UNIVERSIDAD DE GRANADA

DEPARTAMENTO DE ELECTRÓNICA Y
TECNOLOGÍA DE COMPUTADORES

UN NUEVO PROCEDIMIENTO PARA LA MINIMIZACIÓN AND-EXOR Y SU PARALELIZACIÓN EN SISTEMAS DE MEMORIA DISTRIBUIDA

TESIS DOCTORAL

Luis Parrilla Roure

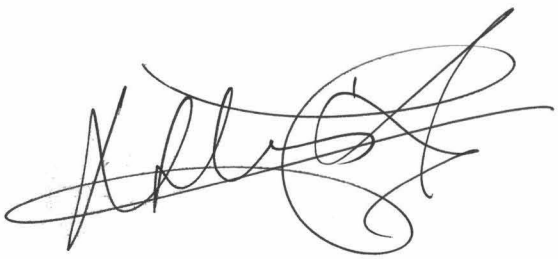
Granada, Abril 1997

D. Antonio Lloris Ruiz,
Catedrático de Universidad,
y
D. Julio Ortega Lopera,
Profesor Titular de Universidad,
ambos del Departamento de Electrónica y
Tecnología de Computadores, de la
Universidad de Granada.

CERTIFICAN:

Que la memoria titulada: "Un nuevo procedimiento para la minimización AND-EXOR y su paralelización en sistemas de memoria distribuida" ha sido realizada por D. Luis Parrilla Roure bajo nuestra dirección, en el Departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada. Esta memoria constituye la tesis que D. Luis Parrilla Roure presenta para optar al grado de Doctor en Ciencias Físicas.

Granada, a 30 de Abril de 1997.



Fdo. Antonio Lloris Ruiz
Codirector de la Tesis.

Fdo. Julio Ortega Lopera
Codirector de la Tesis.

ÍNDICE

PRÓLOGO

Se realiza una breve introducción, y se describe la estructura general del trabajo.

En los últimos años ha surgido un gran interés por la realización de circuitos lógicos utilizando síntesis AND-EXOR debido a que, en general, este tipo de circuitos requieren menor número de puertas que los AND-OR para su realización, y a una serie de trabajos que indican que el test de estructuras AND-EXOR resulta más simple. Éste hecho es especialmente relevante en el diseño de circuitos integrados VLSI por la gran importancia y complejidad que presenta el proceso del test en tales circuitos, y en las estructuras de tipo PLA, donde se consigue un gran ahorro de puertas frente a la síntesis AND-OR.

Las interesantes propiedades de la lógica AND-EXOR son conocidas desde hace cierto tiempo [EVE67], pero no han podido ser aprovechadas hasta la actualidad por dos inconvenientes que presentaba este tipo de síntesis:

a) La puerta EXOR requería mayor área que la OR para su integración.

b) No existía un procedimiento de minimización AND-EXOR que proporcionara resultados satisfactorios en un tiempo razonable.

La cuestión a) se ha resuelto con los avances en la tecnología de fabricación de circuitos integrados, y queda totalmente eliminada si se utilizan dispositivos programables del tipo PLA o FPGA. El inconveniente b) no se encuentra resuelto, aunque la disponibilidad de ordenadores con mayor capacidad de procesamiento y memoria está permitiendo grandes avances en este terreno. Prueba de ello son los procedimientos de minimización EXMIN2 [SAS93] y EXORCISM-MV-2 [SON96], que proporcionan resultados bastante buenos, aunque todavía algo alejados de la síntesis mínima AND-EXOR, según se tendrá oportunidad de comprobar en el Capítulo 6 del presente trabajo.

En esta memoria se desarrolla una solución a la cuestión b) basada en un nuevo planteamiento de la minimización AND-EXOR como problema de optimización combinatoria. El procedimiento de minimización desarrollado, a diferencia de los demás que aparecen en la literatura, utiliza un algoritmo no determinista como es el Enfriamiento Simulado para controlar la aplicación de un conjunto de reglas de reescritura sobre las funciones a minimizar. La utilización de un método no determinista permite decidir qué calidad de la solución se va a obtener según el tiempo de ejecución que se esté dispuesto a esperar. Un procedimiento determinista no admite esta posibilidad, y no puede mejorarse la solución mediante un aumento del tiempo de ejecución.

Por otra parte, se ha tratado de aprovechar la disponibilidad actual de sistemas multiprocesador o, en su defecto, de una serie de ordenadores conectados en red, de la que se dispone en cualquier Departamento dedicado al diseño de sistemas digitales y/o circuitos integrados con el objetivo de reducir los tiempos de ejecución. Para ello se ha procedido a paralelizar el procedimiento desarrollado usando el software de dominio público PVM (Parallel Virtual Machine), muy extendido y utilizado por una gran cantidad de plataformas multiprocesador, y que además permite de forma sencilla utilizar una red de estaciones de trabajo como un sistema multiprocesador de memoria distribuida.

El trabajo se encuentra estructurado en siete capítulos y tres apéndices, cuya breve descripción se presenta a continuación:

- **CAPÍTULO 1: INTRODUCCIÓN.** En este primer capítulo se ha realizado un repaso de la historia de los circuitos y sistemas digitales, llegando hasta la situación actual. Se ha efectuado una clasificación de los circuitos integrados disponibles y de las posibilidades de diseño que ofrecen, haciendo especial hincapié en las FPGAs, por su creciente importancia en la realización de prototipos. Asimismo, se ha introducido la lógica AND-EXOR como una alternativa para el diseño de sistemas digitales en general, y en particular, el basado en este tipo de dispositivos programables. Se han analizado las ventajas e inconvenientes que presenta este tipo de lógica, mencionando las posibles soluciones para estos últimos, y comentando en qué manera contribuye el trabajo aquí presentado a esas soluciones.

- **CAPÍTULO 2: LA LÓGICA BASADA EN LAS PRIMITIVAS AND Y EXOR.** Se introducen los conceptos básicos para seguir el resto del trabajo. Se define el Álgebra de Reed-Muller, sobre la que se realizan los diseños AND-EXOR, y se analizan las distintas posibilidades de representación de una función de conmutación dentro de esta lógica.

- **CAPÍTULO 3: PROCEDIMIENTOS DE MINIMIZACIÓN AND-EXOR.** Se realiza una revisión de los procedimientos de minimización AND-EXOR más representativos, tanto en el caso de procedimientos exactos como de métodos aproximados.

- **CAPÍTULO 4: EL PROCEDIMIENTO DE MINIMIZACIÓN RRMIN2.** Una vez conocida la lógica AND-EXOR y los procedimientos existentes en la literatura para afrontar su minimización, en este capítulo se desarrolla un nuevo procedimiento basado en la aplicación de reglas de reescritura controladas mediante un proceso de Enfriamiento Simulado El nuevo procedimiento, denominado RRMIN2, permite la minimización de funciones completa o incompletamente especificadas con varias salidas.

- **CAPÍTULO 5: PARALELIZACIÓN DE RRMIN2 EN SISTEMAS DE MEMORIA DISTRIBUIDA: PRRMIN.** En este capítulo se aborda la paralelización del algoritmo RRMIN2 para su ejecución en ordenadores paralelos y/o redes de estaciones de trabajo. Se obtiene así un nuevo procedimiento, denominado PRRMIN, que utiliza el software de dominio público PVM para aprovechar plataformas de tipo multicomputador y redes de ordenadores mediante paso de mensajes.

- **CAPÍTULO 6: RESULTADOS EXPERIMENTALES.** Se presentan los resultados que se obtienen con RRMIN2 para la minimización de funciones completamente especificadas, comparándolos con los ofrecidos por los demás procedimientos. Asimismo se analiza el aprovechamiento que RRMIN2 realiza de las indiferencias en el caso de tener funciones incompletamente especificadas y del ahorro en número de puertas cuando se realizan síntesis multifuncionales. Finalmente, se estudia la ganancia de velocidad y mejora de soluciones que se obtienen con el procedimiento PRRMIN al ejecutarlo en una red de estaciones de trabajo.

- **CAPÍTULO 7: CONCLUSIONES Y PRINCIPALES APORTACIONES.** En este último capítulo se recogen las conclusiones y se resumen las principales aportaciones de la presente memoria, así como las líneas a seguir en investigaciones futuras.

Al final de la memoria se incluyen tres apéndices con los siguientes contenidos:

- **APÉNDICE A: Demostración de la Proposición 4.19.** Se incluye la demostración de la Proposición 4.19, que no resulta importante para el seguimiento del Capítulo 4, pero que presenta cierto interés.

- **APÉNDICE B: Esquema de Enfriamiento utilizado en RRMIN2.** Se obtienen algunas expresiones útiles para la descripción del esquema de enfriamiento de RRMIN2.

- **APÉNDICE C: Integración de RRMIN2 en herramientas de diseño.** Se describe el interfaz que utilizan los programas implementados para la realización de las pruebas experimentales, y los programas de conversión desarrollados para faciliar la integración de RRMIN2 en entornos de diseño.

Para facilitar la lectura de la presente memoria conviene tener en cuenta las siguientes convenciones:

- Cada Capítulo se ha dividido en apartados. Cuando se hace alusión a cualquier parte del texto, se indica el número del Capítulo seguido del número del apartado. Ejemplo: Apartado 1.1.3.
- Las figuras y las tablas utilizadas están numeradas por capítulos. Ejemplos: Figura 1.20, Tabla 6.10.
- Las expresiones matemáticas que son referenciadas tienen igualmente una numeración por capítulos. Las referencias a éstas se indican mediante sus números entre paréntesis. Ejemplo: (6.4).
- Las referencias bibliográficas se indican en el texto mediante las tres primeras letras del apellido del autor seguidas por el año de publicación del trabajo. Si hubiera varios trabajos con la misma denominación se añade una letra. Ejemplos: [SAS95], [SAS90a, 93b].

CAPÍTULO 1

INTRODUCCIÓN

En este primer Capítulo se efectúa una breve revisión del proceso de diseño de sistemas digitales, describiéndose la metodología que se utiliza en la actualidad. En este contexto, se introduce la lógica AND-EXOR como una alternativa al diseño AND-OR clásico, analizando las ventajas e inconvenientes que presentan cada una de ellas.

1.1. EL DISEÑO DE SISTEMAS DIGITALES

La metodología de diseño de los sistemas digitales ha venido siempre condicionada por la tecnología disponible, y en especial por el nivel de desarrollo de los circuitos integrados. Al aumentar la complejidad de los circuitos a tratar, han aparecido otros factores muy importantes que afectan al proceso de diseño, como es la testeabilidad de los mismos. En este contexto, el diseño lógico mediante las primitivas AND y EXOR permite obtener diseños más fácilmente testeables y con menor número de puertas que los diseños tradicionales AND-OR (Apartado 1.2).

En un primer apartado (Apartado 1.1.1) se va a comentar brevemente la evolución histórica de los sistemas digitales, describiéndose posteriormente (Apartado 1.1.2) el estado

actual del diseño de este tipo de sistemas. Por su importancia en el desarrollo de sistemas digitales actualmente, el Apartado 1.1.3 se dedica al diseño mediante FPGAs.

1.1.1. Evolución histórica de los sistemas digitales

Los primeros circuitos digitales se desarrollaron en los años 30 con relés, que fueron sustituidos posteriormente por tubos de vacío [NEL95]. En esta primera fase, el principal problema para el desarrollo de sistemas digitales no era el diseño en sí, sino la tecnología empleada. Tanto los relés como los tubos de vacío tenían tiempos medios de vida relativamente bajos, lo que unido a la gran cantidad de elementos que constituían el sistema, hacían que éste no fuera capaz de funcionar correctamente más allá de unas pocas horas. A esto había que añadir el enorme consumo de energía eléctrica que se generaba.

Fue a finales de los años 50, con la comercialización de los diodos y transistores bipolares, cuando se consiguió disponer de elementos con bajo consumo, y una gran rapidez y fiabilidad de funcionamiento. Los circuitos se realizaban mediante elementos discretos, y se utilizaban puertas AND, OR y NOR.

A mediados de los 60 aparecieron los primeros circuitos integrados (CI), de tecnología SSI (Small Scale Integration, baja escala de integración), que permitían incorporar en una sóla pastilla alrededor de unas 4 puertas, que podían ser del tipo NAND, NOR, OR, AND, EXOR o NOT. Se disponía así de una gran variedad de puertas y opciones para la realización de circuitos combinacionales, y se facilitaba y mejoraba notablemente el proceso de diseño.

A principios de los años 70 se desarrollaron los circuitos MSI (Medium Scale Integration, media escala de integración), que admitían hasta 200 puertas por pastilla. Así, era posible incluir en un sólo componente registros, decodificadores, multiplexores, etc.

Antes de finalizar la década de los 70 aparecieron los circuitos LSI (Large Scale Integration, alta escala de integración) [NEL95], que incluían varios cientos de puertas. Era posible integrar en una pastilla circuitos tales como ALUs con registros de desplazamiento, controladores de interrupciones, secuenciadores microprogramados, ROMs, PROMs, etc. A partir de ese momento, el diseño de circuitos digitales se realizará ya utilizando exclusivamente circuitos integrados.

En los años 80 se dio un nuevo salto cualitativo al aparecer los circuitos VLSI (Very Large Scale Integration, muy alta escala de integración), con los que era posible integrar del orden de miles de puertas, e incluir en una sola pastilla una CPU. Esto supondría la aparición de los microprocesadores y los ordenadores personales. En esta misma década se iría aumentando el número de puertas integrables por pastilla, llegando a alcanzarse la cifra de 30000 puertas. Esto permitiría ya el planteamiento de los circuitos ASIC (Application Specific Integrated Circuit, circuito integrado de propósito específico), en los que prácticamente todo el circuito digital que se desea diseñar está integrado en una sóla pastilla. En esta fase se hace ya necesaria la ayuda de ordenadores y software especializado (programas CAD, Computer Aided Design, diseño asistido por ordenador) para poder afrontar el diseño de circuitos tan complejos [DES93]. Se pasa pues a una nueva filosofía en el diseño de los circuitos digitales,

que se mantiene en la actualidad.

En los años 90 se ha aumentado el número de puertas por pastilla (más de 100000) y la velocidad a la que pueden operar los circuitos digitales (cientos de Mhz) [RAB96].

En el apartado siguiente se estudian en detalle los procedimientos que se utilizan para el diseño de circuitos digitales en la actualidad, teniendo en cuenta que se dispone de circuitos integrados con una gran densidad de integración, que prácticamente permiten la realización del circuito con una sola pastilla (o unas pocas si el sistema es muy complejo).

1.1.2. Diseño de Circuitos Digitales en la actualidad

Actualmente se dispone de un abanico muy amplio de circuitos integrados para afrontar el diseño de sistemas digitales. Pueden utilizarse desde circuitos con tecnología MSI con unas pocas puertas lógicas hasta dispositivos lógicos programables VLSI con decenas de miles de puertas. La elección del tipo de dispositivo a utilizar depende fundamentalmente de la complejidad del sistema a diseñar y del número de unidades que se vayan a producir. A continuación se presenta una clasificación de los circuitos integrados digitales y de sus posibilidades de utilización en el diseño de sistemas.

Una primera clasificación distinguiría entre circuitos de propósito general, realizados totalmente por un determinado fabricante para formar parte de una gran cantidad de sistemas distintos y CI diseñados en parte o completamente para la realización de un sistema digital concreto. Así, se tienen [LLO96]:

a) **Circuitos estándar.** Se trata de circuitos integrados fabricados completamente por el constructor del circuito, y que tienen una gran versatilidad para permitir la realización de cualquier sistema. Son circuitos de bajo coste, pero su utilización está restringida a diseños no excesivamente complejos y en los que el volumen de producción no sea muy grande. Se dispone de dos clases de circuitos según la tecnología empleada:
   - a.1) **SSI/MSI**, que incluyen desde puertas lógicas individuales hasta registros o contadores.
   - a.2) **VLSI**, que pueden implementar desde memorias RAM o ROM hasta microprocesadores.

b) **Circuitos de propósito específico (ASIC).** Su utilización se justifica en el caso de grandes volúmenes de producción o en el caso de requerir grandes prestaciones o funciones que no se encuentran realizadas en circuitos estándar. Se pueden clasificar a su vez en tres grupos:
   - b.1) **Circuitos a medida (custom).** El circuito es realizado por el diseñador completamente, hasta el nivel de máscaras, para conseguir la máxima optimización del mismo. Estos circuitos resultan de interés únicamente si se necesita una gran densidad de integración y altas prestaciones. El coste de desarrollo resulta muy elevado.

b.2) **Circuitos semi-a-medida (semi-custom).** En este tipo de circuitos, el trazado de las máscaras se encuentra prefabricado, y el usuario sólo tiene que completar unas determinadas interconexiones. De esta forma se abaratan los costes de desarrollo. Las opciones más utilizadas actualmente son:

- **Gate Array.** Constan de una matriz de puertas en las que el usuario diseña solamente las máscaras de los niveles de metalización y de interconexión entre ellos.
- **Sea Gates.** Son similares a las Gate Arrays, pero las celdas son más sencillas y no existen zonas reservadas para metalizaciones, sino que toda la superficie contiene puertas. Permite niveles muy elevados de integración.
- **Structured Array.** Son Gate Arrays en las que se pueden reservar algunos cuadrantes como memorias RAM. De esta forma se puede incluir en un sólo CI una unidad de de procesamiento y la memoria para los datos a tratar.
- **Standard Cell.** En este tipo de diseño, el fabricante proporciona una biblioteca de celdas o módulos básicos (AND, NOR, etc) que el usuario ensambla para realizar su sistema. Todo este proceso se realiza por software, de manera que el diseñador envía al fabricante de CIs unos archivos con la disposición deseada.

Los tres primeros tipos de circuito integrado mencionados (Gate Array, Sea Gates y Structured Array) se suelen englobar con el término MPGA (Mask-Programmable Gate Array).

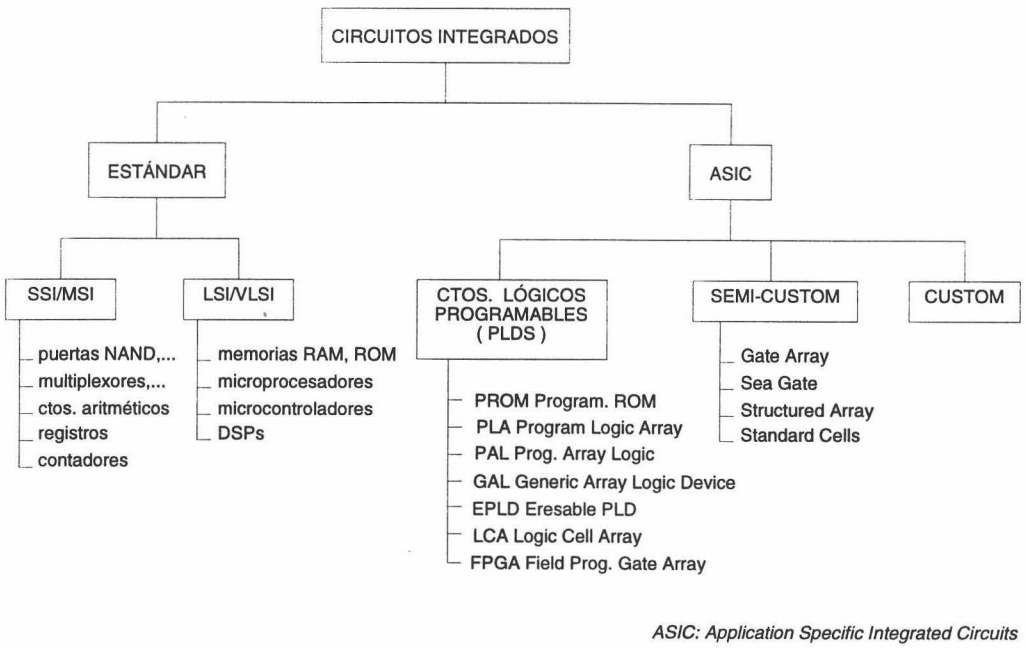
b.3) **Circuitos lógicos programables (PLD).** Se trata de circuitos integrados LSI o VLSI que contienen una estructura regular de circuitos que puede ser particularizada por el usuario para realizar una aplicación específica. Este tipo de circuitos están adquiriendo una gran importancia en el diseño de sistemas digitales debido a que se consigue una gran miniaturización (hasta 20000 puertas lógicas), gran fiabilidad de funcionamiento y menores costos y tiempos de desarrollo del sistema. Existen distintos tipos de circuitos lógicos programables dependiendo de las funciones que puedan realizarse con los mismos y la forma en que se realiza la programación. Los más importantes son los siguientes [TAV94]:

- **PROM** (Programmable ROM). Son memorias ROM programables. Se distinguen varios tipos según el procedimiento que se utilice para su programación (PROM de fusibles, EPROM, EEPROM, etc.).
- **PLA** (Programmable Logic Array). Se trata de matrices programables que constan de dos niveles de puertas, uno para la realización de productos (plano AND) y otro para efectuar las sumas (plano OR o EXOR). Se pueden programar los dos planos.
- **PAL** (Programmable Array Logic). Son similares a las PAL, pero en

este caso sólo se puede programar el plano AND. Suelen disponer de unos biestables en las salidas.

- **GAL** (Generic Array Logic). Se trata de PALs con biestables realizadas con tecnología CMOS programables eléctricamente.
- **EPLD** (Erasable Programmable Logic Device). Son PAL CMOS programables eléctricamente y borrables con radiación ultravioleta que contienen una serie de bloques lógicos configurables (CLB, Configurable Logic Block) que permiten realizar máquinas de estados, funciones combinacionales, etc., y una matriz de interconexion universal (UIM, Universal Interconection Matrix) para interconectar los bloques.
- **FPGA** (Field Programmable Gate Array). Están constituidas por una matriz regular de celdas organizada en un gran número de conjuntos de bloques lógicos configurables cuya interconexión es programable mediante una memoria RAM que llevan incorporada. Los bloques lógicos configurables son mucho más sencillos que los utilizados en los EPLDs, lo que permite una mayor versatilidad, y además se consigue una gran densidad de integración. Debido a la gran importancia que ha adquirido este tipo de circuitos en el diseño de sistemas digitales, se dedica el Apartado 1.1.3 a su estudio en detalle.

A modo de resumen, la Figura 1.1 [LLO96] presenta gráficamente la clasificación realizada.



ASIC: Application Specific Integrated Circuits

Fig. 1.1. Clasificación de los CIs utilizados para la síntesis de sistemas digitales

Una vez escogidos los circuitos integrados adecuados para el sistema que se desee realizar, puede abordarse ya el diseño del mismo. La gran complejidad de los sistemas que se diseñan actualmente hace casi imprescindible la utilización de ordenadores y herramientas software para facilitar el diseño, denominadas genericamente herramientas CAD (Computer Aided Design). Asímismo, se han desarrollado lenguajes específicos para describir de forma sencilla y estructurada (a través de distintos niveles de descripción, desde el nivel de máscara hasta el de sistema), denominados lenguajes de descripción hardware (HDL, Hardware Design Languaje) [PAL96]. La combinación de estos elementos de ayuda con el incremento de prestaciones de los ordenadores y la aparición de dispositivos lógicos programables VLSI, como es el caso de las FPGAs, permite obtener diseños muy complejos con un coste y tiempo de desarrollo cada vez menores.

1.1.3. Diseño lógico utilizando FPGAs

Una FPGA es un dispositivo lógico programable que puede contener actualmente unas 20000 puertas en un chip. Este tamaño es suficientemente grande para implementar varios sistemas digitales en un sólo chip o para realizar un sistema complejo usando varias FPGAs.

La arquitectura de una FPGA es similar a la de una MPGA, y consiste en una matriz de bloques lógicos programables que pueden ser interconectados también de forma programable para realizar diferentes diseños. Según esto, la arquitectura de una FPGA puede dividirse en dos constituyentes: la arquitectura de los bloques lógicos, y la arquitectura de enrutamiento.

1.1.3.1. Arquitectura de los bloques lógicos

El bloque lógico de una FPGA puede ir desde un par de transistores hasta una LUT (Look-Up Table, tabla de consulta) que puede implementar cualquier función de cinco variables (XILINX 3000 series, [XIL96]). Esta gran diferencia de tamaños hace que sea necesario mantener una clasificación de los bloques lógicos según su granularidad. La granularidad puede definirse de distintas formas; como el número de funciones booleanas que el bloque pude realizar, el número de puertas NAND de dos entradas equivalentes, el número total de transistores, etc. En cualquier caso da idea del tamaño que tiene el bloque. Atendiendo a esta definición, los bloques lógicos se clasifican en bloques de grano fino y bloques de grano grueso [ROS93].

- Bloques de grano fino:

a) FPGAs de punto de cruce: En este caso el bloque está formado por una pareja de transistores. Además se dispone de un segundo tipo de bloque lógico llamado baldosa lógica RAM, que puede utilizarse para construir memorias RAM o funciones lógicas aleatorias.

b) FPGA Plessey: El bloque lógico consiste en una puerta NAND de dos entradas.

c) FPGA de Concurrent Logic's: En estas FPGAs, el bloque lógico está constituido por una puerta AND y una EXOR. Por tanto, en este tipo de FPGAs, es necesario utilizar síntesis lógica mediante puertas AND y EXOR para realizar el sistema deseado.

- Bloques de grano grueso:

a) Bloque Lógico de Actel: El bloque que se utiliza está formado por un multiplexor, realizando con él las funciones lógicas que sean necesarias.

b) Bloque Lógico Quicklogic: Es similar al bloque lógico Actel; está formado por un multiplexor de 4 a 1 que está alimentado por puertas AND cuyas entradas están complementadas y sin complementar de forma alternativa.

c) Bloque Lógico de Xilinx: La base del bloque lógico Xilinx es una SRAM (Static RAM, RAM estática) que funciona como una LUT. La tabla verdad de una función lógica de K entradas es almacenada en una SRAM de 2^Kx1 direcciones.

Por supuesto, existen muchos otros bloques lógicos además de los aquí descritos, dado que cada fabricante ha desarrollado su propio bloque para la serie de FPGAs que realiza. Los bloques lógicos, además de la parte combinacional que se ha analizado, suelen disponer de una parte secuencial, generalmente de un biestable por bloque (por ejemplo, Xilix incluye dos biestables tipo D, el de Plessey uno, etc.).

Una cuestión importante que debe determinarse es si resulta más conveniente utilizar bloques de grano fino o grueso. Los bloques de grano fino tienen la ventaja de que los bloques lógicos se utilizan completamente, sin desperdiciar componentes. En contraposición, hará falta un mayor número de conexiones programables entre bloques. Existen varios estudios tratando de determinar cuál es la granularidad óptima que debe tener una FPGA para obtener la mayor densidad y prestaciones posibles, concluyendo que esta granularidad óptima, aunque depende de la aplicación que se desee implementar, en general es relativamente pequeña [ROS93].

1.1.3.2. Arquitectura de enrutamiento

La otra parte programable en una FPGA son las interconexiones entre los bloques lógicos. No se va a entrar en detalle, pero en plan general, el enrutamiento dependerá del tipo de bloque lógico que tenga la FPGA (las necesidades de enrutamiento son distintas en el caso de una FPGA de grano fino que en una de grano grueso), y debe permitir una gran flexibilidad. Suelen consistir en unas pistas que cablean la FPGA por filas y por columnas, y de una serie de cajas de interconexión programables con las que se pueden seleccionar los

bloques que se comunican entre sí. Suelen utilizarse diversos tipos de líneas de conexión, unas que interconectan los bloques lógicos con sus vecinos, otras que comunican las cajas de interconexión entre sí, y otras que sirven para conectar bloques lógicos que se encuentran lejos para evitar diferencias en los retardos. Asimismo se distribuye una señal de reloj por todo el chip, para sincronizar todos los bloques lógicos.

1.1.3.3. El proceso de diseño lógico utilizando FPGAs

El diseño lógico utilizando FPGAs permite un entorno de diseño para la realización rápida de prototipos. Por ejemplo, el sistema de diseño de Xilinx incluye un cable de transferencia para conectar un PC o estación de trabajo a una tarjeta de prototipos. Los diseñadores pueden cargar el diseño a través del cable, y probar el prototipo en la tarjeta. Si es necesario, pueden conectarse generadores de señal y analizadores a la tarjeta para verificar el diseño a la velocidad real de operación del sistema. También es posible utilizar software interactivo para insertar puntos de prueba. Ese punto de prueba puede conectarse a segmentos de E/S y sacarse por uno de los pines sin utilizar del chip y analizar la señal.

Como puede verse, la combinación de reprogramabilidad y potentes herramientas para la realización y depuración de prototipos permiten una nueva metodología de diseño. En la Figura 1.2 puede verse un modelo elaborado y específico para el proceso de diseño de ASICs mediante matrices de puertas (Gate Arrays) en el cual el diseño es verificado por simulación en cada etapa o refinamiento. El problema que plantean los simuladores es que resultan muy lentos si se quiere una gran precisión, de manera que se suelen utilizar simuladores poco precisos en las primeras etapas del diseño e ir aumentando la precisión gradualmente.

En la Figura 1.3 se tiene el proceso de diseño utilizando FPGAs [TRI93]. El diseñador puede reemplazar la simulación por la verificación en tiempo real utilizando un prototipo. Así, las FPGAs reprogramables introducen un cambio en la metodología del diseño de sistemas digitales al promover el uso de prototipos frente a la simulación, con las ventajas que ello implica en cuanto a rapidez y precisión en la verificación de los circuitos. Además se dispone de potentes herramientas software que proporcionan unos entornos de diseño amigables que facilitan enormemente la labor del diseñador, aumentando así el rendimiento del personal humano y de los recursos disponibles.

1.2. LA LÓGICA AND-EXOR EN EL DISEÑO DE SISTEMAS DIGITALES

El diseño de sistemas digitales se ha realizado tradicionalmente mediante síntesis AND/OR (o, mediante una transformación inmediata, utilizando síntesis NAND/NAND o NOR/NOR), basándose en el Álgebra de Boole. Existe una síntesis lógica alternativa, obtenida mediante la sustitución de la operación OR por la EXOR, que da lugar a otra estructura matemática distinta, GF(2), (Apartado 2.2), y que presenta dos ventajas muy importantes:

1) En general, un circuito digital precisa menor número de puertas para su realización

mediante síntesis AND-EXOR que en el caso AND-OR [SAS95].

2) El test de determinadas estructuras AND-EXOR (Formas canónicas de Reed-Muller, Apartado 2.4.1) resulta mucho más sencillo y rápido que usando lógica AND-OR [RED72], [DAM89a], [DAM89b], [ORT91], [ORT93] (No obstante, las formas canónicas no constituyen formas mínimas AND-EXOR, por lo que no siempre se conseguirá obtener ganancia en el número de términos producto con respecto a la síntesis AND-OR si se las utiliza. Por otra parte, no está claro que si se usan formas mínimas AND-EXOR, las estructuras resultantes sean más fácilmente testeables en todos los casos).

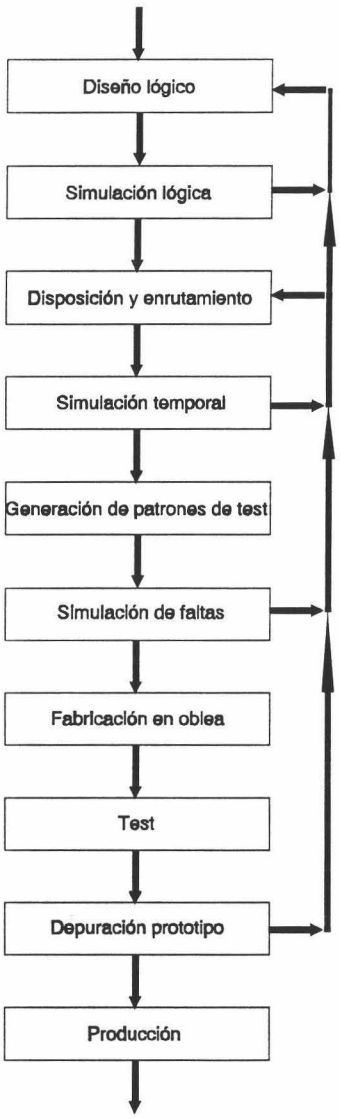


Fig. 1.2. Diseño clásico mediante matrices de puertas.

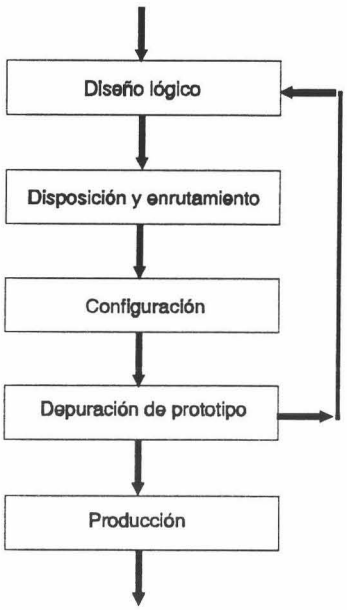


Fig. 1.3. Diseño utilizando FPGAs.

El hecho de que hasta ahora no se haya utilizado de forma generalizada este tipo de diseño, a pesar de sus innegables ventajas, se debe principalmente a dos causas:

1) La puerta EXOR precisaba más área que la OR para integrarse.
2) No existe un minimizador AND-EXOR realmente eficaz, especialmente para funciones con un gran porcentaje de unos en su tabla de verdad, y un número de variables elevado.

Estos dos problemas se están superando con la aparición de nuevos métodos de integración y diseño. Concretamente, el primero de ellos ya no aparece con las tecnologías CMOS y los dispositivos lógicos programables, y el segundo se encuentra en una fase bastante avanzada de resolución con procedimientos tales como los presentados en [SAS93a] o [SON96]. En la presente memoria se desarrolla un nuevo procedimiento para la minimización AND-EXOR (RRMIN2, Capítulo 4) que mejora notablemente los resultados aparecidos hasta el momento, y que además permite una integración sencilla con las herramientas CAD y lenguajes HDL utilizadas en el diseño actual. La principal innovación que introduce RRMIN2 es utilizar un método no determinista, como es el Enfriamiento Simulado [AAR90], combinado con la aplicación de un conjunto de reglas de reescritura convergente [BRA93], para realizar la búsqueda de soluciones. En un conjunto convergente, existen tanto reglas de simplificación como de expansión de la función, lo que permite asegurar la obtención de cualquier expresión AND-EXOR de la función a minimizar. El control de la aplicación de las reglas mediante el Enfriamiento Simulado garantiza,

teóricamente, que es posible explorar todo el espacio de búsqueda, a diferencia de lo que ocurre con los procedimientos deterministas, que se encuentran limitados a una región determinada. Se consigue así resolver, tal y como se mostrará en los resultados experimentales, el comportamiento poco satisfactorio de los otros métodos con funciones de alto porcentaje de unos.

Finalmente, señalar que existen dos tipos de dispositivos especialmente adecuados para la utilización de la lógica AND-EXOR, las PLAs, puesto que utilizan síntesis en dos niveles, en la que se ha demostrado claramente que la lógica AND-EXOR precisa de un número de puertas inferior a la AND-OR [SAS90a], y las FPGAs [HAO96], algunas de las cuales incorporan ya como primitivas en el bloque lógico únicamente puertas AND y EXOR (FPGA de Concurrent Logic's, Apartado 1.1.3.1).

1.3. CONCLUSIONES

En este primer capítulo se ha realizado una revisión de la evolución histórica de los sistemas digitales y sus métodos de diseño, hasta llegar al momento presente. Se han detallado también los elementos y herramientas de que se dispone en la actualidad para abordar el diseño de este tipo de sistemas, haciendo hincapié en las posibilidades que ofrecen los dispositivos lógicos programables, especialmente las FPGAs. Por último, se han presentado las ventajas e inconvenientes que aparecen en el diseño lógico mediante puertas AND y EXOR. Las principales ventajas consisten en que los circuitos precisan de un menor número de puertas para su realización y además el test de los mismos resulta menos complejo que en la lógica AND-OR, y el principal inconveniente viene dado por la inexistencia de un procedimiento de minimización AND-EXOR realmente eficaz. Precisamente, el objetivo de de la presente memoria es desarrollar un nuevo procedimiento de minimización que solvente este problema, permitiendo obtener expresiones AND-EXOR muy cercanas a la solución mínima en un tiempo de ejecución razonable.

CAPÍTULO 2

LA LÓGICA BASADA EN LAS PRIMITIVAS AND Y EXOR

En este Capítulo se introduce la lógica resultante de utilizar las primitivas AND y EXOR en lugar de las AND y OR, y se presentan las distintas descripciones de una función de conmutación dentro de esta lógica. Finalmente, se presenta el problema de la minimización AND-EXOR para cada uno de estos tipos de descripción.

2.1. INTRODUCCION

En el Capítulo anterior se han mencionado las ventajas que presenta el diseño lógico basado en las primitivas AND y EXOR. Estas dos operaciones lógicas pueden asimilarse, si se trabaja con variables bivaluadas, a las operaciones aritméticas producto módulo-2 y suma módulo-2, respectivamente. Esta identificación va a permitir utilizar estructuras y resultados matemáticos conocidos, que serán de gran ayuda para el tratamiento de las funciones expresadas en esta lógica. En la Tabla 2.1 puede verse la equivalencia entre las operaciones aritméticas y lógicas; y en el apartado siguiente se muestra que estas dos operaciones dan lugar a una estructura de cuerpo finito, denominado GF(2) (Cuerpo de Galois de orden 2 [GRE86]), con las propiedades que de ello se derivan.

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

a	b	a⊙b
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 2.1 Operaciones EXOR y AND

Posteriormente, en el Apartado 2.3 se establece la equivalencia entre el Álgebra de Boole y GF(2), proporcionando un procedimiento para pasar de la expresión en una lógica a la otra. En 2.4 se especifican las distintas formas que se utilizan para expresar una función de conmutación en lógica AND-EXOR, y en el Apartado 2.5 se plantea el problema de la minimización de cada uno de estos tipos de expresiones. Finalmente, en 2.6 se presentan las conclusiones del capítulo.

2.2. EL CUERPO GF(2)

En este apartado se describe el cuerpo GF(2), la estructura matemática que resulta de considerar como primitivas las puertas AND y EXOR (Apartado 2.2.1); y la relación existente entre esta estructura y el Álgebra de Boole (Apartado 2.2.2).

2.2.1. Definición y propiedades

A continuación se van a estudiar las propiedades que presentan las operaciones $\oplus$ y $\odot$ definidas sobre el conjunto $B=\{0,1\}$, que permitirán dotar a este conjunto de una estructura de cuerpo. De esta forma se dispondrá de una estructura matemática sobre la que desarrollar el diseño lógico AND-EXOR.

Sean x, y, z tres variables bivaluadas (es decir; $x,y,z\in B$) con las operaciones $\oplus$ y $\odot$, suma módulo-2 y producto módulo-2, respectivamente. Pueden demostrarse entonces, a partir de la tabla verdad de las mismas, las siguientes propiedades [GRE86]:

I. **Propiedad de cierre.** *Las operaciones $\oplus$ y $\odot$ son leyes de composición interna, es decir:*

$\forall x,y\in B$ *a)* $x\oplus y\in B$

b) $x\odot y\in B$

II. **Propiedad asociativa de las leyes de composición interna.** *Las dos leyes de composición interna cumplen la propiedad asociativa:*

$\forall x,y,z\in B,$ *a)* $x\oplus(y\oplus z)=(x\oplus y)\oplus z=x\oplus y\oplus z$

b) $x\odot(y\odot z)=(x\odot y)\odot z=x\odot y\odot z$

III. Conmutatividad de las leyes de composición interna. *Ambas leyes de composición interna son conmutativas:*

$\forall x,y \in B$ a) $x \oplus y = y \oplus x$

b) $x \odot y = y \odot x$

IV. Existencia de elementos neutros. *Existen elementos neutros para ambas leyes de composición interna, siendo el 0 el elemento neutro para la suma módulo-2, y el 1 el elemento neutro para el producto módulo-2:*

$\forall x \in B$ a) $x \oplus 0 = 0 \oplus x = x$

b) $x \odot 1 = x \odot 1 = x$

V. Existencia de elemento simétrico para la suma. $\forall x \in B$ *se tiene que* $x \oplus x = 0$, *y por tanto, se tiene siempre elemento simétrico que además es el propio x, es decir,* $x^{-1} = x$.

VI. Propiedad distributiva del producto respecto de la suma.

$\forall x,y,z \in B$, $x \odot (y \oplus z) = x \odot y \oplus x \odot z$

VII. Existencia de elemento inverso para el producto.

$\forall x \in B,\ x \neq 0\ \exists\ x^{-1} \in B\ /\ x \odot x^{-1} = x^{-1} \odot x = 1$

Atendiendo a las propiedades IIa, IIIa, IVa y V, B con la operación $\oplus$ forma un grupo conmutativo. Combinando estas propiedades con las IIb, IIIb, IVb y VI, se tiene que B con las operaciones $\oplus$ y $\odot$ constituye un anillo (véase que es un anillo no trivial porque los elementos neutros para la suma y el producto son distintos; $0 \neq 1$). Finalmente, la propiedad VII dota a B con $\oplus$ y $\odot$ de una estructura de cuerpo, que se conoce con el nombre de GF(2). Este cuerpo resulta ser un caso particular de álgebra llamada cuerpo finito ó de Galois. Los cuerpos de Galois GF(q) existen sólo cuando el número de valores q es primo o bien se cumple que es potencia entera de un número primo, es decir, $q = p^k$. Utilizando la teoría propia de estos cuerpos, los resultados que se han obtenido pueden extenderse fácilmente a la situación de una aritmética módulo-q. En [GRE90] puede verse un ejemplo de extensión de las técnicas de diseño módulo-2 al caso de la lógica multivaluada, concretamente utilizando GF(4).

Por otra parte, puesto que los números reales constituyen un cuerpo, es de esperar que GF(2) tenga propiedades muy similares a las de los números reales, y así se podrán trasladar muchas de las herramientas que se utilizan con estos últimos a GF(2) (por ejemplo la utilización de matrices y todas las operaciones y propiedades que llevan asociadas). Sin embargo, existe una diferencia fundamental entre estos cuerpos que limita un poco la identificación entre ambos; y es que GF(2) es un cuerpo finito (sólo tiene dos elementos) y en cambio los números reales forman un conjunto infinito no numerable.

Finalmente, apoyándose en la estructura proporcionada por GF(2), es posible definir funciones basadas en las primitivas AND-EXOR, y extenderlas a varias variables de forma

totalmente análoga a como se hace en el conjunto de los reales o en el caso de funciones booleanas. Así, en general se tratarán funciones definidas de la siguiente forma:

$$f: B^n \rightarrow B$$

En el siguiente apartado se describen las relaciones existentes entre las funciones booleanas y su expresión sobre GF(2).

2.2.2. Relaciones entre GF(2) y el Álgebra de Boole

Una vez definido GF(2), resultaría interesante verificar si existe alguna forma de relacionar las expresiones de una función de conmutación en el Álgebra de Boole y en lógica AND-EXOR. Es más, si para cada expresión AND-OR de la función es posible encontrar una expresión AND-EXOR distinta, cualquier sistema digital podrá expresarse con estas primitivas.

Los conectores AND y EXOR en álgebra de Boole vienen dados por:

$$\begin{aligned} x \odot y &= x \cdot y \\ x \oplus y &= x \cdot \bar{y} + \bar{x} \cdot y \end{aligned} \qquad (2.1)$$

De aquí puede demostrarse, utilizando las propiedades fundamentales del Álgebra de Boole y las leyes de De Morgan, que:

$$\begin{aligned} x \cdot y &= x \odot y \\ x + y &= x \odot y \oplus x \oplus y \\ \bar{x} &= x \oplus 1 \end{aligned} \qquad (2.2)$$

Estas relaciones pueden utilizarse para escribir cualquier función booleana en términos de operaciones módulo-2. Una vez establecidas estas relaciones, puede concluirse que el Álgebra de GF(2) cumple las siguientes tres condiciones:

a) **Es funcionalmente completa.** El álgebra es capaz de proporcionar una expresión distinta para cada una de las posibles funciones de conmutación de un número dado de variables. En efecto, al ser el Álgebra de Boole funcionalmente completa y verificarse las relaciones (2.2), el Álgebra de GF(2) tiene que ser funcionalmente completa.

b) **Es flexible.** El álgebra es lo suficientemente flexible como para permitir desarrollar y utilizar algoritmos para manipular las funciones con una cierta facilidad. En el Capítulo 3 se presentan varios algoritmos desarrollados para el cálculo y minimización de expresiones AND-EXOR de funciones de conmutación, por lo que se cumple esta segunda condición.

c) **Es realizable.** Los conectores básicos y las funciones que resultan de ellos son realizables físicamente y de manera sencilla. Según lo expuesto en el Apartado 1.2

GF(2) también verifica esta condición.

En consecuencia, la lógica AND-EXOR es perfectamente válida, al igual que la AND-OR, para diseñar sistemas digitales, presentando además una serie de ventajas que se han detallado suficientemente en 1.2.

En los apartados siguientes se van a dar procedimientos para obtener expresiones AND-EXOR de una función de conmutación, tratar estas expresiones, etc.

2.3. CÁLCULO DE LA EXPRESIÓN MÓDULO-2 DE UNA FUNCIÓN BOOLEANA. FORMA CANÓNICA DE REED MULLER (RM)

Generalmente, al diseñar un sistema digital, la primera expresión que se obtiene de las funciones de conmutación es la tabla verdad. Por tanto, se va a partir de una expresión booleana de la función, y será preciso, como primer paso del diseño AND-EXOR, obtener una expresión en esta otra lógica. Según se ha detallado en el apartado anterior, dada una función booleana, se puede calcular su expresión sobre GF(2) sin más que usar las relaciones (2.2). Sin embargo esto resulta muy laborioso en cuanto el número de variables es alto o la expresión de la función contiene muchos términos. Además, esta forma de hallar la expresión es muy poco apropiada para su implementación en ordenador. A continuación se va a mostrar otro método más eficaz para obtener una forma AND-EXOR de una función booleana.

2.3.1. Cálculo matricial

Considérese una función cualquiera de una variable. Dicha función podrá escribirse de forma general en términos booleanos como:

$$f(x_1)=m_0\overline{x_1}+m_1x_1 \tag{2.3}$$

donde m_i=0 ó 1 (son los valores correspondientes a la tabla verdad de la función).

Sobre GF(2), esta función se escribirá:

$$f(x_1)=g_o\oplus g_1x_1 \tag{2.4}$$

donde g_i=0 ó 1.

Aplicando las relaciones (2.2) puede comprobarse que los coeficientes g_0 y g_1 vienen dados por:

$$\begin{aligned} g_0&=m_0\\ g_1&=m_0\oplus m_1 \end{aligned} \tag{2.5}$$

En forma matricial, estas relaciones se escribirán:

$$\begin{bmatrix} g_0 \\ g_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} m_0 \\ m_1 \end{bmatrix} \tag{2.6}$$

y llamando T_1 a la matriz de transformación se obtiene la siguiente relación matricial:

$$G=T_1M \tag{2.7}$$

A continuación se va a repetir el mismo proceso para una función de dos variables como la siguiente:

$$f(x_1,x_2)=d_0m_0+d_1m_1+d_2m_2+d_3m_3 \tag{2.8}$$

donde m_i son los minterms de la función. En la notación que se va a utilizar en adelante, el subíndice i es el equivalente decimal del número binario construido poniendo un 0 si la variable correspondiente está complementada, y un 1 si no lo está. El bit más significativo corresponde a la variable con subíndice mayor, de manera que, por ejemplo, m_2 corresponderá al producto $x_2\bar{x}_1$. Siguiendo esta notación, puede demostrarse la siguiente relación matricial:

$$\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \end{bmatrix} \tag{2.9}$$

donde los g_i son los coeficientes que definen la función sobre GF(2), o sea:

$$F(x_1,x_2)=g_0 \oplus g_1x_1 \oplus g_1x_2 \oplus g_3x_2x_1 \tag{2.10}$$

En este caso el subíndice i se forma colocando un 1 si la variable aparece en el producto y un 0 si no aparece. El bit más significativo corresponde, al igual que en el caso anterior, a la variable con subíndice mayor.

Tal como se hizo con el caso de una variable:

$$G=T_2M \tag{2.11}$$

donde T_2 viene dada por:

$$T_2 = \begin{bmatrix} T_1 & 0 \\ T_1 & T_1 \end{bmatrix} \tag{2.12}$$

En general, si se desea hacer una transformación de n variables, puede demostrarse que:

$$T_n=\begin{bmatrix} T_{n-1} & 0 \\ T_{n-1} & T_{n-1} \end{bmatrix} \tag{2.13}$$

para $n>0$ y siendo $T_0=[1]$.

Por otra parte, una función de n variables puede escribirse sobre GF(2) como sigue:

$$F(x_1,x_2,\ldots,x_{n-1},x_n)=\bigoplus_{i=0}^{2^n-1} g_i x_n^{e_{i,n}} x_{n-1}^{e_{i,n-1}} \ldots x_1^{e_{i,1}} \tag{2.14}$$

donde la sumatoria es sobre GF(2) y los $e_{i,j}$ son tales que $x_k^0=1, x_k^1=x_k$. A esta forma de desarrollar una función en suma de productos módulo-2 se le denomina expansión o forma canónica de Reed-Muller (RM). Nótese que todas las variables están sin complementar y que se tienen 2^n productos distintos. A estos productos se les denomina piterms por analogía a la nomenclatura utilizada en Álgebra de Boole (minterms) y se les nota por π_i. El subíndice i es en este caso el equivalente decimal del número binario $<e_{i,n}e_{i,n-1}\ldots e_{i,1}>$, donde el bit menos significativo es $e_{i,1}$ (Por ejemplo; $x_3x_1=\pi_5$). Según esta definición, podrá escribirse:

$$F(x_1,x_2,\ldots x_n)=\bigoplus_{i=0}^{2^n-1} g_i\pi_i \tag{2.15}$$

Debe tenerse en cuenta que ésta no es la única forma de escribir una función booleana como suma de productos módulo-2, sino que existen muchas otras expresiones donde pueden aparecen variables complementadas, como se estudiará en el Apartado 2.4

Esta forma de calcular la expansión de Reed-Muller es totalmente equivalente al proceso de utilizar las expresiones dadas en (2.2), pero tiene la ventaja de que es fácilmente implementable en ordenador. Sin embargo, nótese que si en la implementación se utilizan los algoritmos usuales para el producto de matrices, harán falta 4^n-2^n sumas de dos términos (2^n-1 sumas por cada fila multiplicado por las 2^n filas que tiene la matriz T_n). Existe la posiblidad de mejorar el algoritmo de forma que sólo se efectúen las sumas cuando se tengan unos (las sumas que tengan un término igual a cero no es necesario realizarlas), con lo que para T_1 se efectuaría sólo 3-2=1 suma. En general, para T_n se tendrán 3^n unos distribuidos en 2^n filas y por tanto habrá que hacer un total de 3^n-2^n sumas.

2.3.2. Cálculo rápido

Si se aprovecha convenientemente la estructura repetitiva de las matrices de transformación T_n (2.13), puede construirse un algoritmo rápido cuyos grafos para 1, 2 y 3 variables se presentan en las figuras 2.1, 2.2 y 2.3.

El cálculo de la expresión AND-EXOR utilizando este tipo de grafos se realiza de la siguiente forma:

1) Se parte de la expresión de la función como suma de minterms, poniendo en la parte izquierda del mismo los valores 0 ó 1 de los coeficientes m_i según corresponda.

2) Se va avanzando hacia la derecha, efectuando la operación EXOR cuando se produzca la concurrencia entre dos líneas.

3) Al final, una vez que se han completado todos los cálculos, se obtienen los coeficientes g_i del desarrollo de la función como suma de piterms (Ecuación 2.15).

Cada grafo consta de n pasos en los que se realizan 2^{n-1} sumas, realizándose por tanto, un total de $n \cdot 2^{n-1}$ sumas. Esto supone un gran ahorro de operaciones con respecto a los algoritmos basados en el producto usual de matrices que se han comentado anteriormente. El procedimiento para generar el grafo para n variables sería el descrito en el Algoritmo 2.1.

Algoritmo 2.1. Generación de grafos para n variables

1) Se escribe el grafo correspondiente a T_1 (Figura 2.1).
2) Se duplica el grafo obtenido y se coloca debajo.
3) Se considera cada uno de los dos grafos como si fuera un nodo y se entrelazan tal y como se hizo para generar el de T_1. Se obtendrá así la Figura 2.2.
4) Si no se han completado los n pasos, se vuelve al paso 2.

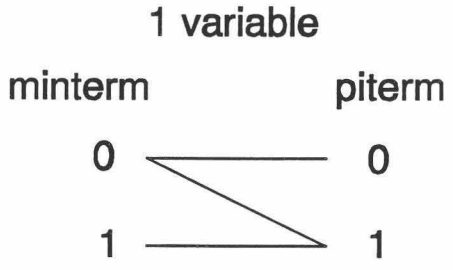


Fig. 2.1. Grafo para una variable

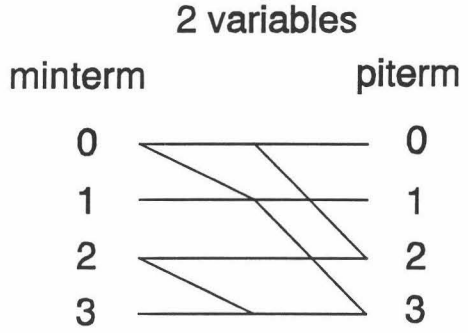


Fig. 2.2. Grafo para dos variables

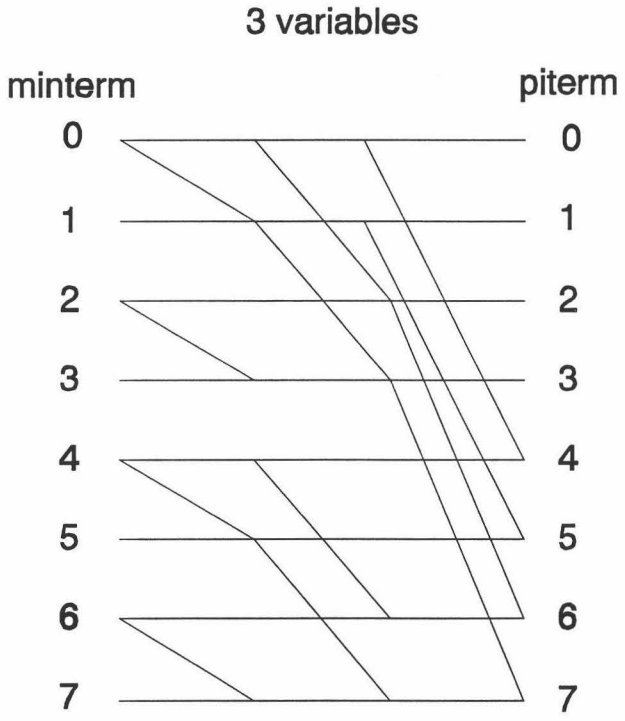


Fig. 2.3. Grafo para tres variables

En el Capítulo 4 (Apartado 4.5.2) se desarrollará una generalización de este tipo de algoritmos para efectuar de forma rápida el producto por cualquier matriz que se obtenga a partir de productos de Kronecker [GRE86]. En los siguientes apartados se estudian las distintas expresiones que suelen utilizarse en lógica AND-EXOR para una función de conmutación.

2.4. EXPRESIONES AND-EXOR DE UNA FUNCIÓN DE CONMUTACIÓN

Al igual que ocurre en el Álgebra de Boole, existen multitud de formas de expresar una función sobre GF(2) utilizando las primitivas AND y EXOR. En los apartados que siguen se van a definir una serie de formas canónicas que suelen utilizarse para expresar las funciones, debido a razones de simplicidad para su tratamiento, facilidad para el test de los circuitos resultantes, facilidad para minimizar las funciones, etc.

2.4.1. Familias de formas canónicas de Reed-Muller

En el Apartado 2.3 se ha descrito una posible forma de obtener la expresión AND-EXOR de una función a partir de su expresión como suma de minterms, la forma canónica de Reed-Muller (RM). Aplicando a esta forma una serie de transformaciones, es posible obtener una jerarquía de formas canónicas de Reed-Muller perfectamente organizada [GRE91]. Estas formas canónicas se encuentran agrupadas en familias que van disminuyendo la rigidez en su estructura a la vez que van aumentando su tamaño (entendiendo como tamaño el número de componentes de la familia). En los siguientes apartados se describen estas familias.

2.4.1.1. Forma canónica de Reed Muller (RM)

Esta forma canónica se ha estudiado en 2.3, y en ella, una función de n variables queda completamente especificada mediante un conjunto de 2^n coeficientes g_i, $0 \le i \le 2^n-1$, que se relacionan con la expresión de la función como suma de minterms mediante la expresión:

$$G=T_nM \tag{2.16}$$

donde T_n es una matriz de transformación verificando $T_n^{-1}=T_n$. Por tanto:

$$M=T_n^{-1}G=T_nG \tag{2.17}$$

La matriz T_n presenta una estructura recursiva de manera que es posible expresarla como:

$$T_n=T_1*T_{n-1} \tag{2.18}$$

donde * es el producto de Kronecker [GRE86]. Aplicando sucesivamente esta relación puede escribirse:

$$T_n=T_1*T_1*...*T_1 \qquad n \text{ veces} \tag{2.19}$$

donde, según se definió en (2.6) y (2.7):

$$T_1=\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix} \tag{2.20}$$

Los términos de esta suma de productos módulo-2 de una función de n variables pueden también identificarse empleando un producto de Kronecker de n vectores base de la forma $[1,x_j]$ $1\leq j\leq n$ verificándose lo siguiente:

$$F(x_1,x_2,...,x_{n-1},x_n)=\{[1,x_n]*[1,x_{n-1}]*...*[1,x_1]\}G \tag{2.21}$$

En los apartados siguientes se obtienen diversas familias de formas canónicas a partir de la forma RM aprovechando la notación introducida en esta expresión.

2.4.1.2. Formas canónicas de Kronecker Reed-Muller (KRM).

En este apartado se introducen las formas canónicas de Kronecker Reed-Muller, de gran importancia debido a que presentan una estructura muy bien definida que permite automatizar su cálculo y efectuarlo de forma rápida (véase el Apartado 4.5.2).

En general, una función de una variable, admite tres representaciones distintas utilizando primitivas AND-EXOR:

$$\begin{aligned} f_0(x_1)&=g_{00}\oplus g_{01}x_1\\ f_1(x_1)&=g_{10}\oplus g_{11}\overline{x_1}\\ f_2(x_1)&=g_{20}\overline{x_1}\oplus g_{21}x_1 \end{aligned} \tag{2.22}$$

Esta relaciones, aplicando (2.2), se pueden escribir en forma matricial como sigue:

$$M=\begin{bmatrix}m_0\\ m_1\end{bmatrix}=\begin{bmatrix}f_0(0)\\ f_0(1)\end{bmatrix}=\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}\begin{bmatrix}g_{00}\\ g_{01}\end{bmatrix} \tag{2.23}$$

$$M=\begin{bmatrix}m_0\\ m_1\end{bmatrix}=\begin{bmatrix}f_1(0)\\ f_1(1)\end{bmatrix}=\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}g_{10}\\ g_{11}\end{bmatrix} \tag{2.24}$$

$$M=\begin{bmatrix} m_0 \\ m_1 \end{bmatrix}=\begin{bmatrix} f_2(0) \\ f_2(1) \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} g_{20} \\ g_{21} \end{bmatrix} \tag{2.25}$$

Invirtiendo estas ecuaciones, se pueden calcular los coeficientes g_{ij} como sigue:

$$\begin{bmatrix} g_{00} \\ g_{01} \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} m_0 \\ m_1 \end{bmatrix} \tag{2.26}$$

$$\begin{bmatrix} g_{10} \\ g_{11} \end{bmatrix}=\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} m_0 \\ m_1 \end{bmatrix} \tag{2.27}$$

$$\begin{bmatrix} g_{20} \\ g_{21} \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} m_0 \\ m_1 \end{bmatrix} \tag{2.28}$$

Sustituyendo el vector M por su expresión en función de $f_0(x_1)$ se obtiene:

$$\begin{bmatrix} g_{00} \\ g_{01} \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} g_{00} \\ g_{01} \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} g_{00} \\ g_{01} \end{bmatrix}=K_0 G \tag{2.29}$$

$$\begin{bmatrix} g_{10} \\ g_{11} \end{bmatrix}=\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} g_{00} \\ g_{01} \end{bmatrix}=\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} g_{00} \\ g_{01} \end{bmatrix}=K_1 G \tag{2.30}$$

$$\begin{bmatrix} g_{20} \\ g_{21} \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} g_{00} \\ g_{01} \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} g_{00} \\ g_{01} \end{bmatrix}=K_2 G \tag{2.31}$$

Las ecuaciones anteriores permiten relacionar las expresiones $f_1(x_1)$ y $f_2(x_1)$ con $f_0(x_1)$ (nótese que $f_0(x_1)$ corresponde a la forma canónica de Reed-Muller, como se deduce de (2.21)). Así, se han obtenido dos representaciones alternativas que pueden obtenerse a partir de la forma canónica RM sin más que aplicar una matriz de transformación (K_1 para obtener $f_1(x_1)$ y K_2 para obtener $f_2(x_1)$).

De forma análoga a como se ha hecho en 2.4.1.1 para la forma RM, es posible asociar un vector base a cada una de estas formas:

$$\begin{aligned} f_0(x_1) &\rightarrow [1,x_1] \\ f_1(x_1) &\rightarrow [1,\overline{x_1}] \\ f_2(x_1) &\rightarrow [\overline{x_1},x_1] \end{aligned} \tag{2.32}$$

Todo este proceso puede generalizarse para n variables, lo que dará lugar a una serie de familias de formas canónicas, que globalmente se conocen con el nombre de formas de Kronecker Reed-Muller (KRM).

2.4.1.2.1. Formas canónicas de Reed-Muller de Polaridad Fija (FPRM)

Según lo estudiado hasta ahora, el desarrollo de RM se obtiene manteniendo todas las variables sin complementar, es decir, cuando se utiliza el vector $[1,x_j]$ para las n variables. Si para algunas de las variables x_i el vector base utilizado es de la forma $[1,\bar{x}_i]$, esas variables aparecerán complementadas. De esta forma puede seleccionarse si se desea que una variable apararezca complementada o sin complementar en el desarrollo AND-EXOR. No obstante, se tiene la limitación de que una variable no puede aparecer en el desarrollo en ambas formas. Para una función de n variables, se podrán obtener 2^n formas canónicas distintas, que se denominan de polaridad fija, y que pueden identificarse por un número de polaridad p, $0 \le p \le 2^n-1$. Este número de polaridad es el equivalente decimal del número binario $p=<p_n p_{n-1} \ldots p_1>$ formado haciendo $p_j=0$ si x_j está sin complementar y $p_j=1$ si la variable se encuentra complementada. Las formas canónicas así definidas se denominan formas canónicas de Reed-Muller de polaridad fija (FPRM), y contienen a la forma canónica RM (corresponde a la forma FPRM con $p=0$).

Finalmente, si se tienen en cuenta las expresiones (2.29) y (2.30), se concluye que cualquier forma FPRM de polaridad p puede obtenerse a partir del desarrollo de RM de polaridad cero usando una matriz de transformación $K_{(p)}$:

$$K_{(p)}=K_{p_n} * K_{p_{n-1}} * \ldots * K_{p_1} \tag{2.33}$$

operando sobre el vector G. Las componentes de $K_{(p)}$ podrán ser K_0 ó K_1.

2.4.1.2.2. Formas Canónicas de Reed-Muller de Polaridad Mixta (MPRM)

Si se incluye el tercer vector base, $[\bar{x}_j,x_j]$, se obtendrá una familia de 3^n formas canónicas, puesto que cada variable puede encontrarse en tres situaciones:

1) Que aparezca en el desarrollo siempre complementada.
2) Que aparezca siempre sin complementar.
3) Que aparezca unas veces complementada y otras sin complementar.

A estas formas se las denomina formas canónicas de Reed-Muller de polaridad mixta

(MPRM), y pueden identificarse mediante un número de polaridad m $0 \leq m \leq 3^n-1$; que es el equivalente decimal del número ternario $m=<m_n m_{n-1} \ldots m_1>$ formado escribiendo $m_j=0$ si la variable x_j se usa sin complementar, $m_j=1$ si se va a usar complementada y $m_j=2$ si se utiliza complementada y sin complementar.

Estas 3^n formas MPRM incluyen a las 2^n FPRM definidas anteriormente, que corresponderían a aquellos números ternarios que sólo contengan ceros y unos.

Al igual que en el apartado anterior, cada miembro de esta familia puede obtenerse como una transformación del desarrollo con polaridad cero empleando una matriz de transformación:

$$K_{(m)}=K_{m_n} * K_{m_{n-1}} * \ldots * K_{m_1} \tag{2.34}$$

operando sobre el vector G. Las componentes de $K_{(m)}$ podrán ser ahora K_0, K_1 ó K_2 (ecuaciones (2.29), (2.30) y (2.31)).

La denominación de formas canónicas de Kronecker Reed-Muller que se aplica a las FPRM y MPRM se debe a que su cálculo se efectua mediante matrices de transformación que se obtienen mediante el producto de Kronecker de otras matrices (ecuaciones (2.33) y (2.34)). Finalmente, nótese que las formas canónicas MPRM no incluyen todas las posibles formas de representar a una función en lógica AND-EXOR. Por ejemplo, la expresión:

$$1 \oplus x_1 \oplus \overline{x_1}\,\overline{x_2} \tag{2.35}$$

no puede obtenerse utilizando ninguna de las polaridades de las formas FPRM ni MPRM (para comprobarlo basta con escribir todos los vectores base para dos variables y ver que en ninguno de ellos aparecen simultaneamente estos tres términos producto).

2.4.1.2.3. Vectores extendido y peso

En este apartado y el siguiente se van a introducir dos vectores, el vector extendido E y el vector peso S, que van a permitir obtener el número de términos producto que contiene una forma canónica KRM sin necesidad de calcular dicha forma canónica.

Según aparece en (2.22), una función de una variable puede escribirse de tres formas sobre GF(2). Estas formas vienen determinadas por unos coeficientes g_i que pueden calcularse, según las ecuaciones (2.26), (2.27) y (2.28), de la forma que aparece en (2.36):

$$\begin{array}{ll} g_{00}=m_0 & g_{01}=m_0 \oplus m_1 \\ g_{10}=m_1 & g_{11}=m_0 \oplus m_1 \\ g_{20}=m_0 & g_{21}=m_1 \end{array} \tag{2.36}$$

En esta expresión hay sólo tres tipos de coeficientes: m_0, m_1 y $m_0 \oplus m_1$. Conociendo

estos tres valores pueden calcularse todos los g_i. Considérese un vector E_1 cuyas componentes sean las que se proporcionan en (2.36); ese vector E_1 podrá calcularse de la siguiente forma utilizando cálculo matricial:

$$E_1=\begin{bmatrix}1 & 0\\ 0 & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}m_0\\ m_1\end{bmatrix} \tag{2.37}$$

Si se define:

$$P_1=\begin{bmatrix}1 & 0\\ 0 & 1\\ 1 & 1\end{bmatrix} \tag{2.38}$$

la Ecuación (2.37) se escribirá:

$$E_1=P_1M \tag{2.39}$$

El vector E_1 contiene los coeficientes necesarios para escribir todas las MPRM de una variable.

Supóngase ahora que se desea obtener el número de productos necesario para sintetizar estas MPRM. Esto puede calcularse utilizando un vector S_1, que vendrá dado por:

$$S_1=\begin{bmatrix}1 & 0 & 1\\ 0 & 1 & 1\\ 1 & 1 & 0\end{bmatrix}\begin{bmatrix}e_0\\ e_1\\ e_2\end{bmatrix} \tag{2.40}$$

donde los e_i son las componentes del vector E_1; o bien si se define:

$$C_1=\begin{bmatrix}1 & 0 & 1\\ 0 & 1 & 1\\ 1 & 1 & 0\end{bmatrix} \tag{2.41}$$

se tendría:

$$S_1=C_1E_1 \tag{2.42}$$

donde S_1 está definido sobre los reales y la componente s_i de S_1 contiene el número de productos necesario para sintetizar la función dada por M en la polaridad i.

Utilizando el producto de Kronecker pueden generalizarse las ecuaciones anteriores para n variables [BIO72] de manera que:

$$E=P_n M \tag{2.43}$$

donde:

$$P_n=P_1*P_{n-1}=P_1*P_1*...*P_1 \quad n \text{ veces} \tag{2.44}$$

siendo P_1 la matriz (2.38).

Al vector E se le denomina vector extendido y está constituido por 3^n componentes, que contienen los coeficientes necesarios para construir todas las formas MPRM de n variables.

Asimismo, para calcular el número de productos necesario para la síntesis de cada una de las MPRM se define el vector S, denominado vector peso, como sigue:

$$S=C_n E \tag{2.45}$$

donde:

$$C_n=C_1*C_{n-1}=C_1*C_1*...*C_1 \quad n \text{ veces} \tag{2.46}$$

La componente i del vector S contiene el número de productos necesario para escribir la forma MPRM con polaridad i de la función de n variables dada por M. Así, es posible, por ejemplo, determinar la polaridad de la forma canónica MPRM que precisa menor número de productos para sintetizar una determinada función sin necesidad de proceder al cálculo de las 3^n formas MPRM.

La estructura de Kronecker de las matrices de transformación que se han ido estudiando hace que las operaciones a realizar tengan propiedades de simetría y sea posible la construcción de algoritmos rápidos, reduciendo así el tiempo de cálculo con ordenadores. Más adelante, en el Capítulo 4, se analizan en profundidad estas propiedades y se aprovechan en la elaboración de un procedimiento de minimización AND-EXOR.

2.4.1.3. Formas canónicas pseudo-Kronecker (PKRM).

A partir de la idea de formas canónicas KRM, en [DAV78] y [GRE91] se describe un procedimiento para construir una nueva familia de formas canónicas denominadas formas canónicas Pseudo-Kronecker Reed-Muller (PKRM). El procedimiento consiste en combinar las tres formas básicas para una variable (Ecuación (2.22)) de manera adecuada para ir construyendo funciones de varias variables. Efectuando todas las combinaciones posibles se obtiene la nueva familia de formas canónicas. A continuación se detalla el proceso de obtención de los miembros de esta familia.

Considérese por ejemplo una función de dos variables. Las tres formas de representación para x_2 serán:

$$f_0(x_1,x_2)=t_{00}+t_{01}x_2 \qquad f_1(x_1,x_2)=t_{10}+t_{11}\overline{x_2} \qquad f_2(x_1,x_2)=t_{20}\overline{x_2}+t_{21}x_2 \tag{2.47}$$

En cada una de estas formas, t_{k0} y t_{k1} ($0\le k\le 2$) se sustituyen por cualquiera de las tres formas de representación para x_1, existiendo por tanto un total de $3x3^2=27$ formas PKRM de 2 variables. Estas 27 formas pueden numerarse mediante un número de polaridad q formado por 3 dígitos ternarios $<q_2q_1q_0>$, que se construye de la siguiente forma:

El dígito menos significativo q_0 corresponde a la representación elegida para t_{k0} ($q_0=0 \Rightarrow t_{k0}=t_0+t_1x_1$; $q_0=1 \Rightarrow t_{k0}=t_0+t_1\overline{x_1}$; $q_0=2 \Rightarrow t_{k0}=t_0\overline{x_1}+t_1x_1$), q_1 a la representación de t_{k1} y q_2 a la representación para x_2 (es decir, $q_1=k$ donde k es un número entre 0 y 2).

En general para n variables se tiene:

$$f_0(x_1,x_2,\ldots,x_n)=t_{00}+t_{01}x_n \qquad f_1(x_1,x_2,\ldots,x_n)=t_{10}+t_{11}\overline{x_n} \qquad f_2(x_1,x_2,\ldots,x_n)=t_{20}\overline{x_n}+t_{21}x_n \tag{2.48}$$

donde t_{k0} y t_{k1} ($0\le k\le 2$) son formas PKRM de n-1 variables $(x_1,x_2,\ldots,x_{n-1})$. Por tanto habrá 3^{2^n-1} formas PKRM, que podrán representarse mediante un número de polaridad q con 2^n-1 dígitos ternarios, en el que:

$$\begin{array}{ll} <q_{2^{n-1}-2}\ldots q_0> & \textit{contiene la representación de } t_{k0} \\ <q_{2^n-3}\ldots q_{2^{n-1}-1}> & \textit{contiene la representación de } t_{kI} \\ <q_{2^n-2}> & =k \end{array} \tag{2.49}$$

Al igual que ocurría con las formas canónicas KRM, es de esperar que se puedan obtener las formas PKRM utilizando cálculo matricial. A continuación se estudia esta posibilidad.

Supóngase que se tiene una función de dos variables que utilice polaridad $q=10=<101>$. En ese caso:

$$f(x_1,x_2)=t_0\oplus t_1\overline{x_1}\oplus t_2\overline{x_2}\oplus t_3x_1\overline{x_2} \tag{2.50}$$

de donde:

$$\begin{array}{l} m_0=f(0,0)=t_0\oplus t_1\oplus t_2 \\ m_1=f(1,0)=t_0\oplus t_2\oplus t_3 \\ m_2=f(0,1)=t_0\oplus t_1 \\ m_3=f(1,1)=t_0 \end{array} \tag{2.51}$$

Estas expresiones pueden reescribirse en términos matriciales como se muestra a continuación:

$$\begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{2.52}$$

Invirtiendo:

$$\begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ m_2 \\ m_3 \end{bmatrix} \tag{2.53}$$

y finalmente, si se nombra a la matriz de transformación obtenida como P_{10} queda:

$$T=P_{10}M \tag{2.54}$$

Por analogía con las formas KRM, que se obtenían a partir de la forma canónica RM, (2.54) puede escribirse, si se tiene en cuenta (2.17), de la siguiente forma:

$$T=P_{10}T_2G=Q_{10}G \tag{2.55}$$

donde Q_{10} será:

$$Q_{10}=\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} K_1 & K_1 \\ 0 & K_0 \end{bmatrix} \tag{2.56}$$

Así, las formas PKRM se van a obtener a partir de la forma canónica de Reed-Muller utilizando matrices de transformación cuyas componentes son las matrices K_i definidas en las ecuaciones (2.29), (2.30) y (2.31).

El proceso de construcción de Q_q puede realizarse directamente de la siguiente forma [GRE91]:

$$Q_q=\begin{bmatrix} K_{q_0} & 0 \\ 0 & K_{q_1} \end{bmatrix} si\ q_2=0 \qquad Q_q=\begin{bmatrix} K_{q_0} & K_{q_0} \\ 0 & K_{q_1} \end{bmatrix} si\ q_2=1 \qquad Q_q=\begin{bmatrix} K_{q_0} & 0 \\ K_{q_1} & K_{q_1} \end{bmatrix} si\ q_2=2 \tag{2.57}$$

Este proceso resulta complicado de generalizar para el caso de más variables, por lo que se define una operación entre matrices denominada pseudo-Kronecker, simbolizada por $\circ$ y que consiste en lo siguiente:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \circ \left[\begin{bmatrix} i & j \\ k & l \end{bmatrix}, \begin{bmatrix} e & f \\ g & h \end{bmatrix} \right] = \begin{bmatrix} a\begin{bmatrix} e & f \\ g & h \end{bmatrix} & b\begin{bmatrix} e & f \\ g & h \end{bmatrix} \\ c\begin{bmatrix} i & j \\ k & l \end{bmatrix} & d\begin{bmatrix} i & j \\ k & l \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ci & cj & di & dj \\ ck & cl & dk & dl \end{bmatrix} \tag{2.58}$$

Utilizando esta definición, para q=10, el vector base vendrá dado por:

$$\left[1 \ \overline{x_2}\right] \circ \left[\left[1 \ x_1\right], \left[1 \ \overline{x_1}\right] \right] = \left[1 \ \overline{x_1} \ \overline{x_2} \ x_1\overline{x_2}\right] \tag{2.59}$$

y la matriz asociada de transformación se calculará:

$$Q_{10} = K_1 \circ \left[K_0, K_1\right] = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \circ \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \right] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.60}$$

Para ilustrar el cálculo de Q_q con un número mayor de variables se va a tratar a continuación un ejemplo con tres variables. En este caso la polaridad vendrá especificada por un número q formado por 7 dígitos ternarios q=<q_6,q_5,...,q_0>. Por ejemplo, para q=1328 =<1211012> se tendría lo siguiente:

$$Q_{1328} = K_1 \circ \left[K_2 \circ \left[K_1, K_1\right], K_0 \circ \left[K_1, K_2\right] \right] \tag{2.61}$$

de donde, haciendo operaciones, se obtiene:

$$Q_{1328} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{2.62}$$

El vector base vendrá dado por:

$$\begin{aligned}&[1\ \overline{x_3}]\circ\Big[\ [\overline{x_2}\ \ x_2]\circ\big[\ [1\ \ \overline{x_1}],[1\ \ \overline{x_1}]\ \big],[1\ \ x_2]\circ\big[\ [1\ \ \overline{x_1}],[\overline{x_1}\ \ x_1]\big]\Big]=\\&=[\overline{x_1}\ \ x_1\ \ \overline{x_2}\ \ \overline{x_2}\overline{x_1}\ \ \overline{x_3}\overline{x_2}\ \ \overline{x_3}\overline{x_2}x_1\ \ \overline{x_3}x_2\ \ \overline{x_3}x_2\overline{x_1}]\end{aligned} \tag{2.63}$$

Finalmente, se va mostrar un procedimiento para obtener de forma sistemática las formas PKRM para una función de n variables. En ese caso, el número de polaridad se escribirá:

$$q=<q_{2^n-2}><q_{2^n-3}\ldots q_{2^{n-1}-1}><q_{2^{n-1}-2}\ldots q_0> \tag{2.64}$$

o bien, nombrando a cada uno de estos tres bloques como q_A, q_B y q_C:

$$q=<q_Aq_Bq_C> \tag{2.65}$$

Con estas definiciones, la matriz de transformación Q_q puede calcularse de la siguiente forma:

$$Q_q=K_{q_A}\circ[Q_{q_B},Q_{q_C}] \tag{2.66}$$

donde Q_{q_B} y Q_{q_C} son matrices de transformación de n-1 variables que a su vez se calculan utilizando nuevamente las expresiones (2.64), (2.65) y (2.66). Se trata, pues, de un proceso recursivo en el que se va rebajando el número de variables en 1 cada vez que se realiza un paso, y que finaliza cuando se llega a n=2; caso particular que se ha analizado en detalle anteriormente.

En el caso de estas formas canónicas, el cálculo del vector peso (el cálculo del vector extendido, por definición, es independiente de si se están tratando formas MPRM o PKRM) se realizará utilizando otra matriz de transformación distinta:

$$S'=R_nE \tag{2.67}$$

En esta última expresión, la multiplicación es sobre los reales y S' contendrá el número de productos necesario para sintetizar cada una de las PKRM.

La matriz de transformación R_n se construye de la siguiente forma:

Se definen:

$$R_1=\begin{bmatrix}1&0&1\\0&1&1\\1&1&0\end{bmatrix} \tag{2.68}$$

y

$$R_1^{(0)}=\begin{bmatrix}1 & 0 & 1\\ 1 & 0 & 1\\ 1 & 0 & 1\end{bmatrix},\ R_1^{(1)}=\begin{bmatrix}0 & 1 & 1\\ 0 & 1 & 1\\ 0 & 1 & 1\end{bmatrix},\ R_1^{(2)}=\begin{bmatrix}1 & 1 & 0\\ 1 & 1 & 0\\ 1 & 1 & 0\end{bmatrix} \tag{2.69}$$

En general:

$$R_k^{(j)}=\begin{bmatrix}\textit{fila } j \textit{ de } R_k\\ \textit{fila } j \textit{ de } R_k\\ \cdots\\ \textit{fila } j \textit{ de } R_k\end{bmatrix} \tag{2.70}$$

donde la matriz $R_k^{(j)}$ tiene el mismo número de filas que R_k y $0\leq j\leq 3^k-1$.

Si se define $N=3^{2^{n-1}-1}-1$, la matriz R_n se construye según la estructura que se presenta a continuación:

$$R_n=\begin{bmatrix}R_{n-1} & 0 & R_{n-1}^{(0)}\\ R_{n-1} & 0 & R_{n-1}^{(1)}\\ \cdots & \cdots & \cdots\\ R_{n-1} & 0 & R_{n-1}^{(N)}\\ 0 & R_{n-1} & R_{n-1}^{(0)}\\ 0 & R_{n-1} & R_{n-1}^{(1)}\\ \cdots & \cdots & \cdots\\ 0 & R_{n-1} & R_{n-1}^{(N)}\\ R_{n-1} & R_{n-1}^{(0)} & 0\\ R_{n-1} & R_{n-1}^{(1)} & 0\\ \cdots & \cdots & \cdots\\ R_{n-1} & R_{n-1}^{(N)} & 0\end{bmatrix} \tag{2.71}$$

Como es fácil observar, se vuelve a obtener un cálculo de tipo recursivo. A título de ejemplo se muestra seguidamente la matriz R_2:

$$R_2 = \begin{bmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{2.72}$$

Según ha podido comprobarse, el tratamiento de las PKRM resulta mucho más complejo que el de las KRM. No obstante, es una familia de formas canónicas mucho más amplia (lo que presenta ventajas de cara a la minimización, como se indicará en el Apartado 2.5.3), y presenta una gran regularidad en su estructura, lo que hace pensar en la posibilidad de desarrollar algoritmos rápidos para su cálculo. Por otra parte, este elevado número de formas PKRM existente impide el realizar un tratamiento simultáneo de todas las PKRM en un ordenador en cuanto se tengan funciones de más de 4 variables (para n=6 hay del orden de 10^{30} PKRMs). Finalmente, nótese que las PKRM incluyen a las MPRM (y por tanto a las FPRM).

2.4.1.4. Formas canónicas cuasi-Kronecker (QKRM)

En ese apartado se va a presentar una última familia de formas canónicas de Reed-Muller, aún más amplia que la de las formas PKRM. La obtención de estas formas canónicas presenta una gran complejidad, y debido a ello, se va a estudiar únicamente el caso n=2. No obstante, este caso particular muestra perfectamente el proceso de cálculo a seguir para obtener todos los miembros de esta familia.

Aunque existe un gran número de expresiones PKRM, éstas no constituyen todas las formas canónicas posibles. Cualquier forma canónica de dos variables puede construirse mediante la combinación de 4 productos de entre los nueve que resultan de hacer el siguiente producto de Kroneker:

$$f=\left[1 \;\; \overline{x_2} \;\; x_2\right]*\left[1 \;\; \overline{x_1} \;\; x_1\right]=\left[1 \;\; \overline{x_1} \;\; x_1 \;\; \overline{x_2} \;\; \overline{x_2}\overline{x_1} \;\; \overline{x_2}x_1 \;\; x_2 \;\; x_2\overline{x_1} \;\; x_2x_1\right] \tag{2.73}$$

Existirán por tanto:

$$\binom{9}{4}=126 \tag{2.74}$$

posibles elecciones, de las cuales sólo 81 corresponden a posibles formas canónicas [GRE91] (véase por ejemplo que la elección de 1, x_1, $\overline{x_1}$, x_2 nunca corresponderá a una forma canónica puesto que $1 \oplus x_1=\overline{x_1}$).

Estas 81 formas canónicas pueden dividirse en tres grupos de 27 formas cada uno según se observa en la Tabla 2.2. El primer grupo corresponde a las 27 formas PKRM ($0 \le q \le 26$), que incluyen a las formas KRM ($0 \le m \le 8$) (y por tanto a las FPRM ($0 \le p \le 3$)); y a la RM ($p=m=q=0$). Las 18 formas PKRM que no son KRM del primer grupo pueden transformarse en un nuevo conjunto de 18 formas intercambiando el orden de las variables x_2 y x_1. Estas nuevas formas aparecen marcadas con un asterisco en los grupos 2 y 3 (nótese que la aplicación de estas permutaciones a una función KRM no introduce ninguna modificación).

