UNIVERSIDAD DE DEUSTO

Facultad de Ingeniería

NUEVO ALGORITMO PARA LA OPTIMIZACIÓN DEL ÁREA DE UN FLOORPLAN NO PARTICIONADO DE ORDEN CINCO

JOSÉ MIGUEL URQUIJO ARAMBURU

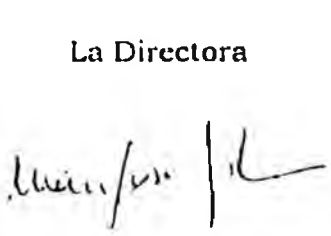
Bilbao, Noviembre de 1999

UNIVERSIDAD DE DEUSTO
Facultad de Ingeniería

NUEVO ALGORITMO PARA LA OPTIMIZACIÓN DEL ÁREA DE UN FLOOORPLAN NO PARTICIONADO DE ORDEN CINCO

Tesis doctoral presentada por D. JOSÉ MIGUEL URQUIJO ARAMBURU
Dirigida por la Dra. María José Gil Larrea y el Dr. José Luis Gutiérrez Temiño

La Directora | El Director | El Doctorando



Bilbao, Noviembre de 1999

UNIVERSIDAD DE DEUSTO
Facultad de Ingeniería

NUEVO ALGORITMO PARA LA OPTIMIZACIÓN DEL ÁREA DE UN FLOOORPLAN NO PARTICIONADO DE ORDEN CINCO

Tesis doctoral presentada por D. JOSÉ MIGUEL URQUIJO ARAMBURU
Dirigida por la Dra. María José Gil Larrea y el Dr. José Luis Gutiérrez Temiño

La Directora El Director El Doctorando

Bilbao, Noviembre de 1999

RESUMEN

El diseño de Circuitos Integrados (CI) se ha convertido en un proceso tan complejo, que impone el uso casi obligado de algoritmos específicos desarrollados sobre herramientas informáticas. En este sentido se han planteado múltiples teorías que dan lugar, en la mayoría de los casos, a soluciones heurísticas más o menos acertadas. Una de las principales problemáticas que se plantea en el diseño de cualquier CI es la obtención del floorplan óptimo en área.

Entre los floorplans hay un tipo especial denominado "no particionado" o "wheel" que consta sólo de cinco módulos y que se caracteriza por tener una estructura tal que la única forma de particionarlo y combinarlo es mediante un corte y una unión Z. La obtención del floorplan óptimo en área implica una búsqueda exhaustiva por todos los nodos que representan las implementaciones de cada módulo o bloque componente. Esta problemática aún no tiene una solución adecuada para ciertos wheels. Cuando el número de módulos y/o de sus implementaciones alcanza cierto grado de complejidad, puede llegar a ser implanteable por el tiempo y la memoria que se necesitaría para su resolución.

El presente trabajo de investigación aporta un algoritmo denominado AWO, que soluciona este problema, determinando las implementaciones adecuadas para la obtención del floorplan no particionado de orden cinco óptimo. Para ello, eliminamos en cada una de las fases del diseño, todas las implementaciones redundantes con zonas desaprovechadas que pudieran generarse. Las pruebas a las que hemos sometido este novedoso algoritmo y los aportados por los diferentes investigadores, ponen de manifiesto que los algoritmos ES y AreaMin no siempre obtienen el área óptima. Además, demuestran que los tiempos de proceso y memoria requeridos para el algoritmo BB y wheels de 25 módulos, así como para el algoritmo OPT y wheels de 625 módulos, son bastante más elevados que los empleados por nuestro algoritmo. Los resultados obtenidos indican que con el algoritmo AWO siempre se obtiene el área óptima, es más rápido y puede manejar floorplans más complejos, ya que el consumo de memoria es menor.

ABSTRACT

Designing Integrated Circuits (IC) has become such a complex process that requires an almost compulsory use of specific algorithms developed with computerised tools. In this respect, a lot of theories have been set forth. In most cases these have given rise to a more or less satisfactory heuristic solutions. Getting the best possible floorplan in area is one of the main problems that arises in the design of any IC.

Among the floorplans there is a special one called non-partitioned or wheel, which consists of only five modules and which is characterised by having such a structure that the only way of dividing and combining it is by means of a cut and a Z association. Obtaining the best possible floorplan in area involves an exhaustive search throughout all the nodes which represent the implementations of each module or forming block. An adequate solution to this problem has not yet been found for certain wheels. Whenever the number of modules and/or their implementations reaches a certain amount it turns out to be impossible due to time and the memory requirements.

Our contribution is an algorithm called AWO, which solves this problem specifying the appropriate implementations so as to obtain the best possible non-partitioned floorplan of scale five. In order to do it, all the redundant implementations which could be generated in each of the design phases have been eliminated. After testing this new algorithm and the ones offered by other researches, the results we have got to show that:

- The best possible area is not always obtained with the AreaMin and ES algorithms, compared with AWO algorithm.
- The processing times and the memory required are much higher for the BB algorithm and 25-module wheels and for the OPT algorithm and 625-module wheels, than for the rest.
- Our algorithm always obtains the best possible area, it is quicker and can handle more complex floorplans since the amount of memory required is lower.

LABURPENA

Zirkuito Integratuen diseinua hain korapilotsu bilakatu da, ezen tresna informatiko oso bereiziak erabiliz algoritmo berriak erabiltzera behartzen gaituen. Zentzu honetan, kasu gehienetan konponbide heuristiko ezberdinak eman dituzten hainbat teoria proposatu dira. Edozein ZI diseinatzeko zailtasunik haundiena areako floorplan hoberena aurkitzea da.

Bada floorplan-en artean bereizi bat wheel edo "partitu eza" deritzona, bost modulok bakarrik osatzen dute eta ezaugarri nagusiak, estruktura partitu ahal izateko eta bateratzeko ebaki baten bidez eta Z elkarte baten beharra dutela dira. Areako floorplanik onena lortzeak modulo edo blokeen implementazioak erakusten dituzten nodo guztien artean bilatze zehatza eskatzen du. Problematika honek ez du, orain arte, wheels batzuentzako erantzun egokirik. Modulo eta edo beraien implementazioen arabera gorengo mailara heldutakoan, planteatu ezinera heldu daitezke bai denboragaitik bai behar izango luketen memoriagatik.

Honako ikerketa lanak AWO izeneko algoritmo bat dakar, arazo honi irtenbidea emanez, bost ordeneko floorplan partitu ezinak lortzeko inplementazio egokiak zehazten dituena. Horretarako, diseinuaren atal bakoitzean sortu daitezkeen inplementazio erredundanteak kenduko ditugu. Algoritmo berri hau gainditu behar izan dituen frogek eta zenbait ikerlarik emandakoak aditzera ematen dute ES eta AreaMin algoritmoen bidez ezin dela beti area onena aurkitu. Eta AWO algoritmoarekin konparatuz, prozesoaren denborak eta beharrezko memoria, BB algoritmoa eta 25eko wheels moduloak baino handiagoa dela, baita 625ko moduloen OPT algoritmoarekiko ere. Ditugun emaitzen arabera gure algoritmoak area onena lortzen du beti, azkarragoa da eta floorplan korapilotsuagoak erabiltzeko gauza da, memoriaren kontsumoa txikiagoa delako.

AGRADECIMIENTOS

Esta tesis doctoral se ha podido realizar gracias al apoyo de diversas personas a las que quiero dedicar mi más sincero agradecimiento:

- A la Dra. María José Gil y al Dr. José Luis Gutiérrez, de la Universidad de Deusto, que aceptaron el desafío de dirigir la presente tesis doctoral y que han posibilitado el desarrollo de la misma a través de sus importantes aportaciones y motivaciones.
- Por su tiempo y atención al Dr. Wai-Kai Chen de la Universidad de Illinois en Chicago y al Dr. Martin D. F. Wong de la Universidad de Texas en Austin, con los que he discutido las soluciones planteadas en la obtención del área óptima de un floorplan no particionado de orden cinco.
- A todos mis compañeros de trabajo y al rector del Instituto Politécnico Jesús Obrero, Javier López Aríztegui, que sin su apoyo y ayuda no hubiera sido posible llevar a cabo dicha tesis.
- A mi mujer, Toñi, que durante todos estos años me ha soportado en los momentos más difíciles y que ha estado a mi lado cuando más lo he necesitado. Así mismo, a mi hijo Aitor, que en este tiempo no le he podido prestar toda la atención que necesitaba y requería.

ÍNDICE

RELACIÓN DE FIGURAS

RELACIÓN DE TABLAS

1. INTRODUCCIÓN

1.1 PROCESO DE DISEÑO DE LOS CIRCUITOS INTEGRADOS

La tecnología empleada en el diseño y fabricación de semiconductores y circuitos integrados (CI) ha experimentado una rápida evolución, desde los años 50 hasta la actualidad. Esta evolución se manifiesta en los cambios producidos desde los circuitos de pequeña escala de integración (SSI), hasta los actuales circuitos de muy alta escala de integración (VLSI). Hoy en día es posible fabricar CI que contengan millones de transistores. Sin embargo, al incrementar la escala de integración, la distribución de los transistores en los CI se torna muy complicada, haciendose imprescindible el uso de herramientas "software" para abordar el problema de optimización del proceso de diseño.

El proceso de diseño de CI de cierto nivel de complejidad consiste en obtener una representación lo suficientemente detallada como para que pueda ser trasladada directamente a un esquema físico, para posteriormente obtener los bloques funcionales que satisfagan el comportamiento requerido. Este proceso se puede describir mediante tres dominios [GAJS85]: funcional, estructural y físico. En la figura 1-1 están representados cada uno de ellos por una línea radial.

El dominio funcional, describe el funcionamiento del sistema pero sin tener en cuenta la parte hardware; es decir, cómo responde un diseño particular a un conjunto dado de especificaciones. El dominio estructural define la arquitectura del sistema, sus componentes y la forma en la que están conectados cada uno de ellos. Por último, el dominio físico describe las propiedades físicas del sistema; es decir, cómo implementar la estructura obtenida en el dominio anterior.

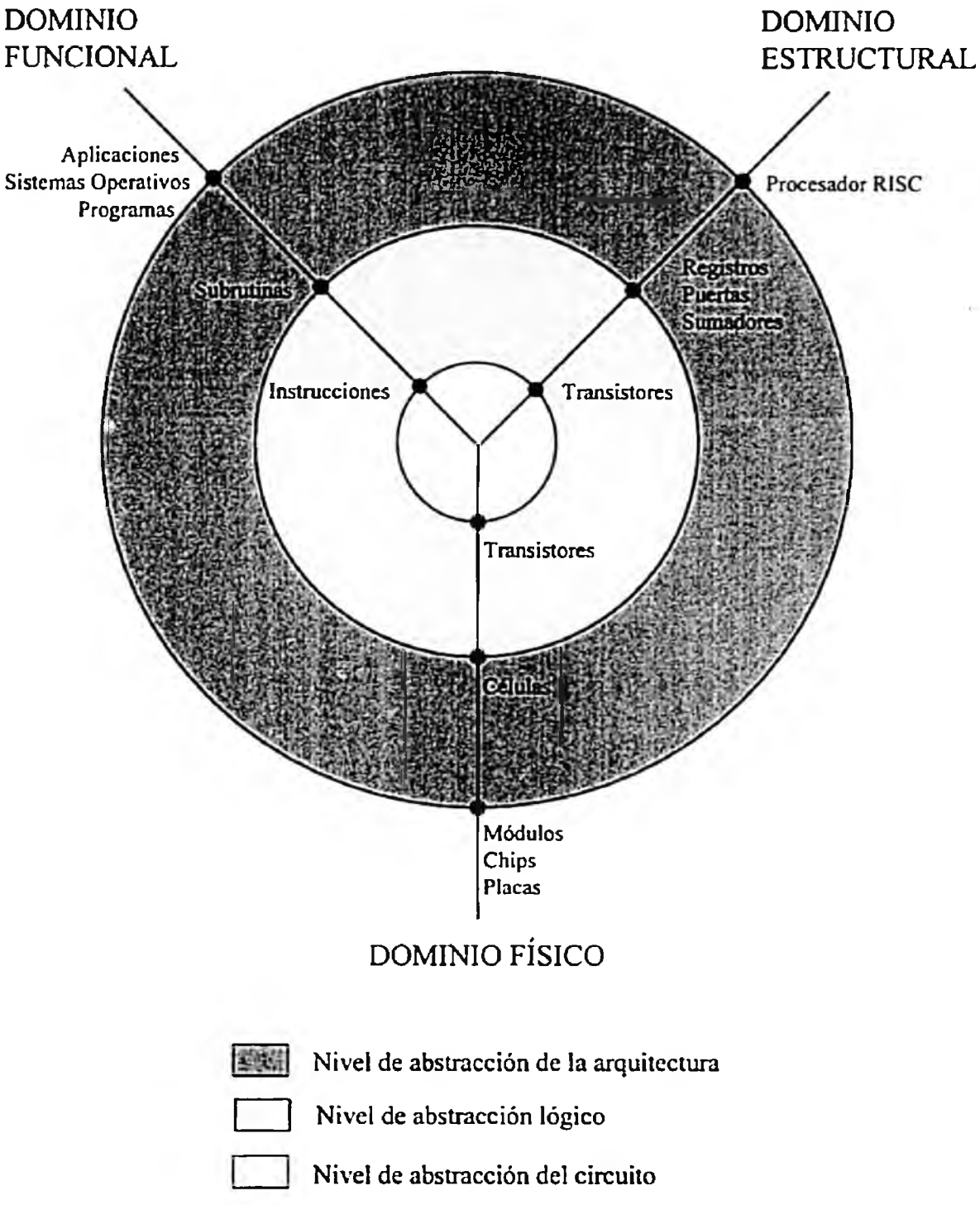


Figura 1-1 Dominios del proceso de diseño de un circuito integrado.

A su vez, cada uno de estos dominios, se puede especificar mediante tres niveles de abstracción comunes en el diseño electrónico: el de arquitectura, el lógico y el del circuito. Los niveles a utilizar dependerán del tipo de diseño así como de la complejidad del sistema, siguiendo siempre el orden de mayor a menor grado de abstracción. Según se va descendiendo de nivel se tiene una información más detallada sobre la implementación. En la figura 1-1 estos niveles están representados mediante círculos concéntricos.

Los objetivos que se pretenden obtener en cada uno de los dominios son muy diversos. Así, en el funcional se debe conseguir que el sistema se comporte acorde a un conjunto de especificaciones, respondiendo en el menor tiempo posible. En el estructural, son reducir el consumo y optimizar la conectividad del circuito, cumpliendo las restricciones impuestas a su comportamiento. En el físico, son los relativos al coste de mercado, a la velocidad y a la especificación del área, así como el cumplimiento de todas las reglas de disposición de las máscaras.

El dominio sobre el que se centra este trabajo de investigación es el físico. El resultado de este dominio es un conjunto de descripciones geométricas correspondientes a unos patrones de máscara en capas, que son utilizados en la realización de los CI. Un CI puede constar de miles o millones de transistores, y cada uno requiere un número determinado de patrones geométricos para su proceso de fabricación.

El diseño físico de un CI se divide en varias fases:

- Partición del circuito.
- Floorplanning.
- Conexionado.
- Compactación.
- Verificación.

El propósito de la partición del circuito, según Bui y otros [BUI87], Fiduccia y Mattheyses [FIDU82] y Kernighan y Lin [KERN70], es dividirlo en partes más pequeñas de modo que el tamaño de los componentes esté dentro de los límites fijados y su complejidad sea asequible (Figura 1-2). Una partición puede dividir un circuito dado en k partes (dos o cinco), lo más similares posible en cuanto a tamaño.

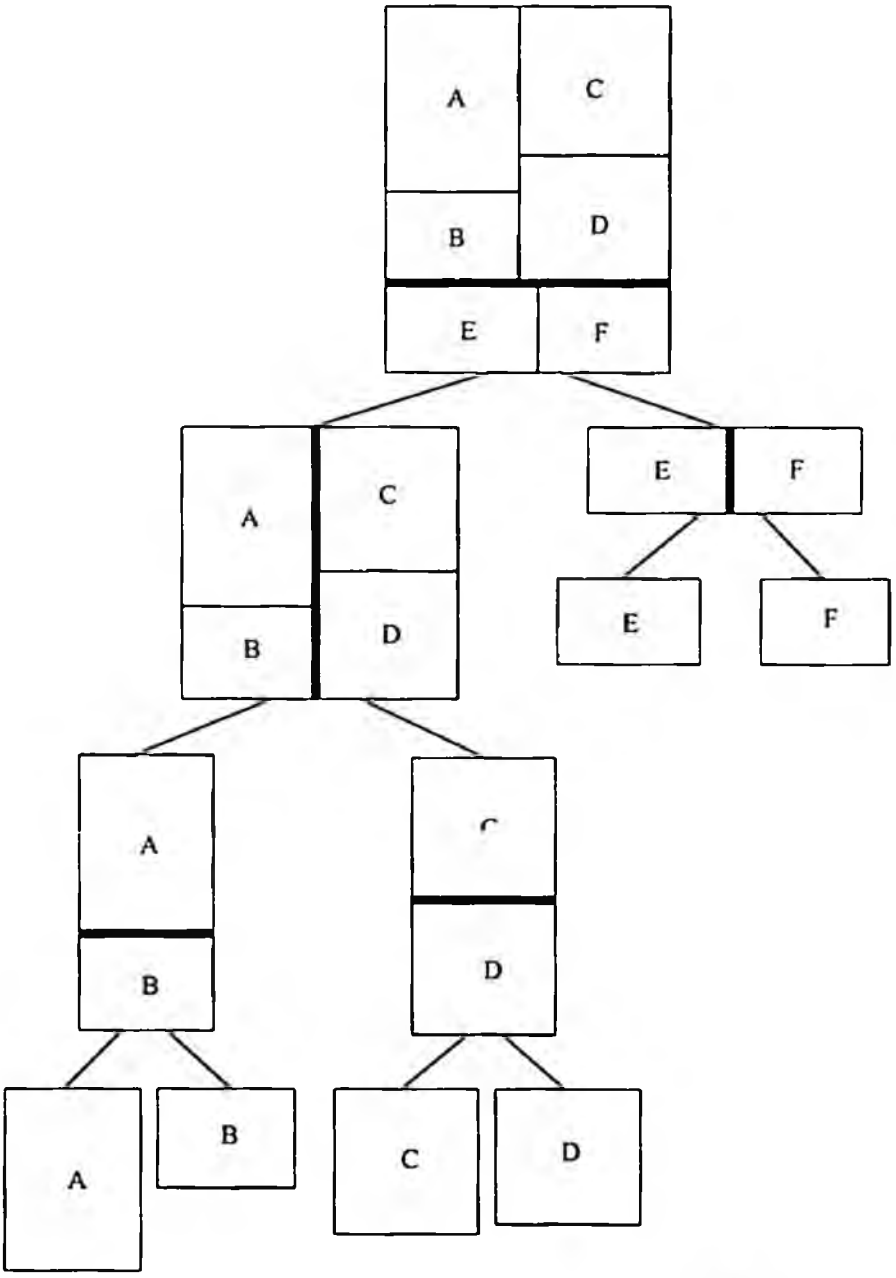


Figura 1-2 Ejemplo de partición con k=2.

Tras particionar el circuito, hay que resolver el problema del elevado tiempo de proceso que supone encontrar las formas, tamaños y posiciones de los módulos, para obtener el floorplan o planificación del layout óptima (Figura 1-3).

Una vez determinadas la forma, tamaño y posición, para cada uno de los elementos del floorplan, se procede a su ubicación. En este caso, el principal problema consiste en situar los elementos de modo que el área del floorplan se minimice obteniéndose todo ello en el menor tiempo de proceso posible. Según Kuh [KUH90], hay dos tipos de elementos a ubicar:

- Células estándar: en las que por lo general, se ignora su tamaño.
- Módulos: en los que es necesario considerar sus especificaciones geométricas (formas y tamaños).

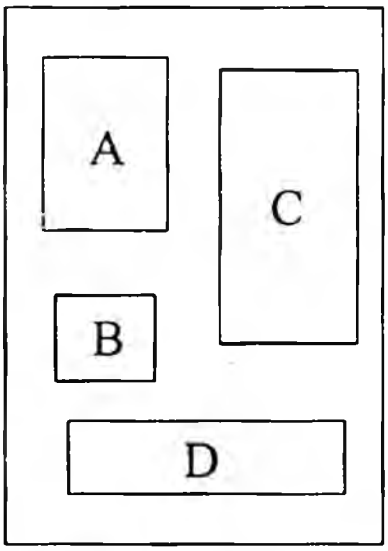


Figura 1-3 Floorplan.

El floorplan obtenido puede conducir a un diseño no implementable físicamente, en cuyo caso, es necesario buscar otro que sí sea realizable. Una

vez logrado el floorplan óptimo, las entradas y salidas de los diferentes módulos se interconectan siguiendo una lista de conexiones, de forma que se satisfagan las especificaciones y restricciones físicas de los componentes, y que el área de distribución del CI sea mínima (Figura 1-4). Esto normalmente se realiza en dos sub-fases conocidas como conexionado global y conexionado detallado.

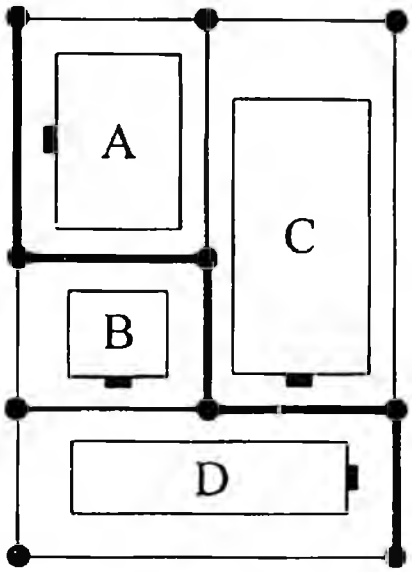


Figura 1-4 Conexionado.

El conexionado global utiliza la lista de canales que pueden ser utilizados para efectuar las conexiones entre los diferentes módulos del circuito. Una vez realizado, con el conexionado detallado se completan las conexiones punto a punto entre los módulos.

Efectuadas las interconexiones necesarias, el layout resultante, es aún susceptible de optimización mediante una compactación del circuito, que como mínimo, elimina espacios inútiles (Figura 1-5). Las reglas seguidas durante el proceso de diseño y el de fabricación no deben violarse durante el proceso de compactación.

Un método de compactación efectivo puede reducir el coste de los circuitos VLSI, dado que el rendimiento de los CI fabricados depende del área total del circuito y de los tiempos de propagación. Cuanto menor sea el valor de estos parámetros, más se incrementará la productividad de los circuitos ([LENG82], [LIAO83] y [GAO89]).

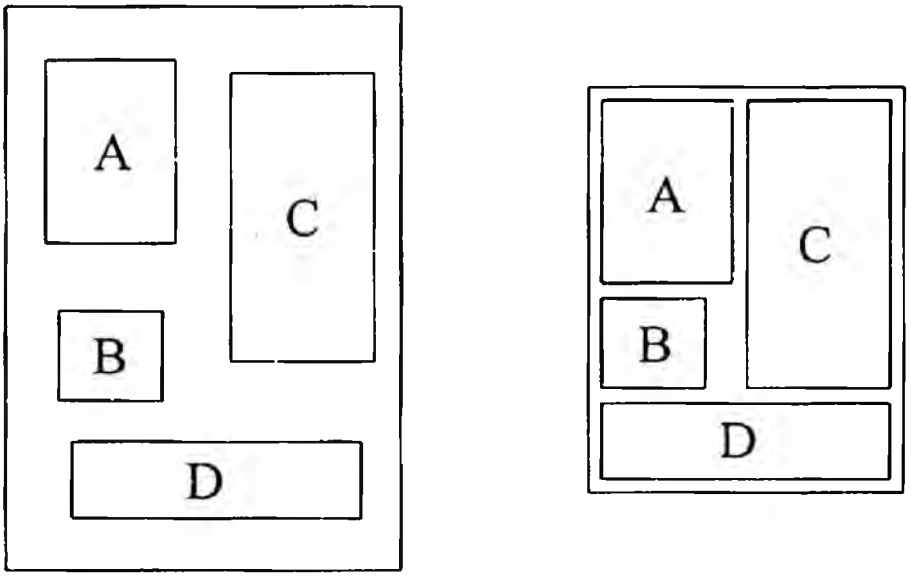


Figura 1-5 Compactación.

