoritmo**), y si se aplica, o no la heurística de ordenación dinámica (**Heur. dinámica**), y cuando se aplica (**Cuando**),

id	Concurrente	BDD	Algoritmo	Heur. dinámica	Cuando
h1	No	ψ	Completo 12	No	Nunca
h2	No	ψ	Completo 12	Sí	Después
h3	No	ψ	Completo 12	Sí	Durante
h4	No	ψ_j y ψ_c	Descompuesto 16	No	Nunca
h5	No	ψ_j y ψ_c	Descompuesto 16	Sí	Después
h6	No	ψ_j y ψ_c	Descompuesto 16	Sí	Durante
h7	Sí	ψ_j y ψ_c	Descompuesto 16	No	Nunca

Tabla 4.10: Combinaciones de descomposición

El cálculo puede realizarse con el BDD completo Algoritmo 12 (h1), y aplicando una heurística dinámica después (h2) y durante de la construcción del BDD (h3). Los cálculos con el BDD descompuesto Algoritmo 16, pueden realizarse de forma no concurrente (h4) y aplicando una heurística dinámica después (h5) y durante la construcción del BDD (h6). Cuando se realiza de forma descompuesta (Algoritmo 16), utilizando dos BDD's en espacios de memoria diferente, no puede aplicarse ninguna ordenación dinámica (h7), dado que el Algoritmo 16, precisa que las variables tengan el mismo orden en cada uno de los BDD's.

Heurística / Modelo	h1	h2	h3	h4	h5	h6	h7
Dell_XML	6,6840	6,8890	2,5345	6,8249	6,9328	2,6114	7,3859
eCos_compat	0,001224	0,001266	0,001191	0,001052	0,001138	0,001022	0,001246
eCos_fs	0,001056	0,001038	0,001012	0,0008469	0,0008969	0,0008167	0,001094
eCos_infra	0,001435	0,001482	0,001396	0,001216	0,001342	0,001178	0,001323
eCos_io	0,008017	0,007937	0,007817	0,008505	0,007948	0,008287	0,01628
eCos_isoinfra	0,007654	0,007845	0,03604	0,006995	0,007262	0,03518	0,007077
eCos_kernel	0,004176	0,004171	0,004016	0,003861	0,004007	0,003718	0,004268
eCos_language	0,006024	0,006128	0,00579	0,006065	0,006083	0,005833	0,01111
eCos_net	0,007308	0,00673	0,007116	0,007486	0,006809	0,007299	0,0117
eCos_redboot	0,0039	0,004159	0,003783	0,003566	0,003976	0,003459	0,002689
eCos_services	0,00366	0,003701	0,003546	0,003521	0,003595	0,003408	0,004893
LinuxKernel_arch	0,006892	0,007062	0,006676	0,00667	0,006874	0,006464	0,004444
LinuxKernel_block	0,003428	0,003487	0,00332	0,003277	0,003339	0,003182	0,003183
LinuxKernel_crypto	0,2055	0,1188	0,6334	0,1977	0,1139	0,6319	0,6075
LinuxKernel_fs	8,4383	5,7094	1,9392	8,7552	5,7055	1,9342	23,1666
LinuxKernel_lib	0,007105	0,00741	0,0764	0,006731	0,006993	0,07584	0,007106
LinuxKernel_mm	0,004897	0,004969	0,004735	0,00471	0,004768	0,004552	0,003002
LinuxKernel_security	0,01076	0,00825	0,01041	0,009738	0,007938	0,009407	0,02516
BankingSoftware	0,008517	0,009045	0,008303	0,007969	0,008531	0,007763	0,009243
HIS	0,004502	0,004482	0,004368	0,004176	0,004232	0,004045	0,00535
Documentation_Generation	0,006461	0,004157	0,006195	0,00524	0,003756	0,005001	0,0181
Graph	0,002105	0,002097	0,002033	0,001892	0,001987	0,001823	0,002004
Web_Portal	0,003927	0,003753	0,00381	0,003538	0,003452	0,003424	0,00726
Electronic Shopping	0,1982	0,06019	0,5899	0,1950	0,07198	0,6011	1,2376

Tabla 4.11: Resultado de tiempo en segundos heurísticas combinadas

Heurística / Modelo	h1	h2	h3	h4	h5	h6	h7
Dell_XML	7138	3186	8330	46099	10780	17690	46099
eCos_compat	32	30	32	36	34	36	36
eCos_fs	27	26	27	28	27	28	28
eCos_infra	34	29	34	35	33	35	35
eCos_io	311	249	311	334	250	334	334
eCos_isoinfra	160	159	156	161	161	160	161
eCos_kernel	113	79	113	116	81	116	116
eCos_language	262	239	262	267	224	267	267
eCos_net	299	152	299	325	205	325	325
eCos_redboot	69	68	69	72	71	72	72
eCos_services	129	117	129	133	121	133	133
LinuxKernel_arch	93	57	93	94	58	94	94
LinuxKernel_block	87	72	87	93	75	93	93
LinuxKernel_crypto	14866	2029	2532	14822	2031	2512	14822
LinuxKernel_fs	396879	4538	6396	341935	2564	3900	341935
LinuxKernel_lib	197	177	296	195	171	224	195
LinuxKernel_mm	79	55	79	70	47	70	70
LinuxKernel_security	711	273	711	517	248	517	517
BankingSoftware	242	216	242	244	221	244	244
HIS	155	117	155	126	96	126	126
Documentation_Generation	531	177	531	154	121	154	154
Graph	57	48	57	57	55	57	57
Web_Portal	217	149	217	131	98	131	131
Electronic Shopping	25275	4903	24345	2998	1584	2011	2998

Tabla 4.12: Resultado del número de nodos heurísticas combinadas

V.p / V.p	h1	h2	h3	h4	h5	h6	h7
h1	-	0,999223	0,680078	0,022745	0,088678	0,000904	0,999680
h2	0,999223	-	0,917132	0,088678	0,257773	0,005715	0,977160
h3	0,680078	0,917132	-	0,680078	0,917132	0,169424	0,411687
h4	0,022745	0,088678	0,680078	-	0,999223	0,977160	0,005715
h5	0,088678	0,257773	0,917132	0,999223	-	0,834923	0,028099
h6	0,000904	0,005715	0,169424	0,977160	0,834923	-	0,000154
h7	0,999680	0,977160	0,411687	0,005715	0,028099	0,000154	-

Tabla 4.13: Valor de p Test de Nemenyi del tiempo

4.5.4 Resultados experimentales

Las Tablas 4.16, y 4.15 muestran el tiempo promedio en segundos de la construcción del BDD juntamente con el cálculo de las características *core* y *dead*, y el número de nodos total respectivamente de las heurísticas de la Tabla 4.10. Los resultados se han realizado con los modelos de configuración del *benchmark*, utilizando la heurística determinista estática FORCE [3], en un procesador Intel Core i5-3230M a 2.8 GHz, con 8 GB de RAM, y 50 ejecuciones para evitar interferencias. Los datos completos de las Tablas 4.11 y 4.12 están disponibles también GitHub[1].

Para la aplicación de los Algoritmos 16, se ha utilizado la descomposición entre la función de las restricciones jerárquicas (ψ_j) y las *crosstree* (ψ_c). Es por ello, que se han excluido del *benchmark* los modelos **Smart Home v2**[1], **Assess4me**[30], **SmartTV** [62], **Model_Transformation** [38] y **Thread** [17], ya que no tienen restricciones *crosstree*.

Las Tablas 4.13 y 4.14 muestran el resultado test de Nemenyi [101] para los resultados de las Tablas 4.16 y 4.15 respectivamente.

Comparativa heurísticas h1 (Algoritmo 12) y h4 (Algoritmo 16)

Si comparamos los resultados de h1 (Algoritmo 12) y h4 (Algoritmo 16), sin aplicar heurísticas de ordenación dinámica, a nivel de tiempo vemos que, en promedio, son similares 0,651 versus 0,670 segundos. Pero en cambio, como muestra la Tabla 4.16, si existen diferencias estadísticamente significativas, con un valor de p, por debajo del valor de referencia $0,05$. La Figura 4.10, muestra el histograma de

[1] https://github.com/robcbean/thesis_data/tree/master/decomp

V.p / V.p	h1	h2	h3	h4	h5	h6	h7
h1	-	0,000329	0,999492	0,917132	0,001796	0,998341	0,917132
h2	0,000329	-	0,002052	1,079e-06	0,999680	2,622e-05	1,079e-06
h3	0,999492	0,002052	-	0,701554	0,009250	0,960668	0,701554
h4	0,917132	1,079e-06	0,701554	-	9,425e-06	0,996793	1,000000
h5	0,001796	0,999680	0,009250	9,425e-06	-	0,000180	9,425e-06
h6	0,998341	2,622e-05	0,960668	0,996793	0,000180	-	0,996793
h7	0,917132	1,079e-06	0,701554	1,000000	9,425e-06	0,996793	-

Tabla 4.14: Valor p test Nemenyi del número de nodos

Heurística	Segundos
h3	0,245623
h6	0,2487637
h2	0,5365232
h5	0,5382955
h1	0,6512082
h4	0,6695775
h7	1,356004

Tabla 4.15: Promedio de nodos

Heurística	Nº de nodos
h2	714
h5	807
h6	1222
h3	1896
h4	17043
h7	17043
h1	18665

Tabla 4.16: Promedio de tiempo (s)

diferencia de tiempo entre la heurística h1 y h4, en el que apreciamos que la diferencia de tiempo tiende a ser positiva. Es decir, el Algoritmo 16 es más lento, en la mayoría de los casos, como confirma también su estadística descriptiva, que si bien la media de la diferencia de tiempo es negativa $-0,1837$, su mediana es positiva $0,0002$.

Si analizamos a nivel de número de nodos los resultados h1 y h4, el número de nodos es menor para el Algoritmo 16 17043 respecto a 18665. Pero no existe mucha diferencia, como confirma el valor de p del test Nemeny, que no detecta diferencias estadísticamente significativas.

Podemos concluir, que aplicar el Algoritmo 16 obtiene peores resultados a nivel de tiempo, y resultados similares a nivel de nodos.