Por ejemplo, la forma PKRM con q=14, tendrá como vector base el siguiente:

$$\left[1 \;\; \overline{x_2}\right] \circ \left[\left[1 \;\; \overline{x_1}\right], \left[\overline{x_1} \;\; x_1\right] \right]=\left[\overline{x_1} \;\; x_1 \;\; \overline{x_2} \;\; \overline{x_2}\overline{x_1}\right] \tag{2.75}$$

y si se intercambian las variables x_1 y x_2 se obtiene el nuevo vector base:

$$\left[1 \;\; \overline{x_1}\right] \circ \left[\left[1 \;\; \overline{x_2}\right], \left[\overline{x_2} \;\; x_2\right] \right]=\left[\overline{x_2} \;\; x_2 \;\; \overline{x_1} \;\; \overline{x_2}\overline{x_1}\right]=\left[\overline{x_1} \;\; \overline{x_2} \;\; \overline{x_2}\overline{x_1} \;\; x_2\right] \tag{2.76}$$

Se puede así introducir una nueva familia, extensión de las PKRM con 27+18=45 formas que denominaremos formas cuasi-Kronecker Reed Muller (QKRM). Estas formas canónicas no tienen una estructura claramente definida como las PKRM, aunque pueden utilizarse los mismos procedimientos para su manipulación: Se puede extender la matriz R_2 y el vector extendido S' añadiendo una serie de filas que correspondan a las nuevas formas.

p	m	q	Grupo 1	q'		Grupo 2	q'		Grupo 3
0	0	0 x	1 0 1 0 0 0 1 0 1	3 x	*	1 0 1 0 0 1 1 0 0	1 x	*	1 0 1 1 0 0 0 0 1
		1 x	1 1 0 0 0 0 1 0 1	x		1 1 0 0 0 1 1 0 0	x		1 1 0 1 0 0 0 0 1
		2	0 1 1 0 0 0 1 0 1		+	0 1 1 0 0 1 1 0 0		+	0 1 1 1 0 0 0 0 1
		3 x	1 0 1 0 0 0 1 1 0	x		1 0 1 1 0 0 0 1 0	x		1 0 1 0 1 0 1 0 0
1	1	4 x	1 1 0 0 0 0 1 1 0	10 x	*	1 1 0 1 0 0 0 1 0	12 x	*	1 1 0 0 1 0 1 0 0
		5	0 1 1 0 0 0 1 1 0		+	0 1 1 1 0 0 0 1 0		+	0 1 1 0 1 0 1 0 0
		6	1 0 1 0 0 0 0 1 1			1 0 1 0 1 0 0 0 1			1 0 1 0 0 1 0 1 0
		7	1 1 0 0 0 0 0 1 1			1 1 0 0 1 0 0 0 1			1 1 0 0 0 1 0 1 0
	2	8	0 1 1 0 0 0 0 1 1	19	*	0 1 1 0 1 0 0 0 1	21	*	0 1 1 0 0 1 0 1 0
2	3	9 x	1 0 1 1 0 1 0 0 0	7	*	1 0 0 1 0 1 0 0 1	5	*	0 0 1 1 0 1 1 0 0
		10 x	1 1 0 1 0 1 0 0 0			1 0 0 1 1 0 0 0 1		+	0 0 1 1 1 0 1 0 0
		11	0 1 1 1 0 1 0 0 0			1 0 0 0 1 1 0 0 1		+	0 0 1 0 1 1 1 0 0
		12 x	1 0 1 1 1 0 0 0 0		+	0 1 0 1 0 1 1 0 0			1 0 0 1 0 1 0 1 0
3	4	13 x	1 1 0 1 1 0 0 0 0	14	*	0 1 0 1 1 0 1 0 0	16	*	1 0 0 1 1 0 0 1 0
		14	0 1 1 1 1 0 0 0 0		+	0 1 0 0 1 1 1 0 0			1 0 0 0 1 1 0 1 0
		15	1 0 1 0 1 1 0 0 0			0 0 1 1 0 1 0 1 0		+	0 1 0 1 0 1 0 0 1
		16	1 1 0 0 1 1 0 0 0		+	0 0 1 1 1 0 0 1 0			0 1 0 1 1 0 0 0 1
	5	17	0 1 1 0 1 1 0 0 0	23	*	0 0 1 0 1 1 0 1 0	25	*	0 1 0 0 1 1 0 0 1
	6	18	0 0 0 1 0 1 1 0 1	2	*	0 0 1 1 0 0 1 0 1	6	*	1 0 0 0 0 1 1 0 1
		19	0 0 0 1 1 0 1 0 1		+	0 0 1 1 0 0 1 1 0			1 0 0 0 0 1 1 1 0
		20	0 0 0 0 1 1 1 0 1		+	0 0 1 1 0 0 0 1 1			1 0 0 0 0 1 0 1 1
		21	0 0 0 1 0 1 1 1 0			1 0 0 0 1 0 1 0 1		+	0 1 0 1 0 0 1 0 1
	7	22	0 0 0 1 1 0 1 1 0	15	*	1 0 0 0 1 0 1 1 0	11	*	0 1 0 1 0 0 1 1 0
		23	0 0 0 0 1 1 1 1 0			1 0 0 0 1 0 0 1 1		+	0 1 0 1 0 0 0 1 1
		24	0 0 0 1 0 1 0 1 1		+	0 1 0 0 0 1 1 0 1			0 0 1 0 1 0 1 0 1
		25	0 0 0 1 1 0 0 1 1			0 1 0 0 0 1 1 1 0		+	0 0 1 0 1 0 1 1 0
	8	26	0 0 0 0 1 1 0 1 1	24	*	0 1 0 0 0 1 0 1 1	20	*	0 0 1 0 1 0 0 1 1

Tabla 2.2. Formas canónicas de Reed-Muller para 2 variables

En general, para n variables se pueden realizar $n!$ permutaciones en cada forma PKRM; sin embargo no todas dan lugar a nuevas formas (recuérdese cómo, en el caso de n=2, las KRM no producen nuevas formas), de manera que resulta muy difícil hacer una enumeración de las mismas, ó incluso saber el número de QKRM que hay para n variables. En cualquier caso, una cota superior para el número de formas QKRM vendrá dada por:

$$n_{QKRM} < n! \cdot 3^{2^n-1} \tag{2.77}$$

Según lo estudiado, puede concluirse que el tratamiento de las formas QKRM resulta mucho más complejo que el de las PKRM, y su utilidad es bastante reducida (para n=3 resulta prácticamente inviable su construcción). No obstante, si se encuentra un procedimiento sistemático y efectivo para su manipulación, el elevado número de formas QKRM podría ser

de utilidad para la construcción de procedimientos de minimización AND-EXOR.

2.4.1.5. Otras formas canónicas

Se presentan ahora otras formas canónicas, con la sóla intención de enumerarlas, puesto que casi carecen de estructura y por tanto su identificación y manipulación resulta prácticamente inviable.

2.4.1.5.1. Formas skew

Estas formas no pueden obtenerse a partir del vector extendido, y por tanto no es posible utilizar los procedimientos de cálculo que se han estudiado anteriormente. Estas formas canónicas se obtienen a partir de modificaciones inconsistentes ("skew") de formas PKRM o QKRM. Para dos variables existen 16 de estas formas, que se encuentran señaladas en la Tabla 2.2 con el signo +.

2.4.1.5.2. Formas residuales

En el caso de n=2, según puede observarse en la Tabla 2.2, quedan 20 formas canónicas sin marcar que prácticamente carecen de estructura y no pueden obtenerse a partir de las PKRM ni QKRM. A estas formas se las denomina formas residuales.

Tanto las formas skew como las formas residuales resultan muy difíciles de enumerar y por tanto su utilización es muy limitada, aunque existen algunos intentos para su clasificación, como el que se presenta en [GRE91], utilizando matrices que operan sobre el vector general f definido en (2.73).

2.4.1.5.3. Formas inconsistentes

Se definen en [COH62] y se obtienen a partir de las formas canónicas de RM y realizando todas las posibles combinaciones que resultan de introducir complementos en todos los literales. Por ejemplo; en el caso de 2 variables se partirá del vector base $[1\ x_1\ x_2\ x_2x_1]$ y se obtendrá un total de 16 formas inconsistentes, que pueden verse en (2.78), y que se encuentran marcadas con una 'x' en la Tabla 2.2.

$$\begin{array}{cccc}
[1\ x_1\ x_2\ x_2x_1] & [1\ \overline{x_1}\ x_2\ \overline{x_2}x_1] & [1\ \overline{x_1}\ x_2\ x_2\overline{x_1}] & [1\ \overline{x_1}\ x_2\ \overline{x_2}\,\overline{x_1}] \\
[1\ x_1\ \overline{x_2}\ x_2x_1] & [1\ \overline{x_1}\ \overline{x_2}\ x_2\overline{x_1}] & [1\ x_1\ x_2\ x_2\overline{x_1}] & [1\ \overline{x_1}\ \overline{x_2}\ \overline{x_2}x_1] \\
[1\ x_1\ x_2\ \overline{x_2}x_1] & [1\ \overline{x_1}\ \overline{x_2}\ \overline{x_2}\,\overline{x_1}] & [1\ x_1\ x_2\ \overline{x_2}\,\overline{x_1}] & [1\ x_1\ \overline{x_2}\ x_2\overline{x_1}] \\
[1\ \overline{x_1}\ \overline{x_2}\ x_2x_1] & [1\ x_1\ \overline{x_2}\ \overline{x_2}x_1] & [1\ \overline{x_1}\ x_2\ x_2\overline{x_1}] & [1\ x_1\ \overline{x_2}\ \overline{x_2}\,\overline{x_1}]
\end{array} \qquad (2.78)$$

Estas formas inconsistentes ya no siguen las inclusiones que venían dándose en las

formas canónicas que se han visto hasta ahora. Por ejemplo, para el caso de 2 variables se tiene que de estas 16 formas, 4 son formas FPRM, otras 4 son PKRM, otras 4 QKRM y las 4 restantes son formas residuales.

2.4.1.5.4. Formas canónicas de polaridad mixta restringida (CRMP)

Las formas canónicas de polaridad mixta restringida (CRMP, [CSA93]) o formas generalizadas de Reed-Muller (GRM, [DAV78]) son las más interesantes de aquellas que no poseen estructura de Kronecker. Para su introducción se va a realizar una comparación entre las FPRM y las CRMP.

Si se representan los literales por x_i^c (x_i^c puede ser x_i o $\overline{x_i}$), una función de n variables puede expresarse en general como suma de productos módulo-2 de la siguiente forma:

$$f(x_1,x_2,...,x_n)=g_0\oplus g_1x_1^c\oplus...\oplus g_nx_n^c\oplus g_{n+1}x_1^cx_2^c\oplus...\oplus g_{x^n-1}x_1^cx_2^c...x_n^c \qquad (2.79)$$

donde g_i=0 ó 1, y x_i^c=x_i ó $\overline{x_i}$.

En una expresión FPRM de $f(x_1,x_2,...,x_n)$, una variable x_i puede aparecer complementada y sin complementar pero no en ambas simultaneamente.

En cambio, en una expresión CRMP, cada variable puede aparecer complementada y sin complementar, pero existe un único coeficiente $g_i\neq0$ para cada subconjunto de variables de un término. De esta definición es inmediato deducir que las CRMP incluyen a las FPRM y a las formas inconsistentes. Sin embargo, no existe ninguna relación entre las CRMP y las MPRM, aunque el nombre pudiera hacer pensar lo contrario. Existen CRMPs que no son MPRM y MPRMs que no son CRMP. Por ejemplo, considérese la expresión:

$$f(x_1,x_2)=x_1x_2\oplus x_1\overline{x_2} \qquad (2.80)$$

Claramente se trata de una MPRM generada a partir del vector base [$\overline{x_2}$ x_2 $x_1\overline{x_2}$ x_1x_2] que se obtiene a partir del producto de Kronecker [1 x_1]*[$\overline{x_2}$ x_2]; pero no es una CRMP puesto que el subconjunto de literales $x_1^cx_2^c$ aparece dos veces.

Asimismo, la expresión:

$$f(x_1,x_2)=1\oplus x_1\oplus\overline{x_2}\oplus\overline{x_1}x_2 \qquad (2.81)$$

es una CRMP pero no es una MPRM. Por tanto, las CRMP no están incluidas en las MPRM, como se quería comprobar. En cuanto a las formas PKRM, evidentemente las CRMP no incluyen a las PKRM puesto que no incluían a las MPRM, y además, si se tiene en cuenta que la función:

$$f(x_1,x_2,x_3)=\overline{x_1}\,\overline{x_2}\oplus x_2\overline{x_3}\oplus x_3x_1 \qquad (2.82)$$

es una forma CRMP pero no es PKRM ([SAS95]), se concluye que las PKRM tampoco incluyen a las CRMP.

Las CRMP pueden representarse mediante un vector binario de 2^n componentes que corresponderán a los coeficientes g_i. Nótese sin embargo que hay que especificar para cada componente la polaridad que se va a utilizar. Por ejemplo, si se tiene

$$[g_1 \ g_a \ g_b \ g_{ab}]=[0 \ 1 \ 0 \ 0] \tag{2.83}$$

se está representando la expresión:

$$0\cdot 1 \oplus 1\cdot a \oplus 0\cdot b \oplus 0\cdot ab \tag{2.84}$$

En cambio, si se tiene:

$$[g_1 \ g_{\bar{a}} \ g_b \ g_{ab}]=[0 \ 1 \ 0 \ 0] \tag{2.85}$$

se está representando $\bar{a}$.

Así, un mismo vector binario corresponde a formas CRMP distintas, con lo que esta notación no es adecuada para hacer consideraciones generales acerca de las CRMP.

Finalmente se va a obtener el número total de formas CMRP que existen para una función de n variables. Para una función booleana de n variables existen:

$$\binom{n}{i} \tag{2.86}$$

subconjuntos de i variables. En cada subconjunto se pueden elegir dos polaridades para cada variable (complementada o sin complementar) con lo que se tienen 2^i polaridades distintas para cada subconjunto. Por tanto, el número total de CRMPs sería:

$$\prod_{i=1}^{n} (2^i)^{\binom{n}{i}}=\prod_{i=1}^{n} 2^{i\binom{n}{i}}=2^{\sum_{i=1}^{n} i\binom{n}{i}}=2^{n2^{n-1}} \tag{2.87}$$

En el Apartado 3.3.3, donde se estudia un procedimiento para su minimización, se profundizará más en este tipo de formas canónicas.

2.4.2. Formas ESOP

En los apartados anteriores se han estudiado las familias de formas canónicas de Reed-Muller. Aunque existen familias muy amplias, como las de las formas QKRM o CRMP, éstas no incluyen todas las posibles expresiones AND-EXOR de una función. La clase más general de expresión AND-EXOR de una función se denomina ESOP (Exclusive Sum Of Products, [SAS95]) y viene dada por:

$$f(x_1,x_2,...,x_n)=\bigoplus x_1^* x_2^* ... x_n^* \tag{2.88}$$

donde x_i^* puede ser 1, x_i ó $\bar{x}_i$. Notese que la sumatoria no tiene índices, es decir, el número

de términos producto que aparecere en la expresión es indeterminado, dependerá en cada caso de la función de conmutación que se esté utilizando. Las formas ESOP incluyen a todas las familias de formas canónicas mostradas en los apartados anteriores, de forma que puede construirse la Figura 2.4, que presenta gráficamente las relaciones existentes entre las distintas formas de representación AND-EXOR.

Una vez introducidas las representaciones AND-EXOR, en el apartado siguiente se plantea el problema de encontrar la expresión con menor número de términos producto de una función de conmutación dada.

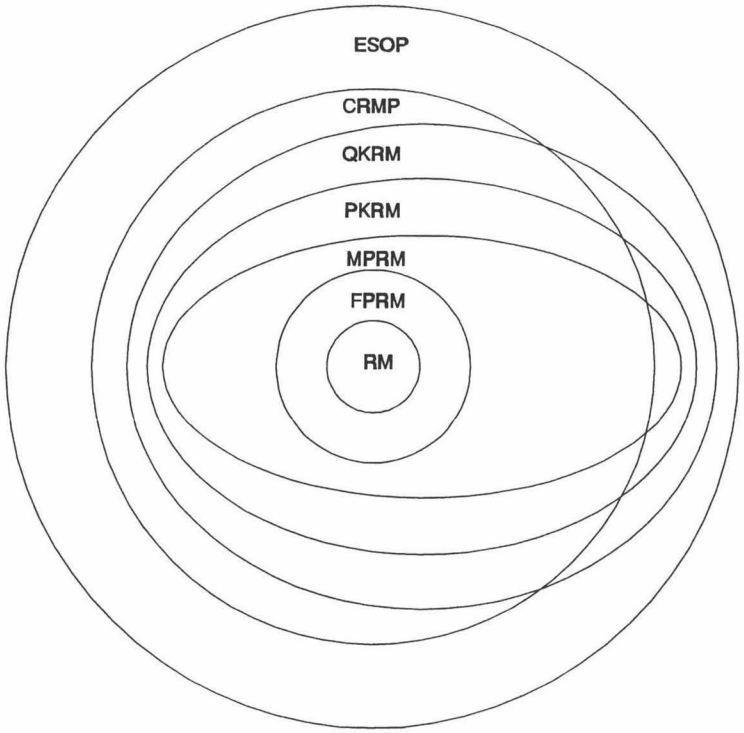


Fig. 2.4. Representaciones AND-EXOR

2.5. EL PROBLEMA DE LA MINIMIZACIÓN AND-EXOR

Al igual que en el Álgebra de Boole, en la lógica AND-EXOR se plantea el problema de encontrar la expresión mínima para la función que se desea sintetizar. En primer lugar debe definirse qué es lo que se va a entender por expresión mínima AND-EXOR:

Definición 2.1: *Se define la expresión mínima AND-EXOR de una función de conmutación f como la expresión ESOP de la misma que:*

1) Contenga menor número de términos producto que cualquier otra ESOP de

esa función.

2) *Teniendo el mínimo número de términos producto, incluya menor número de literales que cualquier otra ESOP.*

En la lógica AND-OR el problema de encontrar la expresión mínima puede considerarse resuelto utilizando los mapas de Karnaugh o el procedimiento de Quine-McCluskey, y en principio, cabría intentar aplicar estos mismos métodos a la minimización AND-EXOR. Sin embargo, como se verá más adelante, en 3.3.2, la minimización basada en estos procedimientos resulta ser aproximada y no ofrece unos resultados excesivamente buenos.

Lo ideal sería encontrar un procedimiento exacto de minimización para las expresiones ESOP, pero actualmente resulta inviable desde el punto de vista práctico. Una opción más sencilla puede consistir en tratar de minimizar sobre familias de formas canónicas en lugar de hacerlo sobre las ESOP (esto, según la Figura 2.4, lleva a la generación de procedimientos aproximados), con el fin de disponer de procedimientos utilizables en la práctica. En los siguientes apartados se trata esta cuestión.

2.5.1. Minimización dentro de las FPRM

Un procedimiento aproximado de minimización AND-EXOR podría consistir en calcular las 2^n formas canónicas FPRM que le correspondan a la función a sintetizar, y escoger aquella que contenga un menor número de términos. Este procedimiento es perfectamente viable para valores elevados de n (para n=20 habría que calcular algo más de un millón de formas, lo que no representa un gran problema para los ordenadores actuales), aunque no proporciona buenos resultados. En efecto, el número de formas canónicas FPRM es muy pequeño en relación con el número de formas ESOP existente, de manera que se están explorando muy pocas expresiones y existen pocas probabilidades de que una de ellas sea la mínima. Así, una mejora podría consistir en tratar de minimizar dentro de las MPRM.

2.5.2. Minimización dentro de las MPRM

De forma análoga al caso de las FPRM, puede plantearse un procedimiento para minimizar dentro de las MPRM obteniendo todas las formas. Sin embargo, al existir 3^n formas, ya no resulta tan viable efectuar el cálculo de todas ellas (aunque los resultados que se obtengan serán mejores), y es preciso recurrir a algunas herramientas adicionales. La utilización de los vectores extendido y peso (Apartado 2.4.1.2.3) permite obtener la polaridad de la forma canónica MPRM con menor número de productos sin necesidad de calcular todas las formas. Después, una vez conocida la polaridad mínima, sólo es necesario calcular la forma canónica asociada a esa polaridad. En el Apartado 4.5.2 se detalla todo este procedimiento.

2.5.3. Minimización dentro de las PKRM

En el caso de las formas PKRM, el tratamiento matricial se complica considerablemente, como se ha puesto de manifiesto en 2.4.1.3, y se suele preferir utilizar técnicas basadas en diagramas de decisión. Concretamente se utilizan diagramas de decisión ternarios EXOR (ETDD), y en [SAS93c] puede verse un procedimiento de minimización basado en los citados árboles. La minimización dentro de esta familia de formas canónicas proporciona resultados muy aceptables, puesto que se efectúa una búsqueda entre un gran número de expresiones (Apartado 2.4.1.3).

2.5.4. Minimización dentro de las QKRM

La minimización dentro de esta familia de formas canónicas no resulta ser de interés, puesto que el tratamiento de las mismas es prácticamente tan complejo como el de las expresiones ESOP. Por ello, aunque esta familia de formas canónicas es muy amplia (Apartado 2.4.1.4), no se ha dedicado demasiado esfuerzo a desarrollar procedimientos de minimizacion para ella.

2.5.5. Minimización dentro de las CRMP

La minimización dentro de esta familia de formas canónicas es de gran interés por su amplitud y por la facilidad para el test [PER90] que presentan los diseños basados en las mismas. Los procedimientos de minimización para este tipo de formas suelen ser parecidos a los de las expresiones generales ESOP, o sea, procedimientos de tipo heurístico. No obstante, se han realizado procedimientos basados en diagramas binarios de decisión (BDD) [ESC95] [SAS93d] [SAS94], que son aplicables para valores de n=6 ó 7 como mucho. En el Apartado 3.3.3 se presenta un procedimiento de minimización para este tipo de formas canónicas.

2.5.6. Minimización ESOP

El problema de encontrar la expresión ESOP mínima es de una enorme complejidad. Nada más que el planteamiento del mismo resulta dificultoso porque no está claro de qué manera pueden obtenerse todas las formas AND-EXOR de una función de conmutación. Si se dispusiera de un método para obtener sistemáticamente todas estas expresiones, se dispondría de una solución (al menos teórica) del problema. Existen algunos procedimientos exactos de minimización AND-EXOR [PER90][PAP79], pero no son de aplicación práctica, bien porque sean aplicables para un número de variables muy reducido, bien porque el tiempo de ejecución que precisarían es completamente inalcanzable para los ordenadores actuales. Es preciso, por tanto, recurrir a procedimientos aproximados. En el Capítulo 3 se recogen los procedimientos más importantes, y en el Capítulo 4 se desarrolla un nuevo procedimiento para

la resolución de este problema utilizando reglas de reescritura (Apartado 3.2.3) dirigidas mediante un proceso de Enfriamiento Simulado.

2.6. CONCLUSIONES

En este Capítulo se ha introducido una descripción alternativa de las funciones de conmutación, utilizando las primitivas AND y EXOR. Estas primitivas dan lugar a una estructura matemática denominada GF(2), de forma análoga a como las primitivas AND-OR originaban el Álgebra de Boole. Asimismo, se han presentado las distintas formas usuales que se utilizan para expresar algebráicamente las funciones en lógica AND-EXOR, introduciendo las familias de formas canónicas de Reed-Muller (RM, FPRM, MPRM, PKRM, QKRM, CRMP), y la forma general de expresar una función como suma de productos sobre GF(2) (ESOP). A partir de estas definiciones se ha abordado la minimización AND-EXOR, esbozando una posible solución para cada tipo de formas canónicas, y mostrando la gran dificultad que implica el desarrollo de un procedimiento de minimización para las formas generales ESOP. En el próximo Capítulo se va a efectuar una revisión de los principales procedimientos existentes en la literatura para la minimización AND-EXOR.

CAPÍTULO 3

PROCEDIMIENTOS DE MINIMIZACIÓN AND-EXOR

En este Capítulo se presentan los métodos de minimización AND-EXOR más representativos de los que aparecen en la literatura. La primera parte se dedica a la descripción de procedimientos que obtienen la expresión ESOP con menor número de términos producto (procedimientos exactos), y la segunda, a mostrar una serie de procedimientos aproximados, (no siempre obtienen la expresión mínima), pero que, según se mostrará, son de una mayor utilidad práctica.

3.1. INTRODUCCIÓN

En el Capítulo anterior se han estudiado diversas formas de expresar una función booleana como suma de productos módulo-2, y se ha introducido el problema de obtener la expresión mínima, bien dentro de alguna de las familias de formas canónicas descritas, o bien dentro del conjunto global de las expresiones ESOP. En este Capítulo se van a describir las soluciones adoptadas por algunos autores para este problema.

Los procedimientos de minimización AND-EXOR pueden clasificarse en dos grupos, uno en el que se pretende encontrar el mínimo absoluto, y un segundo grupo que utiliza métodos aproximados que tratan de acercarse al mínimo realizando el menor número de operaciones posible.

Los procedimientos exactos suelen consistir en búsquedas exhaustivas o en la utilización sistemática de reglas de reescritura. En cualquiera de los dos casos se trata de procedimientos que requieren un alto tiempo de ejecución y gran volumen de memoria para alcanzar la solución final. Los requerimientos de estos procedimientos los hacen inviables en cuanto el número n de variables es moderadamente alto ($n>6$), lo que justifica el uso de procedimientos aproximados.

Los procedimientos aproximados, generalmente se basan en la utilización de las formas canónicas estudiadas en el Capítulo 2, o bien en la utilización de métodos tabulares. Todo esto se verá en detalle al estudiar los distintos procedimientos.

A continuación se incluye un esquema con una breve explicación de los procedimientos, para un mejor seguimiento del capítulo:

Apartado 3.2: PROCEDIMIENTOS EXACTOS DE MINIMIZACIÓN. Se describen cuatro procedimientos exactos de minimización que se estudian en los siguientes apartados:

Apartado 3.2.1: Minimización por división en subfunciones. Se realiza el estudio de un procedimiento de minimización basado en la división de la función de conmutación a minimizar en subfunciones. Se da una serie de resultados teóricos justificando el procedimiento, y se estudian las limitaciones que presenta.

Apartado 3.2.2: Minimización mediante funciones de decisión. Se introduce la función de decisión H de Helliwell como herramienta para la descripción del problema de la minimización AND-EXOR. Posteriormente se desarrollan tres procedimientos (Apartados 3.2.2.2 al 3.2.2.4) para optimizar esta función de decisión y obtener así una expresión mínima exacta de la función que se desea minimizar.

Apartado 3.2.3: Minimización exacta utilizando reglas algebraicas de reescritura. Se da un teorema acerca de la convergencia de los procedimientos de minimización utilizando reglas algebraicas de reescritura y se introducen algunos conjuntos de reglas convergentes, discutiendo el problema que implica el llegar a una solución mínima utilizando tales conjuntos.

APARTADO 3.3: PROCEDIMIENTOS APROXIMADOS DE MINIMIZACION. Se estudian seis procedimientos aproximados de minimización. Se trata de procedimientos heurísticos que tratan de encontrar una solución buena en un tiempo razonable. Se presentan en los siguientes apartados:

Apartado 3.3.1: Minimización por vectores extendidos. Se trata de un procedimiento basado en el concepto de forma canónica. Tiene, más que nada, una importancia teórica, abriendo múltiples posibilidades de realización de procedimientos heurísticos de minimización.

Apartado 3.3.2: Método tabular. Es un procedimiento aproximado que se basa en la misma idea que el método de Quine-McCluskey para funciones booleanas. Es por tanto una adaptación de tal procedimiento para la lógica AND-EXOR.

Apartado 3.3.3: Minimización utilizando CRMPs. Se presenta uno de los muchos procedimientos minimización existentes para las formas canónicas CRMPs, (Apartado 2.4.1.5.4). La idea es buscar la expresión mínima entre las formas CRMP, que presentan una cierta estructura que hace más fácil la búsqueda. El procedimiento es aproximado puesto que las CRMP son sólo una parte del conjunto de todas las formas ESOP.

Apartado 3.3.4: EXMIN2. Se trata de un procedimiento aproximado basado en reglas algebraicas de reescritura (véase el Apartado 3.2.4), bastante elaborado y que proporciona buenos resultados.

Apartado 3.3.5: EXORCISM-MV-2. Este procedimiento se basa en la aplicación de una operación denominada EXORLINK que incluye a las operaciones que utilizan EXORCISM [HEL88] [PER89], Hermes [SAU90] ó EXMIN2 [SAS93a]. Proporciona unos resultados excelentes con tiempos de ejecución muy reducidos.

Apartado 3.3.6: ACEM. El método de minimización ACEM se basa en la idea de identificar mediante una búsqueda exhaustiva, grupos de minterms lo más grandes posibles que sean cubiertos por un término AND-EXOR. Para conseguir tiempos de ejecución reducidos se utiliza una transformada de Hadamard-Walsh. Proporciona muy buenos resultados.

3.2. PROCEDIMIENTOS EXACTOS DE MINIMIZACIÓN

A continuación se van a analizar algunos procedimientos exactos de minimización AND-EXOR. Estos procedimientos, según se ha mencionado con anterioridad, no tienen demasiada utilidad debido a que el problema de obtener la expresión mínima ESOP constituye un problema NP-completo [GAR79], y los tiempos de ejecución necesarios para obtener una solución exacta son excesivos. No obstante, presentan un indudable interés teórico y sirven de base para el desarrollo de procedimientos aproximados.

3.2.1. Minimización por división en subfunciones ([PAP79])

El procedimiento que se presenta seguidamente es capaz de proporcionar la expresión mínima AND-EXOR exacta de una función de conmutación dada. Sin embargo, presenta una restricción importante, y es que el número de términos de la expresión mínima ha de ser $m<6$. Por tanto, el procedimiento carece de aplicación práctica (el número de casos al que es aplicable es muy reducido), pero tiene un interés histórico por ser el primero que abordaba de forma exacta la minimización ESOP.

A continuación se dan una serie de definiciones necesarias para la descripción del procedimiento.

Definición 3.1: *Si x_i^j corresponde a $\bar{x}_i$, x_i, 1 para $j=0,1,2$, respectivamente, un término producto genérico vendrá dado por:*

$$x_1^{j_1} x_2^{j_2} \ldots x_n^{j_n} \quad con \quad j_1,j_2,\ldots,j_n = 0,1,2 \tag{3.1}$$

Por definición, una subfunción f^i $(i=0,1,2)$ con respecto a la variable x_k de una función de conmutación $f(x_1,x_2,\ldots,x_n)$ viene dada por lo siguiente:

$$f^0 = f(x_1,\ldots,x_{k-1},0,x_k,\ldots,x_n) \tag{3.2}$$
$$f^1 = f(x_1,\ldots,x_{k-1},1,x_k,\ldots,x_n) \tag{3.3}$$
$$f^2 = f^0 \oplus f^1 \tag{3.4}$$

Definición 3.2: *Una función $f(x_1,x_2,\ldots,x_n)$ se dirá que es una suma de m términos producto si no puede expresarse como suma de $p<m$ productos, y existe una expresión de la misma como suma de m productos. En ese caso, a este número m se le denomina peso de la función f y se notará por $w(f)=m$ (es decir, el peso de la función es el número de términos de que consta su expresión mínima).*

Las subfunciones presentan una serie de propiedades, de las cuales las más útiles son:

$$f = \bar{x}_k f^0 \oplus x_k f^1 \tag{3.5}$$
$$f = f^0 \oplus x_k f^2 \tag{3.6}$$
$$f = f^1 \oplus \bar{x}_k f^2 \tag{3.7}$$
$$f^0 \oplus f^1 \oplus f^2 = 0 \tag{3.8}$$

Las tres primeras ecuaciones, (3.5), (3.6) y (3.7) muestran las tres formas en que una función puede escribirse a partir de sus subfunciones, y serán utilizadas abundantemente más adelante.

Seguidamente se enuncia una serie de teoremas, cuyas demostraciones pueden verse en [PAP79], y que son la base para el método de minimización:

Teorema 3.3: *Si una subfunción f^i de una función de conmutación $f(x_1,x_2,...,x_n)$ es cero, entonces todas las sumas de productos módulo-2 de la función f pueden obtenerse a partir de los de f^j con $j=1,0,1$ para $i=0,1,2$, respectivamente.*

Corolario 3.4: *Una función de conmutación $f(x_1,x_2,...,x_n)$ es un término producto si y solamente si una subfunción f^i de f es cero y la subfunción distinta de cero de f es un término producto.*

Teorema 3.5: *Una función de conmutación $f(x_1,x_2,...,x_n)$ con f^0, f^1, $f^2 \neq 0$ puede escribirse en la forma:*

$$f=x_1^a u(x_2,x_3,...,x_n)\oplus x_1^b v(x_2,x_3,...,x_n) \tag{3.9}$$

donde u y v son funciones de conmutación de n-1 variables; si y solamente si las funciones u y v son subfunciones de f (f^i y f^j, respectivamente) y $(a,b,i,j)=(1,0,1,0), (1,2,2,0), (2,0,1,2)$.

Corolario 3.6: *Una función de conmutación $f(x_1,x_2,...,x_n)$ con f^0, f^1, $f^2\neq 0$ es una suma módulo-2 de dos términos producto si y solamente si al menos dos de sus subfunciones son términos producto.*

Teorema 3.7: *Una función de conmutación $f(x_1,x_2,...,x_n)$ con f^0, f^1, $f^2\neq 0$ y $w(f)>m-1$ es una suma módulo-2 de m (m=3,4,5) términos producto si y solamente si para $i,j=0,1,2$ y $i\neq j$ se cumple:*

a) $w(f^i)+w(f^j)=m$ *ó*

b) $w(f^i)$, $w(f^j)\neq 1$, $w(f^i)+w(f^j)=m+1$ *y* $w(p\oplus f^j)=w(f^j)-1$
donde p es uno de los términos producto de las expresiones mínimas de la función f^i.

Corolario 3.8: *Una función de conmutación $f(x_1,x_2,...,x_n)$ con f^0, f^1, $f^2\neq 0$ es una suma módulo-2 de 3 términos producto si y solamente si para $i,j=0,1,2$ y $i\neq j$:*

a) $w(f^i)+w(f^j)=3$ *ó*

b) $w(f^i)=w(f^j)=2$ *y f^i, f^j tienen un término común en sus expresiones mínimas.*

Este conjunto de teoremas puede utilizarse con el objeto de implementar un procedimiento de minimización para funciones de conmutación escritas como suma de productos módulo-2, siempre y cuando el número de términos producto de su expresión mínima sea menor que seis. En tal procedimiento, la función a minimizar f se analizará dividiéndola en sus subfunciones f^0, f^1, f^2. Cada una de estas subfunciones se dividirá a su vez en sus propias subfunciones y así sucesivamente, formando un árbol denominado árbol de generación. La descomposición continuará hasta que las subfunciones producidas estén compuestas por un solo término producto. Si una de las subfunciones es cero, la descomposición continuará por las ramas correspondientes a las subfunciones distintas de cero

(ver Figura 3.1).

Las funciones y subfunciones se representan mediante vectores binarios, en los que cada bit representa la presencia o no de un minterm determinado. La función f de la figura 3.1 se representa por:

$$f=[01000001010010000110001110010010] \tag{3.10}$$

lo que significa que los minterms de la función f son:

$$f=\sum m(1,4,7,8,9,13,14,19,2,30) \tag{3.11}$$

donde la notación que se está usando para los minterms no es la usual, sino que por ejemplo, el minterm 1 corresponde al producto $\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4x_5$; y se representa en binario por 00001. Es decir, en la notación que se está usando, el bit menos significativo de la representación numérica del minterm corresponde a la variable con subíndice más alto. Este cambio de notación se debe simplemente a motivos de comodidad al escribir el árbol de generación de las subfunciones. Con esta representación es muy fácil encontrar las subfunciones de f: la representación vectorial de la subfunción f^1 es la mitad izquierda de la de f, f^0 es la parte derecha y f^2 viene dada por la suma módulo-2 bit a bit de los vectores que representan a f^0 y f^1.

Teniendo en cuenta todas estas consideraciones, el procedimiento para minimizar puede resumirse en los siguientes pasos:

Paso 1: *Construir el árbol de generación de f.*

Paso 2: *Encontrar el peso de todas las funciones del árbol de generación partiendo de las ramas y avanzando hacia el nodo principal, de acuerdo con los teoremas 3.2 a 3.8.*

Paso 3: *Expresar f como suma de productos módulo-2 utilizando las ramas adecuadas según convenga, tal y como se describe en el Teorema 3.5.*

Para comprender cómo se realizan todos los pasos y cómo se aplican los teoremas, a continuación se va a estudiar detalladamente la minimización de la función de 5 variables f que se encuentra descrita en la Ecuación (3.11) ó en forma vectorial en (3.10). El proceso de minimización de esta función sería el siguiente:

Paso 1: En la Figura 3.1 puede verse el árbol de generación de f. La construcción de este árbol, una vez puesta f en forma vectorial, es muy sencilla; f^1 corresponde a la mitad izquierda del vector que representa a f, f^0 correspondería a la mitad derecha; y f^2 se construye sumando módulo-2 bit a bit los vectores de f^1 y f^0. A continuación f^0, f^1, f^2 se vuelven a dividir en sus correspondientes subfunciones, repitiendo el proceso hasta que se cumplen las condiciones del Corolario 3.4; en cuyo caso se habrá descompuesto f en términos producto individuales.

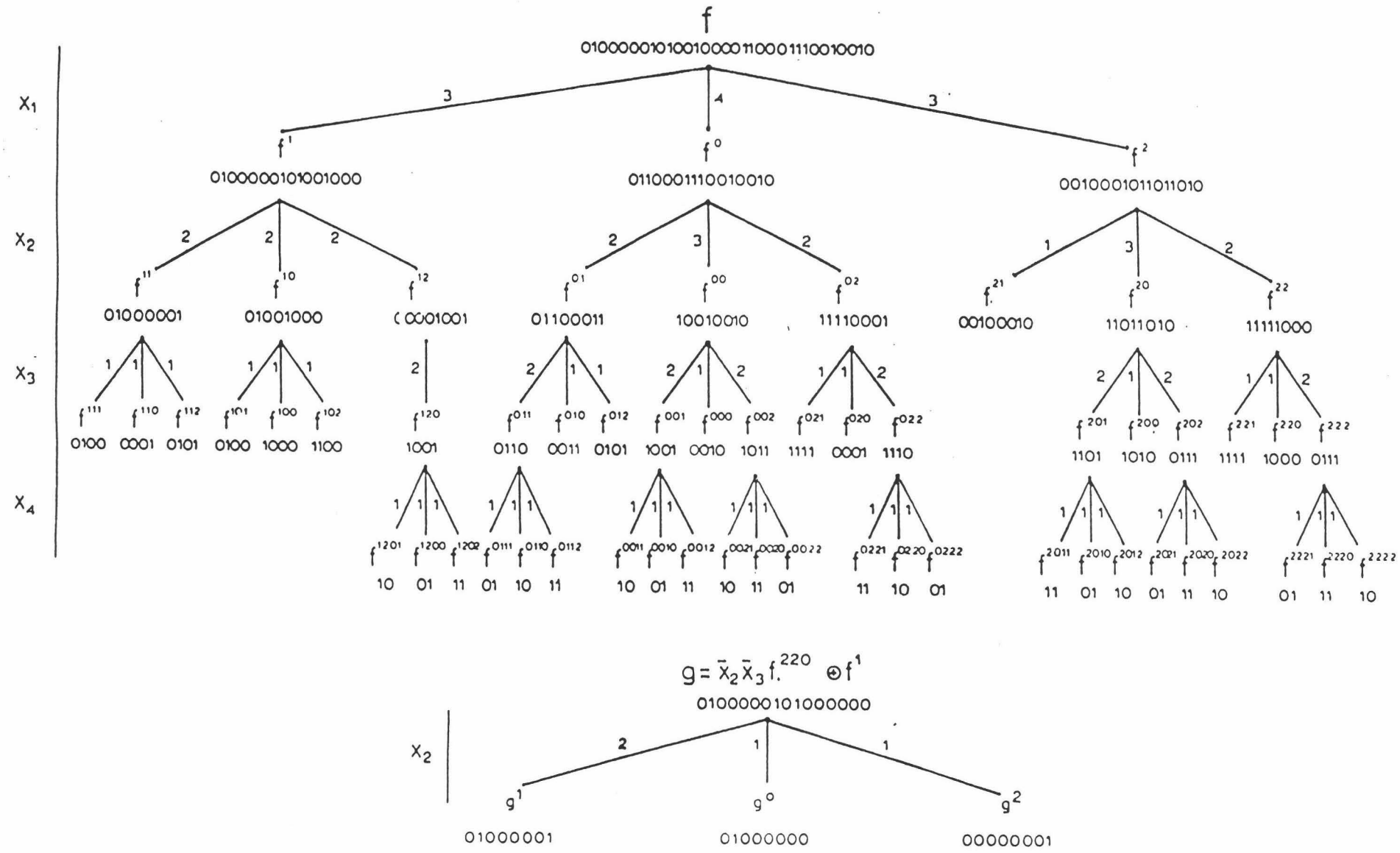


Fig. 3.1. Árbol de generación de la función f

Cada vez que se hace la división en subfunciones se está eliminando una variable (ver ecuaciones (3.5), (3.6) y (3.7)), que se escribe a la izquierda del árbol en cada nivel de subdivisión para después hacer más sencilla la escritura de los terminos producto.

Paso 2: Una vez escrito el árbol de generación hay que encontrar el peso de cada subfunción. Para calcular este peso en las zonas más alejadas del nodo raíz, suele bastar la aplicación directa de los corolarios 3.4 y 3.6. Sin embargo, en ramas cercanas a la raíz del árbol puede ser preciso aplicar el Teorema 3.7 ó el Corolario 3.8, siendo más complicada la determinación del peso. A continuación se estudian en detalle estos casos:

- $w(f^{00})=3$. Se tiene que $w(f^{000})+w(f^{002})=3$, cumpliéndose el apartado a) del Corolario 3.8.
- $w(f^{20})=3$. Idem que en el caso anterior.
- $w(f^{2})=3$. Idem que en los dos casos anteriores.
- $w(f^{1})=3$. En este caso no se cumple el apartado a) del Teorema 3.7, por tanto habrá que comprobar si se cumple b). Nótese que f^{11} y f^{10} son suma de dos términos producto y por consiguiente su expresión mínima puede construirse con la suma de dos subfunciones suyas cualesquiera (Todas sus subfunciones son términos producto), y así las expresiones mínimas de f^{11} y f^{10} pueden contener a f^{111} y f^{101}, respectivamente. Como $f^{111}=f^{101}$, f^{11} y f^{10} pueden tener un término común en sus expresiones mínimas, satisfaciendo el apartado b) del Corolario 3.8.
- $w(f^{0})=4$. $w(f^{0})$, en principio, puede ser 3 (si se cumplen las condiciones del Corolario 3.8) ó 4 (si no se cumplen dichas condiciones). El apartado a) no puede satisfacerse, y en todo caso habrá que estudiar b). Si se inspeccionan las síntesis mínimas de f^{01} y f^{02} no se encuentra ningún término común y por tanto, en efecto, $w(f^{0})=4$.
- $w(f)=5$. Este es la rama más complicada porque no es posible utilizar ningún corolario y es preciso recurrir a la aplicación directa del Teorema 3.7. En principio $w(f)=6$ puesto que $w(f^{1})+w(f^{2})=6$, pero hay que comprobar si $w(f)=5$. El apartado a) del Corolario 3.6 no se cumple, y habrá que ver si se satisface b). Se cumple que $w(f^{1})$, $w(f^{2})\neq 1$; $w(f^{1})+w(f^{2})=6$; y sólo falta comprobar $w(p\oplus f^{1})=w(f^{1})-1$ siendo p uno de los términos producto de una expresión mínima de f^{2}. La expresión mínima de f^{2} será, teniendo en cuenta que $w(f^{2})=3$ y aplicando el Teorema 3.5 para f^{21} y f^{22}:

$$f^{2}=f^{21}\oplus\overline{x_2}f^{22}=f^{21}\oplus\overline{x_2}(x_3f^{221}\oplus\overline{x_3}f^{220})=f^{21}\oplus\overline{x_2}x_3f^{221}\oplus\overline{x_2}\,\overline{x_3}f^{220} \qquad (3.12)$$

Por tanto, los términos producto de la expresión mínima de f^{2} son:

$$\left[f^{21}\quad \overline{x_2}x_3f^{221}\quad \overline{x_2}\,\overline{x_3}f^{220}\right] \qquad (3.13)$$

A fin de simplificar los cálculos, se van a escribir estos tres productos en forma vectorial y poder así operar directamente con la expresión vectorial de f^{1}. Para escribir estos productos en forma vectorial se parte de las expresiones vectoriales, ya conocidas, de las subfunciones f^{21}, f^{221} y f^{220}. Nótese por ejemplo que f^{221} es una función de 2 variables mientras que f^{1} y $\bar{x}_2x_3f^{221}$ lo son de 4. Hay por tanto que escribir la forma vectorial de $\bar{x}_2x_3f^{221}$ partiendo de la forma vectorial de f^{221}. A continuación se enuncia un teorema que indica cómo se escribe la expresión vectorial buscada.

Teorema 3.9: *Sea f^{i} (i=0,1,2) una subfunción de n-1 variables de la que se conoce su forma vectorial $M=[m_{2^{n-1}-1}\ \ldots\ m_0]$, y considérese la expresión $x_j^{i}f^{i}$. Entonces la expresión vectorial de $x_j^{i}M'$ para n variables es:*

a) Si j=0, $M'=[m_{2^{n-1}-1} \ \ldots \ m_0 \ 0 \ \ldots \ 2^{n-1}ceros \ \ldots \ 0]$
b) Si j=1, $M'=[0 \ \ldots \ 2^{n-1}ceros \ \ldots \ 0 \ m_{2^{n-1}-1} \ \ldots \ m_0]$
c) Si j=2, $M'=[m_{2^{n-1}-1} \ \ldots \ m_0 \ m_{2^{n-1}-1} \ \ldots \ m_0]$

Según este teorema, las formas vectoriales de los tres términos producto son:

$$f^{21}=[0010001000100010] \quad (3.14)$$
$$\bar{x}_2x_3f^{221}=[0000000011110000] \quad (3.15)$$
$$\bar{x}_2\bar{x}_3f^{220}=[0000000000001000] \quad (3.16)$$

A continuación se calculan las sumas de cada uno de estos tres términos producto y la subfunción f^1, es decir:

$$e=f^{21}\oplus f^1 \quad (3.17)$$
$$h=\bar{x}_2x_3f^{221}\oplus f^1 \quad (3.18)$$
$$g=\bar{x}_2\bar{x}_3f^{220}\oplus f^1 \quad (3.19)$$

para comprobar si w(e), w(h) o w(g)=2, en cuyo caso se cumplirá el apartado b) del Teorema 3.7. Por ejemplo, si se usa (3.19), se obtiene en forma vectorial:

$$g=[0100000101000000] \quad (3.20)$$

Si se descompone g como se muestra en la Figura 3.1, se observa que w(g)=2, con lo que si se llama $p=\bar{x}_2\bar{x}_3f^{220}$ se está en las condiciones del Teorema 3.7 y w(f)=5.

Paso 3: Finalmente queda escribir la expresión mínima de f. En el paso 2 se llegó a la conclusión de que w(f)=5, utilizando para ello las ramas de f^1 y f^2. Por tanto para la expresión mínima de f habrá que utilizar la Ecuación (3.7). Teniendo en cuenta que se ha utilizado g para hallar el peso mínimo, habrá que escribir f^1 en función de g. Si se opera en (3.19), se obtiene la siguiente expresión:

$$f^1=\bar{x}_2\bar{x}_3f^{220}\oplus g \quad (3.21)$$

Por otra parte, la expresión mínima de g, según la Figura 3.1, viene dada por:

$$g=g^0\oplus x_2g^2 \quad (3.22)$$

La expresión mínima de f^2 ya se ha hallado en (3.12), y por tanto, ya se puede escribir la expresión mínima de f:

$$f=g^0\oplus x_2g^2\oplus\bar{x}_2\bar{x}_3f^{220}\oplus\bar{x}_1f^{21}\oplus\bar{x}_1\bar{x}_2\bar{x}_3f^{220}\oplus\bar{x}_1\bar{x}_2x_3f^{221} \quad (3.23)$$

Si se reunen los términos que contienen a f^{220} se obtiene:

$$f=g^0\oplus x_2g^2\oplus x_1\bar{x}_2\bar{x}_3f^{220}\oplus\bar{x}_1f^{21}\oplus\bar{x}_1\bar{x}_2x_3f^{221} \quad (3.24)$$

y finalmente, sustituyendo las subfunciones por sus expresiones correspondientes:

$$f=x_3x_4\bar{x}_5 \oplus x_2\bar{x}_3\bar{x}_4\bar{x}_5 \oplus x_1\bar{x}_2\bar{x}_3x_4x_5 \oplus \bar{x}_1\bar{x}_4x_5 \oplus \bar{x}_1\bar{x}_2x_3 \qquad (3.25)$$

Como puede verse, en efecto la expresión obtenida para f tiene 5 términos producto, tal y como se había previsto al calcular el peso de la función.

El procedimiento descrito en este apartado proporciona la expresión mínima módulo-2 de una función de conmutación. Ahora bien, según el enunciado del Teorema 3.7, la aplicación del método se restringe a aquellas funciones cuya expresión mínima esté compuesta por menos de seis términos producto, por lo que el procedimiento sólo es aplicable a un número muy limitado de casos.

En el siguiente apartado se describe un procedimiento exacto mucho más general, aplicable ya a cualquier tipo de función AND-EXOR, y que se basa en la utilización de funciones de decisión.

3.2.2. Minimización mediante funciones de decisión

En el Apartado 3.2.1 se ha descrito un procedimiento de minimización AND-EXOR que permite encontrar la solución exacta para funciones de hasta 5 términos producto en su expresión mínima. En los apartados siguientes se van a describir procedimientos exactos para un número arbitrario de variables y términos, basados en la función de decisión H de Helliwell [PER90]. En 3.2.2.1 se define la función H y se establece el esquema de minimización a utilizar. En los apartados 3.2.2.2 a 3.2.2.4 se describen distintos procedimientos para optimizar la función de decisión y obtener el mínimo exacto a partir de la misma.

3.2.2.1. La función de decisión *H*

Los procedimientos de minimización que utilizan una función de decisión para obtener el mínimo AND-EXOR se basan en la misma idea que el método de Petrick para obtener la cobertura mínima de una función booleana. En el método de Petrick, una vez que se han obtenido los implicantes primos de la función a minimizar, se asigna una variable a cada uno de ellos y se construye una función de conmutación (función de decisión) de manera que cualquier combinación de valores de las variables que hagan verdadera a la función, corresponderá a una suma de implicantes primos solución del problema de minimización. La solución óptima se obtiene escogiendo el término producto de la función de Petrick con menor número de literales.

En la lógica AND-EXOR se presentan dos inconvenientes que impiden trasladar el método de Petrick directamente a la minimización ESOP:

a) El concepto de implicante primo no puede aplicarse en lógica AND-EXOR, con lo que no pueden restringirse de manera sencilla los términos producto que han de formar parte de la función de decisión.

b) La construcción de la función de decisión no puede realizarse con lógica booleana debido a que las sumas se efectúan con la primitiva EXOR. Como consecuencia, elegir el número mínimo de productos que satisfacen la función de decisión resultará más complejo que con la función de Petrick.

Según las consideraciones anteriores, el primer problema a resolver es seleccionar el conjunto de términos producto susceptibles de formar parte de la expresión AND-EXOR mínima, y por tanto, de la función de decisión.

En general, dada una función booleana $f(x_1,x_2,...,x_n)$, los términos producto AND-EXOR que forman parte de las soluciones del problema de minimización de f se obtienen como subconjuntos de:

$$\{x_1^{i_1},x_2^{i_2},...x_n^{i_n}\} \ \textit{donde}\ i_j\in\{0,1\}\ \textit{para}\ j=1,2,...,n \tag{3.26}$$

Cada uno de estos subconjuntos se va a nombrar por G_j, y con GG se notará al conjunto de todos los términos producto involucrados en las soluciones de f. En el peor de los casos GG tendrá 3^n elementos (cada variable puede encontrarse en tres estados; complementada, sin complementar o sin aparecer en el producto). A cada uno de los elementos de GG se le asignará una variable de decisión g_j, de manera similar al método de Petrick . Así, se obtendrá un conjunto G con las variables de decisión del problema:

$$G=\{g_1,g_2,...,g_K\} \quad \textit{donde}\ K\leq 3^n \tag{3.27}$$

A falta de un criterio que permita restringir a priori los términos producto que van a aparecer en las soluciones óptimas de f, se van a tomar como elementos de GG los minterms de la función y todos los términos producto que los incluyan; y además los minterms no pertenecientes a la función y todos los términos producto que los incluyan (en realidad se estan escogiendo los 3^n casos posibles, pero se está haciendo una separación entre los términos que provienen de los unos de la función y los que provienen de los ceros, por razones que se verán posteriormente).

Por ejemplo, para la función:

$$f=\sum m(1,2,3)=a\bar{b}+\bar{a}b+ab \tag{3.28}$$

se obtendría lo siguiente:

- Términos correspondientes a los unos de f:

a) Términos producto derivados de ab: $G_0=1$, $G_1=ab$, $G_2=a$, $G_3=\bar{b}$
b) Términos producto derivados de ab: $G_0=1$, $G_4=ab$, $G_2=a$, $G_5=b$
c) Términos producto derivados de $a\bar{b}$: $G_0=1$, $G_6=a\bar{b}$, $G_7=a$, $G_5=b$

- Términos productos correspondientes a los ceros de *f*:

a) Términos producto derivados de $\overline{ab}$: $G_0=1$, $G_8=\overline{ab}$, $G_7=\overline{a}$, $G_3=\overline{b}$

Por tanto, en este caso:

$$GG=\{1,a\bar{b},a,\bar{b},ab,b,\bar{a}b,\bar{a},\bar{a}\,\bar{b}\} \tag{3.29}$$

y así, se tendrán nueve variables de decisión correspondientes a cada uno de los elementos de *GG*:

$$G=\{g_0,g_1,g_2,g_3,g_4,g_5,g_6,g_7,g_8\} \tag{3.30}$$

Una vez que se dispone de los términos producto y de sus variables asociadas, el siguiente paso consiste en construir la función de decisión. Previamente, se van a establecer algunas definiciones y notaciones necesarias.

Definición 3.10: *Se define una ESOP redundante como una ESOP que incluye términos producto que pueden ser eliminados de manera que la ESOP resultante sigue siendo una solución del problema de minimización de f.*

Por ejemplo, si se considera una solución f_1, la expresión:

$$f_2=f_1\oplus ab\oplus a\bar{b}\oplus a \tag{3.31}$$

es una ESOP redundante puesto que:

$$f_2=f_1\oplus a\oplus a=f_1\oplus 0=f_1 \tag{3.32}$$

Resulta inmediato deducir que una ESOP redundante nunca constituirá una solución óptima del problema de minimización de *f*. Si se notan por m_i los unos de la función (o sea los minterms pertenecientes a la misma), por M_i los ceros, por *ON* el conjunto de los m_i, y por *OFF* el conjunto de los M_i; puede construirse la función de decisión $H(g_1,...,g_K)$, conocida por el nombre de función de decisión de Helliwell [PER90], según indica el siguiente Teorema:

Teorema 3.11: *Cada solución ESOP no redundante de la función f es una suma EXOR de términos producto G_j correspondientes a aquellas variables de decisión g_j que aparecen como literales en un producto que satisface la siguiente ecuación lógica:*

$$1=H(g_1,...g_P)=\prod_{m_i\in ON}\Big(\bigoplus_{g_j\in V(m_i)} g_j\Big)\cdot\prod_{M_i\in OFF}\Big(1\oplus\bigoplus_{g_j\in V(M_i)} g_j\Big) \tag{3.33}$$

donde:

$$V(m_i)=\{\textit{variables de los términos producto } G_j \in GG \textit{ tales que } G_j \supseteq m_i\}$$
$$V(M_i)=\{\textit{variables de los términos producto } G_j \in GG \textit{ tales que } G_j \supseteq M_i\} \quad (3.34)$$

cuya demostración puede verse en [PER90].

La expresión (3.33) suele escribirse por comodidad de la siguiente forma:

$$H=\prod_{m_i \in ON} H(m_i)\cdot \prod_{M_i \in OFF} H(M_i)=H_1\cdot H_2 \quad (3.35)$$

Para el ejemplo que se estaba tratando, las funciones H_1 y H_2 se construirían de la siguiente forma:

$$H_1=(g_o \oplus g_1 \oplus g_2 \oplus g_3)\cdot(g_o \oplus g_2 \oplus g_4 \oplus g_5)\cdot(g_o \oplus g_5 \oplus g_6 \oplus g_7) \quad (3.36)$$

$$H_2=(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8) \quad (3.37)$$

y por tanto, la función de decisión H sería:

$$H=H_1\cdot H_2=(g_o \oplus g_1 \oplus g_2 \oplus g_3)\cdot(g_o \oplus g_2 \oplus g_4 \oplus g_5)\cdot(g_o \oplus g_5 \oplus g_6 \oplus g_7)\cdot(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8) \quad (3.38)$$

Una vez que se tiene la función de decisión, las soluciones se obtienen seleccionando (es decir, asignando a valor lógico 1) las variables adecuadas. Por ejemplo, en la función H de (3.38), si se selecciona g_0, se obtendría:

$$H|_{g_0=1}=(1 \oplus g_1 \oplus g_2 \oplus g_3)\cdot(1 \oplus g_2 \oplus g_4 \oplus g_5)\cdot(1 \oplus g_5 \oplus g_6 \oplus g_7)\cdot(1 \oplus 1 \oplus g_3 \oplus g_7 \oplus g_8)=$$
$$=(1 \oplus g_1 \oplus g_3 \oplus g_3)\cdot(1 \oplus g_2 \oplus g_4 \oplus g_5)\cdot(1 \oplus g_5 \oplus g_6 \oplus g_7)\cdot(g_3 \oplus g_7 \oplus g_8) \quad (3.39)$$

Para que se cumpla H es preciso que todos los paréntesis tengan valor 1. Esto puede conseguirse si se anulan todas las variables de los tres primeros paréntesis (ya hay un 1 en cada uno de ellos), y se selecciona una sola de las variables del cuarto. Seleccionando g_8, se tiene:

$$H|_{g_0 g_8=1}=(1 \oplus g_1 \oplus g_2 \oplus g_3)\cdot(1 \oplus g_2 \oplus g_4 \oplus g_5)\cdot(1 \oplus g_5 \oplus g_6 \oplus g_7)\cdot(g_3 \oplus g_7 \oplus 1) \quad (3.40)$$

y finalmente, asignando las demas variables a 0:

$$H|_{g_0 \overline{g}_1 \overline{g}_2 \overline{g}_3 \overline{g}_4 \overline{g}_5 \overline{g}_6 \overline{g}_7 g_8}=(1)\cdot(1)\cdot(1)\cdot(1)=1 \quad (3.41)$$

Por tanto, la expresión ESOP:

$$G_0 \oplus G_8=1 \oplus \overline{a}\,\overline{b} \quad (3.42)$$

es una solución de f. El costo de esta solución sería 2 por tener dos términos producto. No

obstante pueden definirse otras funciones de coste. En general suelen utilizarse las tres siguientes:

a) El número de términos producto incluido el 1.
b) El número de términos producto excluido el 1.
c) El número de literales de la expresión.

Según a), el costo de (3.42) sería 2, según b) 1, y según c) 3.

Una vez obtenida la función de decisión H, el problema de obtener la expresión mínima AND-EXOR queda reducido a encontrar la combinación de variables de menor coste que satisfaga la ecuación H=1. En tal caso, un método exacto de minimización sería el Algoritmo 3.1.

Algoritmo 3.1. Algoritmo exacto de minimización AND-EXOR

a) *Para cada $m_i \in ON$ encontrar el conjunto $GG(m_i)$ de todas sus extensiones (términos producto que incluyen a m_i).*

b) $GG_1 := \bigcup_{m_i \in ON} GG(m_i)$

c) *Para cada $M_i \in OFF$ encontrar el conjunto $GG(M_i)$ de todas sus extensiones.*

d) $GG_2 := \bigcup_{M_i \in OFF} GG(M_i)$

e) *Asignar de manera única variables g_j a los términos de $GG=GG_1 \cup GG_2$. Construir el conjunto G de esas variables.*

f) *Para cada $m_i \in ON$ obtener* $H(m_i) = \bigoplus_{g_j \text{ correspondiente a } G_j \in GG(m_i)} g_j$

g) *Para cada $M_i \in OFF$ obtener* $H(M_i) = \left(1 \oplus \bigoplus_{g_j \text{ correspondiente a } G_j \in GG(M_i)} g_j\right)$

h) *Construir la función H según indica el Teorema 3.2.*

i) *Calcular el conjunto MIN_SOL de todas las soluciones mínimas para H. Los elementos de MIN_SOL serán productos de variables g_j.*

j) *Para cada producto solución $\prod g_j$, obtener la expresión ESOP correspondiente* $\bigoplus G_j$

La forma de efectuar cada uno de los pasos del Algoritmo 3.1 se ha detallado anteriormente a excepción del *i*). En los apartados siguientes se van a describir algunos procedimientos para resolver la ecuación H=1 y cubrir el paso *i*) obteniendo los productos $\prod g_j$ con menor coste. Como se mostrará en 3.2.2.2, el cálculo de *i*) supone la resolución de un problema NP-completo, con lo que no es resoluble en tiempos con complejidad polinomial.

3.2.2.2. Conversión de la función *H* a una GPF equivalente

Uno de los posibles procedimientos para optimizar H consiste en convertir esta función a una GPF equivalente. Una GPF (General Propositional Formula) es una expresión lógica general que describe un problema determinado utilizando los operadores y-lógico y o-lógico [HOP89]. En el problema particular de la optimización de la función *H*, equivaldría a describir cada término producto AND-EXOR mediante los operadores lógicos booleanos AND y OR. Por tanto, quedaría una expresión como suma OR de productos, a la que se le aplicaría el mismo procedimiento que a la función de Petrick para su optimización. Por ejemplo, el término:

$$g_1 \oplus g_2 \oplus g_3 \oplus g_4 \tag{3.43}$$

puede escribirse como una GPF sin más que analizar lógicamente en qué casos vale 1. Resultaría así la siguiente suma de productos:

$$\begin{aligned} &\overline{g}_1\overline{g}_2\overline{g}_3\overline{g}_4 + g_1\overline{g}_2\overline{g}_3\overline{g}_4 + g_1g_2\overline{g}_3\overline{g}_4 + g_1\overline{g}_2g_3\overline{g}_4 + \\ &+\overline{g}_1g_2g_3\overline{g}_4 + \overline{g}_1g_2\overline{g}_3g_4 + \overline{g}_1\overline{g}_2g_3g_4 + g_1g_2g_3g_4 + \overline{g}_1\overline{g}_2\overline{g}_3g_4 \end{aligned} \tag{3.44}$$

Realizando esto mismo con cada uno de los términos que tenga la función a minimizar, se obtendría una GPF para *H*. Si se aplica este método al ejemplo que se estaba tratando en el Apartado 3.2.2.1, se obtendrían las siguientes GPFs para cada uno de los términos:

- Para $t_1=(g_0 \oplus g_1 \oplus g_2 \oplus g_3)$ se tendría:

$$\begin{aligned} GPF(t_1) = (&\overline{g}_0\overline{g}_1\overline{g}_2\overline{g}_3 + g_0\overline{g}_1\overline{g}_2\overline{g}_3 + g_0g_1\overline{g}_2\overline{g}_3 + g_0\overline{g}_1g_2\overline{g}_3 + g_0\overline{g}_1\overline{g}_2g_3 + \\ &\overline{g}_0g_1g_2g_3 + g_0\overline{g}_1g_2g_3 + g_0g_1\overline{g}_2g_3 + g_0g_1g_2\overline{g}_3) \end{aligned} \tag{3.45}$$

- Para $t_2=(g_0 \oplus g_2 \oplus g_4 \oplus g_5)$ se tendría:

$$\begin{aligned} GPF(t_2) = (&\overline{g}_0\overline{g}_2\overline{g}_4\overline{g}_5 + g_0\overline{g}_2\overline{g}_4\overline{g}_5 + g_0g_2\overline{g}_4\overline{g}_5 + g_0\overline{g}_2g_4\overline{g}_5 + g_0\overline{g}_2\overline{g}_4g_5 + \\ &\overline{g}_0g_2g_4g_5 + g_0\overline{g}_2g_4g_5 + g_0g_2\overline{g}_4g_5 + g_0g_2g_4\overline{g}_5) \end{aligned} \tag{3.46}$$

- Para $t_3=(g_0 \oplus g_5 \oplus g_6 \oplus g_7)$ se tendría:

$$\begin{aligned} GPF(t_3) = (&\overline{g}_0\overline{g}_5\overline{g}_7 + g_0\overline{g}_5\overline{g}_6\overline{g}_7 + g_0g_5\overline{g}_6\overline{g}_7 + g_0\overline{g}_5g_6\overline{g}_7 + g_0\overline{g}_5\overline{g}_6g_7 + \\ &\overline{g}_0g_5g_6g_7 + g_0\overline{g}_5g_6g_7 + g_0g_5\overline{g}_6g_7 + g_0g_5g_6\overline{g}_7) \end{aligned} \tag{3.47}$$

- Para $t_4=(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$ se obtendría:

$$GPF(t_4)=(\overline{g_0}\,\overline{g_3}\,\overline{g_7}\,\overline{g_8}+g_0\overline{g_3}\,\overline{g_7}\,\overline{g_8}+g_0\overline{g_3}g_7\overline{g_8}+ \overline{g_0}\,\overline{g_3}g_7g_8+\overline{g_0}g_3\overline{g_7}\overline{g_8}+\overline{g_0}g_3\overline{g_7}g_8+\overline{g_0}g_3g_7\overline{g_8}+g_0g_3g_7g_8) \quad (3.48)$$

con lo que finalmente, la GPF correspondiente a la función H completa vendría dada por:

$$GPF=GPF(t_1)\cdot GPF(t_2)\cdot GPF(t_3)\cdot GPF(t_4) \quad (3.49)$$

Aplicando repetidamente reglas booleanas tales como $a(b+c)=ab+ac$, $\bar{a}\cdot a=0$ ó $a+a=a$; la GPF anterior puede escribirse como una suma de productos AND-OR con literales complementados y sin complementar. Cada uno de estos productos representa una solución de la GPF, y por tanto, de H. Para obtener una solución mínima bastaría, finalmente, con escoger uno de los productos que contenga un menor número de literales.

El problema de decisión para las GPF es, al igual que el de la función de Petrick, NP-completo [GAR79]; y puede concluirse, por tanto, que el problema de optimización de la función H tambien lo es. Como consecuencia, la minimización exacta AND-EXOR utilizando funciones de decisión y GPFs para resolverlas precisan de un elevado tiempo de ejecución y gran cantidad de memoria; no obstante se han propuesto arquitecturas especiales para la resolución de GPFs [HOP90].

3.2.2.3. Manipulación booleana

Otro procedimiento para resolver la función de decisión H consiste en convertirla en una función de decisión restringida, FH, utilizando para ello manipulación booleana. Esta función FH se expresa mediante suma EXOR de productos, y se obtiene mediante la aplicación de las siguientes reglas booleanas:

$$a(bc)=(ab)c \quad (3.50)$$

$$(a\oplus b)c=ac\oplus bc \quad (3.51)$$

$$a(b\oplus c)=ab\oplus ac \quad (3.52)$$

$$a\oplus a=0 \quad (3.53)$$

$$a\oplus 0=a \quad (3.54)$$

$$aa=a \quad (3.55)$$

$$ba=ab \quad (3.56)$$

$$1a=a1=a \quad (3.57)$$

$$b\oplus a=a\oplus b \quad (3.58)$$

La función H se procesa de izquierda a derecha siguiendo el Algoritmo 3.2.

Algoritmo 3.2. Manipulación Boolena

1) Empezando por la izquierda, se aplica la regla (3.50) a los dos primeros términos suma.
2) Se aplican las reglas (3.51) y (3.52) a cada pareja de términos, obteniendo un nuevo término t.
3) Se aplican las reglas (3.55), (3.56) y (3.57).
4) Los términos repetidos se eliminan mediante las reglas de simplicación (3.53) y (3.54).
5) Se ordenan los términos producto utilizando la regla (3.58).
6) La regla (3.50) se aplica al nuevo término t y el siguiente término de H, volviendo al paso 2) hasta que se agoten los términos de H.

La función que se obtiene tras la aplicación de este algoritmo, FH, es una expresión ESOP que puede convertirse en una SOP equivalente de la siguiente forma:

a) Se obtiene una nueva función FH' eliminando de FH aquellos términos producto que se encuentran incluidos en otros productos de FH.
b) Se convierte FH' a una expresión SOP utilizando el siguiente teorema:

Teorema 3.12. *La función SOP de la ecuación (3.60) describe todas las soluciones mínimas de la función de decisión FH:*

$$\sum_{productos \in FH'} P_r = \sum_{productos \in FH'} producto \cdot \Big(\prod_{g_j \in G - \{g_i \,/\, g_i \in producto\}} \overline{g_j} \Big) \tag{3.59}$$

La demostración del teorema puede encontrarse en [PER90].

Como ejemplo de este método, considérese la siguiente función de decisión H:

$$H=(a \oplus b)\cdot(b \oplus c) \tag{3.60}$$

La función de decisión restringida FH vendría dada, aplicando el Algoritmo 3.2, por:

$$FH=a \oplus ac \oplus ab \oplus bc \tag{3.61}$$

Eliminando de (3.61) los términos incluidos en otros, se obtiene la siguiente función FH':

$$FH'=a \oplus bc \tag{3.62}$$

y finalmente, aplicando el Teorema 3.12, se obtiene la expresión SOP:

$$SOP=a\bar{b}\bar{c}+\bar{a}bc \tag{3.63}$$

Para terminar este apartado dedicado a la minimización mediante funciones de decisión, se va a describir un último procedimiento para la resolución de H que se basa en la realización de búsquedas en árbol.

3.2.2.4. Método de búsqueda mediante árbol directo [PER90]

Otro procedimiento para resolver la función H consiste en efectuar una búsqueda en árbol optimizada mediante ramificación y poda. La búsqueda se realiza construyendo todos los subconjuntos posibles de literales positivos, generando una rama para cada uno de los mismos. Muchos de estas ramas no llegan a extenderse debido a las podas que se efectúan, resultando este método el más eficiente de los presentados aquí para optimizar H.

En lo que sigue, se va a notar por $H(k)$ a la función H correspondiente al nodo k. Con esta notación, el proceso de generación de una rama del nodo k a un nodo k+1 será el descrito en el Algoritmo 3.3.

Algoritmo 3.3. Generación de un nodo k+1 a partir de un nodo k

1) Se selecciona un literal g_j y se le asigna el valor 1 en todos los términos de H(k).
2) Si aparecen dos unos en un mismo término, se eliminan ($1 \oplus 1 = 0$).
3) Se crea un nuevo nodo, el nodo k+1, que parte del nodo k y en el que no aparecerá g_j.

El objetivo de la búsqueda es encontrar conjuntos de literales g_j que hagan 1 todos los términos de H. En tal caso, la solución se construirá como la suma de los siguientes dos términos:

- El producto de todos los literales seleccionados en una rama que termina en un nodo con todos los términos de H a 1.
- El producto del resto de las variables pertenecientes a G complementadas.

Queda por describir el algoritmo de ramificación y poda a utilizar, es decir, qué literales se van a seleccionar para efectuar la ramificación en cada nodo y qué criterio se va a seguir para podar las ramas.

Para efectuar las ramificaciones, en primer lugar se va a seleccionar el mejor término del nodo k. Dicho término se define como:

Definición 3.13. *El mejor término del nodo k, que se nombrará por t_{max} se define como el termino de H(k) que maximiza la función de utilidad:*

$$\sum_{\substack{g_k \in t_i \\ t_i \in H(k)}} utilidad(g_i) \tag{3.64}$$

donde la función utilidad(g_j) se define a continuación.

Definición 3.14. *La utilidad de una variable g_j en el nodo k es el número de términos EXOR de la función H(k) que pasan de valor 0 a 1 menos el número de términos que pasan de valor 1 a 0.*

Una vez que se ha seleccionado el mejor término t_{max}, las variables que intervienen en él se ordenan según valores decrecientes de su utilidad, y se generan los nuevos operadores de búsqueda a partir de estas variables. Si el mejor término t_{max} es un término 0, los operadores de búsqueda son las propias variables g_j; mientras que si se trata de un término 1, los operadores son:

- Todos los literales g_j del término.
- El producto de las variables $\bar{g}_j$ correspondientes a esas variables.

En cuanto al procedimiento de poda, se produce la poda de una rama en cuanto lleva un coste acumulado igual al de alguna rama que anteriormente ha finalizado en una solución.

En la Figura 3.2 puede verse la aplicación de este proceso de búsqueda en árbol para el ejemplo que se ha ido tratando.

En el nodo 1 se tiene la función H completa y se selecciona el término subrayado (el término que mayor utilidad presenta). Se ordenan las variables del citado término según su utilidad (g_0, g_2, g_5, g_4) y se procede a efectuar la ramificación. Se comienza por g_0, obteniéndose el nodo 2. Se vuelve a ramificar con respecto a la variable con mayor utilidad del término t_{max} de ese nodo, que resulta ser en este caso g_8, con lo que se obtiene el nodo 3. En este nodo ya se encuentra una solución al problema, puesto que $g_0 g_8 \bar{g}_1 \bar{g}_2 \bar{g}_3 \bar{g}_4 \bar{g}_5 \bar{g}_6 \bar{g}_7$ satisface $H(3)$. La solución obtenida es de costo 2.

De forma análoga se va ramificando con respecto a g_2, generándose el nodo 4. En ese nodo no se encuentra ninguna solución, y se ramifica con respecto a g_6, llegándose a una solución de coste 2 en el nodo 5.

Ramificando con respecto a g_5 se obtiene también una solución de coste 2 en el nodo 7; y finalmente, si se ramifica con respecto a g_4, no se encuentran soluciones en ninguno de los nodos 9, 10 y 11. En estos nodos se produce la poda de las tres ramas puesto que las soluciones que pudieran obtenerse tendrían coste superior a 2.

Este procedimiento mediante búsqueda en árbol genera todas las ESOP mínimas (véase demostracion en [PER90], consiguiendo un gran aumento de la velocidad con respecto a los dos métodos descritos en los apartados 3.2.2.1 y 3.2.2.2. En cualquier caso, todos los procedimientos estudiados para la resolución de la función H precisan de una gran cantidad

nodo1. $(g_0 \oplus g_1 \oplus g_2 \oplus g_3)(g_0 \oplus g_2 \oplus g_4 \oplus g_5)(g_0 \oplus g_5 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

g_0

nodo 2. $(1 \oplus g_1 \oplus g_2 \oplus g_3)(1 \oplus g_2 \oplus g_4 \oplus g_5)(1 \oplus g_5 \oplus g_6 \oplus g_7)(g_3 \oplus g_7 \oplus g_8)$

g_8

nodo 3. $(1 \oplus g_1 \oplus g_2 \oplus g_3)(1 \oplus \bar{g}_2 \oplus \bar{g}_4 \oplus \bar{g}_5)(1 \oplus g_5 \oplus g_6 \oplus g_7)(1 \oplus g_3 \oplus g_7)$

costo=2, SOLUCIÓN=$g_0 g_8 \bar{g}_1 \bar{g}_2 \bar{g}_3 \bar{g}_4 \bar{g}_5 \bar{g}_6 \bar{g}_7$

g_2

nodo 4. $(1 \oplus g_0 \oplus g_1 \oplus g_3)(1 \oplus g_0 \oplus g_4 \oplus g_5)(g_0 \oplus g_5 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

g_6

nodo 5. $(1 \oplus g_0 \oplus g_1 \oplus g_3)(1 \oplus \bar{g}_0 \oplus \bar{g}_4 \oplus \bar{g}_5)(1 \oplus g_0 \oplus g_5 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

costo=2, SOLUCIÓN=$g_2 g_6 \bar{g}_0 \bar{g}_1 \bar{g}_3 \bar{g}_4 \bar{g}_5 \bar{g}_7 \bar{g}_8$

g_5

nodo 6. $(g_0 \oplus g_1 \oplus g_2 \oplus g_3)(1 \oplus g_0 \oplus g_2 \oplus g_4)(1 \oplus g_0 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

g_1

nodo 7. $(1 \oplus g_0 \oplus g_2 \oplus g_3)(1 \oplus \bar{g}_0 \oplus \bar{g}_2 \oplus \bar{g}_4)(1 \oplus g_0 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

costo=2, SOLUCIÓN=$g_1 g_5 \bar{g}_0 \bar{g}_2 \bar{g}_3 \bar{g}_4 \bar{g}_6 \bar{g}_7 \bar{g}_8$

g_4

nodo 8. $(g_0 \oplus g_1 \oplus g_2 \oplus g_3)(1 \oplus g_0 \oplus g_2 \oplus g_5)(g_0 \oplus g_5 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

g_0

nodo 9. $(1 \oplus g_1 \oplus g_2 \oplus g_3)(g_2 \oplus g_5)(1 \oplus g_5 \oplus g_6 \oplus g_7)(g_3 \oplus g_7 \oplus g_8)$

costo=2, poda.

g_2

nodo 10. $(1 \oplus g_0 \oplus g_1 \oplus g_3)(g_0 \oplus g_5)(g_0 \oplus g_5 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

costo=2, poda.

g_5

nodo 11. $(g_0 \oplus g_1 \oplus g_2 \oplus g_3)(g_0 \oplus g_2)(1 \oplus g_0 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

costo=2, poda.

Fig. 3.2. Ejemplo de búsqueda mediante árbol directo

de memoria, y el tiempo de ejecución crece rápidamente al aumentar el número de variables.

En el apartado siguiente se presenta otro procedimiento de minimización exacta AND-EXOR, que también presenta una serie de problemas de aplicación práctica, si bien, puede servir de base para el desarrollo de procedimientos aproximados.

3.2.3. Minimización exacta utilizando reglas algebraicas de reescritura

La primera idea que surge al plantearse el problema de simplificar expresiones módulo-2 de funciones booleanas es tratar de utilizar las propiedades que proporciona el álgebra GF(2) para agrupar términos y llegar a expresiones con menor número de sumas. Por ejemplo, se pueden plantear las siguientes reglas de simplificación propuestas en [EVE67]:

$$x \oplus \bar{x} \rightarrow 1 \tag{3.65}$$

$$x \oplus 1 \rightarrow \bar{x} \tag{3.66}$$

$$xy \oplus \bar{y} \rightarrow 1 \oplus \bar{x}y \tag{3.67}$$

$$xy \oplus \bar{x}\bar{y} \rightarrow \bar{x} \oplus y \tag{3.68}$$

$$xy \oplus \bar{x}\bar{y} \rightarrow x \oplus \bar{y} \tag{3.69}$$

Puesto que la parte derecha de estas reglas contiene siempre un número de términos menor o igual que la parte izquierda, al aplicar estas reglas siempre se obtendrá una expresión más simplificada que la inicial. Por ejemplo, considérese la siguiente función:

$$f(x_1,x_2,x_3) = x_1x_2\bar{x_3} \oplus \bar{x_2}\bar{x_3} \oplus \bar{x_1}x_2x_3 \tag{3.70}$$

Si se aplica (3.67) a los dos primeros términos se obtiene:

$$f(x_1,x_2,x_3) = \bar{x_3} \oplus \bar{x_1}x_2\bar{x_3} \oplus \bar{x_1}x_2x_3 \tag{3.71}$$

y usando ahora (3.65):

$$f(x_1,x_2,x_3) = \bar{x_3} \oplus \bar{x_1}x_2 \tag{3.72}$$

Resulta evidente que este conjunto de reglas puede servir para conseguir una expresión más simple de una función sobre GF(2), pero sin embargo no garantiza el conseguir una expresión mínima. Véase el siguiente ejemplo:

$$f(x_1,x_2,x_3,x_4) = \bar{x_1}x_2\bar{x_3}\bar{x_4} \oplus x_1\bar{x_3} \oplus \bar{x_1}x_4 \oplus x_1\bar{x_2}x_3x_4 \oplus x_3\bar{x_4} \tag{3.73}$$

Si se analiza en detalle esta función puede verse que no es posible aplicar ninguna de las reglas escritas anteriormente; y sin embargo no es una expresión mínima puesto que (3.73) puede escribirse en la forma:

$$f(x_1,x_2,x_3,x_4) = x_1x_2x_3x_4 \oplus \bar{x_1}\bar{x_2}\bar{x_3}\bar{x_4} \oplus 1 \tag{3.74}$$

Así, el conjunto de reglas anterior no garantiza que se obtenga una expresión mínima. Esto lleva de forma natural a plantear una definición de convergencia ([BRA93]) para un conjunto de reglas dado:

Definición 3.15: *Un conjunto de reglas es convergente por definición si y solamente si para cualquier expresión E_0 existe una sucesión de expresiones E_0, E_1, ... , E_n ($n \geq 0$), donde E_{i+1} se obtiene a partir de E_i utilizando alguna de las reglas; y E_n está formada por el número mínimo de productos.*

En principio, según esta definición, el problema de la minimización módulo-2 quedaría reducido a encontrar un conjunto de reglas que fuera convergente (y, como se verá más adelante, esto no es difícil). Sin embargo queda un detalle a resolver, la convergencia garantiza que se puede encontrar el mínimo en un número finito de pasos n, pero eso no implica que el procedimiento tenga que proporcionar ese número n.

A continuación se enuncia un teorema ([BRA93]) acerca de la convergencia de conjuntos de reglas de simplificación:

Teorema 3.16: *Un conjunto de reglas es no convergente si satisface las siguientes dos condiciones:*

a) Cada regla tiene como mucho 2 términos producto en su parte izquierda.

b) Cada regla no tiene mas productos en la parte derecha que en la izquierda.

Según este teorema el conjunto de reglas dado al principio de este apartado es no convergente, como ya se había comprobado. Nótese ademas que para que un conjunto de reglas sea convergente tendrá que contener alguna regla que dé lugar a expresiones con mayor número de términos (algo parecido ocurre cuando se minimiza en el álgebra de Boole; en primer lugar se escribe la función como suma de minterms o producto de maxterms, lo que suele implicar la aparición de un mayor número de términos, y después se aplican los procedimientos de simplificación a la expresión obtenida).

Teniendo en cuenta todas estas consideraciones, se puede intentar construir un conjunto de reglas convergente como el siguiente:

$$x \oplus \bar{x} \rightarrow 1 \tag{3.75}$$

$$1 \oplus 1 \rightarrow 0 \tag{3.76}$$

$$1 \rightarrow x \oplus \bar{x} \tag{3.77}$$

$$0 \rightarrow 1 \oplus 1 \tag{3.78}$$

En [BRA93] se muestra que este conjunto de reglas es capaz de convertir una expresión en cualquier otra (y una de ellas tendrá que ser la expresión mínima), y por tanto se trata de un conjunto convergente. Sin embargo, al proporcionar todas las expresiones posibles resultará muy complicado descubrir cual es la mínima (el número n de la definición de conjunto de reglas convergente puede llegar a ser muy grande). Esto lleva a tratar de conseguir un conjunto de reglas convergente que también llegue a la expresión mínima pero que no genere un número demasiado grande de expresiones. Un conjunto de reglas tal puede ser el siguiente:

$$x \oplus \bar{x} \rightarrow 1 \tag{3.79}$$

$$1 \oplus 1 \rightarrow 0 \tag{3.80}$$

$$1 \rightarrow x \oplus \bar{x} \tag{3.81}$$

$$\bar{x} \rightarrow x \oplus 1 \tag{3.82}$$

La demostración de que este conjunto es convergente puede verse en [BRA91]. Con este conjunto de reglas (o incluso con el que resulta de eliminar (3.79)) se puede llegar a la expresión mínima en un número de pasos, en general, menor que en el caso anterior. Sin embargo, nótese que se sigue teniendo el problema a que se hacía referencia antes; no es posible conocer el número *n* de pasos que hay que dar hasta encontrar la expresión mínima, y así, aunque en teoría un procedimiento basado en este último conjunto de reglas daría el mínimo exacto, en la práctica no se han podido implementar procedimientos que garanticen el alcanzar una solución exacta. No obstante, los mejores métodos aproximados de minimización AND-EXOR; EXMIN2 [SAS93a] (Apartado 3.3.4), EXORCISM-MV-2 [SON96] (Apartado 3.3.5) y el desarrollado en este trabajo, RRMIN2 (Capítulo 4) utilizan conjuntos convergentes de reglas de reescritura como base para efectuar la simplificación de las funciones de conmutación.

3.3. PROCEDIMIENTOS APROXIMADOS DE MINIMIZACIÓN

En la revisión efectuada en los apartados anteriores de procedimientos exactos de minimización, se ha podido comprobar los inconvenientes que presentan dichos algoritmos exactos. Para evitar estos inconvenientes, se ha desarrollado un gran número de procedimientos aproximados, de los cuales se presentan a continuación los más representativos, aunque existen muchos más publicados.

3.3.1. Minimización por vectores extendidos [GRE92]

En el Apartado 2.4.1 se han presentado las familias de formas canónicas de Reed-Muller, y ahora se van a utilizar las definiciones y propiedades que se estudiaron para presentar un procedimiento de minimización aproximado.