Por último, en la fase de verificación se comprueba si se han realizado de forma correcta cada uno de los pasos dados para la obtención del diseño físico del circuito. En primer lugar, se prueba individualmente cada módulo del floorplan con un verificador de tiempos, y posteriormente, el CI de forma íntegra. Para ahorrar trabajo en la determinación del tamaño, la forma y el patillaje de cada módulo, la simulación y prueba de cada uno se realiza antes de ensamblar el layout. Así, si se detecta algún error, es posible subsanarlo evitándose problemas posteriores. Para asegurar la fiabilidad y eficiencia del sistema se realiza además una comprobación final del layout completo.

Por lo tanto, el ciclo de diseño físico de un CI implica iteraciones, tanto dentro de cada fase como entre varias fases; la representación del sistema se va mejorando iterativamente hasta satisfacer las especificaciones del sistema.

1.2 DESARROLLO HISTÓRICO Y SITUACIÓN ACTUAL

La optimización del área de un floorplan debe realizarse con el menor número posible de particiones o uniones, así como con la mínima cantidad de nodos visitados y de implementaciones obtenidas en cada fase, tanto en formas como en tamaños de cada uno de los componentes, ya que así el tiempo de proceso necesario para la obtención de una solución en teoría será menor. Existen numerosos trabajos, y enfoques muy diversos, para la resolución de esta problemática. En la tabla 1-1 se indican los más relevantes.

Inicialmente, la mayoría de los algoritmos abordaban los floorplans particionados, obtenidos mediante cortes o uniones horizontales y/o verticales respectivamente de los módulos que lo conforman, dejando a un lado los no particionados debido a su gran complejidad. El tiempo de proceso de los algoritmos para estos últimos se dispara hasta unos límites que en algunos casos eran implanteables.

Otten, usando la conectividad como información de partida, definió las distancias que tiene que haber entre los módulos, para más tarde, obtener un floorplan con el menor área posible que satisfaciese dichas distancias ([OTTE82] y [OTTE83]). Su algoritmo genera una configuración bidimensional, conservando las distancias entre los módulos, y con ella la obtención del floorplan final aplicando una descomposición jerárquica rectangular particionada.

TIPO DE FLOORPLAN	FORMA DE RESOLUCIÓN	AUTORES
Particionado	Partición rectangular	R.H.J.M.Otten
Particionado	Dualización y partición rectangular	W.R.Heller, G.Sorkin y K.Maling
Particionado	Dualización y partición rectangular	K.A.Kozminski y E.Kinnen
Particionado	Dualización y partición rectangular	B.Lokanathan y E.Kinnen
Particionado	Relaciones en el plano X-Y	L.Stockmeyer
General	Enfriamiento simulado	D.F.Wong y C.L.Liu
General	Grafos duales	S.Wimer, Y.Koren e I.Cederbaum
General	Estructuras de sub-floorplans	K.Chong y S.Sahni
General	Estructuras de sub-floorplans	P.Pan y C.L.Liu
No particionado de orden cinco	Extensión de la técnica de Stockmeyer	T.-C.Wang y D.F.Wong
No particionado de orden cinco	Extensión de la técnica de Stockmeyer	C.-H.J.Chen
No particionado de orden cinco	Extensión de la técnica de Stockmeyer	K.Wang y W.K.Chen
No particionado de orden cinco	Estructuras de sub-floorplans	P.Pan, W.Shi y C.L.Liu
No particionado de orden cinco	Estructuras geométricas	D.Z.Chen y X.Hu

Tabla 1-1 Principales investigaciones realizadas según el tipo de floorplan.

Heller, Sorkin y Maling [HELL82] introdujeron un nuevo enfoque en la resolución de floorplans mediante la dualización y la partición rectangular, más tarde desarrollada por Kozminski y Kinnen [KOZM84]. La solución que plantearon, se basa en lograr mediante grafos la partición de un módulo en sus rectángulos básicos o células, de manera que satisfagan las relaciones de adyacencia existentes entre ellos con el menor área posible. Posteriormente, Lokanathan y Kinnen [LOKA89] desarrollaron un algoritmo para agrupar los módulos en bloques o en supermódulos, de forma que, las uniones entre ellos se formulan previamente mediante un sistema de inecuaciones lineales.

Stockmeyer [STOC83] demostró que en el problema de la orientación, el caso de más complejidad es $O(n^2)$, donde "n" es el número de módulos a considerar, y presentó un algoritmo para determinar de una forma óptima, la orientación de módulos en un tiempo polinómico. Dicho algoritmo, se basa en la composición de las relaciones en el plano X-Y. Además, probó que el problema del floorplan general no jerárquico es NP-completo, ya que no se puede conseguir un algoritmo polinomial que resuelva dicho problema.

Entre los trabajos que utilizan el método de búsqueda aportado por Kirkpatrick, Gelatt y Vecchi [KIRK83] "enfriamiento simulado", para resolver el problema de la optimización del área en floorplans generales, destacan los de Wong y Liu ([WONG86] y [WONG89a]). Los algoritmos que presentan Wong y Liu, consideran simultáneamente la minimización del área y de la longitud del conexionado. Las soluciones obtenidas son cercanas a las óptimas, cuando los módulos son flexibles (módulos rectangulares con forma y dimensión no fijas), pero no así, cuando son rígidos (módulos rectangulares y bloques L con formas y dimensiones fijas).

Wimer, Koren y Cederbaum [WIME88] enunciaron el concepto "espacio desperdiciado cero" y desarrollaron el algoritmo "rama-y-límite" o BB. Con este algoritmo el floorplan óptimo se obtiene en un tiempo de proceso exponencial y por tanto hay ciertos casos que son implanteables. Chong y Sahni

[CHON93], perfeccionaron el algoritmo "rama-y-límite" de Wimer, Koren y Cederbaum explotando la estructura de sub-floorplans que pueden existir dentro de un floorplan general. Sin embargo, dejaron a un lado ciertos casos ya que debido a su complejidad no lo podían resolver.

Basándose en la extensión de las técnicas de particionamiento de Stockmeyer, T.-C. Wang y Wong [WANG92b] y C.-H.J. Chen [CHEN93] concibieron unos algoritmos óptimos para resolver el problema de una clase especial de floorplan no particionado, llamado de orden cinco o "wheel". Estos algoritmos, son más eficaces que el planteado por Wimer, Koren y Cederbaum ya que los tiempos de proceso son considerablemente menores. Así mismo, K. Wang y W.K. Chen [WANG93a] propusieron un nuevo método para resolver los floorplans generales haciendo especial hincapié en los no particionados de orden cinco. Además, demostraron que bajo ciertas restricciones de forma, se puede encontrar una clase de estructuras con espacio desperdiciado cero [WANG93b]. Posteriormente, D.Z. Chen y Hu [CHEN96] perfeccionaron los algoritmos de T.-C. Wang y Wong aprovechando las propiedades geométricas que tienen los módulos y bloques; sin embargo, no siempre obtienen un floorplan no particionado con implementaciones óptimas.

Pan y Liu [PAN95] enfocaron el problema de la resolución del área y tiempo de proceso en los floorplans generales mediante la búsqueda de las realizaciones no redundantes, explotando las estructuras de sub-floorplans que pueden existir dentro de un floorplan. Más tarde, aplicaron esta misma técnica a los floorplans no particionados de orden cinco [PAN96] llegando a mejorar el algoritmo realizado por K. Wang y W.K. Chen.

Sin embargo, la resolución de floorplans no particionados de orden cinco no está resuelto, todavía presenta problemas en cuanto a tiempo de proceso a la hora de obtener el área óptima.

1.3 OBJETIVO DE LA TESIS

Ante la imposibilidad de lograr la optimización de todos los parámetros que intervienen en el diseño físico de un CI, por la enorme complejidad que supone simultanear todos así como el elevado tiempo de proceso, el presente trabajo de investigación se centra en la obtención del floorplan no particionado de orden cinco más óptimo en área en el menor tiempo de proceso posible.

Diversos investigadores han logrado floorplans de este tipo óptimos en área (Tabla 1-1). Sin embargo, o no pueden garantizar que los wheels obtenidos sean los más óptimos por haber utilizado métodos heurísticos o sólo son válidos para floorplans no particionados de orden cinco sencillos.

Nuestro objetivo es desarrollar un método no heurístico y válido para cualquier wheel utilizando nuevas técnicas de construcción. Este método implicará un proceso que cubra las siguientes fases generales:

a) Optimización en área de los módulos que constituyen el floorplan.

b) Construcción del floorplan con los módulos optimizados.

c) Cálculo del área del floorplan.

d) Evaluación del tiempo de proceso.

Para su implementación se ha pretendido desarrollar un conjunto de algoritmos que gestionen la unión de los diferentes bloques eliminando paso a paso todos los elementos redundantes con zonas desaprovechadas.

La presente memoria se estructura en capítulos de la forma que se presenta en la tabla 1-2:

NUEVO ALGORITMO PARA LA OPTIMIZACIÓN DEL ÁREA DE UN FLOORPLAN NO PARTICIONADO DE ORDEN CINCO	
CAPÍTULO 1	Introducción
CAPÍTULO 2	Terminología y definiciones
CAPÍTULO 3	Descripción del problema y soluciones existentes
CAPÍTULO 4	Algoritmo AWO
CAPÍTULO 5	Conclusiones
APÉNDICE A	Resultados comparativos obtenidos por medio de distintos algoritmos
APÉNDICE B	Rutinas más relevantes del algoritmo AWO

Tabla 1-2 Estructura de la memoria.

2. TERMINOLOGÍA Y DEFINICIONES

2.1 FLOORPLAN

Se define como floorplanning la estrategia de distribución del conjunto de módulos y conexiones, que determinan el área activa de un CI. La representación gráfica de dichos módulos sobre un plano se conoce como floorplan (Figura 2-1). Generalmente se exige que estos cumplan ciertos criterios de optimización, el más habitual es circunscribir el conjunto de módulos a un rectángulo de superficie mínima. A veces, la forma y tamaño de dicho rectángulo, ya están predeterminados y el objetivo final en este caso es ubicar en su interior, todos los módulos.

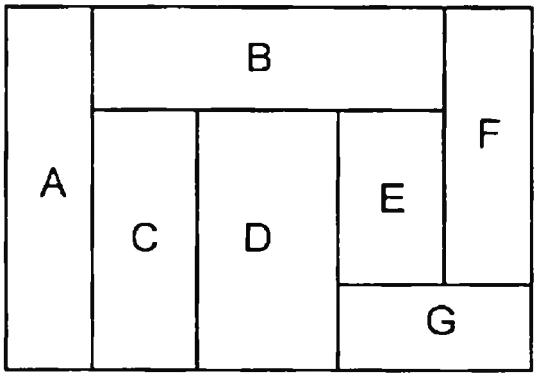


Figura 2-1 Floorplan de 7 módulos.

Cada módulo puede tener múltiples implementaciones, cada una de ellas definida por una anchura y una altura, y en consecuencia tener un tamaño específico.

Agrupando una serie de módulos, se obtiene un bloque que puede tener a su vez distintas formas, tamaños y orientaciones. Sus posibles

implementaciones, que también se representan por la anchura y la altura, determinan cómo deben estar situados cada uno de ellos en el floorplan.

Una vez obtenidas las distintas implementaciones de cada uno de los bloques, éstas se van combinando hasta conseguir aquella o aquellas que obtienen un floorplan óptimo en área.

Para realizar el cálculo del área correspondiente a cada una de las implementaciones, hay que tener en cuenta no sólo el área del elemento obtenido, sino también el área desaprovechada. Cuanto menor sea ésta, menor será el área total del floorplan obtenido en las sucesivas uniones de módulos y/o bloques.

Como se puede observar en la figura 2-2, el área total del floorplan representado, será la suma de cada una de las áreas de los módulos A, B y C más el área desaprovechada o lo que es lo mismo, la anchura total por la altura total:
Área Total = Área de los módulos + Área desaprovechada = (20+12+12)+(3+2) = 49
Área Total = Anchura Total • Altura Total = 7 • 7= 49

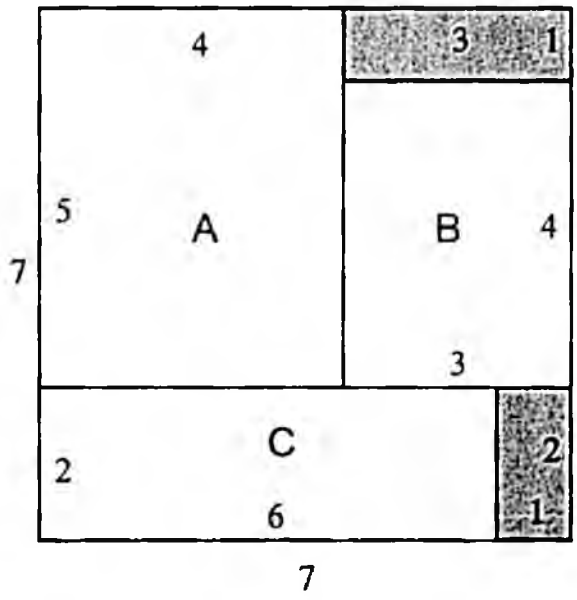


Figura 2-2 Área del floorplan.

2.1.1 Bloque L2

En un bloque L2 obtenido mediante la unión de dos módulos (A y B), tal y como se indica en la figura 2-3, las implementaciones vienen dadas mediante el cuarteto: (wa,wb,ha,hb), donde wa representa la anchura del módulo A, wb la anchura del módulo B, ha la altura del módulo A y hb la altura del módulo B.

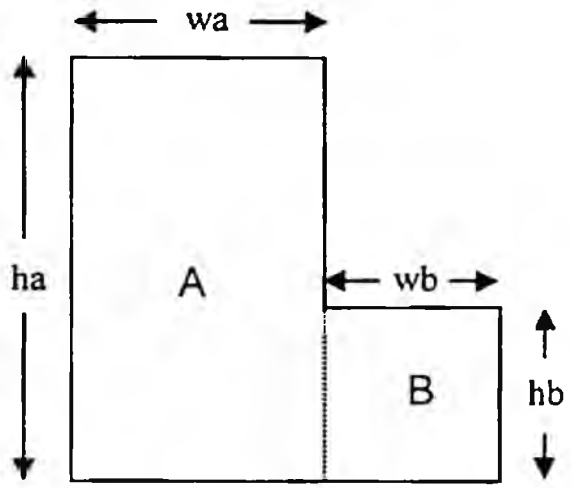


Figura 2-3 Bloque L2.

Como podemos observar en la figura 2-4 un bloque L2 puede tener diferentes configuraciones. Las implementaciones mostradas tanto en la figura 2-4.a como en la figura 2-4.b darán lugar siempre a "wheels" con zonas desaprovechadas ya que la altura del módulo A es menor o igual que la del módulo B.

Ahora bien, cuando la altura del módulo A es mayor que la del módulo B, al tener una configuración más acorde al floorplan de partida, podrán obtenerse con dicho bloque "wheels" que contengan zonas desaprovechadas (Figura 2-4.c y Figura 2-4.e) o podrán ser óptimos en área (Figura 2-4.d).

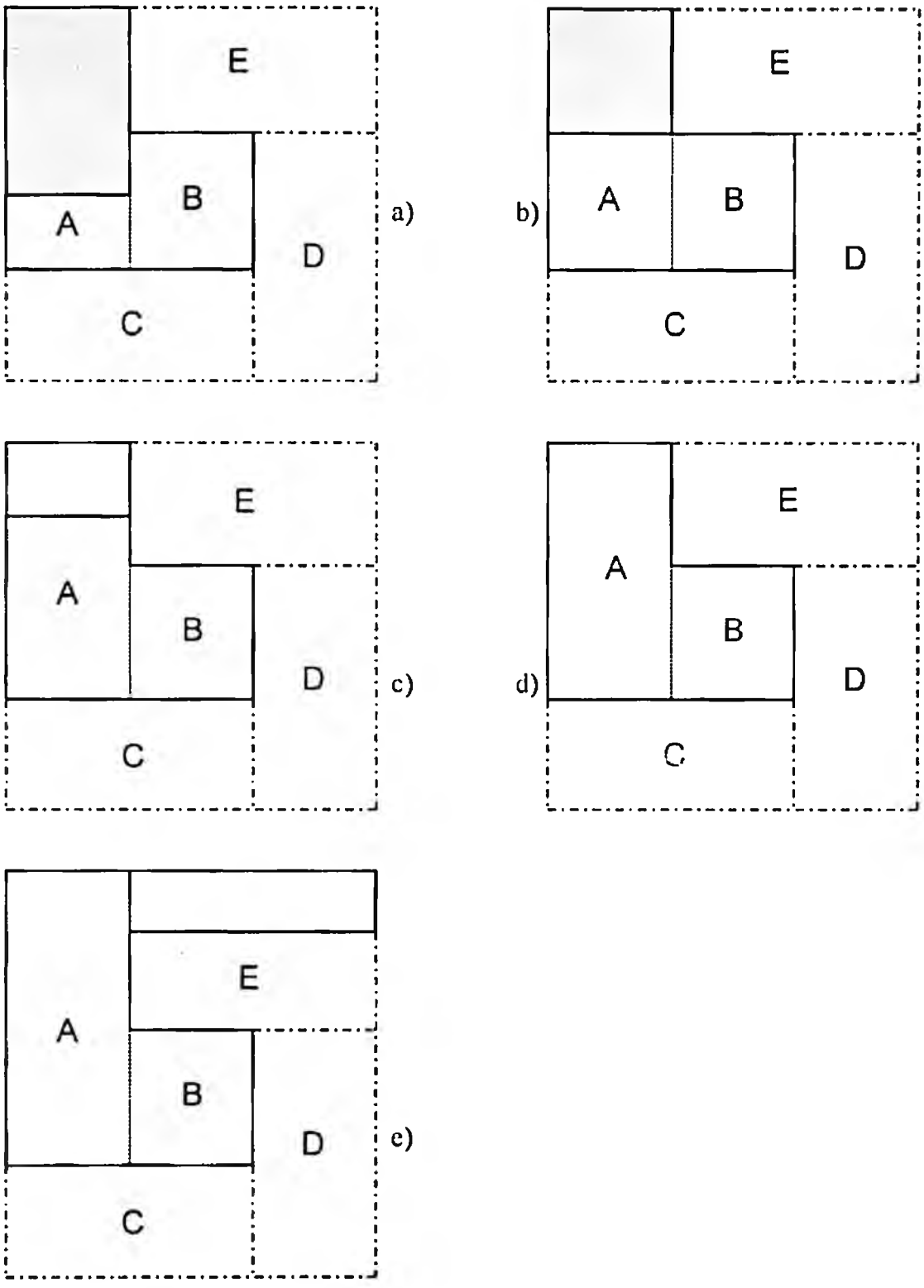


Figura 2-4 Distintos tipos de bloque L2.

2.1.2 Bloque L3

En un bloque L3 obtenido mediante la unión del bloque L2 (A y B) y el módulo C, tal y como se indica en la figura 2-5, las implementaciones vienen dadas mediante el sexteto: (wa,wb,wc,ha,hb,hc), donde wa representa la anchura del módulo A, wb la anchura del módulo B, wc la anchura del módulo C, ha la altura del módulo A, hb la altura del módulo B y hc la altura del módulo C. Esto nos permite conocer en todo momento la anchura y la altura de cada uno de los módulos que conforman este bloque, y en consecuencia, la posibilidad de eliminar muchas uniones futuras que darán lugar a floorplans redundantes con zonas desaprovechadas.

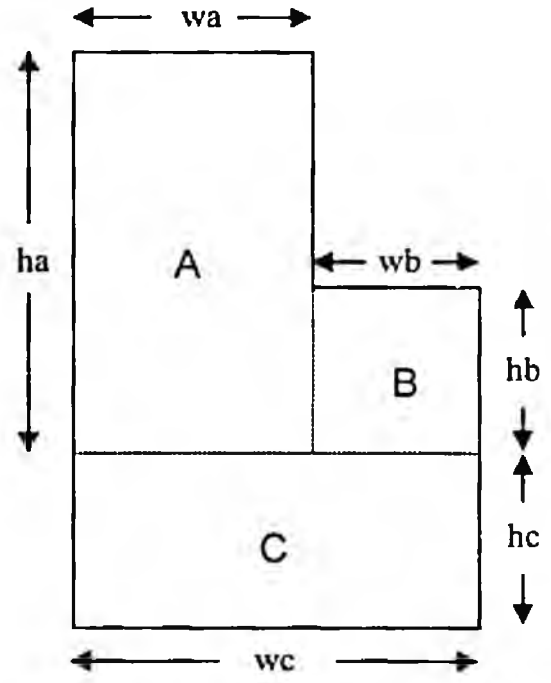


Figura 2-5 Bloque L3.

Un bloque L3 puede tener diferentes configuraciones. Como podemos observar, las implementaciones mostradas en la figura 2-6.a, en la figura 2-6.c y

en la figura 2-6.d darán lugar siempre a "wheels" con zonas desaprovechadas ya que la anchura del módulo C es mayor o menor que la anchura del bloque L2 AB. Sin embargo, cuando la anchura del módulo C es igual que la del bloque L2 AB, al tener una configuración más acorde al floorplan de partida, podrá obtenerse con dicho bloque un "wheel" óptimo en área al no contener zonas desaprovechadas (Figura 2-6.b).

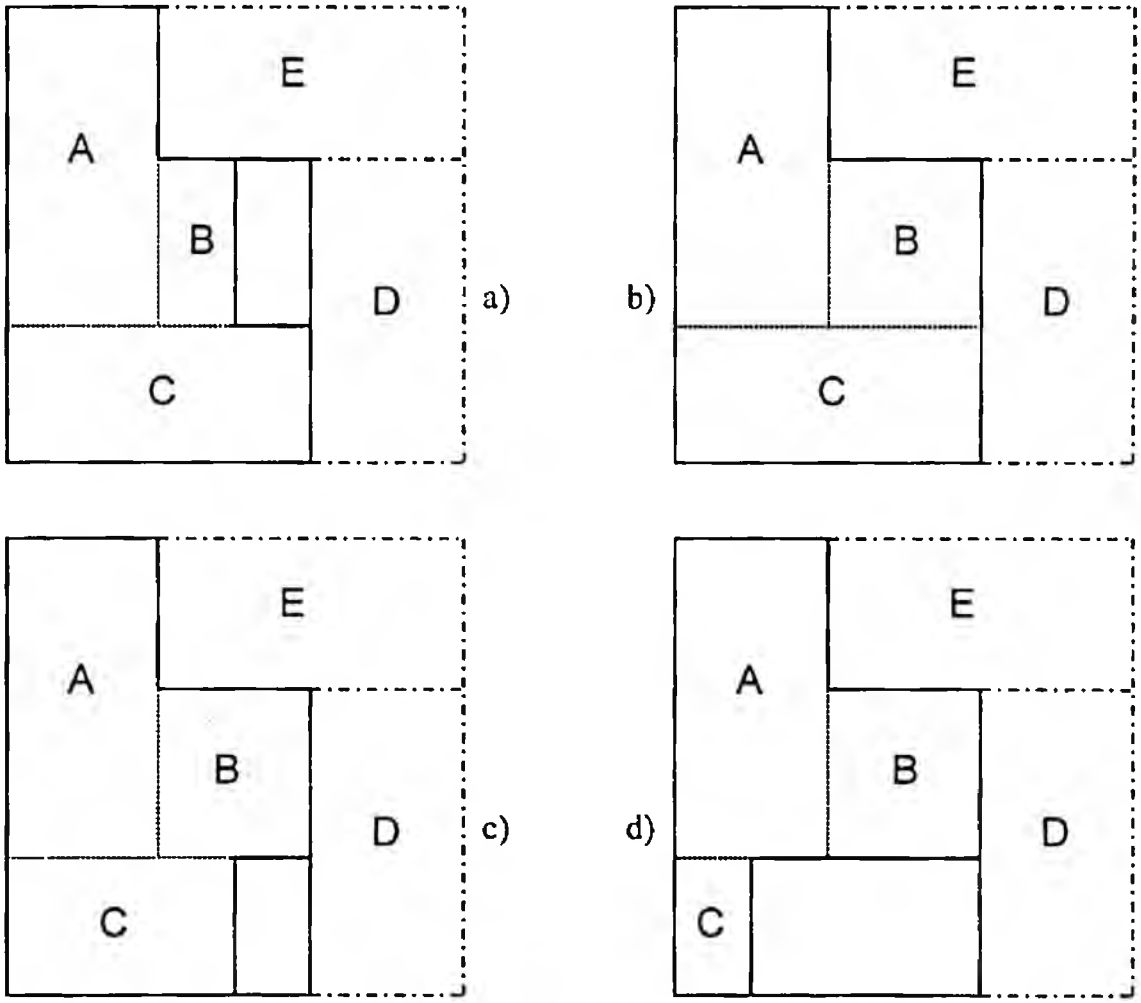


Figura 2-6 Distintos tipos de bloques L3.