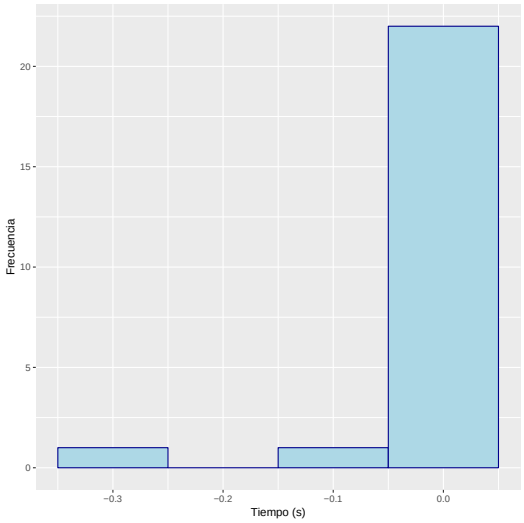


Figura 4.10: Histograma diferencia de tiempo en segundos entre h1 y h4

Heurísticas dinámicas h2 (Algoritmo 12) y h5 (Algoritmo 16)

La aplicación de la heurística de reordenación dinámica de *sifting* [114] después de construir el BDD, combinada con los Algoritmos 12(h2) y 16(h5), no presentan diferencias sustanciales tanto a nivel de número de nodos 714 versus 807, como a nivel de tiempo $0,5365$ versus $0,5383$ segundos. El test de Nemeny obviene valores de p superiores al límite de referencia $0,05$, confirmando que no existen diferencias significativas.

Heurísticas dinámicas h3 (Algoritmo 12) y h6 (Algoritmo 16)

De igual modo heurística de reordenación dinámica aplicada durante la construcción del BDD, su aplicando durante la construcción, combinada con los Algoritmos 12 (h3) y 16 (h6), no identifica diferencias estadísticamente significativas, con valores de p, por debajo del nivel de referencia $0,05$. A nivel descriptivo, tampoco existen diferencias ni a nivel de tiempo ($0,245623$ versus $0,248737$), ni de número de nodos (1222 versus 1896).

La combinación de heurísticas de ordenación, dinámica juntamente con la aplicación del Algoritmo 16, no mejora los resultados, ni a nivel de tiempo, ni de nodos, respecto al Algoritmo 12.

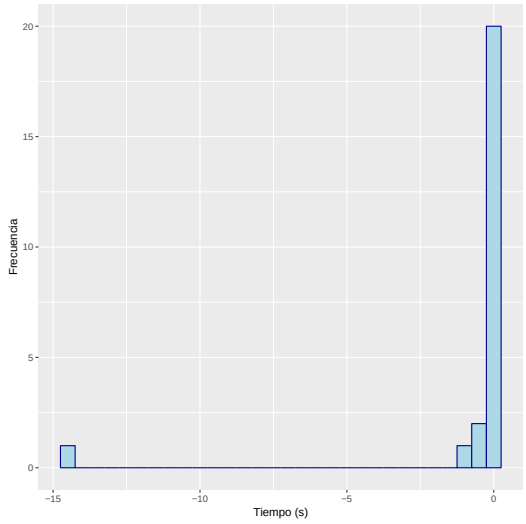


Figura 4.11: Histograma de las diferencias de tiempo entre h1 y h7

Heurística h7 (Algoritmo 16 aplicado de forma concurrente)

La construcción de modo concurrente, de cada uno de los sub-BDD's, no presenta tampoco diferencias estadísticamente significativas, ni a nivel de tiempo, ni a nivel de nodos, con el Algoritmo 12 (h1), como muestra los valores de p de las Tablas 4.16 y 4.15, con valores de p mayores que el nivel de referencia. Si bien es cierto, que, si analizamos los datos promedio, el tiempo empeora bastante $0,651$ versus $1,356$ segundos. Pero, es debido únicamente, a la diferencia significativa que existe con el modelo **LinuxKernel_fs**, con valor de 15 segundos, como muestra el histograma de la Figura 4.11.

4.5.5 Conclusiones

El Algoritmo 16, no mejora el Algoritmo 12, ni el tiempo, ni en número de nodos, corroborando con lo indicado por Somenzi [129]. El coste de la operación `AND`, no es exponencial, en la mayoría de los casos. En consecuencia, los costes son similares. Y, por otra parte, el número de nodos no decrece, por qué no se aplican las reducciones **R1**, **R2** y **R3**.

La ventaja, que aporta el Algoritmo 16, es la descomposición del problema, cuando este es muy grande. La heurística h7, permite distribuir el problema en diferentes procesos, en contraposición, no pueden aplicarse heurísticas, de reorde-

nación dinámica, puesto que el algoritmo requiere que todas las variables, en los sub-BDD's tengan el mismo orden.

4.6 Eliminación de características *dead* y *dead*

4.6.1 Objetivo

Hemos desarrollado una heurística que calcula características *core* y *dead* utilizando sub-BDD's, y puede distribuirse en diferentes computadores, ejecutándose concurrente, en vez del BDD completo. Estas características pueden ser eliminadas del modelo de configuración, reduciendo el problema, sin afectar a su variabilidad.

Primero mostraremos los fundamentos teóricos, y la heurística de cálculo de características *core* y *dead* en sub-BDD's, y el algoritmo que elimina estas características de los modelos de configuración.

A continuación, mostraremos los resultados experimentales, de reducción tanto a nivel nodos como de tiempo, de la eliminación de características *core* y *dead* de los modelos de configuración, junto con su análisis estadístico.

4.6.2 Fundamentos teóricos del cálculo características *core* y *dead*

Lema 7. *Derivación de las características* dead *de un BDD de las* dead *de sus sub-BDD's.*

Dada una función $\psi = \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k$, descompuesta en k sub-BDD's $B = B_1 \wedge B_2 \wedge \ldots \wedge B_k$, si una característica f es dead *en algún sub-BDD $B_i \Rightarrow$ f es* dead *en el BDD completo B.*

Demostración 1. *f es* dead *en $\psi \Leftrightarrow f \wedge \psi$ es insatisfacible ($f \wedge \psi$ es falso siempre, para cualquier asignación). Del mismo modo, f es* dead *en $\psi_i \Leftrightarrow f \wedge \psi_i$ es insatisfacible (es falso siempre).*

$$\begin{aligned} f \wedge \psi_i \text{ falso} &\Rightarrow (f \wedge \psi_1) \wedge (f \wedge \psi_2) \wedge \ldots \wedge (f \wedge \psi_k) \text{ falso} \\ &\Rightarrow f \wedge \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k \text{ falso} \\ &\Rightarrow f \wedge \psi \text{ falso} \end{aligned} \tag{4.8}$$

Observación 1. *La implicación en el sentido contrario no se cumple: f puede ser* dead *en ψ, y no serlo ningún ψ_i parcial.*

Lema 8. *Derivación de las características* core *de un BDD de las* core *de sus sub-BDD's.*

Dada una función $\psi = \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k$, descompuesta en k sub-BDD's $B = B_1 \wedge B_2 \wedge \ldots \wedge B_k$, si una característica f es core *en algún sub-BDD $B_i \Rightarrow$ f es* core *en el BDD completo B.*

Demostración 2. *f es* core *en $\psi \Leftrightarrow \neg f \wedge \psi$ es insatisfacible ($\neg f \wedge \psi$ es falso siempre, para cualquier asignación). Del mismo modo, f es* core *en $\psi_i \Leftrightarrow \neg f \wedge \psi_i$ es insatisfacible (es falso siempre).*

$$\begin{aligned} \neg f \wedge \psi_i \text{ falso} &\Rightarrow (\neg f \wedge \psi_1) \wedge (\neg f \wedge \psi_2) \wedge \ldots \wedge (\neg f \wedge \psi_k) \text{ falso} \\ &\Rightarrow \neg f \wedge \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_k \text{ falso} \\ &\Rightarrow \neg f \wedge \psi \text{ falso} \end{aligned} \tag{4.9}$$

Observación 2. *La implicación en el sentido contrario no se cumple: f puede ser* core *en ψ, y no serlo ningún ψ_i parcial.*

4.6.3 Cálculo de características *core* y *dead* en sub-BDD's

La heurística desarrollada está formada por dos algoritmos el primero de ellos utiliza los lemas 8 y 7, para calcular las características *core* y *dead* de forma concurrente.

El segundo algoritmo una vez obtenida la lista de estas características *core* y *dead*, las elimina del modelo, y se vuelven a aplicar los lemas 8 y 7, de forma recurrente, mientras el primer algoritmo, encuentren características *core* y *dead*.

Aplicación de los lemas 8 y 7

Si podemos calcular individualmente las características *core* y *dead* de cada uno de los sub-BDD's, el problema puede distribuirse, y calcularse de forma concurrente.

El algoritmo concurrente ha sido implementado utilizando la librería **Open MPI** [59], accesible en el repositorio de GitHub[1]. En dicha librearía, a cada proceso concurrente se le asigna un número que se denomina rango (*rank*).

El Algoritmo 17, que distribuye la función ψ, en k sub-funciones necesita $k+1$ procesos concurrentes:

[1] https://github.com/robcbean/test_bdd_heuristic_concur

- Un proceso coordinador que recibirá la información del resto de procesos, y realizará el calculo global (líneas 5-14). Este proceso, verifica no se haya recibido previamente la información de cada uno de los procesos concurrentes (línea 7). En caso de no haberla recibido, revisa si ha enviado, y la almacena (líneas 8-11). Finalmente, cuando recibe la información de todos los procesos, calcula las características *core*/*dead* globales, utilizando el Algoritmo 18.

- Y k procesos, que realizarán los siguientes cálculos de forma concurrente:

  1. Construir el sub-BDD, BDD_i (línea 17).
  2. Calcular la función $reach\_one$ para el BDD_i (líneas 18).
  3. Enviar el resultado de la función $reach\_one$ (línea 19).