3.3.1.1. Formas canónicas

En este apartado se va a ampliar el estudio efectuado en el Apartado 2.4.1 sobre las familias de formas canónicas, generalizando el concepto de forma canónica en la lógica AND-EXOR.

En Álgebra de Boole resulta fácil concluir si una función está expresada en forma canónica o no: basta comprobar que todas las variables están en todos y cada uno de los

productos en el caso de que la función se exprese como suma de minterms (sería totalmente análogo si se tuviera expresada como producto de maxterms). En el caso de funciones definidas sobre GF(2) no resulta tan simple, ya que es posible construir $P=3^n$ productos distintos de n variables (cada variable puede estar ausente, complementada o sin complementar) de los cuales $N=2^n$ son minterms. Este conjunto de 3^n términos posibles puede generarse a partir de un producto de Kronecker de n vectores base de la forma $[1\ \bar{x}_i\ x_i]$ con $1\le i\le n$. Por ejemplo, para $n=2$ se obtendría el vector T representado en (2.74) y que se reproduce a continuación:

$$T=\left[1\ \overline{x_1}\ x_1\ \overline{x_2}\ \overline{x_2}\overline{x_1}\ \overline{x_2}x_1\ x_2\ x_2\overline{x_1}\ x_2x_1\right] \tag{3.83}$$

Cualquier expresión como suma de productos módulo-2 de una función de conmutación debe ser una selección del conjunto de términos así formado, y por tanto existen 2^P expresiones distintas. De estas expresiones, muchas deben de ser equivalentes puesto que sólo existen 2^N funciones distintas de n variables. Por tanto, una función de conmutación que venga dada por su vector verdad M, podrá expresarse de varias formas que tendrán distinto número de términos producto. Asimismo, una forma canónica de una función de n variables debe ser capaz de representar el total de las 2^N funciones de manera única.

En general, sobre GF(2), una forma canónica tendrá una expresión del tipo:

$$f(x_1,x_2,\ldots,x_n)=\bigoplus_{i=0}^{N-1} a_i t_i \tag{3.84}$$

donde t_i son N términos producto distintos seleccionados de T. Los N coeficientes a_i forman el vector función:

$$A=\left[a_0\ a_1\ \ldots\ a_{N-1}\right] \tag{3.85}$$

El vector verdad $M=[m_0\ m_1\ \ldots\ m_{N-1}]$ también tiene N componentes, y puede relacionarse con el vector función de manera única según las expresiones:

$$m_0=f(0,0,\ldots,0,0)\quad m_1=f(0,0,\ldots,0,1)\quad m_2=f(0,0,\ldots,1,0)\quad \ldots\quad m_{N-1}=f(1,1,\ldots,1,1) \tag{3.86}$$

Si se trasladan estas expresiones a forma matricial, se tendrá que M y A se encuentran relacionados mediante una matriz de transformación que se denomina H_n:

$$A=H_nM \tag{3.87}$$
$$M=H_n^{-1}A \tag{3.88}$$

Si la expresión (3.84) representa a una forma canónica, la correspondencia existente entre A y M debe ser una biyección, y por tanto H_n será un matriz de dimensiones NxN invertible.

Cada término t_i del vector T puede asociarse a un vector verdad con N componentes que formará una de las columnas de H_n. Para que H_n tenga inversa, sus columnas deben ser

linealmente independientes, y así, para dar lugar a una forma canónica, deben seleccionarse N términos de T que sean linealmente independientes. Como ejemplo, considérese el caso n=2:

$$t_0=1 \qquad M=[1\ 1\ 1\ 1] \tag{3.89}$$
$$t_1=\overline{x_1}=t_0\oplus t_2 \qquad M=[1\ 0\ 1\ 0] \tag{3.90}$$
$$t_2=x_1 \qquad M=[0\ 1\ 0\ 1] \tag{3.91}$$
$$t_3=\overline{x_2}=t_0\oplus t_6 \qquad M=[1\ 1\ 0\ 0] \tag{3.92}$$
$$t_4=\overline{x_2}\,\overline{x_1}=t_0\oplus t_2\oplus t_6\oplus t_8 \qquad M=[1\ 0\ 0\ 0] \tag{3.93}$$
$$t_5=\overline{x_2}x_1 \qquad M=[0\ 1\ 0\ 0] \tag{3.94}$$
$$t_6=x_2 \qquad M=[0\ 0\ 1\ 1] \tag{3.95}$$
$$t_7=x_2\overline{x_1} \qquad M=[0\ 0\ 1\ 0] \tag{3.96}$$
$$t_8=x_2x_1 \qquad M=[0\ 0\ 0\ 1] \tag{3.97}$$

Los vectores verdad asociados con estos términos suelen colocarse en columnas formando una matriz 4x9 (NxP en el caso general de n variables), y que se denomina G_2 (G_n en general):

$$G_2=\begin{bmatrix}1&1&0&1&1&0&0&0&0\\1&0&1&1&0&1&0&0&0\\1&1&0&0&0&0&1&1&0\\1&0&1&0&0&0&1&0&1\end{bmatrix}=\begin{bmatrix}1&1&0\\1&0&1\end{bmatrix}*\begin{bmatrix}1&1&0\\1&0&1\end{bmatrix}=G_1*G_1 \tag{3.98}$$

En general, para n variables:

$$G_n=G_1*G_{n-1}=G_1*G_1*\ldots*G_1 \qquad n\ veces \tag{3.99}$$

Como puede verse, en ocasiones hay dependencia lineal entre elementos de T. Por ejemplo, en (3.93) se observa que t_4, t_0, t_2, t_6 y t_8 son linealmente dependientes, y en consecuencia no podrán formar parte simultaneamente de la matriz H_2. Cada matriz H_n válida es una matriz NxN que está formada por N columnas de G_n linealmente independientes. Existe, por tanto, un gran número de posibilidades para generar la matriz de transformación H_n. Para n=1 hay 3 elecciones posibles, para n=2 hay 81 elecciones posibles (correspondientes a las 81 formas canónicas representadas en la Tabla 2.2) y para n=3 existen 527121 elecciones posibles [GRE92] (para n>3 no hay estudios hechos por la gran dificultad que ello supone).

Según se estudió en el Apartado 2.4.1, de estas formas canónicas sólo las FPRM y las MPRM muestran una fuerte estructuración, representando sólo una pequeña fracción del total.

A continuación se van a ver unos ejemplos de generación de H_2, mostrándose cómo se producen las distintas formas canónicas.

Supóngase que se seleccionan los términos t_0, t_2, t_6 y t_8. En ese caso, la expresión (3.83) se escribirá:

$$f(x_1,x_2)=a_0t_0\oplus a_1t_2\oplus a_2t_6\oplus a_3t_8=a_0\oplus a_1x_1\oplus a_2x_2\oplus a_3x_2x_1 \tag{3.100}$$

que correspondería al desarrollo RM.

La matriz de transformación correspondiente sería:

$$H_2=\begin{bmatrix}1&0&0&0\\1&1&0&0\\1&0&1&0\\1&1&1&1\end{bmatrix}=\begin{bmatrix}1&0\\1&1\end{bmatrix}*\begin{bmatrix}1&0\\1&1\end{bmatrix}=H_1*H_1 \tag{3.101}$$

Si ahora se toman por ejemplo los términos t_4, t_5, t_7 y t_8, se tiene:

$$f(x_1,x_{29}=a_0\overline{x}_2\overline{x}_1\oplus a_1\overline{x}_2x_1\oplus a_2x_2\overline{x}_1\oplus a_3x_2x_1 \tag{3.102}$$

y la matriz de transformación será:

$$H_2=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}=I \tag{3.103}$$

Por tanto, en este ejemplo $A=M$ y la expresión módulo-2 corresponde a la misma expresión booleana pero enlazada con la operación EXOR en lugar de la OR.

En resumen, cada selección de N productos linealmente independientes puede formar la base de una forma canónica, y así, cualquier expresión de una función particular involucrando a N términos producto que sean linealmente independientes será una forma canónica. Asimismo, cualquier expresión con más de N términos producto debe tener términos linealmente dependientes, y por tanto será susceptible de ser simplificada. Esto hace pensar que probablemente, en una gran cantidad de casos, la expresión mínima pueda encontrarse entre las formas canónicas. Por tanto, puede idearse un procedimiento aproximado de minimización consistente en buscar la expresión mínima de entre todas las formas canónicas.

3.3.1.2. Vectores extendidos

Una expresión módulo-2 de una función de n variables puede representarse por un vector extendido B con P componentes:

$$B=[b_0\ \ b_1\ \ \ldots\ \ b_{P-1}] \tag{3.104}$$

de manera que:

$$f(x_1,x_2,\ldots,x_n)=T\cdot B=[[1\ \ \overline{x_n}\ \ x_n]*[1\ \ \overline{x_{n-1}}\ \ x_{n-1}]*\ldots*[1\ \ \overline{x_1}\ \ x_1]]B \tag{3.105}$$

Por ejemplo, en el caso particular n=2 se tendría:

$$f(x_1,x_2)=b_0\oplus b_1\overline{x_1}\oplus b_2x_1\oplus b_3\overline{x_2}\oplus b_4\overline{x_2}\,\overline{x_1}\oplus b_5\overline{x_2}x_1\oplus b_6x_2\oplus b_7x_2\overline{x_1}\oplus b_8x_2x_1 \tag{3.106}$$

El vector verdad, en ese caso podría calcularse según:

$$m_0=f(0,0)=b_0\oplus b_1\oplus b_3\oplus b_4 \tag{3.107}$$
$$m_1=f(1,0)=b_0\oplus b_2\oplus b_3\oplus b_5 \tag{3.108}$$
$$m_2=f(0,1)=b_0\oplus b_2\oplus b_6\oplus b_7 \tag{3.109}$$
$$m_3=f(1,1)=b_0\oplus b_2\oplus b_6\oplus b_8 \tag{3.110}$$

y si se trasladan estas ecuaciones a forma matricial se obtiene:

$$\begin{bmatrix}m_0\\m_1\\m_2\\m_3\end{bmatrix}=\begin{bmatrix}1&1&0&1&1&0&0&0&0\\1&0&1&1&0&1&0&0&0\\1&1&0&0&0&0&1&1&0\\1&0&1&0&0&0&1&0&1\end{bmatrix}\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\\b_8\end{bmatrix} \tag{3.111}$$

Recordando la expresión (3.98), puede escribirse (3.111) en la forma:

$$M=G_2B \tag{3.112}$$

y en general, para n variables:

$$M=G_nB \tag{3.113}$$

G_1 contiene 4 unos y 2 columnas, por lo que G_n contendrá 4^n unos y 2^n columnas. Como consecuencia de esto, el cálculo directo usando producto de matrices requiere 4^n-2^n sumas. Utilizando un algoritmo rápido, puede demostrarse que es posible realizar el cálculo con sólo $2(3^n$-$2^n)$ sumas ([GRE92]).

La expresión (3.111) puede escribirse también en la forma:

$$\begin{bmatrix}m_0\\m_1\\m_2\\m_3\end{bmatrix}=b_0\begin{bmatrix}1\\1\\1\\1\end{bmatrix}\oplus b_1\begin{bmatrix}1\\0\\1\\0\end{bmatrix}\oplus\ldots\oplus b_8\begin{bmatrix}0\\0\\0\\1\end{bmatrix} \tag{3.114}$$

es decir, M puede calcularse como la suma módulo-2 de las columnas de G_2 que corresponden a las componentes distintas de 0 de B. La expresión (3.114) puede realizar el cálculo

representado en (3.111) utilizando únicamente las columnas que sean necesarias, sin que sea preciso hallar la matriz G_n completa. A continuación se ilustran estas afirmaciones con un ejemplo. Considerese la siguiente función de 3 variables:

$$f(x_1,x_2,x_3)=\overline{x}_2x_1\oplus x_3\overline{x}_2\oplus\overline{x}_3x_2 \tag{3.115}$$

Para hallar las columnas de G_3 correspondientes a cada término producto de esta ecuación, se efectuarán productos de Kronecker con las columnas de G_1, utilizando las siguientes correspondencias:

a) Si x_i no aparece → columna 1 de G_1, es decir, [1 1].
b) Si x_i aparece complementada → columna 2 de G_2, es decir, [1 0].
c) Si x_i aparece sin complementar → columna 3 de G_3, es decir, [0 1].

Aplicando estas correspondencias, las columnas de G_3 correspondientes a cada producto de (3.115) serán:

$$\overline{x}_2x_1=[1\ \ 1]*[1\ \ 0]*[0\ \ 1]=[0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0] \tag{3.116}$$
$$x_3\overline{x}_1=[0\ \ 1]*[1\ \ 1]*[1\ \ 0]=[0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0] \tag{3.117}$$
$$\overline{x}_3x_2=[1\ \ 0]*[0\ \ 1]*[1\ \ 1]=[0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0] \tag{3.118}$$

Efectuando la suma de estos tres vectores, se obtendrá el vector M correspondiente a la función:

$$M=[0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0] \tag{3.119}$$

3.3.1.3. Conjuntos de vectores extendidos

Las ecuaciones (3.107), (3.108), (3.109) y (3.110), para el caso en que M=[0 0 0 0] pueden escribirse:

$$0=b_4\oplus b_0\oplus b_1\oplus b_3 \quad ó \quad b_4=b_0\oplus b_1\oplus b_3 \tag{3.120}$$
$$0=b_5\oplus b_0\oplus b_2\oplus b_3 \quad ó \quad b_5=b_0\oplus b_2\oplus b_3 \tag{3.121}$$
$$0=b_7\oplus b_0\oplus b_1\oplus b_6 \quad ó \quad b_7=b_0\oplus b_1\oplus b_6 \tag{3.122}$$
$$0=b_8\oplus b_0\oplus b_2\oplus b_6 \quad ó \quad b_8=b_0\oplus b_2\oplus b_6 \tag{3.123}$$

Según estas ecuaciones, los valores para b_4, b_5, b_7 y b_8 pueden determinarse para cada combinación posible de b_0, b_1, b_2, b_3 y b_6. Esto implica que existen 2^5=32 elecciones posibles de B que corresponden al vector verdad M=[0 0 0 0]. Esto mismo ocurrirá para cada uno de los otros valores de M, de manera que las 512 formas posibles de B pueden organizarse en 16 conjuntos de 32 vectores cada uno.

A continuación va a introducirse notación para tratar cómodamente estos conjuntos de vectores:

Si i es el equivalente decimal del número binario formado con las componentes de M, en el que m_0 es el bit menos significativo; se notará por C_i al i-ésimo conjunto.

Por otra parte, véase que cuando $b_0=b_1=b_3=b_6=0$, las ecuaciones (3.107) a (3.110) se escriben $b_4=m_0$, $b_5=m_1$, $b_7=m_2$ y $b_8=m_3$. Esta forma particular de vector B se tomará como líder del conjunto.

En general, para n variables, los 2^P vectores extendidos B se dividen en 2^N conjuntos de 2^{P-N} elementos cada uno. En la Figura 3.3 pueden verse estos conjuntos para el caso particular de $n=2$.

Estos conjuntos pueden organizarse en el orden ascendente del equivalente decimal del número binario $p=<b_0b_1b_2b_3b_6>$ en el que b_0 se toma como el bit más significativo. Este número p será el número de fila del conjunto, atendiendo a lo presentado en la Figura 3.3.

Sea ahora $b(i,j)$ el i-ésimo elemento del j-ésimo conjunto. Se cumplen entonces la siguientes propiedades, cuya demostración puede consultarse en [GRE92]:

Propiedad 3.18: *$b(i,j)\oplus b(h,k)=b(i\oplus h,j\oplus k)$, donde las sumas en las coordenadas se realizan bit a bit módulo-2 usando las formas binarias de los número de fila y de conjunto.*

De aquí se sigue:

Propiedad 3.19: *$b(i,j)\oplus b(i,k)=b(0,j\oplus k)$.*

Según esta última propiedad, la suma correspondiente a la misma fila de dos conjuntos diferentes es el líder del conjunto número $j\oplus k$. En particular:

Propiedad 3.20: *$b(i,j)\oplus b(i,0)=b(0,j)$.*

y así la suma de cualquier vector B con su correspondiente número de fila en C_0 dará el líder del conjunto que contiene a B.

Por tanto, el conjunto C_j sumado al conjunto C_k vector por vector producirá el conjunto $C_{j\oplus k}$. Además:

Propiedad 3.21: *$b(i,j)\oplus b(h,j)=b(i\oplus h,0)$.*

y por consiguiente, la suma de dos elementos diferentes del mismo conjunto es un elemento del conjunto C_0. Teniendo en cuenta estas propiedades y que G_n tiene 2^n unos en cada columna, puede afirmarse que cada vector extendido B y el vector que se obtendría complementando todas sus componentes, $\bar{B}$, pertenecen al mismo conjunto.

Todo lo que se ha estudiado en este apartado se utilizará en el apartado siguiente para construir un método de simplificación. Véase que la representación mínima de la suma de productos de un vector verdad, vendrá dada por el vector peso mínimo en el conjunto correspondiente a ese vector verdad. El problema fundamental es encontrar este vector peso

mínimo sin necesidad de realizar una búsqueda exhaustiva a través de los 2^{P-N} elementos del conjunto.

M=[0000]	M=[1000]	M=[0100]	M=[1100]	M=[0010]	M=[1010]	M=[0110]	M=[1110]
000000000	000010000	000001000	000011000	000000010	000010010	000001010	000011010
000000111	000010111	000001111	000011111	000000101	000010101	000001101	000011101
000111000	000101000	000110000	000100000	000111010	000101010	000110010	000100010
000111111	000101111	000110111	000100111	000111101	000101101	001101101	001001101
001001001	001011001	001000001	001010001	001001011	001011011	001000011	001010011
001001110	001011110	001000110	001010110	001001100	001011100	001000100	001010100
001110001	001100001	001111001	001101001	001110011	001100011	001111011	001101011
001110110	001100110	001111110	001101110	001110100	001100100	001111100	001101100
010010010	010000010	010011010	010001010	010010000	010000000	010011000	010001000
010010101	010000101	010011101	010001101	010010111	010000111	010011111	010001000
010010010	010111010	010100010	010110010	010101000	010111000	010100000	010110000
010101101	010111101	010100101	010110101	010101111	010111111	010100111	010110111
011011011	011001011	011010011	011000011	011011001	011001001	011010001	011000001
011011100	011001100	011010100	011000100	011011110	011001110	011010110	011000110
011100011	011110011	011101011	011111011	011100001	011110001	011101001	011111001
011100100	011110100	011101100	011111100	011100110	011110110	011101110	011111110
100011011	100001011	100010011	100000011	100011001	100001001	100010001	100000001
100011100	100001100	100010100	100000100	100011110	100011110	100010110	100000110
100100011	100110011	100101011	100111011	100100001	100110001	100101001	100111001
100100100	100110100	100101100	100111100	100100110	100110110	100101110	100111110
101010010	101000010	101011010	101001010	101010000	101000000	101011000	101001000
101010101	101000101	101011101	101001101	101010111	101000111	101011111	101001111
101101010	101111010	101100010	101110010	101101000	101111000	101100000	101110000
101101101	101111101	101100101	101110101	101101111	101111111	101100111	101110111
110001001	110011001	110000001	110010001	110001011	110011011	110000011	110010011
110001110	110011110	110000110	110010110	110001100	110011100	110000100	110010100
110110001	110100001	110111001	110101001	110110011	110100011	110111011	110101011
110110110	110100110	110111110	110101110	110110100	110100100	110111100	110101100
111000000	111010000	111001000	111011000	111000010	111010010	111001010	111011010
111000111	111010111	111001111	111011111	111000101	111010101	111001101	111011101
111111000	111101000	111110000	111100000	111111010	111101010	111110010	111100010
111111111	111101111	111110111	111100111	111111101	111101101	111110101	111100101

M=[0001]	M=[1001]	M=[0101]	M=[1101]	M=[0011]	M=[1011]	M=[0111]	M=[1111]
000000001	000010001	000001001	000011001	000000011	000010011	000001011	000011011
000000110	000010110	000001110	000011110	000000100	000010100	000001100	000011100
000111001	000101001	000110001	000100001	000111011	000101011	000110011	000100011
000111110	000101110	000110110	000100110	000111100	000101100	000110100	000100100
001001000	001011000	001000000	001010000	001001010	001011010	001000010	001010010
001001111	001011111	001000111	001010111	001001101	001011101	001000101	001010101
001100000	001100000	001111000	001101000	001110010	001100010	001111010	001101010
001110111	001100111	001111111	001101111	001110101	001100101	001111101	001101101
010010011	010000011	010011011	010001011	010010001	010000001	010011001	010001001
010010100	010000100	010011100	010001100	010010110	010000110	010011110	010001110
010101011	010111011	010100011	010110011	010101001	010111001	010100001	010110001
010101100	010111100	010100100	010110100	010101110	010111110	010100110	010110110
011011010	011001010	011010010	011000010	011011000	011001000	011010000	011000000
011011101	011001101	011010101	011000101	011011111	011001111	011010111	011000111
011100010	011110010	011101010	011111010	011100000	011110000	011101000	011111000
011100101	011110101	011101101	011111101	011100111	011110111	011101111	011111111
100011010	100001010	100010010	100000010	100011000	100001000	100010000	100000000
100011101	100001101	100010101	100000101	100011111	100001111	100010111	100000111
100100010	100110010	100101010	100111010	100100000	100110000	100101000	100111000
100100101	100110101	100101101	100111101	100100111	100110111	100101111	100111111
101010011	101000011	101011011	101001011	101010001	101000001	101011001	101001001
101010100	101000100	101011100	101001100	101010110	101000110	101011110	101001110
101101011	101111011	101100011	101110011	101101001	101111001	101100001	101110001
101101100	101111100	101100100	101110100	101101110	101111110	101100110	101110110
110001000	110011000	110000000	110010000	110001010	110011010	110000010	110010010
110001111	110011111	110000111	110010111	110001101	110011101	110000101	110010101
110110000	110100000	110111000	110101000	110110010	110100010	110111010	110101010
110110111	110100111	110111111	110101111	110110101	110100101	110111101	110101101
111000001	111010001	111001001	111011001	111000011	111010011	111001011	111011011
111000110	111010110	111001110	111011110	111000100	111010100	111001100	111011100
111111001	111101001	111110001	111100001	111111011	111101011	111110011	111100011
111111110	111101110	111110110	111100110	111111100	111101100	111110100	111100100

Fig. 3.3. Conjuntos de vectores extendidos para 2 variables

mínimo sin necesidad de realizar una búsqueda exhaustiva a través de los 2^{P-N} elementos del conjunto.

Aunque se ha visto que existe una buena estructuración, es difícil establecer una regla acerca de cómo puede aislarse el vector peso explícitamente en un caso general.

3.3.1.4. Técnicas de simplificación

En este apartado se van a dar reglas orientativas para tratar de encontrar la forma canónica mínima utilizando para ello los conjuntos de vectores extendidos que se definieron en 3.3.1.3.

Cuando un vector verdad coincide con un columna de G_n, quiere decir que habrá un elemento del conjunto correspondiente a ese M con peso 1. En otro caso, habrá que hallar el número mínimo de columnas de G_n que se precisa sumar para obtener M. Equivalentemente, se requiere encontrar el vector peso mínimo en el conjunto de vectores extendidos correspondientes a M.

Cada elemento B de un conjunto puede dividirse en dos componentes, B_r y B_s. B_r contiene las P-N componentes de B que se utilizan para construir el número de fila del conjunto, y B_s contiene las N componentes restantes. Por ejemplo, para el caso n=2 se tendrá:

$$B_r=[b_0 \;\; b_1 \;\; b_2 \;\; b_3 \;\; b_6] \tag{3.124}$$

$$B_s=[b_4 \;\; b_5 \;\; b_7 \;\; b_8] \tag{3.125}$$

En cada conjunto B_r se tienen las 2^{P-N} combinaciones posibles, y por tanto su peso w_r estará en el rango $0 \le w_r \le P-N$. El vector B_s correspondiente a cada B_r puede obtenerse mediante una generalización de las ecuaciones (3.107) a (3.110), y su peso w_s estará dentro del rango $0 \le w_s \le N$. El peso de B será w_r+w_s.

En la primera columna de cada conjunto b_r=0, por tanto w_r=0 y así B_s=M, con lo que w_s es igual al peso de M. Si en B_r b_0=1 y todos los otros b_i=0 para $i \neq 0$, entonces w_r=1 y en esa fila del conjuto $B_s=\bar{M}$, siendo por tanto w_s igual al peso de $\bar{M}$. Puesto que si el peso de M es mayor que 2^{n-1}, el peso de $\bar{M}$ es menor que 2^{n-1}; existirá algún elemento en su conjunto con peso al menos $N/2=2^{n-1}$; y este elemento dará una cota superior del peso de la forma minimizada de cualquier expresión como suma de productos módulo-2 de una función de n variables. Así, es posible restringir la búsqueda limitándose sólo a los miembros de un conjunto que tengan $w_r<2^{n-1}$. Para n=2 esto significa tener que inspeccionar 5 vectores en lugar de 32. Para n=3 habrá que investigar 1159 en lugar de 2^{19}, etc. Esto podría hacerse sistemáticamente seleccionando B_r para dar w_r=0, 1, 2, 2^{n-1}-1 y deduciendo en cada caso el correspondiente B_s y w_s. Sin embargo, véase que este proceso resulta ser largo y complicado, y por tanto de poca aplicación a la hora de implementarlo en computadores. No obstante, este tipo de representación puede ser útil si se encuentra algún procedimiento heurístico capaz de encontrar de forma rápida (aunque con un grado de aproximación menor) una expresión suficientemente simple de la función que se desea minimizar.

3.3.2. Método tabular ([GRE92], [TRA93])

El procedimiento que se presenta a continuación utiliza un método tabular similar al que se aplica en el algoritmo de Quine-McCluskey para funciones booleanas. Por tanto, el método actúa directamente sobre la lista de términos producto dando lugar a otra expresión como suma de productos, pero con un número de términos menor.

Los 3^n términos producto distintos que pueden construirse para n variables pueden indentificarse convenientemente utilizando un código ternario. Si t_i, $0 \le i \le 3^n-1$ es el i-ésimo término producto del vector T generado por el producto de Kronecker de n vectores base de la forma $[1\ \bar{x}_j\ x_j]$, $1 \le j \le n$; el equivalente ternario de i será el que identifique al término. Esto puede comprobarse que es equivalente a pesar cada variable como 0 si no está en el término, como 1 si está presente en forma complementada y como 2 si está presente en forma no complementada.

Una vez realizada esta identificación, dos términos producto cualesquiera que tengan códigos ternarios que difieran sólo en una posición podrán reunirse en un sólo término. Este nuevo término diferirá de los dos anteriores también en esa misma posición y tomará el tercer valor posible. Si se analiza en detalle, es fácil ver que este proceso equivale a aplicar las siguientes reglas de simplificación:

$$1 \oplus x_i \rightarrow \overline{x_i} \tag{3.126}$$

$$1 \oplus \overline{x_i} \rightarrow x_i \tag{3.127}$$

$$x_i \oplus \overline{x_i} \rightarrow 1 \tag{3.128}$$

Estas reglas de simplificación, según el Teorema 3.16 forman un conjunto no convergente, y por tanto este procedimiento de simplificación únicamente será un procedimiento aproximado; no puede garantizar el alcanzar el mínimo.

A continuación se presenta un ejemplo sencillo para ilustrar cómo trabaja el procedimiento.

Considerese n=3 y la pareja de términos:

$$\overline{x_3}\,\overline{x_2}\,\overline{x_1} \oplus x_3\,\overline{x_2}\,\overline{x_1} \equiv 111 \oplus 211 = 011 \equiv \overline{x_2}\,\overline{x_1} \tag{3.129}$$

que escrito en términos de componentes de T sería:

$$t_{13} \oplus t_{22} = t_4 \tag{3.130}$$

Si además de la pareja de términos de (3.129), se tiene la siguiente:

$$\overline{x_3}\,x_2\,\overline{x_1} \oplus x_3\,x_2\,\overline{x_1} \equiv 121 \oplus 221 = 021 \equiv x_2\,\overline{x_1} \tag{3.131}$$

que en términos de T resultaría:

$$t_{16} \oplus t_{25} = t_7 \tag{3.132}$$

pueden reunirse los términos obtenidos en (3.129) y en (3.131) nuevamente, obteniéndose:

$$\overline{x_2}\,\overline{x_1} \oplus x_2\overline{x_1} \equiv 011 \oplus 021 = 001 \equiv \overline{x_1} \tag{3.133}$$

que también puede escribirse:

$$t_4 \oplus t_7 = t_1 \tag{3.134}$$

Por tanto, en este ejemplo se han combinado cuatro términos producto para dar lugar a uno sólo.

Estos términos que difieren tan sólo en una posición de su identificación ternaria se van a denominar términos adyacentes, en analogía a la situación que se produce en el álgebra de Boole. Cada término producto t_i es adyacente a otros 2^n. En cada caso, el subíndice del término adyacente diferirá del primero en 3^k (incluyendo $3^0=1$) o en $2\cdot 3^k$ (incluyendo $2\cdot 3^0=2$).

A partir de estas consideraciones es posible organizar el procedimiento en un método tabular que opera directamente sobre un conjunto de términos producto para descubrir si existen términos adyacentes.

Se va a ilustrar tal procedimiento usando un ejemplo. Considérese la siguiente función:

$$f(x_1,x_2,x_3) = \overline{x_2} \oplus x_2x_1 \oplus \overline{x_3}x_1 \oplus \overline{x_3}\,\overline{x_2} \oplus \overline{x_3}\,\overline{x_2}\,\overline{x_1} \oplus \overline{x_3}x_2\overline{x_1} \oplus x_3\overline{x_2}\,\overline{x_1} \oplus x_3x_2\overline{x_1} \oplus x_3x_2x_1 \tag{3.135}$$

De forma similar al procedimiento de Quine-McCluskey, se forma una tabla en la que en la primera columna se encuentra listado el índice del término producto en ternario y en decimal. Cada miembro de esta lista se compara entonces con cada uno de los otros miembros para determinar si sus códigos ternarios difieren sólo en una posición. Si es el caso, estos dos números y su diferencia son escritos como entrada en la segunda columna y las entradas originales en la primera columna se marcan. Cuando este proceso se completa, se habrán detectado todas las posibles parejas de términos producto. Cualquier término producto que no pueda unirse con otro será identificado automáticamente al encontrarse sin marcar.

El proceso se repite en la segunda columna y ahora para unir dos términos se requiere la misma diferencia original y una nueva diferencia que tendrá que ser de 3^k ó $2\cdot 3^k$. Cualquier comparación con éxito se introduce en la tercera columna, con la pareja de diferencias correspondiente. El proceso vuelve a repetirse hasta que ya no puedan crearse nuevas columnas. La Tabla 3.1 es la correspondiente a la función (3.135). En ella puede verse que los grupos que quedan sin marcar son:

$$(11),\ (3,12),\ (8,26),\ (12,13),\ (25,26),\ (13,16,22,25) \tag{3.136}$$

	Columna 1		Columna 2		Columna 3
010	3	✓	3,12 (9)		13,16,22,25 (3,9)
022	8	✓	8,26 (18)		
102	11	✓	12,13 (1)		
110	12	✓	13,16 (3)	✓	
111	13	✓	13,22 (9)	✓	
121	16	✓	16,25 (9)	✓	
211	22	✓	12,25 (3)	✓	
221	25	✓	22,25 (3)	✓	
222	26	✓	25,26 (1)		

Tabla 3.1. Ejemplo de minimización mediante el método tabular

Término → Grupo ↓	t_3	t_8	t_{11}	t_{12}	t_{13}	t_{16}	t_{22}	t_{25}	t_{26}
(11)			X						
(3,12)	X			X					
(8,26)		X							X
(12,13)				X	X				
(25,26)								X	X
(13,16,22,25)					X	X	X	X	

Tabla 3.2. Tabla de indicendia para el ejemplo de minimización tabular

Ahora es necesario seleccionar el menor número de estos grupos que cubran cada uno de los términos producto originales un número impar de veces (esta es una particularidad de la síntesis AND-EXOR que contribuye a complicar estos procedimientos tabulares). Al igual que se hace en el álgebra de Boole, se suele utilizar una tabla de incidencia para hacer esta selección. En este tipo de tablas, las filas corresponden a los grupos no marcados y las columnas a los términos producto que hay que cubrir. La coincidencia entre un grupo no marcado y un término producto se señala con una cruz. En la Tabla 3.2 puede verse la tabla de incidencia correspondiente al ejemplo de la función (3.135). Una columna que contenga una única cruz significará que el término producto correspondiente sólo puede ser cubierto por un único grupo sin marcar, grupo que se suele denominar esencial. En este caso, la función puede sintetizarse utilizando únicamente los grupos esenciales, siendo muy sencillo encontrar la solución óptima. Los grupos elegidos serían, por tanto, los siguientes:

$$\begin{array}{c} f(x_1,x_2,x_3) \quad (11) \quad (3,12) \quad (8,26) \quad (13,16,22,25) \\ 102 \quad (010 \quad 110) \quad (022 \quad 222) \quad (111 \quad 121 \quad 211 \quad 221 \end{array} \tag{3.137}$$

La expresión simplificada, escrita ya en función de términos producto explícitos, sería:

$$f(x_1,x_2,x_3)=\overline{x_3}x_1 \oplus x_3\overline{x_2} \oplus \overline{x_3}x_2x_1 \oplus \overline{x_1}=t_{11} \oplus t_{21} \oplus t_{17} \oplus t_1 \tag{3.138}$$

Sin embargo, el proceso de simplificación no acaba aquí, puede volver a aplicarse todo el proceso a la nueva función obtenida, y en ocasiones llegarse a una expresión aún más simple. En el ejemplo que se está tratando, si se aplica nuevamente el método tabular sobre la expresión (3.138), se obtiene:

$$f(x_1,x_2,x_3)=x_3\overline{x_2} \oplus \overline{x_3}\,\overline{x_2}x_1 \oplus \overline{x_1} \tag{3.139}$$

obteniéndose un término producto menos. Si se vuelve a aplicar el procedimiento tabular a la expresión (3.139) se obtiene esa misma expresión, habiéndose llegado por tanto a la máxima simplificación que permite el método.

3.3.3. Mimización utilizando CRMPs

En el Apartado 2.4.1.5.4 se definieron las formas CRMP y se enunciaron algunas propiedades que presentan. Una posibilidad para minimizar una función en lógica AND-EXOR de forma aproximada es encontrar el mínimo dentro de una familia de formas canónicas. El que ese mínimo esté próximo al absoluto dependerá fundamentalmente de lo amplia que sea la familia de formas canónicas. En el caso de las CRMPs, según se vió en la Ecuación (2.87), se trata de una familia muy amplia, y por tanto se obtendrá una

aproximación bastante buena.

En primer lugar se van a dar una serie de definiciones que son precisas para la descripción del algoritmo de simplificación.

Definición 3.21: *Se define la diferencia booleana f_{x_i} de una función f con respecto a la variable x_i como:*

$$f_{x_i}=f(x_1,x_2,...,x_i,...x_n)\oplus f(x_1,x_2,...,\bar{x_i},...,x_n) \qquad (3.140)$$

Definición 3.22: *Se define la diferencia booleana de una función f con respecto a un término producto $t=x_{i^c}x_{j^c}...x_{k^c}$ como:*

$$f_t=(...(f_{x_i^c})_{x_j^c}...)_{x_k^c} \qquad (3.141)$$

Definición 3.23: *Sea t un término producto. Se define el conjunto de literales del término t, S(t) como:*

$$S(t)=\{x_i/x_i^c \text{ aparece en } t\} \qquad (3.142)$$

Por ejemplo, $S(x_1\bar{x}_2x_3)=\{x_1,x_2,x_3\}$.

Definición 3.24: *Un término producto se denomina primo con respecto a la función f sí y solamente sí $f_t\equiv 1$. Análogamente, un término producto t se dice que está ausente con respecto a la función f si y solamente si $f_t=0$.*

Definición 3.25: *Una función de conmutación $f(x_1,x_2,...,x_n)$ se denomina impar si y solamente si $f_{x_1...x_n}\equiv 1$. Una función $f(x_1,x_2,...,x_n)$ es par si y solamente si $f_{x_1...x_n}\equiv 0$.*

De estas definiciones es fácil demostrar que se cumplen las siguientes propiedades:

$$f_{x_i}=f_{\bar{x_i}} \quad f_{x_ix_j}=f_{x_jx_i} \quad (x_ix_j)_{x_i}=x_j \qquad (3.143)$$

A continuación se enuncian teoremas que son precisos para el procedimiento de minimización, y cuya demostración puede encontrarse en [CSA93].

Teorema 3.26: *Un término t es primo con respecto a la función f si y solamente si en cualquier forma CRMP de f existe exactamente un término producto t' tal que S(t)=S(t') y no existe un término t'' tal que $S(t)\subset S(t'')$.*

Corolario 3.27: *Las formas CRMP (y FPRM) mínimas de f tendrá un término t' tal que $S(t)\subset S(t')$ y no hay ningún término t'' tal que $S(t)\subset S(t'')$.*

Corolario 3.28: *Para todos los términos existentes $\underline{t}$ de una CRMP de f, existe un término primo t de f tal que $S(\underline{t})\subseteq S(t)$.*

Corolario 3.29: *Toda forma CRMP tiene al menos un término primo.*

Teorema 3.30: *Un término producto t es un término ausente con respecto a la función f si y solamente si en ninguna forma CRMP de f hay un término de f, t' tal que $S(t)\subseteq S(t')$.*

Corolario 3.31: *Toda función par f en forma CRMP tiene al menos un término ausente.*

Corolario 3.32: *Si una función f es par, entonces tiene términos ausentes en alguna de sus formas CRMP.*

Por ejemplo, $f_1=x_1x_2x_3$ y $f_2=x_1x_2x_3\oplus x_1x_2$ son funciones pares, $f_3=x_1x_2\oplus x_2x_3\oplus x_1x_3$ es una función impar y $t=x_1x_2x_3$ es un término ausente de esta función.

Para ilustrar el concepto fundamental de términos primo y ausente (esenciales para los algoritmos de minimización que se darán después), se dan los siguientes ejemplos, en los que los términos primos se encuentran subrayados y se enuncia una lista de los términos ausentes.

Ejemplo 3.33: $1\oplus x_1\oplus x_2\oplus \underline{x_1x_2x_3}$.
Términos ausentes: ninguno.

Ejemplo 3.34: $1\oplus x_1\oplus \underline{x_1x_2}\oplus \underline{x_1x_3}\oplus \underline{x_2x_3}$.
Términos ausentes: $x_1x_2x_3$.

Ejemplo 3.35: $1\oplus \underline{x_1}\oplus x_2\oplus \underline{x_2x_3}$.
Términos ausentes: x_1x_2, x_1x_3, $x_1x_2x_3$.

Ejemplo 3.36: $x_1\oplus x_2\oplus \underline{x_3}\oplus x_4\oplus \underline{x_1x_2x_4}$.
Términos ausentes: x_1x_3, x_2x_3, x_3x_4, $x_1x_2x_3$, $x_1x_3x_4$, $x_2x_3x_4$, $x_1x_2x_3x_4$.

Ejemplo 3.37: $x_1\oplus x_2x_3\oplus \underline{x_1x_2x_3x_4}$.
Términos ausentes: ninguno.

Ejemplo 3.38: $\underline{x_1}\oplus \underline{x_2x_3}\oplus \underline{x_2x_4}\oplus \underline{x_3x_4}$.
Términos ausentes: x_1x_2, x_1x_3, x_1x_4, $x_1x_2x_3$, $x_1x_2x_4$, $x_1x_3x_4$, $x_2x_3x_4$, $x_1x_2x_3x_4$.

Seguidamente se enuncian dos algoritmos para la minimización de formas CRMP. Como de costumbre, antes de aplicar una algoritmo de simplificación AND-EXOR, hay que conseguir una expresión ESOP de la función de conmutación a minimizar. En este caso, se parte de una forma canónica FPRM cualquiera (puede partirse por ejemplo de la forma RM,

para la que en 2.3.2 se dio un algoritmo rápido para su cálculo).

El Algoritmo 3.3 obtiene la expresión CRMP mínima de forma exacta, pero resulta bastante complicado y requiere de un tiempo de ejecución grande. Por esta razón, en su lugar, se suele utilizar un procedimiento que proporciona una aproximación al mínimo CRMP, y que se presenta en el Algoritmo 3.4.

Algoritmo 3.3. Cálculo del mínimo exacto CRMP a partir de una forma FPRM.

1) *Encontrar el conjunto PT de todos los términos primos de la forma FPRM de f.*
2) *Encontrar el conjunto PRT de todos los términos producto correspondientes a los términos primos de PT. Esto se hace para conseguir todas las polaridades de todas las variables en los términos de PT (Para cada término primo t se crean varios términos t' tales que S(t)=S(t')).*
3) *Encontrar el conjunto PTE1 de términos producto que pueden crearse a partir de los términos producto de PRT quitando todos los subconjuntos de literales que sea posible.*
4) *Utilizando un árbol de búsqueda, seleccionar el grupo de subconjuntos de PRT1 tales que la suma módulo-2 de todos esos grupos sea igual a f. Cuando en alguna rama se seleccione un término producto para la solución, eliminar todos los términos producto t' tales que S(t)=S(t') en todas las ramas del arbol que parten de t.*

Algoritmo 3.4. Cálculo del mínimo CRMP aproximado a partir de una forma canónica FPRM.

1) $i:=1$, $f_1=f$;
2) *Descomponer* $f_i=f_{ip}\oplus f_{i+1}$ *donde* f_{ip} *es la suma módulo-2 de los términos primos con respecto a* f_i *tales que:*
   a) Los términos t y t' de f_{ip} *implican que* $S(t')\subset S(t)$.
   b) $(f_i)_t\equiv 1$ *implica* $(f_{ip})_t\equiv 1$.
   c) El conjunto $\{S(t)/(f_{i+1})\equiv 1\}$ *tiene la cardinalidad mínima.*
   $i:=i+1$.
3) *Si* $f_{i+1}\neq 0$, *repetir Paso 2.*
4) *Escribir la solución:* $f=f_{1p}\oplus f_{2p}\oplus\ldots\oplus f_{ip}$.

Este algoritmo se utiliza en la práctica y se encuentra implementado en ordenador con el nombre de Cannes (CANonical Nor Exor Synthesiser) [CSA93].

3.3.4. EXMIN2

En el Apartado 3.2.3 se estudiaron los procedimientos basados en reglas de reescritura, y a continuación se va a estudiar uno de los mejores procedimientos basados en conjuntos

de reglas. Constituye una versión más avanzada del procedimiento EXMIN presentado en [BRA91].

El conjunto de reglas en que se apoya EXMIN2 es el siguiente:

$$\begin{cases} x \oplus x = 0 \\ x \oplus \bar{x} = 1 \\ x \oplus 1 = \bar{x} \\ \bar{x} \oplus 1 = x \end{cases} \tag{3.144}$$

$$xy \oplus \bar{y} = \bar{x}\bar{y} \oplus x \tag{3.145}$$

$$x \oplus y = \bar{x} \oplus \bar{y} \tag{3.146}$$

$$x\bar{y} \oplus \bar{x}y = x \oplus y \tag{3.147}$$

$$xy \oplus \bar{y} = 1 \oplus \bar{x}y \tag{3.148}$$

$$x \oplus y = x\bar{y} \oplus \bar{x}y \tag{3.149}$$

$$1 \oplus \bar{x}y = xy \oplus \bar{y} \tag{3.150}$$

$$1 = x \oplus \bar{x} \tag{3.151}$$

Este conjunto de reglas incluye claramente a las reglas (3.75), (3.76), (3.77) y (3.78), por lo que se trata de un conjunto de reglas convergente. Según lo estudiado en 3.2.3, este conjunto de reglas es capaz de proporcionar el mínimo exacto, pero no es capaz de decidir el momento en que se alcanza el mínimo. Por tanto, es preciso enunciar un algoritmo heurístico que decida las reglas que hay que aplicar, el orden en que deben ser aplicadas y cuándo se debe de dar por finalizado el proceso de minimización.

A la hora de implementar una función lógica se suele partir de la tabla verdad de la misma, con lo que la primera expresión de que se dispone es AND-OR (SOP), siendo necesario conseguir una expresión AND-EXOR (ESOP) de la función antes de aplicar el procedimiento EXMIN2. Una forma rápida de hacerlo es partir de una expresión DSOP (una expresión SOP en la que todos los términos son mutuamente disjuntos, es decir, no hay dos términos producto que cubran simultaneamente un mismo 1), y aplicar el siguiente teorema:

Teorema 3.39: *En una expresión DSOP, los operadores OR pueden sustituirse por los operadores EXOR sin que la función cambie.*

A partir de una expresión DSOP, y aplicando este teorema, se obtiene directamente una expresión ESOP de la función booleana. En [SAS93c] puede encontrarse una demostración del teorema. A continuación, partiendo de la expresión ESOP obtenida, debe aplicarse el Algoritmo 3.5 para para realizar la simplificación.

Algoritmo 3.5. EXMIN2

1) *Aplicar a cada pareja de productos que sea posible, alguna de las reglas (3.144).*
2) *Aplicar a cada pareja de productos las reglas (3.145), (3.146), (3.148) y (3.147), siempre que sea posible. En los productos modificados por estas reglas, aplicar nuevamente alguna de las reglas de (3.144).*
3) *Si (3.147) ó (3.148) se han utilizado en el Paso 2, repetir el citado paso, dando preferencia a la aplicación de (3.148) y (3.147).*
4) *Para cada pareja de productos, si es posible, aplicar nuevamente (3.144).*
5) *Aplicar (3.149) y (3.150).*
6) *Repetir nuevamente los pasos 1, 2, 3 y 4.*
7) *Si al realizar el paso 6 se ha producido alguna reducción de términos producto, ir al paso 2.*
8) *En este paso no se produce una reducción del número de términos. En cambio, se incrementará el número de productos utilizando (3.151), para después tratar de conseguir nuevas reducciones. Para cada variable x_i, se desarrollará la función f que se esté minimizando en la forma indicada en (3.5). A continuación, debe elegirse una variable x_i que incremente el número mínimo de productos en la expresión desarrollada según (3.5). Seguidamente, se simplificará cada una de las dos subfunciones independientemente utilizando los pasos 1 al 7. Después, se vuelve a simplificar la función total. Este paso debe repetirse mientras sea posible conseguir una reducción en el número de términos producto.*

El algoritmo EXMIN2, tal y como está planteado, pretende realizar en primer lugar una minimización en el número de términos producto, y en una segunda fase una minimización en el número de literales; obteniéndose así muy buenos resultados.

Para ilustrar este procedimiento, a continuación se presenta un ejemplo. Considérese la siguiente función de cuatro variables:

$$f(x_1,x_2,x_3,x_4)=\overline{x}_4\overline{x}_3\overline{x}_1+\overline{x}_4\overline{x}_3x_2+x_4\overline{x}_3x_1+x_4x_3x_2\overline{x}_1+\overline{x}_4x_3\overline{x}_2 \tag{3.152}$$

En primer lugar, debe escribirse una forma de esta función que sea una expresión DSOP. Para ello, lo más cómodo es partir del mapa de Karnaugh (Para un número de variables $n>6$ habrá que utilizar otro procedimiento) de la función, representado en la Tabla 3.3, y ver si existe algún 1 que sea cubierto por dos términos. Como puede verse en la citada tabla, existen dos productos que no son disjuntos, y es preciso separarlos. En la Tabla 3.4 puede verse el mapa de Karnaugh correspondiente a la función (3.152) escrita en forma DSOP.

La expresión correspondiente a la forma DSOP de (3.152) será por tanto:

$$f(x_1,x_2,x_3,x_4)=\overline{x}_4\overline{x}_3\overline{x}_1+\overline{x}_4\overline{x}_3x_2x_1+x_4\overline{x}_3x_1+x_4x_3x_2\overline{x}_1+\overline{x}_4x_3\overline{x}_2 \tag{3.153}$$

En virtud del Teorema 3.39, una expresión ESOP de (3.152) será:

$$f(x_1,x_2,x_3,x_4)=\overline{x}_4\overline{x}_3\overline{x}_1\oplus\overline{x}_4\overline{x}_3x_2x_1\oplus x_4\overline{x}_3x_1\oplus x_4x_3x_2\overline{x}_1\oplus\overline{x}_4x_3\overline{x}_2 \tag{3.154}$$

Sobre esta expresión ya puede aplicarse directamente el procedimiento EXMIN2:

x_3 x_4 \ x_1 x_2	00	01	11	10
00	1	1	1	
01			1	1
11		1		
10	1			1

Tabla 3.3. Mapa de Karnaugh de 3.152

x_3 x_4 \ x_1 x_2	00	01	11	10
00	1	1	1	
01			1	1
11		1		
10	1			1

Tabla 3.4. Mapa de Karnaugh DSOP de 3.152

1) No es posible aplicar (3.144).

2) Se aplica (3.145) a los términos producto 2 y 3 de (3.154), obteniéndose:

$$f(x_1,x_2,x_3,x_4)=\overline{x}_4\overline{x}_3\overline{x}_1\oplus\overline{x}_3x_2x_1\oplus x_4\overline{x}_3\overline{x}_2x_1\oplus x_4x_3x_2\overline{x}_1\oplus\overline{x}_4x_3\overline{x}_2 \qquad (3.155)$$

No es posible aplicar ahora (3.144), y se intenta aplicar (3.146). Tampoco se puede, y se aplica (3.148) a los términos 2 y 3 de (3.155), quedando:

$$f(x_1,x_2,x_3,x_4)=\overline{x}_4\overline{x}_3\overline{x}_1\oplus\overline{x}_3x_1\oplus\overline{x}_4\overline{x}_3\overline{x}_2x_1\oplus x_4x_3x_2\overline{x}_1\oplus\overline{x}_4x_3\overline{x}_2 \qquad (3.156)$$

No puede aplicarse (3.144), y por tanto se pasa a 3).

3) En el paso 2 se ha aplicado (3.148), por tanto se repite 2), dando preferencia a (3.148) y (3.147).

2) Vuelve a aplicarse (3.148) a los términos 1 y 2 de (3.156):

$$f(x_1,x_2,x_3,x_4)=\overline{x_3}\oplus x_4\overline{x_3}\,\overline{x_1}\oplus\overline{x_4}\,\overline{x_3}\,\overline{x_2}x_1\oplus x_4x_3x_2\overline{x_1}\oplus\overline{x_4}x_3\overline{x_2} \tag{3.157}$$

3) Puesto que se ha aplicado (3.148), se vuelve al paso 2.
2) Se aplica (3.148) a los productos 3 y 5:

$$f(x_1,x_2,x_3,x_4)=\overline{x_3}\oplus x_4\overline{x_3}\,\overline{x_1}\oplus\overline{x_4}\,\overline{x_2}\oplus x_4x_3x_2\overline{x_1}\oplus\overline{x_4}\,\overline{x_3}\,\overline{x_2}\,\overline{x_1} \tag{3.158}$$

3) De nuevo se vuelve al paso 2.
2) Se vuelve a aplicar (3.148), esta vez a los productos 2 y 5:

$$f(x_1,x_2,x_3,x_4)=\overline{x_3}\oplus\overline{x_3}\,\overline{x_1}\oplus\overline{x_4}\,\overline{x_2}\oplus x_4x_3x_2\overline{x_1}\oplus\overline{x_4}\,\overline{x_3}x_2\overline{x_1} \tag{3.159}$$

En esta ocasión puede aplicarse (3.144) a los productos 1 y 2, y así:

$$f(x_1,x_2,x_3,x_4)=\overline{x_3}x_1\oplus\overline{x_4}\,\overline{x_2}\oplus x_4x_3x_2\overline{x_1}\oplus\overline{x_4}\,\overline{x_3}x_2\overline{x_1} \tag{3.160}$$

3) Se vuelve al paso 2.
2) En esta ocasión, se aplica (3.147) a los productos 3 y 4:

$$f(x_1,x_2,x_3,x_4)=\overline{x_3}x_1\oplus\overline{x_4}\,\overline{x_2}\oplus x_4x_2\overline{x_1}\oplus\overline{x_3}x_2\overline{x_1} \tag{3.161}$$

3) Al haberse aplicado (3.147), se vuelve a 2).
2) Aplicando (3.148) en los productos 1 y 4 se obtiene:

$$f(x_1,x_2,x_3,x_4)=\overline{x_3}\oplus\overline{x_4}\,\overline{x_2}\oplus x_4x_2\overline{x_1}\oplus\overline{x_3}\,\overline{x_2}\,\overline{x_1} \tag{3.162}$$

3) Se vuelve al paso 2.
2) No puede aplicarse ninguna de las reglas pertinentes.
3) Se continúa al siguiente paso.
4) No puede aplicarse (3.144).
5) No pueden aplicarse (3.149) ni (3.150).
6) Al repetir los pasos 1, 2, 3, y 4 no se produce reducción de términos, y por tanto se continúa al paso siguiente.
7) En este paso no se consigue reducir el número de términos, por lo que el proceso de minimización se da por concluido.

Por tanto, la expresión mínima que proporciona EXMIN2 para la función booleana (3.152) es la obtenida en (3.162).

Como ha podido observarse en el ejemplo, EXMIN2 es un algoritmo elaborado, y que proporciona buenos resultados, tal y como se podrá comprobar en el Capítulo 6. En el siguiente apartado se va a describir otro procedimiento de minimización basado también en reglas de reescritura.

3.3.5. EXORCISM-MV-2

EXORCISM-MV-2 [SON96] es un procedimiento de minimización AND-EXOR aplicable a funciones multivaluadas incompletamente especificadas y con varias salidas. Se basa en la aplicación de un conjunto convergente de reglas de reescritura (véase el Apartado 3.2.3), que se unifican mediante una operación entre cubos denominada exorlink.

3.3.5.1. Definiciones y propiedades básicas

A continuación se da una serie de definiciones y propiedades referentes al tratamiento de expresiones AND-EXOR en lógica multivaluada.

Definición 3.40. *Una función incompletamente especificada con entrada multivaluada y salida bivaluada es una aplicación:*

$$f(x_1 \ldots x_n): P_1 x P_2 \ldots x P_n \rightarrow B \tag{3.163}$$

donde x_i es una variable multivaluada, $P_i=\{0,1,\ldots,P_i-1\}$ es el conjunto de valores que puede tomar esa variable y $B=\{0,1,-\}$.

Definición 3.41. *Para cualquier subconjunto $S_i \subseteq P_i$, $X_i^{S_i}$ es un literal de X_i; de manera que:*

$$X_i^{S_i} = \begin{cases} 1 \; si \; X_i \in S_i \\ 0 \; si \; X_i \notin S_i \end{cases} \tag{3.164}$$

Definición 3.42. *Un producto de literales, $X_1^{S_1} X_2^{S_2} \ldots X_n^{S_2}$ se denomina término producto. Un término producto que incluye literales de todas las variables X_1, X_2, ... , X_n se denomina término completo.*

Definición 3.43. *Un mapa de una función de n variables de entrada p-valuada y salida bivaluada consiste en P^n celdas. Las celdas que contienen un 1 se van a denominar minterms verdaderos, las que contienen 0 minterms falsos; y las que contienen indiferencias minterms indiferentes.*

Definición 3.44. *Se define la distancia entre dos términos como el número de variables para las que los literales correspondientes tienen diferentes conjuntos de valores verdad.*

Seguidamente se presentan algunas propiedades del álgebra AND-EXOR que se mantienen para lógica multivaluada:

Propiedad 3.45. *Sean A,B y C literales multivaluadas. Se cumplen entonces las siguientes*

leyes:

1) Ley Asociativa: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$ (3.165)

2) Ley Conmutativa: $A \oplus B = B \oplus A$ (3.166)

3) Identidades:

3.a) $A \oplus A = 0$ (3.167)

3.b) $X_i^{S_i} \oplus X_i^{R_i} = X_i^{S_i \oplus R_i}$ (3.168)

3.c) $X_i^{S_i} X_j^{S_j} \oplus X_i^{R_i} X_j^{R_j} = X_i^{S_i \oplus R_i} X_j^{S_j} \oplus X_i^{R_i} X_j^{S_j \oplus R_j} = X_i^{S_i \oplus R_i} X_j^{R_j} \oplus X_i^{S_i} X_j^{S_j \oplus R_j}$ (3.169)

donde la operación $\oplus$ *entre conjuntos viene dada por:*

$$A \oplus B = (A - B) \cup (B - A) \tag{3.170}$$

3.3.5.2. Ideas básicas para la minimización AND-EXOR

El objetivo de la minimización lógica es reducir el valor de una determinada función de coste. Generalmente, el objetivo prioritario consiste en minimizar el número de términos producto, y secundariamente reducir el número total de entradas a las puertas AND y EXOR. Teniendo en cuenta estas consideraciones, la función de coste que se va a utilizar en EXORCISM-MV-2 es:

$$C = C_T + \frac{C_L}{C_{in}} \tag{3.171}$$

donde:

C_T= Número total de términos en la solución.

C_L= Número total de entradas a las puertas AND y EXOR en la solución.

C_{in}= Número total de entradas a las puertas AND y EXOR en la función inicial.

Para reducir esta función de coste se va a utilizar una serie de operaciones entre cubos (que puede comprobarse fácilmente que consituyen un conjunto convergente de reglas de reescritura [BRA93]), que son las siguientes:

1) Eliminación de dos cubos iguales.
2) Combinación de dos cubos que difieran en una variable.
3) Recolocación de dos cubos que difieran en dos variables.
4) Incremento del número de cubos.

La operación 4), que supone un incremento del valor de la función de coste, se introduce para evitar que el procedimiento quede anclado en mínimos locales. Algunas de estas operaciones sólo pueden aplicarse entre cubos que satisfagan una serie de condiciones.

Este hecho introduce una serie de limitaciones que reducen las posibilidades de encontrar una solución de gran calidad y disminuyen la eficiencia del algoritmo. Para poder paliar estos efectos, se va a introducir en los siguientes apartados una nueva operación entre cubos, denominada exorlink.

3.3.5.3. La operación exorlink multivaluada

En este apartado se define en su forma más general la operación exorlink. A continuación, en diferentes apartados se van a estudiar individualmente los casos particulares que se utilizan en el desarrollo del algoritmo de minimización.

Definición 3.46. *Sean:*

$$T_S = X_1^{S_1} \dots X_n^{S_n}$$
$$T_R = X_1^{R_1} \dots X_n^{R_n} \qquad (3.172)$$

dos términos producto. La exorlink ($\otimes$) de los términos T_S y T_R viene definida por la siguiente expresión:

$$T_S \otimes T_R = \bigoplus \left\{ X_1^{S_1} \dots X_{i-1}^{S_{i-1}} X_i^{(S_i \oplus R_i)} X_{i+1}^{R_{i+1}} \dots X_n^{R_n} \mid \textit{Para todo } i=1,\dots,n \textit{ tal que } S_i \notin R_i \right\} \qquad (3.173)$$

Combinando la operación exorlink con el concepto de distancia dado en la Definición 3.44, se tiene:

Definición 3.47. *Dados dos términos producto T_R y T_S cuya distancia es d, la exorlink de ambos, $T_R \otimes T_S$, se denomina exorlink de distancia d.*

En los apartados que siguen se analizan las operaciones exorlink de distancia 1, 2 y 3; que son las utilizadas por EXORCISM-MV-2.

3.3.5.3.1. Exorlink de distancia 1

Sean dos términos productos T_S y T_R, y X^{S}_i, X^{R}_i una pareja de literales de T_S y T_R, respectivamente. Supóngase que $X^{S}_i \neq X^{R}_i$ y que las demás parejas de literales son iguales. En este caso, se puede efectuar la operación exorlink de distancia 1 entre esos términos. La aplicación de la exorlink de distancia 1 sobre dos términos T_R y T_S genera un único término resultante, es decir, produce una reducción en el número de términos, realizando la operación 2) propuesta en 3.3.5.2. A continuación se presenta un ejemplo ilustrativo:

Ejemplo 3.48. Sean:

$$T_S=X^{1,2,3}Y^{2,3} \quad y \quad T_R=X^{0,1}Y^{2,3} \tag{3.174}$$

En ese caso, la exorlink de ambos será:

$$T_S \otimes T_R=X^{1,2,3 \oplus 0,1}Y^{2,3}=X^{0,2,3}Y^{2,3} \tag{3.175}$$

3.3.5.3.2. Exorlink de distancia 2

Dados dos términos que difieren en dos parejas de sus conjuntos verdad (o sea, $X^S_i \neq X^R_i$ y $Y^S_j \neq Y^R_j$) puede aplicarse la exorlink de distancia 2, apareciendo dos nuevos términos que sustituyen a los originales. Por tanto, esta operación no produce ni una reducción ni un incremento del número de cubos, y efectúa la operación 3) de 3.3.5.2. Nótese ademas que la exorlink de distancia 2 $T_S \otimes T_R$ es distinta de $T_R \otimes T_S$, tal y como puede comprobarse en el siguiente ejemplo:

Ejemplo 3.49: Dados:

$$T_S=X^{0,1,3}Y^{1,3} \quad y \quad T_R=X^{2,3}Y^{0,1} \tag{3.176}$$

se obtienen las siguientes dos exorlink:

$$\begin{aligned} T_S \otimes T_R &= X^{0,1,3}Y^{1,3} \otimes X^{2,3}Y^{0,1}=X^{0,1,2}Y^{0,1} \oplus X^{0,1,3}Y^{0,3} \\ T_R \otimes T_S &= X^{2,3}Y^{0,1} \otimes X^{0,1,3}Y^{1,3}=X^{0,1,2}Y^{1,3} \oplus X^{2,3}Y^{0,3} \end{aligned} \tag{3.177}$$

3.3.5.3.3. Exorlink de distancia 3

La operación exorlink de distancia 3 genera tres términos producto a partir de los dos iniciales. Por tanto produce un aumento del número de términos (realiza la operación 4) del Apartado 3.3.5.2), que resulta útil para salir de mínimos locales y obtener mejores resultados en etapas posteriores.

Ejemplo 3.50. Dados los siguientes cubos en lógica binaria; 000x y 0x11, su exorlink de distancia 3 vendría dada por:

$$000x \otimes 0x11=0111 \oplus 00x1 \oplus 0000 \tag{3.178}$$

3.3.5.4. El procedimiento de minimización EXORCISM-MV-2

En los apartados que siguen se presenta el procedimiento EXORCISM-MV-2 para cada tipo de función (funciones completamente especificadas, funciones con varias salidas, y funciones incompletamente especificadas) para pasar a la descripción completa del algoritmo en el Apartado 3.3.5.4.4.

3.3.5.4.1. Minimización de funciones completamente especificadas

En el caso de disponer de funciones completamente especificadas, el proceso de minimización sería el siguiente:

Algoritmo 3.6. Minimización de funciones completamente especificadas

1) Se efectuúan todas las exorlink de distancia 0 y 1 que sean posibles.
2) Se comprueba si existen cubos que difieran en 2 y se calculan las dos posibles exorlink de distancia 2 ($A \otimes B$ y $B \otimes A$).
3) Se comprueba si alguno de los cuatro nuevos cubos difiere en 0 ó 1 con los demás.
4) Si alguno de los cubos del paso anterior difiere en 0 ó 1 se efectúa la operación exorlink de distancia 2 correspondiente seguida de la exorlink de distancia 0 ó 1.
5) Después de realizar la exorlink de distancia 0 ó 1 se vuelve al paso 1). Si no se hubieran encontrado cubos para efectuar la exorlink de distancia 2 a que se hace referencia en 2), se vuelve a dicho paso para comprobar otros dos cubos.

El algoritmo anterior se repite mientras sea posible conseguir una reducción de la función de coste C definida en (3.171).

3.3.5.4.2. Minimización de funciones con varias salidas

La minimización de una función con varias salidas se va a abordar de dos formas distintas:

a) Descomponiendo la multifunción en funciones de una sóla salida, minimizar cada función por separado y finalmente minimizar el conjunto de todas las funciones nuevamente.
b) Minimizar la multifunción directamente.

El algoritmo EXORCISM-MV-2 permite que el usuario escoja cual de los dos procedimientos prefiere utilizar. El algoritmo correspondiente quedaría como sigue:

Algoritmo 3.7. Minimización de funciones con varias salidas

1) Si se selecciona la opción "Descomponer multifunción", ir al paso 2; en otro caso ir al paso 5.
2) Descomponer la multifunción en un conjunto de funciones de una sola salida.
3) Minimizar cada función por separado.
4) Combinar las funciones de una sola salida para generar una multifunción.
5) Minimizar la multifunción.

3.3.5.4.3. Minimización de funciones incompletamente especificadas

Una función incompletamente especificada puede representarse mediante un conjunto de minterms o cubos verdaderos (conjunto ON) y un conjunto de minterms o cubos indiferentes (conjunto DC). La aproximación que se va a utlizar en EXORCISM-MV-2 para la minimización de funciones incompletamente especificadas consiste simplemente en eliminar del conjunto ON aquellos cubos que sean iguales o estén incluidos en algún cubo del conjunto DC. El algoritmo correspondiente sería:

Algoritmo 3.8. Minimización de funciones incompletamente especificadas

1) Eliminar los cubos ON que sean iguales a algún cubo DC.
2) Eliminar los cubos ON que estén incluidos en algún cubo DC.

En el siguiente apartado se presenta el algoritmo EXORCISM-MV-2 completo.

3.3.5.4.4. El algoritmo EXORCISM-MV-2

Según lo estudiado en apartados anteriores, el proceso de minimización EXORCISM-MV-2 parte de un conjunto de cubos disjuntos que representan a la función a minimizar. En primer lugar se eliminan los cubos que sean iguales, y se efectúan las exorlink de distancia 1 que sean posibles. A continuación se realizan las exorlink de distancia 2 que den lugar a simplificaciones posteriores con exorlink de distancia 0 ó 1. Cuando no sea posible efectuar exorlink de distancia 2 bajo esas condiciones, se pasa a efectuar exorlink de distancia 3 que permitan en etapas posteriores aplicar exorlink de distancia 0 ó 1. Después de una serie de iteraciones, si no es posibles reducir el número de términos, se realiza una serie de exorlink de distancia 2 para minimizar el número de conexiones.

En el caso de tener una función con varias salidas, por defecto se realiza una descomposición en funciones de una salida y se minimizan por separado, tal y como se describió en 3.3.5.4.2.

Para la minimización de funciones incompletamente especificadas, en primer lugar se minimiza el conjunto ON, después se efectúa la intersección entre cada cubo del conjunto ON y los cubos del conjunto DC. Si la intersección genera un cubo vacío, el cubo ON se elimina. Una vez que no puedan eliminarse más cubos mediante intersección, se realizan exoslinks de distancia 2 para tratar de encontrar más intersecciones vacías. Finalmente, cada cubo ON se intenta expandir en cubos DC.

El algoritmo presentado es de tipo heurístico, y no es posible conocer si se ha obtenido una solución mínima. Por tanto, es preciso establecer algún criterio para dar por terminada la ejecución del programa. A continuación se dan cuatro posibles criterios de parada:

a) Función de coste. Puede preseleccionarse un determinado valor de la función de coste de manera que el programa se pare en cuanto se alcance dicho valor.
b) Número de iteraciones. El programa termina después de un determinado número de iteraciones. Se trata de un criterio simple, pero que no garantiza la calidad de los resultados.
c) Tiempo de ejecución. El programa termina cuando se excede un determinado tiempo de ejecución.
d) Mejora de la solución actual. En este método, el programa termina cuando no se ha conseguido mejorar la solución después de un determinado número de iteraciones. Éste es el método utilizado por EXORCISM-MV-2, aunque los tres anteriores pueden utilizarse opcionalmente.

Teniendo en cuenta todas las consideraciones presentadas, el algoritmo final es el que se muestra en el Algoritmo 3.9.

En el apartado siguiente se describe un último procedimiento de minimización, basado en la extracción de cubos, y que recibe el nombre de ACEM.

3.3.6. ACEM [ABO95]

El procedimiento de minimización que se presenta a continuación, denominado ACEM (Acelerated Cube Extraction Method) permite la simplificación de funciones completa e incompletamente especificadas, con entrada y salida binarias. Se basa en el mismo principio que los mapas de Karnaugh; trata de encontrar el menor número de cubos posible que cubran los minterms de la función. Para reducir los tiempos de ejecución utiliza la transformada de Hadamard-Walsh; que además permite la aplicación del método para un número elevado de variables.

Algoritmo 3.9. EXORCISM-MV-2

Entrada: Conjuntos ON y DC disjuntos para un función multivaluada

1) *F:=ON; D:=DC.*

2) *SOLUCION:=F; MIN_COST:=COST(F).*
(COST(F) viene dada por la función de coste C).

3) *Si se selecciona "No descomponer la multifunción", ir al paso 5; en otro caso, ir al paso 4.*

4) *Descomponer la función en un conjunto de funciones de una sola salida. Para cada función simple, efectuar los pasos del 5 al 8.*

5) *Efectuar todas las operaciones exorlink de distancia 0 posibles.*

6) *Efecutar todas las operaciones exorlink de distancia 1 posibles.*

7) *Para cada pareja de cubos de F, comprobar si puede efectuarse una exorlink de distancia 2. En caso afirmativo, comprobar si ello hace posible la realización de operaciones exorlink de distancia 0 ó 1. En caso afirmativo, efectuar la exorlink de distancia 2 seguida de la exorlink de distancia 0 ó 1. En cualquier otro caso, no hacer nada.*

8) *Calcular C. Si el número de cubos no se ha reducido en las tres últimas iteraciones ir al paso 9. En otro caso, ir a 5.*

9) *Si se ha seleccionado "No descomponer la multifunción", ir a 11;*
en otro caso, si las funciones simples se han combinado para formar una multifunción ir a 11,
en otro caso, si todas las funciones de una salida se han minimizado ir a 10;
en otro caso, ir a 5 para minimizar la siguiente función de una salida.

10) *Combinar las funciones de una salida para generar una multifunción. Ir al paso 5.*

11) *Para cada pareja de cubos de F, comprobar si es posible efectuar una exorlink de distancia 3. En caso afirmativo, comprobar si esa operación hace posible una exorlink de distancia 0 ó 1. Si es así, se realiza la exorlink de distancia 3 seguida de la exorlink de distancia 0 ó 1. En cualquier otro caso, no hacer nada.*

12) *Calcular el número de cubos. Si no se ha reducido en las últimas 3 iteraciones, ir a 13; en otro caso, ir a 11.*

13) *Comprobar todas las exorlink de distancia 2 posibles. Para cada una, comprobar si reduce la función de coste COST(F). En caso afirmativo, efectuar la operación. En caso contrario, no hacer nada.*

14) *Si al opción "Indiferencias" no se ha seleccionado, ir a 20; en otro caso ir a 15.*

15) *Si D está vacío, ir a 20; en otro caso ir a 16.*

16) *Realizar una intersección entre cada cubo de F y todos los cubos de D. Si se obtiene un cubo vacío, eliminar el cubo ON correspondiente de F.*

17) *Para cada pareja de cubos de F, comprobar si es posible efectuar una exorlink de distancia 2. En caso afirmativo, comprobar si alguno de los cubos resultantes puede intersectarse con D. En caso afirmativo, realizar la exorlink de distancia 2 y efectuar la intersección. Eliminar los cubos ON correspondientes de F. En otro caso, no efectuar la exorlink de distancia 2.*

18) *Si el número de cubos no se ha reducido en las tres últimas iteraciones, ir a 19; en otro caso ir a 15.*

19) *Expandir cada cubo de F en D si es posible.*

20) *Escribir la salida y terminar el programa*

Definición 3.51. *Considerese un sistema combinacional con n entradas y una sola salida F(X). Un término producto o cubo con n-m (m≤n) literales que cubre un grupo de 2^m minterms se va a denominar m-grupo.*

El procedimiento ACEM va a tratar de cubrir todos los minterms de la función utilizando el menor número de grupos necesario. De forma análoga a como se opera en los procedimientos de minimización AND-OR, en primer lugar se buscan los grupos más grandes, y posteriormente se va reduciendo el tamaño de los mismos hasta que queden cubiertos todos los minterms. Para conseguir la identificación de los grupos sin tener que recurrir a la utilización de mapas se opera en el dominio de la transformada de Hadamard-Walsh [HUR85].

El proceso de minimización viene dado por el siguiente par de ecuaciones:

$$f_{k+1}(X)=f_k \oplus g_k(X) \; ; \; f_0=f(X) \tag{3.179}$$

$$G_{k+1}(X)=G_k(X) \oplus g_k(X) \; ; \; G_0=0 \tag{3.180}$$

donde:

- k=0,1,2,3.. es un índice de iteración.
- f_k y f_{k+1} son la función residual actual y la función residual siguiente, respectivamente.
- $g_k(X)$ es la función del grupo de exploración, correspondiente al grupo de minterms a extraer de la función residual actual en la iteración k-ésima.
- $G_k(X)$ es la cobertura básica actual.

Las ecuaciones (3.179) y (3.180) detallan cómo se realiza la extracción de los grupos especificados por g_k de la función residual actual f_k. Los minterms verdad de f_k cubiertos por g_k se eliminan de f_{k+1}, mientras que los minterms falsos de f_k cubiertos por g_k se reemplazan por minterms verdad en f_{k+1}. En las ecuaciones citadas, sólo quedan por especificar los grupos de exploración g_k. El procedimiento empieza haciendo $m=n$-1 y buscando *m-grupos* de exploración, es decir, grupos de la función residual actual que cubran al menos $\alpha 2^m$ minterms verdaderos de f_k, donde α es una constante predeterminada en el rango $0.5<\alpha<1$.

Definición 3.52. *Se define el tamaño s(g) de un m-grupo de exploración como el número de minterms verdaderos de f_k cubiertos por el m-grupo.*

La idea del procedimiento consiste en extraer el *m-grupo* con mayor tamaño posible. Si se extrae un *m-grupo* de tamaño s, la función residual siguiente habrá disminuido en $2s$-2^m ($\geq(2\alpha$-1$)2^m$) su número de minterms verdaderos con respecto a f_k. En el caso de que existan dos posibles grupos adyacentes, la extracción de éstos puede combinarse. Para los casos en que se encuentren varios grupos candidatos a ser extraidos, debe seguir el orden de prioridad

(de menor a mayor) que se da a continuación:

- Grupos tipo 1: Un *m-grupo* de exploración $p_{n-m-1}x_i$ adyacente a un *(m-1)-grupo* de exploración $p_{n-m-1}\bar{x}_i x_j$ donde p_{n-m-1} es un término producto de n-m-1 literales que no incluye a los literales x_i y x_j. La función de grupo de exploración correspondiente viene dada por $g_k=p_{n-m-1}(1\oplus\bar{x}_i\bar{x}_j)$.
- Grupos tipo 2: Un *m-grupo* de exploracón $p_{n-m-1}x_i$, que tendrá como función de grupo $g_k=p_{n-m-1}x_i$.
- Grupos tipo 3: Un *(m-1)-grupo* de exploración $p_{n-m-1}x_i x_j$ adyacente a otro *(m-1)-grupo* de exploración $p_{n-m-1}\bar{x}_i\bar{x}_j$. La función de grupo correspondiente es $g_k=p_{n-m-1}(x_i\oplus\bar{x}_j)$.

Tras efectuar una serie de iteraciones, se llegará a una situación en la que la función residual actual será $f_k=0$, con lo que la función original $f(X)$ y la cobertura básica G_k se igualan. En ocasiones, puede ser necesario reducir esta cobertura básica G_k, para lo que se van a utilizar una serie de operaciones definidas en [EVE67]; concretamente las de fusión, incremento de orden, exclusión y puenteo.

A continuación se presenta un ejemplo ilustrativo de cómo opera el procedimiento ACEM:

Ejemplo 3.53: Considérese el sistema de cuatro variables:

$$f(X)=\sum m(0,2,4,5,7,8,10,11,12,13) \tag{3.181}$$

cuyo mapa de Karnaugh puede verse en la Tabla 3.5. Tal y como se mencionó anteriormente, se comienza el proceso de minimización buscando *m-grupos* de exploración con m=n-1. En este caso, se empezará buscando *3-grupos*. En la Tabla 3.5 puede observarse que el mayor *3-grupo* de tipo 1 que se encuentra es el formado por el *3-grupo* $\bar{x}_1$ y el *2-grupo* x_1x_3. Asimismo, se tienen dos grupos de exploración de tipo 2; $\bar{x}_1$ y $\bar{x}_2$. Finalmente, pueden encontrarse dos grupos de tipo 3 formados por las parejas de *2-grupos* $\bar{x}_1\bar{x}_3$, x_1x_3 y $\bar{x}_2x_3$, $x_2\bar{x}_3$. De acuerdo con el orden de prioridades proporcionado anteriormente, se procederá a extraer el grupo de tipo 3 constituido por la pareja $\bar{x}_1\bar{x}_3$, x_1x_3. La función de grupo correspondiente será $g_0=\bar{x}_1\oplus x_3$, tras cuya aplicación se obtiene el mapa de la Tabla 3.6. Nótese que el minterm falso marcado con un * en la Tabla 3.5 aparece como un 1 en la 3.6. En el mapa de la Tabla 3.6 puede verse que la función residual $f_1(X)$ está formada por dos *1-grupos*, $\bar{x}_1\bar{x}_2x_3$ y $x_1x_2x_4$. Una vez extraídos estos grupos, la función residual actual queda igualada a 0, obteniéndose la siguiente función optimizada:

$$f(X)=\bar{x}_1\oplus x_3\oplus\bar{x}_1\bar{x}_2x_3\oplus x_1x_2x_4 \tag{3.182}$$

Esta cobertura básica no puede reducirse, y constituye la solución final proporcionada por ACEM.

x_4 x_3 \ x_2 x_1	00	01	11	10
00	1			1
01	1	1	1	
11	1	1	*	
10	1		1	1

$$g_0=\bar{x}_1\oplus x_3$$

Tabla 3.5. Ejemplo 3.53. Mapa de f_0

x_4 x_3 \ x_2 x_1	00	01	11	10
00				
01	1			
11	1		1	
10			1	

$$g_1=\bar{x}_1\bar{x}_2x_3 \qquad : \qquad g_2= x_1x_2x_4$$

Tabla 3.6. Ejemplo 3.53. Mapa de f_1

El procedimiento descrito aquí puede extenderse fácilmente al caso de sistemas incompletamente especificados. Considérese un *m-grupo* de exploración que cubre s_t minterms verdaderos y s_d minterms indiferentes de f_k. Para que se produzca una reducción en el número de minterms en f_{k+1} es necesario que $2^m-(s_t+s_d)<s_t$ y también que $(s_t+s_d)\geq\beta 2^m$, donde β es una constante en el rango $0.5<\beta<1$. Por tanto, en el caso de tener funciones incompletamente especificadas, un *m-grupo* de exploración debe cumplir $(2s_t+s_d)>2^m$ y $(s_t+s_d)\geq\beta 2^m$. La extracción de la cobertura básica finaliza cuando la función residual actual está constituida únicamente por indiferencias. Como ilustración se presenta el siguiente ejemplo:

Ejemplo 3.54. Considérese el sistema de cuatro variables incompletamente especificado:

$$f(X)=\sum m(0,3,6,8,13)+d(2,5,15) \tag{3.183}$$

cuyo mapa de Karnaugh puede verse en la Tabla 3.7. Utilizando β=0.75 para m=3 no se encuentra ningún grupo de exploración. Para m=2 existen tres grupos de exploración, $\bar{x}_4\bar{x}_3$, $\bar{x}_4x_2$, $\bar{x}_3\bar{x}_1$. Se selecciona el cubo $g_0=\bar{x}_4\bar{x}_3$, resultando el mapa de la Tabla 3.8. El minterm falso de g_0 que se encuentra marcado con * en la Tabla 3.7, produce un 1 en f_1, mientras que la indiferencia de g_0 permanece en f_1, según puede verse en la Tabla 3.8. Siguiendo iterativamente el proceso se obtendrían sucesivamente los mapas de las tablas 3.9 y 3.10, llegándose a la siguiente cobertura básica, que no puede ser reducida:

$$f(X)=\bar{x}_4\bar{x}_3\oplus\bar{x}_2x_1\oplus x_4\bar{x}_3\bar{x}_2\oplus\bar{x}_4x_2\bar{x}_1 \tag{3.184}$$

Todo el proceso de búsqueda y extracción de cubos puede realizarse sin necesidad de recurrir a mapas mediante la utilización de la transformada de Hadamard-Walsh. En [ABO95] se describe en detalle todo el proceso, y en [HUR85] puede obtenerse más información acerca de esta técnica espectral.

En el Capítulo 6 pueden verse los resultados que es capaz de obtener el procedimiento ACEM, comparándolos con los obtenidos por EXMIN2, el procedimiento desarrollado en esta memoria (RRMIN2, Capítulo 4), y algunos otros procedimientos.

x_4 x_3 \ x_2 x_1	00	01	11	10
00	1	*	1	d
01		d		1
11		1	d	
10	1			

$g_0=\bar{x}_4\bar{x}_3$

Tabla 3.7. Ejemplo 3.53. Mapa de f_0

x_4 x_3 \ x_2 x_1	00	01	11	10
00		1		d
01		d		1
11		1	d	
10	1	*		

$$g_1=\bar{x}_2x_1$$

Tabla 3.8. Ejemplo 3.53. Mapa de f_1

x_4 x_3 \ x_2 x_1	00	01	11	10
00				d
01		d		1
11			d	
10	1	1		

$$g_2=x_4\bar{x}_3\bar{x}_2$$

Tabla 3.9. Ejemplo 3.53. Mapa de f_2

x_4 x_3 \ x_2 x_1	00	01	11	10
00				d
01		d		1
11			d	
10				

$$g_3=\bar{x}_4x_2\bar{x}_1$$

Tabla 3.10. Ejemplo 3.53. Mapa de f_3

3.4. CONCLUSIONES

En este Capítulo se ha presentado una serie de procedimientos destinados a la minimización de funciones de conmutación en lógica AND-EXOR. Se ha visto que existen dos tipos de procedimientos, uno constituido por aquellos que buscan el mínimo exacto,y otro formado por los que utilizan métodos heurísticos para aproximarse a la solución mínima. Los procedimientos exactos sólo son aplicables a funciones con un número reducido de variables o términos producto debido a la gran cantidad de tiempo de ejecución y/o memoria que precisan. Por el contrario, los procedimientos heurísticos, aunque no siempre obtienen el mínimo exacto, permiten su aplicación a funciones más complejas, resultanto por tanto, de mayor utilidad para el diseño de circuitos que los procedimientos exactos. En el Capítulo 6 pueden verse resultados proporcionados por algunos de estos procedimientos, concretamente EXMIN2 y ACEM, comparados con los de RRMIN2 (Capítulo 4).

CAPÍTULO 4

EL PROCEDIMIENTO DE MINIMIZACIÓN RRMIN2

En este Capítulo se desarrolla un nuevo procedimiento de minimización AND-EXOR basado en reglas de reescritura cuya aplicación se encuentra controlada por un proceso de Enfriamiento Simulado. Este procedimiento, que denominaremos RRMIN2, permite la minimización de funciones de conmutación tanto completa como incompletamente especificadas y con una o varias salidas (síntesis multifuncional).

4.1. INTRODUCCIÓN

En el Capítulo anterior se ha realizado una revisión de los principales procedimientos existentes para la minimización AND-EXOR. De entre ellos, los que mejor resultado proporcionan son los basados en reglas de reescritura (EXMIN2, [SAS93a]) u otro tipo de operaciones que impliquen tanto reducciones como expansiones de la función (EXORCISM-MV-2, [SON96]). El procedimiento que se propone a continuación, al que se denominará en adelante RRMIN2, está basado también en reglas de reescritura, pero en lugar de utilizar un proceso de asignación de reglas determinista, se realiza una selección aleatoria dirigida por un proceso de Enfriamiento Simulado (Simulated Annealing, SA, [KIR83]]) adaptado al problema en cuestión.

El algoritmo de Enfriamiento Simulado ha sido empleado con éxito en la resolución aproximada de problemas de optimización combinatoria muy diversos. Aquí se propone su utilización para la resolución del problema de la minimización AND-EXOR, que al igual que los enunciados anteriormente, constituye un problema de optimización combinatoria NP-completo [GAR79].

La utilización de un procedimiento no determinista de selección de reglas presenta dos ventajas fundamentales frente a los procedimientos deterministas:

- Es menos propenso a permanecer en mínimos locales.
- Permite mayor flexiblidad para controlar la relación {calidad de la solución}/{tiempo de CPU} que se desee. De esta forma, si se está en la fase de realización de prototipos de un diseño VLSI e interesa obtener una solución rápida para efectuar pruebas, pueden ajustarse los parámetros del Enfriamiento Simulado para que el tiempo de CPU sea pequeño, a costa de sacrificar hasta cierto límite prefijado la calidad en la solución, o al contrario si se desea realizar el diseño definitivo.