2.1.3 Bloque 7

En un bloque 7, el obtenido mediante la unión de dos módulos (D y E), las implementaciones vienen dadas mediante el cuarteto: (wd,we,hd,he), donde wd representa la anchura del módulo D, we la anchura del módulo E, hd la altura del módulo D y he la altura del módulo E (Figura 2-7).

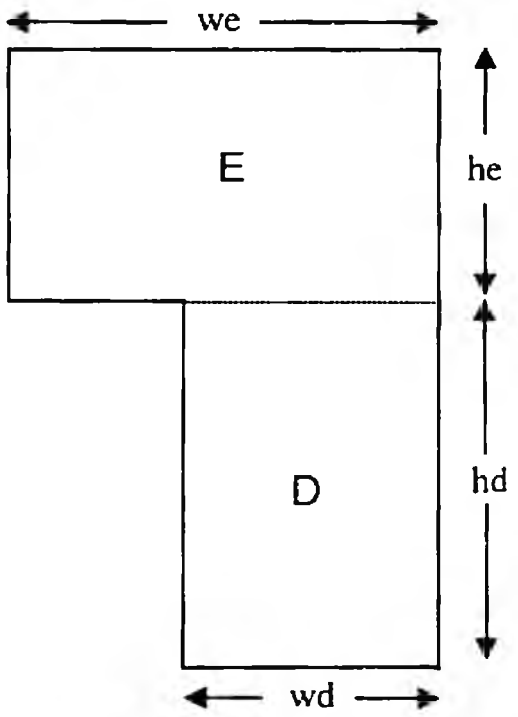


Figura 2-7 Bloque 7.

Como podemos observar en la figura 2-8 un bloque 7 puede tener diferentes configuraciones. La implementación mostrada en la figura 2-8.a dará lugar siempre a un "wheel" con zonas desaprovechadas ya que la anchura del módulo E es mayor que la anchura de los módulos D y B.

Sin embargo, cuando la anchura del módulo E es igual que la anchura de los módulos D y B, al tener una configuración más acorde al floorplan de partida, podrá obtenerse con dicho bloque un "wheel" que sea óptimo en área (Figura 2-8.b). Por último, cuando la anchura del módulo E es menor que la anchura de los módulos D y B, dará lugar siempre a un "wheel" con zonas desaprovechadas (Figura 2-8.c y Figura 2-8.d).

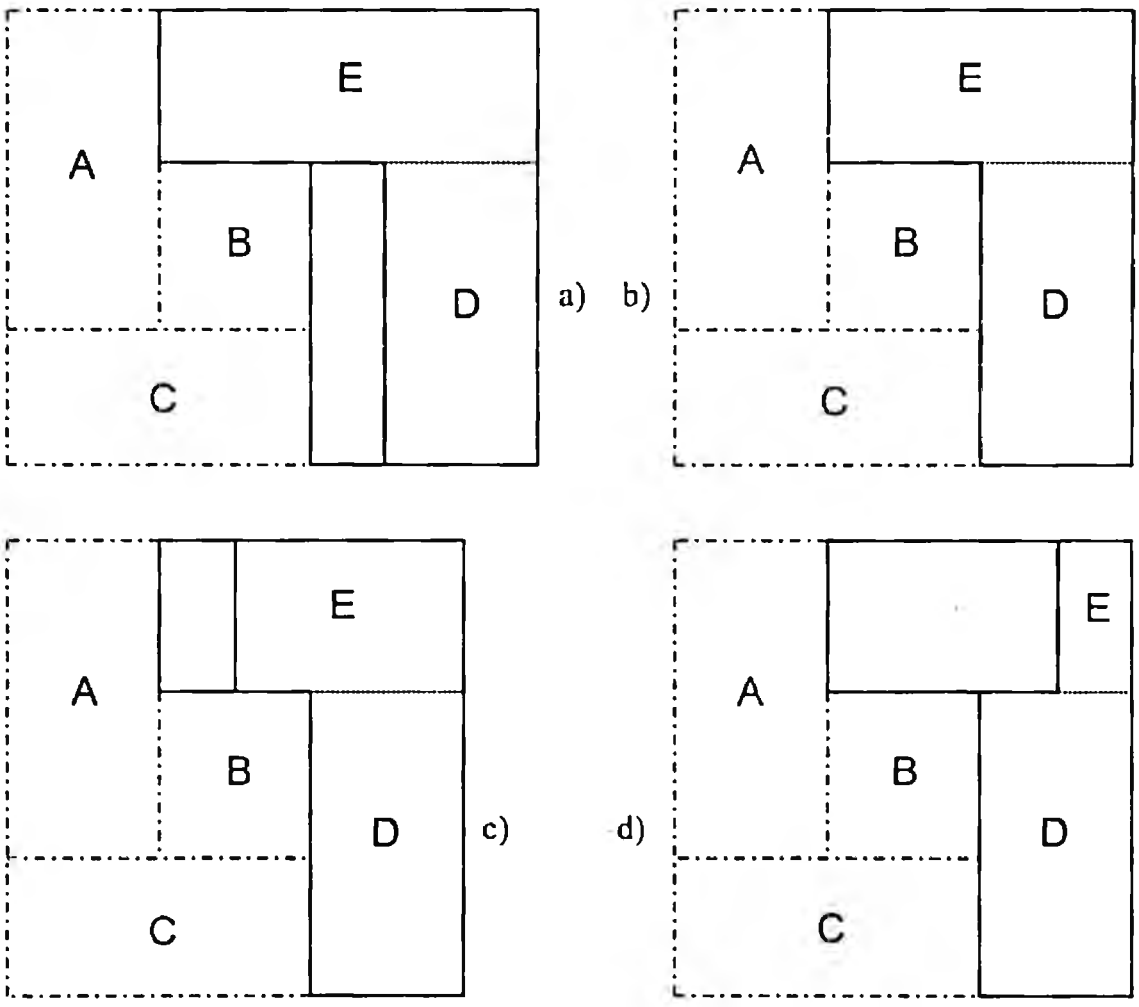


Figura 2-8 Distintos tipos de bloques 7.

2.1.4 Bloque Rectangular

El bloque rectangular se obtiene al unir un bloque L3, representado mediante el sexteto (wa,wb,wc,ha,hb,hc), con un bloque 7 representado mediante el cuarteto (wd,we,hd,he). Dependiendo de las distintas implementaciones de los bloques L3 y 7 el bloque rectangular resultante puede contener zonas desaprovechadas (Figura 2-9.a, Figura 2-9.b y Figura 2-9.c) o por el contrario no contener zonas desaprovechadas (Figura 2-9.d).

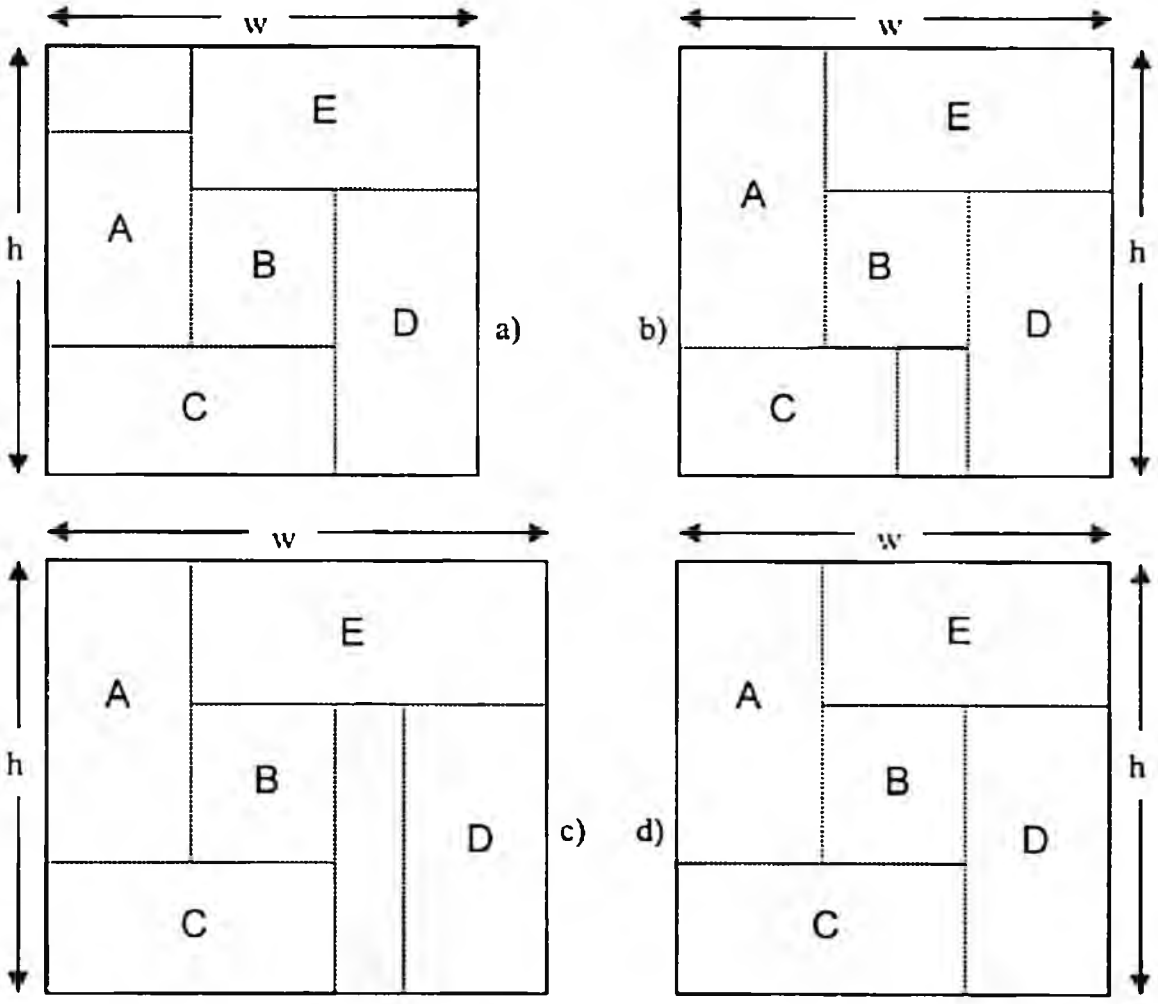


Figura 2-9 Distintos tipos de bloques rectangulares.

Definimos un bloque rectangular mediante el par (w,h), donde **w** representa la anchura y **h** la altura del bloque. Este bloque rectangular puede convertirse a su vez en un nivel superior, en un módulo o rectángulo básico.

2.1.5 Lista-L2

Una lista de implementaciones representada de la siguiente forma $\{(wa_1,wb_1,ha_1,hb_1), (wa_2,wb_2,ha_2,hb_2) \ldots (wa_i,wb_i,ha_i,hb_i) \ldots (wa_j,wb_j,ha_j,hb_j) \ldots (wa_n,wb_n,ha_n,hb_n)\}$, es una lista-L2 si para todo i y j, $1 \leq i \leq j \leq n$ se cumplen las propiedades siguientes:

P1: $wa_i \leq wa_j$

P2: $wb_i \leq wb_j$

P3: $ha_i \geq ha_j$

P4: $hb_i \geq hb_j$

Si se cumplen dichas propiedades para la lista de implementaciones de un bloque L2, todos sus elementos estarán dispuestos en orden creciente de anchuras y en orden decreciente de alturas.

Por ejemplo, las posibles implementaciones de un bloque L2 (Figura 2-10) se pueden agrupar en la lista siguiente: {(1,1,4,3),(2,1,3,2),(4,1,2,1)}.

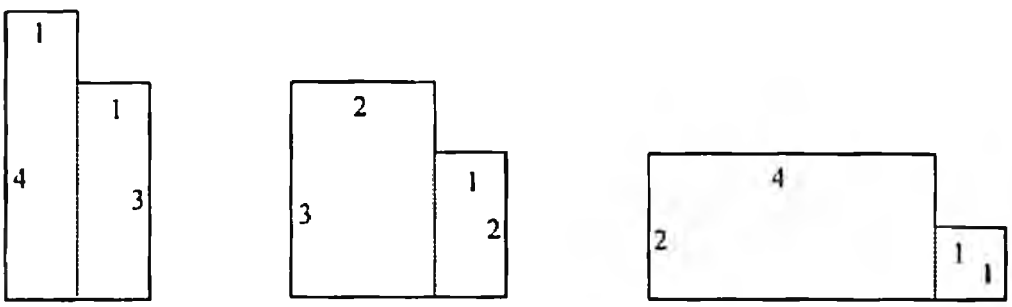


Figura 2-10 Implementaciones de un bloque L2.

Para que ésta sea una lista-L2, cada una de las implementaciones contenidas en ella tendrá que cumplir todas las propiedades anteriormente citadas:

P1: $wa_1=1 \leq wa_2=2 \leq wa_3=4$

P2: $wb_1=1 \leq wb_2=1 \leq wb_3=1$

P3: $ha_1=4 \geq ha_2=3 \geq ha_3=2$

P4: $hb_1=3 \geq hb_2=2 \geq hb_3=1$

Como se puede observar, todas cumplen dichas propiedades y por tanto la lista {(1,1,4,3),(2,1,3,2),(4,1,2,1)} es una lista-L2.

2.1.6 Lista-L3

Una lista de implementaciones representada de la siguiente forma $\{(wa_1,wb_1,wc_1,ha_1,hb_1,hc_1),(wa_2,wb_2,wc_2,ha_2,hb_2,hc_2)...(wa_i,wb_i,wc_i,ha_i,hb_i,hc_i)$ $...(wa_j,wb_j,wc_j,ha_j,hb_j,hc_j)...(wa_n,wb_n,wc_n,ha_n,hb_n,hc_n)\}$ es una lista-L3, si para todo i y j, $1 \leq i \leq j \leq n$ se cumplen las propiedades siguientes:

P1 : $wa_i \leq wa_j$

P2 : $wb_i \leq wb_j$

P5 : $wc_i \leq wc_j$

P3 : $ha_i \geq ha_j$

P4 : $hb_i \geq hb_j$

P6 : $hc_i \geq hc_j$

Si se cumplen dichas propiedades para la lista de implementaciones de un bloque L3, todos sus elementos estarán dispuestos en orden creciente de anchuras y en orden decreciente de alturas.

Por ejemplo, las posibles implementaciones de un bloque L3 (Figura 2-11) se pueden agrupar en la lista siguiente:

{(1,1,2,3,2,3),(2,1,3,2,1,2),(4,1,5,2,1,1)}. Para que ésta sea una lista-L3, cada una de las implementaciones contenidas en ella tendrá que cumplir todas las propiedades anteriormente citadas:

$P1 : wa_1=1 \leq wa_2=2 \leq wa_3=4$

$P2 : wb_1=1 \leq wb_2=1 \leq wb_3=1$

$P5 : wc_1=2 \leq wc_2=3 \leq wc_3=5$

$P3 : ha_1=3 \geq ha_2=2 \geq ha_3=2$

$P4 : hb_1=2 \geq hb_2=1 \geq hb_3=1$

$P6 : hc_1=3 \geq hc_2=2 \geq hc_3=1$

Como se puede observar, todas cumplen dichas propiedades y por tanto la lista {(1,1,2,3,2,3),(2,1,3,2,1,2),(4,1,5,2,1,1)} es una lista-L3.

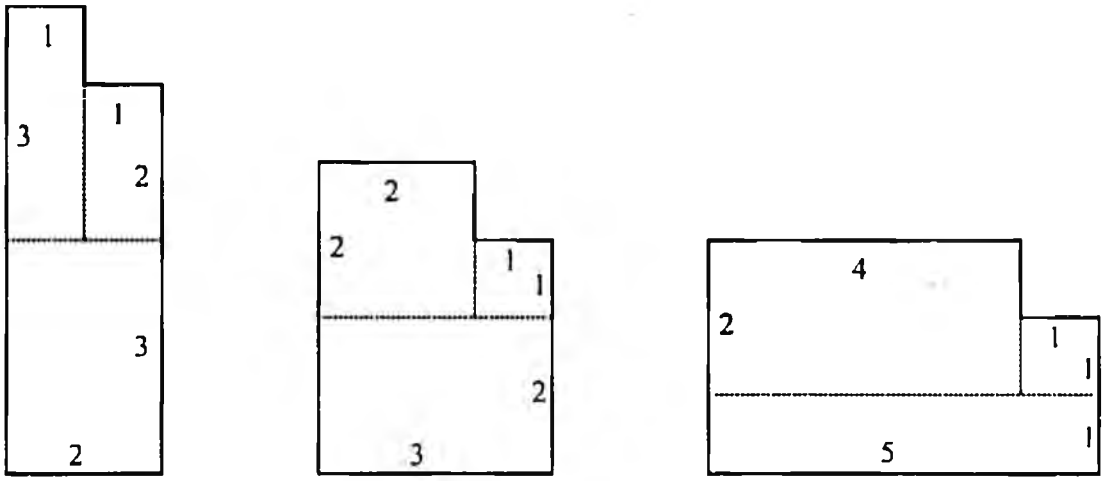


Figura 2-11 Implementaciones de un bloque L3.

2.1.7 Lista-7

Una lista de implementaciones representada de la siguiente forma $\{(wd_1,we_1,hd_1,he_1),(wd_2,we_2,hd_2,he_2) \ldots (wd_i,we_i,hd_i,he_i) \ldots (wd_j,we_j,hd_j,he_j) \ldots (wd_n,we_n,hd_n,he_n)\}$, es una lista-7 si para todo i y j, $1 \leq i \leq j \leq n$ se cumplen las propiedades siguientes:

P7 : $wd_i \leq wd_j$

P8 : $we_i \leq we_j$

P9 : $hd_i \geq hd_j$

P10: $he_i \geq he_j$

Si dichas propiedades se cumplen para la lista de implementaciones de un bloque 7, todos sus elementos estarán dispuestos en orden creciente de anchuras y en orden decreciente de alturas. Como se puede observar la construcción de esta lista-7 es similar a la lista-L2.

Por ejemplo, las posibles implementaciones de un bloque 7 (Figura 2-12) se pueden agrupar en la lista siguiente: {(2,1,3,1),(3,2,2,1),(5,4,1,1)}). Para que ésta sea una lista-7, cada una de las implementaciones contenidas en ella tendrán que cumplir todas las propiedades anteriormente citadas:

P7 : $wd_1=2 \leq wd_2=3 \leq wd_3=5$

P8 : $we_1=1 \leq we_2=2 \leq we_3=4$

P9 : $hd_1=3 \geq hd_2=2 \geq hd_3=1$

P10: $he_1=4 \geq he_2=3 \geq he_3=2$

Como se puede observar, todas cumplen dichas propiedades y por tanto la lista {(2,1,3,1),(3,2,2,1),(5,4,1,1)} es una lista-7.

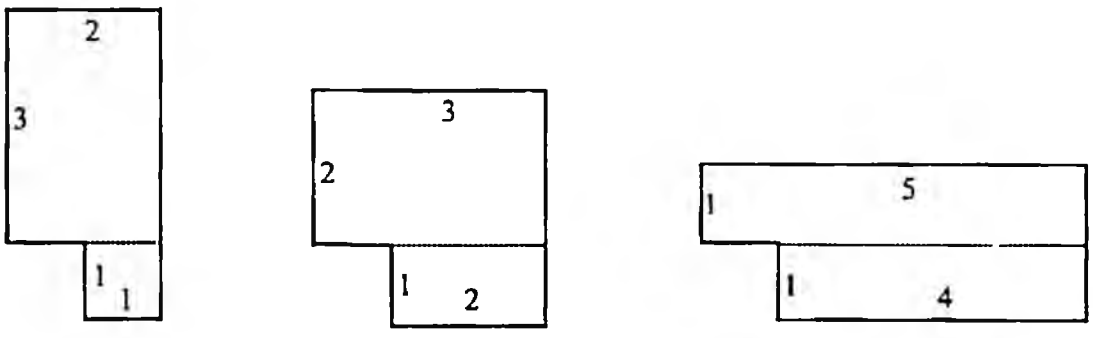


Figura 2-12 Implementaciones de un bloque 7.

2.1.8 Lista-R

Una lista con un conjunto de implementaciones representadas de la siguiente forma $\{(w_1,h_1),(w_2,h_2) \ldots (w_i,h_i) \ldots (w_j,h_j) \ldots (w_n,h_n)\}$, es una lista-R si para todo i y j, $1 \leq i \leq j \leq n$ se cumplen las propiedades siguientes:

P11 : $w_i \leq w_j$

P12 : $h_i \geq h_j$

Por lo tanto, toda lista-R contiene una serie de implementaciones de un bloque rectangular o módulo, en la que todos sus elementos están distribuidos, en orden creciente de anchuras y en orden decreciente de alturas.

Por ejemplo, las posibles implementaciones de un módulo o bloque rectangular (Figura 2-13) se pueden agrupar en la lista siguiente: {(1,6),(2,3),(3,2),(6,1)}. Para que ésta sea una lista-R, cada una de las implementaciones contenidas en ella tendrá que cumplir las propiedades anteriormente citadas:

P11: $w_1=1 \leq w_2=2 \leq w_3=3 \leq w_4=6$ y

P12: $h_1=6 \geq h_2=3 \geq h_3=2 \geq h_4=1$

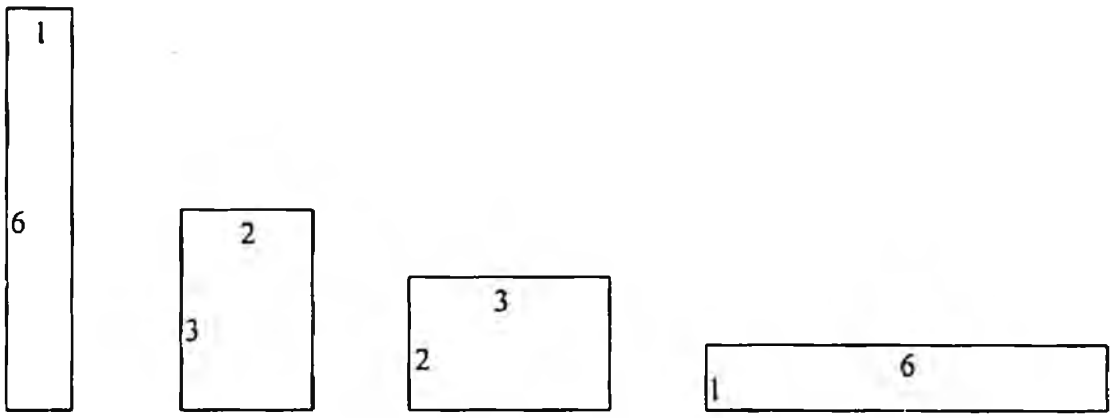


Figura 2-13 Implementaciones de un módulo o bloque rectangular.