Algoritmo 17: Algoritmo concurrente para el cálculo *core* y *dead*

```
1  Function dead_core_concurrent( ψ = ψ1 ∧ ψ2 ∧ ... ψk, nfunc : Integer, nvars : Integer )
2      reach_one_array : array[k];
3      rank = get_rank_id() ;
4      if rank == 0 then
5          for i = 1; ≤ nfunc; i = i + 1 do
6              reach_one_array[i] = NULL;
7          end
8          repeat
9              for i = 1; i ≤ nfunc; i = i + 1 do
10                 if reach_one_array[i] == NULL then
11                     reach_one = async_receive_reach_one(i);
12                     if reach_one ≠ NULL then
13                         reach_one_array[i] = reach_one;
14                     end
15                 end
16             end
17         until is_read_all_reach_one(reach_one_array);
18         (dead, core) = dead_core_multiple_functions(reach_one_array, nfunc, nvars);
19     else
20         bdd = compute_bdd_function(ψ_rank);
21         reach_one = compute_compute_reach_one(bdd);
22         send_async_reach_one(reach_one, 0);
23     end
```

El Algoritmo 18, una vez obtenidos los valores de $reach\_one$ para cada una de las funciones calcula las características *core* y *dead*, de acuerdo con lo que indican los lemas 7 y 8, es decir, si una variable es *core*/*dead* deberá acceder al menos

al nodo 1-terminal/0-terminal en una función, y no acceder al nodo 0-terminal/1-terminal en ninguna función.

Algoritmo 18: Algoritmo cálculo *core* y *dead* para array

```
1  Function dead_core_multiple_functions( reach_one_array : array[k], nfunc : Integer, nvars : Integer )
2      dead = ∅;
3      core = ∅;
4      for i = 1; i ≤ nvars; i = i + 1 do
5          high = Σ_{k=0}^{nfunc} reach_one_array[k].var_high[i];
6          low = Σ_{k=0}^{nfunc} reach_one_array[k].var_low[i];
7          if high > 0 ∧ low = 0 then
8              core = core ∪ x_i ;
9          else
10             if low > 0 ∧ high = 0 then
11                 dead = dead ∪ x_i;
12             end
13         end
14     end
15     return (core, dead)
```

Cálculo de *dead* y *core* en varios BDD's ordenes diferentes

El Algoritmo 16, precisa que todos los sub-BDD's, tengan la misma ordenación de variables, dado que comparten el espacio de memoria. En consecuencia, no pueden aplicarse algoritmos de ordenación dinámica. En cambio, el Algoritmo 18, puede aplicar ordenaciones de variables diferente a cada sub-BDD. Puesto, que los paquetes de BDD's como CUDD [130], guardan para cada variable x_i del BDD su orden inicial, tras una reordenación dinámica, eso permite a los paquetes referenciar a dicha variable con el mismo índice, de modo que, en caso de reordenación, el usuario no se vea afectado.

4.6.4 Fundamentos teóricos de la reducción del problema

Lema 9. *Si una característica f es* core *en un modelo de configuración M_c, codificado mediante un diagrama de características, esta puede ser eliminada.*

Si la característica f, sólo está en las restricciones jerárquicas, se eliminará directamente, y las características hijas subirán de nivel, puesto que su característica padre siempre será cierta.

La relación, jerárquica de una característica f con su característica hija h viene definida por la cláusula de la Ecuación 4.10, que se encuentra en todas las conversiones a lógica proposicional, tal y como se indicó en el Capítulo 1.

$$\neg h \vee f \tag{4.10}$$

Dado, que f siempre es cierta, la Ecuación 4.10 siempre, se evaluará a cierto, en consecuencia, puede ser eliminada.

Si la característica h es obligatoria, se añade la restricción de la Ecuación 4.11, que sólo dependerá de la característica, hija, que será obligatoria, por tanto, podemos eliminar la característica f, del modelo.

$$\neg f \vee h \tag{4.11}$$

En el caso de que la característica f, esté incluida en una restricción *crosstree*, codificada en CNF, sin pérdida de generalidad[133], podemos eliminar la restricción, ya que siempre tendrá valor cierto.

Si en la restricción *crosstree*, la característica f aparece negada, eliminaremos la característica f, de la restricción, dado que su valor sólo dependerá, del resto de características.

Lema 10. *Si una característica f es* dead *en un modelo de configuración M_c, codificado mediante un diagrama de características, sin pérdida de generalidad, f puede ser eliminada y todas sus características hijas.*

Las restricciones de la relación jerárquica entre características vienen definidas, por la Ecuación 4.10, en consecuencia, si f es falsa, h siempre será deberá, tomar valor falso, para hacer la ecuación cierta, y puede ser eliminada de acuerdo con el Lema 10, ya que h se convierte en una característica *dead*.

Si la característica f está negada en una restricción *crosstree* como CNF, siempre será cierta. Por tanto, puede eliminarse, de igual modo a como se realiza cuando f es *core*.

Si f, no está negada en la restricción *crosstree*, la característica f puede eliminarse de la restricción, dado que su valor sólo dependerá del resto de características.

4.6.5 Heurística de reducción del problema

La heurística parte de una función $\psi = \psi_j \wedge \psi_c$ de un modelo de configuración M_c definido mediante un diagrama de características, donde ψ_j son las restricciones jerárquicas del modelo, y ψ_c las restricciones *crosstree*.

El Algoritmo 12 recibe como parámetro un modelo de configuración M_c, calcula la función correspondiente a las restricciones jerárquicas ψ_j, y *crosstree* ψ_c ,y ejecuta el Algoritmo 18, con el que obtiene características *core* y *dead* (líneas 2 a 5). Si obtiene características *core* y *dead* (línea 6), las elimina del modelo (línea 7), aplicación de los Lemas 9 y 10, y vuelve aplicar el Algoritmo 12, de modo recurrente (línea 8).

Algoritmo 19: Algoritmo de reducción del problema

```
1  Function reduce_feature_model( M_c : model)
2      ψ_j = calculate_hierchaly_function(M_c);
3      ψ_c = calculate_crosstree_function(M_c);
4      n_vars = number_of_features(M_c);
5      (dead, core) = dead_core_concurrent(ψ = ψ_j ∧ ψ_c, 2, n_vars);
6      if dead ∩ core ≠ ∅ then
7          M_r = remove_features(M_c, dead, core);
8          M_r = reduce_feature_mode(M_r);
9      else
10         M_r = M_c;
11     end
12     return (M_r)
```

4.6.6 Resultados experimentales

Los modelos *benchmark* que incluyen características *core*/*dead*, junto con *crosstree*, sólo son: **BankingSoftware** [96], **Documentation_Generation** [136], **Graph** [118], **HIS** [81], **Electronic Shopping** [110] y **Web_Portal** [93]. Con el objetivo de validar la heurística, se han incluido los modelos de la Tabla **??**, disponible en el repositorio GitHub[1].

Las Tabla 4.18 y 4.19 muestran los resultados obtenidos, tanto nivel de nodos y tiempo respectivamente, del modelo original, y el modelo reducido, habiendo aplicado el Algoritmo 12, así como el número de nodos. Los resultados están ac-

[1] https://github.com/robcbean/thesis_data/tree/master/modelos

Modelo	Origen	Nº de características
32_141	Generado aleatoriamente con SPLAR [92]	79
10_103	Generado aleatoriamente con SPLAR [92]	103
68_108	Generado aleatoriamente con SPLAR [92]	108
23_111	Generado aleatoriamente con SPLAR [92]	111
25_112	Generado aleatoriamente con SPLAR [92]	112
35_119	Generado aleatoriamente con SPLAR [92]	119
34_120	Generado aleatoriamente con SPLAR [92]	120
27_127	Generado aleatoriamente con SPLAR [92]	127
15_134	Generado aleatoriamente con SPLAR [92]	134
36_139	Generado aleatoriamente con SPLAR [92]	139
1_140	Generado aleatoriamente con SPLAR [92]	140
49_141	Generado aleatoriamente con SPLAR [92]	141
46_79	Generado aleatoriamente con SPLAR [92]	141
62_175	Generado aleatoriamente con SPLAR [92]	175
SPLOT-3CNF-FM-500-50-1.00-SAT-1	Modelo aleatorio SPLOT [92]	500
SPLOT-3CNF-FM-500-50-1.00-SAT-10	Modelo aleatorio SPLOT [92]	500
SPLOT-3CNF-FM-500-50-1.00-SAT-2	Modelo aleatorio SPLOT [92]	500
SPLOT-3CNF-FM-500-50-1.00-SAT-3	Modelo aleatorio SPLOT [92]	500
SPLOT-3CNF-FM-500-50-1.00-SAT-4	Modelo aleatorio SPLOT [92]	500
SPLOT-3CNF-FM-500-50-1.00-SAT-5	Modelo aleatorio SPLOT [92]	500
SPLOT-3CNF-FM-500-50-1.00-SAT-6	Modelo aleatorio SPLOT [92]	500
SPLOT-3CNF-FM-500-50-1.00-SAT-7	Modelo aleatorio SPLOT [92]	500
SPLOT-3CNF-FM-500-50-1.00-SAT-9	Modelo aleatorio SPLOT [92]	500
busybox-1.18.0	busybox [138]	846

Tabla 4.17: Modelos con *dead*, *core* y *crosstree*

cesibles también el repositorio GitHub[1]. Se ha aplicado la heurística de ordenación estática, **malik_fanin** [88].

El tiempo, es el promedio de 50 ejecuciones en un procesador Intel Xeon E5-2660, con 256 GB de RAM. Como podemos observar, en sólo 6 de los 30 modelos de configuración aumenta el número de nodos, y en 7 de los 30 aumentan el tiempo. Por, lo que parece indica, que la heurística mejora los resultados, tanto a nivel de nodos como a nivel de tiempo.