En contraposición, los procedimientos de selección aleatoria, plantean dos inconvenientes:

- No existe un criterio claro para decidir cuándo se ha llegado al final del proceso de minimización.
- Puede darse el caso de que al aplicar el algoritmo varias veces a una misma función se obtengan soluciones distintas (aunque este efecto puede ser paliado usando parámetros adecuados).

Por ser un procedimiento de selección aleatoria de reglas, RRMIN2 presenta las ventajas e inconvenientes citados anteriormente, y que se analizan en el Capítulo 6.

En los apartados que siguen, se describe el planteamiento de la minimización AND-EXOR como un problema de optimización combinatoria (Apartado 4.2) y se presenta el algoritmo de Enfriamiento Simulado como una potente herramienta para la resolución de este tipo de problemas (Apartado 4.3). En el Apartado 4.4 se dan los elementos básicos de un nuevo procedimiento para la minimización AND-EXOR mediante Reglas de reescritura. Finalmente, los apartados 4.5 al 4.10 se dedican al desarrollo del procedimiento RRMIN2.

4.2. LA MINIMIZACIÓN AND-EXOR COMO PROBLEMA DE OPTIMIZACIÓN COMBINATORIA

En el Apartado 2.5 se comentó que la primera cuestión que surge al diseñar un procedimiento de minimización AND-EXOR es la de plantear el problema para poder atacarlo de forma adecuada. En este Apartado se va a mostrar un posible plantemiento del mismo que se va a utilizar posteriormente para desarrollar un nuevo procedimiento de minimización. A

continuación se define el concepto de problema de optimización combinatoria [AAR90]:

Definición 4.1: *Un problema de optimización combinatoria es un problema de minimización o de maximización que viene especificado por un conjunto de casos del problema.*

Definición 4.2: *Un problema de optimización combinatoria puede formalizarse como un par (S,f), donde S es el conjunto finito de todas las posibles soluciones y f es una función denominada función de coste y que está definida sobre los reales:*

$$f: S \rightarrow \mathbb{R} \tag{4.1}$$

En el caso de la minimización, el problema consistirá en encontrar una solución $i_{opt} \in S$ *que satisfaga:*

$$f(i_{opt}) \leq f(i), \text{ para todo } i \in S \tag{4.2}$$

y si se trata de un problema de maximización, i_{opt} *cumplirá:*

$$f(i_{opt}) \geq f(i), \text{ para todo } i \in S \tag{4.3}$$

La solución i_{opt} *se dice que es una solución globalmente óptima del problema, o simplemente un óptimo (mínimo ó máximo);* $f_{opt}=f(i_{opt})$ *denota el coste óptimo, y* S_{opt} *es el conjunto de las soluciones óptimas (la solución no tiene por qué ser única).*

Según las definiciones anteriores, para plantear la minimización AND-EXOR como un problema de optimización combinatoria es preciso especificar un espacio de soluciones finito *S*, y definir una función de coste *f*. Si, por ejemplo, se quiere realizar una minimización dentro de las MPRM (Apartado 2.4.1.2.2), el conjunto *S* estaría formado por las 3^n formas canónicas MPRM correspondientes, y la función de coste sería el número de términos de cada forma.

En el caso de que se desee plantear el problema de manera general para las formas ESOP, la función de coste sería la misma, pero es preciso efectuar alguna consideración para garantizar que *S* tenga un número finito de elementos. Concretamente, según la definición dada en 2.4.2, la forma general de una expresión ESOP es:

$$f(x_1, x_2, \ldots, x_n) = \bigoplus x_1^* x_2^* \ldots x_n^* \tag{4.4}$$

donde x_i^* puede ser 1, x_i ó $\overline{x_i}$. Notese que la sumatoria no tiene índices, es decir, el número de términos producto que aparecere en la expresión es indeterminado, dependerá en cada caso de la función de conmutación que se esté utilizando. Por tanto, puede haber términos repetidos en la sumatoria, lo que podría dar lugar a expresiones con infinitos términos producto. Si se tiene en cuenta que $x \oplus x = 0$, es inmediato el extraer las siguientes conclusiones:

- La sumatoria de (4.4) puede tener infinitos términos.
- Una expresión ESOP que contenga dos términos repetidos no puede ser una expresión mínima.

Por tanto, se puede elegir el espacio de soluciones S como el conjunto de las expresiones ESOP que no tienen términos repetidos, y tener así un conjunto S finito (existen 3^n términos producto distintos, y por consiguiente, el número de combinaciones que pueden formarse con esos 3^n términos será finito), y además las soluciones óptimas pertenecerán a ese conjunto (las expresiones ESOP no pertenecientes a S no pueden ser soluciones mínimas, puesto que podría simplificarse aplicando $x \oplus x = 0$).

Una vez que se ha planteado la minimización AND-EXOR como problema de optimización combinatoria, debe de considerarse algún procedimiento para su resolución. Existen varios algoritmos generales para la resolución de este tipo de problemas, como por ejemplo los algoritmos genéticos (GA) [GOL89], el algoritmo de Enfriamiento Simulado (SA) [AAR90], etc. Los apartados que siguen se dedican a la descripción del algoritmo de Enfriamiento Simulado, utilizado ampliamente en la resolución de problemas de optimización combinatoria [LAA87][WON88], y que va a formar parte de RRMIN2.

4.3. EL ALGORITMO DE ENFRIAMIENTO SIMULADO

Desde su introducción en 1983 [KIR83], el algoritmo de Enfriamiento Simulado (SA, Simulated Annealing) se ha venido aplicando a una gran variedad de problemas de optimización combinatoria en diversas areas como el problema del viajante de comercio [AAR88], el particionamiento de grafos [AAR85], problemas de asignación cuadrática [BON86][BUR84], coloreado de grafos [BHA87][CHA87][JOH88], diseño VLSI (problemas de emplazamiento [BAN90][CAS87a], enrutamiento [LEO86], test [DIS86], etc), diseño de códigos [BEE85][LAA88], procesamiento de imágenes [CAR85][GEM87], Biología [DRE87], Física molecular [LYB87], etc. Debido a la gran cantidad de aplicaciones y a los buenos resultados que ha proporcionado en todas ellas, este algoritmo resulta de gran interés, y los siguientes apartados se dedican a su descripción detallada.

4.3.1. Búsqueda Local

La Búsqueda Local constituye una clase interesante de algoritmos generales de aproximación que están basados en la mejora progresiva del valor de la función de coste mediante la exploración de vecindades. Estos algoritmos guardan una estrecha relación con el algoritmo de Enfriamiento Simulado, y es por esta razón por la que se van a describir y discutir algunas propiedades de los algoritmos de búsqueda local.

Para la utilización de un algoritmo de búsqueda local se precisa de las definiciones dadas en 4.2, es decir, de un conjunto de soluciones y de una función de coste; y además de una estructura de vecindad [AAR90].

Definición 4.3: *Sea (S,f) un problema de optimización combinatoria. Entonces, una estructura de vecindad es una aplicacion:*

$$N: S \rightarrow 2^S \tag{4.5}$$

que define para cada solución $i \in S$ un conjunto $S_i \subset S$ de soluciones que están "cerca" de i en algún sentido. El conjunto S_i se conoce con el nombre de vecindad de la solución i, y cada $j \in S_i$ se denomina solución vecina o vecino de i. Además, se supondrá que $j \in S_i \Leftrightarrow i \in S_j$.

Definición 4.4: *Sea (S,f) un problema de optimización combinatoria y N una estructura de vecindad. Entonces, un mecanismo de generación es una forma de seleccionar una solución j de la vecindad S_i de una solución i.*

Dado un problema de optimización combinatoria y una estructura de vecindad, un algoritmo de búsqueda local (ver Algoritmo 4.1) opera de la siguiente forma:

- Se empieza con una solución inicial, generalmente elegida de forma aleatoria.
- Se intenta encontrar una solución mejor (con menor coste) buscando en la vecindad de la solución actual. Si se consigue encontrar esa solución mejor, ésta reemplaza a la anterior y se vuelve a iniciar la búsqueda.
- Si no es posible encontrar una solución mejor, el algoritmo finaliza.

Algoritmo 4.1. Búsqueda Local

```
BL()
Iniciar(i_inicial);
i=i_inicial;
Repetir
        Generar(j a partir de S_i);
        Si(f(j)<f(i))
                i=j;
        Fin Si
Hasta (f(j)≥f(i) para todo j∈S_i);
Fin BL
```

La idea del algoritmo de Búsqueda Local es restringir la búsqueda a una parte del espacio, que se denomina vecindad, para evitar una búsqueda exhaustiva sobre la totalidad del mismo. Ésta es una característica general de los algoritmos para la resolución de problemas de optimización combinatoria, puesto que los espacios de búsqueda que aparecen en los mismos suelen ser de gran tamaño. Por tanto, la efectividad de este tipo de algoritmos dependerá fundamentalmente de que esta búsqueda restringida esté bien dirigida de forma que

conduzca a la obtención de una buena solución. En el caso de la búsqueda local, la elección de la solución inicial tiene una gran importancia, porque si ésta se encuentra muy "alejada" de la solución mínima, existe una gran probabilidad de que el procedimiento se quede anclado en mínimos locales. Las siguientes definiciones formalizan esta cuestión.

Definición 4.5: *Sea (S,f) un problema de optimización combinatoria y sea N una estructura de vecindad. Entonces $\hat{i} \in S$ se denomina solución localmente óptima o simplemente un óptimo local con respecto a N si el costo de $\hat{i}$ es menor o igual que el de todas las soluciones de su vecindad. Más específicamente, $\hat{i}$ es una solución localmente mínima (también denominado un mínimo local) si:*

$$f(\hat{i}) \leq f(j), \text{ para todo } j \in S_{\hat{i}} \tag{4.6}$$

Definición 4.6: *Sea (S,f) un problema de optimización combinatoria y N una estructura de vecindad. Entonces, N se denomina exacta si, para cada $\hat{i} \in S$ que es localmente óptima con respecto a N, $\hat{i}$ es también una solución globalmente óptima.*

Así, por definición, los algoritmos de búsqueda local terminan en un óptimo local y, salvo en el caso de que se tenga una vecindad exacta, no es posible determinar la distancia existente entre el óptimo local y el óptimo global. El considerar una vecindad exacta equivale a efectuar una enumeración completa de los casos del problema de optimización combinatoria, lo que resulta irrealizable en la práctica. Por otra parte, según se ha comentado anteriormente, la calidad de la solución obtenida en un algoritmo de búsqueda local depende fuertemente de la solución inicial elegida, y desgraciadamente, en la mayor parte de los problemas no se dispone de un procedimiento que proporcione una elección adecuada de esta solución de partida. A pesar de estos inconvenientes, el algoritmo de búsqueda local presenta las ventajas de que es aplicable de forma general, y es flexible. Además, pueden considerarse las siguientes alternativas para paliar de alguna manera las desventajas:

- Ejecutar el algoritmo de búsqueda local para un gran número de soluciones iniciales. Asintóticamente (es decir, en el caso de que se llegaran a probar todos los elementos del espacio de soluciones), este algoritmo encontraría la solución óptima global con probabilidad 1.
- Introducir una estructura de vecindad compleja de manera que se efectúe una búsqueda sobre una parte muy amplia del espacio. Encontrar una estructura tal suele resultar complicado y requiere un gran conocimiento del problema de optimización combinatoria que se esté tratando.
- Aceptar de forma limitada transiciones correspondientes a un incremento en la función de coste. Nótese que en la búsqueda local sólo se admiten transiciones que decrementan el valor de dicha función.

El algoritmo de Enfriamiento Simulado sigue la última alternativa. Este algoritmo, que

debe su nombre a la analogía existente con el proceso físico del enfriamiento de sólidos, encuentra soluciones de una gran calidad, que no dependen de una manera tan determinante de la elección de la solución inicial, siendo por tanto, un algoritmo efectivo y robusto. En la sección siguiente se describe el Algoritmo de Metrópolis, basado directamente en el proceso físico que dio lugar al Algoritmo de Enfriamiento Simulado.

4.3.2. El Algoritmo de Metrópolis

En Física de la Materia Condensada, el "annealing" (enfriamiento) es un proceso térmico para obtener estados de baja energía de un sólido en un baño caliente. El proceso consta de los siguientes dos pasos [BAR76]:

a) Incrementar la temperatura del baño caliente a un valor máximo al cual el sólido funde.
b) Decrementar cuidadosamente la temperatura del baño hasta que las partículas se autoorganicen en el estado de mínima energía del sólido.

En el estado líquido, las partículas del sólido se organizan de forma aleatoria, mientras que en el estado de mínima energía, las partículas se encuentran formando una estructura muy determinada. El estado de mínima energía se alcanza sólo si la temperatura máxima es lo bastante alta y el enfriamiento se realiza de forma suficientemente lenta. Si no se cumplen estos requisitos el sistema caerá en un estado metastable.

El proceso físico descrito anteriormente puede modelarse utilizando una serie de técnicas de cálculo propias de la Física de la Materia Condensada. En [MET53] se introduce un algoritmo simple para simular la evolución de un sólido en un baño caliente en equilibrio térmico. Este algoritmo está basado en técnicas de Monte Carlo, y genera una secuencia de estados del sólido de la siguiente forma:

- Dado un estado i del sólido con energía E_i, el subsiguiente estado j se genera mediante la aplicación de una pequeña perturbación, por ejemplo, el desplazamiento de una partícula. La energía del estado siguiente será E_j.
- Si la diferencia de energía, E_j-E_i es menor o igual que 0, el estado j se acepta como nuevo estado actual.
- Si la diferencia es mayor que 0, el estado j se acepta con una cierta probabilidad, que viene dada por:

$$\exp\left(\frac{E_i - E_j}{k_B T}\right) \tag{4.7}$$

donde T es la temperatura del baño caliente y k_B la constante de Boltzman. La regla de aceptación descrita es conocida como Criterio de Metrópolis, y el algoritmo

correspondiente como Algoritmo de Metrópolis.

Si el decremento de temperatura se efectúa de manera suficientemente lenta, el sólido puede alcanzar el equilibrio térmico en cada temperatura. El equilibrio térmico viene caracterizado por la distribución de Boltzmann [TOD83]. Según esta distribución, la probabilidad de que un sólido se encuentre en un estado i con energía E_i a la temperatura T viene dada por:

$$\wp_T\{X=i\} = \frac{1}{Z(T)} \exp\left(\frac{-E_i}{k_B T}\right) \tag{4.8}$$

donde X es una variable estocástica que representa el estado actual del sólido. $Z(T)$ es la función de partición, que se define como:

$$Z(T) = \sum_j \exp\left(\frac{-E_j}{kBT}\right) \tag{4.9}$$

donde la sumatoria se extiende sobre todos los estados posibles. En el siguiente apartado se describe el algoritmo de Enfriamiento Simulado, y podrá comprobarse el importante papel que juega en el mismo la distribución de Boltzmann.

4.3.3. Enfriamiento Simulado

Combinando los conceptos de Búsqueda Local y el Algoritmo de Metrópolis, puede aplicarse este último para generar una secuencia de soluciones de un problema de optimización combinatoria. Para ello, considerense las siguientes analogías entre un sistema físico de muchas partículas y un problema de optimización combinatoria:

- Las soluciones en un problema de optimización combinatoria son equivalentes a los estados de un sistema físico.
- El coste de una solución es equivalente a la energía de un estado.

Añadiendo a estas analogías un parámetro de control que juegue el mismo papel que la temperatura, el algoritmo de Enfriamiento Simulado puede describirse como una iteración de algoritmos de Metrópolis, evaluación de los mismos, y decremento de la temperatura.

A continuación se dan una serie de definiciones que llevarán a la obtención del algoritmo de Enfriamiento Simulado. Al igual que en la Búsqueda Local, se va a partir de la existencia de una estructura de vecindades y un mecanismo de generación [AAR90].

Definición 4.7: *Sea (S,f) un problema de optimización combinatoria e i y j dos soluciones con costo f(i) y f(j), respectivamente. Entonces, el criterio de aceptación determina cuándo j es*

aceptado partiendo de i aplicando la siguiente probabilidad de aceptación:

$$\wp_c\{aceptar\ j\} = \begin{cases} 1 & Si\ f(j) \leq f(i) \\ \exp\left(\dfrac{f(i)-f(j)}{c}\right) & Si\ f(j) > f(i) \end{cases} \tag{4.10}$$

donde $c \in \mathbb{R}^+$ *denota el parámetro de control.*

El mecanismo de generación corresponde a la perturbación que se introducía en el Algoritmo de Metrópolis (en el que el criterio de aceptación viene dado por (4.7)).

Definición 4.8: *Una transición es una acción combinada que transforma una solución actual en otra. La acción consiste en los siguientes dos pasos:*

i) Aplicación del mecanismo de generación.
ii) Aplicación del criterio de aceptación.

Si se denomina c_k al valor del parámetro de control y L_k al número de transiciones generadas en la iteración k-ésima de algoritmo de Metrópolis, el algoritmo de Enfriamiento Simulado puede describirse en pseudocódigo según se muestra en el Algoritmo 4.2. Este algoritmo puede verse como una generalización de la Búsqueda Local (Apartado 4.3.1) en el que se admite la generación de soluciones con un coste peor que la anterior y permite por tanto, efectuar una búsqueda mucho más amplia en el espacio de soluciones. Esta generación de soluciones con mayor costo, no obstante, depende del parámetro de control c, de forma que a temperaturas altas pueden aceptarse configuraciones con un costo mucho mayor, pero conforme c disminuye, la probabilidad de aceptar soluciones con un peor costo va disminuyendo. Cuando c se encuentra próxima a 0 sólo se aceptan soluciones con costo menor. De esta forma se consigue que el Enfriamiento Simulado escape de los mínimos locales, mejorando de forma notable la Búsqueda Local. Además, este algoritmo sigue siendo simple y de aplicación general.

Nótese que la probabilidad de aceptar soluciones peores se obtiene mediante la comparación entre el valor de $\exp((f(i)-f(j))/c)$ y un número aleatorio generado uniformemente en el intervalo [0,1). Por tanto, la velocidad de convergencia del algoritmo viene determinada por la elección de los parámetros L_k y c_k, k=0,1, ...

Utilizando la teoría de las cadenas finitas de Markov [FEL50], [SEN81], puede demostrarse [AAR90] que bajo ciertas condiciones (elección adecuada de los parámetros anteriores) se tiene garantizada la convergencia asintótica del algoritmo de Enfriamiento Simulado a un óptimo global. No obstante, sería preciso un número infinito de transiciones para alcanzar esta solución, y en la práctica es necesario utilizar un esquema de enfriamiento que se aproxime a estas condiciones de convergencia asintótica. Un esquema de enfriamiento simulado que da buenos resultados en la práctica es el siguiente:

Algoritmo 4.2. Enfriamiento Simulado

```
SA()
Iniciar(x_inicial, c_0, L_0);
k=0;
x=x_inicial;
Repetir
        l=1;
        Mientras(l<=L_k) hacer
                Generar(y a partir de S_x );
                Si (f(y)<=f(x))
                        x=y;
                Fin Si
                Si (f(y)>f(x))
                        Si (exp((f(x)-f(y))/c_k)>aleatorio[0,1))
                                x=y;
                        Fin Si
                Fin Si
                l=l+1;
        Fin Mientras
        k=k+1;
        Calcular_Longitud(L_k);
        Calcular_Control(c_k);
Hasta Criterio_de_fin()
Fin SA
```

- **Valor inicial del parámetro de control, c_0:** El valor c_0 debe ser tal que la probabilidad de aceptar una transición (sea para obtener una solución mejor o una solución peor) sea próxima a 1.
- **Decremento del parámetro de control.** Suele utilizarse la siguiente función de decremento:

$$c_k=\alpha \cdot c_k,\ k=1,2,.... \tag{4.11}$$

donde $0<\alpha<1$, pero puesto que el decremento debe hacerse de forma lenta, α debe ser próxima a 1. Valores típicos suelen estar entre 0.8 y 0.99.
- **Valor final del parámetro de control.** El proceso de Enfriamiento Simulado suele darse por concluido cuando la solución obtenida en el último intento de una cadena de Markov permanece invariada para un número determinado de cadenas consecutivas.
- **Longitud de las cadenas de Markov, L_k.** La longitud de estas cadenas debe ser tal que se consiga obtener condiciones de cuasi-equilibrio para cada valor de c_k. Esto implica que $L_k \rightarrow \infty$ cuando $c_k \rightarrow 0$, por lo que debe acotarse de alguna forma la longitud de estas cadenas para evitar longitudes excesivas. Generalmente, suele utilizarse $L_k=N$, es decir, se elije como longitud de las cadenas de Markov el tamaño de la vecindad.

Una vez descrito un posible esquema de enfriamiento, el algoritmo de Enfriamiento Simulado sólo precisará de un conjunto de soluciones, una función de coste y un mecanismo de generación para ser aplicado a un problema concreto.

En los apartados que siguen se va a desarrollar un nuevo procedimiento de minimización AND-EXOR que va a utilizar un proceso de Enfriamiento Simulado para dirigir la aplicación de un conjunto convergente de reglas de reescritura.

4.4. ELEMENTOS BÁSICOS DE RRMIN2

RRMIN2 es un procedimiento de minimización AND-EXOR basado en reglas de reescritura, y como tal, está constituido por tres elementos basicos:

1) **Una expresión inicial AND-EXOR de la función a minimizar.** Usualmente, la primera descripción que se tiene de una función de conmutación es su tabla verdad; de manera que es necesario pasar esta expresión a lógica AND-EXOR para poder aplicar sobre ella las reglas. La expresión AND-EXOR inicial que utiliza RRMIN2 es la forma canónica MRPM (Apartado 2.4.1.2.2) óptima.

2) **Un conjunto de reglas de reescritura.** Este conjunto de reglas se aplicará a la expresión inicial para ir obteniendo sucesivamente nuevas expresiones AND-EXOR de la función, optimizándola. El conjunto de reglas seleccionado viene detallado en el Apartado 4.6.

3) **Un mecanismo de control de la aplicación de las reglas.** Es necesario disponer de un algoritmo que controle la aplicación de las reglas para que, a pesar de seleccionarse aleatoriamente, cumplan el objetivo de ir optimizando la función; evitándose que la solución quede atrapada en mínimos locales. El algoritmo que se ha elegido es un proceso de Enfriamiento Simulado modificado, tal y como se detalla en 4.7.

En los apartados siguientes se describen y analizan en profundidad cada uno de estos tres elementos integrantes de RRMIN2.

4.5. EXPRESIÓN INICIAL AND-EXOR: FORMA ÓPTIMA MPRM

La expresión inicial que se utiliza en el procedimiento es la forma canónica MPRM (Apartado 2.4.1.2.2 [GRE91]) con menor número de productos. La elección de esta expresión inicial se debe a dos razones:

- En primer lugar, es posible utilizar algoritmos rápidos para su cálculo (Apartado 4.5.2), de manera que se puede obtener la forma MPRM deseada con un costo de

tiempo de CPU muy reducido.

- En segundo lugar, se está eligiendo la mejor expresión de un subconjunto de 3^n expresiones AND-EXOR de la función, con lo que se realiza un proceso de minimización previo a la aplicación de las reglas.

En los apartados siguientes se describe un procedimiento para obtener la forma mínima MPRM.

4.5.1. Minimización dentro de la familia de formas canónicas MPRM

En el Apartado 2.4.2 se han estudiado las formas canónicas KRM, que pueden obtenerse a partir de la forma canónica RM mediante una matriz de transformación apropiada. En 2.3 se estudió cómo conseguir la expresión RM de una función booleana, con lo que se tienen todas las herramientas necesarias para obtener cualquiera de las formas canónicas FPRM (utilizando la matriz (2.33)) ó MPRM (utilizando (2.34)).

El procedimiento para obtener la mejor forma MRPM consistiría pues, en calcular las 3^n formas canónicas y elegir aquella que tenga un menor número de productos. Teniendo en cuenta que la matriz de la transformación $K_{(m)}$ tiene 2^nx2^n componentes, para calcular cada una de las formas canónicas serían precisas, en principio, 4^n-2^n sumas. Si esto se multiplica por las 3^n formas canónicas que hay que calcular, se obtiene un total de:

$$s_t=3^n\cdot(4^n-2^n) \quad sumas \tag{4.12}$$

Por ejemplo, para un número de variables n=8, es preciso realizar más de 428 millones de sumas. Todo ello sin contar las operaciones que hay que realizar para calcular la forma RM, construir las matrices de transformación, obtener el número de productos, buscar el número mínimo, etc.

En cuanto a los requerimientos de memoria, será preciso tener simultamente la matriz de transformación, el vector M correspondiente a la función booleana, el vector transformado y los 3^n números de productos entre los que hay que elegir. Esto da un total de:

$$m_t=4^n+3^n+2^{n+1} \quad números \tag{4.13}$$

Para n=8 variables es preciso mantener en memoria un total de 72609 números.

Como puede verse, el procedimiento resulta laborioso y lento, aunque no tiene unos requerimientos de memoria excesivos para valores de n moderadamente grandes (no obstante, aumentan como 4^n). Como consecuencia, surge la necesidad de mejorar el procedimiento para conseguir una mayor rapidez y un menor gasto de memoria.

Una primera mejora puede introducirse teniendo en cuenta que para conocer el número de productos que tiene una forma canónica MPRM no es preciso hallar su expresión, sino que bastaría con utilizar los conceptos de vector extendido y vector peso introducidos en 2.4.3. Con las citadas definiciones, para encontrar el número mínimo de produtos necesario

únicamente sería preciso calcular el vector extendido E (2.43) y a partir de él calcular el vector peso S (2.45). Una vez calculado S, sólo hay que buscar su componente más reducida, hallando automáticamente la polaridad (y por tanto la forma canónica MPRM) que proporciona el mínimo número de productos. Una vez conocida la polaridad mínima, habría que realizar un producto por la matriz de transformación correspondiente para calcular la forma MPRM mínima.

De esta forma, únicamente es preciso calcular tres productos de matrices, uno con la matriz P_n (2.44), uno con la matriz C_n (2.46) y otro con la matriz $K_{(m)}$, siendo m la polaridad mínima. Teniendo en cuenta que P_n contiene 3^nx2^n componentes, que C_n tiene 3^nx3^n, y que $K_{(m)}$ consta de 2^nx2^n, se tiene que el total de sumas a realizar sería:

$$s_t=3^n\cdot(3^n+2^n-2)+(4^n-2^n) \quad sumas \tag{4.14}$$

Para n=8, es necesario realizar algo menos de 45 millones de sumas, es decir, unas 10 veces menos que para el mismo número de variables usando (4.12). Por tanto, se ha conseguido una reducción sustancial del tiempo necesario para la minimización al introducir los vectores extendidos. En cuanto a los requerimientos de memoria, es preciso almacenar:

$$m_t=3^n\cdot(3^n+2)+2^n \quad números \tag{4.15}$$

Aún así, el número de operaciones a realizar resulta excesivo y es necesario dar un segundo paso en la mejora del procedimiento. Esta mejora va a consistir en utilizar algoritmos rápidos para el cálculo de los vectores extendidos y la transformación final (véase la Tabla 4.1 en 4.5.2.5 para una comparación entre los distintos métodos de cálculo de la MPRM óptima). En el siguiente apartado se explica cómo construir los grafos para la implementación de estos algoritmos rápidos.

4.5.2. Algoritmos rápidos para cálculo con matrices de Kronecker

A continuación se va a estudiar un procedimiento rápido para el cálculo de productos en los que se encuentren involucradas matrices de Kronecker, basado en el uso de grafos. En primer lugar se hará un estudio general (Apartado 4.5.2.1) y después se particularizará para GF(2) (Apartado 4.5.2.2). Finalmente se construirán los grafos precisos para el cálculo del vector extendido (Apartado 4.5.2.3), el vector peso (Apartado 4.5.2.4), y la forma MPRM correspondiente a la polaridad mínima (Apartado 4.5.2.5).

4.5.2.1. Generación de grafos para matrices de Kronecker

En este apartado se va a describir la generación de grafos para el cálculo de productos con matrices producidas a partir de productos de Kronecker. Asímismo se demostrará que el número de operaciones a realizar para el cálculo utilizando esos grafos es menor que el que se precisaría utilizando el producto de matrices usual.

Considérese una matriz K_1 de dimension 2x2:

$$K_1=\begin{bmatrix} K_1^{11} & K_1^{12} \\ K_1^{21} & K_1^{22} \end{bmatrix} \tag{4.16}$$

El producto de esta matriz con un vector V de dos componentes dará lugar a un vector P también de dos componentes:

$$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix}=\begin{bmatrix} k_1^{11} & k_1^{12} \\ k_1^{21} & k_1^{22} \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \tag{4.17}$$

Este producto de matrices puede representarse gráficamente de la siguiente forma:

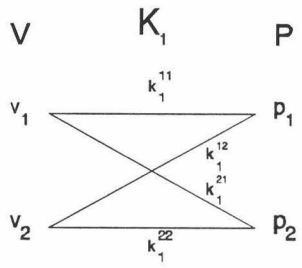


Fig. 4.1. Grafo $P=K_1V$

Un caso particular se detalla a continuación; considérese la siguiente matriz definida sobre GF(2):

$$K_1=\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{4.18}$$

El grafo que de esta matriz sería el de la Figura 4.2. En dicha figura puede verse que en lugar de poner en cada línea la componente correspondiente de K_1, lo que se ha hecho, aprovechando que se está trabajando en GF(2), es unir los nodos con una línea en el caso de que la componente sea un 1, y dejar los nodos sin unir en el caso de tener un 0.

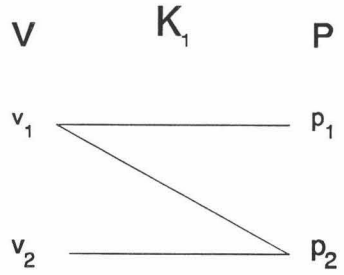


Fig. 4.2. Grafo producto $P=K_1V$ (Caso particular)

Si se desea construir el grafo correspondiente a una matriz 3x3, el procedimiento sería completamente análogo, de forma que si se tiene por ejemplo la matriz:

$$K_2=\begin{bmatrix}1 & 0 & 1\\ 1 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \tag{4.19}$$

el grafo correspondiente sería el representado en la Figura 4.3.

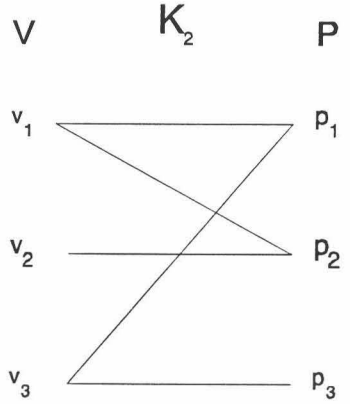


Fig. 4.3. Grafo $P=K_2V$ (Caso particular)

Considérese ahora una matriz K_1 con M_1 filas y N_1 columnas, y una matriz K_2,M_2xN_2. Si se efectúa el producto de Kronecker [GRE86] de estas dos matrices,

$$K=K_2*K_1 \tag{4.20}$$

se obtendrá una matriz K, de dimensiones $M_1\cdot M_2$x$N_1\cdot N_2$, que vendrá dada por:

$$K=\begin{bmatrix} k_2^{11}\cdot K_1 & k_2^{12}\cdot K_1 & \dots & k_2^{1N_2}\cdot K_1\\ k_2^{21}\cdot K_1 & k_2^{22}\cdot K_1 & \dots & k_2^{2N_2}\cdot K_1\\ \dots & \dots & \dots & \dots\\ k_2^{M_21}\cdot K_1 & k_2^{M_22}\cdot K_1 & \dots & k_2^{M_2N_2}\cdot K_1\end{bmatrix} \tag{4.21}$$

Por definición del producto de Kronecker, se puede considerar que la matriz K tiene por componentes a la matriz K_1, y dichas componentes están colocadas según la estructura marcada por K_2. El grafo para calcular el producto $P=K\cdot V$ donde V es un vector de $N_1\cdot N_2$ componentes y P de $M_1\cdot M_2$, tendrá $N_1\cdot N_2$ nodos iniciales y $M_1\cdot M_2$ nodos finales y los pasos que habrá que realizar para llegar a su obtención pueden verse en el Algoritmo 4.3.

Algoritmo 4.3: Generación de grafos.

a) *Escribir el grafo correspondiente a K_1.*
b) *Repetir el grafo obtenido en a) tantas veces como columnas tenga K_2.*
c) *Considerando cada uno de los grafos de K_1 como un único nodo, conectar los nodos según indiquen las componentes de K_2.*
d) *Volver a considerar los nodos individuales y desdoblar las líneas generadas en b).*

Si se tiene una matriz K_1 de 2x2 y una matriz K_2 de 3x3, los grafos que se irían obteniendo al realizar los pasos anteriores para calcular el producto de un vector V de 6 componentes por la matriz K pueden verse en las figuras 4.4 a 4.6 (Nota: en la Figura 4.6 no se han colocado los coeficientes multiplicativos por claridad en el dibujo).

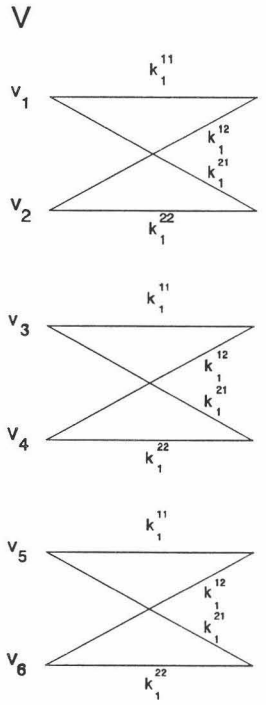


Fig. 4.4. Grafo correspondiente a b). Repetición de K_1.

Obsérvese que en los grafos que se han construido, el número de sumas a realizar viene dado por el número de veces que se produzca la confluencia de dos líneas en un nodo. Las sumas se consideran siempre entre dos componentes; si en un nodo confluyen tres líneas habrá que hacer dos sumas. En general, el número de sumas que se producen en un nodo viene dado por:

$$\textit{número sumas} = (\textit{número de líneas que llegan al nodo}) - 1 \qquad (4.22)$$

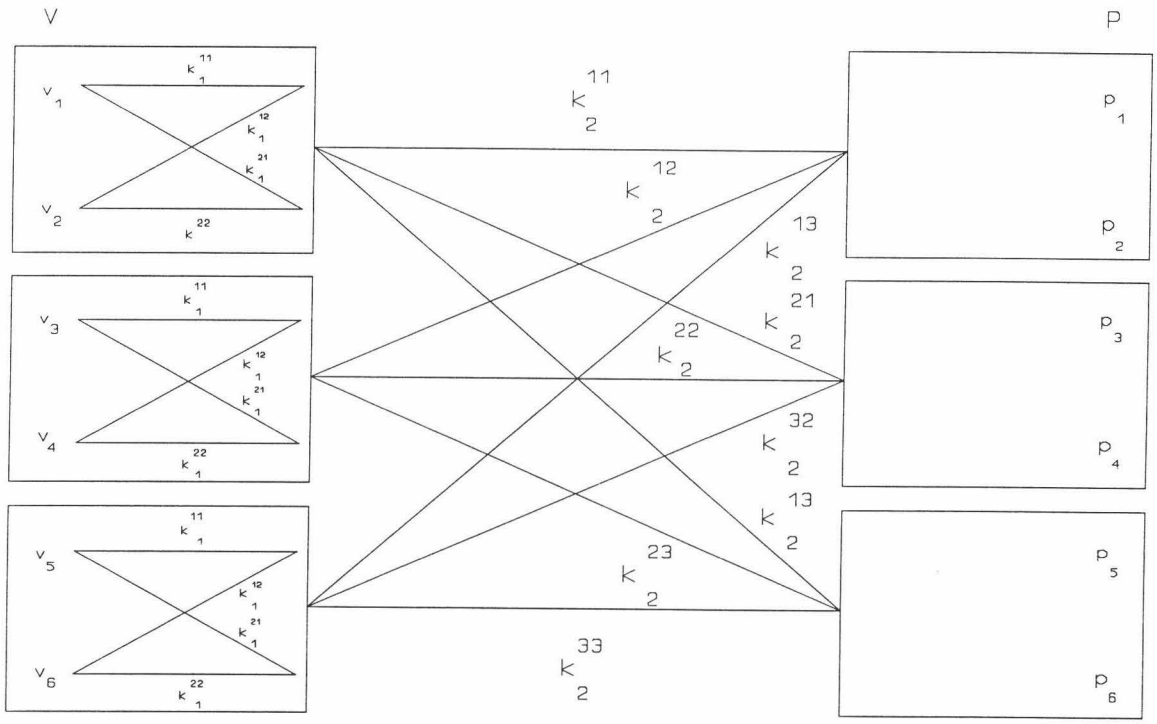


Fig. 4.5. Grafo correspondiente a c).

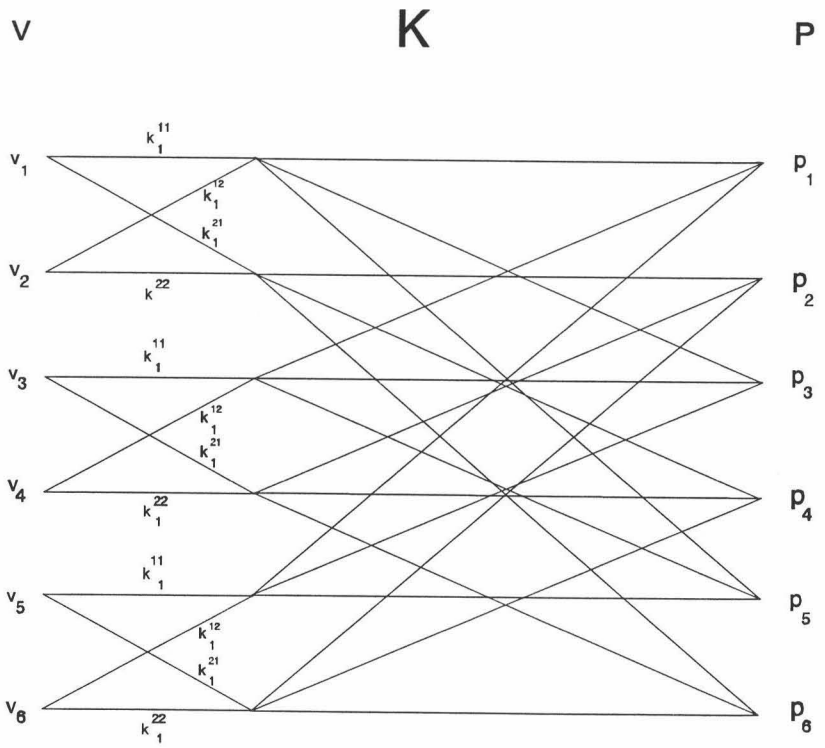


Fig. 4.6. Gráfico correspondiente a d). Desdoblamiento de líneas.

A continuación se demuestran una serie de proposiciones y teoremas que dan el número de sumas que es preciso realizar cuando se efectúa un producto mediante grafos.

Proposición 4.9: *Si se tiene una matriz K_1 con M_1 filas y N_1 columnas, el número de sumas necesarias para realizar el producto $P=K_1 \cdot V$ aplicando el producto de matrices usual viene dado por:*

$$s_{p1}=(N_1-1)M_1 \tag{4.23}$$

Demostración: *Inmediata utilizando la definición de producto de matrices.*

Proposición 4.10: *Si se tiene una matriz K_1 con M_1 filas y N_1 columnas, el número de sumas necesarias para realizar el producto $P=K_1 \cdot V$ usando grafos es:*

$$s_{g_1}=(N_1-1)\cdot M_1 \tag{4.24}$$

Demostración: *A cada nodo final llegan líneas procedentes de todos los nodos iniciales. Por tanto, aplicando (4.22), en cada nodo final se producen N_1-1 sumas. Como hay un total de M_1 nodos finales, el total de sumas es el que aparece en la Ecuación (4.24).*

Proposición 4.11: *Sea K una matriz generada por un producto de Kronecker como el de (4.20). En tal caso, el número de sumas que habrá que realizar para efectuar el producto $P=K \cdot V$ utilizando el producto usual de matrices es:*

$$s_p=(N_1 \cdot N_2-1)\cdot(M_1 \cdot M_2) \tag{4.25}$$

Demostración: *La matriz K tendrá $N_1 \cdot N_2$ columnas y $M_1 \cdot M_2$ filas. Aplicando la Proposición 4.9 se obtiene la Ecuación (4.25) directamente.*

Teorema 4.12: *Sea K una matriz generada por un producto de Kronecker como el de la Ecuación (4.20). En tal caso, el número de sumas que habrá que realizar para efectuar el producto $P=K \cdot V$ utilizando grafos es:*

$$s_g=N_2 \cdot s_{g1}+(N_2-1)\cdot M_2 \cdot M_1 \tag{4.26}$$

donde s_{g1} es el número de sumas necesarias para efectuar el producto por K_1. En particular, si K_1 no está generada por el producto de Kronecker de otras matrices, se tiene:

$$s_g=N_2 \cdot (N_1-1)\cdot M_1+(N_2-1)\cdot M_2 \cdot M_1 \tag{4.27}$$

Demostración: *Se va a seguir el Algoritmo 4.3 para la generación de grafos.*

a) Al generar el grafo de K_1 se producirán s_{g1} sumas. En particular si K_1 no es producto de Kronecker de otras matrices, se producirán un total de $M_1 \cdot (N_1-1)$, según la Proposición 4.10.

b) Si ahora se repite N_2 veces el grafo correspondiente a K_1, se tendrá un total de $s_{g1} \cdot N_2$ sumas.

c) Al realizar este paso se generan un total de $(N_2-1) \cdot M_2$ sumas sin más que aplicar la Proposición 4.10.

d) Al desdoblar las líneas producidas en c), habrá que multiplicar el número de sumas producido en c) por M_1.

Finalmente, el número total de sumas se obtendrá sumando los números obtenidos en b) y d), resultando el valor que aparece en (4.26).

Proposición 4.13: *Sean a,b dos números naturales. Entonces se cumple:*

$$a \cdot b \geq a+b-1 \tag{4.28}$$

Si además se tiene que a>1 y b>1, se cumple:

$$a \cdot b > a+b-1 \tag{4.29}$$

Demostración: *Si a=1, la Ecuación (4.28) queda b≥b. Si b=1, (4.28) queda a≥a. A continuación va a probarse que si a>1 y b>1 se cumple (4.29), con lo que quedará probado además (4.28). Restando b a los dos miembros de (4.29) queda:*

$$ab > a+b-1 \Leftrightarrow ab-b > a+b-b-1 \Leftrightarrow b \cdot (a-1) > a-1 \Leftrightarrow b > 1 \tag{4.30}$$

El último paso se cumple por ser a>1, y la última expresión es cierta por haberla impuesto como condición. Por tanto, (4.29) es cierto y (4.28) también lo es.

Teorema 4.14: *Sean N_1, N_2, M_1, M_2 números enteros. Entonces se verifica:*

$$(N_1 \cdot N_2 - 1) \cdot M_1 \cdot M_2 \geq (N_1 - 1) \cdot M_1 \cdot N_2 + (N_2 - 1) \cdot M_2 \cdot M_1 \tag{4.31}$$

Si además $N_1>1$ y $M_2>1$ entonces:

$$(N_1 \cdot N_2 - 1) M_1 \cdot M_2 > (N_1 - 1) \cdot M_1 \cdot N_2 + (N_2 - 1) \cdot M_2 \cdot M_1 \tag{4.32}$$

Demostración: *Si se desarrollan los paréntesis de (4.31), se obtiene:*

$$N_1 \cdot N_2 \cdot M_1 \cdot M_2 - M_1 M_2 \geq N_1 \cdot M_1 \cdot N_2 - M_1 \cdot N_2 + N_2 \cdot M_2 \cdot M_1 - M_2 \cdot M_1 \tag{4.33}$$

Eliminando el término M_1M_2, y sacando factor común M_1N_2 queda:

$$M_1 \cdot N_2 \cdot N_1 \cdot M_2 \geq M_1 \cdot N_2 \cdot (N_1 + M_2 - 1) \tag{4.34}$$

Si ahora se divide en ambos miembros por M_1N_2 (puede hacerse sin ningún problema porque se está trabajando con números naturales) se obtiene la siguiente expresión:

$$N_1 \cdot M_2 \geq N_1 + M_2 - 1 \tag{4.35}$$

Finalmente, aplicando la Proposición 4.13 a la expresión (4.35) queda probado el Teorema.

Teorema 4.15: *Sea $K^{(n)}$ una matriz generada por el siguiente producto de Kronecker de n matrices:*

$$K^{(n)} = K_n * [K_{n-1} * [\ldots K_3 * [K_2 * K_1]\,]\,] \tag{4.36}$$

donde la matriz K_i tiene M_i filas y N_i columnas.

Entonces, el número total de sumas precisas para efectuar el producto $P=K^{(n)}V$, donde P es un vector de $M_1 \cdot M_2 \cdot \ldots M_n$ componentes y V tiene $N_1 \cdot N_2 \cdot \ldots N_n$ componentes, viene dado por:

$$s_{gn} = N_n s_{gn-1} + (N_n - 1) \cdot \prod_{i=1}^{n} M_i \tag{4.37}$$

donde s_{gn-1} es el número de sumas necesario para efectuar el producto por la matriz:

$$K^{n-1} = K_{n-1} * [K_{n-2} * [\ldots K_3 * [K_2 * K_1]\,]\,] \tag{4.38}$$

Nótese que s_{gn-1} puede hallarse utilizando iterativamente la Ecuación (4.37).

Demostración: *Basta aplicar el Teorema 4.14 realizando las sustituciones:*

$$K_1 = K^{n-1} \qquad K_2 = K_n$$

Por tanto, en este caso K_1 es una matriz de dimensiones $M_{n-1} \cdot M_{n-2} \ldots M_1 x N_{n-1} \cdot \ldots \cdot N_1$ y K_2 una matriz $M_n x N_n$. Aplicando a estas matrices la expresión (4.26), se obtiene directamente (4.37).

Teorema 4.16: *Sea s_{pn} el número de sumas necesarias para realizar el producto $P=K^{(n)} \cdot V$ utilizando producto de matrices y s_{gn} el número de sumas preciso utilizando grafos. Entonces se cumple la siguiente desigualdad:*

$$s_{pn} \geq s_{gn} \tag{4.39}$$

Demostración: *La demostración se realizará aplicando inducción. Para n=2 la desigualdad está probada por el Teorema 4.14. Por tanto, para probar el Teorema únicamente es necesario probar que $s_{pn+1} \geq s_{gn+1}$ suponiendo cierto que $s_{pn} \geq s_{gn}$.*

Según la Proposición 4.11:

$$s_{pn+1} = (N_1 \cdot N_2 \cdot \ldots \cdot N_n \cdot N_{n+1} - 1) \cdot M_1 \cdot M_2 \ldots M_n \cdot M_{n+1} \tag{4.40}$$

Por otra parte, según el Teorema 4.15:

$$s_{gn+1}=s_{gn}N_{n+1}+(N_{n+1}-1)\cdot\prod_{i=1}^{n+1}M_i \tag{4.41}$$

Operando en (4.40):

$$s_{pn+1}=N_{n+1}\cdot M_{n+1}\cdot(N_1\cdot N_2\cdot\ldots\cdot N_n\cdot M_1\cdot M_2\cdot\ldots\cdot M_n)-M_1\cdot M_2\cdot\ldots\cdot M_{n+1}= \tag{4.42}$$

$$=N_{n+1}\cdot M_{n+1}\cdot(N_1\cdot N_2\cdot\ldots\cdot N_n\cdot M_1\cdot M_2\cdot\ldots\cdot M_n-M_1\cdot M_2\cdot\ldots\cdot M_n+M_1\cdot M_2\cdot\ldots\cdot M_n)-M_1\cdot M_2\cdot\ldots\cdot M_{n+1} \tag{4.43}$$

$$=N_{n+1}M_{n+1}\cdot(s_{pn})+M_{n+1}\cdot N_{n+1}\cdot M_1\cdot M_2\cdot\ldots\cdot M_n-M_1\cdot M_2\cdot\ldots\cdot M_{n+1} \tag{4.44}$$

Así, se obtiene:

$$s_{pn+1}=s_{pn}\cdot M_{n+1}\cdot N_{n+1}+(N_{n+1}-1)\cdot\prod_{i=1}^{n}M_i \tag{4.45}$$

Comparando esta última expresión con (4.41), resulta evidente que $s_{pn+1}\geq s_{gn+1}$, si se cumple que $s_{pn}\geq s_{gn}$. Además, si $M_n>1$, se tiene $s_{pn}>s_{gn}$.

La consecuencia inmediata que se extrae del Teorema 4.16 es que el cálculo de un producto del tipo $P=K\cdot V$ donde K es una matriz de Kronecker, precisa de un menor número de operaciones realizándolo por medio de grafos que efectuando el producto usual de matrices (en el ejemplo de la Figura 4.6, son necesarios 18 productos utilizando grafos en lugar de los 30 que habría que efectuar si se utilizara producto de matrices). Este ahorro en el número de sumas se debe a la estructura repetitiva del producto de Kronecker, que permite aprovechar un determinado cálculo para la realización de otros muchos.

4.5.2.2. Generación de grafos sobre GF(2)

En este apartado se va a particularizar el estudio realizado en el Apartado 4.5.2.1 para el caso de matrices definidas sobre GF(2).

Si se tiene una matriz de Kronecker K definida sobre GF(2), sus componentes serán ceros y unos. Por tanto, los coeficientes multiplicativos que aparecen en los grafos no serán necesarios y bastará con dibujar una línea si la componente es un 1 o no dibujar nada si la componente es un 0. Teniendo en cuenta esto, en los grafos se evita el realizar una suma si aparecen componentes cero, con lo que se consigue un ahorro adicional al visto en el Teorema 4.14. Por lo demás, el proceso de generación de los grafos sería exactamente el mismo. Por ejemplo, si se utilizan las matrices de las ecuaciones (4.18) y (4.19) y se realiza con ellas el producto de Kronecker representado en la Ecuación (4.20), el cálculo del grafo, paso a paso, puede verse en las figuras 4.7 a 4.9.

El siguiente teorema proporciona el número de sumas necesario para efectuar el producto de un vector por una matriz de Kronecker sobre GF(2):

Teorema 4.17: *Sea $K^{(n)}$ una matriz generada por el producto de Kronecker de n matrices de la misma forma que en (4.36). Si todas las matrices están definidas sobre GF(2) y U_i es el número de unos de la matriz K_i, entonces:*

$$s_{gn}=N_n \cdot s_{gn-1}+(U_n-M_n)\cdot\prod_{i=1}^{n-1} M_i \tag{4.46}$$

Demostración: *Sólo es preciso efectuar una suma si se tienen unos, por tanto, a la hora de contabilizar las sumas hay que tener en cuenta únicamente los unos que hay en cada columna. Así, en lugar de utilizar el número total de columnas N_n, restarle 1 y multiplicar por el número de filas M_n, se debe usar el número de unos que hay en cada fila, restar 1 y sumar individualmente para cada fila. Finalmente, realizando la suma y efectuando operaciones se obtiene la expresión (4.46) a partir de lo que se obtuvo en (4.26).*

Si se aplica la expresión (4.46) al ejemplo representado en las figuras 4.7 a 4.9, se obtiene un total de s_g=7 sumas (tal y como puede observarse en la Figura 4.9). En cambio, si se realiza un producto usual de matrices, se obtiene s_p=30 sumas, con lo que se consigue un gran ahorro de operaciones si se utiliza el algoritmo rápido basado en grafos. En los apartados siguientes se van a estudiar los casos particulares de los grafos correspondientes al vector extendido E y peso S (Apartado 2.4.1.2.3).

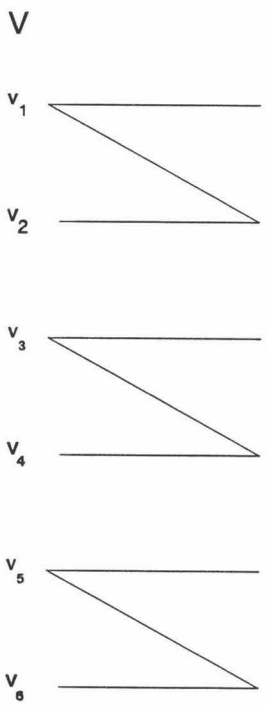


Fig. 4.7. Repeticiones grafo K_1.

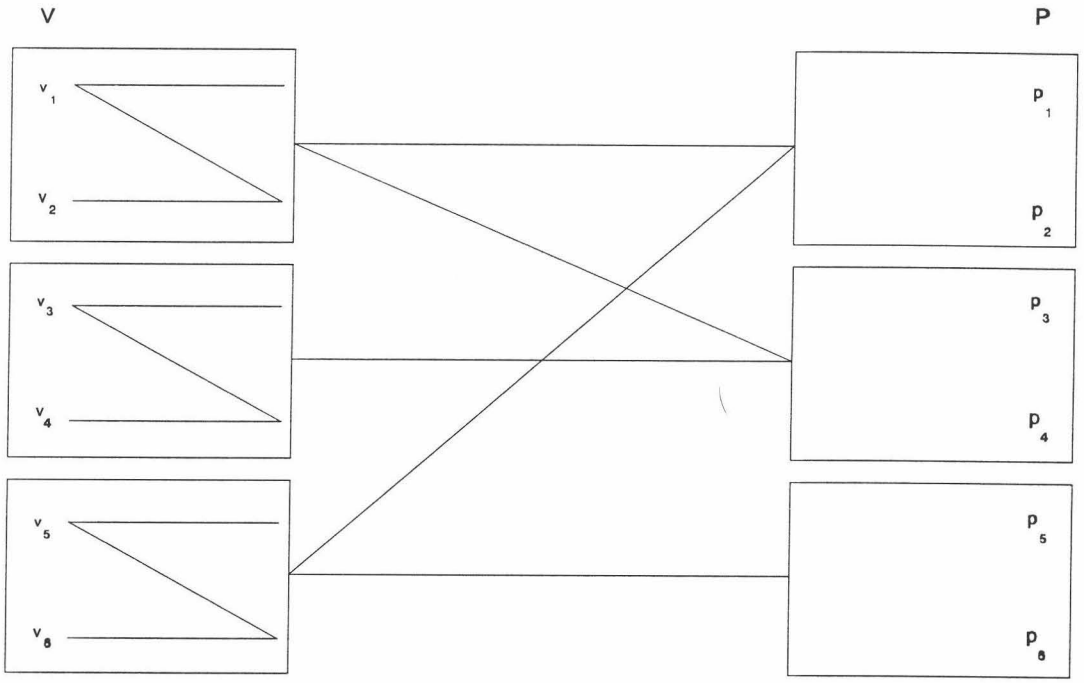


Fig. 4.8. Paso c) del ejemplo.

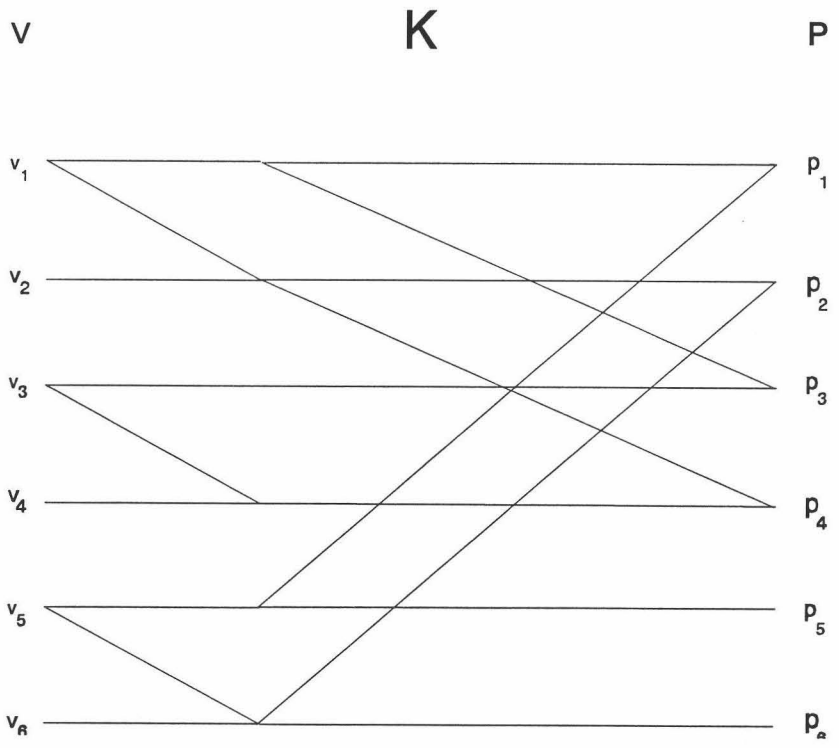


Fig. 4.9. Desdoblamiento de líneas. Resultado final.

4.5.2.3. Generación del grafo para el cálculo del vector extendido.

A continuación se va a obtener el grafo para el cálculo del vector extendido E definido en 2.4.1.2.3, se calculará el número de sumas necesario para tal operación y finalmente se estimarán los requerimientos de memoria precisos.

Si M es el vector verdad de una función booleana f de n variables, el vector extendido E correspondiente a f puede calcularse según la expresión (2.43). La matriz P_n se obtiene por productos de Kronecker sucesivos, tal y como se muestra en (2.44), siendo por tanto aplicable todo lo visto en el Apartado 4.5.2.2.

En primer lugar se va a construir el grafo correspondiente a la matriz P_2, que vendrá dada por:

$$P_2 = P_1 * P_1 \tag{4.47}$$

donde P_1 es:

$$P_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \tag{4.48}$$

En tal caso, el cálculo de (2.44), vendría dado por el grafo representado en la Figura 4.10.

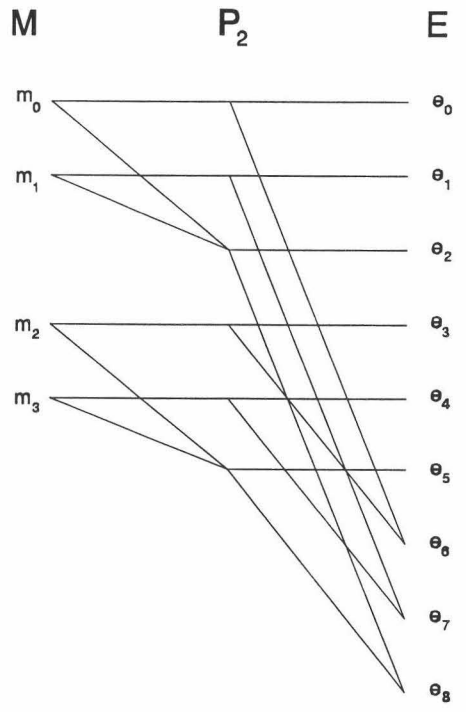


Fig. 4.10. Grafo para el cálculo de E con n=2.

Si se desea construir el grafo correspondiente a P_3, basta tener en cuenta que:

$$P_3=P_1*P_2 \tag{4.49}$$

Según esta última expresión, el grafo de P_3 se construirá repitiendo el de P_2, y conectando según indican las componentes de P_1. El resultado puede verse en la Figura 4.11.

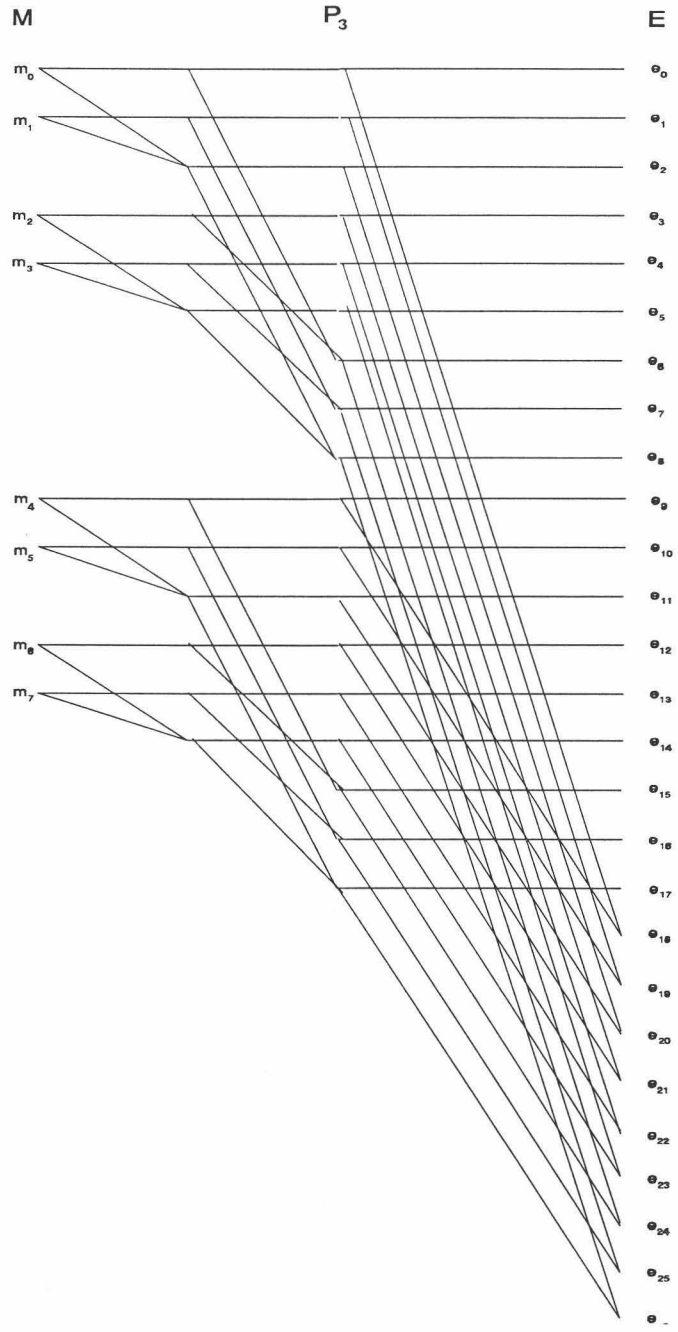


Fig. 4.11. Grafo para el cálculo de E con n=3.

En general, P_n puede obtenerse repitiendo P_{n-1}, y conectando los nodos segúnP_1 (2.45), de manera que existe una forma sistemática de proceder para calcular el vector extendido *E*, y resulta por tanto, apropiado para su utilización en RRMIN2.

A continuación se da una serie de resultados referentes al número de sumas necesarias para realizar el cálculo de un vector extendido *E* correspondiente a una función de *n* variables.

Proposición 4.18: *El número de sumas necesario para calcular el vector extendido E de una función de conmutación f utilizando grafos viene dado por la expresión:*

$$s_{gn}=\sum_{i=0}^{n-1} 2^{n-1-i}\cdot 3^i \tag{4.50}$$

Demostración: *Si se aplica el Teorema 4.17 teniendo en cuenta que:*

$$N_n=2, \quad U_n=4, \quad M_n=3, \quad M_i=3 \tag{4.51}$$

se obtiene de forma inmediata lo siguiente:

$$s_{gn}=2\cdot s_{gn-1}+3^{n-1} \tag{4.52}$$

Si se desarrolla (4.52) se van obteniendo sucesivamente las siguientes expresiones:

$$\begin{aligned} s_{gn}&=2\cdot\left[2\cdot S_{gn-2}+3^{n-2}\right]+3^{n-1}\\ &=2\cdot\left[2\cdot\left[2\cdot s_{gn-3}+3^{n-3}\right]+3^{n-2}\right]+3^{n-1}\\ &=2\cdot\left[2\cdot\left[2\cdot\ldots 2\cdot\left[2\cdot s_{g1}+3^1\right]+3^2\right]+3^3+\ldots\right]+3^{n-1}\\ &=2^{n-1}\cdot s_{g_1}+2^{n-2}\cdot 3+2^{n-3}\cdot 3^2+\ldots 3^n-1 \end{aligned} \tag{4.53}$$

Teniendo en cuenta que $s_{g1}=1$, se obtiene finalmente (4.50).

Proposición 4.19: *Sea a un número entero. Entonces:*

$$(a+1)^n-a^n=\sum_{i=0}^{n-1} a^{n-1-i}(a+1)^i \tag{4.54}$$

Demostración: *La demostración se hace también por inducción, y puede consultarse en el Apéndice A.*

Corolario 4.20: *Dada una función booleana f de n variables, el número de sumas necesario para realizar el cálculo de su vector extendido E viene dado por:*

$$s_{gE}=3^n-2^n \tag{4.55}$$

Demostración: *La obtención de (4.55) es inmediata sin más que aplicar la Proposición 4.19*

a la expresión (4.50).

Teniendo en cuenta el último resultado puede verse que se obtiene un gran ahorro utilizando algoritmos rápidos. Por ejemplo, para el cálculo del vector extendido E de una función de 8 variables se necesitan 1673055 sumas utilizando producto de matrices, mientras que con el algoritmo rápido basado en grafos, únicamente hay que hacer 6305.

Finalmente, obsérvese que para el cálculo de E con grafos únicamente es necesario almacenar simultaneamente 3^n números, puesto que el algoritmo puede aplicarse "in place" (tal y como ocurre, por ejemplo, en la Transformada Rápida de Fourier [BRI88]).

4.5.2.4. Generación del grafo para el cálculo del vector peso.

En este apartado se va a mostrar cómo puede calcularse el vector peso S utilizando algoritmos rápidos. Partiendo del vector extendido E, el vector peso se obtiene según la expresión (2.46). La matriz C_n se calcula a partir de productos de Kronecker de la matriz C_1, según se muestra en (2.47). Al igual que se hizo al estudiar el vector extendido, se va a empezar construyendo el grafo correspondiente a la matriz C_2, que viene dada por:

$$C_2=C_1*C_1 \tag{4.56}$$

donde C_1 es:

$$C_1=\begin{bmatrix}1 & 0 & 1\\ 0 & 1 & 1\\ 1 & 1 & 0\end{bmatrix} \tag{4.57}$$

En tal caso, el cálculo del vector S vendría dado por el grafo de la Figura 4.12.

Para realizar el cálculo del vector S de una función de tres variables, únicamente hay que tener en cuenta que:

$$C_3=C_1*C_2 \tag{4.58}$$

y construir el grafo según se estudió en el Algoritmo 4.3, es decir repitiendo el grafo C_2 según indique C_1 y realizando el correspondiente desdoblamiento de líneas. El resultado puede verse en la Figura 4.13.

Al igual que en el caso del cálculo del vector extendido, el grafo de C_n puede calcularse de forma sistemática: se repetiría el grafo de C_{n-1} y se conectarían los nodos según indican las componentes de C_1. De nuevo, el procedimiento es sistemático y de fácil implementación en ordenador.

El siguiente teorema proporciona el número de sumas necesario para calcular el vector S de una función, supuesto que se conoce el vector E (cómo calcular el vector E y el número de sumas necesario para ello ya se estudiaron en el apartado anterior).

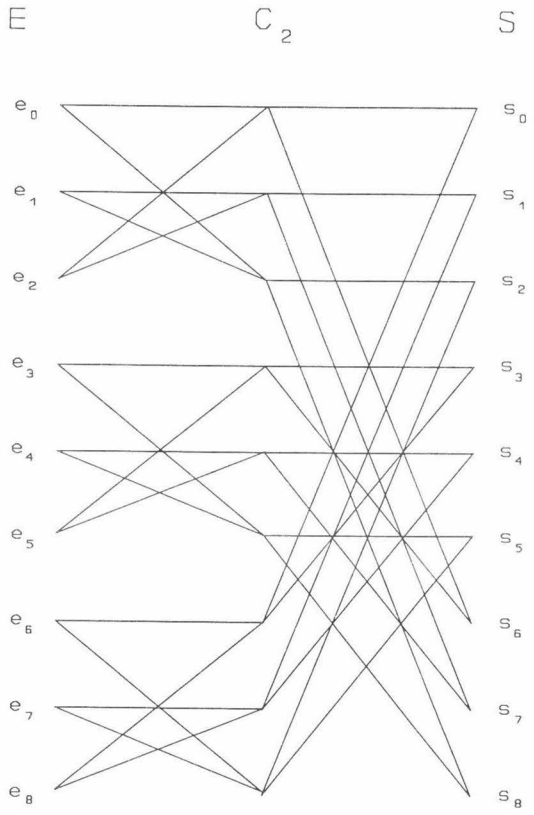


Fig. 4.12. Grafo correspondiente a C_2

Teorema 4.21: *Sea f una función booleana de n variables y E su vector extendido. Entonces, el número de sumas necesario para calcular el vector peso S a partir de E viene dado por:*

$$s_{gS}=n\cdot 3^n \tag{4.59}$$

Demostración: *Si se aplica el Teorema 4.17 con los valores:*

$$N_n=3,\ U_n=6,\ M_n=3,\ M_i=3 \tag{4.60}$$

se llega a la siguiente expresión:

$$s_{gn}=3\cdot s_{gn-1}+3^n \tag{4.61}$$

Si se desarrolla (4.61) se obtienen sucesivamente las expresiones siguientes:

$$\begin{aligned} s_{gn}&=3\cdot\left[3\cdot s_{gn-2}+3^{n-1}\right]+3^n \\ &=3\cdot\left[3\cdot\left[3\cdot s_{gn-3}+3^{n-2}\right]+3^{n-1}\right]+3^n \\ &=3^{n-1}\cdot s_{g1}+3\cdot 3^{n-1}+3^2\cdot 3^{n-2}+\ldots+3^n \end{aligned} \tag{4.62}$$

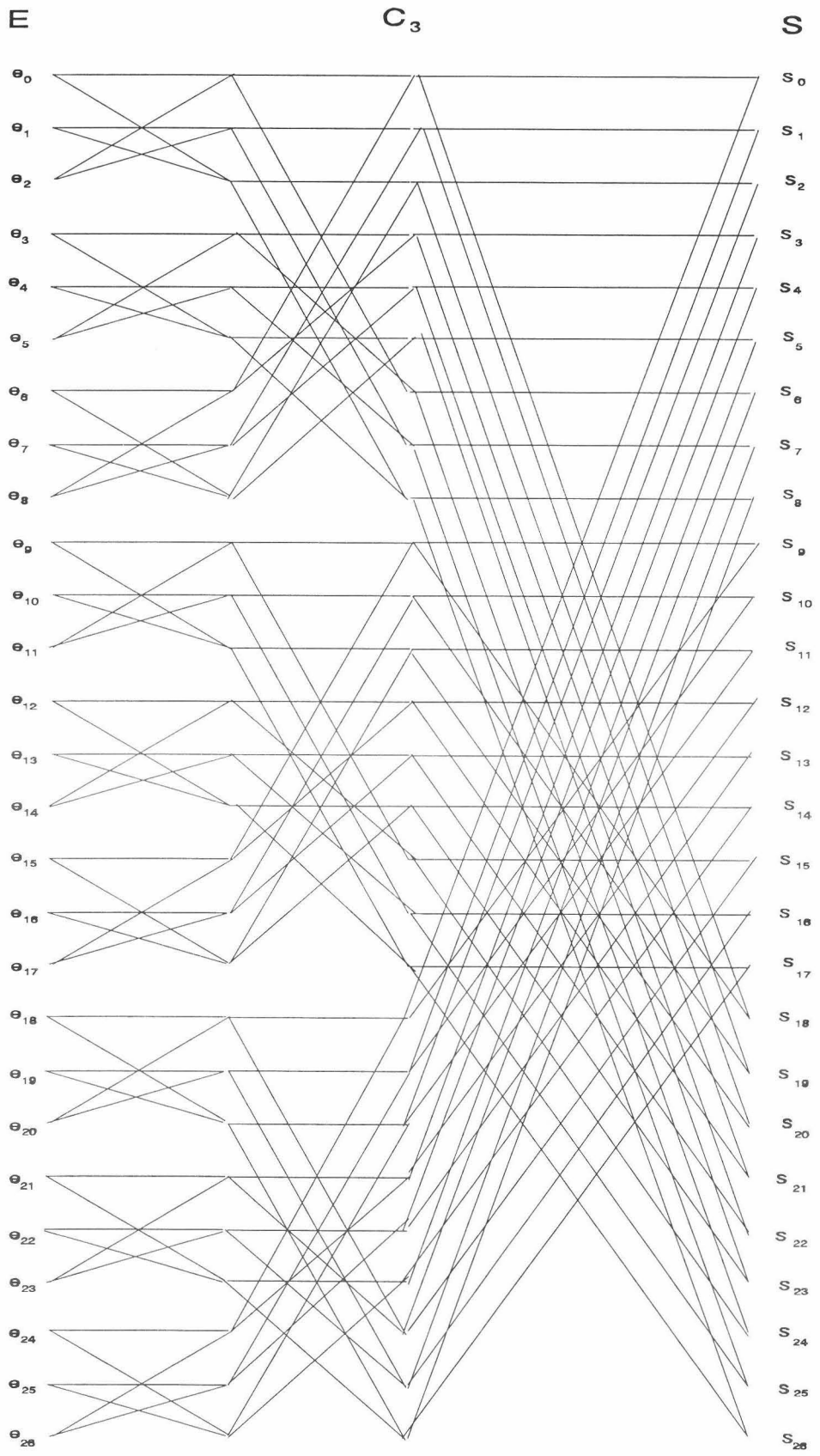


Fig. 4.13. Grafo de C_3

Finalmente, teniendo en cuenta que en este caso $s_{g1}=3$, *se obtiene:*

$$s_{gn}=n\cdot 3^n \tag{4.63}$$

En cuanto a los requerimientos de memoria, obsérvese que en este caso no basta con un sólo vector de 3^n componentes porque al generar un nuevo paso se produce un solapamiento de los valores; no ocurre como en el caso del cálculo de E, en el que al dar un nuevo paso se generaban nuevos nodos sin afectar a los valores que tenían los calculados con anterioridad. Como consecuencia de esto será preciso almacenar $2\cdot 3^n$ números.

4.5.2.5. Generación del grafo para la obtención un forma MPRM con polaridad *m*

Una vez determinada la polaridad mínima m, es preciso hallar la expresión de esa forma canónica MPRM. Esto se haría partiendo de la forma canónica RM, G, y aplicando la transformación $K_{(m)}$, según se detalla en (2.34). En 2.3.2 se describió un algoritmo rápido para el cálculo de la forma RM, por lo que únicamente queda por detallar el algoritmo rápido para hallar:

$$R=K_{(m)}\cdot G \tag{4.64}$$

La matriz $K_{(m)}$ es una matriz generada mediante el producto de Kronecker de las matrices (Véase el Apartado 2.4.1.2):

$$K_0=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\quad K_1=\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}\quad K_2=\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix} \tag{4.65}$$

Los grafos correspondientes a estas tres matrices, serán:

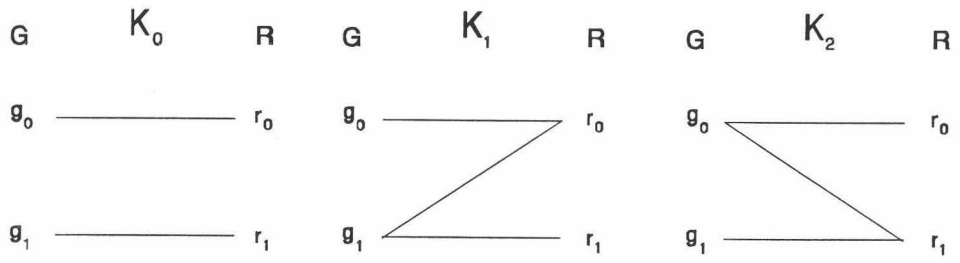


Fig. 4.14. Grafos de K_0, K_1, K_2

Por ejemplo, si se desea obtener la forma MPRM correspondiente a la polaridad 1 de una función de 3 variables, el grafo sería, teniendo en cuenta la Figura 4.14 y lo estudiado en el Apartado 4.5.2.2, el de la Figura 4.15.

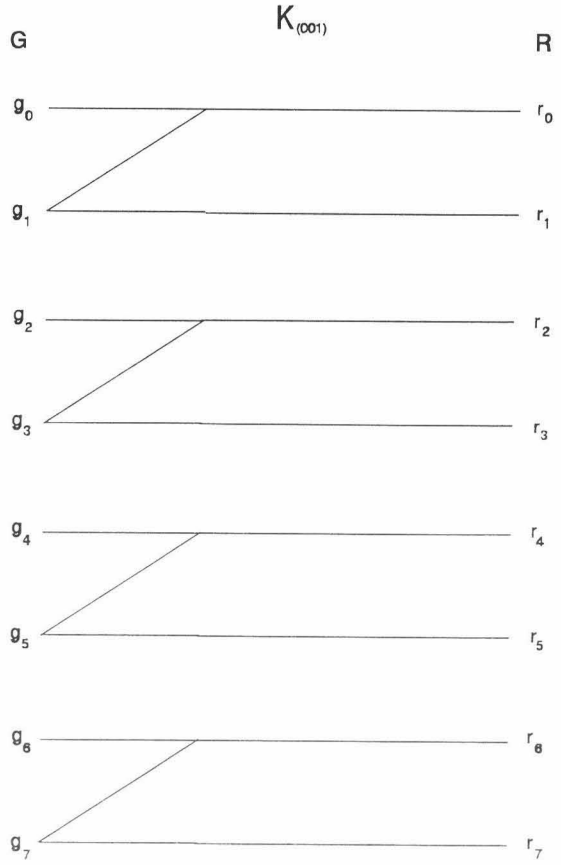


Fig. 4.15. Grafo de $K_{(001)}$

Nótese que en esta ocasión el grafo obtenido depende de la polaridad mínima (y por tanto de la función que se tenga que minimizar), por lo que es preciso implementar un algoritmo que sea capaz de generar el grafo correspondiente a la polaridad que se tenga en cada caso. Este algoritmo puede verse en pseudocódigo en el Apartado 4.5.3.3. Asimismo, debido a la razón expuesta, tampoco será posible determinar el número de sumas preciso para efectuar esta última transformación, aunque se puede acotar tal y como se muestra en la siguiente proposición:

Proposición 4.22: *Sea f una función booleana de n variables y G el vector correspondiente a su forma canónica RM. Entonces, el número de sumas necesario para calcular la forma MPRM de polaridad m a partir de G utilizando grafos está comprendida entre:*

$$0 \leq s_{gR} \leq n \cdot 2^{n-1} \tag{4.66}$$

Demostración: *En el mejor de los casos, es decir, cuando m=0, no es preciso realizar ninguna suma, directamente R=G, con lo que queda probado que* $0 \leq s_{gR}$. *Por otra parte, en el peor de los casos, si m es tal que* $K_{(m)}$ *está constituida únicamente por matrices de la forma*

K_1 y K_2, aplicando el Teorema 4.17 con los valores:

$$N_n=2,\ U_n=3,\ M_n=2,\ M_i=2 \tag{4.67}$$

se llega a la siguiente expresión:

$$s_{gn}=2\cdot s_{gn-1}+2^{n-1} \tag{4.68}$$

Si se desarrolla (4.68) se obtienen sucesivamente las expresiones siguientes:

$$\begin{aligned} s_{gn} &= 2\cdot\left[2\cdot s_{gn-2}+2^{n-2}\right]+2^{n-1} \\ &= 2\cdot\left[2\cdot\left[2\cdot s_{gn-3}+2^{n-3}\right]+2^{n-2}\right]+2^{n-1} \\ &= 2^{n-1}\cdot s_{g1}+2\cdot 2^{n-2}+2^2\cdot 2^{n-3}+\ldots+2^{n-1} \end{aligned} \tag{4.69}$$

Finalmente, teniendo en cuenta que en este caso $s_{g1}=1$, se obtiene:

$$s_{gn}=n\cdot 2^{n-1} \tag{4.70}$$

Por tanto, $s_{gR}\leq n\cdot 2^{n-1}$ como se quería probar.

Según los resultados presentados en estos tres últimos apartados, el número de sumas necesario para sintetizar una función booleana dentro de las MPRM utilizando algoritmos rápidos, vendría dado por:

$$s_t\leq(3^n-2^n)+n\cdot 3^n+n\cdot 2^n \tag{4.71}$$

En esta expresión se han considerado las sumas precisas para calcular el vector extendido, el vector peso, la forma RM, y la forma MPRM de polaridad mínima.

Para los requerimientos de memoria, basta tener en cuenta que hay que almacenar los 2^n números que constituyen M, $2\cdot 3^n$ números precisos para el cálculo de S, y $2\cdot 2^n$ para el cálculo de R. Las operaciones para obtener G pueden hacerse "in place" en el espacio de memoria de M, por lo que no se ha considerado. El cálculo de E es meramente temporal, y los 3^n números necesarios para ello están incluidos en la memoria reservada para el cálculo de S. Análogamente, para el cálculo de R puede utilizarse el espacio de memoria de M, y el que se utilizó para E y S. Por tanto, el total de números necesario es:

$$m_t=2^n+2\cdot 3^n \tag{4.72}$$

Si se comparan estas expresiones con las obtenidas en (4.14) y (4.15), se observa un gran ahorro tanto en el número de operaciones a realizar como en la cantidad de números que es preciso mantener simultaneamente en memoria. Por ejemplo, para n=8, según (4.71) harían falta como mucho 60851 sumas frente a los 45 millones que pronosticaba (4.14). En cuanto a la memoria, (4.15) exige almacenar unos 45 millones de números, (4.13) precisa 72609 números y (4.72) sólamente 13378. La Tabla 4.1 presenta una comparación entre los distintos procedimientos de cálculo estudiados para la obtención de la forma MPRM óptima en el caso

de funciones de n=8 variables.

	Producto usual de matrices y calculando todas las formas canónicas	Producto usual de matrices y utilizando los vectores E y S	Producto mediante algoritmos rápidos y utilización de los vectores E y S
Número de sumas a realizar	$428 \cdot 10^6$	$45 \cdot 10^6$	60581
Números a almacenar en memoria	72609	$43 \cdot 10^6$	13378

Tabla 4.1. Comparación entre los distintos procedimientos de cálculo de la MPRM óptima

En el Apartado siguiente se presenta de forma concreta la forma en que RRMIN2 obtiene la forma MPRM mínima como expresión inicial AND-EXOR.

4.5.3. Obtención de la forma MPRM mínima en RRMIN2

La obtención de la mejor forma MPRM la va a realizar RRMIN2 utilizando las herramientas estudiadas en los apartados anteriores (uso de los vectores extendido y peso y aplicación de algoritmos rápidos). Concretamente, este cálculo se efectúa en tres pasos:

1) Se parte de la representación de la función como suma de minterms, y se obtiene la polaridad de la forma MPRM con menor número de términos producto.

2) Una vez determinada esta polaridad mínima, se calcula la expresión de la forma canónica MPRM correspondiente.

3) La expresión obtenida en el paso 2 se traduce a una notación adecuada para facilitar la aplicación de las reglas de reescritura.

En los apartados que siguen se detalla el procedimiento seguido por RRMIN2 para la realización de estos pasos.

4.5.3.1. Obtención de la polaridad mínima

La función a minimizar es suministrada a RRMIN2 mediante un vector verdad M (Apartado 2.3.1) de 2^n componentes. A partir de este vector M, se calcula el vector extendido

E según se especifica en (2.43):

$$E=P_n \cdot M \tag{4.73}$$

Este producto se efectúa mediante el algoritmo rápido descrito en 4.5.2.3. Posteriormente, a partir de E se obtiene el vector peso S:

$$S=C_n \cdot E \tag{4.74}$$

tal y como se describe en (2.45). Nuevamente, es posible aplicar un algoritmo rápido y S puede calcularse en la forma indicada en 4.5.2.4.

El vector S contiene 3^n componentes correspondientes al número de productos de la forma MPRM con polaridad igual al número de componente. Calculando la menor componente de S se obtiene de forma inmediata la polaridad mínima. Con este procedimiento se consigue saber cuál es la polaridad de la forma MPRM con menor número de productos sin necesidad de calcular todas las formas canónicas.

Para ilustrar el procedimiento, considérese la siguiente función de 3 variables:

$$f(x_1,x_2,x_3)=\sum m(0,1,3,5,6) \tag{4.75}$$

El vector verdad correspondiente es:

$$M=[1\ 1\ 0\ 1\ 0\ 1\ 1\ 0] \tag{4.76}$$

El vector extendido E se obtiene aplicando el grafo de la Figura 4.12, resultando:

$$E=[1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1] \tag{4.77}$$

Si ahora se aplica el grafo correspondiente a C_3, se obtiene el vector peso S correspondiente:

$$S=[6\ 5\ 5\ 5\ 6\ 5\ 5\ 5\ 6\ 5\ 6\ 5\ 5\ 5\ 6\ 6\ 5\ 5\ 5\ 5\ 6\ 6\ 5\ 5\ 5\ 6\ 5] \tag{4.78}$$

Examinando este vector, se encuentra que el número mínimo de productos en el subconjunto de las formas canónicas MPRM es 5. La polaridad seleccionada será la 1. Nótese que si se hubiera cogido como expresión inicial la forma canónica RM como en RRMIN [PAR94] se hubieran obtenido 6 productos en lugar de 5 (se está realizando una minimización previa, tal y como se ha indicado antes). Una vez obtenida la polaridad mínima, es preciso obtener explícitamente la forma canónica MPRM con esa polaridad, tal y como se indica en el Apartado siguiente.