Como se puede observar, todas cumplen dichas propiedades y por lo tanto la lista {(1,6),(2,3),(3,2),(6,1)} es una lista-R.

2.1.9 Implementación Redundante

Sean $X=(wa_X,wb_X,ha_X,hb_X)$ e $Y=(wa_Y,wb_Y,ha_Y,hb_Y)$ dos implementaciones cualesquiera de un bloque L2 (Figura 2-14.a y Figura 2-14.b). Se dice que la implementación X es redundante o que domina a la Y cuando se cumplen las propiedades siguientes:

$$wa_X \geq wa_Y$$

$$wb_X \geq wb_Y$$

$$ha_X \geq ha_Y$$

$$hb_X \geq hb_Y$$

esto mismo se puede representar de la siguiente forma:

P13: $wa_X+wb_X \geq wa_Y+wb_Y$

P14: $\max(ha_X,hb_X) \geq \max(ha_Y,hb_Y)$

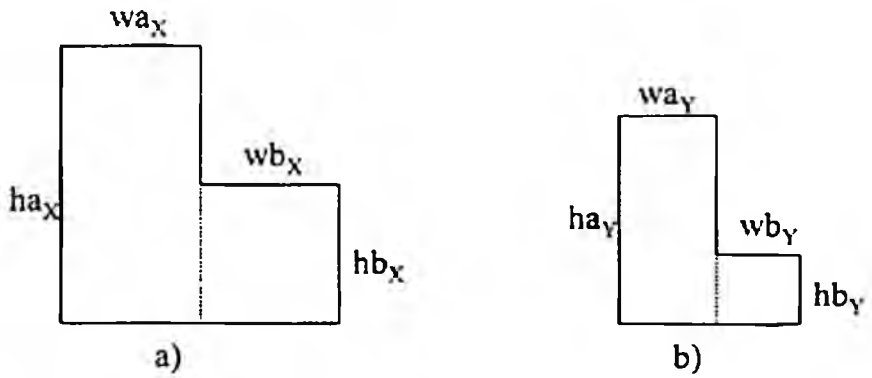


Figura 2-14 Implementación redundante de un bloque L2.

Sean $X=(wa_X,wb_X,wc_X,ha_X,hb_X,hc_X)$ e $Y=(wa_Y,wb_Y,wc_Y,ha_Y,hb_Y,hc_Y)$ dos implementaciones cualesquiera de un bloque L3 (Figura 2-15.a y Figura 2-15.b). Se dice que la implementación X es redundante o que domina a la Y cuando se cumplen las propiedades siguientes:

$wa_X \geq wa_Y$

$wb_X \geq wb_Y$

$wc_X \geq wc_Y$

$ha_X \geq ha_Y$

$hb_X \geq hb_Y$

$hc_X \geq hc_Y$

esto mismo se puede representar de la siguiente forma:

P15: $\max(wa_X+wb_X,wc_X) \geq \max(wa_Y+wb_Y,wc_Y)$

P16: $\max(ha_X+hc_X,hb_X+hc_X) \geq \max(ha_Y+hc_Y,hb_Y+hc_Y)$

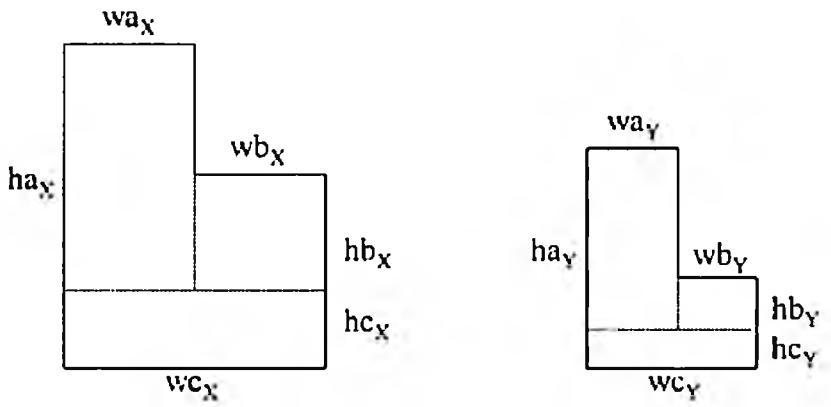


Figura 2-15 Implementación redundante de un bloque L3.

Sean $X=(wd_X,we_X,hd_X,he_X)$ e $Y=(wd_Y,we_Y,hd_Y,he_Y)$ dos implementaciones cualesquiera de un bloque 7 (Figura 2-16.a y Figura 2-16.b). Se dice que la implementación X es redundante o que domina a la Y cuando se cumplen las propiedades siguientes:

$wd_X \geq wd_Y$

$we_X \geq we_Y$

$hd_X \geq hd_Y$

$he_X \geq he_Y$

esto mismo se puede representar de la siguiente forma:

P17: $\max(wd_X, we_X) \geq \max(wd_Y, we_Y)$

P18: $hd_X + he_X \geq hd_Y + he_Y$

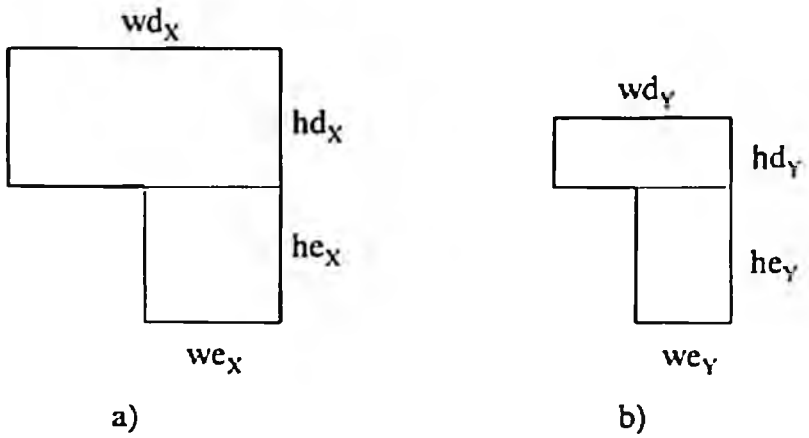


Figura 2-16 Implementación redundante de un bloque 7.

Sean $X=(w_X,h_X)$ e $Y=(w_Y,h_Y)$ dos implementaciones cualesquiera de un bloque rectangular (Figura 2-17.a y Figura 2-17.b). Se dice que la implementación X es redundante o que domina a la Y cuando se cumplen las propiedades siguientes:

P19: $w_X \geq w_Y$

P20: $h_X \geq h_Y$

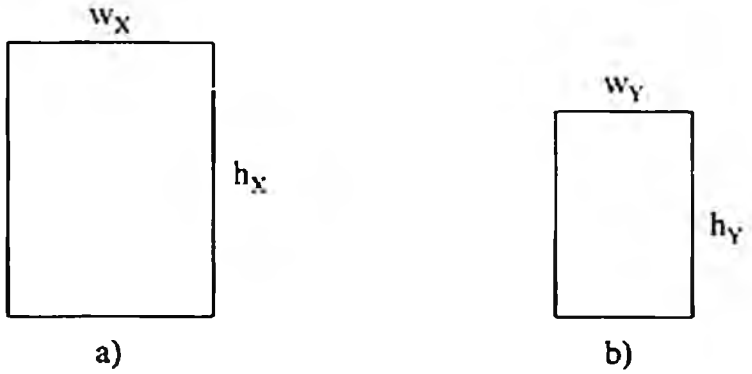


Figura 2-17 Implementación redundante de un bloque rectangular.

Las implementaciones redundantes con zonas desaprovechadas de un bloque no son relevantes para la obtención del área del floorplan óptimo ya que las áreas finales obtenidas son siempre mayores. Por tanto, dichas implementaciones deberán ser eliminadas lo antes posible, para evitar acarrear informaciones irrelevantes en futuras combinaciones con otros bloques o módulos, y perder tiempo en el proceso de obtención de soluciones óptimas.

2.1.10 Listas no Redundantes

Una lista-R no redundante o irreducible, es una lista-R que contiene sólo aquellas implementaciones de bloques rectangulares o módulos que no dominan. De manera similar, una lista-L2, una lista-L3 y una lista-7 irreducibles son listas que sólo contienen aquellas implementaciones que no dominan.

Los algoritmos que trabajan con listas, almacenan el conjunto de todas las implementaciones no redundantes de un bloque rectangular en una lista-R irreducible, mientras que todas las implementaciones no redundantes de un bloque L2, las almacenan en una lista-L2 irreducible. Cuando los floorplans son complejos las implementaciones de los bloques están agrupadas en listas que contienen a su vez un conjunto de sub-listas. Esto mismo se puede aplicar a bloques L3 o 7.

Por ejemplo, las listas-R, L2, L3 y 7 siguientes constan, a su vez, de dos sub-listas representadas de la siguiente forma (Figura 2-18):

Lista-R: {sub-lista1,sub-lista2}={{(1,2),(2,1)},{(1,4),(2,2),(4,1)}}

Lista-L2: {sub-lista1,sub-lista2}={{(1,1,4,3),(2,1,3,2),(4,2,2,1)},{(2,2,3,2),(3,2,4,1)}}

Lista-L3: {sub-lista1,sub-lista2}={{(1,1,2,3,2,3),(3,4,6,2,1,2)},{(4,1,5,2,1,1)}}

Lista-7: {sub-lista1,sub-lista2}={{(1,2,3,4),(2,3,2,3)},{(2,4,3,2),(4,5,1,2)}}

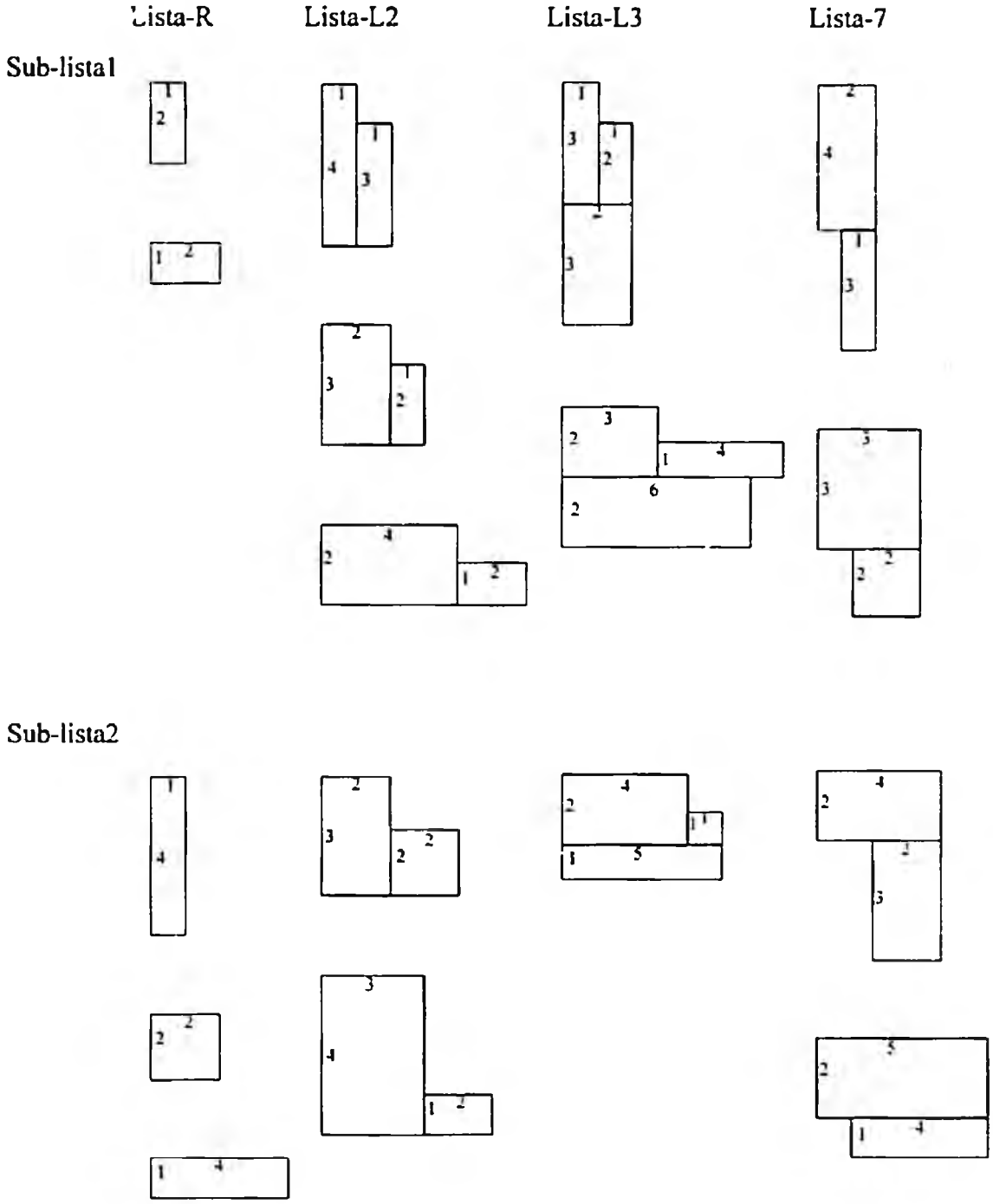


Figura 2-18 Implementaciones no redundantes contenidas en las distintas sub-listas.

2.2 PROBLEMÁTICA DEL DISEÑO DEL FLOORPLAN

Diseñar un floorplan, implica determinar la ubicación de los módulos y/o bloques que lo componen. Dado que éstos pueden tener múltiples implementaciones (formas y/o tamaños), la elección de las idóneas resulta muy laboriosa.

El algoritmo ideal que resuelva este problema, deberá cumplir los siguientes objetivos:

- Minimizar el área total.
- Minimizar el tiempo de proceso en la obtención del floorplan óptimo.
- Facilitar el conexionado posterior.
- Reducir los retardos de la señal.

Sin embargo, algunos de estos objetivos están en contradicción y por tanto, es muy difícil la consecución de todos ellos. Por esta razón, los algoritmos desarrollados hasta el momento, intentan lograr un equilibrio favoreciendo unos objetivos sobre otros.

2.2.1 Método de Descomposición "Top-Down"

El método de descomposición jerárquico "top-down" es el más habitual en la resolución de floorplans. La aplicación de este método consiste en ir dividiendo sucesivamente en bloques lo más homogéneos posibles, de forma que cada uno, es un floorplan más simple. Este proceso se aplica de forma recursiva hasta llegar al nivel de módulo (Figura 2-19).

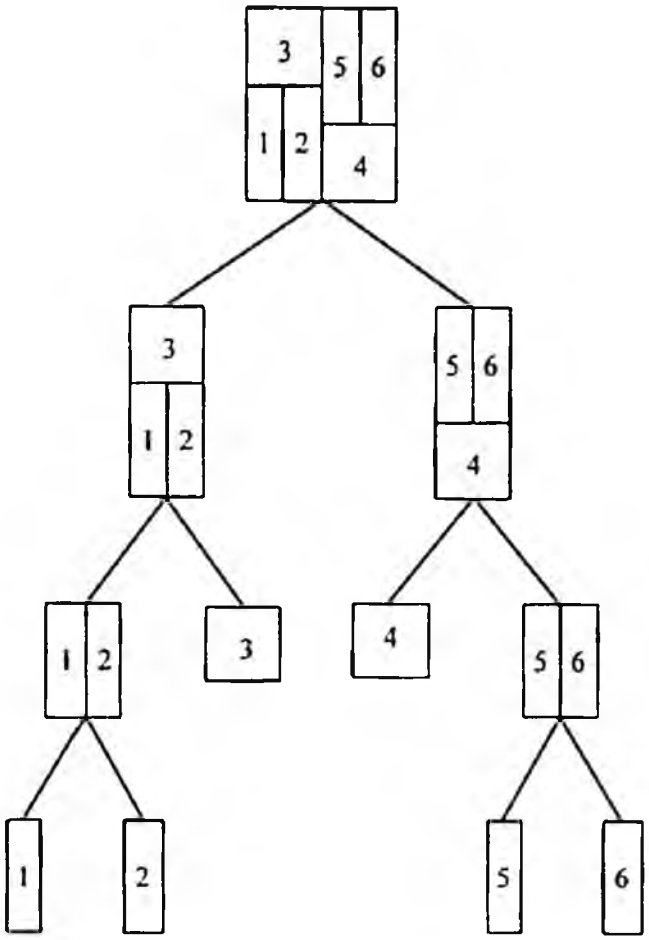


Figura 2-19 Descomposición top-down.

El costo de descomposición del floorplan es difícil de estimar, ya que la organización interna de los bloques no es conocida a priori.

La forma más habitual de dividir un floorplan es la bipartición. Este método tiene el inconveniente de poder dar lugar a dos particiones de tamaños totalmente dispares. Sin embargo, dicha problemática no surge bajo ciertas restricciones ([KERN70], [FIDU82] y [WEI91]).

2.2.2 Método de Síntesis "Bottom-Up"

Lengauer [LENG90] así como Pedram y Preas [PEDR90] establecieron las bases para la aplicación del método de síntesis "bottom-up" para la resolución de floorplans. En este método los módulos que compondrán el floorplan se van agrupando de forma que las conexiones entre ellos, sean óptimas. En la figura 2-20 se muestra un ejemplo de síntesis.

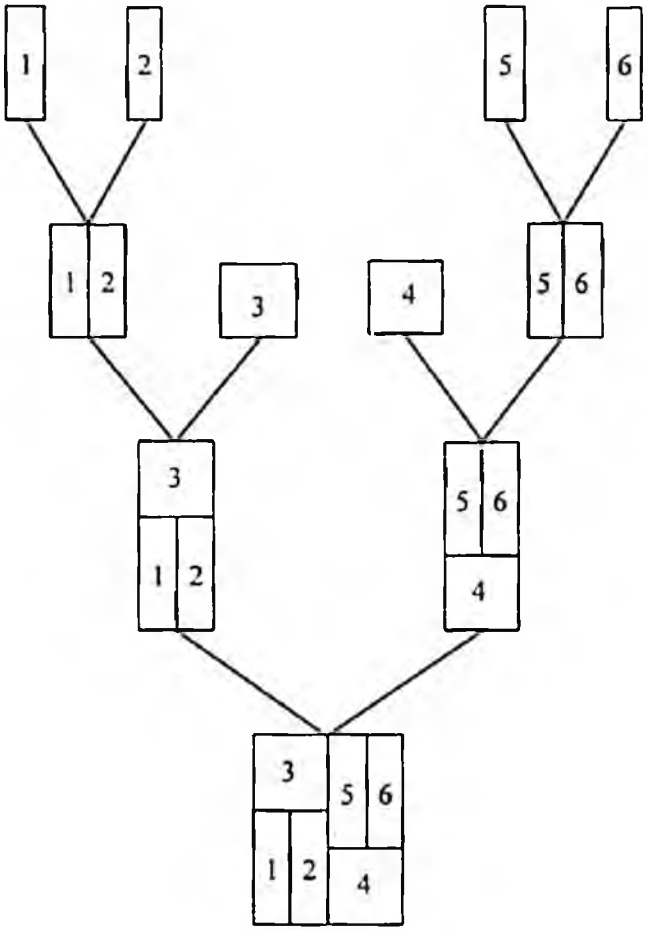


Figura 2-20 Síntesis bottom-up.

Al mismo tiempo que se realiza dicho agrupamiento se calcula la dimensión de cada bloque resultante, con lo que la estimación del área se simplifica.

Cuando los módulos son agrupados en bloques, los bordes correspondientes a cada uno indican la conectividad entre ellos; es decir, el número de posibles uniones. Evaluando dicha conectividad se obtiene el coste de conexión.

La agrupación de módulos y bloques se realiza por grado de conectividad, creando floorplans opcionales.

2.2.3 Método Analítico

Los métodos analíticos se basan en la programación matemática o de métodos numéricos [SUTA91] y entre sus aplicaciones, está la resolución de la problemática del floorplan [SCHR86]. El proceso a realizar, consiste en tomar los módulos y/o bloques de dos en dos minimizando el espacio desperdiciado y maximizando su conectividad con otros bloques.

Para la optimización de la interconexión y del área del floorplan, se introducen una serie de limitaciones en los módulos a la hora de obtener el floorplan, como por ejemplo: la imposibilidad de solapamiento, la homogeneidad, etc.

Entre sus inconvenientes destacan la exactitud que deben tener las formulaciones realizadas así como las limitaciones impuestas. Por ello, el coste real es elevado y las soluciones producidas no son muy eficaces para todo tipo de floorplans, incluidos los de tamaño moderado. Sin embargo, presenta la ventaja de que el floorplan no exige obligatoriamente usar un método de diseño jerárquico.

2.2.4 Método Estadístico

El método estadístico es especialmente útil cuando las especificaciones iniciales son aproximadas y consiste en realizar sucesivas iteraciones hasta lograr la solución óptima deseada. Así, su principal ventaja es la flexibilidad inicial en cuanto a la ubicación de los módulos, pero la razón de su popularidad se debe en gran parte a la carencia de unos buenos métodos algorítmicos específicos [YEAP93].

Por otro lado, el floorplan resultante depende en gran medida, de los parámetros estocásticos de control usados como guía en los procesos aleatorios. A menudo, no hay un conjunto sencillo y adecuado de parámetros a utilizar, por lo que la solución lograda es inestable. Además, en las aproximaciones de tipo estadístico es difícil estimar el coste final de floorplan completo; para ello se requirieren muchos recursos.

En cuanto al método estadístico se puede destacar la utilización de dos técnicas muy importantes: recocido simulado [KIRK83] y algoritmo genético [HOLL73]. Actualmente, se utilizan estas técnicas para la ubicación de cada uno de los módulos en el floorplan ([COHO86], [SARR89] y [SHAH90]).

2.2.5 Método por Grafo Dual

Kozminski y Kinnen [KOZM84] así como Lai y Leinwand [LAI88], propusieron un nuevo método basado en el concepto de grafo dual en el cual se describe las relaciones de adyacencia existentes entre los distintos módulos que componen el correspondiente floorplan.

En la figura 2-21 se muestra el proceso de transformación del circuito en el grafo dual correspondiente al floorplan que lo implementa, paso a paso:

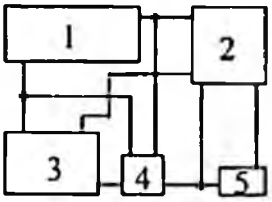
- Partiendo de la implementación de un circuito (Figura 2-21.a) se convierte ésta en un hipergráfico donde los vértices corresponden a módulos y los hiperbordes al conjunto de redes de terminales que conectan los vértices (Figura 2-21.b).

- A continuación, se calcula a partir de los hiperbordes un gráfico de bordes (Figura 2-21.c), al que se añaden los cuatro vértices que son imprescindibles para la consecución de un floorplan rectangular (Figura 2-21.d).

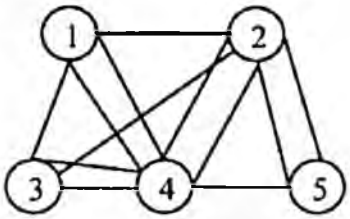
- Por último, se asigna el área rectangular del chip que especifica los límites de los módulos y los tamaños de cada una de sus implementaciones. Una vez terminado este proceso se obtiene el floorplan (Figura 2-21.e).