[1] https://github.com/robcbean/thesis_data/tree/master/reduccion

	Modelo completo			Modelo reducido	
Modelo	**Nº de carac.**	**Nº nodos**	**Reduc**	**Nº de carac.**	**Nº de nodos**
BankingSoftware	176	237	1	160	218
HIS	67	279	1	31	139
Web_Portal	43	335	1	40	224
Electronic Shopping	290	73202	1	261	22623
Documentation_Generation	44	457	1	37	423
Graph	30	60	1	20	58
SPLOT-3CNF-FM-500-50-1.00-SAT-4	500	146528	1	490	146504
SPLOT-3CNF-FM-500-50-1.00-SAT-6	500	643761	1	492	1186644
SPLOT-3CNF-FM-500-50-1.00-SAT-10	500	787441	1	481	607088
SPLOT-3CNF-FM-500-50-1.00-SAT-2	500	478626	1	494	463701
SPLOT-3CNF-FM-500-50-1.00-SAT-7	500	201024	1	481	196444
SPLOT-3CNF-FM-500-50-1.00-SAT-5	500	1692416	1	491	1147508
SPLOT-3CNF-FM-500-50-1.00-SAT-1	500	5833291	1	477	5770315
SPLOT-3CNF-FM-500-50-1.00-SAT-3	500	6215038	1	477	8360215
SPLOT-3CNF-FM-500-50-1.00-SAT-9	500	535500	1	498	205439
25_112	112	1606	1	107	1598
34_120	120	10109	1	115	16461
27_127	127	2079	1	125	2069
46_79	141	72123	2	64	54971
1_140	140	9111727	1	137	6926370
49_141	141	10171	1	128	10151
68_108	108	909	1	98	788
62_175	175	692	1	133	666
36_139	139	1393	1	128	1569
35_119	119	180	1	96	156
32_141	141	61329	2	106	38109
10_103	103	293	1	73	603
15_134	134	284	1	123	278
23_111	111	1176	1	110	1168
busybox-1.18.0	846	51760	1	833	108577
Total	**7745**	**25934026**	**7306**		**25271077**

Tabla 4.18: Resultado del número de nodos de la heurística de reducción

Modelo	Nº de características	T. antes(s)	T. después (s)
BankingSoftware	176	0,00650622	0,0043772
HIS	67	0,00264516	0,00112953
Web_Portal	43	0,00138988	0,00178022
Electronic Shopping	290	0,0323693	0,018598
Documentation_Generation	44	0,00347276	0,00154327
Graph	30	0,0017572	0,000819686
SPLOT-3CNF-FM-500-50-1.00-SAT-4	500	0,679606	0,669368
SPLOT-3CNF-FM-500-50-1.00-SAT-6	500	3,240079	6,561972
SPLOT-3CNF-FM-500-50-1.00-SAT-10	500	5,583173	2,330873
SPLOT-3CNF-FM-500-50-1.00-SAT-2	500	9,803040	9,594019
SPLOT-3CNF-FM-500-50-1.00-SAT-7	500	1,871247	0,919073
SPLOT-3CNF-FM-500-50-1.00-SAT-5	500	3,850569	1,451278
SPLOT-3CNF-FM-500-50-1.00-SAT-1	500	4,112720	4,100080
SPLOT-3CNF-FM-500-50-1.00-SAT-3	500	7,831532	6,158807
SPLOT-3CNF-FM-500-50-1.00-SAT-9	500	24,322480	8,286561
25_112	112	0,913373	0,0226635
34_120	120	0,00644249	0,00614445
27_127	127	0,00534869	0,00598141
46_79	79	7271,143725	2,182188
1_140	140	126,958098	30,908453
49_141	141	19,587473	0,508821
68_108	108	0,00653767	0,00504088
62_175	175	0,00699406	0,00459769
36_139	139	0,00627259	0,00637647
35_119	119	0,00728524	0,00495431
32_141	141	16147,600000	0,368340
10_103	103	0,00504125	0,0028679
15_134	134	0,00766506	0,00652969
23_111	111	0,00610157	0,0063962
busybox-1.18.0	846	16,078024	32,816388
Total		**23643,680968**	**106,956023**

Tabla 4.19: Resultado de tiempo de la heurística de reducción

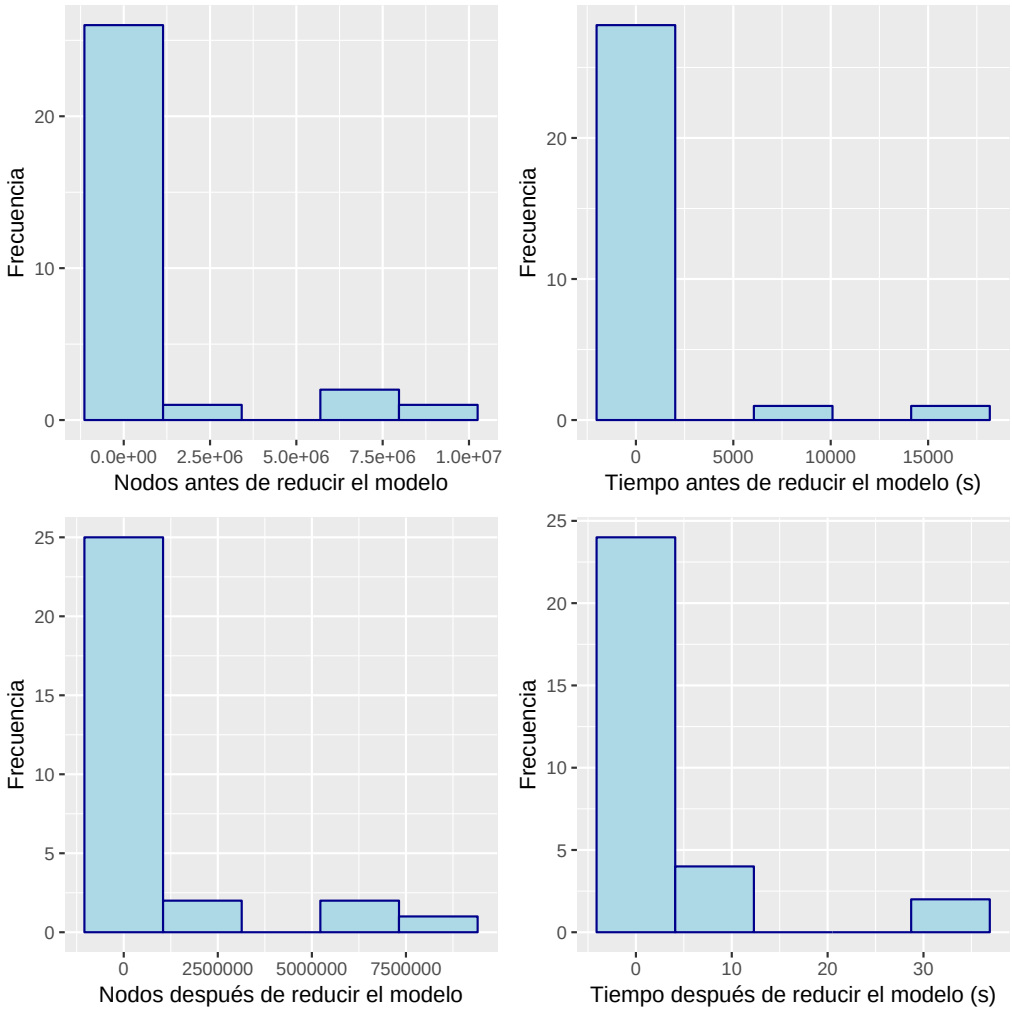


Figura 4.12: Histogramas del nº de nodos y tiempo de construcción del BDD

Transcripción 17

```
Shapiro-Wilk normality test
data:  caracteristicas_antes_reduccion
W = 0.79642, p-value = 5.646e-05
data:  caracteristicas_despues_reduccion
W = 0.79829, p-value = 6.102e-05
```

Para verificar el resultado, se ha realizado como en las secciones anteriores un análisis estadístico. Primero, hemos verificado que existen diferencias significativas aplicando el Test Conover [33], ya que no siguen una distribución normal como muestra el Test Sapiro-Wilk que muestra la Transcripción 17, al número de características antes, y después de aplicar la reducción, cuyo valor de p es menor al límite de referencia $0,05$, tal y como indica la Transcripción 18.

Transcripción 18

```
Pairwise comparisons using Conover's test for a two-way
balanced complete block design
data:  datos_friedman
1
1 <2e-16
```

Una vez verificada la reducción, se ha verificado, el número de nodos del BDD, y el tiempo, antes y después de aplicar la reducción no siguen una distribución normal, como se observa en la Figura 4.12, y confirma el test de Shapiro-Wilk [122], cuyo valor p, está por debajo del límite de referencia, $0,05$, en consecuencia, no podríamos aplicar un modelo parámetrico.

Transcripción 19

```
Shapiro-Wilk normality test
data:  datos$nodos_antes
W = 0.45607, p-value = 1.895e-09

data:  datos$nodos_despues
W = 0.44697, p-value = 1.539e-09

data:  datos$tiempo_antes
W = 0.27209, p-value =  4.137e-11

data:  datos$tiempo_despues
W = 0.49168, p-value = 4.381e-09
```

Al igual que se ha realizado en el Capítulo 3, se ha aplicado el logaritmo en base 2 al número de nodos del BDD, y al tiempo de construcción, obteniendo así una

mejor aproximación a la normalidad tal y como muestra la Figura 4.13 y los resultados del test Shapiro-Wilk, que obtienen valores de p mayores, pero sólo el logaritmo del número de nodos de después de aplicar la reducción, cumple con el mínimo exigible. En consecuencia, el análisis realizado ha sido no paramétrico.