4.3.3.2. Cálculo de la forma MPRM mínima

Una vez que se conoce la polaridad de la forma mínima, sólamente hay que calcular

la forma canónica MPRM correspondiente a esa polaridad. Para ello se sigue el proceso descrito en el Apartado 2.3.1; se obtiene la forma canónica de RM mediante la transformación:

$$G=T_n \cdot M \tag{4.79}$$

utilizando el algoritmo rápido de 2.3.2. A continuación, si m es la polaridad mínima, la forma MPRM buscada viene dada por:

$$R=K_{(m)} \cdot G \tag{4.80}$$

Continuando el ejemplo del apartado anterior, será preciso obtener la forma MPRM correspondiente a la polaridad 1, y para ello habrá que aplicar la matriz de transformación $K_{(001)}$. Esta matriz vendrá dada por:

$$K_{(001)}=K_0 * K_0 * K_1 \tag{4.81}$$

El vector G correspondiente a (4.77) es, aplicando el grafo 2.3:

$$G=[1\ 0\ 1\ 1\ 1\ 1\ 0\ 1] \tag{4.82}$$

y el vector R, aplicando la matriz de (4.81) mediante el grafo de la Figura 4.15, vendrá dado por:

$$R=[1\ 0\ 0\ 1\ 0\ 1\ 1\ 1] \tag{4.83}$$

Como puede verse, la forma MPRM mínima contiene 5 productos, tal y como predecía el vector S.

4.5.3.3. Traducción al formato RRMIN2

Las 2^n componentes del vector R contienen una expresión de la función en polaridad $m=<m_n m_{n-1} \ldots m_1>$. Así, la componente i del vector R corresponde al producto i del vector:

$$B_{(m)}=B_{m_n} * B_{m_{n-1}} * \ldots * B_{m_1} \tag{4.84}$$

donde:

$$B_{m_i}=\begin{cases}[1,x_i] & si\ m_i=0\\ [1,\overline{x_i}] & si\ m_i=1\\ [\overline{x_i},x_i] & si\ m_i=2\end{cases} \tag{4.85}$$

Esta notación no resulta adecuada para la aplicación de reglas de reescritura, y es

preciso utilizar otra. El formato que se va a utilizar para las expresiones AND-EXOR en RRMIN2 consiste en un vector L de 3^n componentes, de manera que a un determinado término producto, le corresponde el número de componente $l(i)$, tal que, expresado en ternario, $l(i)=<l(i)_n l(i)_{n-1}...l(i)_1>$, se tiene:

$$\begin{aligned} &l(i)_j=0 \textit{ si la variable } j \textit{ no aparece en el producto} \\ &l(i)_j=1 \textit{ si la variable aparece } \text{sin } \textit{complementar} \\ &l(i)_j=2 \textit{ si la variable aparece complementada} \end{aligned} \tag{4.86}$$

La conversión de una notación a otra se efectúa según se describe en el Algoritmo 4.4, donde:

m=polaridad de la forma mínima MPRM.

$binit(i,j)$=función que devuelve el dígito binario j del número i.

Algoritmo 4.4: Conversión a formato RRMIN2.

```
Convertir(R,L,m)
        i=0;
        Mientras (i<2^n) hacer
                Si (R[i])
                        posicion=0;
                        j=i;
                        nvariable=0;
                        Mientras(nvariable<n)
                                Si (binit(j,nvariable))
                                        Si (binit(m,nvariable)=0)
                                                posicion=posicion+3^nvariable;
                                        Fin Si
                                        Si(binit(m,nvariable)=1)
                                                posicion=posicion+2*3^nvariable
                                        Fin Si
                                        Si (binit(m,nvariable)=2)
                                                posicion=posicion+3^nvariable
                                        Fin Si
                                Fin Si
                                Otro
                                        Si (binit(m,nvariable)=2)
                                                posicion=posicion+2*3^nvariable
                                        Fin Si
                                Fin Otro
                        nvariable=nvariable+1
                        Fin Mientras
                        L[posicion]=1;
                Fin Si
        i=i+1;
        Fin Mientras
Fin Convertir
```

Volviendo nuevamente al ejemplo, el vector $B_{(001)}$ vendría dado por:

$$\begin{aligned} B&=[1\ x_3]*[1\ x_2]*[1\ \overline{x_1}] \\ B&=[1\ \ \overline{x_1}\ \ x_2\ \ x_2\overline{x_1}\ \ x_3\ \ x_3\overline{x_1}\ \ x_3x_2\ \ x_3x_2\overline{x_1}] \end{aligned} \tag{4.87}$$

con lo que la expresión AND-EXOR correspondiente será:

$$f(x_1,x_2,x_3)=1\oplus x_2\overline{x_1}\oplus x_3\overline{x_1}\oplus x_3x_2\oplus x_3x_2\overline{x_1} \tag{4.88}$$

Observando el vector B, se comprueba que con la notación proporcionada por el vector R, no es posible aplicar reglas de reescritura ni obtener todas las formas AND-EXOR de la función. Así, será preciso pasar a la notación RRMIN2 y calcular el vector L según el Algoritmo 4.4. Para el ejemplo propuesto, el resultado obtenido sería:

$$L=[1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0] \tag{4.89}$$

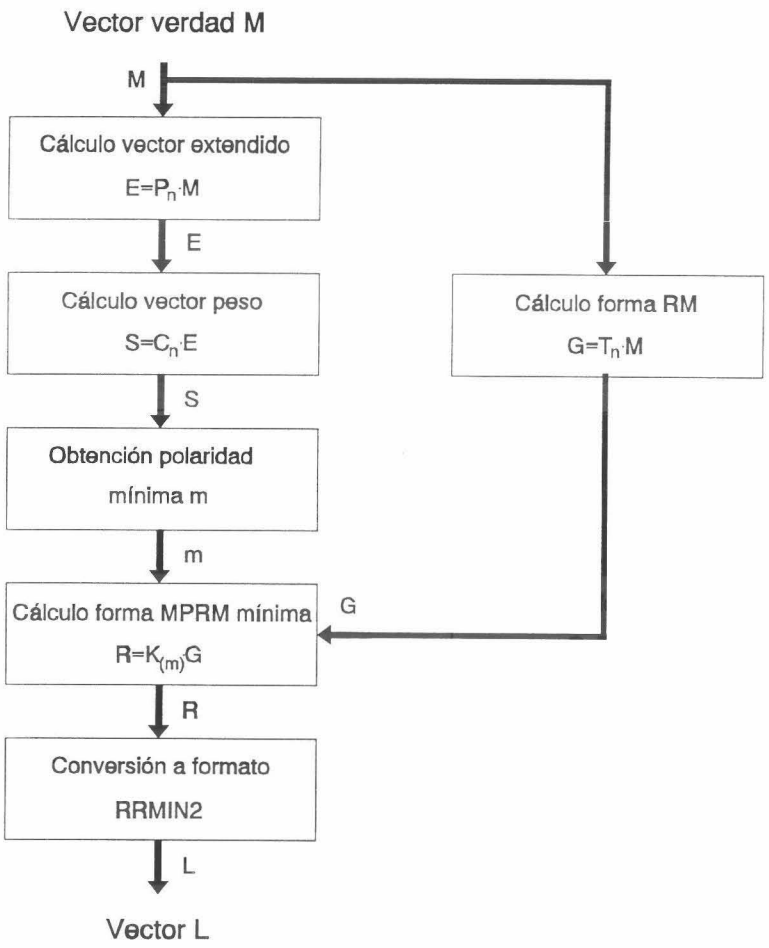


Fig. 4.16. Obtención de la expresión inicial AND-EXOR en RRMIN2

Teniendo en cuenta (4.86) se comprueba que la representación de la función es:

$$f(x_1,x_2,x_3)=1\oplus\overline{x}_1x_2\oplus\overline{x}_1x_3\oplus x_2x_3\oplus\overline{x}_1x_2x_3 \tag{4.90}$$

que coincide con (4.87). Con este último paso se tiene ya preparada la expresión inicial AND-EXOR sobre la que aplicar las reglas de reescritura.

En la Figura 4.16 puede verse el organigrama completo del proceso de obtención de la forma AND-EXOR inicial que se ha descrito en estos apartados. En los apartados que se presentan a continuación se efectúa una descripción detallada de las reglas de reescritura que actuarán sobre esta expresión inicial para ir simplificando la función.

4.6. REGLAS DE REESCRITURA

El procedimiento RRMIN2 se basa en la aplicación de reglas de reescritura sobre una expresión inicial AND-EXOR. El conjunto de reglas de reescritura utilizado debe ser un conjunto convergente de reglas ([BRA93], Apartado 3.2.3) para garantizar, al menos en teoría, que el procedimiento es capaz de encontrar la expresión mínima. El conjunto de reglas que se propone en RRMIN2 es el siguiente:

grupo 1:	*grupo* 2:	*grupo* 3:	*grupo* 4:	
	$x_i\oplus\overline{x}_i\rightarrow 1$	$1\rightarrow x_i\oplus\overline{x}_i$		(4.91)
$1\oplus 1\rightarrow 0$	$1\oplus x_i\rightarrow\overline{x}_i$	$\overline{x}_i\rightarrow 1\oplus x_i$	$0\rightarrow 1\oplus 1$	
	$1\oplus\overline{x}_i\rightarrow x_i$	$x_i\rightarrow 1\oplus\overline{x}_i$		

El conjunto de reglas anterior incluye a las reglas (3.75) a (3.78), por lo que constituye un conjunto de reglas convergente, y por tanto, si las reglas se aplican en un orden adecuado deberían de llevar a la expresión mínima AND-EXOR. Por otra parte, éste conjunto de reglas es más reducido y simple (las reglas se aplican siempre sobre dos términos producto únicamente) que el propuesto por otros autores para sus procedimientos ([SAS93a], por ejemplo), con lo que se consigue acelerar el proceso de aplicación de las mismas. Las reglas se han clasificado en cuatro grupos: el grupo 1 incluye una regla de simplificación que anula productos completos, el grupo 2 incluye cuatro reglas de simplificación que involucran variables, el grupo 3 está constituido por reglas de expansión que también involucran a variables; y finalmente, el grupo 4 está formado por una regla de expansión que genera términos producto completos. En los siguientes apartados se analizan cada uno de estos tipos de regla, detallando cómo se realiza su aplicación sobre la expresión AND-EXOR que contiene el vector *L*. El proceso de selección de la regla a aplicar en cada momento se estudia en el Apartado 4.7.

4.6.1. Reglas del grupo 1

El grupo 1 está formado por una única regla, $1 \oplus 1 \rightarrow 0$. Puede aplicarse cuando un término producto aparezca dos o más veces en la expresión AND-EXOR de la función. Así, para comprobar si se puede aplicar, habrá que buscar las componentes del vector *L* que tengan un valor igual o mayor que 2. Para garantizar que el procedimiento va a buscar uniformemente, aunque no exhaustivamente por todo el espacio, no es conveniente realizar una búsqueda secuencial, puesto que se estaría favoreciendo a las primeras componentes de *L*. En consecuencia, debe realizarse una búsqueda aleatoria con distribución uniforme entre dichas componentes.

Para llevar a cabo esta búsqueda, se elije aleatoriamente una de entre las componentes de *L* que tengan dos o más productos y se aplica la regla, restando 2 a esa componente. Puesto que las nuevas expresiones de *L* no siempre van a ser aceptadas (la aceptación o no dependerá del proceso de Enfriamiento Simulado, Apartado 4.7), se mantienen dos copias de la expresión en cada paso, una almacenada en *L*, y la otra en un vector *L2*. Las reglas se aplican sobre *L2*, modificándolo, de manera que en *L* se mantiene siempre una copia de la configuración anterior para el caso en que no se acepte la nueva configuración. De esta forma, la aplicación de la regla, en pseudocódigo, puede verse en el Algoritmo 4.5, donde:

nmasdos = número de componentes de *L* con valor mayor o igual que dos. Se calcula en *Generar(L,L2)* (Véase Algoritmo 4.13 más adelante).

aleatorio() = Función que devuelve un número aleatorio en el intervalo [0,1).

regla_1(i) = Función que aplica la regla 1 sobre la posición *i* de *L2*. Se describe en el Algoritmo 4.6.

Algoritmo 4.5: Aplicación de las reglas del grupo 1

```
Aplicar_1(L,L2,nmasdos)
        error=0;
        Si (nmasdos <1)
                error=1;
                Devolver error;
        Fin Si
        limite=aleatorio()*nmasdos+1;
        p=0; i=0;
        Mientras(limite>p) hacer
                Si(L[i]>=2) p=p+1; Fin Si
                i=i+1;
        Fin Mientras
        i=i-1;
        regla_1(i);
        Devolver error;
Fin Aplicar_1
```

Algoritmo 4.6: Aplicación de las reglas del grupo 1 sobre la posición i de L2

```
regla_1(i)
        L2[i]=L2[i]-2;
Fin regla_1
```

Con este algoritmo, se aplicaría la regla del grupo 1 si es posible hacerlo. En caso contrario se devuelve un código de error para indicar que no se ha aplicado.

4.6.2. Reglas del grupo 2

El grupo 2 está formado por las reglas:

$$x_i \oplus \overline{x_i} \rightarrow 1 \tag{4.92}$$

$$1 \oplus x_i \rightarrow \overline{x_i} \tag{4.93}$$

$$1 \oplus \overline{x_i} \rightarrow x_i \tag{4.94}$$

Estas reglas se aplican sobre variables individuales y reducen en un término producto la expresión de la función. Para ello debe sacarse factor común y obtener alguna de las expresiones a la izquierda de (4.92), (4.93) ó (4.94). Por ejemplo, en la función siguiente:

$$f(x_1,x_2,x_3) = x_1x_2x_3 \oplus x_1x_2\overline{x_3} \oplus x_1x_3 \tag{4.95}$$

sacando factor común x_1x_2:

$$f(x_1,x_2,x_3) = x_1x_2(x_3 \oplus \overline{x_3}) \oplus x_2x_3 \tag{4.96}$$

puede aplicarse la regla (4.92) y obtener:

$$f(x_1,x_2,x_3) = x_1x_2 \oplus x_1x_3 \tag{4.97}$$

En definitiva, para aplicar estas reglas, los términos involucrados deben ser de la forma:

$$yx_k \oplus y\overline{x_k} \tag{4.98}$$

$$y \oplus yx_k \tag{4.99}$$

$$y \oplus y\overline{x_k} \tag{4.100}$$

donde y es un producto de variables distintas a x_k. La siguiente proposición da un criterio de aplicabilidad de las reglas del grupo 2, utilizando la notación del vector L.

Proposicion 4.23 (Criterio de aplicabilidad del grupo 2): *Sea L un vector de 3^n componentes como el definido en 4.5.3.3. Entonces, puede aplicarse alguna de las reglas del grupo 2 sobre dos componentes l(i), l(j) (i<j) de L si y solamente si se cumplen simultáneamente las siguientes dos condiciones:*

1) $l(i)\neq 0$ y $l(j)\neq 0$.
2) $j-i=3^p$ ó $j-i=2\cdot 3^p$ con $0\leq p\leq 3^{n-1}$.

Demostración:

$\Rightarrow$) *Si la regla es aplicable entre las componentes l(i) y l(j), es inmediato que se cumple 1). En cuanto a la segunda condición, basta tener en cuenta las expresiones (4.108) a (4.109). En efecto, según dichas expresiones, los dos términos producto sobre los que se va a aplicar la regla son idénticos salvo en lo que se refiere a la variable x_k. Si se expresan i y j en ternario: $i=<i_n...i_k...i_1>$, $j=<j_n...j_k...j_1>$, lo anterior se traduce en que i y j tengan la misma expresión ternaria salvo en el dígito k. Si j>i pueden darse tres casos:*

$$\begin{aligned} j_k=1 \quad i_k=0 &\Rightarrow j-i=3^{n-1} \\ j_k=2 \quad i_k=0 &\Rightarrow j-i=2\cdot 3^{n-1} \\ j_k=2 \quad i_k=1 &\Rightarrow j-i=3^{n-1} \end{aligned} \qquad (4.101)$$

En cualquiera de ellos, $j-i=3^p$ ó $j-i=2\cdot 3^p$, donde p=k-1, y por tanto $0\leq p\leq n-1$.

$\Leftarrow$) *La necesidad de 1) es obvia porque si no existen los términos producto, no puede aplicarse ninguna de las reglas del grupo 2. Si se cumple 2), quiere decir que la expresión ternaria de j e i difieren unicamente en un dígito (Véase $\Rightarrow$)). En concreto, si $j-i=3^p$ ($0\leq p\leq n-1$), se tendrá que los términos producto correspondientes son distintos únicamente en la variable x_k (k=p+1), presentándose dos casos posibles:*

$$j-i=3^p=3^{k-1}\Rightarrow \left[\begin{array}{l} \text{si } j_k=1 \text{ y } i_k=0 \text{ es posible aplicar } 1\oplus x_k \rightarrow \overline{x_k} \\ \text{si } j_k=2 \text{ y } i_k=1 \text{ es posible aplicar } x_k\oplus \overline{x_k} \rightarrow 1 \end{array}\right. \qquad (4.102)$$

Si $j-i=2\cdot 3^p$ se tendrá nuevamente que los términos producto son idénticos salvo en la variable x_k, y que es posible aplicar la regla:

$$j-i=2\cdot 3^p=2\cdot 3^{k-1}\Rightarrow j_k=2 \text{ y } i_k=0 \text{ se puede aplicar } 1\oplus \overline{x_k}\rightarrow x_k \qquad (4.103)$$

con lo que queda probada la proposición.

Una vez que se dispone de un criterio para la aplicabilidad de las reglas, el siguiente paso consiste en aplicarlas. En la notación del vector *L*, la aplicación de este grupo de reglas, teniendo en cuenta (4.86), quedaría:

$$x_k \oplus \overline{x_k} \rightarrow 1 \equiv \begin{bmatrix} l(i-3^{k-1}) = l(i-3^{k-1}) + 1 \\ l(i) = l(i-1) \\ l(j) = l(j-1) \end{bmatrix} \tag{4.104}$$

$$1 \oplus x_k \rightarrow \overline{x_k} \equiv \begin{bmatrix} l(j+3^{k-1}) = l(j+3^{k-1}) + 1 \\ l(i) = l(i) - 1 \\ l(j) = l(j) - 1 \end{bmatrix} \tag{4.105}$$

$$1 \oplus \overline{x_k} \rightarrow x_k \equiv \begin{bmatrix} l(i+3^{k-1}) = l(i+3^{k-1}) + 1 \\ l(i) = l(i) - 1 \\ l(j) = l(j) - 1 \end{bmatrix} \tag{4.106}$$

donde $l(i)$ y $l(j)$ están en las condiciones de la Proposición 4.23.

Finalmente, queda por establecer el método de búsqueda de las componentes sobre las que se aplicarán las reglas. Al igual que en el caso del grupo 1, no resulta conveniente efectuar búsquedas secuenciales, y se van a realizar búsquedas aleatorias uniformes sobre las componentes de L. Así, en primer lugar se va a seleccionar aleatoriamente una componente i de entre las que cumplan $l(i) \neq 0$, y después se pueden considerar dos opciones para buscar una componente j que esté en las condiciones de la Proposición 4.23:

1) Elegir aleatoriamente una componente j de entre las que cumplan $j>i$ y $l(j) \neq 0$, y posteriormente comprobar la condición 2) de la Proposición 4.23. No es preciso realizar ningún cálculo previo, pero el espacio de búsqueda puede estar constituido por un gran número de componentes si la función tiene muchos términos.

2) Seleccionar aleatoriamente una componente j de entre las que cumplan la condición 2) de la Proposición 4.23. y después comprobar que $l(j) \neq 0$. Supone un mayor cálculo preliminar, puesto que hay que calcular previamente las j que cumplen las condiciones, pero hay que buscar entre un número mucho menor de componentes.

La segunda opción resulta más ventajosa puesto que implica la búsqueda entre $2(n\text{-}1)$ componentes (complejidad lineal con el número de variables) mientras que la opción 1) depende del número de términos producto de la función (en promedio aumentará como el número de componentes del vector L, es decir, se tendrá complejidad exponencial), que puede ser muy elevado.

Para la elección de las componentes, según se ha indicado anteriormente, se va a seguir un proceso de selección aleatoria. Este proceso debe ser tal que si es posible aplicar una regla del grupo 2 sobre la expresión de la función, en efecto se aplique. Por ejemplo, considérese la elección de la componente i. Debe procurarse que se prueben todas las componentes distintas de 0 de L antes de desechar la aplicación de reglas del grupo 2. Si se

tiene que el número de componentes distintas de 0 es *n_posiciones*, la probabilidad de no elegir una componente *k* en un experimento aleatorio es:

$$p_1 = 1 - \frac{1}{n\_posiciones} \tag{4.107}$$

Si el experimento se repite *n_vecesi* veces, la probabilidad de que no se haya elegido esa componente *k* será:

$$p_{n\_vecesi} = (1 - \frac{1}{n\_posiciones})^{n\_vecesi} \equiv p_{falloi} \tag{4.108}$$

Este parámetro p_{falloi} es la probabilidad de dejarse una componente sin explorar al elegir una primera componente para la aplicación de las reglas del grupo 2, tras repetir el proceso de selección aleatoria *n_vecesi*. Para escoger el valor de *n_vecesi*, se fija la p_{falloi} máxima que se está dispuesto a admitir, y se calcula *n_vecesi* según:

$$n\_vecesi = \frac{\ln(p_{falloi})}{\ln(1 - \frac{1}{n\_posiciones})} \tag{4.109}$$

donde *n_posiciones*>1 y $0<p_{falloi}<1$.

De forma análoga, una vez seleccionada una componente *i*, deben probarse varias componentes *j* de las que cumplen las condiciones 2) de la Proposición 4.23 hasta encontrar una que cumpla $l(j)\neq 0$. Si el número de estas componentes es *n_posicionesj*, el número de veces a repetir la elección de un *j* será:

$$n\_vecesj = \frac{\ln(p_{falloj})}{\ln(1 - \frac{1}{n\_posicionesj})} \tag{4.110}$$

Teniendo en cuenta todas estas consideraciones, la aplicación de las reglas del grupo 2 es tal como se muestra en el Algoritmo 4.7, donde:

n_posiciones=número de componentes de *L* distintas de 0. Se calcula en *Generar(L,L2)* (Véase Algoritmo 4.13).
aleatorio()=Función que devuelve un número aleatorio en el intervalo [0,1).
regla_2(i)=Función que aplica una regla del grupo 2 sobre las posiciones i y j de L2. Se describe en Algoritmo 4.8.

En el apartado siguiente se estudia la aplicación de las reglas del grupo 3.

Algoritmo 4.7: Aplicación de las reglas del grupo 2:

```
Aplicar_2(L,L2,n_posiciones)
        error=1;
        n_vecesi=log(p_falloi)/log((n_posiciones-1)/n_posiciones);
        indice1=0;
        Mientras (indice1<n_vecesi) hacer
                limite1=n_posiciones*aleatorio();
                limite1=limite1+1;
                p=0; i=0;
                Mientras (limite1>p) hacer
                        Si (L[i])
                                p=p+1;
                        Fin Si
                        i=i+1;
                Fin Mientras
                i=i-1;
                Si (i=3^n-1)
                        Devolver error;
                Fin Si
                maxi=3^n-1-i;
                limite2=log(maxi)/log(3)+1;
                n_posicionesj=2*limite2;
                Si((i+2*3^limite2)>(3^n-1))
                        n_posicionesj=n_posicionesj-1;
                Fin Si
                n_vecesj=log(p_falloj)/log((n_posicionesj-1)/n_posicionesj);
                Mientras (indice2<n_vecesj) hacer
                        doble=2*aleatorio();
                        j=i+3limite2*aleatorio();
                        Si (doble)
                                j=j+3limite2*aleatorio();
                        Fin Si
                        Si (j<3n)
                                error=regla_2(i,j);
                        Fin Si
                        Si (error=0)
                                Devolver error;
                        Fin Si
                        indice2=indice2+1;
                Fin Mientras
                indice1=indice1+1;
        Fin Mientras
Devolver error
Fin Aplicar_2
```

Algoritmo 4.8: Aplicación de las reglas del grupo 2 sobre las posiciones i,j de L2

```
regla_2(i,j)
        error=0;
        Si (K[j]=0) error=1; Devolver error; Fin Si
        diferencias=0;
        cociente1=i;
        cociente2=j;
        Mientras (cociente1 O cociente2) hacer
                resto1=cociente1 mod 3;
                resto2=cociente2 mod 3;
                Si (resto1-resto2)
                        diferencias=diferencias+1;
                        valor=resto1;
                Fin Si
                cociente1=cociente1/3;
                cociente2=cociente2/3;
        Fin Mientras
        Si (diferencias=1)
                indice=0;
                Mientras (indice<n) hacer
                        paso=3^indice;
                        diferencia=j-i;
                        Si (diferencia=paso)
                                Si (valor=0)
                                        L2[j+paso]=L2[j+paso]+1;
                                        L2[i]=L2[i]-1;
                                        L2[j]=L2[j]-1;
                                        Devolver error;
                                Fin Si
                                Si (valor=1)
                                        L2[i-paso]=L2[i-paso]+1;
                                        L2[i]=L2[i]-1;
                                        L2[j]=L2[j]-1;
                                        Devolver error;
                                Fin Si
                        Si (diferencia=2*paso)
                                L2[i+paso]=L2[i+paso]+1;
                                L2[i]=L2[i]-1;
                                L2[j]=L2[j]-1;
                                Devolver error;
                        Fin Si
                indice=indice+1;
                Fin Mientras
        Fin Si
        Devolver error;
Fin Regla2
```

4.6.3. Reglas del grupo 3

Las reglas que forman el grupo 3 son:

$$1 \rightarrow x_i \oplus \overline{x_i} \tag{4.111}$$

$$\overline{x_i} \rightarrow 1 \oplus x_i \tag{4.112}$$

$$x_i \rightarrow 1 \oplus \overline{x_i} \tag{4.113}$$

Estas tres reglas producen una expansión de la función, aumentando el número de términos en una unidad. Se trata de reglas que se aplican a variables individuales y que pueden aplicarse siempre que exista algún término producto en la función (en caso contrario se tendría la función 0 y no tendría sentido minimizar). Nótese que (4.111), (4.112) y (4.113) realizan, respectivamente, la operación inversa que (4.92), (4.93) y (4.94), de manera que el proceso de aplicación en notación del vector L se hará de forma análoga a (4.104), (4.105) y (4.106). Concretamente:

$$1 \rightarrow x_k \oplus \overline{x_k} \equiv \begin{cases} l(i) = l(i) - 1 \\ l(i+3^{k-1}) = l(i+3^{k-1}) + 1 \\ l(i+2\cdot 3^{k-1}) = l(i+2\cdot 3^{k-1}) + 1 \end{cases} \tag{4.114}$$

$$\overline{x_k} \rightarrow 1 \oplus x_k \equiv \begin{cases} l(i) = l(i) - 1 \\ l(i-3^{k-1}) = l(i-3^{k-1}) + 1 \\ l(i-2\cdot 3^{k-1}) = l(i-2\cdot 3^{k-1}) + 1 \end{cases} \tag{4.115}$$

$$x_k \rightarrow 1 \oplus \overline{x_k} \equiv \begin{cases} l(i) = l(i) - 1 \\ l(i-3^{k-1}) = l(i-3^{k-1}) + 1 \\ l(i+3^{k-1}) = l(i+3^{k-1}) + 1 \end{cases} \tag{4.116}$$

donde i es tal que $l(i) \neq 0$.

Para realizar estas aplicaciones, será preciso buscar aleatoriamente la componente i entre aquellas que cumplan l(i)≠0, y posteriormente una variable sobre la que aplicar la regla de expansión. Si la variable escogida no aparece en el término producto se utilizará (4.114), si aparece complementada, (4.115), y si aparece sin complementar, (4.116). En este caso no es preciso repetir los experimentos aleatorios, puesto que siempre será posible aplicar una de estas reglas. Así, el algoritmo para efectuar la aplicación de las reglas de este grupo sería el mostrado en Algoritmo 4.9, donde:

n_posiciones=número de componentes de L distintas de 0. Se calculan en *Generar(L,L2)* (Véase Algoritmo 4.13).

aleatorio()=Función que devuelve un número aleatorio en el intervalo [0,1).

regla_3(i,variable)=Función que aplica una regla del grupo 2 a la componente *i*. Se

describe en el Algoritmo 4.10.

Algoritmo 4.9: Aplicación de las reglas del grupo 3

```
Aplicar_3(L,L2,n_posiciones)
        limite=n_posiciones*aleatorio();
        limite=limite+1;
        p=0; i=0;
        Mientras (limite>p) hacer
                Si (L[i])
                        p=p+1;
                Fin Si
                i=i+1;
        Fin Mientras
        i=i-1;
        variable=n*aleatorio();
        regla_3(i,variable);
Fin Aplicar_3
```

Algoritmo 4.10: Aplicación de las reglas del grupo 3 sobre la posición i de L2

```
regla_3(i,variable)
        paso=3^variable;
        resto=i;
        indice=n-1;
        Mientras (indice>=variable) hacer
                cociente=resto;
                valor=3^indice;
                resto = cociente mod valor;
                cociente=cociente/valor;
                indice=indice-1;
        Fin Mientras
        Si (cociente=0)
                L2[i]=L2[i]-1;
                L2[i+paso]=L2[i+paso]+1;
                L2[i+2*paso]=L2[i+2*paso]+1;
        Fin Si
        Si (cociente=1)
                L2[i]=L2[i]-1;
                L2[i-paso]=L2[i-paso]+1;
                L2[i+paso]=L2[i+paso]-1;
        Fin Si
        Si (cociente=2)
                L2[i]=L2[i]-1;
                L2[i-paso]=L2[i-paso]+1;
                L2[i-2*paso]=L2[i-2*paso]+1;
        Fin Si
Fin regla_3
```

Finalmente, en el siguiente apartado se detalla la aplicación de las reglas del grupo 4, con lo que se completa el conjunto de reglas de RRMIN2.

4.6.4. Reglas del grupo 4

El grupo 4 está constituido por una única regla:

$$0 \rightarrow 1 \oplus 1 \tag{4.117}$$

Se trata de una regla de expansión que genera dos términos producto nuevos. Por tanto, supone un aumento de dos unidades en el número de productos. Puede aplicarse siempre, y por tanto, la componente de aplicación se seleccionará aleatoriamente entre el total de componentes. Una vez seleccionada esa componente i, la aplicación queda reducida, en notación del vector L, a efectuar la operación $l(i)=l(i)+2$.

Su aplicación puede verse en el Algoritmo 4.11, donde *aleatorio()* tiene el mismo significado que en los algoritmos anteriores, y *regla_4(i)* es la función que aplica la regla del grupo 4 sobre la componente i, tal y como se describe en el Algoritmo 4.12.

Algoritmo 4.11: Aplicación de las reglas del grupo 4

```
Aplicar_4(L,L2)
        i=3^n*aleatorio();
        regla_4(i);
Fin Aplicar_4
```

Algoritmo 4.12: Aplicación de la regla del grupo 4 sobre la componente i

```
regla_4(i)
        L2[i]=L2[i]+2;
Fin regla_4
```

En los apartados 4.6.1 a 4.6.4 se ha descrito el conjunto de reglas de reescritura utilizado por RRMIN2, pero no se ha especificado cuál es el procedimiento que se sigue para su selección. En los apartados siguientes (4.6.5 y 4.7) se trata esta cuestión.

4.6.5. Selección aleatoria de los grupos de reglas

Para seleccionar la regla a aplicar en cada momento, RRMIN2 elige aleatoriamente

uno de los cuatro grupos de reglas descritos en los apartados anteriores, intenta aplicar alguna de las reglas del grupo seleccionado, obtiene una nueva expresión de la función (si ha conseguido aplicar alguna regla), y finalmente, mediante un proceso de Enfriamiento Simulado decide si acepta la nueva expresión o la desecha. En este apartado se va a estudiar la elección del grupo de reglas a aplicar, y en 4.7 se detalla el proceso de Enfriamiento Simulado.

En principio, lo más sencillo sería hacer que los cuatro grupos de reglas tuvieran la misma probabilidad de ser seleccionados. Sin embargo, es fácil llegar a la conclusión de que no es el más adecuado, puesto que las reglas de expansión siempre son aplicables, lo que llevaría a que la evolución a expresiones más simples fuera muy costosa. Así, resulta conveniente dar probabilidades distintas a cada uno de los grupos de reglas, de manera que la selección de los mismos se efectuaría según se muestra en el Algoritmo 4.13, donde *p_grupo1, p_grupo2, p_grupo3* y *p_grupo4* son las probabilidades asignadas a cada uno de los grupos de reglas. Para asignar valores a estas probabilidades se ha dividido el proceso de minimización en dos tramos:

- **Tramo 1.** La expresión inicial AND-EXOR obtenida, la mejor forma MPRM (Apartado 4.13) ,normalmente contiene un número de productos bastante superior al mínimo AND-EXOR ([PAR94]). Para bajar este número de productos, es conveniente aplicar inicialmente únicamente reglas de simplificación. Teniendo en cuenta que la forma MPRM va a producir un vector L en el que sólamente hay unos (se trata de una forma canónica), el grupo 1 no va a ser aplicable, y en este tramo las probabilidades que se van a utilizar son:

$$p\_regla1=0$$
$$p\_regla2=1$$
$$p\_regla3=0$$
$$p\_regla4=0$$

Naturalmente, después de un determinado número de iteraciones, se llegará a un mínimo local y no será posible aplicar en más ocasiones reglas del grupo 2. Se pasará entonces al tramo 2 (Véase el Algoritmo 4.13 para más detalle).

- **Tramo 2.** Una vez que se ha alcanzado el mínimo local, es necesario utilizar reglas de expansión para poder salir del mismo, de forma que se usan las siguientes probabilidades:

$$p\_regla1=p\_regla1i$$
$$p\_regla2=p\_regla2i$$
$$p\_regla3=p\_regla3i$$
$$p\_regla4=1-p\_regla1-p\_regla2-p\_regla3$$

donde *p_regla1i, p_regla2i, p_regla3i* dependerán del tipo de función que se tenga, y han sido determinadas experimentalmente de la forma que se detalla en el Apartado 6.3.1.1.

Algoritmo 4.13: Selección de los grupos de reglas

```
Generar(L,L2)
        error=0;
        n_posiciones=0;
        nmasdos=0;
        indice=0;
        Mientras (indice<3^n) hacer
                Si (L2[indice])
                        n_posiciones=n_posiciones+1;
                Fin Si
                Si (L2[indice] >=2)
                        nmasdos=nmasdos+1;
                Fin Si
                indice=indice+1;
        Fin Mientras;
        Si (n_posiciones<2)
                Devolver error;
        Fin Si
        Si (tramo=1)
                p_grupo1=0; p_grupo2=1; p_grupo3=0; p_grupo4=0;
        Fin Si
        Si (tramo=2)
                p_grupo1=p_grupo1i; p_grupo2=p_grupo2i;
                p_grupo3=p_grupo3i; p_grupo4=1-p_grupo1-p_grupo2-p_grupo3;
        Fin Si
        n_grupo=1;
        aleat=aleatorio();
        aleat=aleat-p_grupo1;
        Si (aleat>0) n_grupo=n_grupo+1;
        Fin Si
        aleat=aleat-p_grupo2;
        Si (aleat>0) n_grupo=n_grupo+1;
        Fin Si
        aleat=aleat-p_grupo3;
        Si (aleat>0) n_grupo=n_grupo+1;
        Fin Si
        Si (n_grupo=1) error=Aplicar_1(L,L2,nmasdos);
        Fin Si
        Si (n_grupo=2) error=Aplicar_2(L,L2,n_posiciones);
        Fin Si
        Si (n_grupo=3) Aplicar_3(L,L2,n_posiciones);
        Fin Si
        Si (n_grupo=4) Aplicar_4(L,L2);
        Fin Si
        Devolver error;
Fin Generar
```

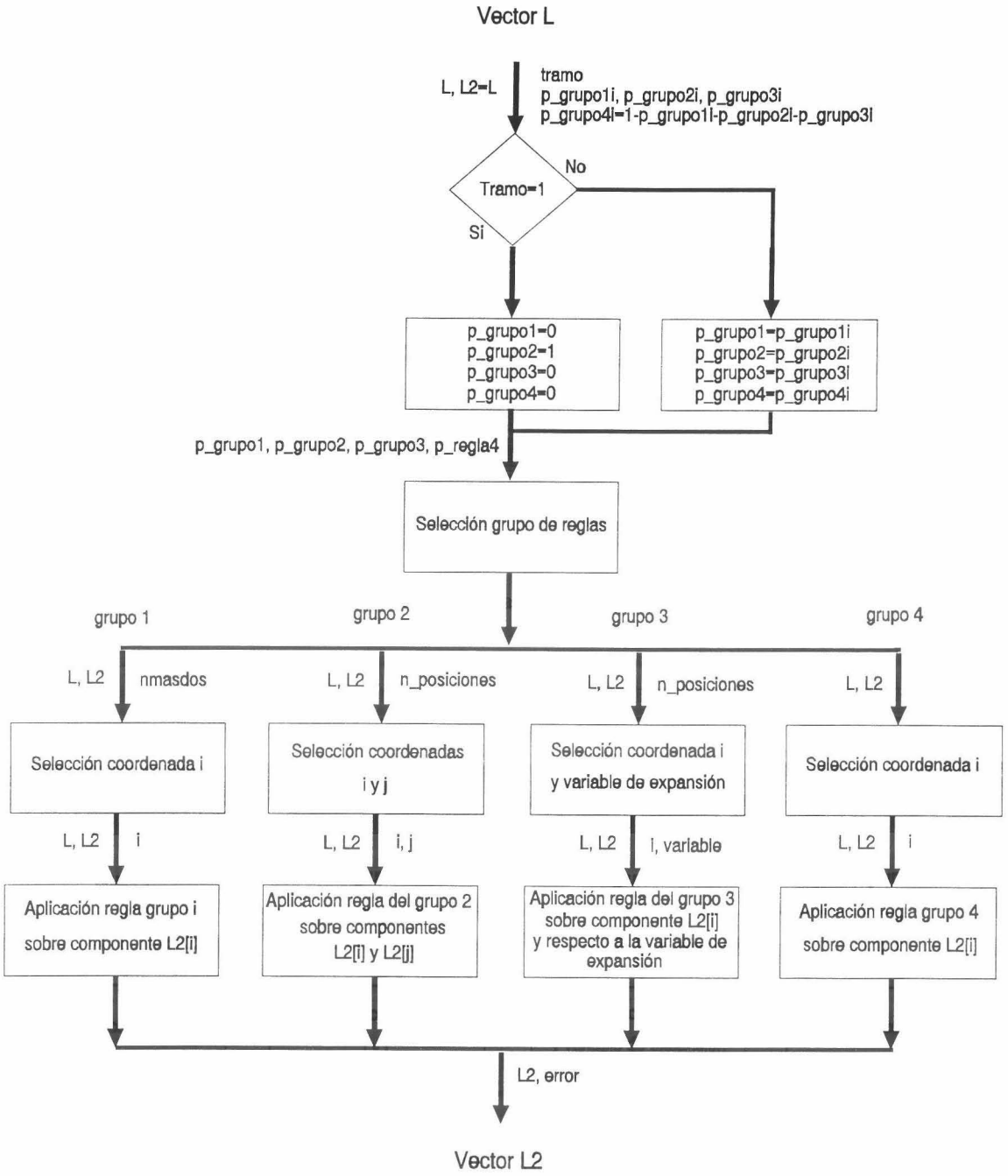


Fig. 4.17. Obtención de un nuevo estado L2 a partir del estado actual L.

El mecanismo de paso de un tramo a otro se detalla en el apartado siguiente, y la implementación puede verse en el Algoritmo 4.19.

El Algoritmo 4.13, en definitiva, genera una nueva expresión de la función (*L2*) a partir de la actual (*L*), seleccionando el grupo de reglas a aplicar en cada momento. Nótese que no siempre se va a generar una configuración nueva (puede ocurrir que se seleccione el grupo 1 o el 2 y no sea posible aplicar una regla de dichos grupos). En la Figura 4.17 puede verse un organigrama del proceso de selección y aplicación de las reglas de reescritura en RRMIN2.

En el apartado siguiente se detalla el procedimiento seguido para la aceptación o no de la nueva configuración obtenida por *Generar(L,L2)*.

4.7. CONTROL DE LA APLICACIÓN DE LAS REGLAS

Tal y como se ha adelantado en apartados anteriores, la aceptación o no de una determinada configuración obtenida por la aplicación de las reglas de reescritura, se va a realizar utilizando un proceso de Enfriamiento Simulado (Apartado 4.3.3). Dado que la minimización AND-EXOR puede formularse como un problema de optimización combinatoria según, se detalló ene l Apartado 4.2, se ha escogido este algoritmo para controlar la aplicación de reglas en el procedimiento RRMIN2. En el apartado que sigue se definen el conjunto de soluciones y la función de coste, que son los dos elementos que no se proporcionaron en 4.2 y que son necesarios para la definición formal del problema de optimización combinatoria.

4.7.1. Espacio de soluciones y función de coste en RRMIN2

Para tener planteado adecuadamente un problema de optimización combinatoria (Apartado 4.2) es necesario tener bien determinado el espacio de soluciones *S* y la función de coste f. En el caso de la minimización AND-EXOR, y utilizando la notación del vector *L* introducida en 4.5.3.3, un buen espacio de soluciones vendría dado por:

$$S'=\{L \;/\; l(i)=0,1 \;\; 0<i<3^n\} \qquad (4.118)$$

Nótese que *S'* es un conjunto de soluciones apropiado porque:

1) *S'* es un conjunto finito.
2) *S'* incluye a las soluciones óptimas S_{opt}. En efecto, en una solución óptima siempre se da $l(i)<2$ puesto que en caso contrario es aplicable la regla de simplificación del grupo 1 (4.91).

Sin embargo, se plantea el problema, de que la aplicación de reglas del grupo 4 van a originar configuraciones que no pertenecen a este conjunto de soluciones. Por ello, es necesario utilizar una extensión de este conjunto, *S*:

$$S=\{L \ / \ l(i)=0,1,...,h \quad 0<i<3^n\} \tag{4.119}$$

donde h es un número natural. El valor de h, en principio, es desconocido, pero es fácil ver que es un valor finito. Una cota superior para h sería el número de veces que se aplique la regla (4.117) durante el proceso de minimización, que tiene que ser un valor finito puesto que el algoritmo de Enfriamiento Simulado es finito. De esta forma S incluye a S_{opt} por ser $S' \sqsubset S$, S es finito, y ya se pueden incluir configuraciones originadas por la aplicación de la regla de expansión del grupo 4 (4.117).

En cuanto a la función de coste, f, va a venir dada por:

$$f(x)=n\_productosL(x)+LRR(x) \quad x \in S \tag{4.120}$$

donde *n_productosL* es el número de términos producto correspondiente al estado x y *LRR(x)* (Literal Reduction Rate, [LLO93]) es el cociente entre el número de literales de la configuración actual y el número de literales de la función inicial. De esta forma, el problema de la minimización AND-EXOR consistirá en encontrar una solución $x_{opt} \in S$ que satisfaga:

$$f(x_{opt}) \leq f(x), \quad para\ todo\ x \in S \tag{4.121}$$

La función de coste utilizada corresponde a elegir como estado óptimo aquel que:

1) Contenga el menor número de términos producto (la función *LRR(x)* es siempre menor o igual que 1).
2) De entre aquellos que tengan ese número mínimo de productos, el que incluya un menor número de literales.

Una vez que se tiene bien definido el problema de optimización combinatoria, puede aplicarse el algoritmo de Enfriamiento Simulado para su resolución aproximada. En la siguiente sección se describe la aplicación del citado algoritmo.

4.7.2. Aceptación de las reglas mediante Enfriamiento Simulado

El algoritmo de Enfriamiento Simulado, de forma general, puede escribirse tal y como se muestra en el Algoritmo 4.14.

En dicho algoritmo quedan por especificar:

- **La configuración inicial $x_{inicial}$.** Según se ha indicado en el Apartado 4.5, la configuración inicial que se utiliza es la forma MPRM con menor número de productos.
- **El valor inicial del parámetro de control, c_0.** Lo fija el usuario y debe ser tal que la probabilidad de aceptar una transición sea próxima a 1 (Apartado 4.3.3). En 6.4 se realiza un estudio para determinar los valores adecuados de este parámetro.

Algoritmo 4.14: Enfriamiento Simulado

```
SA()
Iniciar(x_inicial, c_0, l_0);
k=0;
x=x_inicial;
Repetir
        l=1;
        Mientras(l<=l_k) hacer
                Generar(y a partir de S_x );
                Si (f(y)<=f(x))
                        x=y;
                Fin Si
                Si (f(y)>f(x))
                        Si (exp((f(x)-f(y))/c_k)>aleatorio())
                                x=y;
                        Fin Si
                Fin Si
                l=l+1;
        Fin Mientras
        k=k+1;
        Calcular_Longitud(l_k);
        Calcular_Control(c_k);
    Hasta Criterio_de_fin()
Fin SA
```

- **La longitud inicial de las cadenas de Markov, l_0.** En el esquema de enfriamiento en tiempo polinomial descrito en [AAR90] (Apartado 4.3.3) se propone hacer la longitud de las cadenas de Markov igual al tamaño de la vecindad. De esta forma, se trata de garantizar que se va a efectuar una buena exploración de las transiciones posibles, pero acotando el tiempo de ejecución. En el caso de RRMIN2, el tamaño de la vecindad es excesivo, por lo que la determinación de l_0 se ha realizado experimentalmente de la forma que se detalla en 6.3.1.2.
- **La función *Generar(y a partir de S_x).*** Esta función, en el caso de RRMIN2, viene dada por la función *Generar(L,L2)* (Algoritmo 4.13) descrita en el Apartado 4.6.5.
- **La función de coste *f*.** Se ha definido en el Apartado 4.7.1 (4.120), y viene dada por *f(x)=n_productosL(x)+LRR(x)*. Para determinar el número de productos y de literales (necesario para calcular LRR(x)) de la configuración actual, se va a usar la función *Costo_L(L)*. En el Algoritmo 4.15 se muestra el pseudocódigo para el cálculo de la función de coste, donde *n_productosL* contiene el número de productos del vector *L* en la configuración actual y *n_litL* el número de literales. Nótese que se cuenta un literal por cada dígito distinto de 0 en la expresión ternaria del índice *i* (Véase Apartado 4.5.3.3) y por cada término producto que aparezca en la posición *L*[*i*].
- **La función *Calcular_Longitud(l_k).*** Tal y como se ha comentado anteriormente, el

tamaño de la vecindad en RRMIN2 es muy grande (sólamente aplicando la regla del grupo 4 se obtienen 3^n transiciones posibles), y ha de utilizarse otro método para determinar l_k. Se ha optado por el procedimiento más simple, manteniendo la misma longitud de las cadenas de Markov durante todo elproceso de minimización, de manera que esta función simplemente hace $l_k=l_0$, tal y como muestra el Algoritmo 4.16.

Algoritmo 4.15: Cálculo de la función de coste

```
Costo_L(L)
i=0;
n_productosL=0;
n_litL=0;
Mientras (i<3^n) hacer
        n_productosL=n_productosL+L[i];
        j=0;
        modulo=i;
        Mientras (j<n) hacer
                Si (modulo % 3)
                        n_litL=n_litL+L[i];
                Fin Si
                modulo=modulo/3;
                j=j+1
        Fin Mientras
        i=i+1;
Fin Mientras
Fin Costo_L
```

Algoritmo 4.16: Cálculo de la longitud del paso k

```
Calcular_Longitud(l_k)
        l_k=l_0;
Fin Calcular_Longitud
```

- **La función *Calcular_Control*(c_k).** Viene dada por el Algoritmo 4.17 donde b y Δ_0 son parámetros proporcionados por el usuario y Δ_m es un valor mínimo para Δ, que tiene el fin de evitar que se den casos en que el número de iteraciones a realizar en el proceso de Enfriamiento Simulado sea infinito. Este procedimiento de bajada de la temperatura es algo distinto a los utilizados usualmente. En [AAR90] se propone como primera aproximación utilizar:

$$c_{k+1}=\alpha \cdot c_k \tag{4.122}$$

donde α es una constante menor que 1. Los valores típicos para α son $0.8 \leq \alpha \leq 0.99$. Sin embargo, este método no permite un control demasiado preciso sobre la bajada de la temperatura. En esa misma referencia se propone otro procedimiento más elaborado, que viene dado por la expresión:

$$c_{k+1}=\frac{c_k}{1+\frac{c_k \ln(1+\delta)}{3\sigma_{c_k}}} \tag{4.123}$$

donde σ_{c_k} es la desviación típica de la función de coste a la temperatura c_k y δ es el parámetro que utilizaría el usuario para controlar la bajada de la temperatura. La dependencia con la desviación típica permite adaptar la bajada de la temperatura según las variaciones de la función de coste. En el caso de tener temperaturas bajas, o si se cae en un mínimo local, se puede producir una caída brusca de la temperatura, que no resulta conveniente para el procedimiento RRMIN2. En el Apéndice B se realiza un estudio más detallado del proceso de enfriamiento que se sigue en RRMIN2.

Algoritmo 4.17: Cálculo de la temperatura en el paso k

```
Calcular_Control(c_k)
        c_k= c_{k-1}-Δ_k;
        Δ_k=Δ_{k-1}/b;
        Si (Δ_k<Δ_m)
                Δ_k=Δ_m;
        Fin Si
Fin Calcular_Control
```

- ***Criterio_de_Fin().*** Viene dada por el Algoritmo 4.18, donde c_f es otro parámetro que fija el usuario. Nótese que se ha optado por utilizar una temperatura final para establecer el fin del proceso de enfriamiento. Existen otras posibilidades, como la descrita en [AAR90], donde el criterio de fin viene dado por la expresión:

$$\frac{c_k}{\langle f \rangle_\infty}\left.\frac{\partial \langle f \rangle_c}{\partial c}\right|_{c=c_k} < \varepsilon_s \tag{4.124}$$

donde ε_s es una constante que se fijaría el usuario, denominada criterio de parada,

$\langle f \rangle_\infty$ es el coste esperado a temperatura infinita, y $\langle f \rangle_c$ es el coste esperado a la temperatura c. Este procedimiento presenta el inconveniente de que puede dar lugar a una finalización prematura del proceso en caso de caída en un mínimo local (la derivada podría hacerse negativa en caso de no salir pronto de él), o simplemente por fluctuaciones de la función de coste a temperaturas altas.

Otro método bastante utilizado consiste en terminar el proceso de enfriamiento si no se ha mejorado la función de coste después de un determinado número de iteraciones. Puesto que en la minimización AND-EXOR se tienen mínimos locales que precisan de una transformación profunda de la función de conmutación tratada para salir de ellos, no resulta aconsejable utilizar ninguna de las dos posibilidades mencionadas, y se ha optado por fijar una temperatura final.

En cuanto al valor de c_f, debe de ser tal que a esa temperatura no se produzcan transiciones a estados de coste más elevado. Nótese que en ese caso se tendría:

$$\left.\frac{\partial \langle f \rangle_c}{\partial c}\right|_{c=c_f} \simeq 0 \qquad (4.125)$$

con lo que se estaría cumpliendo el criterio dado en (4.124).

Algoritmo 4.18: Criterio de fin

```
Criterio_de_Fin()
        Si (c_k < c_f)
                Terminar;
        Fin Si
Fin Criterio_de_Fin
```

En resumen, el algoritmo de Enfriamiento Simulado que controla la aceptación de las reglas de reescritura en RRMIN2, precisa de una serie de parámetros externos que son los siguientes:

1) c_0 ó Temperatura Inicial (utilizando la nomenclatura usual en algoritmos de Enfriamiento Simulado).
2) Δ_0 ó Incremento Inicial de Temperatura.
3) c_f ó Temperatura Final.
4) b.

Mediante estos parámetros es posible controlar el tiempo de ejecución y la calidad de los resultados a obtener. En el Apartado 6.4 se presentan estudios detallados de los efectos que produce la variación de cada uno de estos parámetros en el proceso de minimización. En el apartado siguiente se describe el algoritmo de Enfriamiento Simulado concreto que se utiliza en RRMIN2.

4.7.3. Aceptación de las reglas en RRMIN2

En el procedimiento de minimización RRMIN2 se va a efectuar un proceso de Enfriamiento Simulado con dos modificaciones:

1) La primera modificación consiste en ir almacenando una copia de la mejor configuración que se ha generado hasta el momento (esta copia se denomina *Lop*). De esta forma se consigue ahorrar tiempo de ejecución, puesto que no es preciso efectuar el proceso de enfriamiento con incrementos de temperatura tan pequeños y hasta temperaturas tan bajas como para garantizar que el estado final obtenido sea la mejor solución.

2) Para evitar comparaciones de valores en coma flotante se han modificado las comparaciones que involucran a la función de coste de la forma que aparece en el Algoritmo 4.19. Nótese que debido a la modificación 1 y a que tiene prioridad la minimización en número de productos, sólo se tiene en cuenta el número de literales en caso de que el número de productos sea igual al óptimo.

En cuanto a las probabilidades de selección de cada grupo de reglas, ya se explicó en 4.6.5 que se van a considerar dos tramos, uno en el que únicamente se van a aplicar reglas de simplicación del grupo 2, y un segundo tramo en que se van a utilizar todos los grupos de reglas con unas probabilidades que se han determinado experimentalmente y que se detallan en 6.3.1.1. El tramo 1 da como resultado un mínimo local, del que sólo es posible salir pasando al tramo 2. El criterio que se sigue para efectuar este paso es que en la última cadena de Markov de longitud l no se haya conseguido aplicar en ninguna ocasión una de las reglas del grupo 2. El Algoritmo 4.19 presenta el proceso de aceptación de las reglas en RRMIN2, teniendo en cuenta estas cuestiones y el Apartado 4.7.2. Los parámetros y funciones utilizados en ese algoritmo han sido explicados suficientemente en las secciones anteriores.

En el Apartado 4.8 se describe una mejora del procedimiento para el caso de que la función a minimizar contenga un número de unos superior al 50%. Los Apartados 4.9 y 4.10, muestran la ampliación de los algoritmos expuestos para abordar la minimización de funciones incompletamente especificadas y con múltiples salidas (síntesis multifuncional).

Algoritmo 4.19: Control de la aplicación de las reglas en RRMIN2

```
Control_ES()
L2=L;
Lop=L;
parar=0;
costo_L(L)
c=c0;
Δ=Δ0;
tramo=1;
Mientras (parar=0) hacer
        l=0;
        Calcular_Longitud(lk);
        contador=0;
        Mientras (l<lk )hacer
                Generar(L,L2);
                Costo_L(L);
                Costo_L2(L2);
                Costo_Lop(Lop);
                Si (n_productosL2 < n_productosL)
                        L=L2;
                        Si (n_productosL2 < n_productosLop)
                                Lop=L2;
                                contador++;
                        Fin Si
                Fin Si
                Otro
                        Si (n_productosL >= n_productosL)
                                Si (exp((n_productosL-n_productosL2)/c)>aleatorio()) L=L2;
                                Fin Si
                        Fin Si
                Fin Otro
                Si (n_productosL2=n_productosLop) Lop=L2
                Fin Si
                L2=L;
                l=l+1;
        Fin Mientras
        c=c-Δ;
        Δ=Δ/b;
        Si (Δ < Δm)  Δ= Δm; Fin Si
        Si (c<Tfinal) parar=1; Fin Si
        Si (contador=0) tramo=2; Fin Si
        L=Lop;
        L2=Lop;
Fin Mientras;
Fin Control_ES
```

4.8. MINIMIZACIÓN DE FUNCIONES CON MÁS DE UN 50% DE UNOS

Las funciones con un gran número de unos en su tabla verdad, generalmente llevan a la obtención de una forma MPRM óptima con un elevado número de términos producto. La minimización mediante aplicación de reglas de este tipo supone un gran esfuerzo y da lugar a tiempos de ejecución grandes o a resultados no demasiado satisfactorios (véanse resultados obtenidos por otros procedimientos en la Tabla 6.10).

Para solucionar este problema en RRMIN2, se ha aprovechado la siguiente propiedad de la lógica AND-EXOR:

$$\overline{f(x)}=f(x)\oplus 1 \tag{4.126}$$

o escrito de otra forma:

$$f(x)=\overline{f(x)}\oplus 1 \tag{4.127}$$

Utilizando esta última expresión, si se tiene una función con un gran número de unos en su tabla verdad, podría complementarse dicha tabla, obtener la expresión inicial AND-EXOR, sumarle un 1 (con lo cual se tiene ya $f(x)$ en forma AND-EXOR) y minimizar la función resultante (que resultará mucho más fácil por tener pocos unos). El proceso sería:

1) Complementar la tabla verdad de $f(x)$ (cambiar los unos por ceros y viceversa).
2) Obtener la forma MPRM óptima de $\bar{f}(x)$ y sumarle 1.
3) Minimizar la expresión obtenida.

El pseudocódigo correspondiente puede verse en el Algoritmo 4.20 donde:

- *Expresión_inicial()* es la función que obtiene la expresión inicial AND-EXOR, y que correspondería al organigrama de la Figura 4.16.
- *Control_ES()* es la función que se encarga de minimizar la expresión incial AND-EXOR proporcionada. Se ha detallado anteriormente en el Algoritmo 4.19.

Finalmente, en la Figura 4.18 se presenta el organigrama de este algoritmo.

4.9. OPTIMIZACIÓN DE FUNCIONES INCOMPLETAMENTE ESPECIFICADAS

En general, las funciones de conmutación correspondientes a sistemas reales contienen indiferencias. En el contexto del Álgebra de Boole, es posible arrastrar estas indiferencias a través de los métodos de minimización usuales (Mapas de Karnaugh, Procedimiento de Quine-McCluskey) y efectuar su asignación de la manera más conveniente al final de todo el proceso. En los procedimientos de minimización AND-EXOR no es posible, salvo en algunos casos, efectuar este arrastre, y es necesario efectuar una asignación previa de las indiferencias.

Algoritmo 4.20: Minimización de funciones con más de un 50% de unos

```
Minimizar_funcion()
        i=0;  nunos=0;  comple=0;
        Mientras (i<maxmin) hacer
        Si (M[i]=1) nunos++;   Fin Si
        Si (nunos>2^(n-1))
                comple=1;
                i=0;
                Mientras(i<maxmin) hacer
                        Si (M[i]=1) M[i]=0;  Fin Si
                        Otro
                                Si (M[i]=0) M[i]=1;  Fin Si
                        Fin Otro
                Fin Mientras
        Fin Si
        Expresión_inicial();
        Si (comple=1) L[0]++;  Fin Si
        Control_ES();
Fin Minimizar_funcion
```

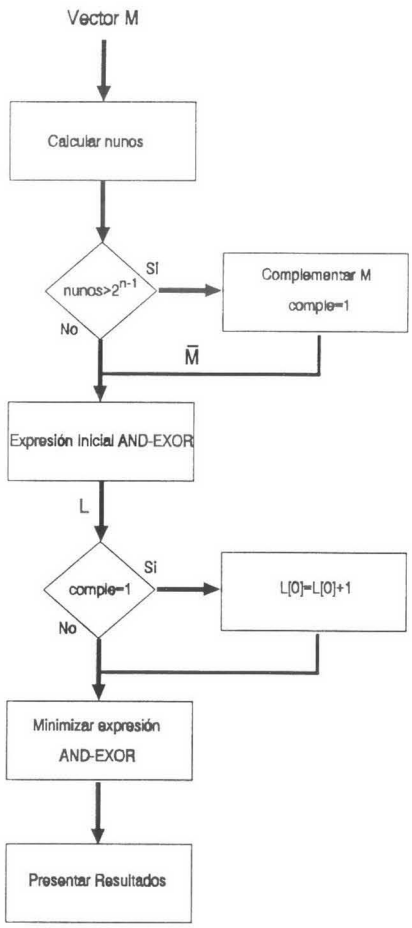


Fig. 4.18. Minimización de funciones con más de un 50% de unos.

En el caso concreto de RRMIN2, claramente resulta inviable el mantener las indiferencias sin asignar durante las transformaciones previas (al calcular la mejor forma MRPM) o el proceso de Enfriamiento Simulado. Teniendo en cuenta estas consideraciones existen cuatro posibilidades para asignar valores a las indiferencias:

a) Asignar valores aleatorios a las indiferencias.
b) Asignar valores prefijados a las indiferencias (por ejemplo ponerlas todas a 0).
c) Dar los 2^d (d es el número de indiferencias de la función) posibles valores distintos a las indiferencias, minimizar las 2^d funciones que se obtendrían y escoger entre ellas las que tengan una expresión más simple (esto constituiría un procedimiento exacto para la asignación de indiferencias).
d) Intentar construir un algoritmo heurístico que asigne valores a las indiferencias a partir de la información conocida de la función (es decir, buscar un procedimiento aproximado para la asignación de las indiferencias).

Los métodos a) y b) pueden utilizarse únicamente cuando no importe realmente el obtener una expresión mínima de la función y, evidentemente, no proporcionan ninguna fiabilidad ni garantizan buenos resultados.

El procedimiento c) sólo es aplicable para funciones que contengan un número muy pequeño de indiferencias, ya que en caso contrario, el tiempo preciso para realizar todas las operaciones está fuera del alcance de los ordenadores actuales.

Por tanto, el método más útil y factible desde el punto de vista de los resultados y tiempo preciso para llegar a ellos, es el procedimiento d), es decir, tratar de encontrar un buen procedimiento aproximado.

Aunque el procedimiento sea aproximado, siempre debe haber algún tipo de realimentación entre el resultado obtenido y la asignación efectuada (resulta imposible encontrar una buena asignación a priori, sin realizar ningún tipo de verificación del resultado que se obtiene), de forma que será preciso realizar minimizaciones de las funciones que resulten de efectuar las asignaciones de indiferencias. Si se realizan los procesos de minimización completos, se va a necesitar un gran tiempo de ejecución, por lo que se ha optado por efectuar la realimentación únicamente con las formas MPRM. Así, se van a efectuar dos aproximaciones:

1) En primer lugar no se van a efectuar todas las asignaciones posibles de las indiferencias, sino únicamente algunas.

2) En segundo lugar, al realizar la realimentación con las formas MPRM, y no sobre todo el procedimiento RRMIN2, se está optimizando la asignación de las indiferencias para obtener la forma MPRM más reducida, lo que no implica que esa asignación sea la mejor para obtener el mejor resultado a través de todo el proceso de minimización.

En definitiva, se está buscando un procedimiento para la asignación óptima de indiferencias dentro de la familia de formas canónicas MPRM. Existen varios procedimientos aproximados de asignación de indiferencias para procedimientos basado en formas canónicas de Reed-Muller, como por ejemplo los que aparecen en [MCK93], [VAR91], [HAR90] ó [GRE87], sin embargo todos ellos van encaminados a obtener la mejor forma RM. En las siguientes secciones se desarrolla un procedimiento aproximado para la asignación de indiferencias dentro de las MPRM.

4.9.1. ASIGMIN: Un procedimiento aproximado para la asignación de indiferencias

El procedimiento que se va a presentar parte de la expresión booleana de la función de conmutación que se desea minimizar. Tal función vendrá representada por un vector verdad M (o sea, como suma de minterms), en el que cada componente podrá tomar tres valores: 0, 1 y 2. Un valor 0 en una componente significará que el minterm no aparece en la función, un 1 que sí aparece, y un 2 indicará que se trata de una indiferencia.

El punto de partida de los algoritmos, será por tanto un vector de la forma:

$$M=[m_0 \;\; m_1 \;\; \ldots \;\; m_{2^n-1}] \tag{4.128}$$

donde $m_i \in \{0,1,2\}$ y n es el número de variables. El proceso de asignación de indiferencias consta de dos fases: En la primera se realiza una primera asignación de las indiferencias para llegar a una función que se denominará función base. En la segunda fase, se introducirá una serie de cambios en la función base, obteniendo varias funciones, entre las que se elegirá la que proporcione una solución más simple.

A continuación, para ilustrar el porqué de las operaciones que se realizan en los algoritmos y facilitar la comprensión de los mismos, va a estudiarse el caso más simple de las funciones de dos variables expresadas sobre GF(2) utilizando sólo su forma RM.

Considérese una función booleana de dos variables:

$$f=d_0m_0+d_1m_1+d_2m_2+d_3m_3 \tag{4.129}$$

Según se vio en el Apartado 2.3, si esta función se pasa a GF(2) usando su forma RM, se obtiene:

$$f=g_0\oplus g_1x_1\oplus g_2x_2\oplus g_3x_2x_1 \tag{4.130}$$

donde

$$\begin{aligned} g_0&=m_0\\ g_1&=m_0\oplus m_1\\ g_2&=m_0\oplus m_2\\ g_3&=m_0\oplus m_1\oplus m_2\oplus m_3 \end{aligned} \tag{4.131}$$

Analizando las expresiones de (4.131), puede verse que si se garantizan las igualdades:

$$m_0=m_2 \qquad m_1=m_3 \tag{4.132}$$

automáticamente g_2=0 y g_3=0, en cuyo caso sólo se tendrían como mucho 2 términos producto en la expresión RM. Además, si se cumple (4.132), en ningún caso se obtiene mejor resultado al asignar las indiferencias de otra forma. Seguidamente se ilustrará esta afirmación.

Existen sólo 4 funciones de 2 variables que cumplan (4.132), que se enuncian a continuación indicando el número de productos necesario para su síntesis RM:

$$[\ 0\ 0\ 0\ 0\ ] \rightarrow 0\ \textit{productos} \tag{4.133}$$
$$[\ 0\ 1\ 0\ 1\ ] \rightarrow 1\ \textit{producto} \tag{4.134}$$
$$[\ 1\ 0\ 1\ 0\ ] \rightarrow 2\ \textit{productos} \tag{4.135}$$
$$[\ 1\ 1\ 1\ 1\ ] \rightarrow 1\ \textit{producto} \tag{4.136}$$

De la función [0 0 0 0] no es necesario ocuparse porque no existe ninguna posibilidad mejor. La función (4.134), si proviene de una función con una indiferencia, puede ser:

$$\begin{matrix}[\ 2\ 1\ 0\ 1\ ]\\ [\ 0\ 2\ 0\ 1\ ]\\ [\ 0\ 1\ 2\ 1\ ]\\ [\ 0\ 1\ 0\ 2\ ]\end{matrix} \tag{4.137}$$

Las otras posibilidades de asignación, distintas a la de (4.132) serán por tanto:

$$\begin{matrix}[\ 1\ 1\ 0\ 1\ ]\\ [\ 0\ 0\ 0\ 1\ ]\\ [\ 0\ 1\ 1\ 1\ ]\\ [\ 0\ 1\ 0\ 0\ ]\end{matrix} \tag{4.138}$$

Ninguna de estas cuatro asignaciones corresponde a la función [0 0 0 0], de forma que todas precisarán al menos de un témino producto, y por consiguiente no mejoran la síntesis (4.134). Exactamente lo mismo ocurre con la función (4.136), y sólo queda alguna duda en el caso de (4.133). Esta última función puede provenir, en caso de tener una indiferencia, de las siguientes 4 funciones:

$$\begin{matrix}[\ 1\ 0\ 2\ 0\ ]\\ [\ 1\ 0\ 1\ 2\ ]\\ [\ 2\ 0\ 1\ 0\ ]\\ [\ 1\ 2\ 1\ 0\ ]\end{matrix} \tag{4.139}$$

El número de productos necesario si no se cumple (4.132) será en cada caso:

$$\begin{matrix}[\ 1\ 0\ 0\ 0\ ] \rightarrow 3\ \textit{productos}\\ [\ 1\ 0\ 1\ 1\ ] \rightarrow 3\ \textit{productos}\\ [\ 0\ 0\ 1\ 0\ ] \rightarrow 2\ \textit{productos}\\ [\ 1\ 1\ 1\ 0\ ] \rightarrow 2\ \textit{productos}\end{matrix} \tag{4.140}$$

Como puede verse, en ninguna ocasión se mejora la asignación (4.134).

Nótese ahora que al tratar de garantizar (4.132) pueden ocurrir tres cosas:

a) Que se consiga sin mayores problemas.
b) Que se tengan indiferencias en m_0 y m_2 ó en m_1 y m_3.
c) Que no sea posible.

En caso de darse a) se habrá acabado el proceso de síntesis de la función base. Si se están en los casos b) ó c) se procederá a tratar de conseguir:

$$m_0=m_1 \qquad m_2=m_3 \tag{4.141}$$

Es decir, lo que se está haciendo para llegar a la función base es, de forma gráfica:

- Dividir M por la mitad:

$$M=[\ \textit{Parte}\ 1\ |\ \textit{Parte}\ 2\ ] \tag{4.142}$$

- Igualar *Parte* 1 y *Parte* 2. En caso de no conseguirse por ocurrir b) ó c), se vuelve a subdivir cada una de las dos partes:

$$M=[\ \textit{Parte}\ 1.1\ |\ \textit{Parte}\ 1.2\ |\ \textit{Parte}\ 2.1\ |\ \textit{Parte}\ 2.2\ ] \tag{4.143}$$

y se procede a igualar Parte 1.1 con Parte 1.2 y Parte 2.1 con Parte 2.2. En caso de tener una función de más de dos variables el proceso de subdivisión e igualación continuaría hasta que no fuera posible volver a partir la función.
- Si aún quedan indiferencias sin asignar, se asignarían a 0 si la función tiene menos de un 50% de unos, y a 1 en caso contrario.

Continuando con el ejemplo que se estaba tratando, en el caso de tener dos indiferencias y garantizar (4.132), también se obtiene un resultado óptimo: con las funciones (4.132) y (4.136) resulta evidente, mientras que en el caso de (4.134) las posibilidades son:

$$\begin{matrix}[\ 0\ 1\ 2\ 2\ ]\\ [\ 2\ 2\ 0\ 1\ ]\end{matrix} \tag{4.144}$$

y los términos productos necesarios:

$$\begin{matrix}[\ 0\ 1\ 0\ 0\ ] \rightarrow 2\ \textit{productos}\\ [\ 0\ 1\ 1\ 0\ ] \rightarrow 2\ \textit{productos}\\ [\ 0\ 1\ 1\ 1\ ] \rightarrow 3\ \textit{productos}\end{matrix} \tag{4.145}$$

con lo que no se mejora la asignación de (4.134). Finalmente, en el caso de 4.130, las posibilidades son:

$$\begin{matrix}[\ 2\ 2\ 1\ 0\ ]\\ [\ 1\ 0\ 2\ 2\ ]\end{matrix} \tag{4.146}$$

y las posibles asignaciones distintas a (4.135) serán:

$$\begin{array}{l}[\,0\;0\;1\;0\,] \rightarrow 2\ \textit{productos} \\ [\,0\;1\;1\;0\,] \rightarrow 2\ \textit{productos} \\ [\,1\;0\;0\;0\,] \rightarrow 3\ \textit{productos} \\ [\,1\;0\;1\;1\,] \rightarrow 3\ \textit{productos}\end{array} \qquad (4.147)$$

En resumen, si se consigue garantizar (4.132), se obtiene una asignación óptima para las indiferencias (el caso de 3 indiferencias no se analiza por no tener sentido para funciones con dos variables; siempre se obtendría [0 0 0 0] ó [1 1 1 1]).

Por supuesto esto es sólo un caso particular; en cuanto se tiene un número mayor de variables el problema de la asignación se complica y no siempre se consigue un resultado óptimo. Sin embargo, el ejemplo sirve para mostrar el fundamento del algoritmo de obtención de la función base: se trata de aprovechar la propiedad de paridad de la operación EXOR para eliminar el mayor número de términos producto en la expresión sobre GF(2).

Incluso en el caso de 2 variables, si no se garantiza (4.132) y se pasa por tanto a tratar de obtener (4.141), no se consigue siempre la mejor asignación. Considérese por ejemplo la función:

$$[\,1\;0\;0\;2\,] \qquad (4.148)$$

Aplicando el algoritmo de obtención de la función base queda:

$$[\,1\;0\;0\;0\,] \rightarrow 4\ \textit{productos} \qquad (4.149)$$

en cambio, existe una asignación mejor:

$$[\,1\;0\;0\;1\,] \rightarrow 3\ \textit{productos} \qquad (4.150)$$

Esto se debe a que se está aplicando un algoritmo heurístico que trata de optimizar la asignación de indiferencias del mayor número de casos posibles pero con un pequeño número de operaciones. Por tanto, siempre quedan algunos casos particulares que el algoritmo no cubre. Para tratar de cubrir algunos de estos casos particulares, se utiliza un segundo algoritmo para optimizar la función base, que consiste en suponer que dicha función es una buena aproximación de la asignación óptima, y simplemente cambiando la asignación de una de las indiferencias se llega al resultado buscado. Por tanto, lo que se hace es cambiar la asignación de la primera indiferencia, minimizar sobre GF(2), volver a la función base original, cambiar la segunda indiferencia, minimizar ... y así sucesivamente. De esta forma, por ejemplo, se detectaría que la asignación (4.149) es mejor que la (4.150).

Por último, para funciones con un número de variables mayor, puede ocurrir que sea contraproducente realizar demasiadas subdivisiones, debido a que pueden introducirse demasiados unos. Para controlar este problema se define el parámetro de profundidad, que viene dado por:

par_profundidad=nº de componentes que tiene cada parte en la última subdivisión (4.151)

Es decir, con el parámetro de profundidad se obliga a que se produzcan subdivisiones hasta que cada parte tenga 1, 2, 4, ó cualquier otro número de componentes, siempre que sea potencia de 2 y menor o igual que 2^{n-1}, siendo n el número de variables. En los dos apartados siguientes puede verse ya una definición formal de los dos algoritmos descritos aquí.

4.9.1.1. Cálculo de la función base

En este apartado se va a efectuar el cálculo de la función base. Para ello se parte de la representación descrita en 4.9.1. En primer lugar se fija el denominado parámetro de profundidad, ($1 \leq par\_profundidad \leq 2^{n-1}$, siendo *par_profundidad* potencia de 2). A continuación se divide el vector M en dos partes iguales y se comprueba si es posible asignar valores a las indiferencias de manera que ambas mitades sean idénticas. En caso afirmativo, se procede a efectuar la asignación de las mismas. En caso negativo, no se asignan valores a las indiferencias. Si existiera el problema de que en la misma coordenada de las dos partes se tuvieran indiferencias, éstas se dejarían sin asignar por el momento. Seguidamente, cada una de las partes en que se ha dividido M vuelve a dividirse en dos partes iguales con lo que se tendrán en total 4 partes; éstas se comparan de dos en dos, de manera que ambas provengan de la división de una misma parte. Nuevamente se trata de igualarlas efectuando asignaciones a las indiferencias si ello es posible. El proceso se repite hasta que el número de componentes de las partes obtenidas sea igual al parámetro de profundidad previamente fijado. Finalmente, las indiferencias que hayan quedado sin asignar, se ponen a 0.

El parámetro de profundidad que se ha mencionado anteriormente define el número de veces que se produce el proceso de subdivisión, lo que permite controlar en cierta manera la cantidad de unos que se introducen en la asignación de las indiferencias. Si se efectúan pocas subdivisiones hay más probabilidad de que se llegue al fin del proceso con indiferencias sin asignar, y por tanto se pondrán automáticamente a 0. Por el contrario, si se elige un parámetro de profundidad bajo, habrá una probabilidad mayor de que todas las indiferencias queden asignadas, y por tanto puede haber un mayor número de unos en la expresión de la función base. Dependiendo del tipo de función que se tenga, convendrá elegir un parámetro de profundidad mayor o menor para conseguir una mejor optimización. Todo esto se verá en detalle en el Capítulo 6 al analizar los resultados experimentales (Apartado 6.4.1).

En el Algoritmo 4.21 se presenta el pseudocódigo para el cálculo de la función base. Las funciones utilizadas se describen en los algoritmos 4.22, 4.23 y 4.24.

Para ilustrar este algoritmo se presenta el siguiente ejemplo. Considérese la siguiente función de tres variables, y tómese *par_profundidad*=2:

$$M=[2\ 0\ 1\ 2\ 2\ 0\ 1\ 1] \qquad (4.152)$$

Algoritmo 4.21: Cálculo de la Función Base

```
1)  tamaño_trozo=2^(n-1).
    número_trozos=2.
2)  Mientras (tamaño_trozo≥par_profundidad) hacer
        trozo=0.
        Mientras (trozo<número_trozos) hacer
            Primer trozo(trozo*tamaño_trozo,tamaño_trozo).
            Segundo_trozo((1+trozo)*tamaño_trozo,tamaño_trozo).
            Igualar(M1,M2,tamaño_trozo).
            trozo=trozo+2.
        Fin Mientras.
        número_trozos=número_trozos*2.
        tamaño_trozo=tamaño_trozo/2.
    Fin Mientras.
3)  Poner a cero todas las indiferencias no asignadas si n_unos≤2^(n-1). En otro caso, asignar a 1
    dichas indiferencias.
4)  Fin del procedimiento.
```

Algoritmo 4.22: Función Primer_trozo

```
Primer_trozo(primera_coordenada,num_coordenadas)
i=0.
Mientras (i<num_coordenadas) hacer
        M1[i]=M[primera_coordenada+i].
        i=i+1.
Fin Mientras
Fin Primertrozo.
```

Algoritmo 4.23: Función Segundo_trozo

```
Segundo_trozo(primera_coordenada,num_coordenadas).
i=0.
Mientras (i<num_coordenadas) hacer
        M2[i]=M[primera_coordenada+i].
        i=i+1.
Fin Mientras.
Fin Segundo_trozo.
```

Algoritmo 4.24: Función Igualar

```
Igualar(M1,M2,tamaño_trozo).
i=0.
Mientras (i<tamano_trozo) hacer
        Si (M1[i]≠2 y M2[i]≠2 y M1[i]≠M2[i])
                Fin Igualar.
        Fin Si.
        i=i+1.
Fin Mientras.
i=0.
Mientras(i<tamaño_trozo) hacer
        Si (M1[i]=2 y M2[i]≠2)
                M1[i]=M2[i]
        Fin Si.
        Si (M2[i]=2 y M1[i]≠2)
                M2[i]=M1[i]
        Fin Si.
        i=i+1
Fin Mientras.
Fin Igualar.
```

En primer lugar se divide el vector M en dos partes:

$$M=[2\ \ 0\ \ 1\ \ 2\ \ |\ \ 2\ \ 0\ \ 1\ \ 1] \tag{4.153}$$

A continuación se trata de igualar ambas partes. En este caso es posible hacerlo asignando un 1 a la indiferencia colocada en la componente m_3. Véase que las indiferencias colocadas en m_0 y m_4 quedan sin asignación en este paso. Por tanto, queda:

$$M=[2\ \ 0\ \ 1\ \ 1\ \ |\ \ 2\ \ 0\ \ 1\ \ 1] \tag{4.154}$$

Seguidamente, cada parte se subdivide en dos partes:

$$M=[2\ \ 0\ \ |\ \ 1\ \ 1\ \ |\ \ 2\ \ 0\ \ |\ \ 1\ \ 1] \tag{4.155}$$

En este caso, comparando dos a dos las partes que provienen de una misma parte del paso anterior, se observa que no es posible igualarlas, quedando por tanto las indiferencias sin asignar. Puesto que el número de componentes de cada una de las partes es igual al parámetro de profundidad, ya no se vuelven a producir más subdivisiones, con lo que se procede a sintetizar como ceros las indiferencias que han quedado (se tienen solamente 4 unos):

$$M=[0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1] \tag{4.156}$$

El vector así obtenido es el vector base que se utilizará como punto de partida en el Apartado siguiente.

4.9.1.2. Optimización de la función base

En este algoritmo se parte de la función base generada en el Algoritmo 4.21. En esta segunda fase del procedimiento de asignación de las indiferencias, se presupone que la función base obtenida supone una buena asignación para las indiferencias, y que la solución óptima está próxima a ella. Concretamente, se supone que la solución óptima para este procedimiento puede obtenerse cambiando la asignación de una sóla de las indiferencias. Por tanto, en primer lugar se realizará la minimización AND-EXOR de la función base utilizando cualquiera de los procedimientos que existen para ello. Después se complementará la asignación que se haya hecho de la primera indiferencia y se volverá a minimizar. A continuación se pondrá la asignación de la indiferencia a su valor original y se repetirá el proceso con todas las indiferencias. Finalmente, se elige la asignación de indiferencias que dé mejor solución al minimizar sobre GF(2).

Como puede verse, utilizando este procedimiento únicamente hay que realizar d minimizaciones (d es el número de indiferencias que contiene la función a minimizar) en lugar de las 2^d que son necesarias con el procedimiento exacto (Ver posibilidad c del Apartado 4.9).

Escrito en pseudocódigo, si se denomina *MB* al vector que representa a la función base, d al número de indiferencias y *pos_indif* al vector conteniendo las posiciones de las indiferencias, resulta el Algoritmo 4.25. Una vez aplicado este algoritmo, se obtiene la asignación de las indiferencias y la expresión mínima de la función con esa asignación. Obsérvese que estos algoritmos son completamente generales y pueden aplicarse conjuntamente con cualquier procedimiento de minimización AND-EXOR. En el caso de RRMIN2, los algoritmos presentados se van a aplicar solamente sobre el proceso de obtención de la forma óptima MPRM, con el objeto de ahorrar tiempo de ejecución. En la Figura 4.19 puede verse el organigrama correspondiente.

En el apartado siguiente se detalla el procedimiento que se utiliza en RRMIN2 para realizar la minimización de funciones con varias salidas (síntesis multifuncional).

4.10. SÍNTESIS MULTIFUNCIONAL CON RRMIN2

En este apartado se especifica la forma en que RRMIN2 realiza la minimización de funciones con varias salidas. En general, el problema de la minimización de funciones con varias salidas puede abordarse de tres formas:

1) Minimizando por separado cada una de las salidas.
2) Minimizando todas las salidas simultaneamente.

Algoritmo 4.25: Optimización de la función base

1) *i=0.*
Mientras (i<d) Hacer
Complementar MB[pos_indif[i]].
Minimizar MB
Complementar MB[pos_indif[i]].
Fin mientras.
2) *Elegir la asignación que proporcione la mejor síntesis de la función.*
3) *Fin del procedimiento.*

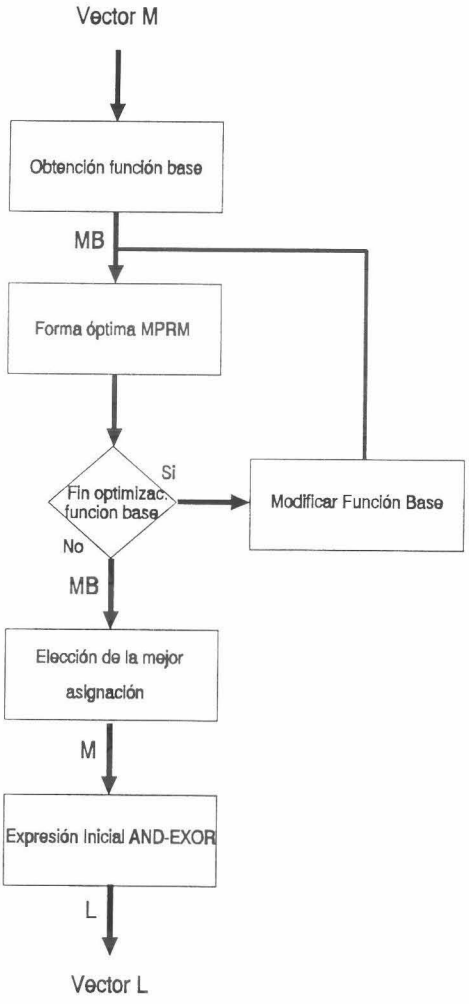


Fig. 4.19. Asignación de indiferencias en RRMIN2.

3) Combinando las posibilidades 1) y 2), es decir, minimizando en primer lugar las salidas por separado y posteriormente, y partiendo del resultado obtenido, efectuar la minimización simultánea de todas las salidas.

El procedimiento 1) es de aplicación inmediata si se dispone de un procediminetno de minimización unifuncional, aunque evidentemente proporciona peores resultados que el 2). El segundo procedimiento proporciona mejores resultados pero supone una mayor complejidad conceptual y un mayor costo de tiempo de CPU. Para mejorar el tiempo de CPU se puede utilizar la opción 3), que permite rebajar de forma rápida el número de productos mediante una minimización individual y posteriormente minimizar las salidas simultaneamente.

En RRMIN2 se ha optado por utilizar por defecto el procedimiento 3), puesto que proporciona mejores prestaciones y precisa de un tiempo de CPU razonable. No obstante, el usuario puede elegir entre cualquiera de las tres opciones.