Hay que tener en cuenta que, si hay dos vértices adyacentes en el grafo dual, las células correspondientes serán también adyacentes en el floorplan y viceversa y por consiguiente, la compactación, será mayor.

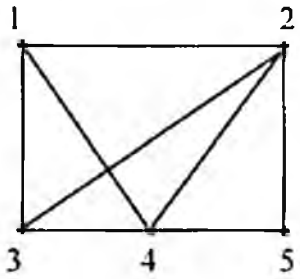
Este método no tiene una gran aceptación en la industria ya que es muy laborioso el desarrollo del gráfico correspondiente. Así mismo, es difícil evaluar el coste final del floorplan conociendo tan sólo la construcción parcial del mismo.



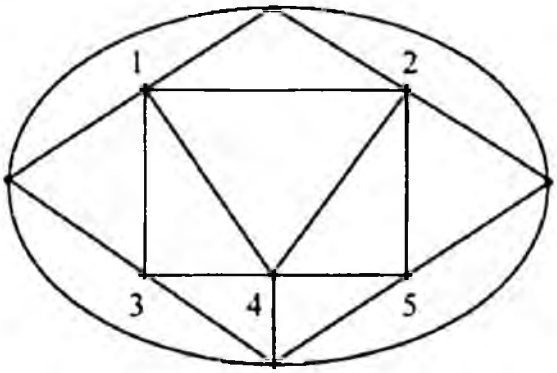
a)



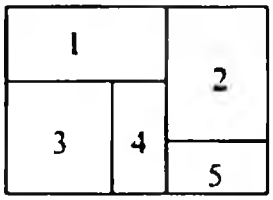
b)



c)



d)



e)

Figura 2-21 Método por grafo dual.

2.3 ORDEN DOS

Al aplicar sobre un floorplan un método de descomposición top-down, si éste se puede particionar bien mediante un corte horizontal ⊖, o bien mediante un corte vertical ⓘ, se dice que es un floorplan de orden dos. El resultado de aplicar una de estas particiones al floorplan es dos rectángulos hijos básicos. En la figura 2-22 se representa con un árbol los sucesivos cortes horizontales y verticales aplicados al floorplan V1.

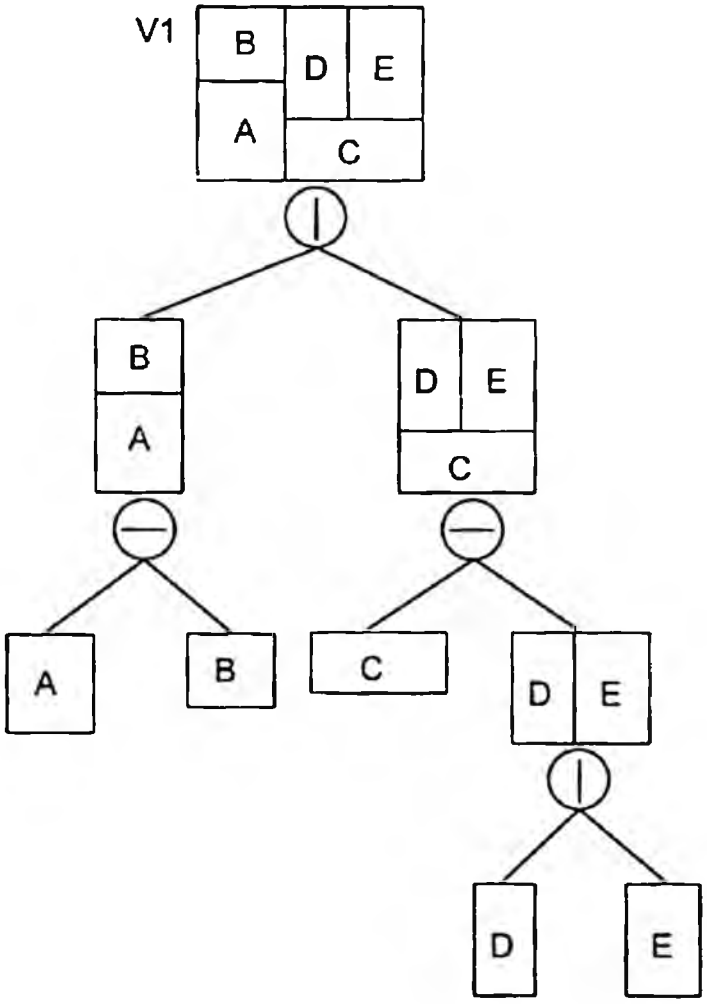


Figura 2-22 Floorplan de orden dos con cinco módulos.

De la misma manera, si al aplicar sobre dos bloques el método de síntesis bottom-up mediante una unión vertical ⓘ u horizontal ⊖, se obtiene un floorplan, éste se dice que es de orden dos.

2.3.1 Unión Vertical

El algoritmo UNIÓN-V combina las implementaciones de dos módulos o rectángulos básicos, generando un nuevo rectángulo (Figura 2-23), cuya altura es la del mayor de ellos max(ha,hb) y la anchura es la suma de las anchuras de ambos wa+wb.

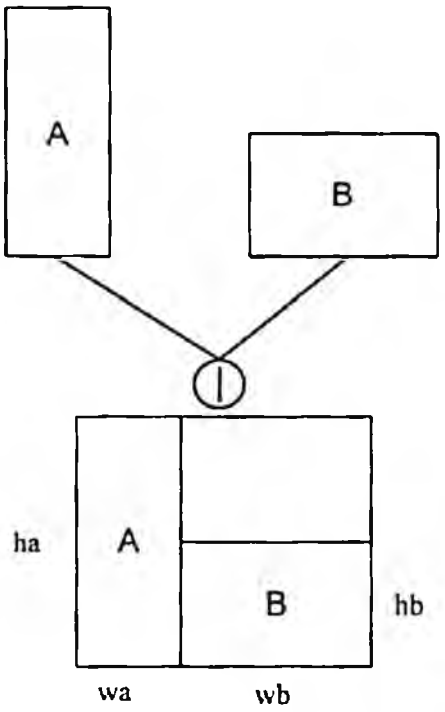


Figura 2-23 Unión vertical.

Sean dos módulos, A y B, cuyas implementaciones están recogidas en las listas $L_A=\{(wa_i,ha_i) \mid 1\le i\le m\}$ y $L_B=\{(wb_j,hb_j) \mid 1\le j\le k\}$ respectivamente. Las m

implementaciones de la lista L_A y las k implementaciones de L_B están clasificadas en orden creciente de anchura y en orden decreciente de altura.

Por tanto, si los pares (wa,ha) y (wb,hb) denotan respectivamente la anchura y la altura de los módulos A y B, el bloque resultante AB estará definido por (wa+wb,max(ha,hb)).

El algoritmo UNIÓN-V genera una lista L_{AB} mediante la comparación de las alturas de las implementaciones contenidas en L_A y L_B (Figura 2-24):

- Si la altura del par de L_A es mayor que la de L_B combinará el siguiente elemento de L_A con el mismo par de L_B.
- Si la altura del par de L_A es menor que la de L_B unirá el siguiente elemento de L_B con el mismo par de L_A.
- Por el contrario, si las alturas de los pares de L_A y de L_B son iguales agrupará el siguiente elemento de L_A con el siguiente de L_B.

```
ALGORITMO UNIÓN-V
    CREAR una lista vacía L_AB
    i=1
    j=1
    WHILE i≤m AND j≤k DO
        (wa_i,ha_i)= es el elemento i de L_A
        (wb_j,hb_j)= es el elemento j de L_B
        INSERTAR (wa_i+wb_j,max(ha_i,hb_j)) en L_AB
        IF (ha_i>hb_j) THEN i=i+1
                       ELSE IF (ha_i<hb_j) THEN j=j+1
                                           ELSE  i=i+1
                                                 j=j+1
                            ENDIF
        ENDIF
    END
```

Figura 2-24 Algoritmo Unión-V.

La lista generada por este algoritmo no contiene pares redundantes y está ordenada en creciente por anchura y en decreciente por altura manteniendo cada par de dicha lista los punteros de los que se obtuvo.

En el peor caso, el orden de operaciones para la obtención de los n módulos, es de $O(n^2)$ ya que se generarán $O(n)$ pares en $O(n)$ niveles.

2.3.2 Unión Horizontal

El algoritmo UNIÓN-H es similar al UNIÓN-V salvo que el rectángulo resultante tiene la anchura del módulo mayor $max(wa,wb)$ y la altura es la suma de las alturas de ambos $ha+hb$ (Figura 2-25).

Sean dos módulos, A y B, cuyas implementaciones están recogidas en las listas $L_A=\{(wa_i,ha_i) \mid 1\leq i\leq m\}$ y $L_B=\{(wb_j,hb_j) \mid 1\leq j\leq k\}$ respectivamente. Las m implementaciones de la lista L_A y las k implementaciones de la lista L_B, están ordenadas en orden creciente de anchura y en orden decreciente de altura.

Por lo tanto si los pares (wa,ha) y (wb,hb) denotan respectivamente la anchura y la altura de los módulos A y B, el bloque resultante AB estará definido por (max(wa,wb),ha+hb).

El algoritmo UNIÓN-H genera una lista L_{AB} mediante la comparación de las anchuras de las implementaciones contenidas en L_A y L_B (Figura 2-26):

- Si la anchura del par de L_A es mayor que la de L_B combinará el siguiente elemento de L_A con el mismo par de L_B.
- Si la anchura del par de L_A es menor que la de L_B unirá el siguiente elemento de L_B con el mismo par de L_A.

- Por el contrario, si las anchuras de los pares de L_A y de L_B son iguales agrupará el siguiente elemento de L_A con el siguiente de L_B.

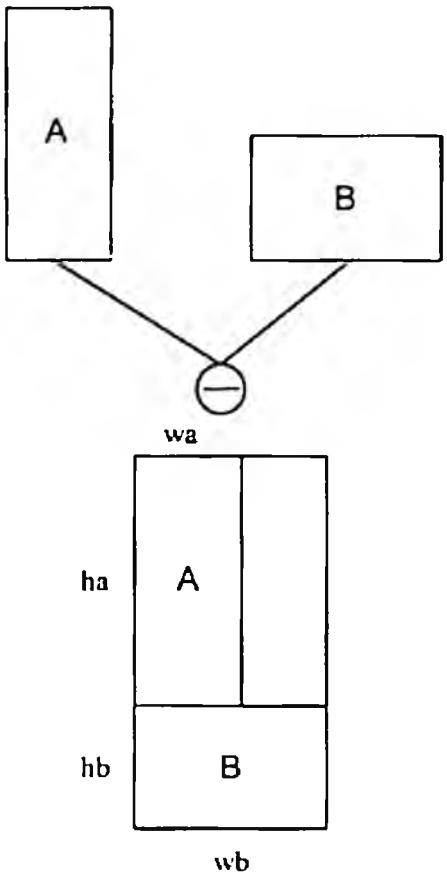


Figura 2-25 Unión horizontal.

La lista generada por este algoritmo no contiene pares redundantes y está ordenada en creciente por anchura y en decreciente por altura manteniendo cada par de dicha lista los punteros de los que se obtuvo.

En el peor caso, el orden de operaciones para la obtención de los n módulos, es de $O(n^2)$ ya que se generarán $O(n)$ pares en $O(n)$ niveles.

```
ALGORITMO UNIÓN-H
    CREAR una lista vacía L_AB
    i=1
    j=1
    WHILE i≤m AND j≤k DO
        (wa_i,ha_i)= es el elemento i de L_A
        (wb_j,hb_j)= es el elemento j de L_B
        INSERTAR (max(wa_i,wb_j),ha_i+hb_j) en L_AB
        IF (wa_i>wb_j) THEN i=i+1
                       ELSE IF (wa_i<wb_j) THEN j=j+1
                                           ELSE i=i+1
                                                j=j+1
                            ENDIF
        ENDIF
    END
```

Figura 2-26 Algoritmo Unión-H.

2.4 ORDEN CINCO

Al aplicar sobre un floorplan un método de descomposición top-down, si éste no se puede particionar con un corte horizontal ⊖, o con un corte vertical ①, pero sí mediante un corte Z Ⓩ (combinación de uno vertical y dos horizontales, o bien de uno horizontal y dos verticales), se dice que es un floorplan de orden cinco.

Una vez efectuados dichos cortes, se obtienen dos floorplans hijo, uno de los cuales es un bloque L3 y otro un bloque 7 (Figura 2-27), que pueden a su vez, particionarse mediante cortes horizontales ⊖, verticales ① y/o Z Ⓩ, hasta la consecución de los módulos.

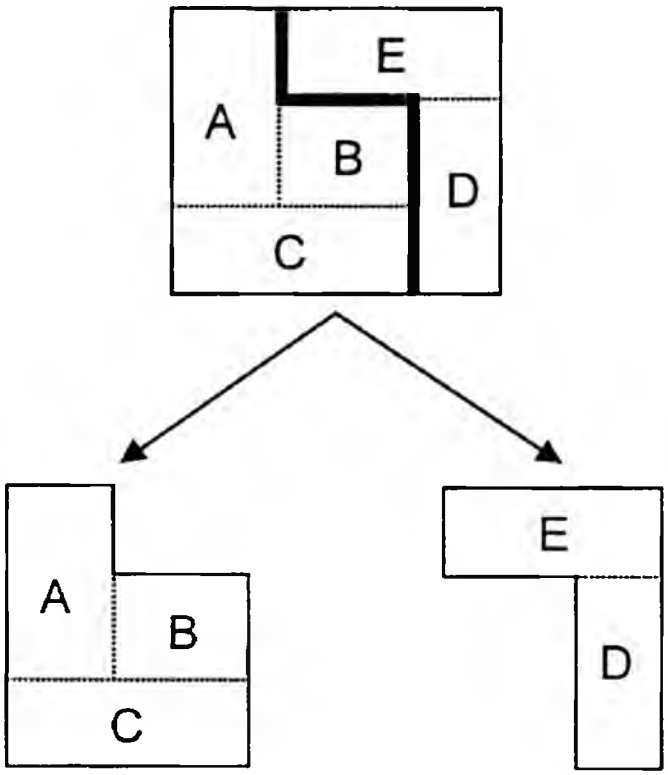
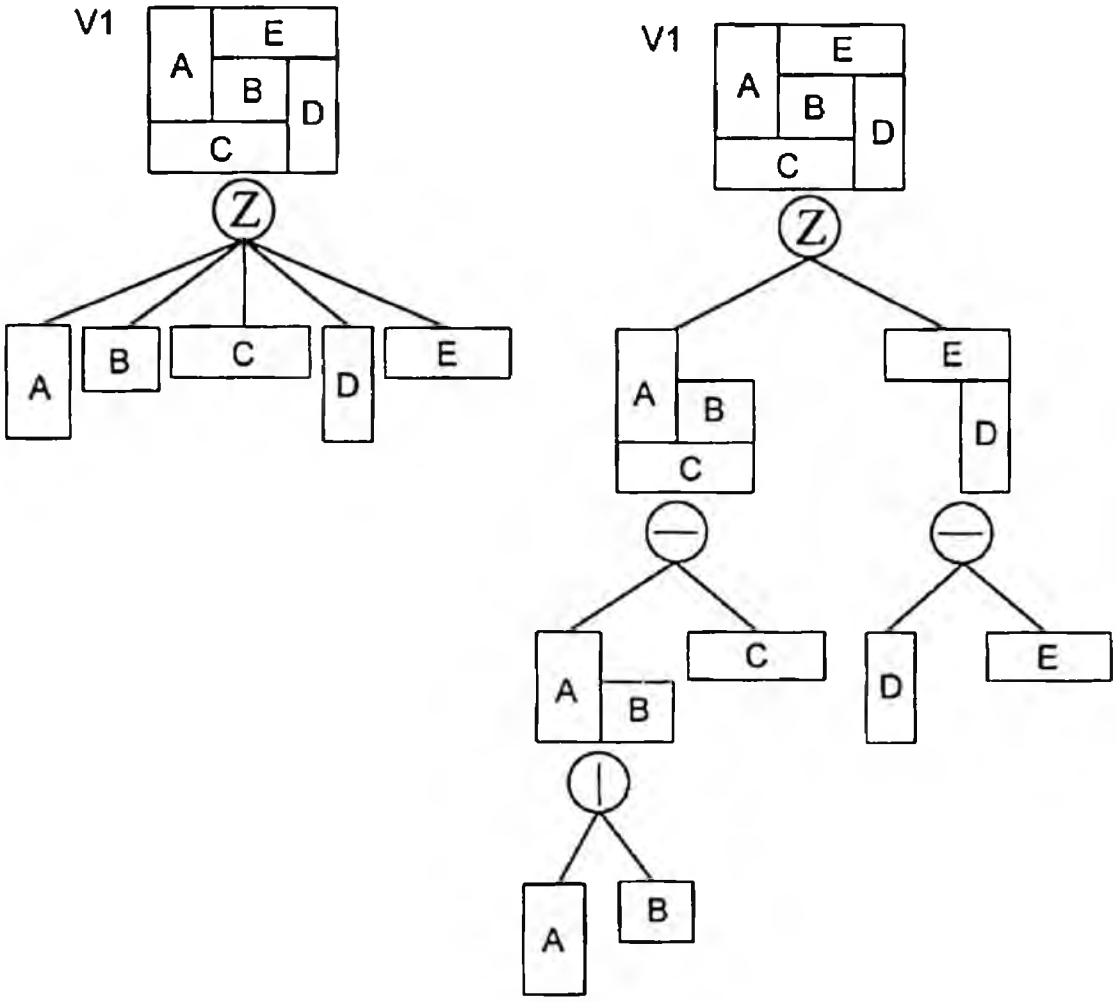


Figura 2-27 Descomposición de un floorplan de orden cinco en bloques L3 y 7.

Se pueden plantear dos formas de representar el árbol de descomposición al aplicar un corte Z ⓩ sobre el floorplan V1 (Figura 2-28). En la figura 2-28.b se puede observar el resultado de dicho corte así como de aplicar sucesivos cortes horizontales ⊖ y verticales ⓘ hasta la obtención de los bloques A, B, C, D y E componentes.

También se pueden plantear floorplans multimodulares (Figura 2-29). Los bloques A, B, C, D y E obtenidos en el proceso de descomposición, pueden ser módulos o a su vez bloques de orden dos o de orden cinco.



a) *b)*

Figura 2-28 Dos formas de representar el árbol de descomposición top-down de un mismo floorplan de orden cinco.

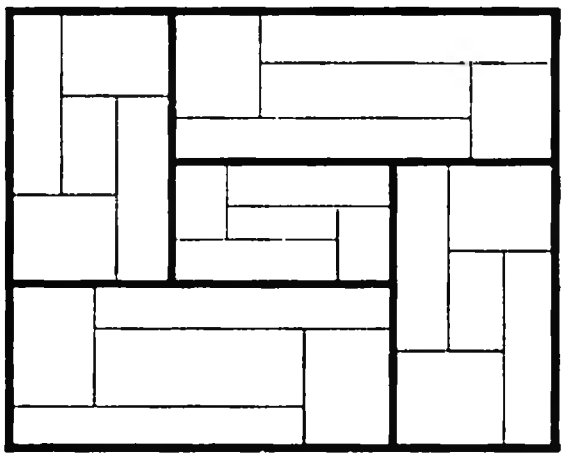


Figura 2-29 Floorplan multimodular de orden cinco.

Si al aplicar sobre un bloque L3 y un bloque 7 el método de síntesis bottom-up, éstos no se pueden combinar con una unión horizontal ⊖, o con una unión vertical ①, pero sí mediante una unión Z ② (una vertical y dos horizontales o bien de una horizontal y dos verticales), se dice que el bloque o el floorplan resultante es de orden cinco o wheel (Figura 2-30).

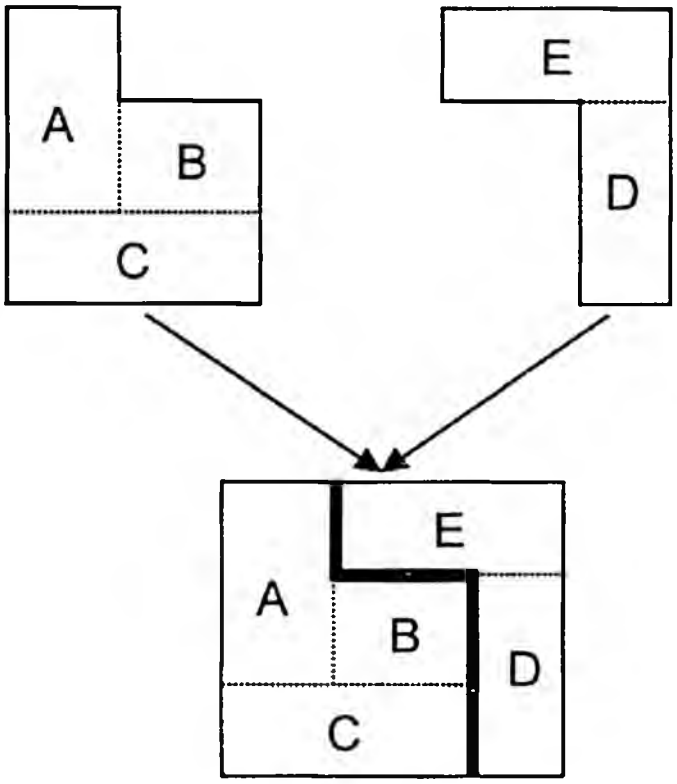
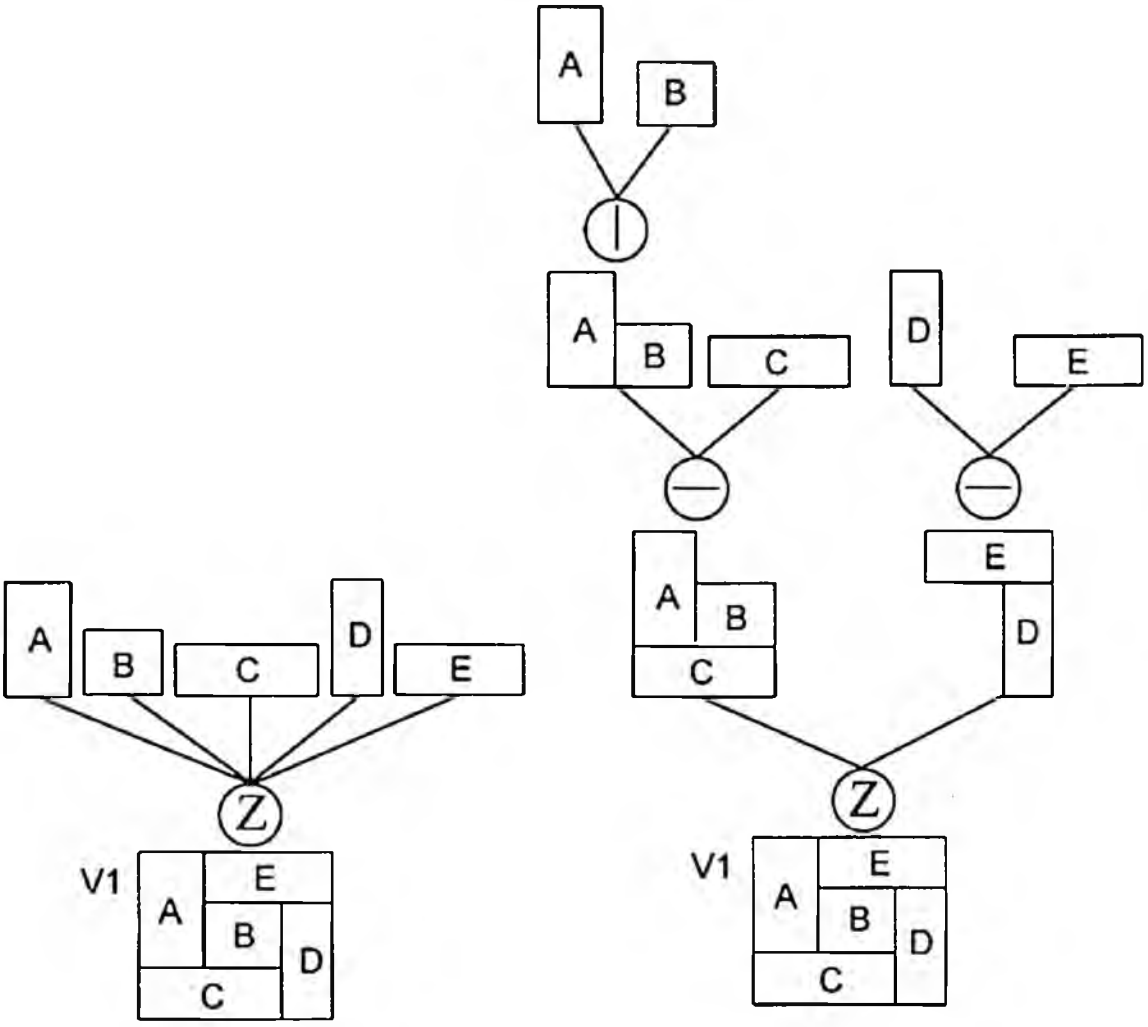


Figura 2-30 Floorplan de orden cinco mediante la unión de bloques L3 y 7.