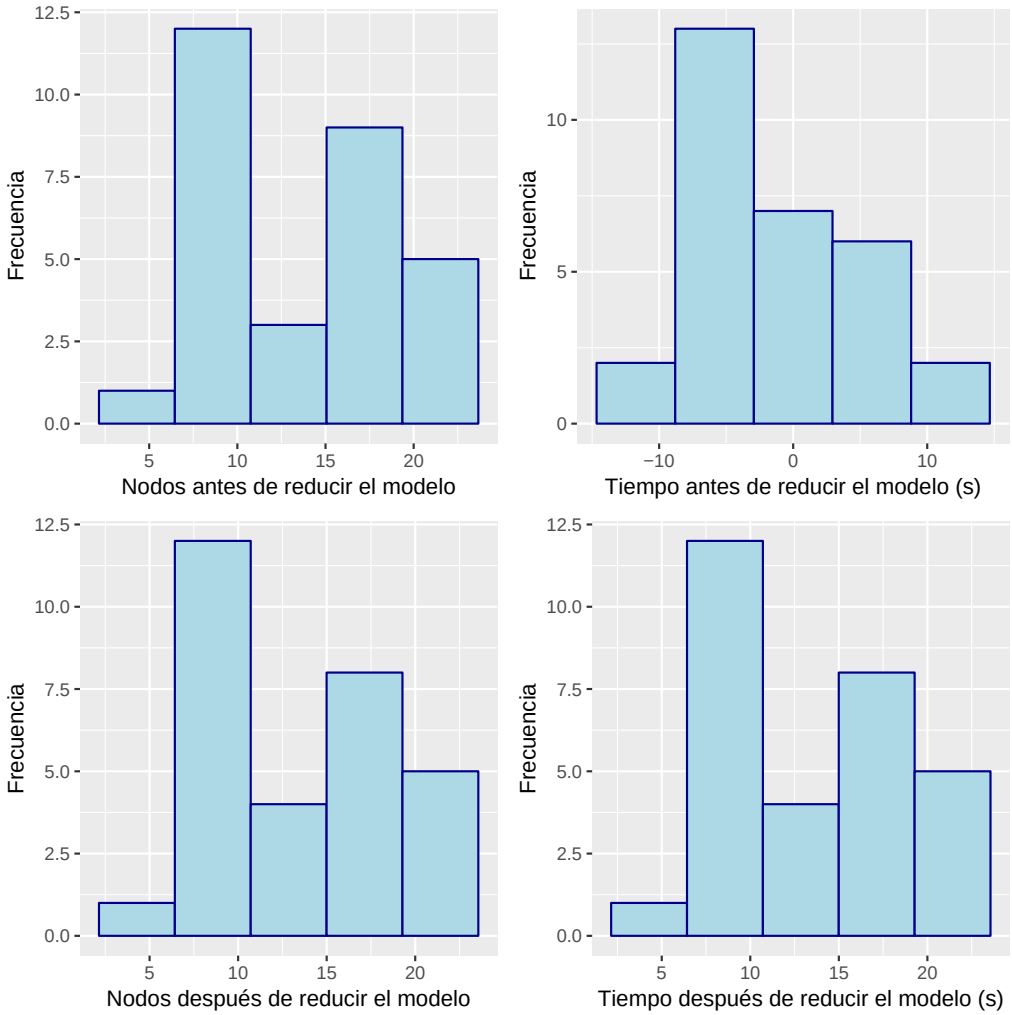


Figura 4.13: Histogramas del log. base 2 del nº de nodos y tiempo de construcción del BDD

Transcripción 20

```
Shapiro-Wilk normality test

data:  datos$log_nodos_antes
W = 0.92093, p-value = 0.02835
```

```
data:  datos$log_nodos_despues
W = 0.93149, p-value = 0.05379

data:  datos$log_tiempo_antes
W = 0.87729, p-value = 0.00177

data:  datos$log_tiempo_despues
W = 0.87838, p-value = 0.002184
```

4.6.6.1 Identificación de la diferencia de resultados

El modelo de análisis a realizar al igual que en los anteriormente, es un modelo de medidas repetidas, aplicado a únicamente a dos medidas repetidas en ese caso, antes y después de aplicar la reducción. Por lo que en vez de aplicar el Test de Friedman [54], hemos aplicado el Test Conover and Iman [33]. Cuyo, resultado tanto a nivel de nodos como a nivel de tiempos es estadísticamente significativo con valores de p inferiores al límite de referencia $0,05$.

Transcripción 21

```
  Pairwise comparisons using Conover's test for a two-way
 balanced complete block design

 data:  datos_friedman$nodos
  1
 1 2.5e-14

 data:  datos_friedman$tiempo
  1
1 2.5e-14
```

Para verificar, que existe una reducción del tanto se ha realizado un análisis de regresión lineal utilizando el logaritmo en base dos, tanto de los nodos como del tiempo. Ya que como se muestra la imagen existe una relación lineal como muestra la Figura 4.14, y el coeficiente de correlación de Pearson, cercano a $0,9$ en ambos casos. Hay que tener en cuenta, que, en el caso del tiempo, influye también el tiempo de proceso, que, aunque sea el resultado del promedio de 50 ejecuciones, puede verse afectado.

Con el objetivo de verificar la reducción, tanto del tiempo se realizó un modelo de regresión lineal con las siguientes ecuaciones, donde $|B_r|$ es el número de nodos del BDD reducido, y $|B|$ el número de nodos antes de la reducción. Y T_r el tiempo de construcción del BDD reducido y T el tiempo de antes de la reducción.

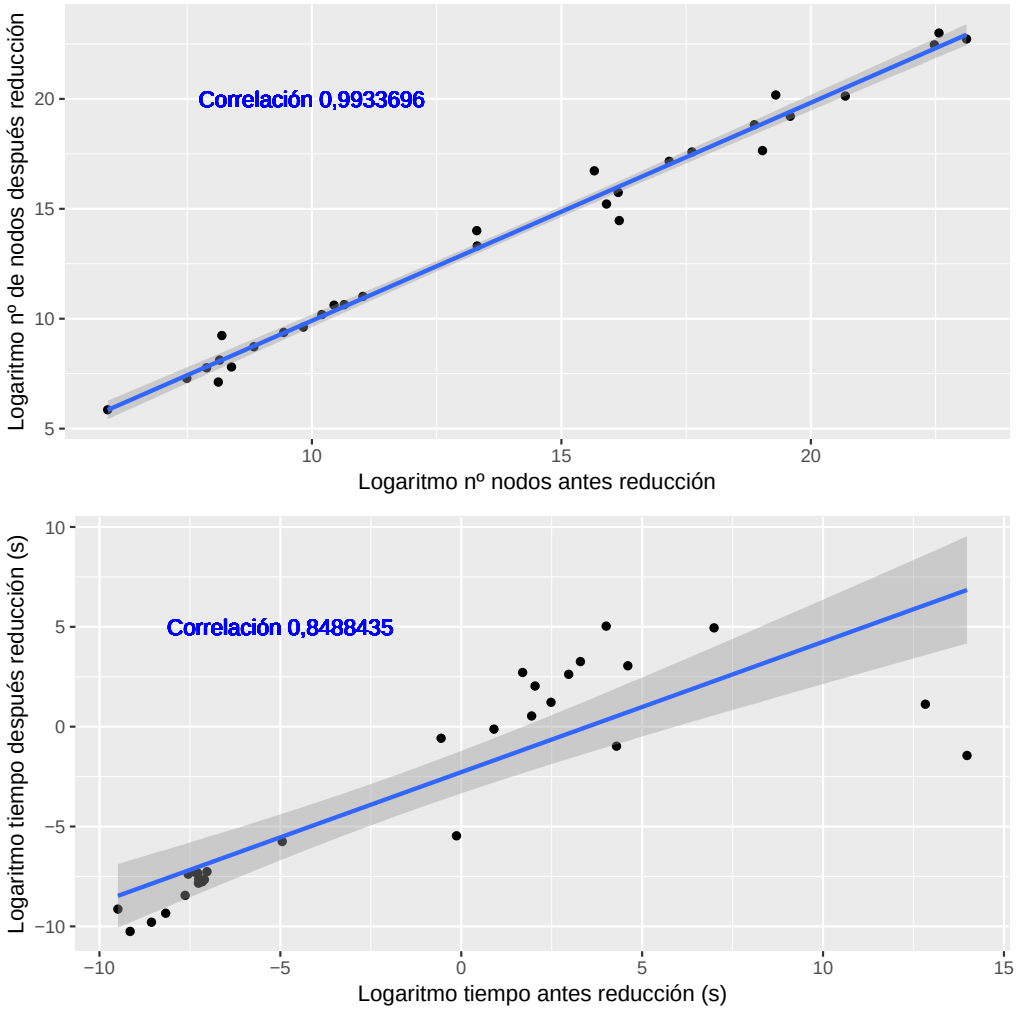


Figura 4.14: Logaritmo en base 2 del nº de nodos y el tiempo antes y después

$$log_2(|B_r|) = \beta_n \cdot log_2(|B|) \tag{4.12}$$

$$log_2(T_r) = \beta_t \cdot log_2(T) \tag{4.13}$$

En ambos casos, no se ha incluido el valor de intercepción, puesto que cuando el tamaño del BDD original es 0, lógicamente el reducido también será 0, dado que se trata de un BDD vacío. A nivel, de tiempo también se ha eliminado el valor de intercepción puesto que eso implicaría un tiempo de construcción del BDD fijo, algo que no es así por qué siempre depende del tamaño del BDD.

Las ecuaciones 4.16 y 4.17 muestran los valores de β_n y β_t obtenidos aplicando el modelo de regresión lineal no paramétrico de Jaeckel [77].

$$\beta_n = 0,9968518 \tag{4.14}$$

$$\beta_t = 0,9140854 \tag{4.15}$$

$$log_2(|B_r|) = 0,9968518 \cdot log_2(|B|) \tag{4.16}$$

$$log_2(T_r) = 0,9140854 \cdot log_2(T) \tag{4.17}$$

Hay que notar, que el valor de β_n está muy cercando a 1, pero hemos de tener en cuenta que la reducción es logarítmica. Por otra parte, el valor β_t es mucho menor lo que afirma los datos experimentales que muestra que la reducción, de tiempo es mucho mayor a la del número de nodos.

El resultado de test de dispersión *drop in* [69], que es el más robusto ([89], [111], [94] y [69]) obtiene valores de p por debajo de $0,05$, como muestran las transcripciones de **R** 22 y 23,, lo que indica que es relevante estadísticamente.

Transcripción 22

```
rfit.default(formula = datos$log_nodos_despues ~ datos$log_nodos_antes - 1)
Coefficients:
Estimate Std. Error t.value   p.value
datos$datos$log_nodos_antes 0.9968518  0.0038821  256.78 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Multiple R-squared (Robust): 0.9597852
Reduction in Dispersion Test: 668.2619 p-value: 0
```

Transcripción 23

```
rfit.default(formula = datos$log_tiempo_despues ~ datos$log_tiempo_antes - 1)
Coefficients:
Estimate Std. Error t.value   p.value
datos$log_tiempo_antes 0.914085   0.034553  26.455 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Multiple R-squared (Robust): 0.8145591
Reduction in Dispersion Test: 122.9915 p-value: 0
```

4.6.7 Conclusiones

La heurística desarrollada obtiene dos resultados significativos. El primero, permite identificar sin construir el problema completo, características *core*/*dead* del proble-

ma. Por otra parte, consigue reducir el problema, eliminando las características *core* y *dead* identificadas, en subproblemas más pequeños.

La reducción del modelo del problema implica también una reducción del propio problema, tanto a nivel de tiempo, como a nivel de nodos como han demostrado los resultados experimentales.

La descomposición del problema que se ha evaluado, es la más simple posible, separando por una parte las características jerárquicas, y las *crosstree*. Una descomposición, que potencialmente mejorará aún más sería la identificación de las características *core*/*dead*, descomponiendolo en modelos conexos.

Por otra parte, se pueden realizar múltiples descomposiciones del problema, con el objetivo de reducirlo. Por ejemplo, se puede descomponer por un lado el problema en las características jerárquicas, y las *crosstree*, y, por otra parte, las características que formen un clúster. Como, por ejemplo, el que propone Narodytska and Walsh [100].