En consecuencia, el proceso de minimizar una función de varias salidas con RRMIN2 consta de dos partes:

a) Minimización individual de cada salida.
b) Minimización simultánea de todas las salidas, partiendo del resultado obtenido tras la aplicación de a).

Para aplicar la parte a) no hay que efectuar modificaciones apenas, simplemente hay que repetir el proceso de minimización descrito en los apartados anteriores para cada salida. La parte b) sí requiere de algunas modificaciones y de la reestructuración de algunos algoritmos, y se va a detallar en las secciones posteriores.

Puesto que en RRMIN2 se efectúan dos procesos de minimización, uno preliminar dentro de las formas MPRM y otro mediante reglas de reescritura, se ha dividido la minimización simultánea en dos fases:

- Realización de síntesis multifuncional al obtener la forma óptima MPRM (este proceso sólo es necesario en caso de que se elija la opción 2. En las otras dos opciones sólo es necesario minimizar una sóla salida MRPM)
- Minimización multifuncional durante el proceso de Enfriamiento simulado.

Estas dos cuestiones se abordan en los apartados que siguen.

4.10.1. Minimización simultánea de varias salidas dentro de las MPRM

Tal y como se ha mencionado en 4.10, el proceso de minimización simultánea de varias salidas requiere de algunas modificaciones. En concreto, habrá que tener en cuenta en las funciones de coste los términos producto que son compartidos por varias funciones. Para ello, basta tener presente que el vector extendido E contiene los términos producto que hay que sintetizar. Así, si se efectúa la operacion OR con los *n_func* vectores E (*n_func* es el

número de salidas a sintetizar) para obtener un vector extendido del sistema global *EG*, se estará considerando el hecho de que existen términos producto compartidos (en efecto, al efectuar la operación OR se están marcando los términos producto que se utilizan al menos por una función, pero sólo se tienen en cuenta una vez, aunque los utilicen varias funciones). Por lo demás, el proceso sigue siendo el mismo, se obtiene el vector peso *ST* a partir del *EG*, se escoge la polaridad que supone un mínimo coste, y se calcularía la forma MPRM de polaridad *m* para cada una de las *n_func* salidas. Una vez calculadas las formas MPRM, se obtendrían los *n_func* vectores *L* correspondientes.

En resumen, el proceso es exactamente el mismo, sólo que repetido para cada una de las *n_func* funciones. La única diferencia estriba en que la polaridad mínima *m* se ha obtenido considerando cuál sería el mínimo coste para todas las salidas consideradas globalmente. En la Figura 4.20 puede verse el diagrama de flujo correspondiente.

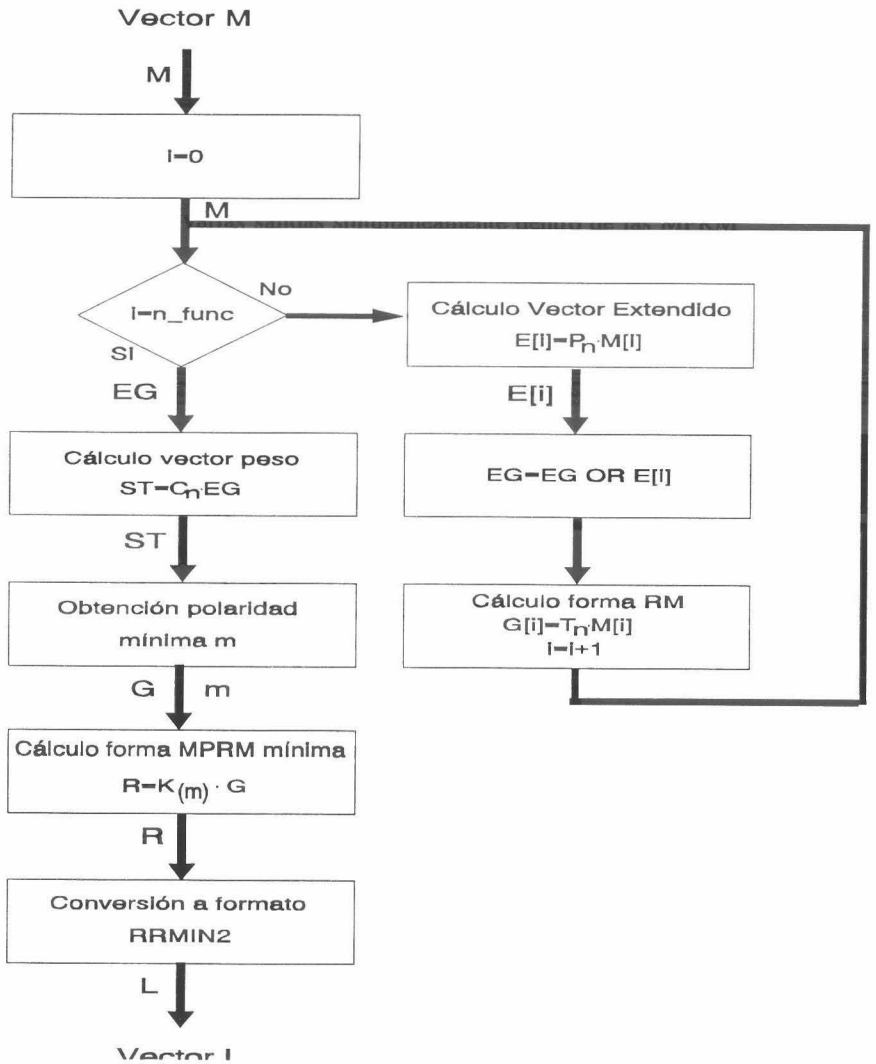


Fig. 4.20. Minimización de varias salidas simultáneamente dentro de las MPRM

Nótese que en la Figura 4.20 se ha utilizado la notación *M[i]* para refererise al vector M correspondiente a la función *i*. En los algoritmos que se van a presentar en los apartados posteriores, la coordenada *j* de la función *i* vendrá dada por *M[i*maxmin+j]*. Asimismo, la coordenada *j* del vector *L* correspondiente a la función *i* será *L[i*maxcor+j]*.

4.10.2. Minimización multifuncional a través del proceso de Enfriamiento Simulado

Una vez que se han obtenido los vectores *L*, éstos tendrán que someterse al proceso de Enfriamiento Simulado descrito en 4.5. En principio, bastará con modificar la función de coste de manera que se tenga en cuenta al sistema global. Esta nueva función de coste, en lugar de tener en cuenta todos los términos producto de todos los vectores *L*, considerará únicamente el mayor valor de la componente *i* (recuérdese que cada componente corresponde a un término producto) de todos los vectores *L*, de manera que, al igual que ocurría en el Apartado 4.7.1, se va a tener en cuenta el hecho de que algún término producto sea compartido por varias funciones. En el Algoritmo 4.26 se describe cómo quedaría el algoritmo del cálculo de la función de coste en el caso de tener una función con varias salidas.

Otra modificación que hay que realizar corresponde a la función encargada de generar el nuevo estado aplicando reglas de reescritura al estado anterior. Ésta función, *Generar(L,L2)*, descrita en el Algoritmo 4.13, deberá ahora de aplicar una regla a cada una de las salidas, quedando en la forma que se presenta en el Algoritmo 4.27. En este algoritmo puede observarse que las funciones de aplicación de las reglas incluyen una nueva variable de entrada que les indica la función sobre la que han de aplicar las reglas. De esta forma, sería necesario modificar las funciones *Aplicar_1, Aplicar_2, Aplicar_3* y *Aplicar_4*, descritas en los algoritmos 4.5, 4.7, 4.9 y 4.11. Las nuevas funciones no se van a escribir nuevamente puesto que las modificaciones son muy simples; unicamente hay que cambiar el acceso a las coordenadas del vector *L*, sustituyendo *L[i]* por *L[i+indice2*maxcor]*.

Por otra parte, si se tiene en cuenta que la opción por defecto es realizar una minimización previa de las salidas individualmente, el estado del cual se va a partir en el proceso de Enfriamiento Simulado de la multifunción puede ser un estado de baja energía. En esa situación, puede ser interesante partir de una temperatura más baja que la que se utilice para la optimización de las salidas individuales, y para ello, se va a introducir un nuevo parámetro denominado *t_div*. El citado parámetro, lo único que va a hacer es dividir a la temperatura inicial c_0, el incremento inicial de temperatura Δ_0 y a la temperatura final c_f, de manera que se realizará un proceso de enfriamiento con el mismo número de iteraciones, pero en un rango de temperaturas más bajas. En 6.5.1 puede verse un estudio experimental sobre este parámetro.

Finalmente, en la Figura 4.21 puede verse el organigrama completo para la minimización de funciones con varias salidas, según se escoja la opción 1, la 2 o la 3. Con ello, queda completamente descrito el procedimiento de minimización RRMIN2, incluyendo la minización de funciones incompletamente especificadas y con varias salidas. En el próximo capítulo se desarrolla una versión de RRMIN2 para la ejecución del algoritmo de

minimización en sistemas multiprocesador (PRRMIN). El comportamiento experimental de estos dos procedimientos se presenta en el Capítulo 6.

4.11. CONCLUSIONES

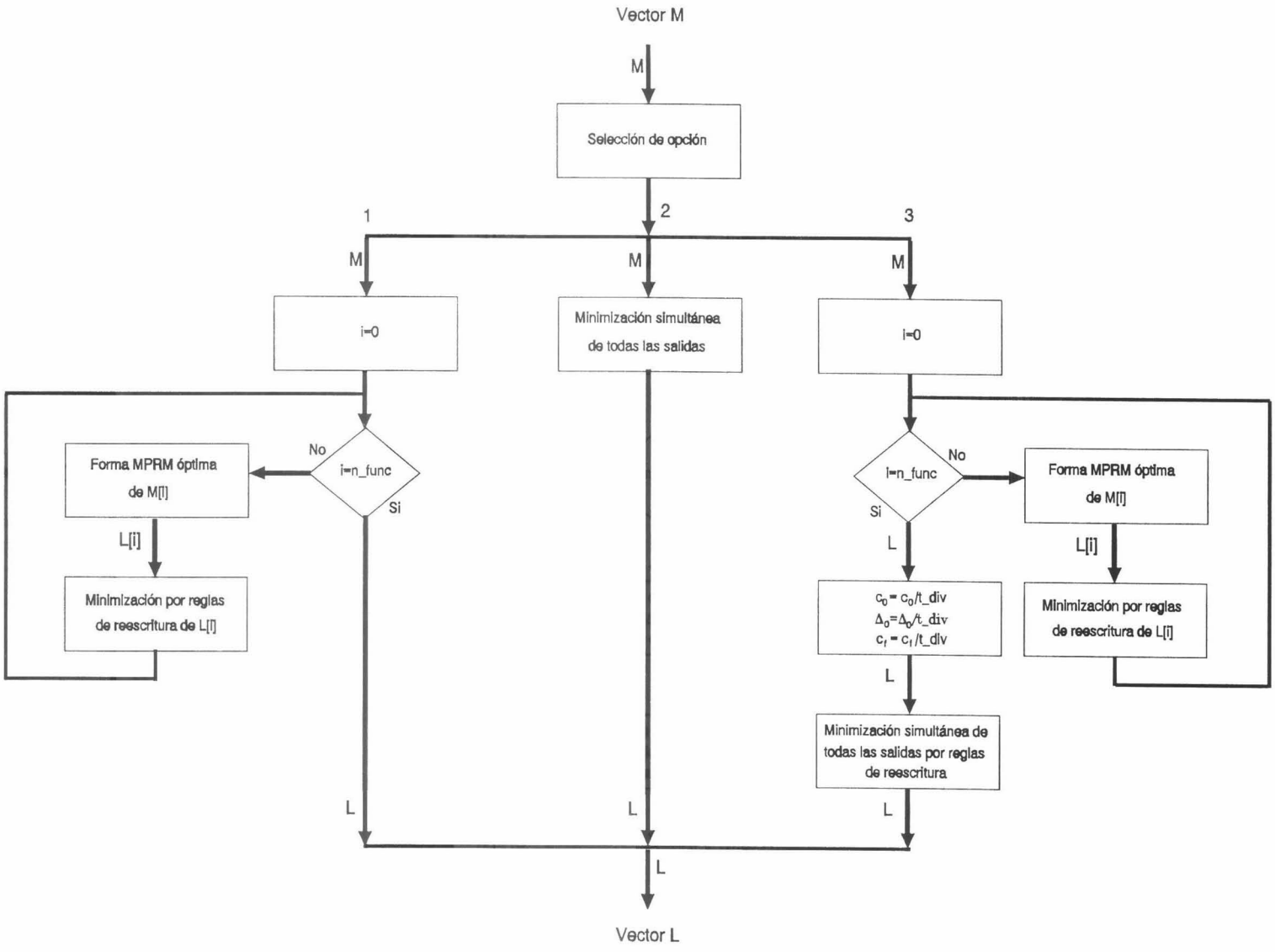
En este Capítulo se ha desarrollado un nuevo procedimiento para la minimización AND-EXOR de funciones de conmutación. El procedimiento, denominado RRMIN2 permite la minimización de funciones completa e incompletamente especificadas, y con varias salidas. Para ello, efectúa previamente una minimización dentro de la familia de formas canónicas MPRM mediante un conjunto de algoritmos rápidos que se han desarrollado en el Apartado 4.5. La aplicación de estos algoritmos rápidos permite obtener la forma óptima MPRM en unos tiempos de ejecución muy reducidos.

Algoritmo 4.26. Cálculo de la función de coste para funciones con varias salidas

```
Costo_L(L)
n_productosL=0;
n_litL=0;
indice=0;
Mientras(indice<maxcor) hacer
        temporal=0;
        indice2=0;
        Mientras(indice2<nfunc) hacer
                Si (L[indice2*maxcor+indice] > temporal)
                        temporal=L[indice2*maxcor+indice];
                Fin Si
                Si (temporal)
                        n_productosL=n_productosL+temporal;
                        modulo=indice;
                        indice3=0;
                        Mientras(indice3<n) hacer
                                Si (modulo % 3)
                                        n_litL=n_litL+temporal;
                                        modulo=modulo/3;
                                Fin Si
                                indice3=indice3+1;
                        Fin Mientras
                Fin Si
                indice2=indice2+1;
        Fin Mientras
        indice=indice+1
Fin Mientras
Fin Costo_L
```

Algoritmo 4.27: Selección de los grupos de reglas en funciones con varias salidas

```
Generar(L,L2)
        indice2=0;
        Mientras(indice2<nfunc) hacer
                error=0;
                n_posiciones=0;
                nmasdos=0;
                indice=0;
                tempporal=indice2*maxcor
                Mientras (indice<3^n) hacer
                        Si (L2[indice+temporal]) n_posiciones=n_posiciones+1;
                        Fin Si
                        Si (L2[indice+temporal] >=2) nmasdos=nmasdos+1;
                        Fin Si
                        indice=indice+1;
                Fin Mientras;
                Si (n_posiciones<2) Devolver error;
                Fin Si
                Si (tramo=1) p_grupo1=0; p_grupo2=1; p_grupo3=0; p_grupo4=0;
                Fin Si
                Si (tramo=2) p_grupo1=p_grupo1i; p_grupo2=p_grupo2i;
                        p_grupo3=p_grupo3i; p_grupo4=1-p_grupo1-p_grupo2-p_grupo3;
                Fin Si
                n_grupo=1;
                aleat=aleatorio();
                aleat=aleat-p_grupo1;
                Si (aleat>0) n_grupo=n_grupo+1;
                Fin Si
                aleat=aleat-p_grupo2;
                Si (aleat>0) n_grupo=n_grupo+1;
                Fin Si
                aleat=aleat-p_grupo3;
                Si (aleat>0) n_grupo=n_grupo+1;
                Fin Si
                Si (n_grupo=1) error=Aplicar_1(L,L2,nmasdos,indice2);
                Fin Si
                Si (n_grupo=2) error=Aplicar_2(L,L2,n_posiciones,indice2);
                Fin Si
                Si (n_grupo=3) Aplicar_3(L,L2,n_posiciones,indice2);
                Fin Si
                Si (n_grupo=4) Aplicar_4(L,L2,indice2);
                Fin Si
                Devolver error;
                indice2=indice2+1;
        Fin Mientras
Fin Generar
```


Vector M
M
Selección de opción
1
2
3
M
M
M
i=0
Minimización simultánea de todas las salidas
i=0
No
i=n_func
Si
Forma MPRM óptima de M[i]
L[i]
Minimización por reglas de reescritura de L[i]
i=n_func
No
Si
Forma MPRM óptima de M[i]
L[i]
Minimización por reglas de reescritura de L[i]
L
$c_0 = c_0/t\_div$
$\Delta_0 = \Delta_0/t\_div$
$c_f = c_f/t\_div$
L
Minimización simultánea de todas las salidas por reglas de reescritura
L
L
L
L
Vector L

Posteriormente, sobre esta forma inicial AND-EXOR se aplica un conjunto convergente de reglas de reescritura mediante un proceso de Enfriamiento Simulado, lo que garantiza que utilizando un tiempo de ejecución suficientemente grande, se es capaz de obtener el mínimo exacto.

Para la minimización de funciones incompletamente especificadas, se ha desarrollado un procedimiento aproximado para la asignación de las indiferencias, ASIGMIN, que precisa de muy poco tiempo de ejecución y proporciona resultados muy aceptables.

Finalmente, para abordar la minimización de funciones con varias salidas, se utiliza una combinación de los dos procedimientos usuales: en primer lugar se realiza una minimización individual para cada salida, y después, partiendo del resultado obtenido se efectúa una minimización simultánea de las mismas. De esta forma se consiguen resultados tan buenos como se obtendrían realizando la minimización simultánea de todas las salidas, y se ahorra bastante tiempo de ejecución mediante la minimización previa de las salidas individuales.

En el Capítulo 5 se desarrolla una versión paralela de RRMIN2, denominada PRRMIN, apta para ser ejecutada en sistemas multiprocesador. PRRMIN permite aprovechar ordenadores con varios procesadores o un conjunto de ordenadores conectados en red para obtener los mismos resultados que con RRMIN2, pero precisando menor tiempo ejecución. En el Capítulo 6 se presentan los resultados experimentales obtenidos con RRMIN2, comparándolos con los de otros procedimientos que aparecen en la literatura y se analizan las mejoras de prestaciones que proporciona la versión paralela PRRMIN.

CAPÍTULO 5

PARALELIZACIÓN DE RRMIN2 EN SISTEMAS DE MEMORIA DISTRIBUIDA: PRRMIN

En el Capítulo anterior ha podido comprobarse que el principal inconveniente que presenta la utilización de un algoritmo de Enfriamiento Simulado es el gran tiempo de ejecución que se consume. Con el objetivo de paliar este problema, en este capítulo se aborda la paralelización del algoritmo RRMIN2 para su ejecución en sistemas multiprocesador. Como resultado, se va a obtener un algoritmo, que se denominará PRRMIN, y cuyo comportamiento experimental se analizará en el Capítulo 6.

5.1. INTRODUCCIÓN

En el Capítulo 4 se ha desarrollado un nuevo procedimiento para la minimización AND-EXOR que, según puede observarse en el Capítulo 6, presenta un comportamiento excelente en cuanto a la calidad de la solución que obtiene. Asimismo, se comprueba que el procedimiento precisa de elevados tiempos de ejecución para llegar a la solución final, debido a la utilización de un proceso de Enfriamiento Simulado [SOH95]. Este problema puede tratar de solucionarse utilizando procesamiento paralelo, usando bien multiprocesadores de memoria compartida, o bien multiprocesadores de memoria distribuida. En los departamentos de diseño VLSI suele ser común disponer de varias estaciones de trabajo conectadas en red, más que de un supercomputador con varios procesadores utilizando memoria compartida. Por esta

razón, el estudio de la paralelización se ha orientado hacia sistemas de memoria distribuida, y más concretamente al caso de una red de estaciones de trabajo utilizando paso de mensajes. No obstante, el algoritmo obtenido, PRRMIN, puede utilizarse en cualquier tipo de multiprocesador.

Para la realización del programa paralelo se ha utilizado PVM (Parallel Virtual Machine) [GEI94], una herramienta que permite utilizar un conjunto heterogéneo de computadores conectados en red como si se tratase de un sólo multicomputador. Teniendo en cuenta que PVM se está convirtiendo en un lenguaje paralelo estándard disponible en los últimos computadores paralelos, la portabilidad del programa desarrollado es considerable, tal y como se ha indicado anteriormente.

El capítulo consta de cuatro apartados, además de esta introducción. En el Apartado 5.2 se realiza un breve resumen de los métodos de paralelización en los problemas de optimización combinatoria; en 5.3 se realiza una revisión de los procedimientos más usuales de paralelización del algoritmo de Enfriamiento Simulado. En 5.4 se desarrolla la paralelización del algoritmo RRMIN2, obteniendo un procedimiento de minimización paralelo, PRRMIN, apto para ser ejecutado en sistemas de memoria distribuida. Finalmente, en el Apartado 5.5 se presentan las conclusiones del capítulo.

5.2. PARALELISMO EN LOS PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA

En el análisis que se ha hecho en el Apartado 4.2 de la minimización AND-EXOR, se ha planteado ésta como un problema de optimización combinatoria, aplicando en RRMIN2 un procedimiento eficaz para la resolución de este tipo de problemas: el Enfriamiento Simulado. Puesto que se está tratando la minimización AND-EXOR como un problema de optimización combinatoria, se puede utilizar alguna de las estrategias usuales para la paralelización de este tipo de problemas.

Los algoritmos para la optimización combinatoria se basan en procedimientos eficaces de búsqueda en el espacio de las soluciones posibles. Ese espacio se suele describir mediante un grafo que, en muchos casos, tiene forma de árbol [WAH85]. Según el árbol sea de tipo AND, OR, o AND/OR, el problema de optimización combinatoria asociado se resuelve usualmente mediante procedimientos basados en algoritmos de Divide-y-Vencerás [HOR83], Ramificación-y-Poda (Branch-and-Bound) [YAN94], o Búsqueda α-β [POW90], respectivamente.

La paralelización de estos procedimientos se realiza, normalmente, distribuyendo el espacio de búsqueda entre los procesadores disponibles. Usualmente, en estos problemas es fácil dividir el espacio en un número arbitrario de partes, pero éstas suelen requerir tiempos de cómputo diferentes, de manera que tras una distribución inicial de la carga entre los procesadores, unos pueden terminar su trabajo mucho antes que otros. Se trata de problemas irregulares en los que debido a esta desigual distribución del trabajo a realizar, se tendrán algoritmos con una eficacia pequeña a no ser que el procedimiento equilibre dinámicamente la carga [KUM94].

Los procedimientos paralelos más frecuentemente utilizados para problemas de optimización combinatoria como el que se considera se basan en la paralelización del procedimiento de Ramificación-y-Poda, puesto que el espacio de búsqueda se modela mediante un árbol OR. Se pueden clasificar en dos grupos [YAN94]:

1) **Ramificación-y-Poda Globalmente Paralelos.** En estos procedimientos se parte de un nodo que se expande en varios, éstos a su vez se expanden en otros nodos, y así sucesivamente se van asignando nodos a procesadores. Al principio habrá menos tareas que procesadores y, por consiguiente no todos los procesadores estarán ocupados. Llegará un momento en el que habrá tantas o más tareas que procesadores. Entonces habrá que ir asignando las tareas que quedan pendientes a los procesadores que van terminando su trabajo. En la Figura 5.1 puede verse un ejemplo de este tipo de paralelismo. En concreto, se ha considerado un problema de optimización combinatoria en el que en cada estado se tienen cuatro transiciones posibles, ejecutándose en un computador con 8 procesadores. En la Tabla 5.1 puede verse cómo iría variando la ocupación de los procesadores. Se supone que cada tarea precisa una unidad de tiempo para completarse y se desperdician los tiempos de comunicación. En el Nivel 0 se tiene una sóla tarea, de manera que sólo el procesador 1 está ocupado. En el Nivel 1 ya hay cuatro procesadores ocupados; y al llegar al Nivel 2 el número de tareas supera ya al de procesadores (se tienen 9 tareas para 8 procesadores). El tiempo de ejecución para todo el árbol sería de 7 unidades de tiempo.

Procesador / Tiempo	P1	P2	P3	P4	P5	P6	P7	P8
0	1	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1
6	1	1	1	1	1	0	0	0
7	0	0	0	0	0	0	0	0

Tabla 5.1. Ocupación de los procesadores en función del tiempo.

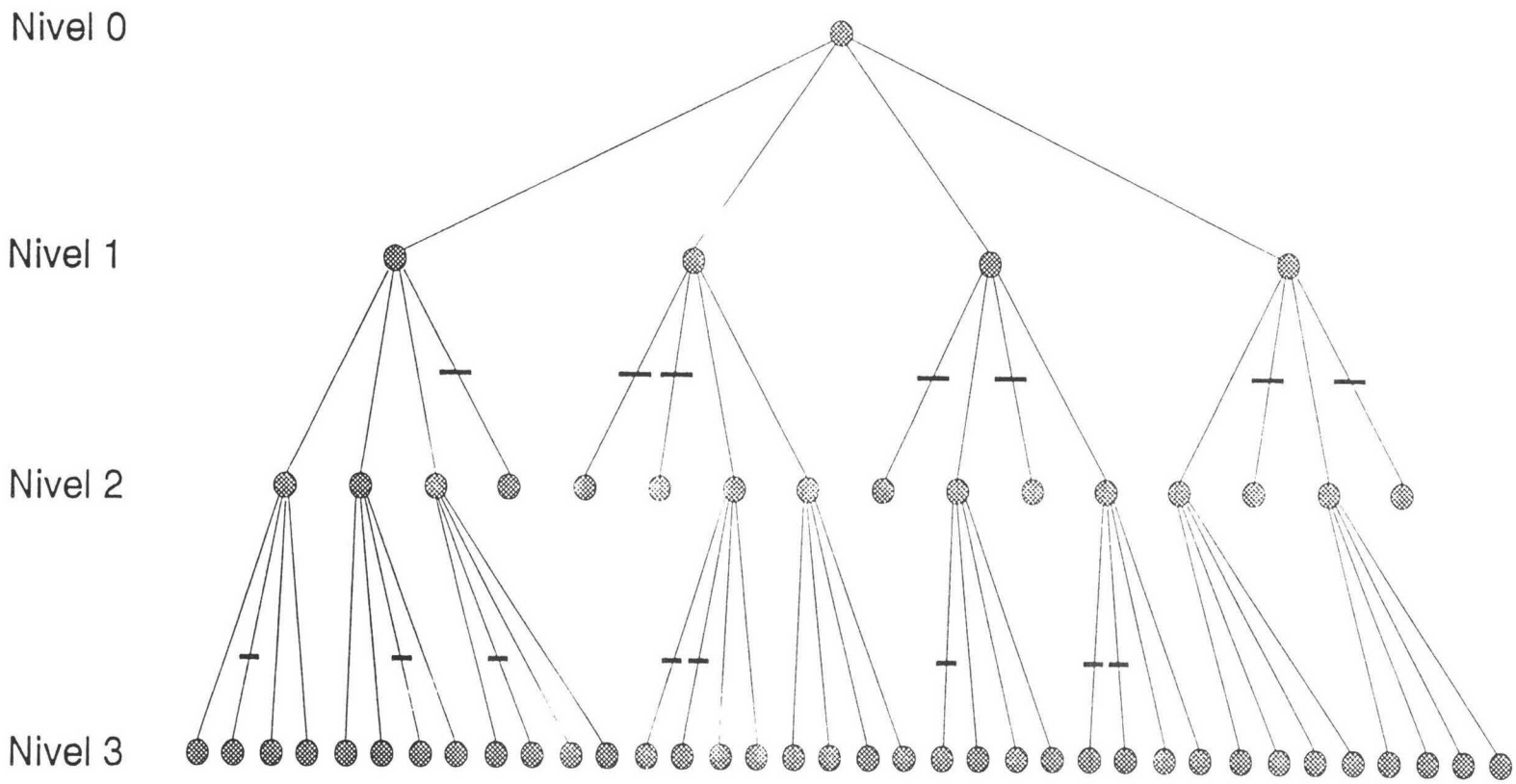


Figura 6.1. Ejemplo de Ramificación-y-Poda Globalmente Paralelos

2) **Ramificación-y-Poda por Descomposición.** Estos procedimientos realizan una partición del árbol de búsqueda antes de ejecutar el algoritmo, de manera que haya tantos subárboles como procesadores. La ventaja que presentan este tipo de algoritmos es el gran ahorro que se consigue en comunicaciones. Sin embargo, es difícil obtener una partición inicial en la que todos los árboles tengan una carga similar. En la Figura 5.2 puede verse, para el mismo ejemplo de la Figura 5.1, la forma de operar de este procedimiento y el problema indicado. La partición se ha hecho suponiendo un desconocimiento del espacio de búsqueda. El subárbol asignado al procesador 1 al final ha resultado contener 10 tareas, mientras que, por ejemplo, el subárbol del procesador 3 sólo 2. El tiempo de ejecución para todo el árbol en este caso sería de 10 unidades de tiempo. Consiguientemente, si no se quiere reducir la eficacia del algoritmo paralelo habría que añadir algún mecanismo que permita que los procesadores que van terminando la búsqueda en el subárbol que se les había asignado, puedan recibir tareas de otros procesadores [KUM94].

Estos dos procedimientos generales presentan graves inconvenientes para su utilización en RRMIN2:

- En el caso del procedimiento 1), la Ramificación-y-Poda Globalmente Paralelos, se tiene el problema de que el Enfriamiento Simulado precisa del estado anterior para generar un nuevo estado. Por tanto, todas las tareas que se han distribuido entre los procesadores, deben de comunicar sus resultados para poder generar el estado siguiente. Esto ocasiona un gran volumen de comunicaciones que reduce la eficacia del procedimiento

- El procedimiento 2) presenta el inconveniente de que en el problema de la minimización AND-EXOR resulta muy difícil efectuar una partición adecuada del espacio de búsqueda, debido a que se puede llegar a cada uno de los estados, en general, de muchas maneras diferentes.

Estos inconvenientes, comunes en los procedimientos de optimización que emplean Enfriamiento Simulado, obligan a utilizar métodos de paralelización más específicos. En el apartado siguiente se realiza un estudio de métodos de paralelización especialmente diseñados para algoritmos de Enfriamiento Simulado.

5.3. PARALELIZACIÓN DEL ENFRIAMIENTO SIMULADO

Debido a la popularidad de este algoritmo para la resolución de problemas de optimización combinatoria, y a que consume un tiempo de ejecución considerable, se ha propuesto una gran número de procedimientos para su paralelización. Al abordar la paralelización de un algoritmo basado en Enfriamiento Simulado, es preciso tener en cuenta que este tipo de algoritmos implica (véase el Apartado 4.3.3):

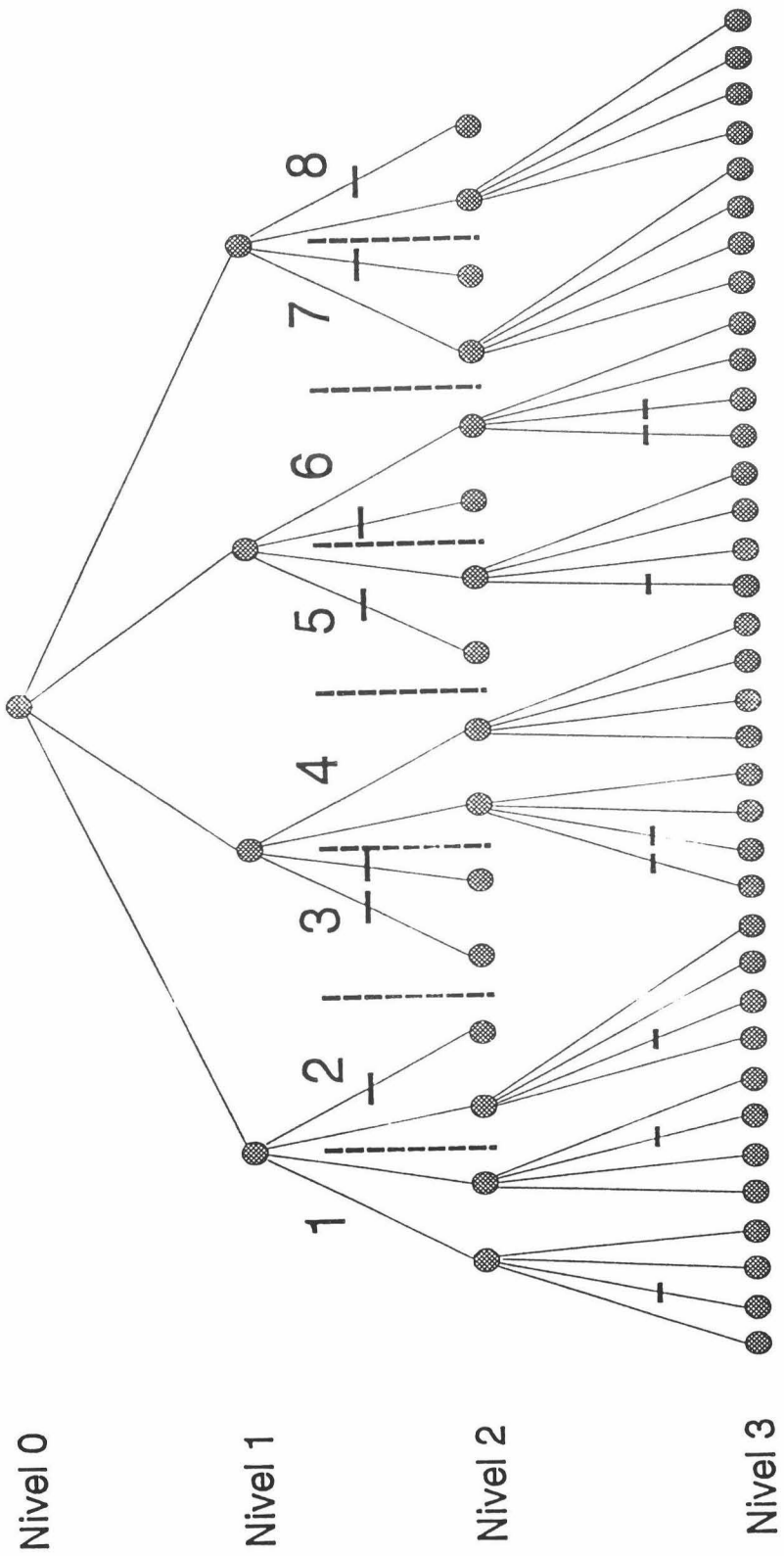


Figura 5.2. Ejemplo de Ramificación-y-Poda por Descomposición

1) Determinar una nueva solución a partir de la solución actual.
2) Calcular la diferencia en costo de la nueva solución con respecto a la que se tiene.
3) Decidir si la solución se acepta o no.
4) Sustituir la solución actual por la nueva solución, si ésta es aceptada.

Según de qué forma se trate cada una de estas cuatro etapas, aparecen distintos tipos de procedimientos. Inicialmente, puede hacerse una clasificación en dos grandes grupos:

- Procedimientos dependientes del problema. Es decir, procedimientos en los que se tienen en cuenta las características específicas del problema. Este tipo de métodos implican una paralelización del paso 1.

- Procedimientos independientes del problema. Este tipo de procedimientos son aplicables a cualquier tipo de problema que utilice Enfriamiento Simulado.

La mayoría de los procedimientos dependientes del problema se han desarrollado para optimizar la colocación de celdas en circuitos integrados VLSI. Ejemplos de estos procedimientos pueden encontrarse en [CAS87], [MAL88], [DAR87], [KRA87], [BAN90] y [CHA96]. Casi todos estos procedimientos se basan en efectuar una partición adecuada del espacio de búsqueda (es decir, utilizan Ramificación-y-Poda por Descomposición), y por tanto, resultan de poca utilidad para el problema que se tiene planteado. Por esta razon, los procedimientos que se van a analizar aquí son todos del segundo tipo, independientes del problema.

Antes de pasar a la revisión de los procedimientos, debe hacerse notar que un programa paralelo de Enfrimiento Simulado de aplicación general debe cumplir dos premisas fundamentales:

a) Debe conseguirse una ganancia de velocidad aceptable.
b) No deben alterarse las condiciones de convergencia del algoritmo secuencial.

5.3.1. Cadenas de Markov Múltiples

Este procedimiento, descrito en [CHA96], modela el proceso de Enfriamiento Simulado como un árbol en el que se van proponiendo cambios de estados, y éstos se aceptan o no según la evaluación de una función de coste. El árbol de búsqueda es, esencialmente, una cadena de Markov (Apartado 4.3.3) y la paralelización puede abordarse iniciando cadenas diferentes en cada procesador (dando una semilla para la generación aleatoria distinta a cada uno). Cada cadena exploraría el espacio completo independientemente, efectuando los cuatro pasos descritos en 5.3. Una vez que cada procesador ha completado el esquema de enfriamiento, se comparan las soluciones y se selecciona la mejor.

Así planteado, este método no consigue ninguna ganancia de velocidad, puesto que

cada procesador realiza el mismo trabajo que el algoritmo secuencial. La idea que se propone para conseguir ganancia de velocidad es reducir el número de transiciones de cada cadena en un factor de 1/N donde N es el número de procesadores. De esta forma, debería de conseguirse una ganancia de velocidad teórica de N, y se cumpliría la primera de las premisas citadas anteriormente. Sin embargo, no se cumpliria la segunda, puesto que al reducir el tamaño de las cadenas de Markov se están alterando las condiciones de convergencia (la demostración de la convergencia del algoritmo de Enfriamiento Simulado se basa precisamente en el uso de una cadena de Markov, homogénea para describir el proceso de enfriamiento, [AAR90]), y por tanto, la calidad de la solución obtenida. Para evitar este inconveniente en lo posible, debe incorporarse algún mecanismo de interacción entre las distintas cadenas. Se han probado dos modos de comunicación, una síncrona y otra asíncrona, para efectuar esta interacción, dando lugar a dos tipos de cadenas múltiples:

1) **Cadenas de Markov Múltiples síncronas (MMC síncronas).** En este modo de interacción, denominado en [LEE92] MMC síncrono con intercambio periódico, las soluciones se comparan en intervalos fijos de ejecución. Con este método, cada cadena de Markov actualiza sus datos cada cierto número de iteraciones con la mejor solución, y continúa con ella su proceso de optimización. Este punto de intercambio sirve como final de un segmento de cómputo, y se comporta como una barrera de sincronización.

La forma de implementar este modo es utilizar un proceso padre que en la barrera de sincronización recibe los costos de las soluciones generadas por los procesos hijos, determina cual de ellas es la mejor, y la envía mediante un "broadcast" a los hijos. Este método presenta el inconveniente de que se pierde tiempo en la barrera de sincronización (hay que esperar a que terminen todos los procesos para poder empezar otro segmento de cómputo).

2) **Cadenas de Markov Múltiples Asíncronas (MMC asíncronas).** Una mejora sencilla del procedimiento anterior consiste en hacer que cada proceso, al acabar, envíe el costo de su solución al padre. Éste compara esta solución con la mejor que le haya llegado hasta ese momento. Si la mejora, el padre guarda ésta como la mejor solución, y ordena al hijo que prosiga con ella sus cálculos. En caso contrario, comunica al hijo la mejor solución de que dispone y éste continúa el proceso de enfriamiento a partir de ella.

De esta forma se consigue que nigún proceso esté nunca parado, aunque puede ocurrir que haya hijos que no estén trabajando con la mejor solución en un momento dado.

En [CHA96] se dan resultados obtenidos con MMC asíncronas para el problema de la colocación de celdas en circuitos VLSI, obteniendo ganancias máximas con 4 procesadores de 2.53 en un SUN 4/690 MP y 2.97 en un Intel iPSC/860. Todo ello sin que se produjeran

variaciones significativas en la calidad de la solución. No obstante, debe tenerse en cuenta que se ha variado la longitud de las cadenas de Markov, y esto puede ocasionar en otro tipo de problema que la convergencia empeore, sobre todo en el caso de que las cadenas utilizadas sean cortas (como es el caso de RRMIN2).

5.3.2. Procedimiento de ROUSSEL-DREYFUS

El procedimiento que se presenta a continuación, desarrollado en [ROU90], trata de paralelizar el Enfriamiento Simulado de forma independiente del problema; y sin alterar las condiciones de convergencia del algoritmo secuencial. Para ello, considera dos rangos diferentes de temperatura, que vienen determinados por la razón de aceptación $\chi(T)$, (relación entre el número de transiciones aceptadas y el número de transiciones posibles a una temperatura dada, [AAR90]) En un proceso de Enfriamiento Simulado, $\chi(T)$ disminuye al hacerlo la temperatura debido a dos causas:

- El sistema se aproxima a un mínimo, y se producen pocas transiciones que hagan disminuir la energía.
- El valor de $\exp(-\Delta E/T)$, donde ΔE es la variación de la función de coste y T la temperatura, es muy pequeño, y por tanto la probabilidad de aceptar transiciones que incrementen la energía es muy reducida.

Esta redución de $\chi(T)$ puede aprovecharse para desarrollar métodos de paralelización. En efecto, a bajas temperaturas se pueden utilizar K, $K<1/\chi(T)$, procesadores, de manera que cabe esperar que al menos una de las K transiciones que se han intentado simultáneamente sea aceptada. En este supuesto, debería obtenerse una ganancia de velocidad del orden de K.

Quedaría por resolver qué método se utiliza en el rango de temperaturas altas. En ese caso, el procedimiento propone que cada procesador evalúe únicamente una transición. Una vez que todos los procesadores hayan terminado, se elige aleatoriamente una de esas transiciones, y todos los procesadores continúan a partir de ella. La ganancia de velocidad se obtendría al contabilizar varios intentos de transición en cada segmento de cálculo de los procesadores (lo que equivale a reducir el tamaño de las cadenas de Markov, al igual que se hacía en 5.3.1).

Por tanto, el procedimiento consta de dos modos de funcionamiento:

1) Modo de temperaturas altas, $\chi(T)\geq 1/K$. Cada procesador evalúa sólo una transición y se elige una de ellas aleatoriamente. La memoria de los procesadores se actualiza con la nueva solución, y se efectúa una nueva iteración. Para mantener la misma distribución de probabilidad que en la cadena de Markov secuencial, el cálculo de las transiciones intentadas se realiza como sigue:

- Si se han intentado K transiciones y al menos una se ha aceptado, se consideran como transiciones intentadas el número:

$$\frac{K+1}{K-r+1} \quad \textit{donde } r=n^{\underline{o}} \textit{ de transiciones rechazadas} \qquad (5.1)$$

- Si no se acepta ninguna de las K transiciones, el algoritmo considera que se han intentado K transiciones.

2) Modo de temperaturas bajas, $\chi(T)<1/K$. Lo procesadores buscan transiciones en paralelo, hasta que una de ellas es aceptada. En ese momento, los procesadores se sincronizan y continúan el enfriamiento a partir de esa nueva solución.

Este procedimiento de paralelización presenta dos inconvenientes:

a) A temperaturas altas los procesadores intercambian información en cada iteración. Esto genera un gran volumen de comunicaciones y hace que el tiempo de ejecución paralelo sea en ocasiones superior al secuencial en ese rango de temperaturas.
b) En el modo de temperaturas bajas, al aceptarse la primera transición generada, se está favoreciendo un determinado tipo de transición (aquellas que tardan menor tiempo en generarse), de manera que pueden estar reduciendose las posibilidades de escapar de mínimos locales (se están alterando las condiciones de convergencia).

En el siguiente apartado se presentan otros procedimientos que tratan de resolver estos problemas mediante la utilización de cómputo especulativo.

5.3.3. Paralelización especulativa del Enfriamiento Simulado

Otro procedimiento para la paralelización del Enfriamiento Simulado consiste en utilizar una técnica de computación especulativa mediante un Árbol Binario Especulativo. El cómputo binario especulativo (BSC, Binary Speculative Computation, [WIT91]), define tres tipos de procesadores, denominados raiz, acepta, y rechaza, según puede verse en la Figura 5.3.

Cada procesador efectúa una secuencia de modificación-evaluación-decisión distinta, de acuerdo con el siguiente esquema:

- El procesador raíz, que está en la iteración *i*, envía su estado actual, que denominaremos DATOS, a sus dos procesos hijos (rechaza y acepta) antes y despues de efectuar la modificación del estado actual.
- Cada uno de los hijos, al recibir su estado, empieza a trabajar con ellos sin saber si ese estado es correcto o no. El hijo acepta recibe la modificación DATOS', bajo la suposición de que se ha aceptado ese estado. Asimismo, envía DATOS' a sus dos hijos rechaza y acepta, antes y después de modificar DATOS' en la iteración i+1. Así, su hijo rechaza recibirá directamente DATOS' y empezará a trabajar en la iteración

i+2; y su hijo acepta recibirá DATOS''. Este proceso continúa hasta que se hayan terminado los procesadores.

- Cuando no quedan más procesadores, se envía al nodo raíz el estado actual y se vuelve a empezar todo el proceso.

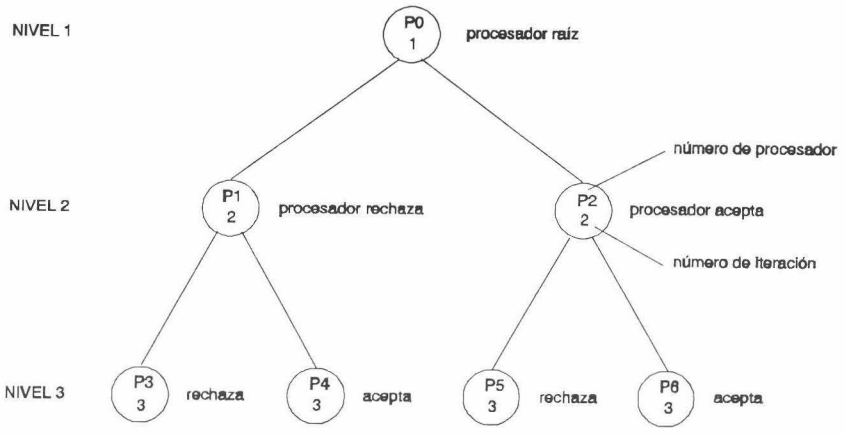


Figura 5.3. Árbol Binario Especulativo con 3 niveles y 7 procesadores.

Bajo estas condiciones, se obtendrá ganancia en velocidad cuando se produzcan muchos estados rechazados. Por ejemplo, en la Figura 5.3, si se rechazan los estados obtenidos en las 3 iteraciones, se habrían completado 3 iteraciones en el mismo tiempo que se precisa en el algoritmo secuencial para efectuar 1. En el caso de que se acepten todos los estados, no se obtendría ganancia en velocidad. Dado que, en general, se aceptan muchas menos transiciones de las que se intentan, el procedimiento debe comportarse bien, especialmente a temperaturas bajas.

Existen en la literatura dos mejoras a este procedimiento, que se describen a continuación.

5.3.3.1. Paralelización especulativa mediante árboles no equilibrados

El principal problema que plantea el procedimiento que se acaba de describir es la gran cantidad de comunicaciones que se producen. El nodo raiz envía la solución al nodo rechaza, genera una transición, y envía la nueva solución al nodo acepta. Cada nodo transfiere a continuación las soluciones a sus hijos. Después de efectuar una decisión, el nodo final de la rama correcta envía su solución al nodo raíz. En el caso de redes grandes, la solución tiene que viajar a través de una gran cantidad de nodos para alcanzar el nodo raíz, y así, en algunos experimentos [WIT90] se han obtenido tiempos de ejecución mayores en el algoritmo paralelo que en el secuencial.

El algoritmo que se presenta a continuación se centra en minimizar el volumen de comunicaciones para obtener una mayor eficiencia. Se basa en la observación de que la diferencia entre la solución en un nodo cualquiera y su hijo o padre es como máximo de una transición. Así, la idea principal consiste en comunicar sólo las transiciones nuevas.

Inicialmente, el nodo raíz envía la solución inicial a todos los procesadores. De esta forma, todos los nodos tienen la solución actual al principio de cada fase de cálculo. Al empezar, el nodo raíz efectúa la primera transición, y los demás procesadores únicamente reciben el número de transiciones que se requieren para esa operación.

La Figura 5.4 muestra un árbol no equilibrado que consta de 10 procesadores. Cada nodo tiene dos hijos como máximo, de los cuales, el de la izquierda es el nodo rechaza y el de la derecha el nodo acepta. El nodo 0 es el procesador raiz, las flechas indican la dirección de la comunicación, y el número que tienen al lado es el número de transiciones que se precisan transferir.

Puesto que el nodo 1 es el nodo rechaza del raiz, no requiere ninguna transición, al igual que el nodo 3. La transición generada por el nodo 1 se envía la nodo 4. El nodo 2 necesita la transición generada por el procesador raíz, por tanto recibe esa transición, genera una nueva y envía ambas al nodo 5. Esta técnica se aplica a todos los nodos de la red. Cada nodo evalúa su solución inmediatamente después de comunicarse con sus hijos.

Una vez que se ha identificado la rama correcta, el nodo final transfiere el número de transiciones que se requieren para actualizar a sus vecinos, que a su vez, van enviando los números de transiciones para actualizar a sus vecinos. El proceso sigue hasta actualizar todos los nodos. La nueva iteración comienza cuando el nodo raíz finaliza la comunicación con sus hijos.

La Figura 5.5 muestra el proceso de actualización de los datos en la misma red de procesadores de 5.4. Se supone que la rama correcta finaliza en el nodo 8. Así, la transición producida por dicho nodo, se ha aceptado. Los nodos 7 y 9 sólo necesitan esa transición para actualizar sus soluciones. El nodo 5 necesita dos transiciones, las generadas por el nodo 8 y el 5. El nodo 5 actualiza la solución en paralelo con su padre y su nodo hijo rechaza. El mismo proceso se aplicaría al resto de los nodos.

Este algoritmo presenta las siguientes ventajas con respecto al anterior:

- Para problemas de gran tamaño, el volumen de comunicaciones se reduce considerablemente.
- El tiempo de comunicación depende sólo de la forma del árbol y del número de nodos. Por tanto, al aumentar el tamaño del problema, las prestaciones del método mejoran, al contrario de lo que ocurría con el procedimiento de [WIT91].
- A bajas temperaturas se requieren menor número de transiciones y , por tanto, la eficiencia del algoritmo se incrementa.

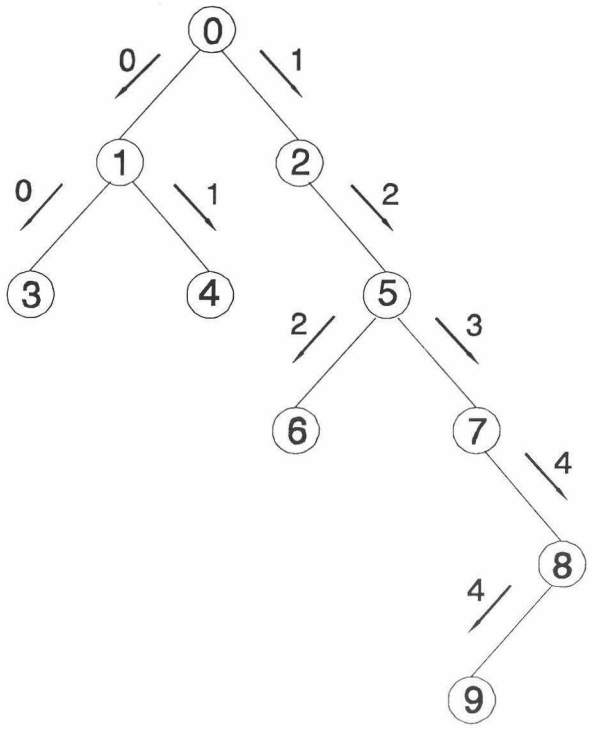


Figura 5.4. Transmisión de transiciones en un árbol no equilibrado de 10 procesadores

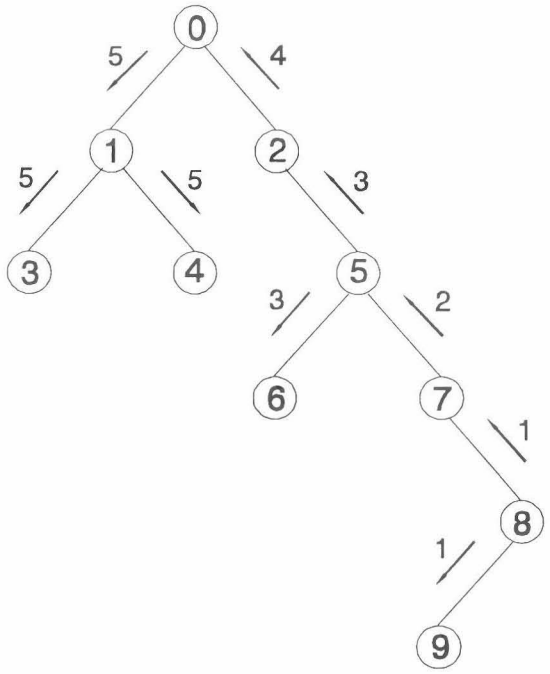


Fig 5.5. Actualización de los nodos del árbol no equilibrado de 10 procesadores

Finalmente, en este procedimiento se tiene en cuenta que la razón entre el número de transiciones aceptadas y rechazadas va variando según avanza el proceso de enfriamiento. Así, para un mejor aprovechamiento de los nodos de la red, el método varía la forma del árbol no balanceado cada vez que se produce una variación de la temperatura. Para ello efectúa una estimación de la ganancia de velocidad a cada temperatura en función de las probabilidades de aceptación, de rechazo, y los tiempos de evaluación, transición, cálculo y comunicación [TAR95].

5.3.3.2. Paralelización mediante árboles N-arios generalizados

Este procedimiento utiliza árboles N-arios en lugar de árboles binarios para efectuar la paralelización, mediante una técnica denominada Cómputo Especulativo Generalizado (GSC, Generalized Speculative Computation, [SOH95]). En este método no se tienen nodos acepta o rechaza, y además cada procesador efectúa una iteración diferente. La idea consiste en enviar a cada procesador un índice de bucle en tiempo de ejecución, y que a partir de él realice la transición correspondiente, la evalúe y decida. Una vez realizadas estas tres etapas, el procesador utiliza una bandera para indicar el resultado de su decisión. Los resultados de las decisiones se comparan y el procesador con índice de iteración más bajo que haya aceptado su transición es el que se toma para iniciar el siguiente nivel. Para que la versión paralela genere la misma secuencia de decisión que el Enfriamiento Simulado secuencial, se utilizan las mismas semillas en todos los procesadores para generar los números aleatorios. Concretamente se utiliza como semilla el número de iteración.

En la Figura 5.6 puede verse la ejecución de 11 iteraciones mediante GSC en un árbol de tres niveles. El primer nivel del árbol especulativo N-ario contiene siete procesadores ejecutando siete iteraciones simultaneamente. Supóngase que los procesadores 3, 4 y 6 aceptan sus transiciones (los círculos rellenos indican transiciones aceptadas). En el árbol, la transición aceptada por el procesador con índice más bajo es la que se tiene en cuenta, mientras que las decisiones tomadas por los procesadores con índice más alto son erróneas. Así, la transición calculada por el procesador 4, y que ha tratado como aceptada, debe ser modificada con la solución obtenida por el procesador 3. De manera similar, la decisión tomada por el procesador 6 es errónea y sus datos deben ser actualizados también con los del procesador 3.

El segundo nivel empieza en la iteración 5, ejecutando las iteraciones de la 5 a la 11 en 7 procesadores actuando en paralelo. Cada procesador recibe el índice de iteración 4; a partir del cual efectúan la transición correspondiente. El siguiente índice de iteración puede obtenerse fácilmente sumandole al número de procesador el índice de iteración correcto (en este caso proporcionado por el procesador 3; el 4) incrementado en 1. Por ejemplo, el procesador 2 ejecutaría la iteración 7. En este segundo nivel, se obtiene que los procesadores 4 y 5 han encontrado transiciones aceptables. Nuevamente se considerará correcta la transición del procesador con índice más bajo; y así, se selecciona la transición efectuada por el procesador 4. El tercer nivel empezaría en este ejemplo con la iteración 10.

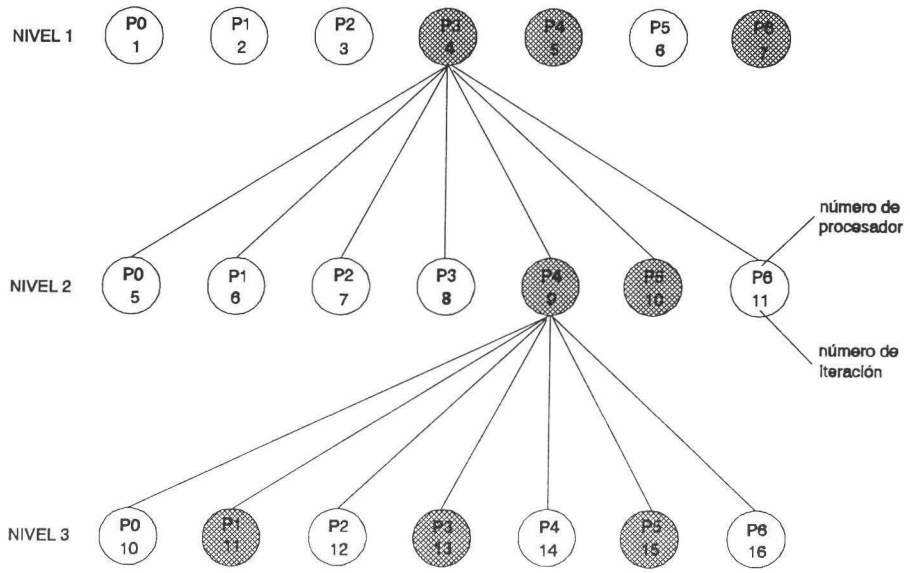


Figura 5.6. Árbol especulativo N-ario.

Al igual que con los otros procedimientos basados en cálculo especulativo, este procedimiento proporcionará mayor ganancia en velocidad cuanto mayor sea el número de rechazos. De hecho, en los resultados que se proporcionan en [SOH95], se menciona que la ganancia en velocidad disminuye el aumentar la temperatura.

5.4. PARALELIZACIÓN DEL ALGORITMO RRMIN2: PRRMIN

En el Apartado 5.3 se ha efectuado una revisión de los procedimientos que aparecen en la literatura para la paralelización del Enfrimiento Simulado. Cualquiera de ellos podría haberse utilizado para abordar la paralelización de RRMIN2; pero se ha optado por desarrollar un procedimiento específico debido a ciertos inconvenientes que presentan los procedimientos citados. Concretamente, el método del Apartado 5.3.1. presenta problemas de convergencia, puesto que las cadenas de Markov utilizadas en RRMIN2 son muy cortas por cuestiones de optimización del algoritmo en tiempo de ejecución y prestaciones. El procedimiento de 5.3.2 genera un gran volumen de comunicaciones a temperaturas altas, lo que resulta especialmente problemático si se pretende utilizar una red de estaciones de trabajo, dado que se dispone de un ancho de banda reducido. Además, a temperaturas bajas, produce modificaciones en la convergencia del proceso de enfriamiento. Finalmente, los procedimientos basados en cálculo especulativo (Apartado 5.3.3) mantienen bien la convergencia, pero su ganancia en velocidad depende de la temperatura. A temperaturas bajas, se producen una gran cantidad de transiciones rechazadas, con lo que estos procedimientos producen valores muy buenos de ganancia en velocidad. Sin embargo, a temperaturas altas, cuando se aceptan una gran cantidad de transiciones, la ganancia en velocidad disminuye. Teniendo en cuenta que en

RRMIN2 se van modificando las probabilidades de aplicación de las reglas con la temperatura para procurar que se produzcan transiciones aceptadas en todo el rango de temperaturas, el cómputo especulativo tampoco representa una solución óptima.

5.4.1. Consideraciones generales para la paralelización de RRMIN2

En la paralelización de RRMIN2 deben tenerse presente las siguientes cuestiones:

a) La paralelización va a efectuarse en un sistema de memoria distribuida, por lo que habrá que enviar las actualizaciones de los datos a través de la red de interconexión.
b) Se plantea la ejecución del procedimiento paralelo en una red de estaciones de trabajo, con la limitación de ancho de banda que ello supone.
c) Según ha tenido ocasión de comprobarse, en la minimización AND-EXOR mediante RRMIN2, más del 99% del tiempo de procesamiento se consume en escapar de mínimos locales.

Las cuestiones a) y b) condicionan al procedimiento paralelo en el sentido de que el volumen de comunicaciones debe minimizarse todo lo posible. La cuestión c) proporciona un criterio para abordar la paralelización; ya que debe procurarse que los procesadores busquen las salidas de los mínimos locales en paralelo.

La idea fundamental del procedimiento se explica a partir de la Figura 5.7. Los puntos representan estados del sistema, el número que hay al lado de cada nodo el coste asociado a ese estado; y el punto rodeado por un círculo es el estado óptimo. En el gráfico de la figura aparecen dos posibles caminos hacia el mínimo, nombrados con las letras A y B, de manera que el tiempo medio para llegar al mínimo utilizando un sólo procesador vendría dado por:

$$L=L(A)P(A)+L(B)P(B) \tag{5.2}$$

donde L(A) es el tiempo de ejecución del camino A, P(A) es la probabilidad de elegir el camino A; idem para B. Si se consideran ambos caminos equiprobables, y se anota una unidad de tiempo de ejecución por cada transición, se tendría:

$$L=8\cdot 0.50+3\cdot 0.50=5.50 \ \textit{unidades de tiempo} \tag{5.3}$$

En el caso de que se tuvieran dos procesadores, y ambos exploraran el espacio independientemente, caben cuatro posibilidades:

a) Los dos procesadores toman el camino A.
b) El procesador 1 toma el camino A y el procesador 2 el camino B.
c) El procesador 1 toma el camino B y el procesador 2 el camino A.
d) Los dos procesadores toman el camino B.

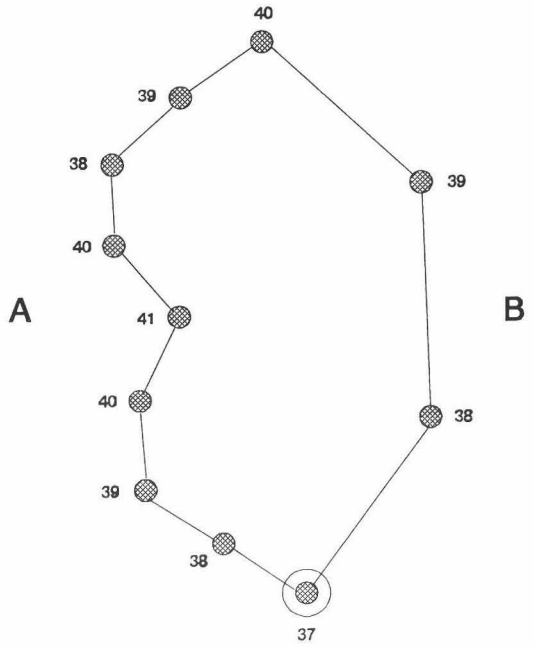


Figura 5.7. Dos posibles caminos para alcanzar un estado de coste mínimo.

Teniéndolas en cuenta, el tiempo medio de ejecución para alcanzar el mínimo en este caso vendría dado por:

$$L_2 = L(A)P(AA) + L(B)\cdot[P(AB) + P(BA) + P(BB)] \qquad 6.4$$

donde P(AA) es la probabilidad de que los dos procesadores tomen el camino A, P(AB) la probabilidad de que el procesador 1 tome el camino A y el procesador 2 el B, P(BA) la probabilidad de que el procesador 1 tome el camino B y el procesador 2 el A; y P(BB) la probabilidad de que ambos tomen el camino B.

Por tanto, la ganancia de velocidad que se obtiene al emplear dos procesadores viene dada por:

$$s = \frac{L}{L_2} = \frac{L(A)P(A) + L(B)P(B)}{L(A)P(AA) + L(B)\cdot[P(AB) + P(BA) + P(BB)]} \qquad (5.5)$$

Considerando que a), b), c) y d) son equiprobables, el tiempo de ejecución usando dos procesadores será:

$$L_2 = 8\cdot 0.25 + 3\cdot 0.75 = 4.25 \ \ u.d.t. \qquad (5.6)$$

y la ganancia de velocidad:

$$s=\frac{L}{L_2}=\frac{5.50}{4.25}\cong 1.30 \tag{5.7}$$

Asímismo, nótese que P(A)=2P(AA), y P(AA)=P(AB)=P(BA)=P(BB), con lo que la expresión 5.5 puede escribirse:

$$s=\frac{L}{L_2}=\frac{2+2\dfrac{L(B)}{L(A)}}{1+3\dfrac{L(B)}{L(A)}} \tag{5.8}$$

de donde se concluye, como era de esperar, que la ganancia de velocidad se aproxima a 2 cuando L(B)<<L(A).

Con este ejemplo se ha mostrado que puede obtenerse ganancia de velocidad si dos procesadores exploran el espacio de soluciones independientemente. No obstante, en el espacio real de búsqueda existen una gran cantidad de ramificaciones y es usual pasar por una gran cantidad de nodos antes de llegar al estado de mínima energía. Por tanto, si se dejan dos procesadores explorando el espacio independientemente sin que haya ninguna comunicación entre ellos, lo más probable es que ambos acaben anclados en mínimos locales, precisando el mismo tiempo de ejecución en llegar al mínimo que si se tuviera un sólo procesador. En la Figura 5.8 se presenta un espacio de búsqueda más ramificado y con mayor número de nodos. Se han dibujado unicamente los nodos que presentan ramificación. El número que aparece en cada rama es el número de transiciones de estado necesarias para pasar el siguiente nodo dibujado.

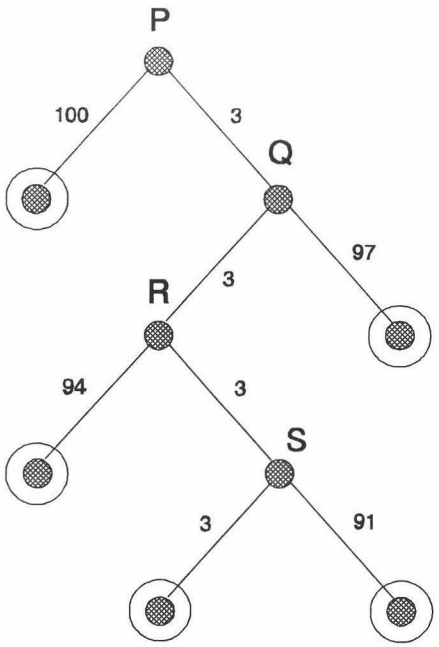


Figura 6.8. Espacio de búsqueda con varios caminos para llegar al estado óptimo.

Nótese que todos los caminos que llevan hasta el mínimo precisan de 100 transiciones, excepto uno, que denominaremos camino correcto, C, y que sólo necesita 12 transiciones. Teniendo en cuenta que todos los caminos incorrectos, I, tienen la misma longitud; el tiempo medio de ejecución puede escribirse:

$$L=L(C)P(C)+L(I)\cdot[1-P(C)] \tag{5.9}$$

donde L(C) es el tiempo del camino correcto, P(C) la probabilidad de tomar el camino correcto, y L(I) el tiempo que necesita cualquiera de los caminos incorrectos.

Si se tiene un sólo procesador:

$$P(C)=P_P(C)P_Q(C)P_R(C)P_S(C) \tag{5.10}$$

donde $P_X(C)$ es la probabilidad de que se tome el camino correcto en el nodo de ramificación X. Esta probabilidad vale 1/2 en todos los casos (se tienen dos ramas), y por tanto, para un procesador:

$$L=12\cdot(0.5)^4+100\cdot[1-(0.5)^4]=94.5 \ \ u.d.t. \tag{5.11}$$

Si ahora se utilizan dos procesadores, pero sin que exista comunicación entre ellos, la probabilidad de que alguno de ellos consiga completar el camino correcto vendrá dada por:

$$\begin{aligned} P_2^S(C)=&P_P(CC)\cdot P_Q(CC)\cdot P_R(CC)\cdot[P_S(CI)+P_S(IC)+P_S(CC)]+\\ &+P_P(CC)\cdot P_Q(CC)\cdot[P_R(CI)+P_R(IC)]\cdot P_S(C)+\\ &+P_P(CC)\cdot[P_Q(CI)+P_Q(IC)]\cdot P_R(C)\cdot P_S(C)+\\ &+[P_P(CI)+P_P(IC)]\cdot P_Q(C)\cdot P_R(C)\cdot P_S(C) \end{aligned} \tag{5.12}$$

donde $P_X(IC)$ sería la probabilidad de que el primer procesador tome un camino incorrecto, y el procesador 2 el camino correcto en la ramificación X. Análogo significado tienen $P_x(CI)$, $P_x(CC)$, $P_x(II)$. En cuanto a $P_x(Y)$, es la probabilidad de que el procesador que ha tomado la rama correcta en la anterior bifurcación tome el camino Y, donde Y puede ser el camino correcto C, o el camino incorrecto I. Sustituyendo estas probabilidades por su valores, se obtiene el siguiente tiempo medio de ejecución para dos procesadores sin comunicación:

$$L_2^S=12\cdot0.12+100\cdot0.88=89.44 \ \ u.d.t. \tag{5.13}$$

y por tanto, la ganancia de velocidad es:

$$s^S=\frac{L}{L_2^S}=\frac{94.50}{89.44}=1.06 \tag{5.14}$$

Como puede verse, se obtiene muy poca ganancia de velocidad. Por tanto, tal y como se ha indicado anteriormente, es preciso que los procesadores se comuniquen para optimizar la búsqueda que efectúan. En la Figura 5.9 se contempla esta posibilidad. Se ha establecido

una serie de barreras de sincronización en la que los dos procesadores ponen en común las mejores soluciones que han encontrado. Si se supone que estas barreras van a provocar que en el caso de que en la ramificación anterior cada procesador haya ido por una rama, los dos procesadores sigan por el camino correcto a partir de dicha barrera (si los dos iban por el camino correcto o los dos por el camino incorrecto, no hay posibilidad de elección), la expresión para P(C) será ahora:

$$P_2^C(C)=\left[P_P(CC)+P_P(CI)+P_P(IC)\right]\cdot\left[P_Q(CC)+P_Q(CI)+P_Q(IC)\right]\left[P_R(CC)+P_R(CI)+P_R(IC)\right]\cdot\left[P_S(CC)+P_S(CI)+P_S(IC)\right] \tag{5.15}$$

obteniéndose un tiempo medio de ejecución:

$$L_2^C=12\cdot(0.75)^4+100\cdot\left[1-(0.75)^4\right]=72.16 \tag{5.16}$$

La ganancia de velocidad obtenida en este caso sería:

$$s^c=\frac{L}{L_2^C}=\frac{94.50}{72.16}\cong 1.31 \tag{5.17}$$

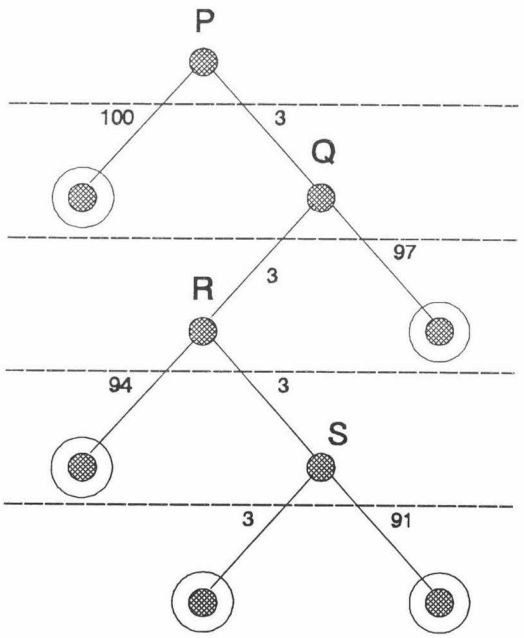


Figura 5.9. Ejemplo de espacio de búsqueda ramificado con barreras de sincronización

Como consecuencia de lo expuesto, pueden extraerse las siguientes conclusiones:

1) Al dejar que varios procesadores exploren un espacio de búsqueda independientemente, se reduce el tiempo medio para obtener una determinada solución.

2) Si se tienen un gran número de caminos distintos para alcanzar esa solución, la ganancia de velocidad obtenida es muy pequeña.
3) La ganancia de velocidad puede aumentarse si se introduce un mecanismo de comunicación entre los procesadores que vaya actualizando la solución.
4) En caso de que los procesadores se comuniquen para ir seleccionando el mejor camino, la mejor estrategia a seguir es que cada procesador vaya por un camino distinto, explorando la mayor cantidad de espacio posible.

En el apartado siguiente se propone un algoritmo paralelo para la minimización AND-EXOR basado en todas las consideraciones que se han ido mencionando.

5.4.2. El algoritmo PRRMIN

Como consecuencia de lo expuesto en 5.4.1, el algoritmo que se propone para la paralelización de RRMIN2 debería de cumplir dos condiciones:

a) Cada procesador debe explorar una parte del espacio distinta, es decir, no debe haber dos procesadores que exploren un mismo camino.
b) Debe proporcionarse un mecanismo de comunicación entre los procesadores para dirigir la exploración de manera que aumente la probabilidad de elegir el camino correcto.

La condición a), en principio, supondría efectuar una partición del espacio de búsqueda, lo que plantea problemas, según se mencionó en 5.2. Sin embargo, la gran ramificación del espacio de búsqueda hace muy poco probable que P procesadores que estén explorando ese espacio sigan el mismo camino después de algunas iteraciones del Enfriamiento Simulado. Para que la búsqueda efectuada por cada procesador sea independiente, bastará con que cada uno utilice una semilla distinta para la generación aleatoria de sus reglas de reescritura.

Para la condición b), cada K iteraciones del Enfrimiento Simulado se va a seleccionar la mejor solución de entre las que que hayan encontrado los P procesadores, y el siguiente segmento de K iteraciones se realizará partiendo de esa solución. En la Figura 5.10 puede verse el esquema general de funcionamiento de PRRMIN.

Como puede verse, se va a utilizar un procesador padre que va a ser el encargado de enviar la función a minimizar a cada procesador, recoger las soluciones intermedias generadas por cada uno, seleccionar la mejor, ordenar el envío de esa solución a todos los procesadores hijos, llevar el control del proceso de Enfriamiento Simulado, elaborar las estadísticas y realizar la presentación por pantalla. Los procesadores hijos van a efectuar K iteraciones del proceso de Enfriamiento Simulado partiendo de la función que les proporcione el padre en cada instante.

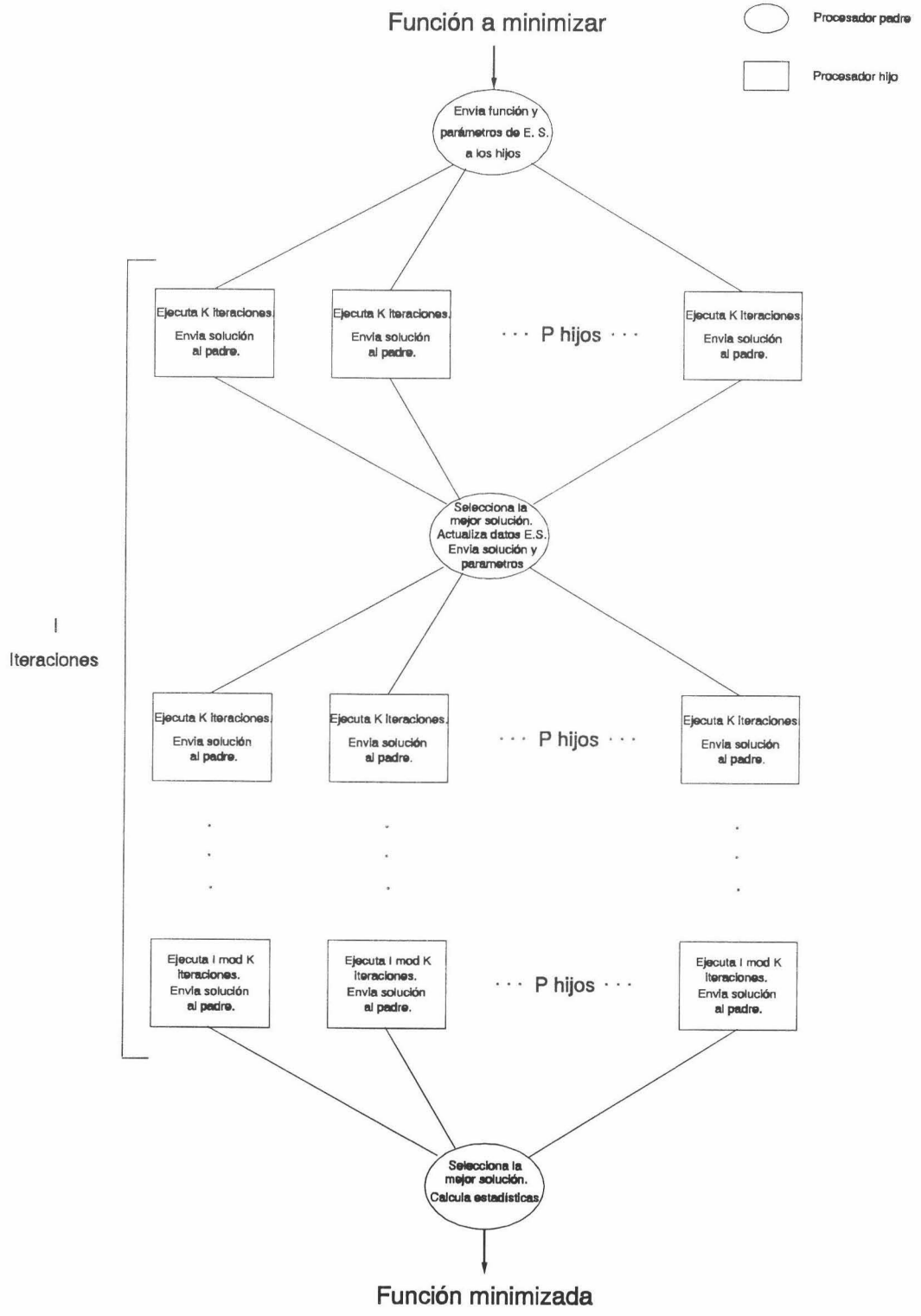


Figura 5.10. Esquema general del procedimiento PRRMIN

En los algoritmos 5.1 y 5.2 pueden verse los pseudocódigos que ejecutarían cada uno de los dos tipos de procesadores. El Algoritmo 5.1 corresponde al proceso padre, y en él se utilizan las mismas definiciones dadas en Algoritmo 4.15. y las siguientes funciones y definiciones nuevas:

- *n_iter*= número de iteraciones a ejecutar por cada hijo
- *n_hijos*= número de procesos hijos a crear
- *hijos*= vector que contiene los identificadores de proceso (pid) de cada uno de los hijos
- *Generar(hijos[i])*. Genera un proceso un hijo y almacena su identificador de proceso (pid) en la posición *i* del vector *hijos*
- *Broadcast(n_hijos,hijos,dato 1,dato 2,...,dato n)*. Envia a los *n_hijos* procesos que aparezcan en el vector *hijos* los *n* datos *dato 1 ... dato n*.
- *Recibir(hijo[i],dato)*. Recibe del hijo *i* la estructura de datos *dato* (*dato* puede ser un valor, un vector, una matriz, etc). La función recibir detiene la ejecución del programa hasta que llega el dato que se está esperando.
- *Presenta_resultados()*. Genera la salida con los resultados obtenidos.
- *Terminar_hijos(hijos)*. Manda una señal para que finalicen los programas hijos.

En el Algoritmo 5.2 pueden verse las instrucciones que serían ejecutadas por los procesadores hijos. Todas las funciones y variables son ya conocidos de algoritmos anteriores.

Finalmente, quedaría comentar que la principal cuestión que plantea este procedimiento de paralelización es el número K de iteraciones que se deja ejecutar a cada procesador sin que haya comunicaciones. Este número K debe ser:

1) Lo suficientemente pequeño para que los procesadores que no están explorando caminos "buenos" sigan desperdiciando tiempo de ejecución.
2) Lo suficientemente grande para que los procesadores sean capaces de salir de posibles mínimos locales. Si K es demasiado pequeño, podría ocurrir que el mínimo local que constituye la mejor solución hallada en las K iteraciones anteriores no pueda ser remontado por ninguno de los procesadores, viéndose afectada la calidad de la solución. Además, cuanto menor sea K mayor será el volumen de comunicaciones producido.

El valor óptimo de K (*n_iter* en el algoritmo PRRMIN) puede obtenerse experimentalmente, tal y como se reflejará en el Apartado 6.6.1.

5.5. CONCLUSIONES

En este Capítulo se ha abordado la paralelización del procedimiento de minimización AND-EXOR RRMIN2. En el Apartado 5.2, teniendo en cuenta que se ha tratado la minimización AND-EXOR como un problema de optimización combinatoria, se han analizado

los procedimientos más usuales que se utilizan para la paralelización de este tipo de problemas. Estos métodos no han resultado ser del todo apropiados por las características del espacio de búsqueda que aparece en la minimización lógica. En 5.3, y puesto que RRMIN2 se fundamenta en un proceso de Enfriamiento Simulado, se ha estudiado una serie de procedimientos independientes del problema para la paralelización del Enfriamiento Simulado. Nuevamente, estos procedimientos han presentado algunos inconvenientes para su utilización y como consecuencia, se ha optado por desarrollar un procedimiento específico para la paralelización de RRMIN2. Los fundamentos del método se han presentado en 5.4.1; y en el Apartado 5.4.2 se ha desarrollado el procedimiento paralelo PRRMIN, que permite la minimización AND-EXOR de funciones de conmutación en sistemas multiprocesador de memoria distribuida. En el siguiente capítulo se presenta un estudio experimental del comportamiento de PRRMIN en una red de estaciones de trabajo.

Algoritmo 5.1. Proceso padre de PRRMIN

```
n_iter=K;
n_hijos=P;
i=0;
Mientras(i<n_hijos) hacer
        Generar(hijos[i]);
Fin Mientras
L2=L;
Lop=L;
parar=0;
Mientras(parar=0) hacer
        c=c0;
        Δ=Δ0;
        Broadcast(hijos,n,nf,c,Δ,Tfinal,n_iter,p_regla0,p_regla1,p_regla2,c_corrección);
        Broadcast(hijos,L2);
        miniloc=LONG_MAX;
        jbucle=0;
        Mientras (j_bucle<n_hijos) hacer
                Recibir(hijo[j_bucle],L);
                Costo_L(L);
                Si (n_productosL<mini)
                        Lop=L;
                Fin Si
                Si (n_productosL<miniloc)
                        L2=L;
                Fin Si
                Recibir(hijo[j_bucle],c,Δ);
        Fin Mientras
        Si (c<Tfinal)
                parar=1;
        Fin Si
Fin Mientras
Terminar_hijos(hijos);
Presenta_resultados();
```

Algoritmo 5.2. Proceso hijo de PRRMIN.

```
longitud=L;
        parar=0;
        Mientras(1) hacer
                Mientras (parar=0) hacer
                Recibir(n,nf,c,Δ,T_final,n_iter, p_regla0, p_regla1, p_regla2, c_corrección);
                l=0;
                        Mientras (l<longitud) hacer
                                Generar(L,L2);
                                Costo_L(L);
                                Costo_L2(L2);
                                Costo_Lop(Lop);
                                Si (n_productosL2 < n_productosL)
                                        L=L2;
                                        Si (n_productosL2 < n_productosLop)
                                        Lop=L2;
                                        Fin Si
                                Fin Si
                                Si (n_productosL >= n_productosL)
                                        Si (exp((k*TRR(L)-TRR(L2))/c)>aleatorio())
                                                L=L2;
                                        Fin Si
                                Fin Si
                                L2=L;
                                l=l+1;
                        Fin Mientras
                        c=c-Δ;
                        Δ=Δ/b;
                        iteración=iteración+1;
                        Si (Δ < Δ_m)
                                Δ= Δ_m;
                        Fin Si
                        Si (c<T_final  O iteración>=n_iter)
                                parar=1;
                        Fin Si
                Fin Mientras
                Enviar(padre, Lop);
                Enviar(padre, c, Δ);
        Fin Mientras
```

CAPÍTULO 6

RESULTADOS EXPERIMENTALES

En este Capítulo se efectúa el ajuste y la caracterización del procedimiento RRMIN2. Asimismo se presentan los resultados que es capaz de proporcionar, comparándolos con otros procedimientos de minimización. Finalmente se estudia la versión paralela PRRMIN, analizando la ganancia de velocidad y eficacia que se consigue en una red de estaciones de trabajo.