La obtención de un mismo floorplan de orden cinco se puede representar mediante dos árboles diferentes (Figura 2-31). En la figura 2-31.b se puede observar el resultado de aplicar sucesivas uniones verticales ① y horizontales ⊖ así como el de una unión Z ② para la obtención del floorplan resultante V2 de orden cinco.

Los bloques A, B, C, D y E de los que partimos pueden ser a su vez módulos o bloques de orden dos o de orden cinco.



a) *b)*

Figura 2-31 Dos formas de representar el árbol de síntesis bottom-up de un mismo floorplan de orden cinco.

2.5 FLOORPLANS JERÁRQUICOS

Un floorplan jerárquico es aquel que puede descomponerse mediante cortes Z Ⓩ, horizontales ⊖ y/o verticales ⓘ en bloques de tipo L2, L3, 7 o rectangulares.

Así mismo, se consideran también floorplans jerárquicos aquellos que permiten la síntesis de sus bloques mediante uniones Z Ⓩ, horizontales ⊖ y/o verticales ⓘ.

Todo floorplan jerárquico, tanto si se efectúa un proceso de descomposición como un proceso de síntesis, se puede representar mediante un árbol binario en el que cada nodo puede corresponder a un módulo o a un bloque.

Cuando un floorplan jerárquico contiene sólo floorplans de orden dos se denomina floorplan particionado, si sólo contiene floorplans de orden cinco se denomina no particionado y si tiene a la vez de orden dos y de orden cinco se dice que el floorplan es general.

2.5.1 Floorplan Particionado

Si al aplicar sobre un floorplan un proceso de descomposición top-down éste se puede particionar en dos bloques rectangulares y así sucesivamente hasta la consecución de los módulos a través de cortes horizontales ⊖ y/o verticales ⓘ sucesivos, entonces podemos afirmar que se trata de un floorplan particionado. Todo floorplan particionado es de orden dos y por tanto, se puede representar por un árbol binario llamado árbol particionado (Figura 2-32).

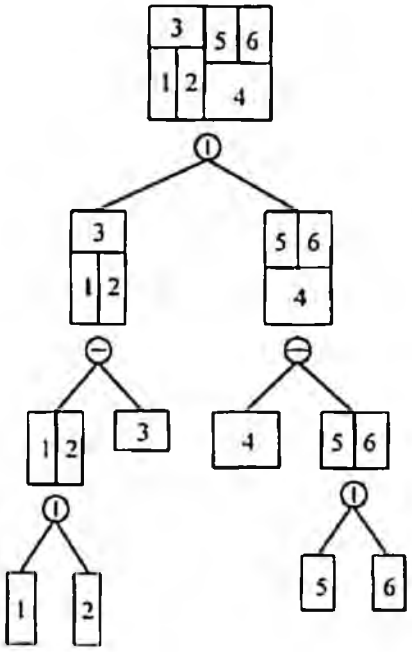


Figura 2-32 Árbol de un floorplan particionado utilizando el método top-down.

Por otro lado, un mismo floorplan particionado puede tener más de un árbol particionado. La figura 2-33 muestra diferentes árboles particionados que representan el mismo floorplan.

Si al aplicar sobre los módulos el método de síntesis bottom-up, éstos se pueden unir en bloques rectangulares y así sucesivamente hasta la consecución del floorplan a través de uniones horizontales ⊖ o verticales ⓘ sucesivas, entonces podemos afirmar que el floorplan obtenido es particionado y por consiguiente de orden dos.

En la figura 2-34, al aplicar el método de síntesis bottom-up, en cada nodo interno del árbol aparece el símbolo ⊖ o ⓘ que indica la unión horizontal o vertical, a realizar en los bloques o módulos.

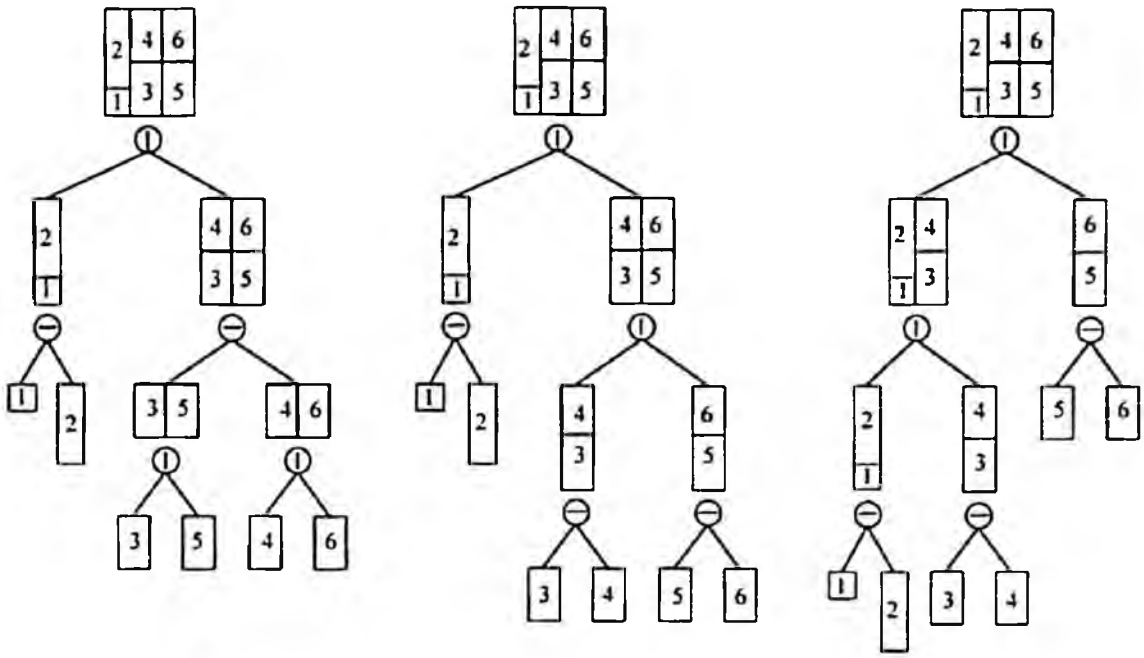


Figura 2-33 Árboles distintos para un mismo floorplan usando el método top-down.

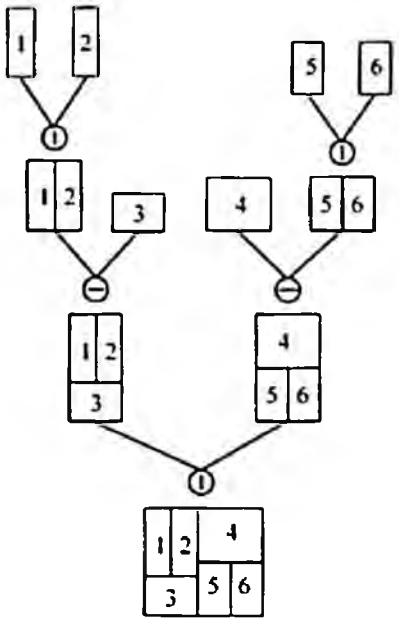


Figura 2-34 Árbol de un floorplan particionado utilizando el método bottom-up.

La figura 2-35 muestra diferentes árboles binarios particionados que representan el mismo floorplan obtenido al utilizar el método de síntesis bottom-up.

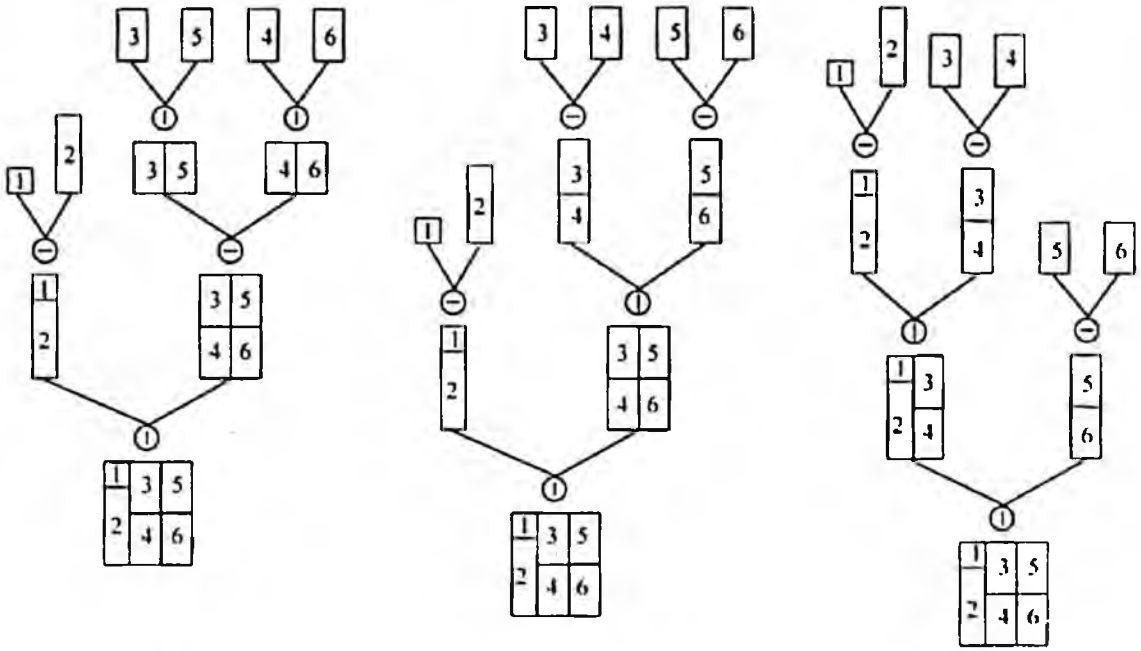


Figura 2-35 Árboles distintos para un mismo floorplan usando el método bottom-up.

2.5.2 Floorplan no Particionado

El floorplan no particionado se compone de n módulos que pueden ser obtenidos a través de un corte Z ② y de sucesivos cortes horizontales ⊖ y/o verticales ①.

Entre los floorplans no particionados hay un floorplan especial que consta sólo de cinco módulos denominado "wheel". Se caracteriza por tener una estructura tal que, por mucho que se varíe la ubicación de sus cinco módulos, sólo se puede conseguir una distribución imagen reflejada de la original (Figura 2-36).

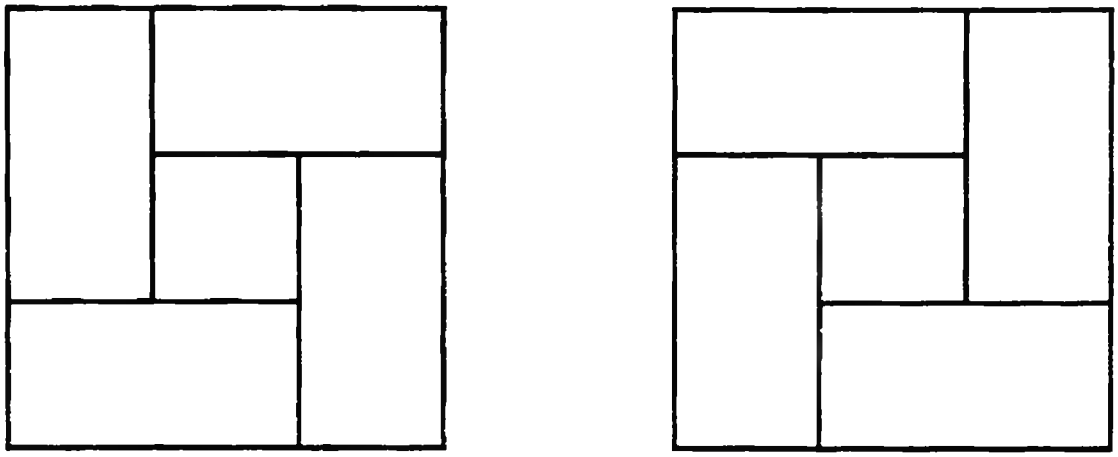


Figura 2-36 Wheel y su imagen reflejada.

Si al aplicar sobre él un proceso de descomposición top-down, éste se puede particionar en un primer momento mediante un corte Z ⓩ obteniéndose un bloque L3 y un bloque 7, y a su vez cada uno de éstos bloques pueden ser particionados hasta la consecución de los módulos a través de cortes horizontales ⊖ o verticales ①, entonces podemos afirmar que todo floorplan no particionado es de orden cinco.

Por otro lado, puede ocurrir que un wheel incluya a su vez otros wheels y/o estructuras particionadas. En la figura 2-37, podemos observar el árbol de un floorplan con doble estructura no particionada al aplicar el método de descomposición top-down.

En cada nodo interno del árbol representado, que corresponde a un bloque, aparece el símbolo ⊖, ① o ⓩ indicando respectivamente el corte horizontal, vertical o Z que se realiza.

El objetivo a conseguir al aplicar el método de síntesis bottom-up es el de obtener el floorplan no particionado mediante la unión de los módulos iniciales.

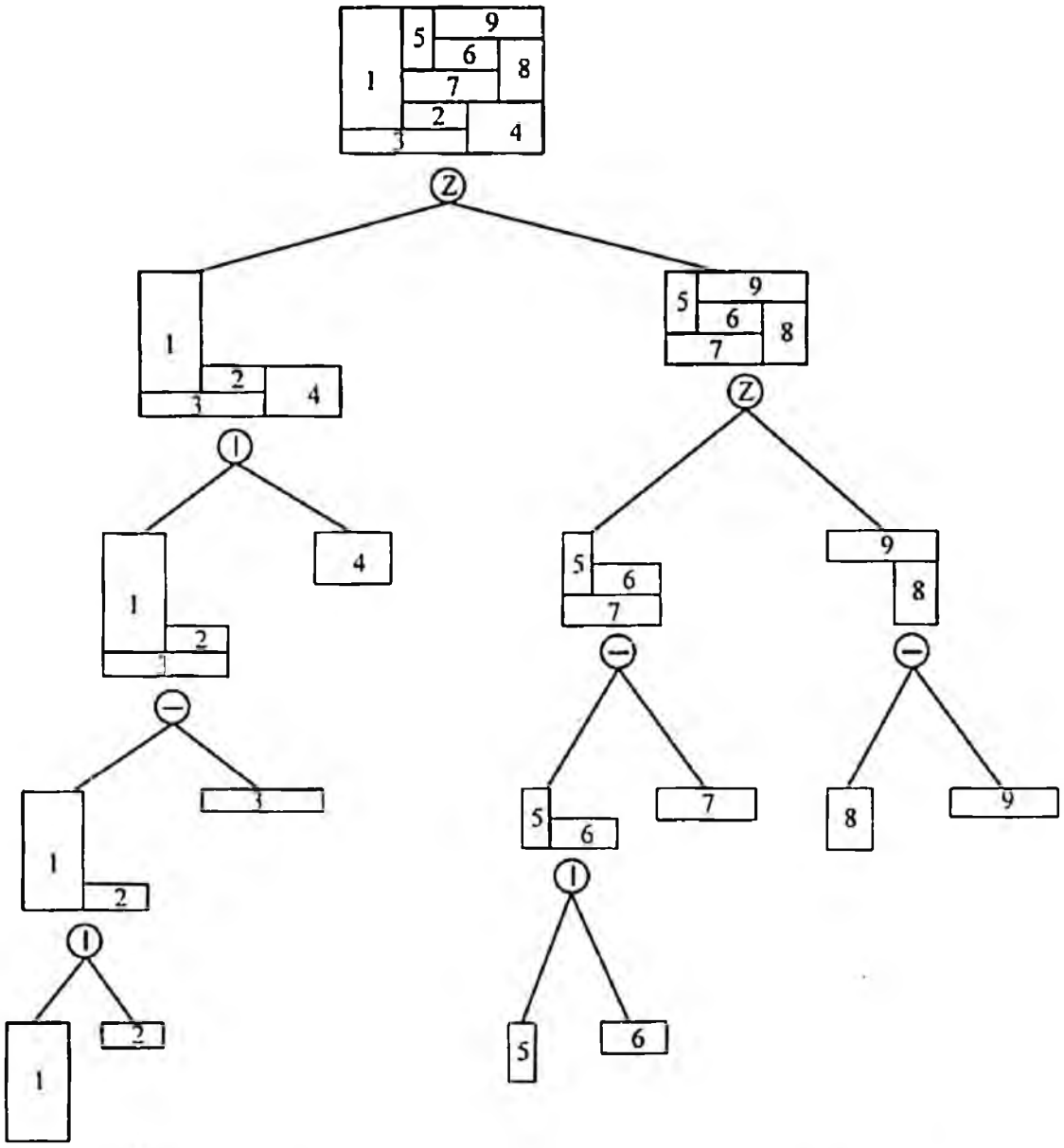


Figura 2-37 Floorplan con doble estructura no particionada por el método top-down.

Para que esto sea factible se irán efectuando uniones horizontales ⊖ o verticales ① entre los distintos módulos y bloques hasta lograr dos bloques: un bloque L3 y otro bloque 7. Una vez obtenidos estos dos bloques se realizarán con ellos una unión Z ② con la que se conseguirá un floorplan no particionado de orden cinco.

También en este caso, puede ocurrir que un wheel incluya a su vez otros wheels y/o estructuras particionadas. En la figura 2-38, podemos observar el árbol de un floorplan con doble estructura no particionada al aplicar el método de síntesis bottom-up.

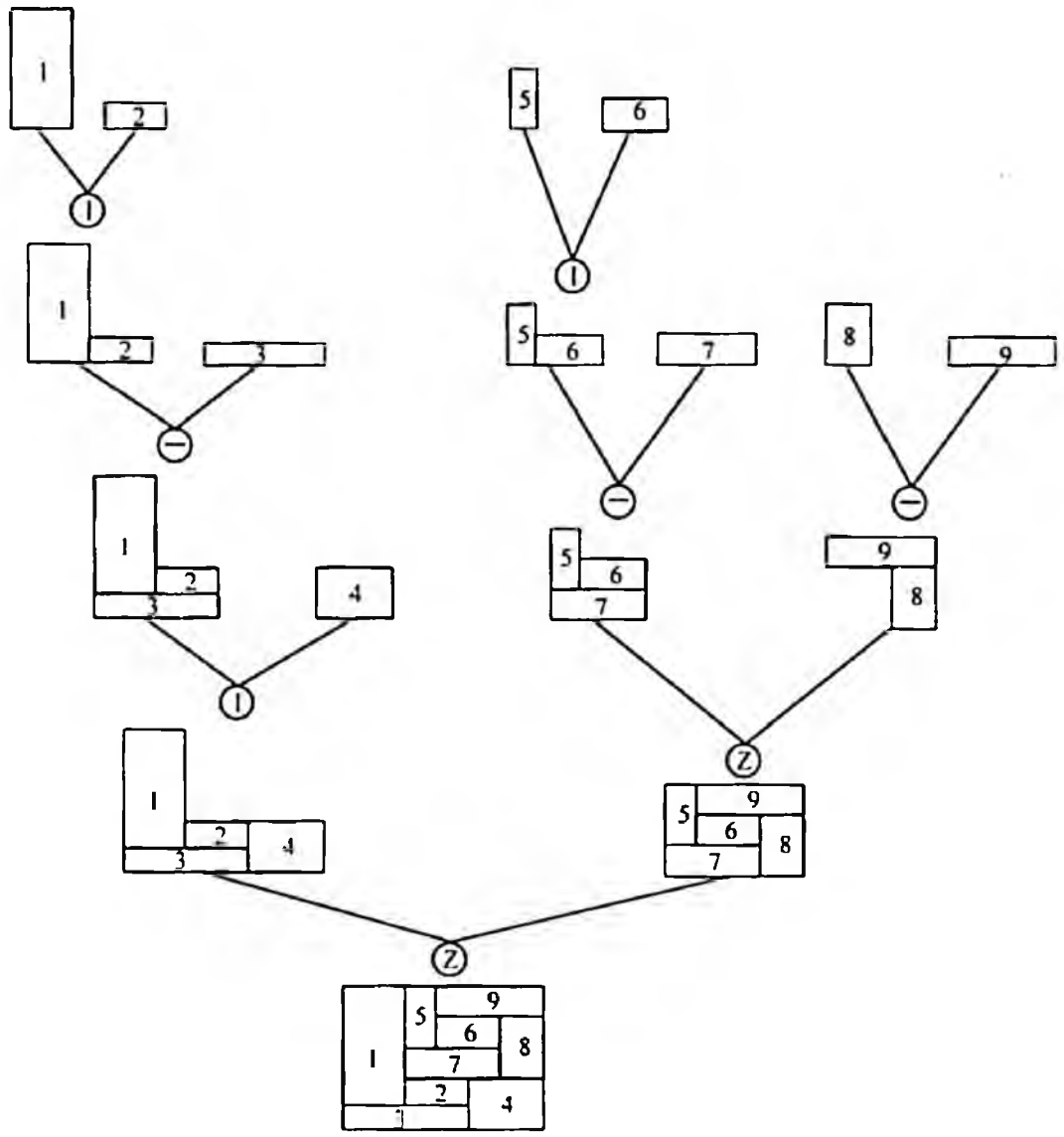


Figura 2-38 Floorplan con doble estructura no particionada por el método bottom-up.

En cada nodo interno del árbol representado, que corresponde a un bloque, aparece el símbolo ⊖, ① o Ⓩ indicando respectivamente la unión horizontal, vertical o Z que se realiza.

2.5.3 Floorplan General

Cuando un floorplan consta de floorplans particionados y no particionados se dice que es general.