La descomposición del problema combinada con la heurística desarrollada abre una nueva vía de investigación, a realizar en próximos trabajos.

Este no es final, ni tan siquiera el principio del final. Es tal vez, el final del principio.

Winston Churchill

CAPÍTULO 5

Conclusiones

5.1 Introducción

En este capítulo, se resumen los resultados obtenidos en los capítulos anteriores, las contribuciones fundamentales de esta tesis doctoral, y las futuras líneas de investigación.

5.2 Análisis sistemático de las heurísticas existentes

En el Capítulo 3, se ha concluido que no existe una heurística de ordenación estática capaz de obtener los mejores resultados para cualquier modelo de configuración. Si bien es cierto que las heurísticas de ordenación estáticas **FORCE** [4], **mendonca** [94] y **MINCE** [3] son las que mejores resultados obtienen en general.

Por otra parte, la heurística de ordenación dinámica **sifting** [114], es la que obtiene el mejor compromiso entre el número de nodos y el tiempo de construcción.

Los mejores resultados a nivel de nodos se obtienen combinando una heurística de ordenación estática y una heurística de ordenación dinámica, aplicada durante y después de la construcción del BDD completo.

5.3 Heurísticas de ordenación desarrolladas

5.3.1 Reducción del tamaño del BDD

En el Capítulo 4, se ha propuesto el uso del *span* que, dada una ordenación de variables, se define como la suma de la distancia entre las variables de una cláusula, como estimador del tamaño del BDD. Utilizando esta información, se puede seleccionar una heurística antes de la construcción del BDD.

Se ha desarrollado una nueva heurística, que utiliza la agrupación de PSG's, combinada con la heurística de ordenación dinámica de *sifting* agrupado [105]. Experimentalmente hemos demostrado que dicha heurística efectívamente reduce el tamaño de los BDD's.

5.3.2 Cálculo de características *core* y *dead*

Se han desarrollado algoritmos que calculan las características *core*/*dead* de modo eficiente utilizando BDD's. Estos algoritmos obtienen mejores resultados con BDD's de menor tamaño, gracias a las heurísticas de ordenación propuestas.

5.3.3 Descomposición del problema

El algoritmo de cálculo de características *core*/*dead* se ha dividido para ser aplicado con modelos de configuración descompuestos en varios BDD's. Si bien, es cierto que esta descomposición no mejora la eficiencia del algoritmo, ni a nivel de tiempo, ni de nodos, sí permite descomponer el problema en diferentes subsistemas. Dado que, a veces, como en el caso de los sistemas operativos **eCos** o **Linux**, no puede construirse el BDD completo del sistema, la descomposición propuesta facilita la detección parcial de características *core*/*dead*.

Se ha desarrollado también una heurística de reducción del problema basada en el cálculo de características *core*/*dead*, que descompone el problema utilizando varios BDD's, e identificándolas en cada uno de ellos. Luego, se eliminan estas características, reduciendo así el problema.

5.4 Contribuciones

En esta tesis se ha realizado el primer análisis sistemático de heurísticas de ordenación de BDD's, tanto estáticas como dinámicas, aplicadas a modelos de configuración.

Hemos demostrado que no existe una *bala de plata*, es decir, una heurística que permita codificar eficientemente los modelos de configuración utilizando BDD's. Puesto que cada modelo de configuración tiene su propia bala de plata. Para resolver este problema, se ha utilizado el *span* como herramienta de selección. Permitiendo así descartar las malas heurísticas.

Nuestra heurística de agrupación de variables utilizando PSG's, para modelos de configuración mejora la heurística de ordenación dinámica de *sifting*. Esta heurística además puede ser combinada con la selección de heurística estática, de nuestro resultado anterior.

Los algoritmos de cálculo de características *core/dead* utilizando BDD's han mejorado los tiempos de los algoritmos existentes. Además, el uso de las heurísticas de codificación eficiente de modelos de configuración permite reducir todavía más estos tiempos.

Para aquellos modelos que no puedan ser codificados, se han desarrollado variaciones de los algoritmos de cálculo de características *core/dead* que permiten su descomposición, y su ejecución de forma concurrente.

Hemos desarrollado un nuevo algoritmo de reducción del problema. Este algoritmo que puede ser aplicado sin construir el modelo completo, utiliza el anterior algoritmo y elimina las características *core/dead* de cada uno de los subproblemas. Y también puede ser calculado concurrentemente.

5.5 Futuras líneas de investigación

La descomposición del problema permite su reducción y la aplicación de heurísticas focalizadas en cada uno de los subproblemas. Las descomposiciones realizadas se han limitado únicamente a separar las características jerárquicas respecto a los *crosstree*.

Las técnicas de descomposición pueden realizarse o bien en anchura, utilizando técnicas como la descomposición de Shannon, o en profundidad, selecciona-

do componentes conexas utilizando técnicas como las de *clustering*. Como, por ejemplo, la heurística **narodyska** [100]. Además, pueden aplicarse o tanto a nivel fórmula Booleana, como a nivel de grafo, o combinando ambas técnicas.

La descomposición de problemas de lógica Boolena y de grafos son áreas de investigación, con mucha producción científica. Por ejemplo, una búsqueda de los términos *Boolean*, *Decomposition* y *Function*, arroja un resultado de más de 900 entradas, y *Graph Decomposition*, más de 18.000,00, en Web of Science[112] en marzo de 2019, y requiere de un análisis exhaustivo.

Las próximas líneas de investigación, se centrarán en la búsqueda de técnicas de descomposición del problema, aplicadas a modelos de configuración y utilizando BDD's. Siguiendo el método científico:

Formulación de hipótesis, que en nuestro caso es: El uso de técnicas de descomposición utilizando de fórmulas Booleanas, como de Grafos, mejoraría la codificación de BDD's para modelos de configuración. Para ello, realizaremos un análisis exhaustivo de las técnicas existentes, realizando una revisión por expertos.

Recogida de observaciones, que en nuestro caso será la implementación de las técnicas de descomposición para obtener resultados experimentales.

Contraste de hipótesis, que en nuestro caso será la realización de análisis estadísticos de los datos experimentales. Para así, identificar si existen las técnicas adecuadas para modelos de configuración.

Readaptación de las hipótesis, que en nuestra caso será el desarrollo de nuevas técnicas de descomposición utilizando los resultados previos obtenidos.

Bibliografía

[1] Alferez, M., Lopez-Herrejon, R. E., Moreira, A., Amaral, V., and Egyed, A. (2011). Supporting consistency checking between features and software product line use scenarios. In *12th International Conference on Software Reuse (ICSR), Pohang, Korea*, pages 20–35.

[2] Aloul, F., Markov, I., and Sakallah, K. (2001). Faster SAT and smaller BDDs via common function structure. *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 443–448.

[3] Aloul, F. A. (2004). MINCE : A Static Global Variable-Ordering Heuristic for SAT Search and BDD Manipulation. *Computer*, 10(12):1562–1596.

[4] Aloul, F. A., Markov, I. L., and Sakallah, K. A. (2003). FORCE: A Fast and Easy-To-Implement Variable-Ordering Heuristic. *Proceedings of the 13th ACM Great Lakes Symposium on VLSI*, pages 116–119.

[5] Aloul, F. A., Ramani, A., Markov, I. L., and Sakallah, K. A. (2002). Generic ILP versus specialized 0-1 ILP: An update. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 450–457. ACM.

[6] Andrews, J. D. and Bartlett, L. (1998). Efficient basic event orderings for binary decision diagrams. *Reliability and Maintainability Symposium, 1998. Proceedings., Annual*, pages 61–68.

[7] Aziz, A., Tasiran, S., and Brayton, R. (1994). BDD Variable Ordering for Interacting Finite State Machines. *31st Design Automation Conference*.

[8] Baguley, T. (2012). *Serious Stats: A guide to advanced statistics for the behavioral sciences*. Palgrave Macmillan.

[9] Bak, Kacper (2013). *Modeling and Analysis of Software Product Line Variability in Clafer*. PhD thesis, University of Waterloo.

[10] Bartlett, L. and Andrews, J. (2002). Choosing a heuristic for the "fault tree to binary decision diagramçonversion, using neural networks. *IEEE Transactions on Reliability*, 51(3):344–349.

[11] Batory, D., Benavides, D., and Ruiz-Cortes, A. (2006). Automated Analysis of Feature Models: Challenges Ahead. *Commun. ACM*, 49(12):45–47.

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

[13] Berge, C. (1973). *Graphs and Hypergraphs*. North-Holland Mathematical Library.

[14] Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., and W\kasowski, A. (2013). A Survey of Variability Modeling in Industrial Practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 7:1—-7:8, New York, NY, USA. ACM.

[15] Berger, T., She, S., and Lotufo, R. (2010). Variability Modeling in the Real : A Perspective from the Operating Systems Domain. *Measurement*, pages 73–82.

[16] Berndl, M., Lhoták, O., Qian, F., Hendren, L., and Umanee, N. (2003). Points-to analysis using BDDs. *ACM SIGPLAN Notices*, 38(5):103.

[17] Beuche, D. B. (2003). *Composition and Construction of Embedded Software Families*. PhD thesis, Otto-von-Guericke-Universitet Magdeburg, Germany.

[18] Beyer, D. (2001). Improvements in BDD-based reachability analysis of timed automata. *FME 2001: Formal Methods for Increasing Software Productivity*, pages 318–343.

[19] Beyer, D., Lewerentz, C., and Noack, A. (2003). Rabbit : A Tool for BDD-Based Verification of Real-Time Systems. *15th International Conference on Computer Aided Verification*, pages 122–125.

[20] Boender, J. (2011a). *A formal study of Free Software distributions*. PhD thesis, Universite Paris Diderot.

[21] Boender, J. (2011b). Formal verification of a theory of packages. *ECEASST*, 48.

[22] Bollig, B., Lobbing, M., and Wegener, I. (1996). On the effect of local changes in the variable ordering of ordered decision diagrams. *Information Processing Letters*, 59(5):233–239.

[23] Bouissou, M. (1996). An Ordering Heuristic for Building Binary Decision Diagrams from Fault-Trees. *Reliability and Maintainability Symposium, 1996 Proceedings. International Symposium on Product Quality and Integrity., Annual*, pages 208–214.