6.1. INTRODUCCION

En el Capítulo 4 se ha desarrollado un procedimiento de minimización AND-EXOR denominado RRMIN2. Los apartados que siguen van a presentar los resultados experimentales obtenidos con una implementación en lenguaje C del procedimiento, denominada TESTRRMIN2. El programa está pensado para ejecutarse bajo sistemas operativos tipo UNIX, y se ha compilado y ejecutado con éxito en plataformas Intel Pentium con sistema operativo Linux, SUN SPARC con sistemas operativos SunOs 4.1.3 y Solaris 2.5.1, Power Challenge con sistema IRIX 6.2 e IBM RS6000 con sistema AIX 4.2. Como puede verse, se trata de una implementación portable (el lenguaje y las funciones utilizadas son completamente estándar) que puede utilizarse en cualquier tipo de ordenador. En el Apartado 6.2 se detallan las

posibilidades que ofrece este programa, que se va a utilizar para obtener los resultados experimentales.

En los apartados 4.6 y 4.7, al describir el procedimiento RRMIN2, ha ido apareciendo una serie de parámetros cuya determinación ha de hacerse experimentalmente. El ajuste de estos parámetros no es trivial debido a la gran cantidad de los mismos, y se aborda en el Apartado 6.3 con el objetivo de obtener las mejores prestaciones posibles del procedimiento. Una vez ajustados estos parámetros, en 6.4 se procede a la caracterización del procedimiento y a su comparación con otros métdos de minimización. El Apartado 6.5 muestra los resultados obtenidos en la minimización de funciones incompletamente especificadas, y 6.6 se dedica a la optimización de funciones con varias salidas.

Para el estudio del procedimiento de minimización paralelo PRRMIN se ha efectuado una implementación, denominada TESTPRRMIN, en la que se utiliza un software de dominio público, denominado PVM (Parallel Virtual Machine, [GEI94]), que permite utilizar procesamiento paralelo mediante paso de mensajes en sistemas de memoria compartida y distribuida. El Apartado 6.7 se dedica a la descripción de TESTPRRMIN, y el 6.8 al estudio de los resultados obtenidos con el mismo. Finalmente, en 6.9 se presentan las conclusiones del capítulo.

6.2. EL PROGRAMA TESTRRMIN2

Tal y como se ha comentado en la introducción, se ha desarrollado una implementación en C del procedimiento RRMIN2 dirigida al estudio experimental del mismo. El nombre que se le ha dado a este programa es TESTRRMIN2, y permite una serie de opciones que se detallan a continuación:

1) Minimización AND-EXOR mediante el procedimiento RRMIN2 de una función (con una o varias salidas, completa o incompletamente especificadas) partiendo de su tabla de verdad. La expresión de la función debe estar almacenada en un fichero en formato Espresso [BRA84] o en formato RRMINM. La salida puede obtenerse en formato RRMINL o Verilog [PAL96] (véase el Apéndice C para una descripción de los formatos de fichero, herramientas de conversión desarrolladas, etc).
2) Minimización AND-EXOR mediante el procedimiento RRMIN2 de una función partiendo de una expresión AND-EXOR de la misma. Dicha expresión debe suministrarse mediante un fichero en formato RRMINL. La salida se obtiene en los mismos formatos que en 1).
3) Minimización AND-EXOR mediante el procedimiento RRMIN2 de una serie de funciones generadas aleatoriamente. Las funciones son generadas por el propio programa, y tras la minimización se presentan una serie de resultados estadísticos cuyo contenido y significado se detalla más adelante.

En caso de seleccionarse la opción 1), el programa precisa las siguientes entradas:

- Nombre y formato del fichero que contiene la función.
- Nombre y formato del fichero de salida.
- Temperaturas inicial y final, incremento de temperatura, parámetro b y longitud l_0 de las cadenas de Markov, del proceso de Enfriamiento Simulado.
- Probabilidades de aplicación de cada uno de los grupos de reglas.

Si se selecciona la opción 2), se precisará:

- Nombre del fichero de entrada
- Nombre y formato del fichero de salida
- Temperaturas inicial y final, incremento de temperatura, parámetro b y longitud l_0 de las cadenas de Markov del proceso de Enframiento Simulado.
- Probabilidades de aplicación de cada uno de los grupos de reglas.

Finalmente, la opción 3) requiere:

- Número de variables (n).
- Número de salidas (n_func).
- Número de unos por salida (n_unos).
- Número de indiferencias por salida (n_indif).
- Número de funciones a generar (n_pru).
- Temperaturas inicial y final, incremento de temperatura, parámetro b y longitud l de las cadenas de Markov, del proceso de Enframiento Simulado (c_0, c_f, Δ_0, b, l_0).
- Probabilidades de aplicación de cada uno de los grupos de reglas (p_regla1i, p_regla2i, p_regla3i).
- Nombre y formato del fichero de salida.
- Nombre del fichero de resultados estadísticos.

A todos los parámetros de entrada se les puede asignar valores por defecto mediante un fichero de configuración. De esta forma, una vez que se obtengan los valores óptimos para los parámetros en el Apartado 6.3, y se introduzcan en el fichero de configuración, el usuario podrá minimizar las funciones que desee sin necesidad de conocer internamente cómo opera el procedimiento.

Como salida, el programa proporciona las funciones minimizadas junto con los siguientes valores para cada una de ellas:

- Número de términos producto de la función.
- TRR obtenido con respecto a la expresión inicial (suma de minterms si se está en las opciones 1 ó 3; expresión inicial AND-EXOR si se elige la opción 2).
- LRR (Literal Reduction Rate [LLO93]) obtenido con respecto a la expresión inicial (suma de minterms en caso de elegir las opciones 1 ó 3; expresión inicial AND-EXOR

si se escogió la opción 2).

En caso de que se seleccione la opción 3, además se proporcionan una serie de resultados estadísticos, que son los siguientes:

- Número medio de términos producto de las funciones minimizadas (MP).
- Desviación típica del número medio de términos producto (σ_{MP}).
- Intervalo de error para el número medio de productos con una confianza del 95% [PHE84] (E_{MP}).
- TRR medio obtenido con respecto a la expresión como suma de minterms (TRR).
- TRR medio obtenido con respecto a la expresión MPRM óptima (TRRM).
- Número medio de literales de las funciones minimizadas (ML).
- Desviación típica del número medio de literales (σ_{ML}).
- Intervalo de error para el número medio de literales con una confianza del 95% [ref] (E_{ML}).
- LRR medio obtenido (LRR).
- Número de funciones que sería necesario minimizar (tamaño de la muestra) para que el valor medio del número de productos tenga un error máximo de ±0.1 con una confianza del 95%. Si el número de funciones que se han minimizado supera este valor, el número medio de términos producto obtenidos se habrá obtenido con ese error y esa confianza (N_areal).

De esta forma, el programa permite obtener todos los datos necesarios para la caracterización de RRMIN2, así como su comparación con otros procedimientos. En el apartado siguiente se detalla el ajuste de una serie de parámetros de RRMIN2 que es preciso determinar experimentalmente.

6.3. AJUSTE Y CARACTERIZACIÓN DEL PROCEDIMIENTO RRMIN2

En la descripción efectuada en el Capítulo 4 del procedimiento de minimización RRMIN2, quedaron algunos parámetros sin especificar, debido al carácter experimental de los mismos. En este apartado se van a obtener los valores óptimos correspondientes a esos parámetros, utilizando para ello el programa TESTRRMIN2 comentado anteriormente. Posteriormente se efectuará una caracterización del comportamiento del procedimiento en función de los parámetros de control del Enfriamiento Simulado.

6.3.1. Ajuste de los parámetros de RRMIN2

El procedimiento RRMIN2, según se estudió en el Capítulo 4, requiere una serie de parámetros para poder ser aplicado sobre una función. Estos parámetros pueden agruparse de la siguiente forma:

1) Parámetros característicos del Enfriamiento Simulado:

1.a) Parámetros de control del Enfriamiento Simulado (Apartado 4.7.2):
- c_0 (Temperatura inicial)
- Δ_0 (Incremento inicial de temperatura)
- c_f (Temperatura final)
- b (Parámetro de disminución de Δ_0

1.b) Longitud inicial de las cadenas de Markov:
- l_0

2) Parámetros característicos de RRMIN2:

2.a) Probabilidades de aplicación de las reglas de reescritura:
- p_regla1i, p_regla2i, p_regla3i (Apartado 4.6.5).

2.b) Probabilidades de fallo al aplicar las reglas del grupo 1:
- p_{falloi} (Ecuación 4.117, Apartado 4.6.2).
- p_{falloj} (Ecuación 4.119, Apartado 4.6.2).

3) Parámetros característicos del procedimiento de asignación de indiferencias ASIGMIN:
- par_produndidad (Apartado 4.8.1)

4) Parámetros para minimización multifuncional:
- tdiv (Apartado 4.9.2)

Se tiene pues una gran cantidad de parámetros, de manera que la tarea de encontrar el valor óptimo de los mismos constituye un problema muy complejo. Para resolverlo debe tenerse en cuenta que sería deseable que el procedimiento se controle externamente (de cara al usuario) mediante los parámetros de control del Enfriamiento Simulado (listados en 1.a). Esto implica el ajuste previo del resto de los parámetros, que debe efectuarse con una estrategia adecuada, partiendo de unos valores razonables para aquellos parámetros cuyo comportamiento sea más o menos conocido, y ajustando en primer lugar los parámetros sobre los que se posea menor información. Concretamente, el proceso que se va a seguir es el siguiente:

1) Fijar los parámetros del proceso de Enfriamiento Simulado a valores usuales
2) Fijar los valores de p_{falloi} y p_{falloj}.
3) Ajustar las probabilidades de aplicación de las reglas del grupo 2.
4) Ajustar la longitud de las cadenas de Markov, l_0.
5) Obtener los valores de p_{falloi} y p_{falloj} que minimicen el tiempo de ejecución.
6) Optimizar el valor de par_profundidad.
7) Obtener el mejor valor de t_{div}.

A continuación se detallan los pasos anteriores:

1) **Fijar los parámetros del proceso de Enfriamiento Simulado a valores usuales.** Así se va a prefijar unos valores para c_0, c_f, b y l_0; dejando el incremento de temperatura Δ_0 para controlar el tiempo de ejecución. Para la elección de las temperaturas inicial y final se seguirá el criterio de tratar de cubrir todo el rango de temperaturas, es decir, desde temperaturas a las que la probabilidad de efectuar una transición que aumente la función de coste sea cercana a 1 (temperaturas altas); hasta temperaturas a las que esa probabilidad sea cercana a 0 (véase discución al respecto realizada en 4.7.2). Para la temperatura inicial, en [AAR90] se propone que sea tal que la razón de aceptación $\varkappa$ sea próxima a 1. La razón de aceptación se define como el cociente entre las transiciones aceptadas y las posibles a una temperatura dada. Esta razón de aceptación, aproximadamente, puede expresarse como:

$$\chi \simeq \frac{m_1 + m_2 \cdot \exp\left(\frac{-\overline{\Delta f}^+}{c}\right)}{m_1 + m_2} \tag{6.1}$$

donde f es la función de coste (Apartado 4.2), m_1 es el número de transiciones posibles de un estado i a otro j tales que $f(j) \leq f(i)$, m_2 es el úmero de transiciones a las que $f(j) > f(i)$, y $\overline{\Delta f}^+$ es la media del incremento de coste de las m_2 transiciones a estados de mayor coste. En el caso de RRMIN2, en un estado determinado i, se tiene que $m_2 >> m_1$ (normalmente es difícil aplicar alguna regla de simplificación, y en cambio, existen del orden de 3^n reglas de expansión aplicables en cada estado. Véase el Apéndice B para un estudio más detallado). Por tanto, 6.1 puede aproximarse por:

$$\chi \simeq \exp\left(\frac{-\overline{\Delta f}^+}{c}\right) \tag{6.2}$$

En cuanto al incremento medio del coste, $\overline{\Delta f}^+$, las reglas de expansión del grupo 3 implican un incremento de 1 y la del grupo 4 de 2. Para evitar que la función de coste aumente excesivamente (sobre todo a temperaturas altas), debe de restringirse la posibilidad de aplicar la regla del grupo 4 (por su propia naturaleza se podría aplicar en todos los casos y supone un incremento de coste elevado). Por ello, se va a tomar una temperatura inicial a la que la probabilidad de aceptación sea de 0.95, pero para transiciones con un incremento medio de coste de 1 (si se efectúa un cálculo del incremento medio de coste para las reglas de reescritura propuesto, el incremento medio de coste que se obtendría es 2, véase Apéndice B). Se obtiene en ese caso que:

$$0.95 = \exp\left(-\frac{1}{c_0}\right) \tag{6.3}$$

Despejando en esa expresión, se obtiene que:

$$c_0 = -\frac{1}{\ln(0.95)} \approx 20 \tag{6.4}$$

Para la elección de la temperatura final, puede elegirse aquella a la que la probabilidad de aumentar de la función de coste en una unidad sea de 0.05, o sea:

$$c_f = -\frac{1}{\ln(0.05)} \approx 0.33 \tag{6.5}$$

Finalmente, como parámetro b se ha optado por un valor que haga disminuir en un 0.1% el decremento de temperatura en cada iteración:

$$b = 1.0001 \tag{6.6}$$

En cuanto a la longitud de las cadenas de Markov, l_0 (un valor constante, tal y como se detalló en 4.7.2), se suele utilizar un valor del orden del número de transiciones posibles a partir de un determinado estado (tamaño de la vecindad) [AAR90]. En el caso de RRMIN2, ese número de transiciones es, al menos, del orden de 3^n (El número de posibilidades de aplicar la regla del grupo 4 a una función de n variables es precisamente 3^n); un número muy elevado que daría lugar a tiempos de ejecución muy grandes. Para evitar este inconveniente se ha optado por utilizar para los ajustes del resto de los parámetros, una longitud razonablemente larga (a partir de la cual se cumplen las estimaciones efectuadas en [AAR90] que permiten utilizar un valor no dependiente de la temperatura):

$$l_0 = 100 \tag{6.7}$$

Posteriormente, en 6.3.1.2 se obtendrá el valor óptimo de l_0, comprobándose que resulta adecuado utilizar longitudes cortas para mejorar el tiempo de ejecucion.

2) **Fijar los valores de p_{falloi} y p_{falloj}.** Estos parámetros representan las probabilidades de no encontrar una primera y una segunda componente sobre la que aplicar una regla del grupo 2, respectivamente; y en principio pueden asignárseles valores teóricos. Pueden fijarse a un valor bajo (de forma que se garantice con bastante seguridad que si es posible aplicar una regla de este grupo, en efecto se va a aplicar), pero sin que ello implique un número de intentos exageradamente grande para aplicar la regla. Un valor inicial podría ser 0.1 para ambas probabilidades:

$$\begin{aligned} p_{falloi} &= 0.1 \\ p_{falloj} &= 0.1 \end{aligned} \tag{6.8}$$

3) **Ajustar las probabilidades de aplicación de las reglas del grupo 2.** Estos tres parámetros son los menos previsibles en cuanto a su comportamiento, y por tanto, los primeros que deben ajustarse. Así, se van a ir probando valores para las probabilidades de aplicación de las reglas, hasta conseguir encontrar aquellos que proporcionan los mejores resultados. Nótese que al tratarse de probabilidades, no es necesario que el valor de estos parámetros sea demasiado preciso, pudiendo variarse de 0.05 en 0.05 por ejemplo.

4) Una vez determinadas las probabilidades de aplicación de las reglas, **ajustar la longitud de las cadenas de Markov, l_0**, hasta obtener el resultado de referencia en el menor tiempo posible.

5) **Obtener los valores de p_{falloi} y p_{falloj} que minimicen el tiempo de ejecución** necesario para obtener los resultados anteriores.

Con este último ajuste se obtendrían los valores para todos los parámetros de RRMIN2 necesarios para la minimización de funciones completamente especificadas con una sóla salida. Posteriormente, sería necesario ajustar los parámetros par_profundidad y tdiv para poder efectuar la minimización de funciones incompletamente especificadas y con varias salidas, respectivamente. Por tanto:

6) **Optimizar el valor de par_profundidad.**
7) **Obtener el mejor valor de tdiv.**

En los apartados que siguen se efectúan los ajustes mencionados en 3), 4) y 5). Todos estos ajustes se han efectuado en una estación de trabajo SUN ULTRA 1 con sistema operativo Solaris 2.5.1.

6.3.1.1. Ajuste de las probabilidades de aplicación de las reglas

Se va a comenzar, pues, por la optimización de los parámetros p_regla1i, p_regla2i y p_regla3i. Utilizando los valores prefijados en las ecuaciones (6.4) a (6.8), y asignando un incremento de temperatura Δ_0=0.5, se han realizado los experimentos que se muestran en la Tabla 6.1. En dicha tabla se presentan los valores medios para el número de productos (MP) y literales (ML) obtenidos para distintas combinaciones de estos parámetros, así como las desviaciones típicas correspondientes (σ_{MP} y σ_{ML}) y los errores máximos con un 95% de confianza (E_{MP} y E_{ML}). Nótese que se ha partido de la igualdad de probabilidades para los cuatros grupos de reglas, y posteriormente se han probado todas las combinaciones posibles con un aumento de p_regla2i y p_regla1i de 0.05. El ensayar estos valores se justifica en base a que el mejor resultado debe obtenerse con una combinación de probabilidades en la que p_regla1i y p_regla2i sean bastante mayores que p_regla3i y p_regla4i (el procedimiento debe

buscar la simplificación de las funciones, salvo en un pequeño número de casos en los que se producirán expansiones para salir de mínimos locales), según se deduce de lo expuesto en 3.2.2 sobre la minimización mediante reglas de reescritura.

En el caso de las pruebas efectuadas en la Tabla 6.1, el mejor resultado se obtiene en la fila en negrita (con las desviaciones típicas obtenidas, y teniendo en cuenta que el resultado más proximo es MP=13.09, si se efectúa un contraste de hipótesis se puede asegurar con un 95% de confianza que, en efecto, esa combinación es la mejor de las analizadas). Como era de esperar, la combinación de probabilidades supone un valor mayor para p_regla1 y p_regla2 que para los otros dos grupos de reglas. El proceso a seguir a continuación consistirá en obtener resultados para todas las combinaciones de probabilidades que se encuentren alrededor de ésta (introduciendo variaciones de 0.05 en los valores de las probabilidades, según se ha comentado anteriormente). Se obtiene así la Tabla 6.2, en la cual se han marcado en negrita las dos mejores combinaciones (ambas obtienen resultados muy parecidos, siendo indistinguibles estadísticamente). En consecuencia, deben examinarse las combinaciones alrededor de las dos combinaciones marcadas en negrita, revisando nuevamente las medias obtenidas. Repitiendo el proceso sucesivamente, se llega a una situación en la que no es posible encontrar mejores resultados. Para el caso que se está tratando, de funciones de 6 variables con un 50% de unos, esa mejor combinación ha resultado ser la presentada en la Tabla 6.3. Para la obtención de este último valor se han minimizado 2000 funciones distintas para poder asegurar que el valor medio 11.60 obtenido para MP es estadísticamente distinto del valor 11.66 obtenido para otra combinación).

Naturalmente, el ajuste de las probabilidades de aplicación de grupos de reglas no debe restringirse a un sólo caso particular, sino que debe efectuarse para distintos números de variables y porcentajes de unos. Así, la experiencia se ha repetido para n=6, 8 y 10 variables y un porcentaje de unos del 25% y 50% (el comportamiento de RRMIN2 para funciones con un 75% de unos debe ser prácticamente el mismo que en el caso de un 25%, según el Apartado 4.8). En la Tabla 6.4 aparece la recopilación de los resultados obtenidos, observándose que en todos los casos la mejor combinación de probabilidades es, con un 95% de confianza, p_regla1i=0.20, p_regla2i=0.50, p_regla3i=0.25, p_regla4i=0.05 con un error máximo de ±0.05. El hecho de que esta combinación de probabilidades sea la misma independientemente del porcentaje de unos de la función y del número de variables de la misma simplifica notablemente el ajuste del procedimiento.

6.3.1.2. Optimización de la longitud de las cadenas de Markov

El siguiente parámetro que se va a optimizar es la longitud de las cadenas de Markov. En principio, según los estudios teóricos reflejados en [AAR90], son deseables cadenas de Markov largas y dependientes de la temperatura ($l_k \to \infty$ cuando $c_k \to 0$) para garantizar las condiciones de convergencia. Sin embargo, como se observará en los resultados obtenidos, el procedimiento RRMIN2 no presenta problemas de convergencia, y la utilización de cadenas cortas permite ahorrar tiempo de ejecución.

p_regla1i	p_regla2i	p_regla3i	MP	σ_{MP}	E_{MP}	ML	σ_{ML}	E_{ML}
0.25	0.25	0.25	14.04	1.41	0.12	54.5	8.1	0.7
0.20	0.30	0.25	14.09	1.41	0.12	53.8	8.0	0.7
0.25	**0.30**	**0.20**	**12.75**	**1.32**	**0.12**	**48.8**	**7.5**	**0.7**
0.25	0.30	0.25	13.09	1.41	0.12	49.6	7.9	0.7
0.30	0.20	0.25	14.32	1.52	0.13	55.9	8.8	0.7
0.30	0.25	0.30	13.08	1.35	0.12	50.6	7.7	0.7
0.30	0.25	0.25	13.54	1.59	0.14	52.1	9.1	0.8

Tabla 6.1. Primera tabla ajuste probabilidades de aplicación de reglas. Parámetros utilizados:
n=6, n_unos=32, n_indif=0, n_func=1, l_0=100, c_0=20, Δ_0=0.5, c_f=0.33, b=1.0001, n_pru=500, p_{falloi}=0.1, p_{falloj}=0.1

p_regla1i	p_regla2i	p_regla3i	MP	σ_{MP}	E_{MP}	ML	σ_{ML}	E_{ML}
0.20	0.35	0.20	12.73	1.34	0,11	49.2	7.7	0.7
0.25	**0.35**	**0.15**	**12.15**	**1.23**	**0,11**	**46.4**	**6.9**	**0.6**
0.25	0.35	0.20	12.26	1.33	0,12	46.4	7.5	0.7
0.30	0.30	0.20	12.51	1.19	0,10	48.7	6.8	0.6
0.30	**0.30**	**0.15**	**12.14**	**1.49**	**0,13**	**46.7**	**8.4**	**0.7**

Tabla 6.2. Segunda tabla ajuste probabilidades de aplicación de reglas. Parámetros utilizados:
n=6, n_unos=32, n_indif=0, n_func=1, l_0=100, c_0=20, Δ_c=0.5, c_f=0.33, b=1.0001, n_pru=500, p_{falloi}=0.1, p_{falloj}=0.1

p_regla1i	p_regla2i	p_regla3i	MP	σ_{MP}	E_{MP}	ML	σ_{ML}	E_{ML}
0.20	0.50	0.25	11.60	1.18	0.10	44.1	6.7	0.6

Tabla 6.3. Resultado ajuste probabilidades de aplicación de reglas para 6 variables y un 50% de unos.
Parámetros utilizados:
n=6, n_unos=32, n_indif=0, n_func=1, l_0=100, c_0=20, Δ_c=0.5, c_f=0.33, b=1.0001, n_pru=2000, p_{falloi}=0.1, p_{falloj}=0.1

l_0	Δ_0	TRR	σ_{TRR}	E_{TRR}	MP	σ_{MP}	E_{MP}	ML	σ_{ML}	E_{ML}	TF(s)
100	0.5	0.367	0.021	0.003	46.9	2.7	0.4	246	18	2	30.8
50	0.25	0.367	0.022	0.003	47.0	2.8	0.4	249	18	3	30.6
20	0.10	0.366	0.022	0.003	46.9	2.8	0.4	247	18	3	30.3
10	0.05	0.366	0.021	0.003	46.8	2.7	0.4	247	17	2	30.2
6	0.032	0.366	0.022	0.003	46.9	2.8	0.4	249	18	3	27.9
5	**0.03**	**0.364**	**0.021**	**0.003**	**46.6**	**2.8**	**0.4**	**242**	**18**	**3**	**23.9**
4	0.022	0.366	0.022	0.003	46.8	2.6	0.4	249	17	2	28.0
1	0.0065	0.366	0.023	0.003	46.8	2.9	0 4	248	19	3	29.0

Tabla 6.4. Resultados ajuste de las longitudes de las cadenas de Markov para 8 variables. Parámetros utilizados: n=8, n_indif=0, n_func=1, n_unos= 128, c_0=20, c_1=0.33, b=1.0001, n_pru=200, p_{falloi}=0.1, p_{falloj}=0.1

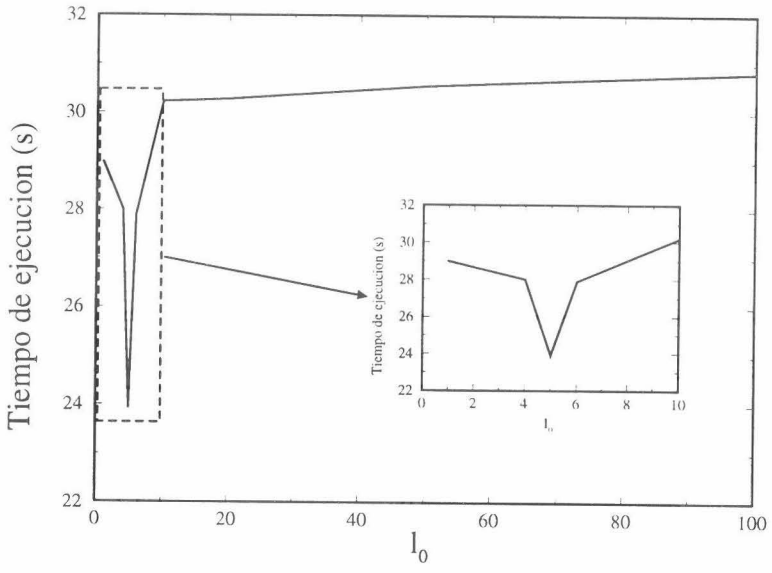


Fig. 6.1. Tiempo de ejecución frente a l_0 para n=8 variables y un 50% de unos. TRR exigido: 0.37

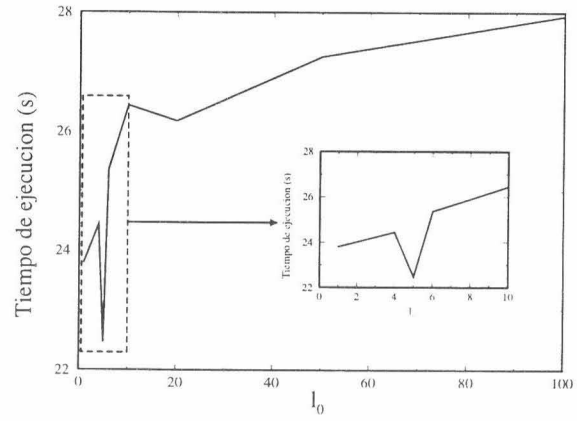


Fig. 6.2. Tiempo de ejecución frente a l_0 para n=8 variables y un 25% de unos
TRR exigido: 0.54

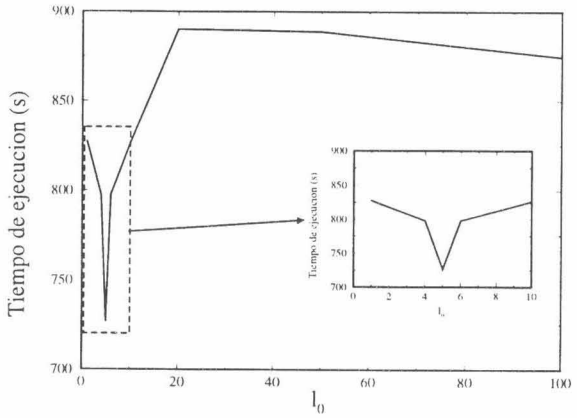


Fig. 6.3. Tiempo de ejecución frente a l_0 para n=10 variables y un 50% de unos.
TRR exigido: 0.40

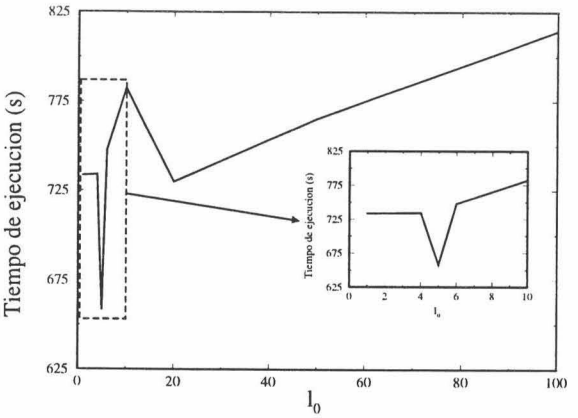


Fig. 6.4. Tiempo de ejecución frente a l_0 para n=10 variables y un 25% de unos.
TRR exigido: 0.54

Las experiencias que se han realizado consisten en fijar una determinada calidad para la solución (exigiendo un TRR determinado en cada caso), y medir el tiempo de ejecución (TF) necesario para distintos valores de l_0. El error máximo para TF que se ha obtenido para este tipo de experimentos es de ±(0.01s+1% de TF) (valor hallado a partir de las desviaciones típicas de los tiempos necesarios para minimizar varias series de funciones de 6, 8 y 10 variables), que se tiene en cuenta en esta tabla y en las posteriores. El tiempo de ejecución (y por tanto la calidad de la solución) se ha controlado mediante el incremento inicial de temperatura Δ_0, manteniendo fijas las temperaturas inicial y final, asi como el resto de los parámetros a los valores que aparecen en la Tabla 6.5. En las figuras 6.1 a 6.4 se representan gráficamente los tiempos obtenidos para 8 y 10 variables con un 25% y un 50% de unos utilizando esos mismos parámetros (el caso de funciones con un 75% de unos es equivalente al de un 25% según 4.8). En todos los casos se ha obtenido que l_0=5 es el valor óptimo; unas cadenas de Markov muy cortas. Esto se debe a que en el Enfriamiento Simulado (al igual que ocurre en el proceso físico de enfrimiento de cristales), es muy importante que la temperatura baje lentamente. Si las cadenas de Markov son cortas, se consigue que se pueda utilizar un incremento de temperatura más pequeño con el mismo tiempo de ejecución. Por tanto, de cara a optimizar el tiempo de ejecución, el procedimiento se comporta mejor si se baja la temperatura muy lentamente, aunque las cadenas de Markov sean cortas (lo que equivale a efectuar pocas transiciones a cada temperatura) que si se utilizan cadenas largas e incrementos de temperaturas mayores. Más adelante, en el Apartado 6.4, (figuras 6.28 a 6.30) se observa cómo en efecto resulta más ventajoso disminuir el parámetro Δ_0 que aumentar l_0.

Nótese que esta forma de bajar la temperatura es sensiblemente distinta a la que se suele utilizar en los procesos de Enfriamiento Simulado. Normalmente, a cada temperatura se trata de alcanzar un estado de cuasi-equilibrio, mediante cadenas de Markov muy largas (del orden de la vecindad), consiguiéndose de esta forma mantener las condiciones que permiten asegurar la convergencia del procedimiento según la demostración incluida en [AAR90]. En el caso de RRMIN2, el tamaño de la vecindad no permite utilizar cadenas del orden de la la vecindad, y se ha ooptado por hacer descender la temperatura de forma muy lenta pero sin llegar a la condición de equilibrio en cada valor de c. Los resultados experimentales que se obtienen muestran que el algoritmo converge, aunque no puede asegurarse que si se utiliza este procedimiento con otras aplicaciones del Enfriamiento Simulado el comportamiento vaya a ser el mismo (una demostración teórica no sería en absoluto trivial puesto que se tienen condiciones de no equilibrio, habría que utilizar cadenas de Markov inhomogeneas, etc.).

Una vez ajustado este parámetro, sólo queda optimizar p_{falloi} y p_{falloj} con el único obejtivo de mejorar el tiempo de ejecución. El siguiente apartado se ocupa de ello.

6.3.1.3. Ajuste de la aplicación de las reglas del grupo 2

Finalmente, se va a tratar de optimizar el tiempo de ejecución variando los valores de p_{falloi} y p_{falloj}. Tal y como se ha comentado en otras ocasiones, estos parámetros representas las

probabilidades de no elegir la primera y la segunda componente, adecuadas para aplicar alguna de las reglas del grupo 2, respectivamente. En principio, estas probabilidades deberían mantenerse con valores pequeños para asegurar que si es posible aplicar una de estas reglas, en efecto va a ser aplicada. Sin embargo, en una gran cantidad de ocasiones, puede ocurrir que no sea posible realizar la aplicación de las reglas, y se esté desperdiciando tiempo de ejecución efectuando intentos inútiles. Por esta razón, es conveniente probar distintos valores de estos parámetros y optimizar así al máximo el tiempo de ejecución. La Tabla 6.5 presenta las experiencias y resultados para el caso de 8 variables y un 50% de unos. En la misma se observa que el valor de estos parámetros afecta de manera muy importante al tiempo de ejecución. En la Tabla 6.6 pueden verse los valores óptimos para distintos números de variables y porcentajes de unos.

p_{falloi}	p_{falloj}	Δ_0	MP	σ_{MP}	E_{MP}	ML	σ_{ML}	E_{ML}	TF(s)
0.1	0.1	0.03	47.0	2.97	0.4	248	19	3	46.9
0.2	0.1	0.025	47.0	2.78	0.4	243	18	2	46.2
0.3	0.1	0.023	46.1	2.81	0.4	240	18	3	40.2
0.4	0.1	0.0225	46.4	2.78	0.4	246	18	3	36.6
0.5	0.1	0.022	46.6	2.81	0.4	246	18	3	33.9
0.6	**0.1**	**0.019**	**46.2**	**2.73**	**0.4**	**243**	**18**	**2**	**33.2**
0.7	0.1	0.018	47.6	2.92	0.4	251	19	3	31.3

Tabla 6.5. Resultados ajuste probabilidades de fallo aplicación reglas grupo 2 para 8 variables y un 50% de unos.
Parámetros utilizados:
n=8, n_indif=0, n_func=1, n_unos= 128, c_0=20, c_l=0.33, b=1.0001, l_0=5, n_pru=200

n	n_unos	p_{falloi}	p_{falloj}
6	32	0.4	0.1
6	16	0.4	0.1
8	128	0.6	0.1
8	64	0.6	0.1
10	512	0.7	0.1
10	256	0.7	0.1
11	1024	0.8	0.1

Tabla 6.6. Resultados ajuste probabilidades de fallo aplicación reglas grupo2

En general, se observa que p_{falloj} no afecta al proceso de ajuste y su valor óptimo es siempre 0.1, y que p_{falloi} toma valores bastante elevados. La explicación de estos resultados es la siguiente:

- p_{falloj} es la razón máxima de fallo que se admite al elegir la segunda componente sobre la que aplicar las reglas del grupo 2 (4.119), una vez que se ha encontrado una primera componente válida. Puesto que el número de segundas componentes se encuentra bastante restringido según 4.6.2, no es necesario utilizar demasiado tiempo de ejecución a su comprobación, y siempre será mejor asegurarse de que en efecto puede aplicarse una regla de este grupo. Por tanto, es lógico que el valor de p_{falloj} sea pequeño y no dependa prácticamente del número de variables.

- p_{falloi}, en cambio, va a delimitar el número de intentos para encontrar la primera componente sobre la que aplicar las reglas (4.118). Esto requiere ya un tiempo de ejecución mucho mayor puesto que existen un gran número de componentes a comprobar, y además para cada componente elegida habrá que verificar si existe una segunda componente que sea útil. De esta forma, si no es posible aplicar una regla del grupo 2, se estará desperdiciando una gran cantidad de tiempo efectuando todos los intentos citados. Esto lleva a que los valores de p_{falloi} óptimos tengan valores grandes y que dependan fuertemente del número de variables.

En la Figura 6.5 puede verse la representación gráfica de p_{falloi} frente al número de variables, junto con la recta de regresión lineal correspondiente a esos datos. Los parámetros correspondientes al ajuste por mínimos cuadrados son los siguientes:

$$\begin{array}{c} r=0.99 \\ a=0.076\pm0.008 \\ b=-0.04\pm0.07 \end{array} \qquad (6.9)$$

donde r es el coeficiente de correlación lineal, a la pendiente de la recta, y b la ordenada en el origen. El valor de r, próximo a 1, indica que, en efecto, los datos obtenidos pueden ajustarse mediante una recta y obtener la siguiente expresión para p_{falloi}:

$$p_{falloi}=\begin{cases} 0.076\cdot n-0.04 & si\ n<13 \\ 0.9 & si\ n\geq 13 \end{cases} \qquad (6.10)$$

donde se ha tenido en cuenta que p_{falloi} ha de ser menor que 1, y por tanto, a partir de 13 variables se va a fijar a valor 0.9.

Con el ajuste de estos dos parámetros, se tiene completado el proceso de ajuste del procedimiento para la minimización de funciones completamente especificadas con una salida.

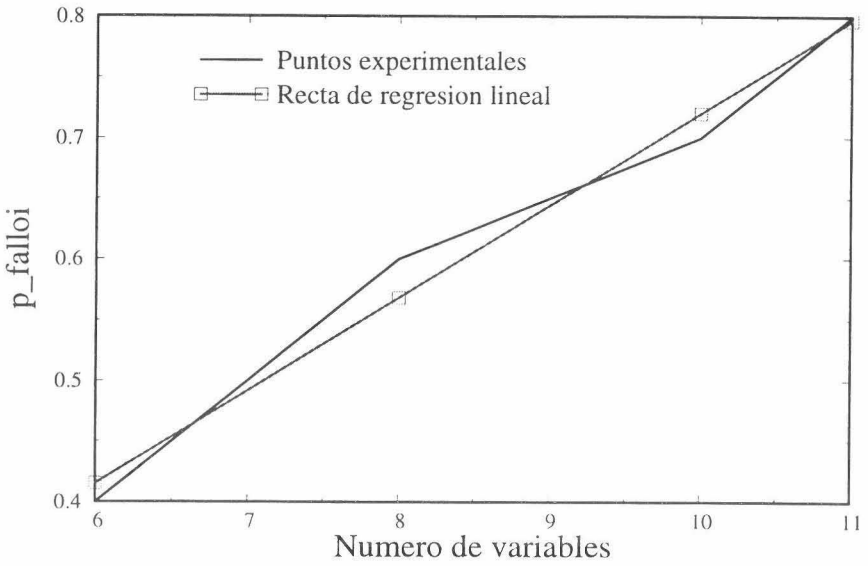


Fig. 6.5. Representación de p_{falloi} frente al número de variables n

A modo de resumen, en (6.11) se recogen los valores óptimos de los parámetros:

$$\begin{gathered} p\_regla1i=0.2 \\ p\_regla2i=0.5 \\ p\_regla3i=0.25 \\ l_0=5 \\ p_{falloj}=0.1 \end{gathered} \qquad (6.11)$$

$$p_{falloi}=\begin{cases} 0.076 \cdot n-0.04 & si\ n<13 \\ 0.9 & si\ n\geq 13 \end{cases}$$

y en la Figura 6.6 se presenta un diagrama de flujo del proceso de ajuste.

En los apartados siguientes se va a efectuar la caracterización del método en función de los parámetros de control del Enfriamiento Simulado. Posteriormente, se van a obtener resultados experimentales estadísticamente fiables y se van a comparar con otros procedimientos de minimización AND-EXOR.

6.4. CARACTERIZACIÓN DE RRMIN2

En este apartado se va a realizar la caracterización experimental del procedimiento de minimización RRMIN2 en función de los parámetros de control del proceso de Enfriamiento Simulado. Estos parámetros serán la temperatura inicial c_0, la temperatura final c_f, el incremento inicial de temperatura Δ_0 y el parámetro b.

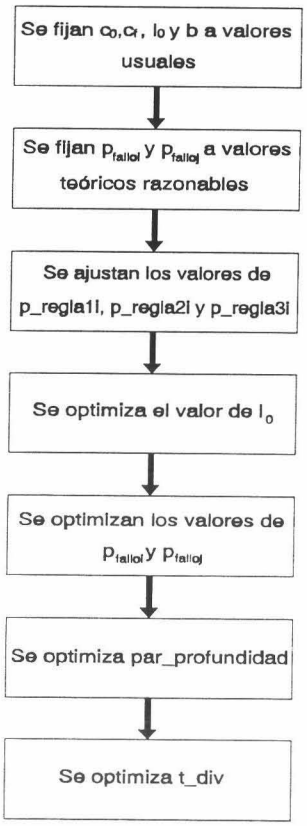


Fig. 6.6. Diagrama de flujo del proceso de ajuste de parámetros

c_f	MP	σ_{MP}	E_{MP}	TF (s)
0.001	11.32	0.99	0.06	4.73
0.01	11.33	0.98	0.06	4.64
0.1	11.31	1.02	0.06	4.64
0.33	11.32	1.02	0.06	4.54
1	11.36	0.99	0.06	4.36
2	11.37	1.07	0.07	4.19
5	11.42	1.04	0.07	3.33
10	11.60	1.08	0.07	2.16

Tabla 6.7. Número medio de productos para distintos valores de la temperatura final. Parámetros utilizados: n=6, n_indif=0, n_func=1, nunos=32, c_0=20, Δ_0=0.01, b=1.0001, n_pru=500

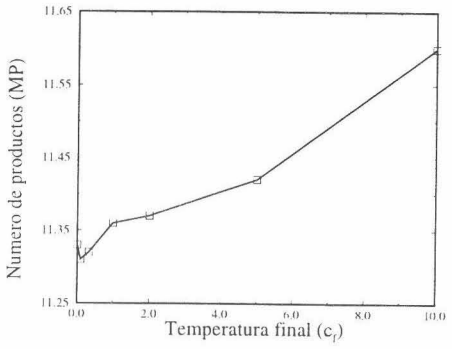


Fig. 6.7. Número medio de productos frente a la temperatura final para n=6 variables

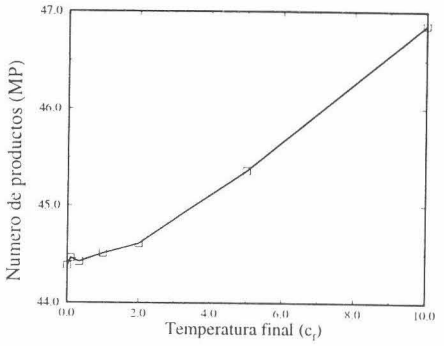


Fig. 6.8. Número medio de productos frente a la temperatura final para n=8 variables.

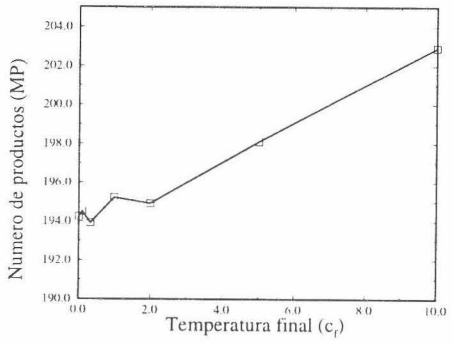


Fig. 6.9. Número medio de productos frente a la temperatura final para n=10 variables

En primer lugar se va a analizar el comportamiento del procedimiento en función de la temperatura final c_f, debido a que éste parámetro es el menos flexible en cuanto a su variación para tratar de conseguir mejores resultados. Un aumento de c_f debe provocar un empeoramiento de los resultados, mientras que la disminución de la temperatura final mejorará los resultados (puesto que se producirá un aumento del número de iteraciones a realizar en el proceso de Enfriamiento, véase el Apéndice B), pero de forma moderada (al analizar más adelante los efectos de la variación de otros parámetros como c_0 o Δ_0 se obtendrán mejoras mucho mayores). En la Tabla 6.7 se puede comprobar lo anterior. La Figura 6.7 muestra gráficamente los resultados correspondientes a la tabla, y las figuras 6.8 y 6.9 confirman este comportamiento para funciones con 8 y 10 variables. Nótese que a partir de c_f=0.33, ya no se producen mejoras significativas en los resultados y así, c_f no resulta ser un parámetro adecuado para mejorar los resultados mediante su modificación. Para experimentos posteriores se va a mantener c_f=0.33 ,como se ha venido haciendo hasta ahora.

El siguiente parámetro que se va a estudiar es la temperatura inicial c_0. El aumento de esta temperatura se traducirá en un incremento del número de iteraciones, y por tanto, debería suponer una mejora en los resultados que se obtengan. La Figura 6.10 así lo muestra para el caso de 6 variables. A diferencia de lo que ocurría con c_f, el aumento de la temperatura inicial permite un margen de mejora muy superior, aunque, como se verá un poco más adelante, se desperdicia tiempo de ejecución (lo cual resulta lógico, puesto que si la temperatura aumenta excesivamente, se aceptan demasiadas transiciones hacia estados de superior coste, disminuyendo la eficacia del procedimiento). Las figuras 6.11 y 6.12 muestran los resultados para n=8 y 10 variables, observándose claramente cómo disminuye el número medio de productos preciso para realizar las funciones a medida que se aumenta la temperatura inicial.

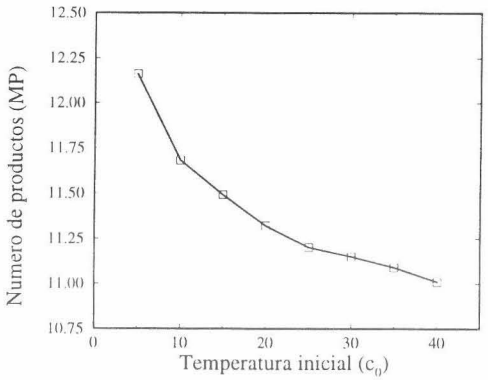


Fig. 6.10. Número medio de productos frente a la temperatura inicial c_0 para n=6 variables. Parámetros utilizados: n=6, Δ_0=0.01, c_f=0.33, b=1.0001, n_pru=500

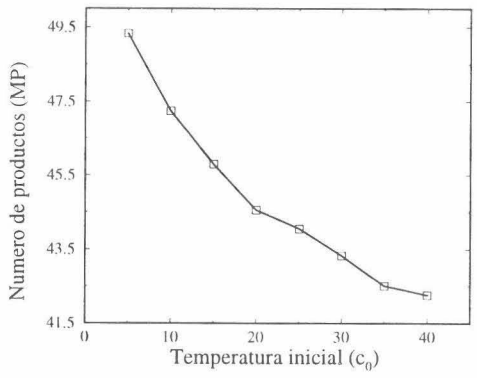


Fig. 6.11. Número medio de productos frente a la temperatura inicial para n=8 variables. Parámetros utilizados: Δ_0=0.01, c_f=0.33, b=1.0001, n_pru=200.

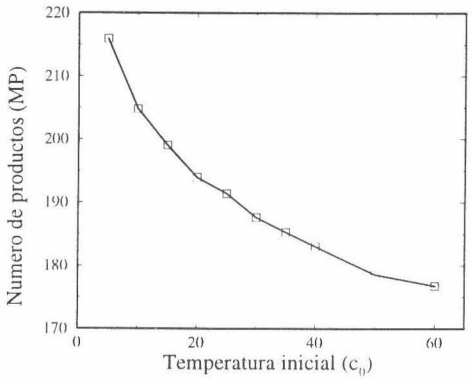


Fig. 6.12. Número medio de productos frente a la temperatura inicial c_0 para n=10 variables. Parámetros utilizados: Δ_0=0.01, c_f=0.33, b=1.0001, n_pru=100

Otro parámetro de control muy adecuado para mejorar las soluciones es el incremento inicial de temperatura Δ_0, puesto que permite un rango muy amplio de variación, y se puede seleccionar con precisión el grado de bondad en la solución final. En las figuras 6.13 a 6.15 se representa el número medio de productos frente al incremento inicial de temperatura para n=6, 8 y 10 variables, respectivamente, observándose cómo mejora la calidad de la solución al reducir Δ_0. Nótese que el margen de mejora obtenido es muy similar al que se presentaba en las figuras 6.10 a 6.12, en las que se hacía variar la temperatura inicial c_0, de manera que, en principio, ambos parámetros resultarían válidos para mejorar las soluciones.

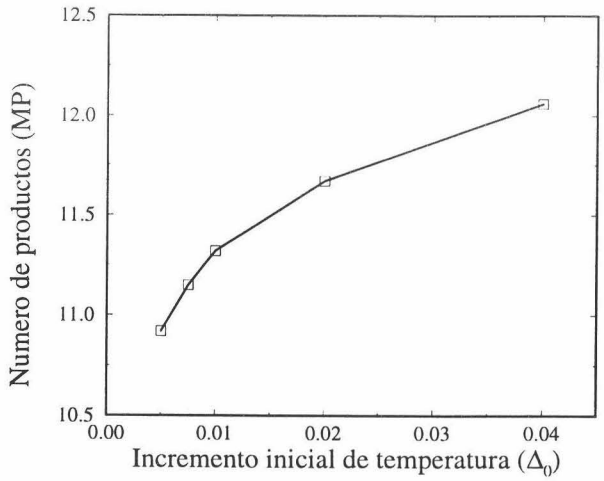


Fig. 6.13. Número medio de productos frente al incremento inicial de temperatura Δ_0 para n=6 variables. Parámetros utilizados: c_0=20, c_f=0.33, b=1.0001, npru=350

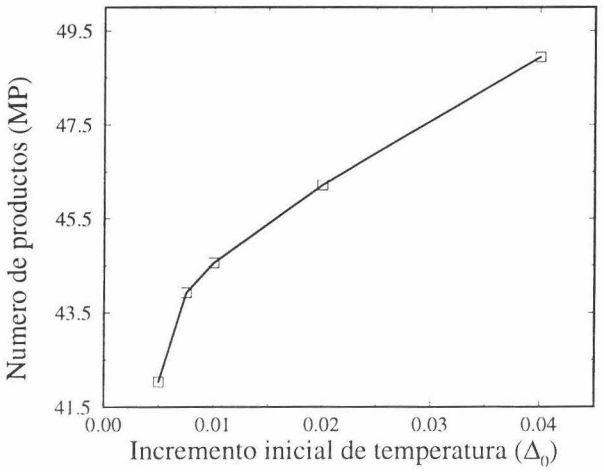


Fig. 6.14. Número medio de productos frente al incremento inicial de temperatura Δ_0 para n=8 variables. Parámetros utilizados: c_0=20, c_f=0.33, b=1.0001, npru=200

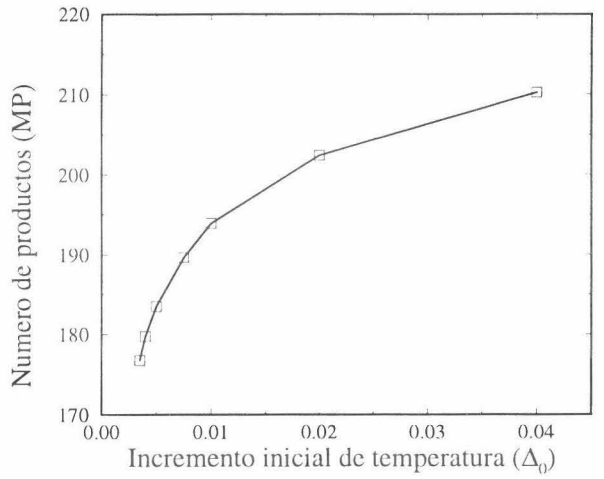


Fig. 6.15. Número medio de productos frente al incremento inicial de temperatura Δ_0 para n=10 variables. Parámetros utilizados: c_0=20, c_f=0.33, b=1.0001, npru=100

Si se realiza una representación del tiempo de ejecución frente el número medio de productos, tal y como se muestra en la Figura 6.16 para n=6 variables, puede verse que las soluciones de gran calidad precisan de menor tiempo para ser alcanzadas si se obtienen bajando Δ_0 que si se hace aumentar la temperatura inicial c_0. Lo mismo ocurre para n=8 y 10 variables, según se presenta en las figuras 6.17 y 6.18, por lo que se puede concluir que resulta más ventajoso modificar Δ_0 que la temperatura inicial para obtener resultados mejores. No obstante, c_0 presenta también un buen comportamiento, y se pueden utilizar ambos parámetros conjuntamente (haciendo aumentar c_0 y disminuir Δ_0) para obtener mejoras en las soluciones.

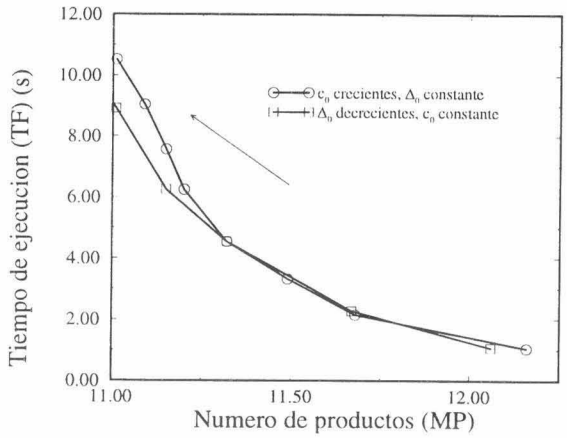


Fig. 6.16. Tiempo de ejecución frente al número medio de productos para n=6 variables variando c_0 y Δ_0. Parámetros utilizados: c_f=0.33, b=1.0001, n_pru=500, c_0=20 cuando varía Δ_0, Δ_0=0.01 cuando varía c_0.

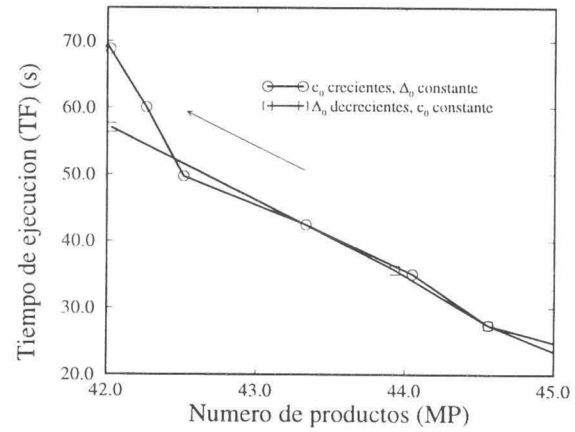


Fig. 6.17. Tiempo de ejecución frente al número medio de productos para n=8 variables variando c_0 y Δ_0. Parámetros utilizados: c_f=0.33, b=1.0001, n_pru=200, c_0=20 cuando varía Δ_0, Δ_0=0.01 cuando varía c_0.

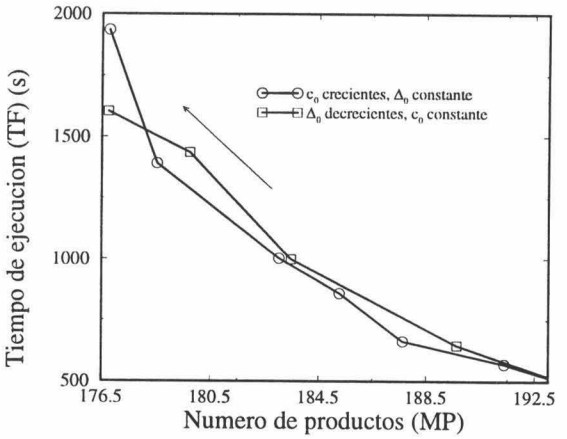


Fig.6.18.Tiempo de ejecución frente al número medio de productos para n=10 variables variando c_0 y Δ_0. Parámetros utilizados: c_f=0.33, b=1.0001, n_pru=100, c_0=20 cuando varía Δ_0, Δ_0=0.01 cuando varía c_0.

El siguiente parámetro de control que se va a analizar es el parámetro b introducido en 4.7.2. Un aumento de b implica un descenso más lento de la temperatura (el decremento de temperatura en la iteración k, Δ_k, se dividirá por un número mayor para pasar a la iteración k+1), y por tanto debe producirse una mejora en las soluciones obtenidas. Las gráficas 6.19 a 6.21 muestran estas mejoras al modificar b para funciones de n=6, 8 y 10 variables, respectivamente. No obstante, si se compara el tiempo de ejecución necesario para alcanzar esos resultados variando b y el que se obtuvo modificando el incremento inicial de temperatura Δ_0, se observa que, de nuevo, es mejor actuar sobre Δ_0. Las gráficas 6.22 a 6.24 muestran claramente este hecho, haciéndose especialmente patente la diferencia en el caso de soluciones de gran calidad.

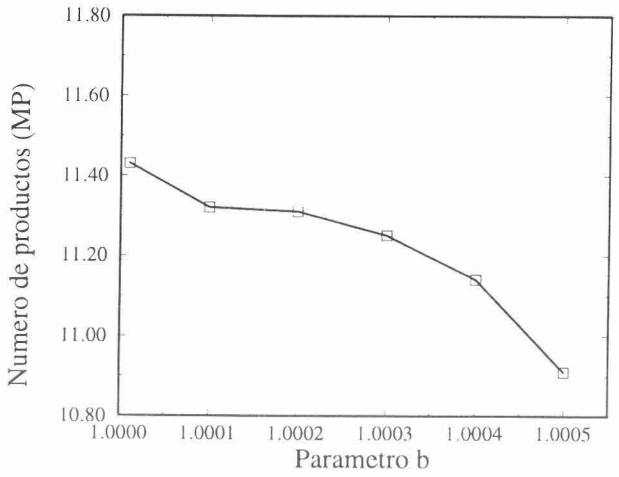


Fig. 6.19. Número medio de productos frente al parámetro b para n=6 variables.
Parámetros utilizados: c_0=20, Δ_0=0.01, c_t=0.33, n_pru=500

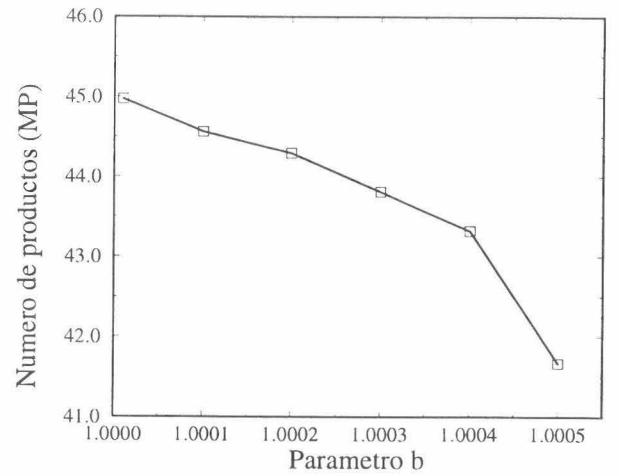


Fig. 6.20. Número medio de productos frente al parámetro b para n=8 variables.
Parámetros utilizados: c_0=20, Δ_0=0.01, c_t=0.33, n_pru=200

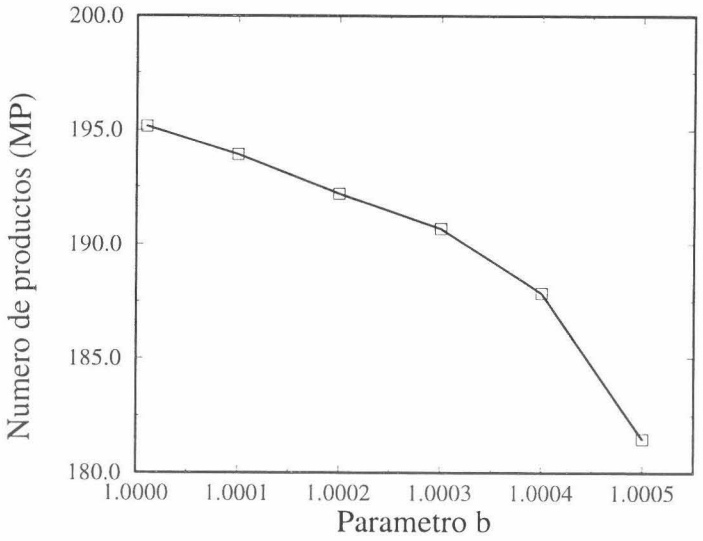


Fig. 6.21. Número medio de productos frente al parámetro b para n=10 variables. Parámetros utilizados: c_0=20, Δ_0=0.01, c_f=0.33, n_pru=100

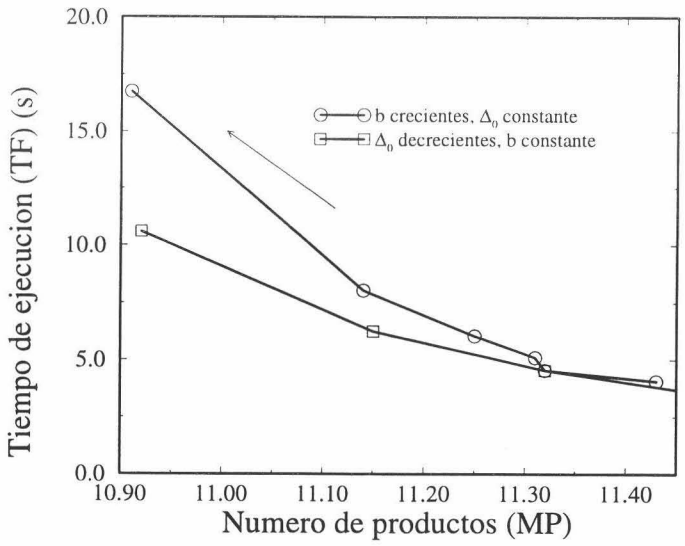


Fig 6.22. Tiempo de ejecución frente al número medio de productos para n=6 variables variando b y Δ_0. Parámetros utilizados: c_0=20, c_f=0.33, n_pru=500. Δ_0=0.01 cuando varía b, b=1.0001 cuando varía Δ_0.

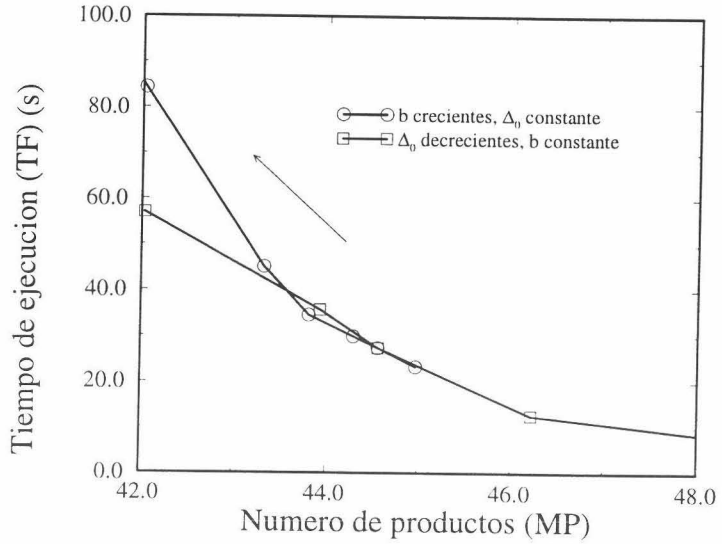


Fig 6.23. Tiempo de ejecución frente al número medio de productos para n=8 variables variando b y Δ_0. Parámetros utilizados: c_0=20, c_1=0.33, n_pru=200. Δ_0=0.01 cuando varía b, b=1.0001 cuando varía Δ_0.

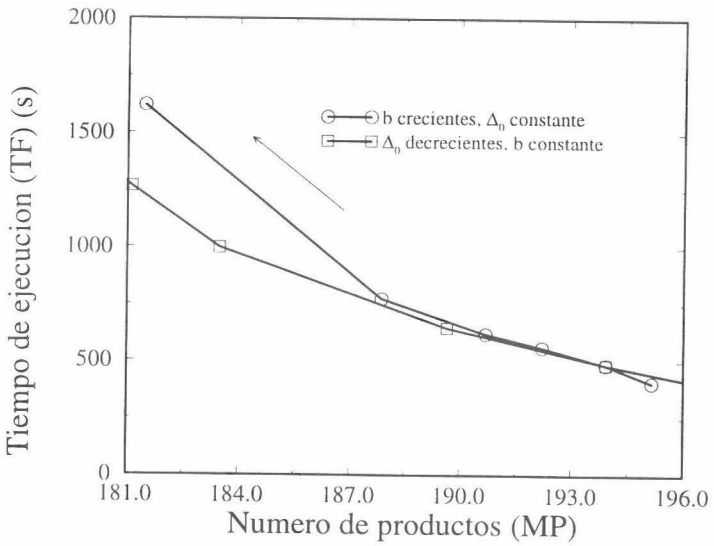


Fig 6.24. Tiempo de ejecución frente al número medio de productos para n=10 variables variando b y Δ_0. Parámetros utilizados: c_0=20, c_1=0.33, n_pru=100. Δ_0=0.01 cuando varía b, b=1.0001 cuando varía Δ_0.

Finalmente, para completar el estudio del RRMIN2 y corroborar las afirmaciones realizadas en 6.3.1.2, se va a analizar el comportamiento de la longitud inicial de las cadenas de Markov, l_0, comparando las mejoras que supone en la solución con las producidas por el decremento inicial Δ_0. En las gráficas 6.25 a 6.27 puede observarse que un aumento de l_0 produce una disminución del número de términos producto de las funciones a minimizar, pero las figuras 6.28 a 6.30 confirman que estas mejoras precisan de mayor tiempo de ejecución que las producidas por Δ_0, y por tanto, es preferible bajar la temperatura más lentamente que usar cadenas largas en el procedimiento RRMIN2. Además, el máximo acercamiento entre las curvas correspondientes a ambos parámetros se da en l_0=5 (de hecho, en ese valor, ambas curvas coinciden, como es lógico), que por tanto será el valor óptimo de l_0, coincidiendo con lo obtenido en 6.3.2. Nótese, sin embargo, que las diferencias de ejecución entre ambas series de curvas no son excesivamente grandes, de manera que también puede ser una muy buena opción combinar un aumento de este parámetro con una disminución del incremento inicial Δ_0 (y si se desea también un aumento de la temperatura inicial c_0) para generar soluciones de una alta calidad.

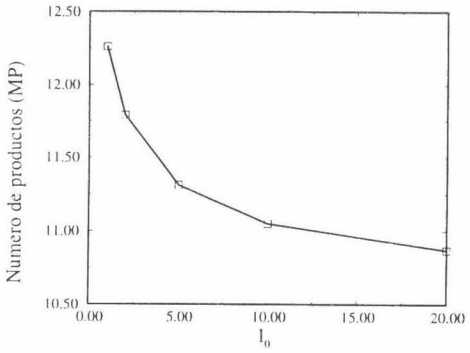


Fig. 6.25. Número medio de productos frente a l_0 para n=6 variables

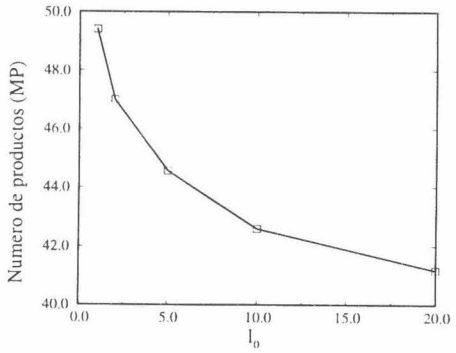


Fig. 6.26. Número medio de productos frente a l_0 para n=8 variables.

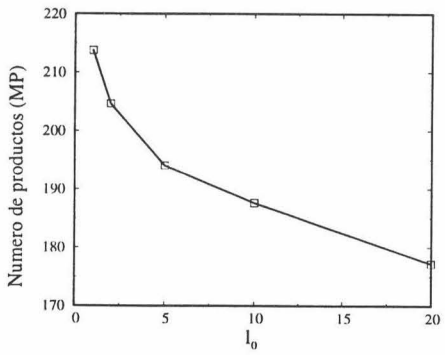


Fig. 6.27. Número medio de productos frente a l_0 para n=10 variables

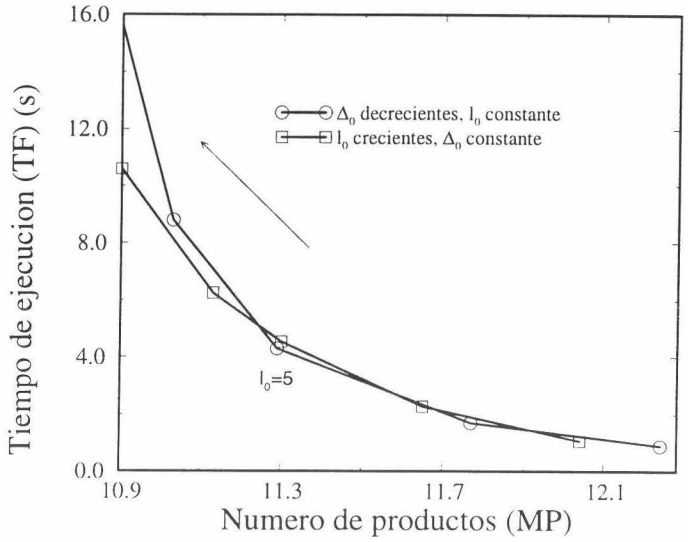


Fig. 6.28. Tiempo de ejecución frente al número medio de productos para n=6 variables variando l_0 y Δ_0. Parámetros utilizados: c_0=20, c_l=0.33, n_pru=500. Δ_0=0.01 cuando varía l_0,l_0=5 cuando varía Δ_0.

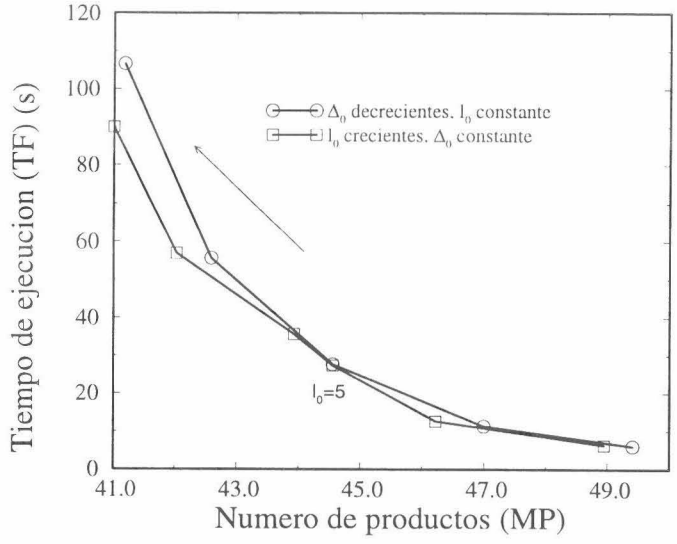


Fig. 6.29. Tiempo de ejecución frente al número medio de productos para n=8 variables variando l_0 y Δ_0. Parámetros utilizados: c_0=20, c_l=0.33, n_pru=200. Δ_0=0.01 cuando varía l_0,l_0=5 cuando varía Δ_0.

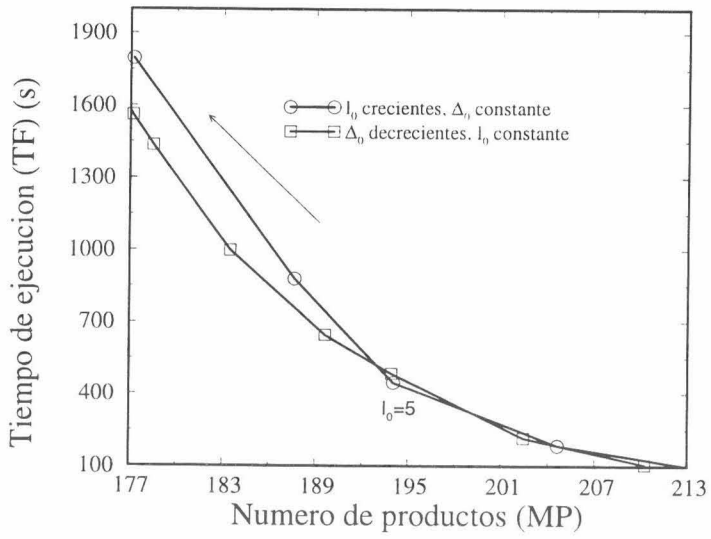


Fig. 6.30. Tiempo de ejecución frente al número medio de productos para n=10 variables variando l_0 y Δ_0. Parámetros utilizados: c_0=20, c_f=0.33, n_pru=100. Δ_0=0.01 cuando varía l_0, l_0=5 cuando varía Δ_0.

Como conclusiones de la caraterización efectuada pueden extraerse las siguientes:

1) El mejor parámetro para controlar la calidad de las soluciones y el tiempo de ejecución es el incremento inicial de temperatura Δ_0, aunque c_0 y l_0 presentan también un buen comportamiento, pudiéndose combinar variaciones de estos tres parámetros si fuera necesario.
2) El resto de los parámetros pueden utilizarse también para modificar la bondad de las soluciones, pero a costa de desperdiciar tiempo de ejecución.
3) Los valores propuestos teóricamente para c_0, c_f y b presentan un comportamiento satisfactorio, por lo que se mantendrán en las pruebas posteriores. Asimismo, se confirma que existe un valor óptimo para las cadenas de Markov, y coincide con el obtenido en 6.3.2.

6.4. MINIMIZACIÓN UNIFUNCIONAL CON RRMIN2. COMPARACIÓN CON OTROS PROCEDIMIENTOS

Utilizando la información obtenida en los dos apartados anteriores, ya se puede utilizar RRMIN2 para abordar la minimización de funciones completamente especificadas con una salida. A continuación se van a presentar los resultados obtenidos y se van a comparar con los presentados por otros autores.

Aunque a lo largo del capítulo se ha ido detallando la forma en que se han ido realizado las experiencias, resulta conveniente remarcar las condiciones en que se han efectuado las que se presentan en este apartado:

a) Los experimentos se han realizado utilizando la opción 3 del programa TESTRRMIN2 (véase 6.2), generando series de funciones aleatorias.

b) En cada caso, el número de funciones aleatorias que se ha generado ha sido tal que el valor medio obtenido para el número de términos productos de las mismas tenía un error máximo de ±0.1 con el 95% de confianza (obtenido mediante estimación por intervalos aplicando el teorema del límite central [PHE84]).

c) Se han obtenido medidas de lo siguiente:
   - Número medio de términos producto para cada serie de funciones (MP)
   - Desviación típica del número de términos producto (σ_{MP})
   - Error estimado con un 95% de confianza para el número de términos producto (E_{MP}).
   - TRR medio con respecto al número de minterms para cada serie de funciones (TRR)
   - Número medio de literales para cada serie de funciones (ML).
   - Desviación típica del número de literales (σ_{ML}).
   - Error estimado eon un 95% de confianza para el número de literales (E_{ML}).
   - LRR medio con respecto al número de literales en la expresión como suma de minterms para cada serie de funciones (LRR).

Bajo estas condiciones, se han obtenido los resultados de la Tabla 6.9, en la que se recogen experimentos para funciones con n=6,7,8,9,10 y 11 variables con un 25%, un 50% y un 75% de unos en sus expresiones como suma de minterms. Las claves para identificar las entradas en las tablas son las siguientes:

- n: Número de variables de las funciones
- %UNOS: Porcentaje de unos en la expresión como suma de minterms de cada función.
- n_pru: Número de funciones aleatorias generadas para su minimización.
- Δ_0: Incremento inicial de temperatura del proceso de Enfriamiento Simulado.
- TF: Tiempo de ejecución necesario para minimizar una función en una estación de trabajo SUN ULTRA1 a 143Mhz.

Los resultados presentados en esta tabla corresponden a una minimización profunda de las funciones generadas, exigiendo unos valores mínimos para el TRR y LRR muy reducidos, según se detalla en la Tabla 6.8.

En la Tabla 6.10 se realiza una comparación entre RRMIN2 y los procedimientos más importantes descritos en la literatura. Dicha tabla incluye dos columnas de resultados para cada procedimiento, una con el número medio de términos productos (MP), y otra con el tiempo de ejecución necesario para obtener ese resultado (TF se ha obtenido en un ordenador tipo PC con procesador 486 a 33Mhz para todos los procedimientos, excepto en el caso de RRMIN2, que se ha utilizado una estación SUN ULTRA1 a 143Mhz. El símbolo * significa

que el tiempo de ejecución ha resultado ser inferior a 0.1s).

%UNOS	TRR	LRR
25%	0.550	0.420
50%	0.360	0.240
75%	0.210	0.140

Tabla 6.8. TRR y LRR mínimos exigidos en función del número de unos de la función para obtener los resultados de la Tabla 6.9

n	%UNOS	n_pru	Δ_0	MP	σ_{MP}	E_{MP}	TRR	ML	σ_{ML}	E_{ML}	LRR	TF(s)
6	25%	1500	0.01	8.74	0.87	0.04	0.546	37.7	4.6	0.2	0.393	4.19
6	50%	1500	0.01	11.32	1.02	0.05	0.354	43.4	4.8	0.2	0.226	4.54
6	75%	1500	0.01	9.87	1.02	0.05	0.206	38.3	4.9	0.3	0.133	4.10
7	25%	1200	0.01	16.93	1.50	0.08	0.529	88.3	4.5	0.3	0.394	9.02
7	50%	1200	0.01	21.99	1.86	0.11	0.344	99.3	5.0	0.3	0.222	10.50
7	75%	1200	0.01	18.02	1.49	0.08	0.188	89.0	5.6	0.3	0.132	9.14
8	25%	600	0.005	32.63	2.10	0.17	0.510	207.3	15.3	1.2	0.405	23.8
8	50%	600	0.01	44.6	2.7	0.2	0.348	234.3	18.1	1.4	0.229	27.4
8	75%	750	0.01	34.59	2.32	0.17	0.180	209.8	14.4	1.0	0.137	23.3
9	25%	400	0.005	64.7	3.2	0.3	0.505	465	20	2	0.404	220
9	50%	400	0.007	89.6	3.9	0.4	0.350	531	31	3	0.239	163
9	75%	400	0.007	67.0	2.8	0.3	0.174	479	28	3	0.139	147
10	25%	200	0.0033	126.1	4.3	0.6	0.493	1027	38	5	0.401	1698
10	50%	200	0.004	179.7	5.5	0.8	0.351	1176	42	6	0.226	1551
10	75%	200	0.0035	128.6	4.4	0.6	0.167	1040	40	6	0.135	1459
11	25%	50	0.0023	248.3	5.5	1.5	0.485	2260	102	30	0.401	13640
11	50%	150	0.0025	362.2	9.6	1.5	0.354	2650	122	20	0.235	11830
11	75%	50	0.003	252.7	4.5	1.3	0.165	2300	94	30	0.136	10860

Tabla 6.9. Resultados obtenidos con RRMIN2 para la minimización de funciones completamente especificadas de una salida. En todos los casos se ha utilizado c_0=20, c_f=0.33 y b=1.0001

		ACEM		EXMIN2		Even-Kohavi-Paz		Heliwell-Perkowski		RRMIN2 (Nivel A)		RRMIN2 (Nivel B)			
n	%UNOS	MP	TF(s)	MP	TF(s)	MP	TF(s)	MP	TF(s)	MP	TF(s)	MP	TF(s)	%Mejora Nivel A	%Mejora Nivel B
6	25%	10.4	*	9.7	*	9.4	*	12.4	*	8.74	4.19	10.16	*	20%	3%
6	50%	13.2	*	12.9	*	13.9	*	15.6	*	11.32	4.54	13.92	*	23%	0%
6	75%	11.6	*	10.6	*	14.4	*	14.8	*	9.87	4.10	11.20	*	30%	15%
7	25%	19.9	*	17.4	0.1	17.5	*	26.0	*	16.93	9.02	20.02	*	20%	1%
7	50%	25.1	*	22.9	0.3	25.5	0.1	26.9	0.1	21.99	10.50	25.00	1.11	14%	0%
7	75%	21.4	*	22.0	0.3	29.4	0.2	28.8	0.1	18.02	9.14	21.12	*	41%	20%
8	25%	38.8	*	33.8	0.4	33.3	0.4	51.8	0.2	32.63	55.1	39.10	0.52	21%	1%
8	50%	51.8	*	46.3	1.3	50.7	0.9	57.7	0.5	44.6	27.4	50.7	3.13	16%	2%
8	75%	40.2	*	42.7	1.7	58.2	1.3	56.8	0.4	34.59	23.3	41.1	0.21	43%	20%
9	25%	78.2	0.1	65.0	2.3	65.5	1.7	109.7	1.1	64.7	220	78.4	1.42	23%	1%
9	50%	103.7	0.2	90.2	7.7	98.7	6.2	115.8	2.9	89.6	163	101.9	26.3	14%	0%
9	75%	76.7	0.1	92.8	9.2	113.3	9.6	113.8	3.2	67.0	147	94.1	0.41	48%	5%
10	25%	153.5	0.4	126.6	12.5	126.1	13.2	217.6	15.8	126.1	1698	154.9	6.0	24%	1%
10	50%	205.4	0.6	182.2	37.6	192.4	47.8	218.9	17.8	179.7	1551	199.2	270	11%	0%
10	75%	156.3	0.4	196.5	48.8	380.6	70.5	215.5	19.2	128.6	1459	187.1	2.30	53%	5%
11	25%	308.2	1.4	249.4	63.8	248.3	98.3	416.9	111.5	248.3	13640	300.3	33.3	23%	2%
11	50%	406.8	2.2	365.2	198.0	380.6	367.0	424.8	104.0	362.2	11830	387.1	1982	9%	2%
11	75%	308.7	1.4	400.6	259.1	435.4	531.1	426.0	105.9	252.7	10860	388.3	11.51	55%	1%

Tabla 6.10. Comparación de RRMIN2 con otros procedimientos.

Para RRMIN2 se han obtenido dos series de resultados, una que se ha denominado Nivel A, y que corresponde a una minimización en la que se aprovechan a fondo las posibilidades de RRMIN2 (y que coinciden con los resultados presentados en la Tabla 6.9), y otra (Nivel B) en la que se han tratado de igualar los resulados con los de los otros procedimientos para realizar comparaciones de tiempos de ejecución. Las columnas *%Mejora Nivel A* y *%Mejora Nivel B* presentan la mejora media en tanto por ciento obtenida en el MP por RRMIN2 con respecto a los otros procedimientos de la tabla. En el caso del Nivel A, se ha mejorado en todos los casos los resultados proporcionados por los demás autores, y además la mejora media varía entre un 9% y un 55%. Los tiempos de ejecución obtenidos en este nivel son bastante elevados, sobre todo si se tiene en cuenta que las pruebas de los otros procedimientos se han realizado en un PC 486 a 33Mhz [ABO95], mientras que las de RRMIN2 se han efectuado en una estación SUN ULTRA1 a 143 Mhz (para comparar, deberían multiplicarse aproximadamente por 5 los tiempos obtenidos para RRMIN2). No obstante, tampoco resulta excesivo invertir 3 horas para minimizar de forma definitiva una función de 11 variables. En caso de que se deseen hacer minimizaciones de peor calidad pero de forma más rápida, puede recurrirse a optimizaciones como las del Nivel B. En ese caso, la mejora media varía entre un 0% y un 20% (es decir, se obtienen unos resultados similares a los de los otros métodos), y los tiempos de ejecución son del mismo orden, e incluso inferiores, que en los otros procedimientos. En cualquier caso, si se dispone de un sistema multiprocesador o de varias estaciones de trabajo conectadas en red, puede utilizarse la versión paralela PRRMIN, que permite rebajar considerablemente el tiempo necesario para obtener los resultados (véase el Apartado 6.6).

Los procedimientos que se han utilizado para comparar en la Tabla 6.10 son los más representativos, con la excepción de EXORCISM-MV2 (Apartado 3.3.5, [SON96]). El hecho de que éste procedimiento no aparezca en dicha tabla se debe a que no se dispone de datos estadísticos del mismo, sus autores sólo proporcionan los resultados de aplicar el procedimiento a una serie de funciones "benchmark". La comparación de procedimientos de minimización AND-EXOR mediante la optimización de unas pocas funciones no parece ser el método más adecuado habida cuenta del elevadísimo número de funciones de conmutación existentes (sólamente en el caso de 6 variables se tienen 2^{64} funciones distintas), pudiéndose dar el caso de que un procedimiento sea excelente para minimizar esas pocas funciones y no dé tan buenos resultados para el resto de las funciones. Por tanto, en esta memoria se han realizado pruebas y comparaciones estadísticamente significativas con el objeto de tener una referencia válida del comportamiento de los procedimientos al enfrentarse a una función cualquiera que el usuario desee minimizar. No obstante, por completar el estudio, se presenta también una tabla comparativa de resultados al minimizar una serie de funciones "benchmark" con EXMIN2, EXORCISM-MV2 y RRMIN2 en el Apartado 6.5 (muchas de estas funciones tienen varias salidas, y se ha preferido mostrar estos resultados en la parte de minimización multifuncional).

6.4. MINIMIZACIÓN DE FUNCIONES INCOMPLETAMENTE ESPECIFICADAS

Una vez analizado el comportamiento de RRMIN2 para funciones completamente especificadas de una salida, en este apartado se van a obtener resultados para la optimización de funciones con indiferencias. Para efectuar la asignación de indiferencias, en el Apartado 4.9 se desarrolló un procedimiento denominado ASIGMIN. Para adaptar este método a cada tipo de función, se dispone de un parámetro, par_profundidad (4.156), cuyo ajuste se efectúa en 6.4.1. Posteriormente, en 6.4.2 se presentan los resultados obtenidos al minimizar funciones incompletamente especificadas con RRMIN2.