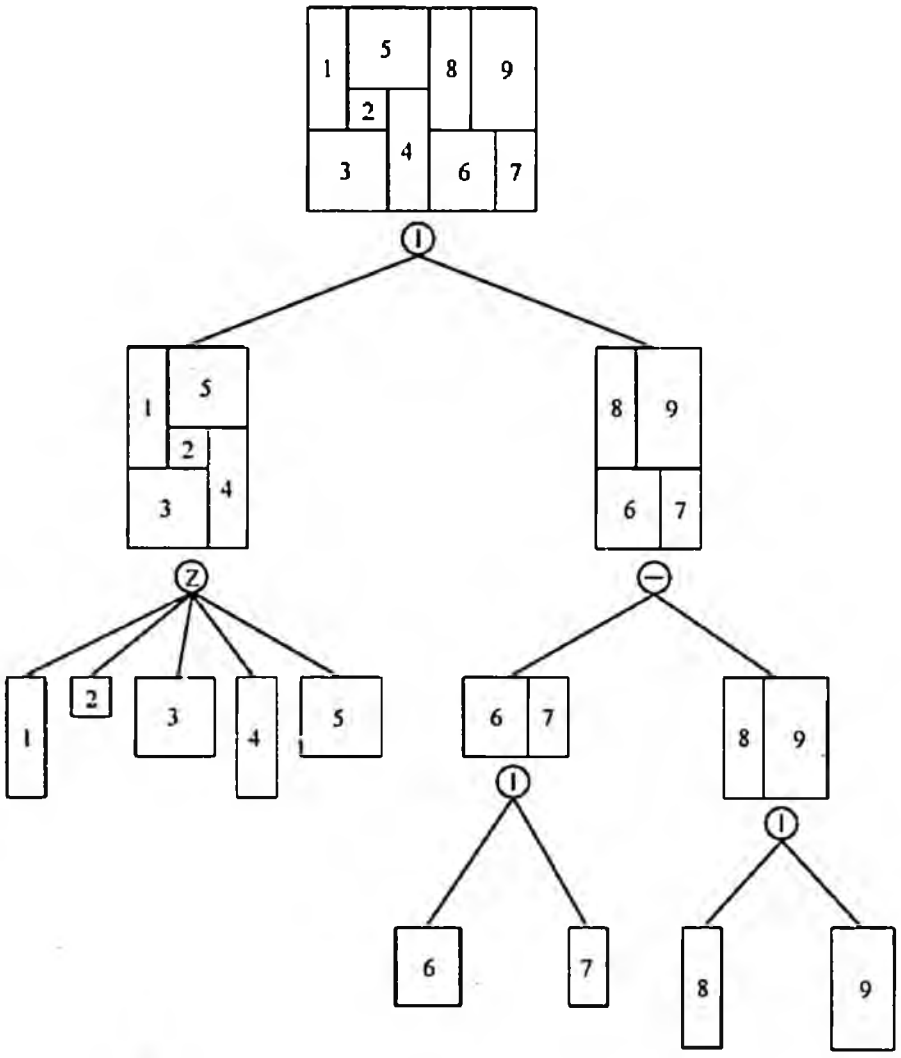


Figura 2-39 Árbol de un floorplan general utilizando el método top-down.

Si al aplicar sobre él un método de descomposición top-down nos encontramos con una estructura particionada, ésta la podemos particionar mediante sucesivos cortes horizontales ⊖ y/o verticales Ⓘ hasta la consecución de los módulos componentes; en caso contrario, cuando nos

encontremos con una estructura no particionada, ésta la podremos particionar mediante un corte Z Ⓩ (Figura 2-39).

Si al aplicar sobre los módulos de los que consta un floorplan general un método de síntesis bottom-up podemos conseguir mediante sucesivas uniones horizontales ⊖ y/o verticales Ⓘ una estructura particionada y una estructura no particionada, mediante una unión Z Ⓩ (Figura 2-40).

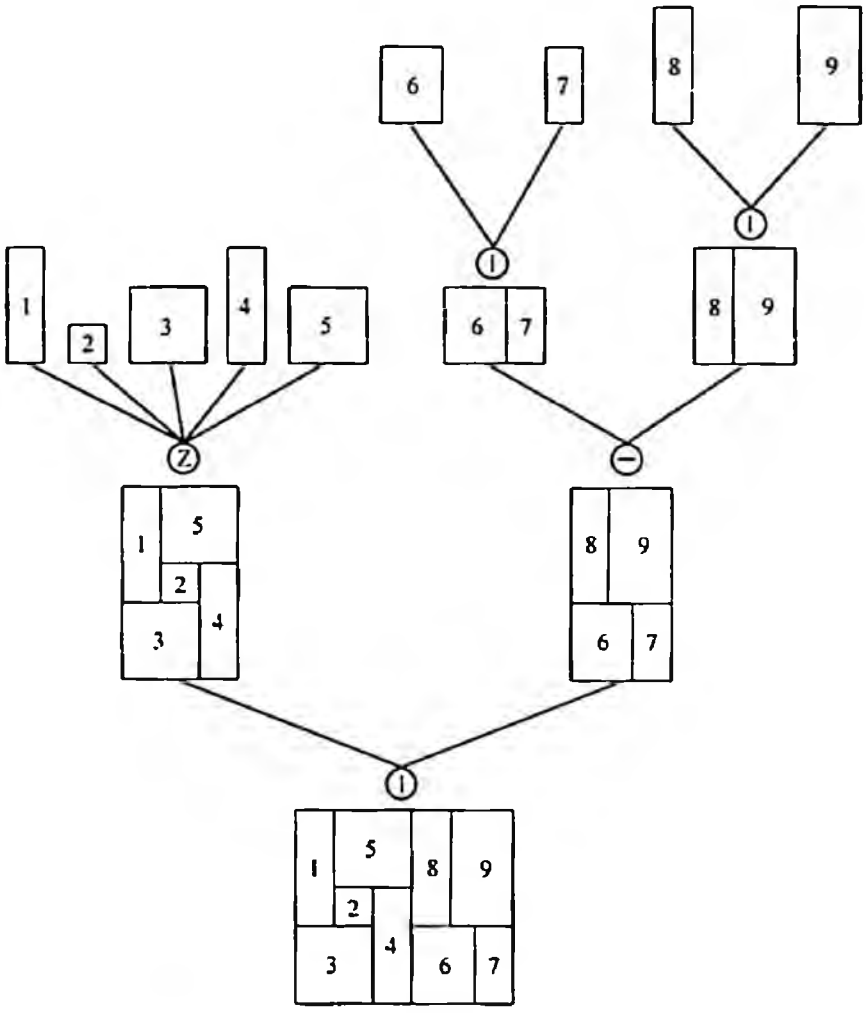


Figura 2-40 Árbol de un floorplan general utilizando el método bottom-up.

3. DESCRIPCIÓN DEL PROBLEMA Y SOLUCIONES EXISTENTES

3.1 PROBLEMÁTICA DEL FLOORPLAN

El floorplanning es un paso muy importante en el diseño físico de los circuitos VLSI. En primer lugar se establece la topología óptima del floorplan, es decir las posiciones relativas de los módulos componentes. A continuación, se determinan tanto la implementación de cada módulo como el área total del floorplan para que ésta sea mínima.

En los floorplans particionados, el problema de optimización del área se puede resolver en un tiempo $O(n^2)$, donde n es la cantidad de módulos que componen el floorplan [OTTE83].

En los floorplans no particionados, para resolver el problema de optimización del área hay que realizar una búsqueda exhaustiva por todos los nodos del árbol. Esta búsqueda es factible para aquellos casos que sean sencillos (pocos módulos con pocas implementaciones cada uno de ellos). Sin embargo, cuando el número de módulos y el de sus correspondientes implementaciones es importante, el problema puede llegar a ser implanteable debido a un tiempo de proceso excesivamente elevado.

Es decir, si se plantea un problema con m módulos y p implementaciones para cada uno de ellos, habría que realizar una búsqueda por p^m nodos hasta obtener las implementaciones óptimas del floorplan no particionado (Tabla 3-1).

Número de Módulos	Implementaciones de cada Módulo	Número Total de Búsquedas a Realizar
5	5	$5^5 = 3.125$
5	6	$6^5 = 7.776$
5	7	$7^5 = 16.807$
5	8	$8^5 = 32.768$
5	9	$9^5 = 59.049$
5	10	10^5
25	5	$5^{25} \approx 2,98 \cdot 10^{17}$
25	6	$6^{25} \approx 2,84 \cdot 10^{19}$
25	7	$7^{25} \approx 1,34 \cdot 10^{21}$
25	8	$8^{25} \approx 3,77 \cdot 10^{22}$
25	9	$9^{25} \approx 7,17 \cdot 10^{23}$
25	10	10^{25}
125	5	$5^{125} \approx 2,35 \cdot 10^{87}$
125	6	$6^{125} \approx 1,85 \cdot 10^{97}$
125	7	$7^{125} \approx 4,33 \cdot 10^{105}$
125	8	$8^{125} \approx 7,69 \cdot 10^{112}$
125	9	$9^{125} \approx 1,90 \cdot 10^{119}$
125	10	10^{125}
625	5	5^{625}
625	6	6^{625}
625	7	7^{625}
625	8	8^{625}
625	9	9^{625}
625	10	10^{625}

Tabla 3-1 Búsquedas a realizar según el número de módulos y de implementaciones.

Por tanto, la consecución de la solución final óptima requiere tantos procesos que el tiempo de ejecución puede llegar a ser en ciertos casos implanteable. Por otro lado, la realización de la búsqueda por todos los nodos no significa que todas las implementaciones obtenidas sean útiles ya que muchas de ellas dan lugar a floorplans no particionados redundantes. Así, una vez seleccionadas las implementaciones útiles habrá que elegir de entre ellas, las que sean óptimas en área.

Existen diversos algoritmos que resuelven, con limitaciones, la problemática planteada. Aunque todos ellos logran las mismas o parecidas implementaciones finales, no todos lo consiguen en unos tiempos de proceso razonables.

3.2 SOLUCIONES EXISTENTES

Entre los algoritmos desarrollados por los diferentes investigadores que han abordado este problema, cabe destacar el algoritmo rama-y-límite (BB) de Wimer, Koren y Cederbaum [WIME89] cuyos tiempos de proceso son exponenciales, el algoritmo OPT de T.-C. Wang y Wong [WANG92b] que es una extensión de la técnica utilizada por Stockmeyer [STOC83], el algoritmo ES de K. Wang y Chen [WANG93a] que restringe las formas de los módulos para conseguir estructuras con espacio desperdiciado cero, y el algoritmo pseudopolinomial AreaMin de Pan y Liu [PAN95].

Tanto T.C. Wang y Wong como K. Wang y Chen dan una gran importancia a las implementaciones redundantes y las eliminan en la mayoría de los casos, no cuando se empiezan a generar si no en fases posteriores de sus algoritmos. Esto implica el acarreo innecesario de configuraciones, lo que supone un gran problema de tiempo de proceso.

3.2.1 Algoritmo de Rama-y-Límite o BB

Wimer, Koren y Cederbaum [WIME88] enunció el concepto de espacio desperdiciado cero y desarrollaron el algoritmo "rama-y-límite" o BB (Figura 3-1) que utiliza grafos para la consecución de floorplans generales [WIME89].

```
ALGORITMO RAMA-Y-LÍMITE
  A_min = ∞
  ASIGNAR en la raíz, nivel 0, las longitudes iniciales de los arcos de G y H
  WHILE NOT la raíz vuelva hacia atrás IR al nivel i+1 DO
    BEGIN IF A ≥ A_min
              THEN RETROCEDER
              ELSE IF se han agotado todas las implementaciones posibles
                       del presente bloque
                   THEN IR al nivel i-1
                   ELSE IF una hoja es rechazada
                            THEN IF A< A_min
                                     THEN  A_min = A
                                           IR al nivel i+1
                                     ELSE  IR al nivel i+1
                            ELSE IR a la siguiente implementación
    END
```

Figura 3-1 Algoritmo rama-y-límite

Este algoritmo determina en primer lugar, los módulos que pueden ser considerados en un nivel dado del árbol. Estos módulos al unirse entre sí formando bloques representan layouts parciales en el que se asignan unos niveles y dentro de ellos se examinan todas las posibles dimensiones del bloque resultante.

A continuación, asigna a los arcos de los grafos G y H (Figura 3-2) las anchuras y alturas del módulo o bloque del nivel correspondiente. En caso de que el área de los nodos padre sea más grande que el área de algunos de los previamente examinados, esos nodos no se atraviesan.

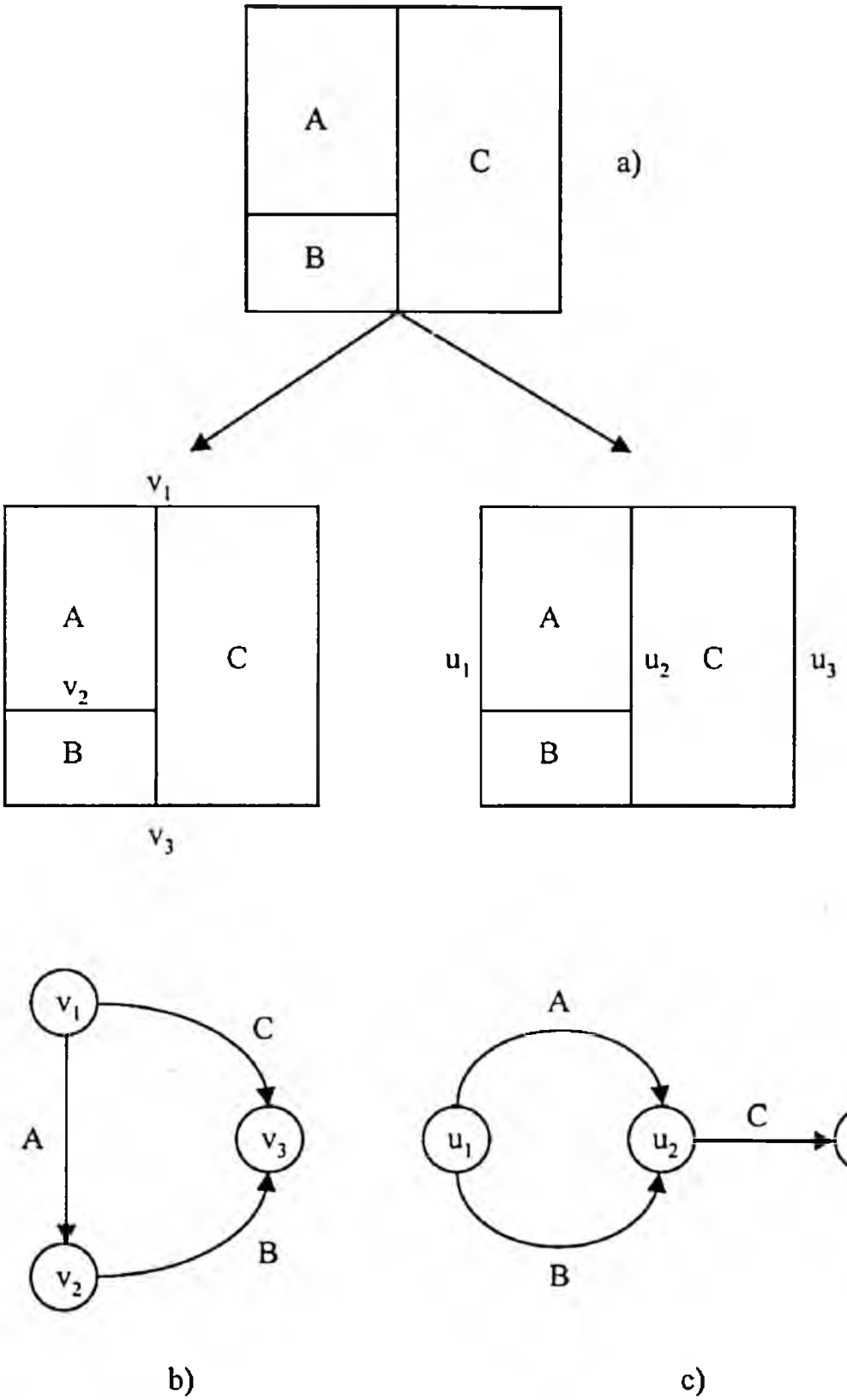


Figura 3-2 Representación de: a) un floorplan, b) el grafo $\mathcal{G}$ y c) el grafo $\mathcal{H}$.

La eficacia de este algoritmo está determinada por los siguientes factores:

1. Tiempo de obtención del área mínima de cada hoja.
2. Tiempo de obtención del límite más bajo del área resultante al colocar los bloques en cada una de las hojas (si se consigue un límite más bajo habrá que volver hacia atrás inmediatamente, reinicializar los valores de los arcos de $\mathcal{G}$ y $\mathcal{H}$).
3. El orden de análisis de las dimensiones de un bloque en el correspondiente nivel del árbol.

3.2.2 Algoritmo OPT

El algoritmo OPT es una extensión de la técnica de Stockmeyer y T.C.-Wang y Wong lo aplican para la resolución del problema de obtención de un floorplan no particionado de orden cinco óptimo en área (Figura 3-3).

```
ALGORITMO OPT
  INPUT:  Un floorplan no particionado de orden cinco y el conjunto de
          implementaciones de cada módulo.
  OUTPUT: El conjunto de implementaciones no redundantes óptimas en área
          de un wheel.
  BEGIN
    Proceso α;
    Proceso β;
    Proceso γ;
    Proceso δ;
  END
```

Figura 3-3 Algoritmo OPT.

El objetivo de este algoritmo es la localización de las implementaciones óptimas de los rectángulos básicos que componen el floorplan, teniendo en cuenta que para cada módulo, existe una lista-R con un conjunto de

implementaciones de la forma $\{(w_1,h_1),(w_2,h_2),...(w_n,h_n)\}$ donde w_i representa la anchura y h_i la altura.

Mediante el método de sintesis bottom-up, se van uniendo los rectángulos básicos o módulos de forma recursiva, hasta conseguir el floorplan deseado. Una vez conocida el área óptima, mediante un proceso de descomposición top-down, se obtiene el floorplan o floorplans con las implementaciones óptimas de cada rectángulo básico o módulo.

Dicho algoritmo consta de cuatro procesos en el que cada uno de ellos realiza la unión de dos módulos o de un bloque y un módulo (Figura 3-4):

- Proceso α: combina todas las implementaciones del módulo B_1 con las de B_2. Éstas se introducen en una única lista que contiene tantas sub-listas como implementaciones tenga el módulo B_1.
- Proceso β: une las implementaciones del bloque obtenido en el proceso anterior B_1B_2, con las del módulo B_3. Estas implementaciones obtenidas se introducen en una única lista que contiene a su vez tantas sub-listas como sub-listas tenga el bloque B_1B_2. Cada vez que efectúa dicha unión comprueba si el elemento resultante es redundante con los anteriormente obtenidos, y si lo es lo elimina.
- Proceso γ: combina las implementaciones del bloque obtenido, $B_1B_2B_3$, con las del módulo B_4. En este proceso tiene en cuenta cuatro casos diferentes:
    a) Cuando la altura del módulo B_2 más la altura del módulo B_3 es mayor que la del módulo B_4, $h(b_2)+h(b_3)>h(b_4)$, obtiene un bloque $B_1B_2B_3B_4$ con una nueva zona desaprovechada.
    b) Cuando la altura del módulo B_2 más la altura del módulo B_3 es menor que la del módulo B_4, $h(b_2)+h(b_3)<h(b_4)$, también obtiene un bloque $B_1B_2B_3B_4$ con una nueva zona desaprovechada.

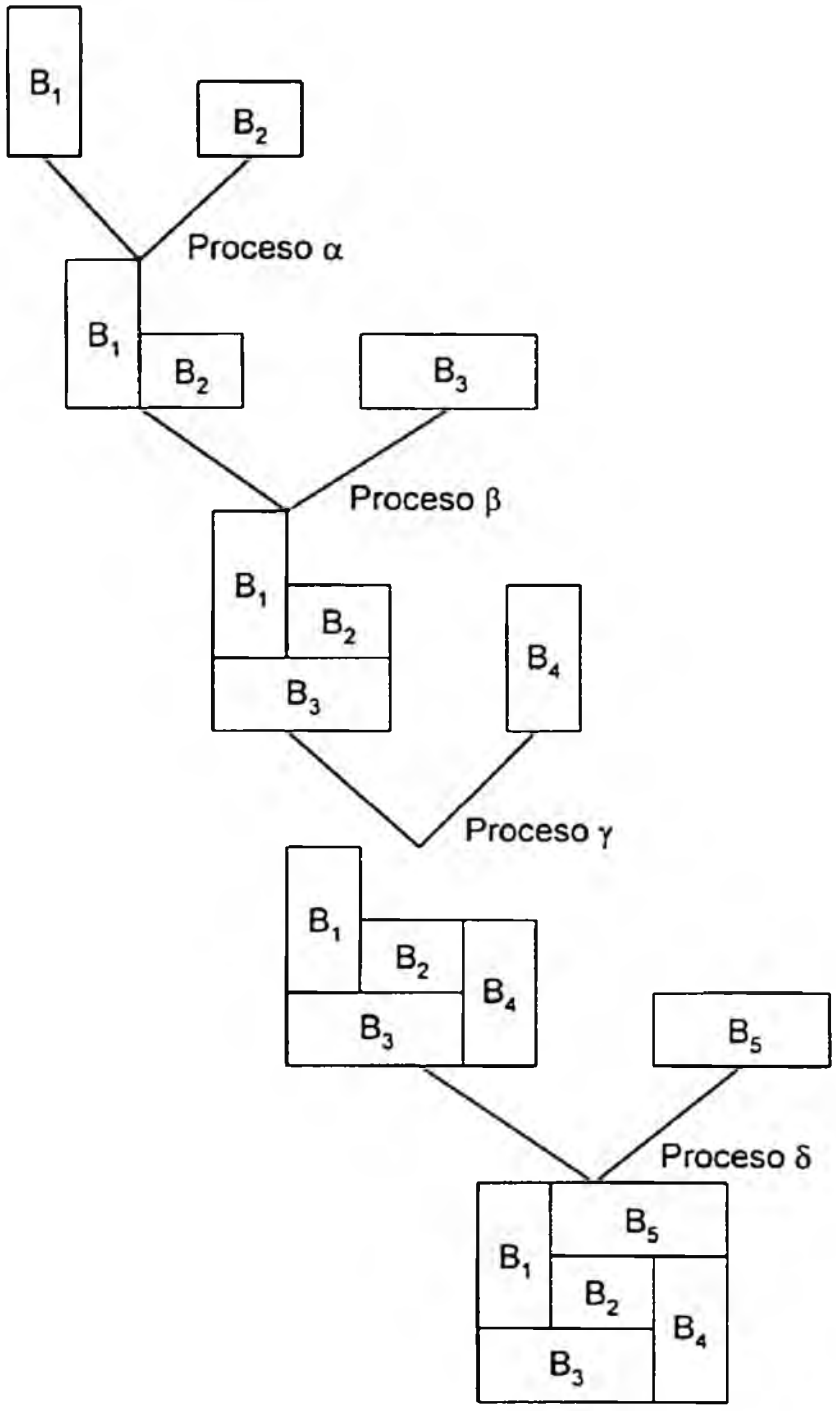


Figura 3-4 Arbol del algorimo OPT.

c) Cuando la altura del módulo B_2 más la altura del módulo B_3 es igual que la del módulo B_4, $h(b_2)+h(b_3)=h(b_4)$, obtiene un bloque $B_1B_2B_3B_4$ que no añade zonas desaprovechadas.

d) Cuando la altura del módulo B_4 es mayor que la del módulo B_2 y menor que la del módulo B_1, $h(b_1)>h(b_4)>h(b_2)$, obtiene un bloque $B_1B_2B_3B_4$ con una nueva zona desaprovechada.

Las implementaciones del bloque $B_1B_2B_3B_4$ obtenidas se introducen en una única lista que contiene a su vez tantas sub-listas como sub-listas tenga el bloque $B_1B_2B_3$. Una vez efectuada la unión comprueba si el elemento resultante es redundante con los anteriormente obtenidos, y si lo es lo elimina.

- Proceso ô: une las implementaciones del bloque $B_1B_2B_3B_4$ con las del módulo B_5 e introduce todas ellas en una única lista. A continuación, clasifica las anchuras en orden creciente y las alturas en orden decreciente para comprobar si el elemento resultante es o no redundante con los anteriormente obtenidos. Si es así, se eliminan todas aquellas soluciones que sean dominantes, de forma que terminado dicho proceso, obtiene en teoría el conjunto de implementaciones óptimas en área del floorplan no particionado de orden cinco.

3.2.3 Algoritmo ES

El algoritmo ES es una extensión de la técnica de Stockmeyer, y K. Wang y Chen lo aplican para la resolución del problema de obtención de un floorplan no particionado de orden cinco óptimo en área demostrando que bajo ciertas restricciones de forma, se puede encontrar una clase de estructuras con espacio desperdiciado cero (Figura 3-5).