[24] Brglez, F., Bryan, D., and Kozminski, K. (1989). Combinational profiles of sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems,*, pages 1929–1934 vol.3.

[25] Bryant, R. (1986). Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

[26] Buch, P., Narayan, A., Newton, A., and Sangiovanni-Vincentelli, A. (1997). Logic synthesis for large pass transistor circuits. *Computer-Aided Design, 1997. Digest of Technical Papers., 1997 IEEE/ACM International Conference on*, pages 663–670.

[27] Butler, K. M., Ross, D. E., Kapur, R., and Mercer, M. R. (1991). Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered Binary Decision Diagrams. *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pages 417–420.

[28] Cabodi, G., Camurati, P., and Quer, S. (1996). Improved reachability analysis of large finite state machines. In *Computer-Aided Design, 1996. ICCAD-96. Di-*

gest of Technical Papers., 1996 IEEE/ACM International Conference on, pages 354–360.

[29] Caldwell, A. A. E., Kahng, A. B. A., Markov, I. I. L., and Angeles, L. (2000). Can recursive bisection alone produce routable, placements? *Proceedings 37th Design Automation Conference*.

[30] Chaudy, Y., Connolly, T., and Hainey, T. (2013). Specification and Design of a Generalised Assessment Engine for GBL Applications. In *7th European Conference on Games-Based Learning, At Porto, Portugal, Volume: 1*.

[31] Cobo, M. J., López-Herrera, A. G., Herrera-Viedma, E., and Herrera, F. (2012). SciMAT: A new science mapping analysis software tool. *Journal of the American Society for Information Science and Technology*, 63(8):1609–1630.

[32] Conover, W. J. (1999). *Practical nonparametric statistics*. Wiley series in probability and statistics: Applied probability and statistics. Wiley.

[33] Conover, W. J. and Iman, R. L. (1979). On multiple-comparisons procedures. *Los Alamos Sci. Lab. Tech. Rep. LA-7677-MS*.

[34] Corder, G. W. and Foreman, D. I. (2009). *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach*. New Jersey: Wiley.

[35] Coudert, O. (1994). Two-level logic minimization: an overview. *Integration, the VLSI Journal*, 17(2):97–140.

[36] Coudert, O., Corporate, B., Clayes, L., France, B., Christophe, J., and Words, K. (1993). 10 20 Fault Tree Analysis: Prime Implicants and Beyond. *Reliability and Maintainability Symposium, 1993. Proceedings., Annual*.

[37] Crama, Y. and L. Hammer, P. (2011). *Boolean Functions: Theory, Algorithms, and Applications*. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

[38] Czarnecki, K. and Helsen, S. (2003). Classification of Model Transformation Approaches. *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*.

[39] Czarnecki, K. and Wasowski, A. (2007). Feature Diagrams and Logics: There and Back Again. In *11th International Software Product Line Conference (SPLC 2007)*, pages 23–34.

[40] Darwiche, A. (2002). A compiler for deterministic, decomposable negation normal form. *Proc. of AAAI*, pages 627–634.

[41] Davis, M., Logemann, G., and Loveland, D. (1962). A Machine Program for Theorem-proving. *Commun. ACM*, 5(7):394–397.

[42] Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.*, 7:1–30.

[43] Di Cosmo, R. and Boender, J. (2010). Using Strong Conflicts to Detect Quality Issues in Component-based Complex Systems.

[44] Dongen, S. (2000). A Cluster Algorithm for Graphs. Technical report, University of Utrecht, Amsterdam, The Netherlands, The Netherlands.

[45] Drechsler, Rolf and Drechsler, N. and Günher, W. (200). Fast exact minimization of BDD's. *IEEE Transactions on Computer-Aided Design (2000)*, 19(3):384–389.

[46] Ebendt, R., Günther, W., and Drechsler, R. (2003). An improved branch and bound algorithm for exact BDD minimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(12):1657–1663.

[47] Edwards, C. R. and Hurst, S. L. (1978). A Digital Synthesis Procedure Under Function Symmetries and Mapping Methods. *IEEE Transactions on Computers*, C-27(11):985–997.

[48] Eén, N. and Sörensson, N. (2003). An Extensible SAT-solver. In Giunchiglia, E. and Tacchella, A., editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer.

[49] Eén, N. and Sörensson, N. (2006). Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability Boolean Modeling and Computation*, 2(3-4):1–26.

[50] Elsevier (2016). Scopus. `http://www.scopus.com/`.

[51] Falkowski, B. J. (2003). Generalized Reed-Muller forms as a tool to detect symmetries. *IEEE Transactions on Computers*, 52(7):975–976.

[52] Flores, R., Krueger, C., and Clements, P. (2012). Mega-scale Product Line Engineering at General Motors. In *16th International Software Product Line Conference*, SPLC '12, pages 259–268, New York, NY, USA. ACM.

[53] for Discrete Mathematics, C. and Science, T. C. (2001). DIMACS Challenge benchmarks.

[54] Friedman, M. (1937). The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association*, 32(200):675–701.

[55] Friedman, S. J. and Supowit, K. J. (1987). Finding the optimal variable ordering for binary decision diagrams. In *IEEE Transactions on Computers*, number 5 in DAC '87, pages 348–356, New York, NY, USA. ACM.

[56] Fujii, H., Ootomo, G., and Hori, C. (1993). Interleaving Based Variable Ordering Methods for Ordered Binary Decision Diagrams. *International Conference on Computer Aided Design*, pages 38–41.

[57] Fujita, M., Fujisawa, H., and Kawato, N. (1988). Evaluation and improvement of Boolean comparison method based on binary decision diagrams.

[58] Fujita, M., Matsunaga, Y., and Kakuda, T. (1991). On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. *Design Automation. EDAC., Proceedings of the European Conference on*, pages 50–54.

[59] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. (2004). Open {MPI}: Goals, Concept, and Design of a Next Generation {MPI} Implementation. In *Proceedings, 11th European PVM/MPI Users'Group Meeting*, pages 97–104, Budapest, Hungary.

[60] Garey, M., Johnson, D., and Stockmeyer, L. (1976). Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267.

[61] Garey, M., Johnson, D., Stockmeyer, L., and Batory, D. (2005). Feature Models, Grammars, and Propositional Formulas. In *Proceedings of the 9th International Conference on Software Product Lines*, volume 3714 LNCS of *SPLC'05*, pages 7–20, Berlin, Heidelberg. Springer-Verlag.

[62] Gebizli, C. S. and Sözer, H. (2016). Model-Based Software Product Line Testing by Coupling Feature Models with Hierarchical Markov Chain Usage Models. In *2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 278–283.

[63] Gil, Y., Kremer-Davidson, S., and Maman, I. (2010). *Sans Constraints? Feature Diagrams vs. Feature Models*, pages 271–285. Springer Berlin Heidelberg, Berlin, Heidelberg.

[64] Greenhouse, S. W. and Geisser, S. (1959). On methods in the analysis of profile data. *Psychometrika*, 24(2):95–112.

[65] Grumberg, O., Livne, S., and Markovitch, S. (2003). Learning to Order BDD Variables in Verification. *J. Artif. Intell. Res. (JAIR)*, 18:83–116.

[66] Heradio, R., Fernández-Amorós, D., Cerrada, J. A., and Abad, I. (2013). A literature Review on Feature Diagram Product Counting and its Usage in Software Product Line Economic Models. *International Journal of Software Engineering and Knowledge Engineering*, 23(8):1177.

[67] Heradio, R., Perez-Morago, H., Fernández-Amorós, D., Bean, R., Cabrerizo, F. J., Cerrada, C., and Herrera-Viedma, E. (2016). Binary Decision Diagram Algorithms to Perform Hard Analysis Operations on Variability Models.

[68] Heradio-Gil, R., Fernandez-Amoros, D., Cerrada, J. A., and Cerrada, C. (2011). Supporting commonality-based analysis of software product lines. *IET Software*, 5(6):496–509.

[69] Hettmansperger, T. and McKean, J. W. (2011). *Robust Nonparametric Statistical Methods*. CRC Press, second edition.

[70] Hong, Y. and Beerel, P. A. (1997). Safe BDD Minimization Using Don't Cares. In *In Design Automation Conf*, pages 208–213.

[71] Hung, W. N. N., Song, X., Aboulhamid, E. M., and Driscoll, M. A. (2002). BDD minimization by scatter search. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 21(8):974–979.

[72] Huth, M. and Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning About Systems*. Cambridge University Press, New York, NY, USA.

[73] Huynh, H. and Feldt, L. S. (1976). Estimation of the Box Correction for Degrees of Freedom from Sample Data in Randomized Block and Split-Plot Designs. *Journal of Educational and Behavioral Statistics*, 1(1):69–82.

[74] Iman, R. L., Hora, S. C., and Conover, W. J. (1984). Comparison of Asymptotically Distribution-Free Procedures for the Analysis of Complete Blocks. *Journal of the American Statistical Association*, 79(387):674–685.

[75] Ishiura, N., Sawada, H., and Yajima, S. (1991). Minimization of binary decision diagrams based on exchanges of variables. *1991 IEEE International Conference on ComputerAided Design Digest of Technical Papers*, pages 472–475.

[76] Isles, A. J., Brayton, R. K., and Sangiovanni-vincentelli, A. L. (1997). Reachability Analysis Using Partitioned-ROBDDs Amit Narayan 3 Reachability Analysis : Monolithic ROBDDs. *Electrical Engineering*, pages 388–393.

[77] Jaeckel, L. A. (1972). Estimating Regression Coefficients by Minimizing the Dispersion of the Residuals. *Ann. Math. Statist.*, 43(5):1449–1458.

[78] Jung, W. S., Han, S. H., and Ha, J. (2004). A fast BDD algorithm for large coherent fault trees analysis. *Reliability Engineering & System Safety*, 83(3):369–374.

[79] J.W., T. (1953). The problem of multiple comparasions. Artículo no publicado.

[80] Kang, K. C., Cohen, S. G., Hess, J. a., Novak, W. E., and Peterson, a. S. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. *Distribution*, 17(November):161.

[81] Kang, K. C., Lee, J., and Donohoe, P. (2002). Feature-oriented product line engineering. *IEEE Software*, 19(4):58–65.

[82] Kloke, J. and McKean, J. W. (2014). *Nonparametric Statistical Methods Using R*. Chapman & Hall/CRC The R Series. Taylor & Francis.