6.4.1. Ajuste de par_profundidad

La optimización de funciones incompletamente especificadas requiere la asignación adecuada de las indiferencias para conseguir un buen aprovechamiento de las mismas. En RRMIN2 se realiza esta asignación mediante el procedimiento ASIGMIN, desarrollado en 4.9.1, y que busca efectuar la asignación óptima para obtener la forma MPRM (expresión inicial AND-EXOR utilizada en RRMIN2, 4.5), con menor número de términos producto. El parámetro de profundidad, *par_profundidad*, definido en 4.9.1, permite ajustar el proceso de asignación de indiferencias para obtener un mejor rendimiento en función de las características de la función (número de variables, número de unos de la función y número de indiferencias de la misma).

El estudio que se ha realizado para obtener el valor óptimo de *par_profundidad* (PPO, Parámetro de Profundidad Óptimo) consiste en minimizar series de funciones con distinto número de variables, unos e indiferencias probando todos los valores posibles de par_profundidad (recuérdese que $1 \leq par\_profundidad \leq 2^n$). Se han realizado pruebas para funciones con n=6, 8 y 10 variables y porcentajes de unos del 25%, 37.5%, 50% y 75%. Los porcentajes de indiferencias que se han utilizado en cada caso dependen del número de unos (piénsese por ejemplo que en una función con un 75% de unos no tiene sentido utilizar un porcentaje de indiferencias igual o superior al 25% puesto que se obtendría la función 1). Concretamente, se han utilizado los siguientes:

- En funciones con un 25% de unos: 6.25%, 12.5%, 25% , 37.5%, 50% y 62.5% de indiferencias.
- En funciones con un 37.5% de unos: 6.25%, 12.5%, 25%, 37.5% y 50%
- En funciones con un 50% de unos: 6.25%, 12.5%, 25% y 37.5% de indiferencias.
- En funciones con un 75% de unos: 6.25% y 12.5% de indiferencias.

En la Figura 6.31 puede verse el ajuste de *par_profundidad* efectuado para el caso de n=6 variables, n_unos=32 (50%) y n_indif=16 (12.5%). La gráfica muestra cómo el número medio de términos producto disminuye al aumentar *par_profunidad*, concluyéndose que en esta ocasión PPO=32. Para otros valores, la dependencia que se obtiene es distinta, y así, por

ejemplo, en la Figura 6.32, correspondiente a n=6 variables, n_unos=16 (25%) y n_indif=16 (12.5%) el número de productos presenta un mínimo en *par_profundidad=4*, resultando PPO=4. El número de funciones que se han probado (*n_pru*) para obtener estas gráficas ha sido tal que pudiera rechazarse con un 95% de confianza la hipótesis de que dos valores para la media de productos MP fuesen iguales. En la Tabla 6.11 se muestran los resultados obtenidos para el PPO en funciones de n=6 variables, y en las tablas 6.12 y 6.12, los PPOs para funciones de 8 y 10 variables, respectivamente.

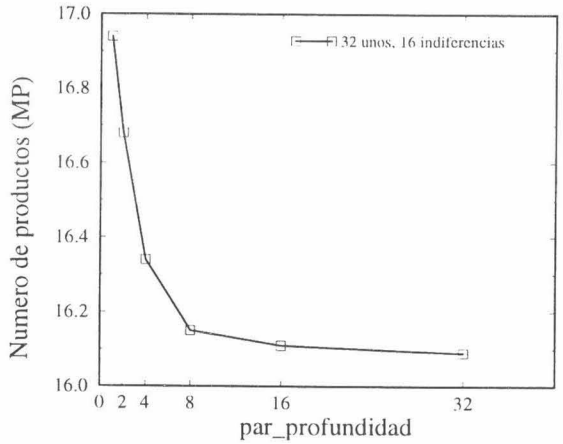


Fig. 6.31. Obtención del PPO para n=6, n_unos=32 y n_indif=16

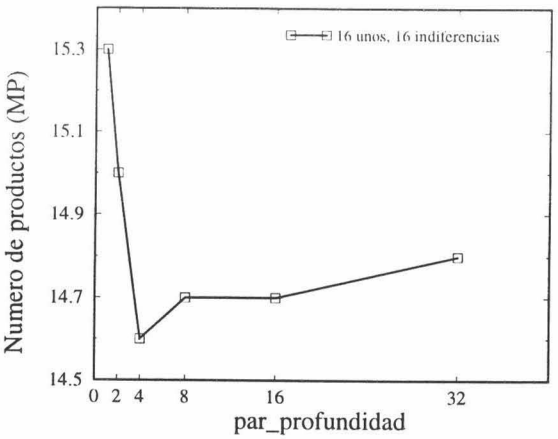


Fig. 6.32. Obtención del PPO para n=6, n_unos=16, n_indif=16

Al igual que se ha hecho con el resto de parámetros de RRMIN2, sería deseable obtener una expresión matemática lo más sencilla posible que proporcione el PPO adecuado para cada tipo de función. Ésta expresión debería de calcular el PPO en función del número de variables (n), el número de unos de la función (n_unos) y el número de indiferencias

(n_indif). Analizando los resultados que se presentan en las Tabla 6.11, no parece existir una relación sencilla que permita utilizar líneas de regresión o algún procedimiento similar para efectuar el ajuste de los datos (y además, las Figuras 6.31 y 6.32 ponen de manifiesto comportamientos muy diferentes al variar el número de unos).

n=6			n=8			n=10		
n_unos	n_indif	PPO	n_unos	n_indif	PPO	n_unos	n_indif	PPO
16	4	2	64	16	4	256	64	8
16	8	8	64	32	16	256	128	32
16	16	16	64	64	32	256	256	64
16	24	8	64	96	16	256	384	32
16	32	1	64	128	1	256	512	1
16	40	32	64	160	128	256	640	512
24	4	2	96	16	4	384	64	8
24	8	2	96	32	4	384	128	8
24	16	1	96	64	1	384	256	1
24	24	8	96	96	16	384	384	32
24	32	32	96	128	128	384	512	512
32	4	2	128	16	4	512	64	8
32	8	2	128	32	4	512	128	8
32	16	32	128	64	128	512	256	512
32	24	32	238	96	128	512	384	512
48	4	4	192	16	8	768	64	16
48	8	32	192	32	128	768	128	512

Tabla 6.11. Valores óptimos de par_profundidad (PPO).

No obstante, pueden extraerse las siguientes conclusiones de esta tabla:

a) En general se observa que un aumento del número de indiferencias implica un valor más alto para el PPO, salvo algunas excepciones notables, como es el caso n=6, n_unos=16, n_indif=32, en el que se tiene PPO=1. Si se analiza en detalle ese caso, puede observarse que el número de unos y el de ceros es idéntico, al igual que ocurre para n=6, n_unos=24,

n_indif=16, o para n=8, n_unos=64, n_indif=128, etc. Es decir, en todos aquellos casos en que el número de unos es igual que el de ceros, se obtiene PPO=1.

b) En las filas de la tabla, al pasar de n=6 a n=8 o de n=8 a n=10 el PPO aumenta en un factor de 2 salvo en aquellos casos en que $PPO=2^{n-1}$.

El principal problema para realizar el ajuste de los datos, lo constituyen las excepciones mencionadas en a) y b). La excepción a que se hace referencia en a) es debida a la forma de realizar la asignación de indiferencias de ASIGMIN, que procura conseguir que los unos (y los ceros) de la función estén situados en posiciones simétricas (véase el Apartado 4.9.1.1) y se produzca así el mayor número de cancelaciones posible al calcular la forma MPRM mínima. Si la función tiene el mismo número de unos que de ceros se está favoreciendo esa búsqueda de simetría de la función y resulta ventajoso realizar el mayor número de pasadas posibles (Algoritmo 4.21) para realizar la asignación. Por tanto, el PPO debe incluir en su expresión alguna función de n_unos y n_ceros (n_ceros=2^n-n_unos-n_indif, con lo que en realidad esta función sería sólo función de n_unos) que dé cuenta de la simetría de la función. Una posibilidad podría ser:

$$f_{sim}(n\_unos)=\frac{1}{2}\cdot\left(\frac{n\_unos}{n\_ceros}+\frac{n\_ceros}{n\_unos}\right)-1=\frac{1}{2}\cdot\left(\frac{n\_unos}{2^n-n\_unos-n\_indif}+\frac{2^n-n\_unos-n\_indif}{n\_unos}\right)-1 \quad (6.12)$$

Esta función mide la desviación que se tiene con respecto a la situación n_unos=n_ceros, en la que $f_{sim}=0$. Si se analizan ahora los casos correspondientes a un mismo valor de la función de simetría y del número de indiferencias n_indif, para estudiar la dependencia del PPO con n_unos, se obtienen resultados como los de la Tabla 6.12, es decir, no existe dependencia del PPO con el número de unos (salvo la existente a través de la función f_{sim}, por supuesto). Por tanto, sólo queda determinar la dependencia de PPO con el número de indiferencias y con el número de variables. Para determinar estas dependencias, se va a trabajar con el $\log_2$(PPO) puesto que éste varía en potencias de 2, y resulta más cómodo ajustar variaciones lineales que exponenciales.

n=6			
n_unos	n_indif	f_{sim}(n_unos)	PPO
16	24	0.083	8
24	24	0.083	8
24	8	0.042	2
32	8	0.042	2

Tabla 6.12. Resultados para PPO en funciones con igual valor de f_{sim} (n=6 variables)

En primer lugar se va a estudiar la dependencia de $\log_2$(PPO) con el número de indiferencias, n_indif, manteniendo fijo el número de variables (tomando n=6), y quitando los puntos incluidos en la excepción a que se hace referencia en b) (se procederá al estudio de estos puntos más adelante). Se va a empezar probando un análisis de regresión lineal de $\log_2$(PPO) frente a f_{sim}(n_unos)·$\log_2$(n_indif). El coeficiente de correlación que se obtiene no es bueno (r=0.36), y por tanto, debe buscarse otro tipo de relación. Si se trata de ajustar linealmente $\log_2$(PPO) frente a f_{sim}(n_unos)·n_indif, se obtiene un coeficiente de correlación mucho mejor (r=0.80), pero insuficiente. Probando un ajuste de $\log_2$(PPO) frente a f_{sim}(n_unos)·n_indif^2 tampoco se obtienen resultados satisfactorios, por lo que se ha optado por intentar una dependencia intermedia entre las dos anteriores, concretamente se ha representado $\log_2$(PPO) frente a f_{sim}(n_unos)·n_indif·$\log_2$(n_indif). En la Figura 6.33 puede comprobarse que en ese caso se obtiene una correlación lineal buena (r=0.96), con los siguientes valores para la pendiente y la ordenada en el origen:

$$\begin{aligned} a &= 0.234 \pm 0.026 \\ b &= 0.34 \pm 0.21 \end{aligned} \qquad (6.13)$$

En resumen, para n=6 puede extraerse la siguiente expresión para el PPO:

$$\log_2(PPO) = 0.0234 \cdot f_{sim}(n\_unos) \cdot n\_indif \cdot \log_2(n\_indif) + 0.34 \qquad (6.14)$$

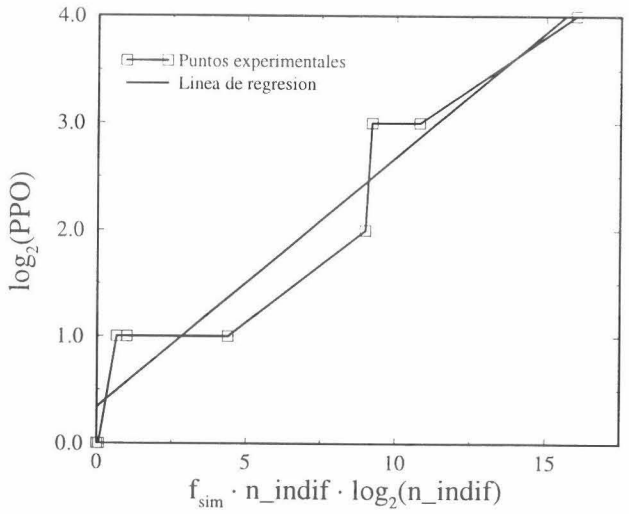


Fig. 6.33. Ajuste por mínimos cuadrados del $\log_2$(PPO) para n=6 variables

Si ahora se quiere incluir en esta expresión la dependencia con n, basta con tener en cuenta lo mencionado en a): en la misma fila (lo que implica tener la misma relación n_indif/2^n) el paso de n=6 a n=8 ó de n=8 a n=10 implica multiplicar por 2 el PPO (o en

términos de logaritmos en base 2, sumar una unidad). Por tanto, se obtendría la siguiente expresión para tener en cuenta el número de variables n:

$$\log_2(PPO)=90\cdot f_{sim}(n\_unos)\cdot\left(\frac{n\_indif}{2^n}\right)\cdot\left(\frac{\log_2(n\_indif)}{n}\right)+\frac{1}{2}(n-6)+0.34 \qquad (6.15)$$

En esta expresión, sin embargo, no se incluye la excepción referida en a). Ésta excepción se produce cuando la función de simetría f_{sim}(n_unos)=0, de manera que si se multiplica el término 1/2·(n-6) por un cociente en que el numerador sea f_{sim}, y el denominador sea f_{sim}+j donde j es un número pequeño, se resolvería la cuestión:

$$\log_2(PPO)=90\cdot f_{sim}(n\_unos)\cdot\left(\frac{n\_indif}{2^n}\right)\cdot\left(\frac{\log_2(n\_indif)}{n}\right)+\frac{1}{2}(n-6)\cdot\frac{f_{sim}(n\_unos)}{f_{sim}(n\_unos)+0.0001}+0.34 \qquad (6.16)$$

Finalmente, para tener en cuenta la excepción de b), se puede utilizar esta misma expresión, pero teniendo en cuenta que el PPO no puede pasar del valor 2^{n-1}, y por tanto, la expresión final que se obtiene es:

$$A=90\cdot f_{sim}(n\_unos)\cdot\left(\frac{n\_indif}{2^n}\right)\cdot\left(\frac{\log_2(n\_indif)}{n}\right)+\frac{1}{2}(n-6)\cdot\frac{f_{sim}(n\_unos)}{f_{sim}(n\_unos)+0.0001}+0.34$$

$$\log_2(PPO)=\begin{cases}A & si\ A<n-1\\ n-1 & si\ A\geq n-1\end{cases} \qquad (6.17)$$

Esta expresión proporciona el PPO correcto en todos los casos contemplados en la Tabla 6.11.

6.4.2. Resultados de la minimización de funciones incompletamente especificadas

Para comprobar la efectividad de la asignación de indiferencias en RRMIN2, se han realizado minimizaciones de series aleatorias de funciones con distinto número de variables, porcentaje de unos y porcentaje de indiferencias. Para ello se ha utilizado el procedimiento ASIGMIN, desarrollado en 4.9, utilizando como valores del PPO los obtenidos en 6.4.1. La Tabla 6.13 muestra los resultados obtenidos, comparando con los dos procedimientos básicos de asignación de indiferencias, igualarlas todas a 0 (columna A0) e igualarlas todas a 1 (columna A1). Los resultados se han obtenido realizando un número de minimizaciones igual al que aparece en la Tabla 6.9 para cada caso. En consecuencia, las desviaciones típicas y los errores con un 95% de confianza son similares a los de la citada tabla, y se han omitido en esta ocasión. Se observa que en todos los casos RRMIN2 consigue mejorar, o en el peor de

n=6					n=8					n=10				
n_unos	n_indif	MP	A0	A1	n_unos	n_indif	MP	A0	A1	n_unos	n_indif	MP	A0	A1
16	16	8.73	8.74	11.32	64	64	33.36	33.60	44.56	256	256	134.7	135.6	193.3
24	4	10.37	10.41	11.06	96	16	41.44	41.74	43.84	384	64	175.8	176.5	182.7
24	8	10.22	10.41	11.32	96	32	41.14	41.74	44.56	384	128	175.2	176.5	193.3
24	12	10.17	10.41	11.58	96	48	41.10	41.74	44.60	384	192	174.9	176.5	196.7
24	16	9.90	10.41	11.45	96	64	40.42	41.74	43.10	384	256	172.1	176.5	178.6
32	4	10.83	11.32	11.58	128	16	43.12	44.56	44.60	512	64	192.3	193.3	196.7
32	8	10.77	11.32	11.45	128	32	42.80	44.56	43.10	512	128	180.1	193.3	180.2
32	12	10.57	11.32	10.84	128	48	39.20	44.56	39.20	512	192	154.0	193.3	156.7
32	16	9.72	11.32	9.87	128	64	34.59	44.56	34.59	512	256	134.3	193.3	134.3
48	8	6.93	9.87	6.93	192	32	33.60	34.59	33.62	768	128	83.9	134.3	85.4

Tabla 6.13. Resultados de la minimización de funciones incompletamente especificadas

los casos igualar, estas asignaciones básicas. En algunos casos, la mejora alcanza hasta el 5%, mientras que el incremento de tiempo de ejecución por realizar la asignación de indiferencias no supone en ninguna situación un incremento superior al 1%. Por tanto, aunque la mejora obtenida no es demasiado elevada, resulta ventajoso utilizar este método de asignación de indiferencias puesto que no supone prácticamente ningún costo en tiempo de ejecución.

Nótese por otra parte que la ventaja obtenida al utilizar ASIGMIN es mayor cuando se tienen pocas indiferencias y un número intermedio de unos, debido a que esas situaciones son las que requieren de una mayor habilidad para realizar la asignación. En los demás casos, el proceso de minimización puede enmascarar con mayor facilidad los defectos de realizar una asignación de todas las indiferencias a 0 ó a 1 (según sea lo adecuado en cada caso).

6.5. MINIMIZACION DE FUNCIONES CON VARIAS SALIDAS

Para abordar la minimización de funciones con varias salidas, RRMIN2 dispone de tres opciones, según se describió en 4.10. Estas opciones consisten en minimizar individualmente cada una de ellas, minimizarlas conjuntamente, o bien, como opción por defecto, realizar una minimización previa de las salidas individualmente y posteriomente optimizar conjuntamente las salidas del resultado obtenido. Esta última opción implica que la minimización conjunta se realiza a partir de una multifunción que se encuentra ya previamente optimizada, es decir, en un estado de baja energía si se habla en términos de Enfriamiento Simulado. Por tanto, puede ser conveniente iniciar la optimización conjunta a una temperatura más baja de lo habitual para evitar que se produzca un número excesivo de transiciones a estado con función de coste más elevada. Para ello se introdujo en 4.10.2 el parámetro t_div, que simplemente divide a los parámetros c_0, Δ_0 y c_f para realizar un proceso de Enfriamiento Simulado con el mismo número de iteraciones (Apéndice B), pero en un rango de temperaturas más bajas. El Apartado 6.5.1 muestra el ajuste realizado para este parámetro, y en 6.5.2 se analizan los resultados presentados por RRMIN2 para la minimización de funciones con varias salidas. En ese mismo apartado se incluye una tabla comparativa de RRMIN2 con EXMIN2 y EXORCISM-MV2 para la optimización de una serie de funciones "benchmark".

6.5.1. Ajuste del parámetro t_div

Para realizar el ajuste del parámetro t_div se ha realizado la minimización de series de funciones de n=6, 8 y 10 variables, con 2, 4 y 6 salidas. La Tabla 6.14 y la Figura 6.34 muestran los resultados obtenidos para funciones de 6 variables y 4 salidas, concluyendo que el mejor t_div en ese caso es t_div=16. Realizando un estudio análogo en las demás situaciones, se obtendría la Tabla 6.15, de donde pueden extraerse las siguientes conclusiones:

a) El parámetro t_div depende exclusivamente del número de variables, n.
b) Al aumentar el número de variables, t_div disminuye, alcanzando su valor mínimo (t_div=1) al llegar a n=10 variables.

t_div	MP	σ_{MP}	E_{MP}
1	43.62	2.09	0.11
2	43.25	1.94	0.10
4	43.11	2.04	0.10
8	42.14	1.93	0.10
16	41.11	2.15	0.11
20	41.65	2.26	0.11

Tabla 6.14. Obtención del valor óptimo de t_div para funciones de n=6 variables y n_func=4 salidas
Parámetros utilizados: n_unos=32, n_indif=0

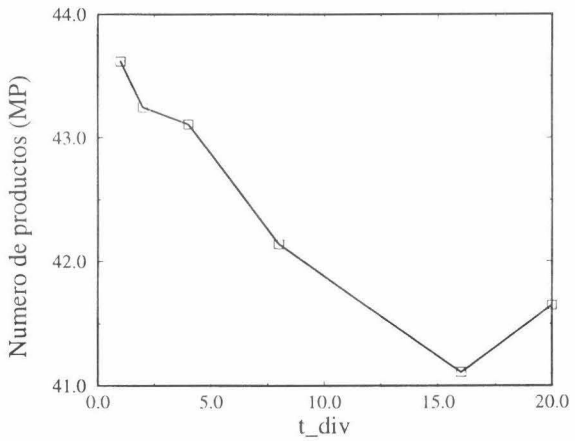


Fig. 6.34. Representación del número de términos producto MP frente a t_div para funciones de n=6 variables y n_func=4 salidas.

Si se efectúa un ajuste lineal de los valores presentados en la Tabla 6.15, se obtiene lo siguiente para la pendiente y la ordenada en el origen:

$$\begin{aligned} a &= -4 \\ b &= 40 \end{aligned} \tag{6.18}$$

Por tanto, t_div puede expresarse en términos de n de la siguiente forma:

$$t\_div = 4 \cdot (10 - n) \tag{6.19}$$

n	n_func	t_div óptimo
6	2	16
6	4	16
6	6	16
8	2	8
8	4	8
8	6	8
10	2	1
10	4	1
10	6	1

Tabla 6.15. t_div óptimo para distintos valores del número de variables n y del número de salidas n_func. Parámetros utilizados: n_unos=32, 128 y 512 (para n=6, 8 y 10 variables, respectivamente), n_indif=0

6.5.2. Resultados obtenidos con RRMIN2 para minimización multifuncional

Con los valores de t_div proporcionados en 6.5.1, se han realizado pruebas para funciones de n=6, 8 y 10 variables con n_func=2, 4 y 6 salidas, obteniéndose los resultados presentados en la Tabla 6.16. La columna MPF corresponde al número de términos producto necesarios para sintetizar una de las salidas de la función. De esta forma, es posible comparar con el coste que se tendría en caso de minimizar cada salida por separado (filas correspondientes a n_func=1). También se incluye otra columna, TPF, que corresponde al tiempo de ejecución que se consume por cada salida a sintetizar, permitiendo así la comparación con el caso de minimizar las salidas por separado. Puede observarse cómo un aumento en el número de salidas permite rebajar el valor de MPF, gracias a la compartición de términos producto por parte de varias salidas. Ésta disminución varía entre un 5% y un 14% para funciones con 6 salidas, invirtiendo aproximadamente el doble de tiempo de ejecución que si se efectúa la síntesis por separado.

Finalmente, y para incluir una comparación con el procedimiento EXORCISM-MV-2, la Tabla 6.14 presenta los resultados obtenidos por éste método, EXMIN2 y el procedimiento desarrollado en este trabajo, RRMIN2, al minimizar una serie de funciones benchmark [SON96]. En dicha tabla, aparece comparado el número de términos producto MP, y el número total de entradas a puertas, ETP, observándose cómo RRMIN2 iguala o supera a los otros dos procedimientos en todos los casos. Para las funciones xor5, 5xp1, rd53, rd73 y 9sym se obtiene el mismo número de términos productos que EXORCISM-MV2, pero se mejora notablemente el número de entradas a puertas (por ejemplo, para la función rd73 se consigue una reducción del 16% con respecto a EXORCISM-MV-2 y de un 25% con respecto a

EXMIN2), y en el caso de rd84 se consigue mejorar en una unidad el valor de MP obtenido con EXORCISM-MV2 y en 3 el proporcionado por EXMIN2. Los tiempos de ejecución que aparecen están tomados en una estación SPARC Station 1+ para EXMIN2, una SPARC Station 1 para EXORCISM-MV-2 y una SPARC ULTRA 1 en el caso de RRMIN2.

n	n_func	MP	MPF	TF	TPF
6	1	11.32	11.32	4.54	4.54
6	2	21.49	10.75	17.8	8.90
6	4	41.11	10.28	36.3	9.07
6	6	59.49	9.92	54.6	9.10
8	1	44.56	44.56	27.4	27.4
8	2	84.8	42.41	102.6	51.1
8	4	169.5	42.37	205	51.3
8	6	252.9	42.14	320	53.3
10	1	193.3	193.3	445	445
10	2	376.2	188.1	1705	853
10	4	746	186.6	3500	875
10	6	1107	184.5	5460	910

Tabla 6.16. Resultados obtenidos para minimización multifuncional.
Parámetros utilizados: n_unos=32, 128 y 512 para n=6, 8 y 10 variables, respectivamente

			EXMIN2			EXORCISM-MV-2			RRMIN2		
benchmark	n	n_func	MP	ETP	TF	MP	ETP	TF	MP	ETP	TF
9sym	9	1	53	433	25	51	425	39	51	425	362
5xp1	5	10	34	186	13	33	178	13.6	33	170	205
xor5	5	1	-	-	-	5	10	0.2	5	10	0.02
rd53	5	3	15	60	2	14	57	1.3	14	57	236
rd73	7	4	42	221	20	38	191	24.6	38	177	350
rd84	8	4	59	330	45	57	317	168	56	300	420

Tabla 6.14. Comparación de los procedimientos EXMIN2, EXORCISM-MV2 y RRMIN2 al minimizar algunas funciones benchmark

Los apartados que siguen van a presentar los resultados de ganancia en velocidad y/o mejora de soluciones que se consiguen al aprovechar el procesamiento paralelo utilizando el procedimiento PRRMIN.

6.6. RESULTADOS OBTENIDOS MEDIANTE PRRMIN

En el Capítulo 5 se abordó la paralelización del procedimiento RRMIN2 para aprovechar las posbilidades que ofrecen las arquitecturas multiprocesador o la conexión de varias estaciones de trabajo mediante una red local, obteniendo un procedimiento denominado PRRMIN. El programa utilizado para implementar y probar este procedimiento se ha denominado TESTPRRMIN, y está escrito en lenguaje C, incorporando las bibliotecas de funciones proporcionadas por el software de dominio público PVM (5.1, [GEI94]) para aprovechar las posibilidades del procesamiento paralelo mediante paso de mensajes. El programa TESTPRRMIN tiene las mismas entradas que TESTRRMIN2, y además incluye otras dos:

- Número de procesadores que se van a utilizar (n_proc).
- Número de iteraciones entre intercambio de soluciones (n_iter).

En 6.6.1 se efectúa un ajuste del parámetro n_iter definido en 5.4.2, y en 6.6.2 se presentan las mejoras de prestaciones obtenidos mediante este procedimiento. El sistema de multiprocesamiento que se ha utilizado para realizar las medidas experimentales ha consistido en 6 estaciones de trabajo SUN IPX conectadas mediante una red Ethernet de 10Mbps

6.6.1. Ajuste del parámetro n_iter

Según se describió en 5.4.2, el procedimiento PRRMIN consiste, básicamente, en hacer que varios procesadores ejecuten trozos del proceso de Enfriamiento Simulado siguiendo caminos distintos. Cada cierto número de iteraciones, los procesadores ponen en común las soluciones obtenidas, decidiéndose cuál es la mejor. Esta mejor solución se envía a todos los procesadores, que continúan el proceso de Enfrimiento Simulado, partiendo de esa solución común, pero siguiendo caminos diferentes. El parámetro n_iter controla el número de iteraciones del proceso de enfriamiento que transcurren entre dos puestas en común de las soluciones. Éste número de iteraciones debe ser lo bastante pequeño como para que se aproveche óptimamente la potencia del multiprocesamiento (es decir, se explore el mayor número de caminos posible), y lo suficientemente grande como para que los procesadores sean capaces de salir del mínimo local que corresponde a esa mejor solución que se ha puesto en común.

El ajuste que se ha llevado a cabo es de tipo empírico, y así, se han obtenido los resultados que reflejan las gráficas 6.35 a 6.37, donde se ha representado el número de términos producto frente a n_iter para funciones de n=6, 8 y 10 variables, respectivamente,

utilizando 4 procesadores.En esas figuras puede observarse cómo se produce un mínimo para el valor n_iter=200. Ésta misma situación se repite para n_proc=2 y 6 procesadores, y por tanto, el valor óptimo de n_iter es independiente del número de variables y procesadores. Asímismo puede comprobarse que tampoco depende del número de unos de la función, y es por tanto, un valor característico del procedimiento PRRMIN.

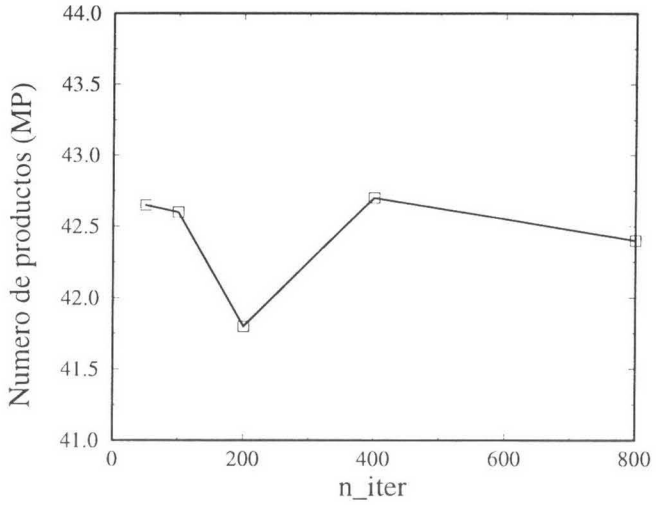


Fig. 6.35. Representación del número de productos frente a n_iter para n=6 variables y n_proc=4

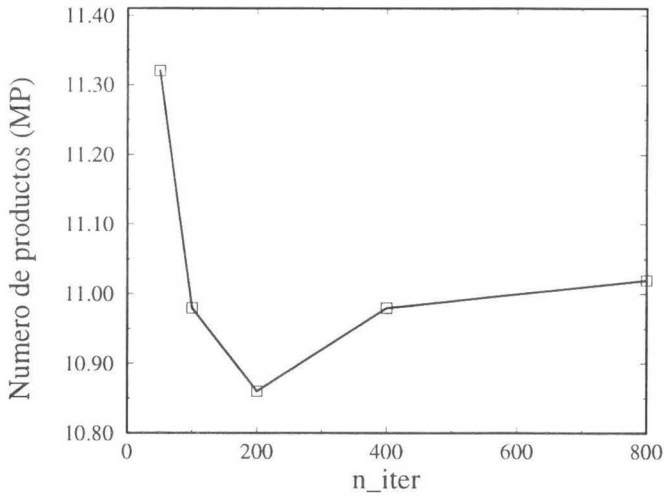


Fig. 6.36. Representación del número de productos frente a n_iter para n=8 variables y n_proc=4

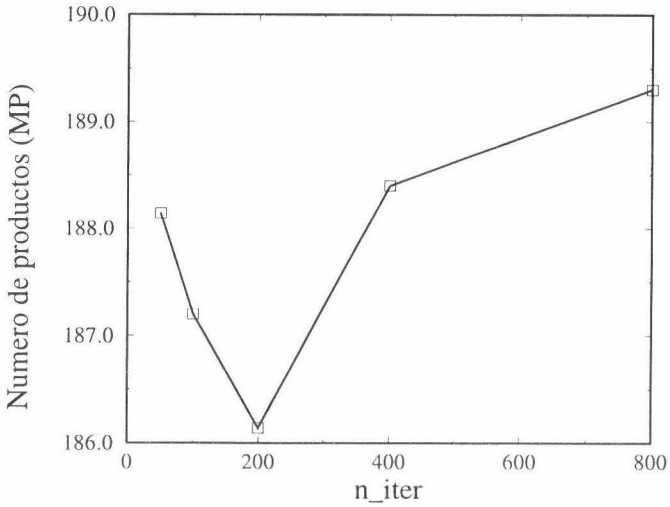


Fig. 6.37. Representación del número de productos frente a n_iter para n=10 variables y n_proc=4

6.6.2. Mejoras de prestaciones obtenidas con PRRMIN

El estudio experimental de PRRMIN se ha realizado bajo dos puntos de vista, por una parte se ha analizado qué ganancia de velocidad se obtiene al tratar de obtener soluciones con un TRR fijado, en función del número de procesadores, y por otra parte, se ha fijado el tiempo de ejecución y se ha estudiado qué mejora de la solución se obtiene al ir aumentando el número de elementos de procesamiento. En los apartados siguientes se muestran y analizan los resultados obtenidos para cada una de estas posibilidades.

6.6.2.1. Ganancia de velocidad proporcionada por PRRMIN

Para realizar un estudio de la ganancia de velocidad, se han fijado una serie de TRRs y se ha medido el tiempo de ejecución necesario para obtenerlos mediante un sistema con n_proc=1, 2, 4 y 6 procesadores. Por ejemplo, la Figura 6.38 muestra la situación de n=6 variables, exigiendo un TRR=0.505, obteniéndose una ganancia de 1.64 para n_proc=2, 3.44 para n_proc=4 y 4.54 para n_proc=6. Las figuras 6.39 y 6.40 muestran las ganancias obtenidas para TRR=0.416 con funciones de n=8 variables y exigiendo un TRR=0.413 para funciones de n=10 variables, respectivamente. Se observa que la ganancia en velocidad disminuye al aumentar el número de variables, debido fundamentalmente a que las necesidades de comunicación aumentan de forma exponencial con n (el número de valores a enviar por la red en cada intercambio de soluciones aumenta con 3^n). La Figura 6.41 presenta la eficacia obtenida por PRRMIN en función del número de variables y para distinto número de procesadores. Se observa que para 2 y 4 procesadores las eficacias son parecidas,

mientras que para n_proc=6 descienden ligeramente. En cualquier caso, siempre se mantienen por encima del 65%, y en ocasiones alcanzan valores cercanos al 85%, un resultado muy bueno para un algoritmo como RRMIN2 (el Enfriamiento Simulado es un procedimiento inherentemente secuencial, la minimización AND-EXOR no admite la partición del espacio de búsqueda, y además las comunicaciones en una red de estaciones de trabajo resultan bastante lentas).

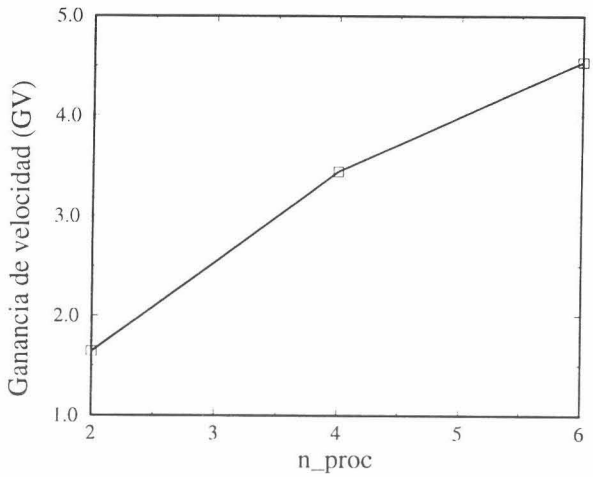


Fig. 6.38. Ganancia de velocidad para funciones de n=6 variables.

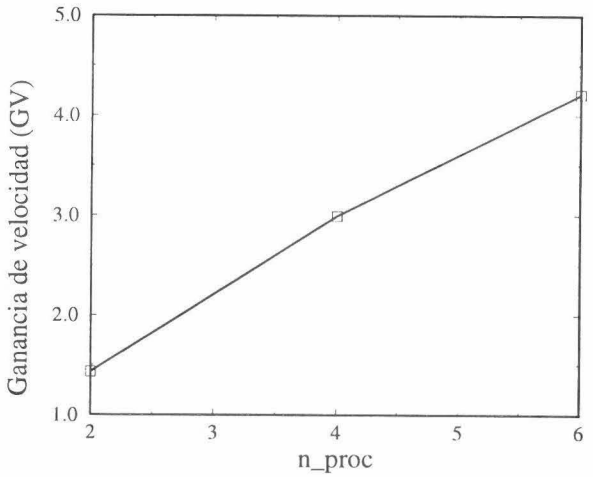


Fig. 6.39. Ganancia de velocidad para funciones de n=8 variables.

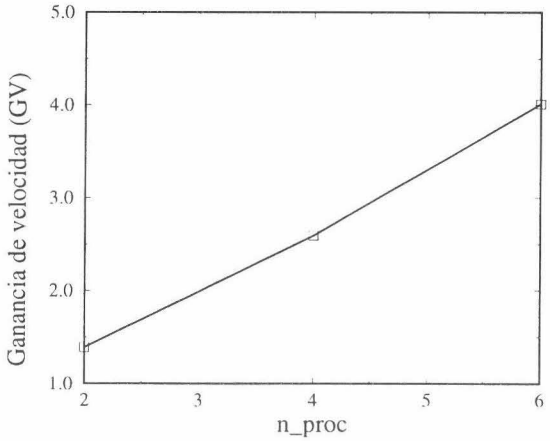


Fig. 6.40. Ganancia de velocidad para n=10 variables.

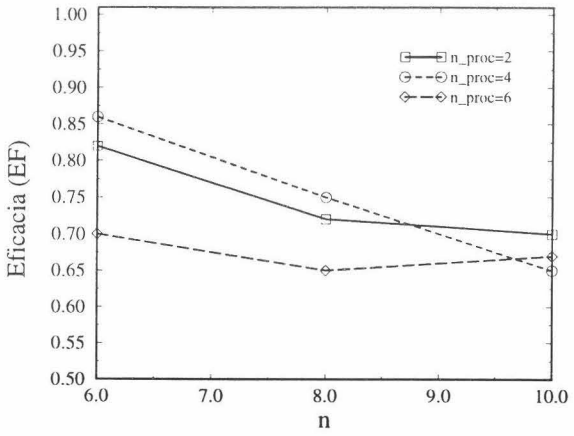


Fig. 6.41. Representación en función del número de variables y para distinto número de procesadores.

6.6.2.2. Mejora de las soluciones mediante PRRMIN

La otra posibilidad que se tiene de aprovechar las posibilidades de procesamiento paralelo ofrecidas por PRRMIN es mejorar la solución obtenida en un tiempo determinado mediante la utilización de varios procesadores. Así, se ha efectuado un estudio sobre cómo mejora la solución al aumentar el número de procesadores n_proc, manteniendo fijo el tiempo

de ejecución. De esta forma, se obtienen las figuras 6.42 a 6.44 en las que se representa el número de términos producto MP frente al número de procesadores para n=6, 8 y 10 variables, respectivamente. Las mejoras obtenidas llegan hasta un 10% para el caso de utilizar 6 procesadores.

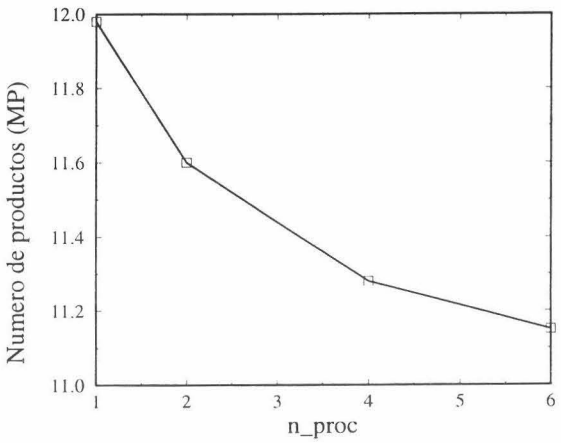


Fig. 6.42. Mejora de la solución para n=6 variables

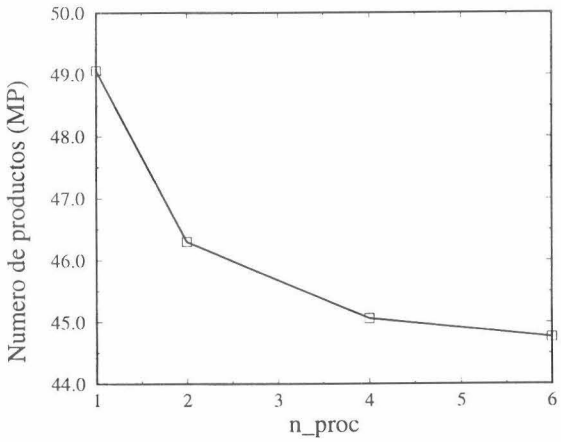


Fig. 6.42. Mejora de la solución para n=8 variables

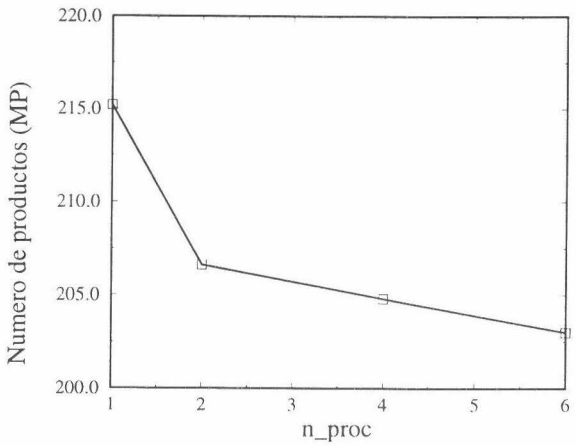


Fig. 6.44. Mejora de la solución para n=10 variables

6.7. CONCLUSIONES

En este capítulo se han presentado los resultados experimentales proporcionados por el procedimiento RRMIN2 y su versión paralela PRRMIN. Las conclusiones que pueden extraerse del análisis de estos resultados son las siguientes:

- El procedimiento RRMIN2 permite la minimización de funciones completa e incompletamente especificadas con una o varias salidas.

- Los resultados obtenidos por RRMIN2 para la minimización de funciones completemente especificadas de una salida mejoran sensiblemente los proporcionados por los demás procedimientos de minimización AND-EXOR aparecidos en la bibliografía.

- La minimización de funciones incompletamente especificadas con RRMIN2 basándose en la asignación de indiferencias proporcionada por el método ASIGMIN, desarrollado en esta memoria, resulta ventajosa con respecto a otras asignaciones más inmediatas, sin que suponga un incremento apreciable en el tiempo de ejecución

- En el caso de funciones con varias salidas, RRMIN2 permite aprovechar términos producto comunes, de forma que el número de términos producto medio necesario para sintetizar una salida disminuye al aumentar el número de éstas.

- El procedimiento RRMIN2 proporciona mejores resultados que EXMIN2 y EXORCISM-MV2 al minimizar las funciones benchmark 5xp1, 9sym,rd53, rd74y rd84.

- La versión paralela de RRMIN2, el procedimiento PRRMIN permite aprovechar los sistemas multiprocesador para:

a) Obtener soluciones de una determinada calidad utilizando menor tiempo de ejecución

b) Obtener mejores soluciones utilizando un mismo tiempo de ejecución.

CAPITULO 7

CONCLUSIONES Y PRINCIPALES APORTACIONES

En este último Capítulo se enuncian las conclusiones finales de la memoria, indicando las principales aportaciones efectuadas, y planteando futras líneas de investigación a seguir.

7.1. INTRODUCCION

La aparición de nuevos dispositivos lógicos programables como las FPGAs, ha hecho despertar un gran interés en la lógica AND-EXOR al desaparecer los inconvenientes de tipo tecnológico que planteaba la implementación de funciones con primitivas AND-EXOR. El atractivo de este otro tipo de diseño lógico reside en que los circuitos resultantes precisan de un menor número de puertas y el test de los mismos resulta más sencillo que en el diseño AND-OR. No obstante, la minimización AND-EXOR plantea un problema de difícil resolución y los procedimientos existentes para realizarla no presentan un buen comportamiento en todos los casos, especialmente en funciones con un porcentaje alto de unos en su tabla de verdad.

El objetivo de la presente memoria ha sido desarrollar un procedimiento de minimización AND-EXOR que tuviera un comportamiento satisfactorio para cualquier tipo de funciones de conmutación, incluyendo cualquier porcentaje de unos. Además, se ha buscado la paralelización del procedimiento con el fin de aprovechar las posibilidades que ofrece su ejecución en sistemas multiprocesador, y más concretamente en sistemas de memoria distribuida tales como redes de estaciones de trabajo, comunes en departamentos dedicados al diseño de sistemas digitales.

7.2. CONCLUSIONES Y PRINCIPALES APORTACIONES

Las conclusiones que pueden extraerse del presente trabajo son las siguientes:

a) En la actualidad, los dispositivos lógicos programables, especialmente las FPGAs, han adquirido una gran importancia, al permitir realizar diseños bastante complejos en muy poco tiempo, con una gran fiabilidad, y con un costo muy reducido. Además, abaratan el proceso de diseño de sistemas de gran complejidad al poder utilizarse como prototipos. Concretamente, en este tipo de dispositivos, así como en PLAs, el costo de una puerta EXOR es idéntico al de una puerta OR, de forma que pueden aprovecharse perfectamente las ventajas que ofrece la lógica AND-EXOR:

- Los circuitos diseñados en esta lógica precisan de menor número de puertas para su realización.
- El test resulta más sencillo que en la lógica AND-OR.

b) Se ha efectuado un estudio detallado de la estructura matemática GF(2), sobre la que se apoya la síntesis lógica AND-EXOR, analizando las posibles formas de expresar una función de conmutación como suma EXOR de términos producto.

c) Se ha hecho hincapié en las formas canónicas de representación por su fuerte estructuración, que permite obtener rápidamente expresiones AND-EXOR de una función de conmutación.

d) Se ha realizado una clasificación de las formas canónicas, resutando las MPRM de especial interés por constituir una familiar amplia y que admite la construcción de algoritmos rápidos para su cálculo.

e) Se ha introducido el problema de la minimización AND-EXOR, indicando cómo se podría resolver dentro de las familias canónicas, y mostrando su dificultad en el caso de su planteamiento más general, utilizando formas ESOP.

f) Se ha efectuado una revisión de los procedimientos de minimización AND-EXOR existentes en la literatura. Estos procedimientos pueden clasificarse en dos grupos:

- Procedimientos exactos, que buscan la expresión mínima mediante algún tipo de búsqueda exhaustiva, resultando poco útiles en la práctica debido al gran tiempo de ejecución que consumen.
- Procedimientos aproximados, que buscan una solución próxima al mínimo mediante

algún tipo de heurística, resultando de mayor utilidad que los exactos por precisar mucho menor tiempo para obtener una solución. Los procedimientos que mejores resultados obtienen de los aparecidos en la bibiografía son los basados en la aplicación de reglas de reescritura, como es el caso de EXMIN2 ó EXORCISM-MV-2, aunque no presentan comportamientos satisfactorios en todos los casos, sobre todo si las funciones contiene un porcentaje alto de unos.

g) Se ha planteado la minimización AND-EXOR como un problema de minimización combinatoria, y se ha propuesto el algoritmo de Enfriamiento Simulado para abordar su resolución. Por tanto, a diferencia de lo que ocurre con los restantes procedimientos, se va a utilizar un método no determinista para realizar la búsqueda de soluciones.

h) Se ha desarrollado un nuevo procedimiento para la minimización AND-EXOR, denominado RRMIN2, capaz de minimizar funciones completa ó incompletamente espcificadas, con una ó varias salidas, y que actúa de la siguiente forma:

- En primer lugar, se obtiene una forma inicial AND-EXOR partiendo de la expresión como suma de minterms de la función. Esta forma inicial es la forma canónica MPRM óptima, que se obtiene a través de la aplicación de una serie de algoritmos rápidos desarrollados en la memoria a tal efecto.
- Después, el procedimiento optimiza la expresión AND-EXOR mediante de reglas de reescritura cuya aplicación es controlada mediante un proceso de Enfriamiento Simulado. Debido a las especiales características de la minimización AND-EXOR, ha sido necesario desarrollar un método distinto de los usuales para controlar la bajada de la temperatura.

i) Para minimizar funciones incompletamente especificadas se ha desarrollado un procedimiento aproximado de asignación de las indiferencias, denominado ASIGMIN, y que actúa sobre las formas MPRM.

j) Para minimizar funciones con varias salidas, RRMIN2 realiza primero una minimización de cada salida individualmente, y despues, partiendo de la solución obtenida, efectúa una minimización conjunta de todas ellas.

Con el objetivo de aprovechar recursos tales como redes de estaciones de trabajo conectadas en red, o cualquier otro tipo de sistema multiprocesador, se ha abordado la paralelización de RRMIN2, para lo cual:

k) Se ha efectuado una revisión de los procedimientos usuales de paralelización para problemas de optimización combinatoria, y más concretamente, para el algoritmo de Enfriamiento Simulado. Debido a cuestiones de optimización de comunicaciones y a las características del árbol de búsqueda obtenido en RRMIN2, se ha optado por un procedimiento híbrido de los existentes.

l) Se ha obtenido una versión paralela de RRMIN2, a la que se ha denominado PRRMIN, apta para ser ejecutada en cualquier sistema multiprocesador. El método consiste en hacer que cada procesador realiza una búsqueda por separado, y cada cierto número de iteraciones, pongan

en común las soluciones obtenidas, seleccionando la mejor de ellas, y continuando la búsqueda a partir de la misma.

Una vez desarrollados los procedimientos RRMIN2 y PRRMIN, se han implementando en lenguaje C y se ha utilizado el software de dominio público PVM basado en paso de mensajes para la versión paralela. A los programas realizados se les ha denominado TESRRMIN2 y TESTPRRMIN, respectivamente, y se han utilizado para obtener los resultados experimentales:

m) Se han ajustado los parámetros de RRMIN2 para obtener las mejores prestaciones con el menor tiempo de ejecución posible. Como resultado de este ajuste se han obtenido cadenas de Markov muy cortas, lo que implica que el proceso de Enfriamiento Simulado no se realiza llegando a un estado de cuasi-equilibrio en cada temperatura. Por tanto, la demostración de convergencia del mismo no es aplicable, y se tiene un esquema de enfriamiento distinto al utilizado usualmente.

n) RRMIN2 no presenta ningún problema de convergencia, y se ha realizado una caracterización del mismo en función de los parámetros de control del Enfriamiento Simulado, determinándose que el mejor parámetro para controlar el tiempo de ejecución y/o la calidad de la solución es el incremento inicial de temperatura.

o) Se han obtenido resultados estadísticamente fiables para la minimización de funciones completamente especificadas de una salida con 6, 8 y 10 variables, y un 25%, 50% y 75% de unos. Los resultados se han comparado con los procedimientos más significativos, obteniéndose que RRMIN2 mejora, con respecto a la media de los otros métodos, entre un 9% y un 58% el número de términos producto obtenido para la funciones a minimizar. Los tiempos de ejecución invertidos por RRMIN2 son mayores que los de los otros procedimientos.

p) Se han minimizado funciones incompletamente especificadas, obteniendose mejores resultados mediante el método de asignación de indiferencias ASIGMIN, que si se realizan las asignacines triviales.

q) Se ha realizado la minimización de funciones con varias salidas, observando cómo el número de términos productos necesarios por salida disminuye al aumentar el número de las mismas, al aumentar la probabilidad de tener términos comunes.

r) Se ha comparado RRMIN2 con EXMIN2 y EXORCISM-MV-2 al minimizar una serie de funciones "benchmark", obteniéndose los mejores resultados con RRMIN2.

s) Se han realizado pruebas para analizar la ganancia en velocidad y mejora de prestaciones que se obtienen al utilizar el procedimiento paralelo PRRMIN, obteniendo ganancias de hasta 4.54 para obtener una misma solución y mejoras de hasta un 15% en el número de términos producto invirtiendo el mismo tiempo de ejecución sobre los resultados obtenidos con un sólo procesador al utilizar 6 estaciones de trabajo conectadas en red.

7.3. LÍNEAS DE INVESTIGACIÓN FUTURAS

El trabajo desarrollado en la presente memoria ha mostrado dos posibles líneas de investigación de interés:

a) En primer lugar, se ha aplicado un esquema de Enfriamiento Simulado que en lugar de llegar a situaciones de cuasi-equilibrio en cada temperatura, utiliza unos decrementos muy pequeños de temperatura. El resultado que se ha obtenido es una situación de convergencia pasando por estado de no-equibrio, que probablemente se produzca de manera particular en el caso de RRMIN2. Sería interesante tratar de realizar un estudio teórico para determinar bajo qué condiciones se puede dar este tipo de convergencia mediante situaciones de no-equilibrio (tratar de encontrar una cota superior para el incremento de temperatura inicial que garantiza la convergencia, por ejemplo), puesto que se ha comprobado que produce un gran ahorro de tiempo de ejecución.

b) En segundo lugar, se podría tratar de ampliar el procedimiento RRMIN2 a la minimización de funciones multivaluadas. Para ello, habría que generalizar la notación del vector L al caso de variables multivaluadas y reformular las reglas de reescritura de manera adecuada para que su aplicación resulte sencilla y rápida.

c) Adaptación evaluación de prestaciones de PRRMIN en multicomputadores y multiprocesadores, donde los costos de comunicación son más reducidos.

d) Posibilidad de adaptar el método basado en reglas de reescritura y Enfriamiento Simulado para la minimización multifuncional AND-OR.

APÉNDICE A

DEMOSTRACIÓN DE LA PROPOSICIÓN 4.19

A continuación se presenta la demostración de la proposición 4.19. Se trata de una demostración puramente matemática, y por esa razón se ha dejado para este Apéndice.

A.1. DEMOSTRACION DE LA PROPOSICION 4.19

Proposición 4.19: *Sea a un número entero. Entonces:*

$$(a+1)^n - a^n = \sum_{i=0}^{n-1} a^{n-1-i}(a+1)^i \qquad \text{(A.1)}$$

Demostración: La demostración se va a efectuar por inducción. Para n=2, se muy fácil ver que se cumple A.1 (para n=1 también es evidentemente cierta, pero no aporta nada a la demostración, por eso se empieza en n=2); basta con sustituir en la citada expresión,

obteniendo:

$$(a+1)^2-a^2=a+(a+1) \tag{A.2}$$

que es cierto.

Para completar la demostración, habrá que demostrar que se cumple:

$$(a+1)^{n+1}-a^{n+1}=\sum_{i=0}^{n} a^{n-i}(a+1)^i \tag{A.3}$$

suponiendo que A.1 es cierta. En efecto:

$$(a+1)^{n+1}-a^{n+1}=(a+1)(a+1)^n-a\cdot a^n= \tag{A.4}$$

$$=a(a+1)^n-a\cdot a^n+(a+1)^n=a\left[(a+1)^n-a^n\right]+(a+1)^n= \tag{A.5}$$

sustituyendo A.1, se tiene:

$$=a\sum_{i=0}^{n-1} a^{n-1-i}(a+1)^i+(a+1)^n= \tag{A.6}$$

Finalmente, metiendo a dentro de la sumatoria y después $(a+1)^n$, se obtiene:

$$=\sum_{i=0}^{n-i} a^{n-1}(a+1)^i+(a+1)^n=\sum_{i=0}^{n} a^{n-i}(a+1)^i \tag{A.7}$$

con lo que queda demostrada la proposición.

APÉNDICE B

ESQUEMA DE ENFRIAMIENTO SIMULADO UTILIZADO EN RRMIN2

En este Apéndice se obtienen algunas expresiones relacionadas con el esquema de Enfriamiento Simulado que se utiliza en RRMIN2.

B.1. ESQUEMA DE ENFRIAMIENTO UTILIZADO EN RRMIN2

A continuación se va a detallar el esquema de enfriamiento que se ha utilizado en RRMIN2, obteniendo expresiones referentes al número de iteraciones (es decir, el número de cadenas de Markov completadas) realizadas en función de los parámetros del control del Enfriamiento, y alguna comparación con los esquemas de enfriamiento usuales.

Los parámetros de control que se utilizan son la temperatura inicial c_0, la temperatura final c_f, el incremento inicial de temperatura Δ_0 y el parámetro b. En el esquema de enfriamiento que se está utilizando, el incremento de temperatura en cada paso del enfriamiento va a ir disminuyendo, de manera que en el paso i:

$$\Delta_i = \frac{\Delta_{i-1}}{b} \tag{B.1}$$

En función del incremento de temperatura inicial, se tendrá:

$$\Delta_i = \frac{\Delta_0}{b^i} \tag{B.2}$$

y por lo tanto, la temperatura a la que se efectúa la iteración i será:

$$c_i = c_{i-1} - \Delta_i = c_{i-1} - \frac{\Delta_0}{b^i} \tag{B.3}$$

Efectuando operaciones en la expresión anterior, se obtiene:

$$c_i = c_0 - \Delta_0 \cdot \frac{1 - \frac{1}{b^i}}{1 - \frac{1}{b}} \tag{B.4}$$

La iteración *iter* en que se alcanza la temperatura final T_f vendría dada por:

$$c_f = c_0 - \Delta_0 \cdot \frac{1 - \frac{1}{b^{iter}}}{1 - \frac{1}{b}} \tag{B.5}$$

de donde despejando:

$$iter = \frac{\ln\left(\frac{\Delta c_0 - (T_0 - T_f)\cdot(1 - \frac{1}{b})}{\Delta c_0}\right)}{\ln(\frac{1}{b})} \tag{B.6}$$

Puesto que la primera iteración empieza en i=0, el número total de iteraciones que se efectúan en una minimización de RRMIN2 sería iter+1, pero teniendo en cuenta que la comparación para el final del proceso se efectúa antes de realizar la iteración correspondiente, el número de iteraciones que se realizan es precisamente, iter.

Nótese que en determinadas situaciones el argumento del logaritmo neperiano del numerador de (B.6) puede hacerse negativo, con lo cual iter $\rightarrow \infty$ y el algoritmo entraría en un bucle infinito. Para evitar esto, se ha impuesto un límite inferior al incremento de temperatura, de manera que éste no puede ser nunca menor que un cierto valor Δ_m (Véase 4.7.2), que por razones de precisión en los cálculos en coma flotante, se ha fijado a:

$$\Delta_m = 0.00005 \tag{B.7}$$

Introduciendo el límite inferior para el incremento de temperatura Δ_m en el cálculo del número de iteraciones, (B.6) se tranformaría en (B.8), expresión que relaciona los parámetros de control del proceso de Enfrimiento Simulado con el número de iteraciones que se van a efectuar en el mismo.

$$iter = \begin{cases} \dfrac{\ln\left(\dfrac{\Delta_0 - (c_0 - c_f)\cdot\left(1-\dfrac{1}{b}\right)}{\Delta_0}\right)}{\ln\left(\dfrac{1}{b}\right)} & si \ \dfrac{(c_0 - c_f)\cdot\left(1-\dfrac{1}{b}\right)}{\Delta_0} > 1 \\ 1 + \dfrac{\ln\left(\dfrac{\Delta_0}{\Delta_m}\right)}{\ln(b)} + \dfrac{c_0 - c_f - \Delta_0 \cdot \left(\dfrac{1-\left(\dfrac{1}{b}\right)^{\frac{\ln\left(\frac{\Delta_0}{\Delta_m}\right)}{\ln(b)}}}{1-\dfrac{1}{b}}\right)}{\Delta_m} & si \ \dfrac{(c_0 - c_f)\cdot\left(1-\dfrac{1}{b}\right)}{\Delta_0} \leq 1 \end{cases} \tag{B.8}$$

APÉNDICE C

INTEGRACIÓN DE RRMIN2 EN HERRAMIENTAS DE DISEÑO

En este último Apéndice se describe el interfaz que presentan los programas TESTRRMIN2 y TESTPRRMIN, así como las herramientas de conversión desarrolladas que permiten integran RRMIN2 en las herramientas de diseño más usuales.

C.1. INTERFAZ DE TESTRRMIN2 Y TESTPRRMIN

Los programas desarrollados, TESTRRMIN2 y TESTPRRMIN tienen el mismo interfaz, salvo los parámetros adicionales que precisa el último. Por esta razón, de ahora en adelante se hará referencia únicamente a TESTRRMIN2. Existen tres posibilidades para introducir parámetros a TESTRRMIN2:

a) Mediante parámetros introducidos en la línea de órdenes.
b) En modo interactivo, contestando a una serie de preguntas sobre los valores de los parámetros.
c) Mediante un fichero de configuración.

Por defecto se espera la introducción de parámetros desde la línea de órdenes, de manera que si se quiere entrar en modo interactivo, hay que ejecutar la orden:

testrrmin2 -i

En cuanto a la opción c), TESTRRMIN2 incluye un fichero de configuración por defecto, llamado *.config*. Este fichero incluye los valores por defecto de los parámetros calculados en el Capítulo 6. Si no se indica nada en la línea de órdenes, TESTRRMIN2 utiliza este fichero de configuración, y da prioridad a los valores nuevos que se introduzcan en la línea de órdenes, o en interactivo. Si se desea introducir un nuevo conjunto de valores por defecto, éste debe incluir en un fichero cuyo formato se describen en C.1.2, y se debe ejecutar:

testrrmin2 -c fichero_configuración

En el siguiente apartado se proporciona el formato completo de la línea de órdenes de TESTRRMIN2.

C.1.1. Parámetros de TESTRRMIN2

En la notación que sigue, los parámetros entre corchetes son opcionales, y los que aparecen entre paréntesis significa que se han de introducir si no aparecen en .config (si no se ha puesto la opción -c) o en el fichero de configuración especificado mediante la opción -c. resto obligatorios. Si un parámetro no aparece entre corchetes ni entre paréntesis, significa que es obligatorio.

testrrmin2 [-i] [-c fichero_de_configuración] (opcion) (n) (n_func) (n_pru) (n_indif) (c_0) (Δ_0) (c_f) *(b) (p_regla1i) (p_regla2i) (p_regla3i)* (l_0) ($p_{fallo i}$) ($p_{fallo j}$) *(PPO) (t_div) fichero_resultados fichero_estado fichero_entrada fichero_salida*

En el caso de TESTPRRMIN habría que añadir dos parámetros más, (n_proc) y (n_iter).

Si se tiene un fichero de configuración especificado, sólo hay que poner en la línea de órdenes aquellos que no aparezcan en el citado fichero, y en el mismo orden en que se han enunciado.

Si se utiliza el modo interactivo, se ignoran todos los parámetros que se introduzcan, incluso mediante fichero de configuración, y el usuario es preguntado por todos y cada uno de los mismos.

El parámetro *opción* hace referencia a las opciones descritas en el Capítulo 6, y los parámetros que se introduzcan deberán ser coherentes con la opción elegida.

C.1.2. Formato del fichero de configuración

El formato de una entrada del fichero de configuración es el siguiente:

.parámetro valor

donde *parámetro* puede ser cualquiera de los citados anteriormente, y *valor* una asignación válida que se desee hacer del mismo. Nótese que cada asignación tiene que efectuarse en una línea distinta, y que el primer carácter de cada línea debe ser un ".".

Los nombres que se utilizan para los parámetros son los siguientes:

.op	opción	.b	b
.n	n	.p1	p_regla1
.nf	n_func	.p2	p_regla2
.np	n_pru	.p3	p_regla3
.ni	n_indif	.l	l_0
.ti	c_0	.pi	p_{falloi}
.dt	Δ_0	.pj	p_{falloj}
.tf	c_f	.ppo	PPO
.tdiv	t_div		

C.2. HERRAMIENTAS DE CONVERSIÓN DE RRMIN2

Junto con los programs TESTRRMIN2 y TESTPRRMIN, se han desarrollado unas herramientas de conversión para integrar el procedimiento RRMIN2 en programas de diseño de circuitos integrados y sistemas digitales. El formato de entrada que acepta RRMIN2 se basa en la notación del vector L introducida en el Capítulo 4, que no es estándar. Usualmente, para describir funciones de conmutación se suele utilizar un formato llamada ESPRESSO, y por ello, se ha desarrollado un programa denominado e2L que convierte ficheros ESPRESSO a ficheros RRMIN2. De esta forma, se consigue que TESTRRMIN2 acepte funciones de conmutación dadas de forma estándar.

En cuanto a la salida, RRMIN2 vuelve a proporcionarla en notación del vector L, mientras que las herramientas CAD normalmente utilizan como entrada algún lenguaje HDL como puede ser Verilog o VHDL. Por ello, se ha realizado un programa, denominado L2v que convierte ficheros en formato RRMIN2 a lenguaje VERILOG.

El formato de estos dos programas es:

e2L fichero_espresso fichero_RRMIN2
L2v fichero_RRMIN2 fichero_verilog

De esta forma, RRMIN2 puede utilizarse para coger como entrada una función de

conmutación dada en formato ESPRESSO, minimizarla en lógica AND-EXOR, y generar un circuito de salida en formato Verilog que pueda ser aprovechado por cualquier herramienta CAD

BIBLIOGRAFÍA

[ABO95] Aborhey, S.: "ACEM: A minimisation method for exclusive-or sum of products expressions". *Proc. IFIP WG 10.5 (Reed-Muller '95), pp. 110-115. Agosto 1995.*

[AAR85] Aarts, E.H.L.; Van Laarhoven, P.J.M.: "A new polynomial time cooling schedule". *Proc. IEEE Int. Conf. on Computer-Aided Design, pp. 206-208. Santa Clara. 1985.*

[AAR88] Aarts, E.H.L.; Korst, J.H.M.; Van Laarhoven, P.J.M.: "A quantitative analysis of the simulated annealing algorithm: a case study for the traveling salesman problem". *Journ. of Statistical Physics, No. 50, pp. 189-206. 1988.*

[AAR90] Aarts, E.; Korts, J.: *"Simulated Annealing and Boltzman machines: A stochastic approach to combinatorial optimization and neural computing".* Wiley & Sons. 1990.

[BAN90] Banerjee, P.; Jones, M.H.; Sargent, J.S.: "Parallel simulated annealing algorithms for standard cell placement on hypercube multiprocessors". *IEEE Trans. Parallel and Distributed Systems, Vol. 1, pp. 91-106. Jan. 1990.*

[BAR76] Barker, J.A.; Henderson, D.: "What is a liquid ? Understanding the states of matter". *Reviews of Modern Physics, Vol. 48, pp. 587-671. 1976.*

[BHA87] Bhasker, J.; Sahni, S.: "Optimal linear arrangement of circuit components". *Journ. of VLSI and Computer Systems, Vol. 2, pp. 87-109.*

[BEE85] Beenker, G.F.M.; Claasen, T.A.C.M.; Hermens, P.W.C.: "Binary sequences with maximally flat amplitude spectrum". *Philips Journ. of Research, Vol. 40, pp. 289-304. 1987.*

[BIO72] Bioud, G.; Davio, M.: "Taylor expansions of Boolean functions and their derivatives". *Philips Research Reports, Vol. 27, pp. 1-6. 1972.*

[BON86] Bonomi, E.; Lutton, J.-L.: "The asymptotic behaviour of quadratic sum assignment problems: a statistical mechanics approach". *European Journ. of Operational Research, Vol. 26, pp. 295-300. 1986.*

[BRA91] Brand, D.; Sasao, T.: "On the minimization of AND-EXOR expressions". *Res. Rep. RC 16739, IBM T.J. Watson Research Center. Abril 1991.*

[BRA93] Brand, D.; Sasao, T.: "Minimization of AND-EXOR Expressions Using Rewrite Rules". *IEEE Trans. on Computers, Vol. 42, No. 5. Mayo 1993.*

[BRI88] Brigham, E.O.: *"The fast fourier transform and its applications".* Prentice-Hall, New Jersey. 1988.

[BUR84] Burkard, R.E.; Rendl, F.: "A themodynamically motivated simulation procedure for combinatorial optimization problems". *European Journ. of Operational Research, Vol. 17, pp. 169-174. 1984.*

[CAR85] Carnevalli, P.; Coletty, L.; Patarnello, S.: "Image processing by simulated

annealing". *IBM Journ. of Research and Development, No. 29, pp. 569-579. 1985.*

[CAS87a] Casotto, A.; Sangiovanni-Vicentelli, A.L.: "Placement of standard cells using simulated annealing on the connection machine". *Proc. IEEE Int. Conf. on Computer-Aided Design, pp. 350-353. Santa Clara, 1987.*

[CAS87b] Casotto, A.; Romeo, F.; Sangiovanni-Vincentelli, A.: "A parallel simulated annealing algorithm for the placement of macro-cells". *IEEE Trans. Computer-Aided Design, Vol. CAD-6, pp. 838-847. 1987.*

[CHA87] Chams, M.; Hertz, A.; De Werra, D.: "Some experiments with simulated annealing for couloring graphs". *European Journ. of Operational Research, Vol. 32, pp. 260-266. 1987.*

[CHA96] Chandy, J.A.; Banerjee, P.: "Parallel Simulated Annealing Strategies for VLSI Cell Placement". *Proc. 9th International Conf. on VLSI Design. Bangalore, India. January 1996.*

[COH62] Cohn, M.: "Inconsistent canonical forms of switching functions". *IRE Transaction Electronic Computers, Vol. 11, pp. 284-285. 1962.*

[CSA93] Csanky, L.; Perkowski, M.A.; Schäfer, I.: "Canonical restricted mixed-polarity exclusive-OR sums of products nad the efficient algorithm for their minimisation". *IEE Proceedings-E, Vol. 140, No. 1. January 1993.*

[DAM89a] Damarla, T.; Karpovski, M.G.: "Detection of stuck-at and bridging faults in Reed-Muller canonical (RMC) networks". *IEE Proceedings. Vol. 136, Pt E, No. 5. Septiembre 1989.*

[DAM89b] Damarla, T.; Karpovski, M.G.: "Fault detection in combinational networks by Reed-Muller Transforms". *IEEE Trans. on Computers, Vol. 38. No. 6, pp. 788-797. Junio 1989.*

[DAR87] Darema, F.; Kirkpatrick, S.; Northon, V.A.: "Parallel algorithms for chip placement by simulated annealing". *IBM J. Research and Development, VOl. 31, No. 3, pp. 391-402. 1987.*

[DAV78] Davio, M.; Deschamps, J.-P.; Thayse, A.: *"Discrete and Switching Functions".* (St. Sephorinm, Switzerland:-Georgi). 1978.

[DES93] Deschamps, J.P.: *"Diseño de circuitos integrados de aplicación específica".* Ed. Paraninfo, Madrid. 1993.

[DIS86] Distante, F.; Piuri, V.: "Optimum behavioral test procedure for VLSI devices: a simulated annealing approach". *Proc. IEEE Int. Conf. on Computer Design, pp. 31-35. Port Chester. 1986.*

[DRE87] Dress, A.; Kruger, M.: "Parsimonious phylogenetic tress in metric spaces and simulated annealing". *Advances in Applied Mathematics, Vol. 8, pp. 8-37. 1987.*

[ESC95] Escobar, M.; Somenzi, F.: "Synthesis of AND-EXOR expressions via satisfiability". *IFIP WG. 10.5, pp. 80-87. Agosto 1995.*

[EVE67] Eve, S.; Kohavi, I.; Paz, A.: "On minimal modulo-2 sums of products for

switching functions". *IEEE Trans. Elec. Comp., Vol. EC-16, pp. 671-674. Octubre 1967.*

[FEL50] Feller, W.: *"An introduction to Probability Theory and Its Applications I".* Wiley, New York. 1950.

[GAR79] Garey, M.R.; Johnson, D.S.: *"Computers and Intractability: A Guide to the Theory of NP-Completeness".* W.H. Feeman and Co. San Francisco, 1979.

[GEI94] Gist, A.; et al.: *"PVM 3 User's Guide and Reference Manual".* Tec. Rep. ORNL/TM-12187. Mayo, 1994.

[GEM87] Geman, S.: *"Stochastic relaxation methods for image restoration and expert systems".* Automated Image Analysis: Theory and Experiments, Academic Press, New York. 1987.

[GOL89] Goldberg, D.E.: *"Genetic Algorithms in Search, Optimization and Machine Learning".* Addison Wesley, 1989.

[GRE86] Green, D.H.: *"Modern Logic Design".* Addison-Wesley Publishing Company, Wokingham, England. 1986.

[GRE87] Green, D.H.: "Reed-Muller expansions of incompletely specified functions". *IEE Proceedings-E, Vol. 134, No. 5, pp. 228-236. September 1987.*

[GRE90a] Green, D.H.: "Reed-Muller expansions with fixed and mixed polarities over GF(4)". *IEE Proceedings-E, Vol. 137, No. 5, pp. 380-388. September 1990.*

[GRE91] Green, D.H.: "Families of Reed-Muller canonical forms". *INT. J. Electronics, Vol. 70, No. 2, pp. 259-280. 1991.*

[GRE92] Green, D.H.; Khuwaja, G.A.: "Simplification of switching functions expressed in Reed-Muller algebraic form". *IEE Proceedings-E, Vol. 139, No. 6. Noviembre 1992.*

[HAO96] Haomin, W.; Perkowski, M.A.: "Generalized Partially-Mixed-Polarity Reed-Muller Expansion and Its Fast Computation". *IEEE Trans. on Computers, Vol. 45, pp. 1084-1088. 1996.*

[HAR90] Harking, B.: "Efficient algorithm for canonical Reed-Muller expansions of Boolean functions". *IEE Proceedings-E, Vol. 137, No. 5, pp.366-370. September 1990.*

[HEL88] Helliwell, M.; Perkowski, M.A.: "A fast alforithm to minimize multi-output mixed-polarity generalized Reed-Muller forms". *Proc. 25th ACM/IEEE Design Automation Conf. , Anaheim, pp. 427-432. Junio 1988.*

[HOP89] Ho, P.M.; Perkowski, M.A.: "Systolic Architecture for Solving Combinatorial Problems of Logic Design". *Proc. International Symposium on Circuits and Systems ISCAS'89. Mayo 1989.*

[HOP90] Ho, P.M.; Perkowski, M.A.: "Performance Analysis of a Parallel Architecture for Solving Combinatorial Problems". *17th Intern. Symp. on Computer Architecture. Seattle, WA. Mayo 1990.*

[HOR83] Horowitz, E.; Zorat, A.: "Divide and Conquer for Parallel Processing". *IEEE Trans. on Computers, Vol. C-32, No. 6, pp. 582-585. June 1983.*

[HUR85] Hurst, S.L.; Miller, D.M.; Muzio, J.C.: *"Spectral Techniques in Digital Logic"*. Academic Press, New York. 1985.

[JOH88] Johnson, D.S.; Aragon, C.R.; McGeoch, L.A.; Schevon, C.: "Optimization by simulated annealing: an experimental evaluation, Part II", *AT&T Bell Laboratories, Murray Hill (NJ), 1988.*

[KIR83] Kirkpatrick, S.; Gelatt, C.D.: "Optimization by simulated annealing". *Science, No. 220, pp. 671-680. 1983.*

[KRA87] Kravitz, S A.; Rutenbar, R.A.: "Placement by simulated annealing on a multiprocessor". IEEE Trans. Computer-Aided Design, Vol. CAD-6, pp. 534-549. July 1987.

[KUM94] Kumar, V.; Grama, A.Y.; Vempaty, N.R.: "Scalable Load Balancing Techniques for Parallel Computers". *J. Parallel and Distributed Comp., 22, pp. 60-79. 1994*

[LAA87] Laarhoven, P.J.M., Aarts, E.H.L.: *"Simulated Annealing: Theory and Applications"*. Reidel, Dordrecht. 1987.

[LAA88] Laarhoven, J.J.M. Van; Aarts, E.H.L.; Van Lint, J.H.; Wille, L.T.: "New upper bounds for the football pool problem for 6,7 and 8 matches". *Journ. of Combinatorial Theory A. 1988.*

[LEE92] Lee, S.-Y.; Lee, D.-G.: "Asynchronous communication of multiple Markov chains in parallel simulated annealing". *Proc. of the International Conf. on Parallel Processing (St. Charles, IL). pp. III:169-176). Aug. 1992.*

[LEO86] Leong, H.W.: "A new algorithm for gate matrix layout". *Proc. IEEE Int. Conf. on Computer-Aided Design, pp. 316-319. Santa Clara. 1986.*

[LLO93] Lloris, A., Gómez, J.F., Román, R.: "Entropic minimization of multiple-valued functions". *IEEE Proc. of the Int. Symp. on Multiple Valued Logic, pp. 24-28. 1993.*

[LLO96] Lloris, A., Prieto, A.: *"Diseño Lógico"*. McGraw Hill. 1996.

[LYB87] Lyberatos, A.; Wohlfarth, P.; Chantrell, R.W.: "Simulated annealing: an application in fine particle magnetism". *IEEE Trans. on Magnetics, Vol. 21, pp. 1277-1282. 1982.*

[MAL88] Mallela, S.; Grover, L.: "Clustering based simulated annealing for standard cell placement". *Proc. 25th IEEE Design Automation Conf. pp. 312-317. 1988*

[MCK93] McKenzie, L.; Almaini, A.E.A.; Miller, J.F.; Thomson, P.: "Optimization of Reed-Muller logic functions". *Int. J. Electronics, Vol. 75, No. 3, pp. 451-466. 1993.*

[MET53] Metrópolis, N.; Rosenbluth, A.; Rosenbluth, M.; Teller, A.; Teller, E.: "Equation of state calculations by fast computing machines". *Journ. of Chemical Physics, Vol. 21, pp. 1087-1092. 1953.*

[NEL95] Nelson, V.P.; Nagle, H.T.; Carroll, B.D.; Irwin, J.D.: *"Digital Logic Circuit Analysis & Design"*. Prentice Hall, New Jersey. 1995.

[ORT91] Ortega, J.; Lloris, A.; Prieto, A.; Pelayo, F.J.: "Aplicación del espectro de Reed-Muler a la generación de patrones de test". *VI Congreso de circuitos*

integrados, Universidad de Cantabria (Santander). Pp. 453-458. Noviembre 1991.

[ORT93] Ortega, J.; Lloris, A.; Prieto, A.; Pelayo, F.J.: "Test-Pattern Generation Based on Reed Muller Coefficients". *IEEE Trans. on Computers, Vol. 42, No. 8, pp. 968-980. Agosto 1993.*

[PAL96] Palnitkar, S.: *"Verilog HDL. A guide to Digital Design and Synthesis"*. Sunsoft Press, California. 1996.

[PAP79] Papakonstantinou, G.: "Minimization of Módulo-2 Sum of products". IEEE Transactions on Computers, Vol. C-28, No. 2, pp. 163-167. February 1979.

[PAR94] Parrilla, L.; Ortega, J.; Lloris, A.: "Using simulated annealing in the minimisation of AND-EXOR functions". *Electronic Letters, Vol. 30, No. 22, pp. 1838-1839. Octubre 1994.*

[PER89] Perkowski, M.A.; Helliwell, M.; Wu, P.: "Minimization of multiple-valued input multi-output mixed-radix exclusive sums of products for incompletely specified boolean functions". *Proc. 19th Int. Symp. Multiple-Valued Logic, pp. 256-263. Mayo 1989.*

[PER90] Perkowski, M.A.; Chrzanowska-Jeske, M.: "An exact algorithm to minimize mixed-radix exclusive sums of products for incompletely specified boolean functions". *Proc. of the IEEE ISCAS'90, pp. 1652-1655. New Orleans, 1-3 May 1990.*

[PHE84] Phee, D.H.: *"Probabilidad y Estadística (3): Inferencia Estadística"*. Ed. Limusa. 1984.

[POW90] Powley, C.; Ferguson, C.; Korf, R.: "Parallel heuristic search: Two approaches". En *"Parallel Algorithm for Machine Intelligence and Vision"* (Ed. Kumar, V. et al.). Springer-Verlag, NY, 1990.

[RAB96] Rabaey, J.M.: *"Digital Integrated Circuits. A design perspective"*. Prentice Hall Electronics and VLSI series, New Jersey. 1996.

[RED72] Reddy, S.M.: "Easily Testable Realizations for Logic Functions". *IEEE Trans. on Computers, Vol. 21, No. 11, pp.1183-1188. Noviembre 1972.*

[ROS93] Rose, J.; El Gamal, A.; Sangiovanni-Vincentelli, A.: "Architectura of Field-Programmable Gate Arrays". *Proceedings of the IEEE, Vol. 81. No.7. Julio 1993.*

[ROU90] Roussel-Ragot, P.; Dreyfus, G.: "A Problem Independent Parallel Implementation of Simulated Annealing: Models and Experiments". *IEEE Trans. on Computer Aided Degisn, Vol. 9, No. 8, pp. 827-835. Agosto 1990.*

[SAS90a] Sasao, T.; Vesslich, Ph.W.: "On the Complexity of mod-2 sum PLA's". *IEEE Trans. on Computers, Vol. 39, pp. 262-266. Febrero 1990.*

[SAS90b] Sasao, T.: "EXMIN: A Simplification Algorithm for Exclusive-OR-Sum-of-Products Expressions for Multiple-Valued Input Two-Valued Output Functions". *20th Int. Symp. on Multiple-Valued Logic, pp. 128-135. Mayo 1990.*

[SAS93a] Sasao, T.: *"EXMIN2: A Simplification Algorithm for Exclusive-OR-Sum-of-*

Products Expressions for multiple valued input Two valued Output functions". IEEE Tran. on CAD, Vol. 12, pp. 621-632. Mayo 1993.

[SAS93b] Sasao, T.: "An exact minimization of AND-EXOR expressions using BDDs". *IFIP 10.5 Workshop on Application of the Reed-Muller Expansion in Circuit Design, Sept. 1993.*

[SAS93c] Sasao, T.: *"Logic Synthesis and Optimization"*. Kluwer International Seri, pp. 287-312.1993.

[SAS93d] Sasao, T.: "An exact minimization of AND-EXOR expressions using BDDs". *IFIP 10.5 Workshop on Application of the Reed-Muller Expansion in Circuit Design. Septiempre 1993.*

[SAS94] Sasao, T.; Debnath, D.: "An exact minimization algorithm for generalized Reed-Muller expressions". *IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS'94), pp. 460-465, Taipei, Taiwan. Diciembre 1994.*

[SAS95] Sasao, T.: "Representation of logic functions using EXOR operators". *Proc. IFIP WG 10.5 (Reed-Muller '95), pp. 11-20. Makuhari, Chiba (Japan), Agosto 1995.*

[SAU90] Saul, J.M.: "An Improved Algorithm for the Minimization of Mixed Polarity Reed-Muller Representations". *Int. Conf. on Comput. Design: VLSI in Comput. and Processors, pp. 372-375. Septiembre 1990.*

[SEN81] Seneta, E.: *"Non-negative Matrices and Markov Chains"*. Springer-Verlag, New York. 1981.

[SOH95] Shon, A.: "Parallel N-ary Speculative Computation of Simulated Annealing". *IEEE Trans. on Parallel and Distributed Systems, Vol. 6, No. 10, pp. 997-1005. October 1995.*

[SON96] Song, N.; Perkowski, M.A.: "Minimization of Exclusive Sum-of-Productos Expression for Multiple-Valued Input, Incompletely Specified Functions". *IEEE Trans. on CAD, Vol. 15, No. 4. Abril 1996.*

[TAV94] Tavernier, C.: *"Circuitos Lógicos Programables"*. Ed. Paraninfo, Madrid. 1994.

[TOD83] Toda, M.; Kubo, R.; Saitô, N.: *"Statistical Physics"*. Springer-Verlag, Berlín. 1983.

[TRA93] Tran, A.; Lee, E.: "Generalisation of tri-state map and a composition method for minimisation of Reed-Muller polynomials in mixed polarity". *IEE Proceedings-E, Vol. 140, No.1. Enero 1993.*

[TRI93] Trimberger, S.: "A Reprogrammable Gate Array and Applications". *Proceedings of the IEEE, Vol. 81, No. 7, pp. 1030-1041. Julio 1993.*

[VAR91] Varma, D.; Trachtenberg, E.A.: "Computation of Reed-Muller expansions of incompletely specified Boolean functions from reduced representations". *IEE Proceedings-E, Vol. 138, No. 2, pp. 85-92. March 1991.*

[WAH85] Wah, B.W.; Li, G.; Yu, C.F.: "Multiprocessing of Combinatorial Search Problems". *IEEE Trans. on Computer, pp. 93-108. June 1985.*

[WIT90] Witte, E.E.: *"Parallel simulated annealing using speculative computation"*. MSc thesis, Dept. of Computer Sciencie, Washingtong Univ. 1990

[WIT91] Witte, E.E., Chamberlain, R.D.; Franklin, M.A.: "Parallel simulated annealing using speculative computation". *IEEE Trans. Parallel and Distributed Systems, Vol. 2, No. 4, pp. 483-494. Abril 1991.*

[WON88] Wong, D.F.; Leons, H.W; Liu, C.L.: *"Simulated Annealing for VLSI design"*. Kluwer, Boston. 1988.

[YAN94] Yang, M.K.; Das, C.R.: "Evaluation of Parallel Branch-and-Bound Algorithm on a Class of Multiprocessors". *IEEE Trans. on Parallel and Distrubuted Systems, Vol. 4, No. 1, pp. 74-86. Enero, 1994.*

[XIL96] Xilinx: *"The Programmable Logic Data Book"*. Xilinx Inc, California. 199