[83] Labs, P. (2018). Geekbench 4. `https://www.geekbench.com`.

[84] Lam, M. S., Whaley, J., Livshits, V. B., Martin, M. C., Avots, D., Carbin, M., and Unkel, C. (2005). Context-sensitive Program Analysis as Database Queries. *ACM Symposium on Principles of Database Systems*, pages 1–12.

[85] Laporte, G. (1992). The Traveling Salesman Problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:231–247.

[86] Lesta, U., Schaefer, I., and Winkelmann, T. (2015). Detecting and Explaining Conflicts in Attributed Feature Models. *Workshop on Formal Methods and Analysis in SPL Engineering, London, UK,.*

[87] Lind-Nielsen, J. (2016). {BuDDy}: A {B}inary {D}ecision {D}iagram library.

[88] Malik, S., Wang, A., Brayton, R., and Sangiovanni-Vincentelli, A. (1988). Logic verification using binary decision diagrams in a logic synthesis environment.

[89] Markatou, M. and He, X. (1994). Bounded Influence and High Breakdown Point Testing Procedures in Linear Models. *Journal of the American Statistical Association*, 89(426):543–549.

[90] Mauchly, J. W. (1940). Significance Test for Sphericity of a Normal n-Variate Distribution. *The Annals of Mathematical Statistics*, 11(2):204–209.

[91] Mendonca, M. (2009). *Efficient Reasoning Techniques for Large Scale Feature Models*. Ph.d. dissertation, University of Waterloo.

[92] Mendonca, M., Branco, M., and Cowan, D. (2016). SXFM Format.

[93] Mendonca, M. and Cowan, D. (2010). Decision-making coordination and efficient reasoning techniques for feature-based configuration. *Science of Computer Programming*, 75(5):311–332.

[94] Mendonca, M., Wasowski, A., Czarnecki, K., and Cowan, D. (2008). Efficient compilation techniques for large scale feature models. *Proceedings of the 7th international conference on Generative programming and component engineering - GPCE '08*, page 13.

[95] Michingan, U. O. (2016). Iscas high-level models. `http://web.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html`.

[96] Millo, J.-V., Ramesh, S., Krishna, S. N., and Narwane, G. K. (2013). *Compositional Verification of Software Product Lines*, pages 109–123. Springer Berlin Heidelberg, Berlin, Heidelberg.

[97] Mishchenko, A. (2003). Fast computation of symmetries in Boolean functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(11):1588–1593.

[98] Moller, D., Mohnke, J., and Weber, M. (1993). Detection of symmetry of Boolean functions represented by ROBDDs. *Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on*, pages 680–684.

[99] Nam, G. ., Aloul, F., Sakallah, K. A., and Rutenbar, R. A. (2004). A comparative study of two Boolean formulations of FPGA detailed routing constraints. *IEEE Transactions on Computers*, 53(6):688–696.

[100] Narodytska, N. and Walsh, T. (2007). Constraint and Variable Ordering Heuristics for Compiling Configuration Problems. *International Journal of Computational Intelligence and Applications*, pages 149–154.

[101] Nemenyi, P. (1963). *Distribution-free multiple comparisons*. PhD thesis, Princeton University.

[102] Nohrer, A. and Egyed, A. (2011). Optimizing User Guidance during Decision-Making. In *2011 15th International Software Product Line Conference*, pages 25–34.

[103] Nusbaumer, O., Zio, E., Nuclear, L., and Plant, P. (2010). Analytical Solutions of Large Fault Tree Models using BDD: New Techniques and Applications. *International Conference on Probabilistic Safety Assessment and Management, PSAM*.

[104] Panda, S., Somenzi, F., and Plessier, B. F. (1994). Symmetry detection and dynamic variable ordering of decision diagrams.

[105] Panda, S., Somenzi, F., Shirpa, P., Somenzi, F., Panda, S., and Somenzi, F. (1995). Who are the variables in your neighbourhood. *ComputerAided Design 1995 ICCAD95 Digest of Technical Papers 1995 IEEEACM International Conference on*, pages 74–77.

[106] Paskevicius, P., Damasevicius, R., and Štuikys, V. (2012). *Quality-Oriented Product Line Modeling Using Feature Diagrams and Preference Logic*, pages 241–254. Springer Berlin Heidelberg, Berlin, Heidelberg.

[107] Perez-Morago, H., Heradio, R., Fernández-Amorós, D., Bean, R., and Cerrada, C. (2015). Efficient Identification of Core and Dead Features in Variability Models. {*IEEE*} *Access*, 3:2333–2340.

[108] Pil, F. K. and Holweg, M. (2004). Linking Product Variety to Order-Fulfillment Strategies. *Interfaces*, 34(5):394–403.

[109] Pohl, R., Stricker, V., and Pohl, K. (2013). Measuring the structural complexity of feature models.

[110] Quan Lau, S. (2006). *Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates*. PhD thesis, Waterloo, Ontario, Canada.

[111] Rashid, M. M. (2000). Comparisons of rank-based methods and normal theory methods for unbalanced one-way repeated measures data. *Journal of the Italian Statistical Society*, 9(1):199.

[112] Reuters, T. (2016). Web of science. `http://wokinfo.com/`.

[113] (Roger) Jiao, J., Simpson, T. W., and Siddique, Z. (2007). Product family design and platform-based product development: a state-of-the-art review. *Journal of Intelligent Manufacturing*, 18(1):5–29.

[114] Rudell, R. (1993). Dynamic Variable Ordering for Ordered Binary Decision Diagrams. *Proceedings of 1993 International Conference on Co*.

[115] Ruijters, E. and Stoelinga, M. (2015). Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15:29–62.

[116] Safilian, A., Maibaum, T., and Diskin, Z. (2015). *The Semantics of Cardinality-Based Feature Models via Formal Languages*, pages 453–469. Springer International Publishing, Cham.

[117] Sanghavi, J. and Ranjan, R. (1996). CAL BDD.

[118] Schobbens, P. Y., Heymans, P., and Trigaux, J. C. (2006a). Feature Diagrams: A Survey and a Formal Semantics. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 139–148.

[119] Schobbens, P. Y., Heymans, P., and Trigaux, J. C. (2006b). Feature Diagrams: A Survey and a Formal Semantics. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 139–148.

[120] Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., and Bontemps, Y. (2007). Generic semantics of feature diagrams. *COMPUTER NETWORKS*, 51(2):456–479.

[121] Scholl, C., Moller, D., Molitor, P., and Drechsler, R. (1999). BDD minimization using symmetries. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(2):81–100.

[122] Shapiro, S. S. and Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 3(52).

[123] She, S., Lotufo, R., Berger, T., Wasowski, A., and Czarnecki, K. (2010). The Variability Model of the Linux Kernel. In *Fourth International Workshop on Variability Modelling of Software-intensive Systems (VAMOS'10)*, Linz, Austria.

[124] She, Steven (2013). *Feature Model Synthesis*. PhD thesis, University of Waterloo.

[125] Sheskin, D. J. (2007). *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 4 edition.

[126] Shiple, T., Hojati, R., Sangiovanni-Vincentelli, A., and Brayton, R. (1994). Heuristic Minimization of BDDs Using Don't Cares. *31st Design Automation Conference*.

[127] Siegel, S. and Castellan, N. J. (1988). *Nonparametric statistics for the behavioral sciences*. McGraw–Hill, Inc., second edition.

[128] Snelting, G., Snelting, G., Robschink, T., Robschink, T., Krinke, J., and Krinke, J. (2006). Efficient path conditions in dependence graphs for software safety analysis. *ACM Trans. Softw. Eng. Methodol.*, 15(4):410–457.

[129] Somenzi, F. (1999). Binary Decision Diagrams. *NATO Advanced Study Institute on Calculational System Design*, 173(3):173.

[130] Somenzi, F. (2016). Cudd : Cu decision diagram package release 3.0.0. `http://vlsi.colorado.edu/~fabio/CUDD/html/index.html`.

[131] Tartler, R. (2013). Mastering Variability Challenges in Linux and Related Highly-Configurable System Software.

[132] Thüm, T., Apel, S., Kästner, C., Kuhlemann, M., Schaefer, I., and Saake, G. (2004). Analysis strategies for software product lines.

[133] Tseitin, G. S. (1968). On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic*, Part II:115–125.

[134] Umans, C., Villa, T., and Sangiovanni-Vincentelli, A. L. (2006). Complexity of two-level logic minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(7):1230–1246.

[135] Urquhart, A. (1987). Hard examples for resolution. *Journal of the ACM (JACM)*, 34(1):209–219.

[136] van Deursen, A. and Klint, P. (2001). Domain-Specific Language Design Requires Feature Descriptions. *Journal of Computing and Information Technology*, 10:2002.

[137] Velev, M. N. and Bryant, R. E. (1999). Superscalar Processor Verification Using Efficient Reductions of the Logic of Equality with Uninterpreted Functions to Propositional Logic. In Pierre, L. and Kropf, T., editors, *Correct Hardware Design and Verification Methods*, pages 37–53, Berlin, Heidelberg. Springer Berlin Heidelberg.

[138] Vlasenko Denys (2018). BusyBox: The Swiss Army Knife of Embedded Linux.

[139] Wang, F. (2004). Efficient verification of timed automata with BDD-like data structures. *International Journal on Software Tools for Technology Transfer*, 6(1):77–97.

[140] Whaley, J. (2016). JavaBDD.

[141] Yin, Z., Ma, X., Zheng, J., Zhou, Y., Bairavasundaram, L. N., and Pasupathy, S. (2011). An Empirical Study on Configuration Errors in Commercial and Open Source Systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 159–172, New York, NY, USA. ACM.

[142] Zimmerman, D. W. and Zumbo, B. D. (1993). Relative Power of the Wilcoxon Test, the Friedman Test, and Repeated-Measures ANOVA on Ranks. *The Journal of Experimental Education*, 62(1):75–86.

Declaración

Por la presente declaro que he realizado este trabajo, sin la asistencia prohibida de terceros y sin hacer uso de ayudas distintas a las especificadas; las referencias tomadas directa o indirectamente han sido identificadas como tales. Este trabajo no se ha presentado previamente en forma idéntica o similar a ninguna junta de examen.

UNED

Madrid,

La tesis ha sido escrita en Cornellá de Llobregat en 16 de marzo de 2019

Está página está intencionadamente en blanco