El objetivo de este algoritmo es la localización de las implementaciones óptimas de los rectángulos básicos que componen el floorplan, teniendo en cuenta que para cada módulo, existe una lista-R con un conjunto de implementaciones de la forma $\{(w_1,h_1),(w_2,h_2)...(w_i,h_i)...(w_n,h_n)\}$ donde w_i representa la anchura y donde h_i la altura.

Mediante el método de síntesis bottom-up, se van uniendo los rectángulos, de forma recursiva, hasta conseguir el floorplan deseado. Una vez conocida el área óptima, mediante un proceso de descomposición top-down, se obtiene el floorplan o floorplans con las implementaciones optimas de cada rectángulo.

ALGORITMO ES
INPUT: Un floorplan no particionado de orden cinco y el conjunto de implementaciones de cada módulo
OUTPUT: El conjunto de implementaciones no redundantes óptimas en área de un wheel
BEGIN
Proceso R^2L1.
Proceso RL^21.
Proceso R^2L2.
Proceso L^2R1.
Proceso Clasificación-Eliminación.
END

Figura 3-5 Algoritmo ES

Dicho algoritmo consta de cinco procesos y cada uno de ellos efectúa la unión de dos módulos o de un bloque y un módulo (Figura 3-6):

- Proceso R^2L1: combina todas las implementaciones del módulo B_3 con las de B_5. Éstas se introducen en una única lista que contiene tantas sub-listas como implementaciones tenga el módulo B_3.
- Proceso RL^21: une las implementaciones del bloque obtenido en el proceso anterior, B_3B_5, con las del módulo B_2 distinguiendo tres casos:
    a) Cuando la altura del módulo B_2 es mayor que la del bloque B_3B_5, el bloque $B_3B_5B_2$ obtenido contiene una zona desaprovechada determinada por la diferencia entre la altura del módulo B_2 y la del bloque B_3B_5.

b) Cuando la altura del módulo B_2 es menor que la del bloque B_3B_5, el bloque $B_3B_5B_2$ obtenido contiene una zona desaprovechada determinada por la diferencia entre la altura del bloque B_3B_5 y el módulo B_2.

c) Cuando la altura del módulo B_2 es igual que la del bloque B_3B_5, el bloque $B_3B_5B_2$ obtenido no contiene zonas desaprovechadas.

Las implementaciones del bloque $B_3B_5B_2$ obtenidas, se introducen en una única lista que contiene a su vez tantas sub-listas como sub-listas tenga el bloque B_3B_5.

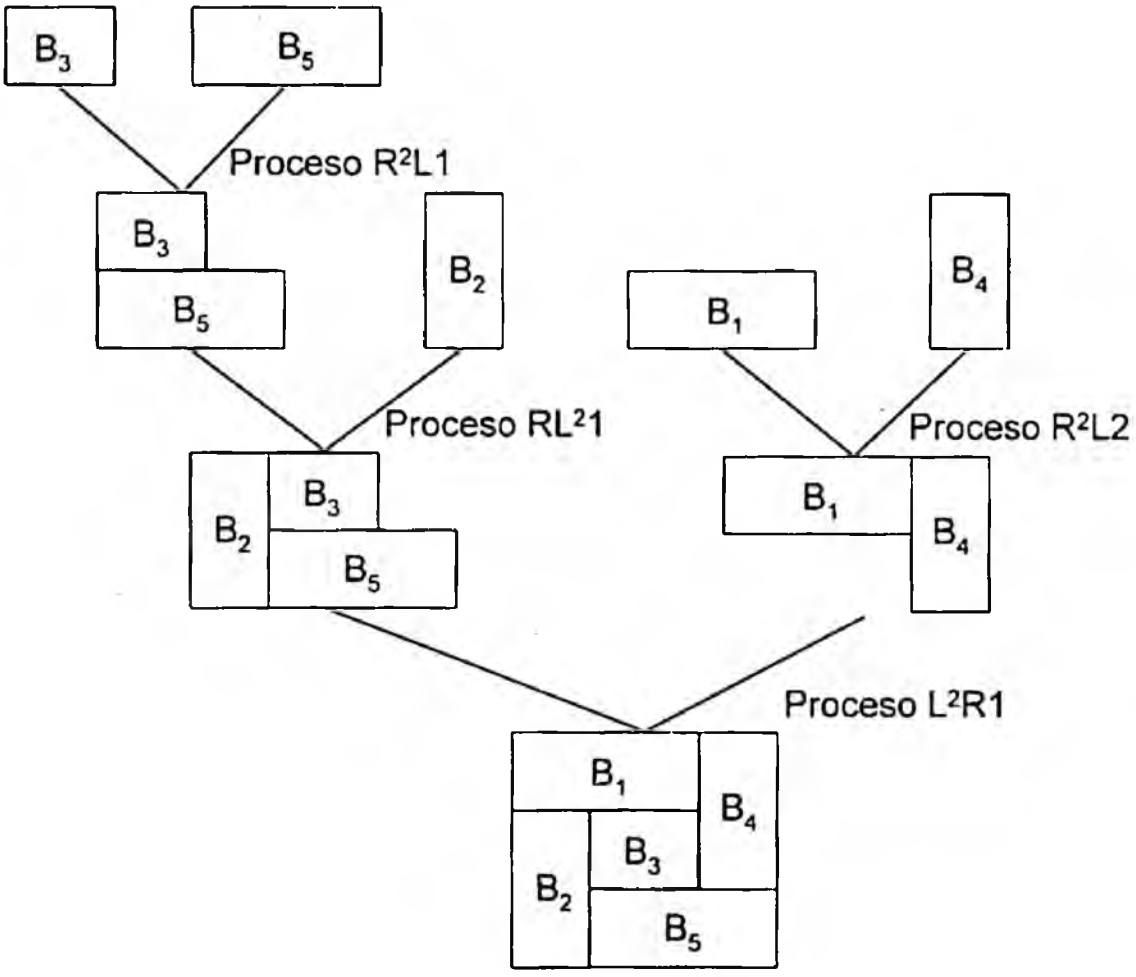


Figura 3-6 Árbol del algoritmo ES.

- Proceso R^2L2: combina todas las implementaciones del módulo B_1 con las de B_4. Éstas se introducen en una única lista que contiene a su vez tantas sub-listas como implementaciones tenga el módulo B_1.

- Proceso L^2R1: une las implementaciones del bloque B_1B_4 con las del bloque $B_3B_5B_2$ y distingue tres casos:
  a) La anchura del módulo B_1 es mayor que la diferencia entre las anchuras de los módulos B_5 y B_3 más la del módulo B_2. $w(b_1)>w(b_5)-w(b_3)+w(b_2)$. En este caso el wheel obtenido contiene zonas desaprovechadas.
  b) La anchura del módulo B_1 es mayor que la diferencia entre las anchuras de los módulos B_5 y B_3 más la del módulo B_2. $w(b_1)<w(b_5)-w(b_3)+w(b_2)$. En este caso el wheel obtenido contiene zonas desaprovechadas.
  c) Cuando la anchura del módulo B_1 es igual que la diferencia entre las anchuras de los módulos B_5 y B_3 más la del módulo B_2, $w(b_1)=w(b_5)-w(b_3)+w(b_2)$ el floorplan no particionado de orden cinco resultante no contiene zonas desaprovechadas.

  Las implementaciones obtenidas se introducen en una única lista-R que contiene a su vez tantas implementaciones como uniones se han realizado.

- Proceso Clasificación-Eliminación: la lista obtenida en el proceso anterior contiene un conjunto de implementaciones que no están clasificadas bajo ningún concepto y que puede contener elementos redundantes. Por ello, clasifica la lista en orden creciente de anchura y en orden decreciente de altura, y a continuación, elimina todas aquellas soluciones que sean dominantes. De esta forma obtiene el conjunto de implementaciones óptimas en área del wheel.

3.2.4 Algoritmo AreaMin

Pan y Liu enfocaron el problema de la resolución de los wheels óptimos en área y del tiempo de proceso mediante la búsqueda de las implementaciones no redundantes, explotando las estructuras de sub-floorplans que pueden existir dentro de un floorplan (Figura 3-7). Este mismo proceso lo extendieron a los floorplans generales.

ALGORITMO AreaMin
INPUT: Un floorplan no particionado de orden cinco F y el conjunto de implementaciones de cada módulo.
OUTPUT: El conjunto de implementaciones no redundantes de F.
BEGIN
IF F es un módulo
THEN RETURN el conjunto de implementaciones no redundantes del módulo;
ELSE P ← la unión de los sub-floorplans F_1, F_2, ... F_t en F, siendo B_i bloques de los que se compone F_i;
FOR i=1 TO t DO $L(B_i)$ ← AreaMin(F_i);
CALL al algoritmo Clase 1, Clase 5 o Clase 9 que a través de la unión P obtiene L(P);
RETURN L(P);
END

Figura 3-7 Algoritmo AreaMin.

Sea F un floorplan obtenido mediante P uniones de un conjunto de sub-floorplans F_1, F_2, ... F_t donde cada uno de ellos se compone a su vez de B_i bloques. Pan y Liu, mediante su algoritmo AreaMin [Pan95], determinan recursivamente las implementaciones no redundantes de cada F_i para posteriormente, a través de otro algoritmo, combinarlas hasta la consecución del floorplan óptimo en área. Dicho proceso podrá ser, o el algoritmo de Stockmeyer cuando los floorplans sean particionados, o el planteado por él mismo cuando los floorplans sean no particionados.

A la hora de generar los wheels óptimos en área, Pan y Liu establecen nueve clases distintas de relaciones entre los módulos y las dimensiones de cada una de las implementaciones (Figura 3-8):

- Las clases 1, 2, 3 y 4 se dan cuando la anchura w(r) y la altura h(r) del wheel están determinadas por dos módulos. Estas clases son (Figuras 3-8.a, 3-8.b, 3-8.c y 3-8.d):

  Clase 1: $w(r)=w(b_1)+w(b_2)$ $h(r)=h(b_2)+h(b_3)$
  Clase 2: $w(r)=w(b_1)+w(b_2)$ $h(r)=h(b_1)+h(b_4)$
  Clase 3: $w(r)=w(b_3)+w(b_4)$ $h(r)=h(b_1)+h(b_4)$
  Clase 4: $w(r)=w(b_3)+w(b_4)$ $h(r)=h(b_2)+h(b_3)$

- Las clases 5 y 6 se dan cuando la anchura w(r) del wheel está determinada por dos módulos y la altura h(r) del wheel lo está por tres módulos. Estas clases son (Figura 3-8.e y Figura 3-8.f):

  Clase 5: $w(r)=w(b_1)+w(b_2)$ $h(r)=h(b_2)+h(b_5)+h(b_4)$
  Clase 6: $w(r)=w(b_3)+w(b_4)$ $h(r)=h(b_2)+h(b_5)+h(b_4)$

- Las clases 7 y 8 se dan cuando la anchura w(r) del wheel está determinada por tres módulos y la altura h(r) del whheel lo está por dos módulos. Estas clases son (Figura 3-8.g y Figura 3-8.h):

  Clase 7: $w(r)=w(b_1)+w(b_5)+w(b_3)$ $h(r)=h(b_1)+h(b_4)$
  Clase 8: $w(r)=w(b_1)+w(b_5)+w(b_3)$ $h(r)=h(b_2)+h(b_3)$

- La clase 9 se da cuando la anchura w(r) y la altura h(r) del wheel están determinadas por tres módulos (Figura 3-8.i):

  Clase 9: $w(r)=w(b_1)+w(b_5)+w(b_3)$ $h(r)=h(b_2)+h(b_5)+h(b_4)$

Esta distinción la realiza mediante tres algoritmos que siguiendo su terminología son respectivamente: Clase 1 que detecta los casos de las Clases 1, 2, 3 y 4, Clase 5 que a su vez detecta las Clases 5, 6, 7 y 8, y por último la Clase 9.

El tiempo de proceso necesario así como el número total de búsquedas a realizar para la obtención de la lista final con sus correspondientes implementaciones es pseudopolinomial.

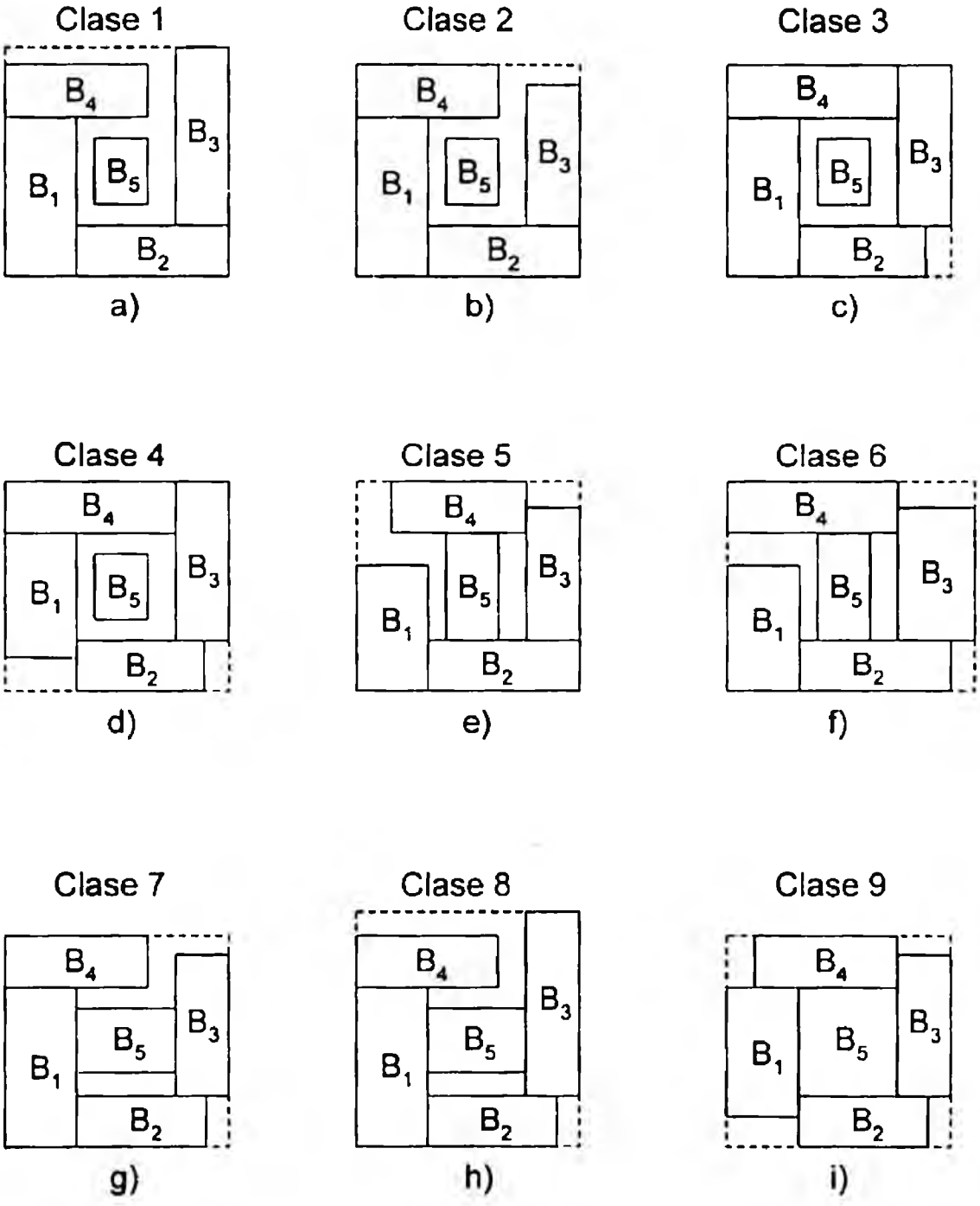


Figura 3-8 Las nueve clases planteadas por PAN y LIU.

3.2.5 Comparativa

La potencia del algoritmo se determina por el tiempo de proceso a la hora de obtener entre todas las implementaciones no redundantes, el floorplan no particionado óptimo en área.

En las tablas 3-2, 3-3 y 3-4, recogemos el rendimiento de los algoritmos OPT [WANG92b], ES [WANG93a], AreaMin [PAN95] y la implementación del algoritmo desarrollado por Arvindam, Kumar y Rao [ARVI89] rama-y-límite o BB. Esta información es el resultado de la comparativa realizada por los autores de dichos algoritmos.

Llama la atención que en dicha comparativa siempre recogen los datos publicados por el autor correspondiente sin tener en cuenta que la tecnología de los ordenadores evoluciona rápidamente y que por lo tanto, los resultados relativos a los tiempos de proceso serían distintos si se procesaran todos los algoritmos en un mismo tipo de ordenador.

La tabla 3-2 indica, por una parte el número de implementaciones no redundantes contenidas en una lista-R y por otra el área óptima. Todos los algoritmos estudiados producen las mismas áreas óptimas, sin embargo, el problema se plantea en la cantidad de nodos que deben visitarse para obtener dicha solución en un tiempo de proceso factible.

La información dada en la tabla 3-3, indica el tiempo de proceso necesario para la obtención de las implementaciones finales no redundantes contenidas en una lista-R, que dan lugar a floorplans no particionados óptimos en área. Además, en dicha tabla se indica el número total de búsquedas que realizan cada uno de los algoritmos dependiendo de la prueba.

De todos los algoritmos estudiados, el de peor comportamiento es el de rama-y-límite o BB. Cuando el número de implementaciones que tiene cada

uno de los 25 módulos es superior a seis, el tiempo de proceso para este algoritmo BB es realmente exorbitante, mientras que el resto de los algoritmos (OPT, ES y AreaMin) obtienen, en dichos casos, las implementaciones no redundantes en muy pocos segundos. Por tanto, el algoritmo de Wimer, Koren y Cederbaum, BB, es en principio, implanteable para casos muy complejos.

Implementaciones de los Módulos A=B=C=D=E	Área Mínima
{(1,4),(2,2),(4,1)}	121
{(1,6),(2,3),(3,2),(6,1)}	176
{(1,16),(2,8),(4,4),(8,2),(16,1)}	484
{(1,12),(2,6),(3,4),(4,3),(6,2),(12,1)}	352
{(1,24),(2,12),(3,8),(4,6),(6,4),(8,3),(12,2),(24,1)}	660

Tabla 3-2 Áreas e implementaciones finales para un wheel con 25 módulos.

Implementaciones de los Módulos A=B=C=D=E	Nº Total de Búsquedas	Tiempo de Proceso en Segundos BB	OPT	ES	AreaMin
{(1,4),(2,2),(4,1)}	325	59	0,3	0,06	0,02
{(1,6),(2,3),(3,2),(6,1)}	425	1.506	0,7	0,10	0,03
{(1,16),(2,8),(4,4),(8,2),(16,1)}	525	10.903	1,8	0,13	0,13
{(1,12),(2,6),(3,4),(4,3),(6,2),(12,1)}	625	>2 días	2,0	0,50	0,18
{(1,24),(2,12),(3,8),(4,6),(6,4),(8,3),(12,2),(24,1)}	825	>2 días	4,3		0,43

Tabla 3-3 Tiempo de proceso en segundos para un wheel con 25 módulos.

La tabla 3-4 incluye, por una parte el número total de búsquedas que se deberían realizar dependiendo de la prueba y por otra, el número real de nodos visitados por cada uno de los algoritmos mencionados. Si se tuviera que realizar el número total de búsquedas, algunos de los casos planteados serían

irrealizables. Sin embargo, la mayoría de las búsquedas se pueden descartar ya que las soluciones producidas son redundantes. Esto implica que el número de nodos explorados para la obtención del floorplan se puede reducir considerablemente.

También en este aspecto, el que peor se comporta es el de rama-y-límite o BB. Cuando cada uno de los módulos tiene más de seis implementaciones, el número de nodos visitados es excesivo por lo que dicho algoritmo para casos relativamente complejos no es operativo.

Implementaciones de los Módulos A=B=C=D=E	Nº Total de Búsquedas	Número de Nodos Visitados			
		BB	OPT	ES	AreaMin
{(1,4),(2,2),(4,1)}	3^{25}	1.620	341	235	168
{(1,6),(2,3),(3,2),(6,1)}	4^{25}	37.706	644	568	365
{(1,16),(2,8),(4,4),(8,2),(16,1)}	5^{25}	321.985	1.805	1.632	776
{(1,12),(2,6),(3,4),(4,3),(6,2),(12,1)}	6^{25}	>500.000	1.966	2.948	994
{(1,24),(2,12),(3,8),(4,6),(6,4),(8,3),(12,2),(24,1)}	8^{25}	>500.000	4.759		2.031

Tabla 3-4 Número de nodos visitados para un wheel con 25 módulos.

3.3 OTROS ALGORITMOS APLICABLES A FLOORPLANS GENERALES

A la hora de optimizar el área de un floorplan general nos encontramos con floorplans particionados y no particionados. Anteriormente, hemos visto los algoritmos más destacables que resuelven el problema en los no particionados. A continuación, vamos a exponer los algoritmos de Otten y de Stockmeyer que resuelven de una forma definitiva la problemática en los floorplans particionados, y que por tanto se usarán siempre que nos encontremos con dichos floorplans.

3.3.1 Algoritmo de Otten con Módulos de Infinitas Implementaciones

El algoritmo presentado por Otten [OTTE83] asume que cada módulo tiene un número infinito de posibles implementaciones. Estas implementaciones se pueden representar con una función lineal decreciente que se obtiene mediante el método de síntesis bottom-up.

Por ejemplo, sea F un floorplan y T su correspondiente árbol cuyo nodo interno U tiene como hijos U_1 y U_2 (Figura 3-9). Las implementaciones de cada uno de los rectángulos básicos U_1 y U_2 se agrupan en las listas LU_1 y LU_2 que son respectivamente:

$LU_1=\{(2,8),(4,4),(8,2)\}$ $LU_2=\{(3,10),(5,6),(6,5),(10,3)\}$

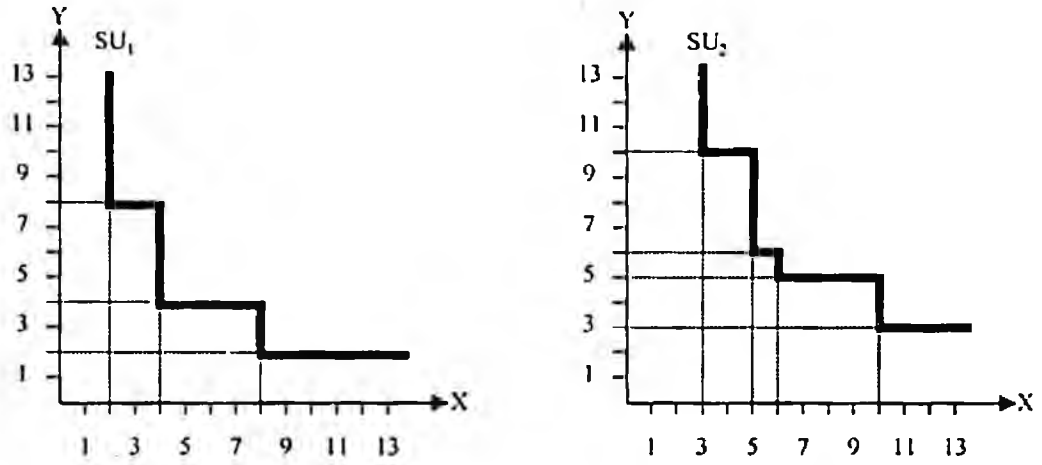


Figura 3-9 Función lineal de U_1 y U_2.

El nodo interno U corresponde a un corte y cada sub-árbol que nace de él a un floorplan FU de segmentación. Si SU es la función que representa las configuraciones de FU; SU_1 y SU_2 serán a las funciones de configuración de U_1 y U_2 respectivamente. Por tanto, SU se puede obtener sumando SU_1 y SU_2 en la dirección del eje X o en la del eje Y, dependiendo del nodo interno U, es decir, del tipo de unión (Figura 3-10).

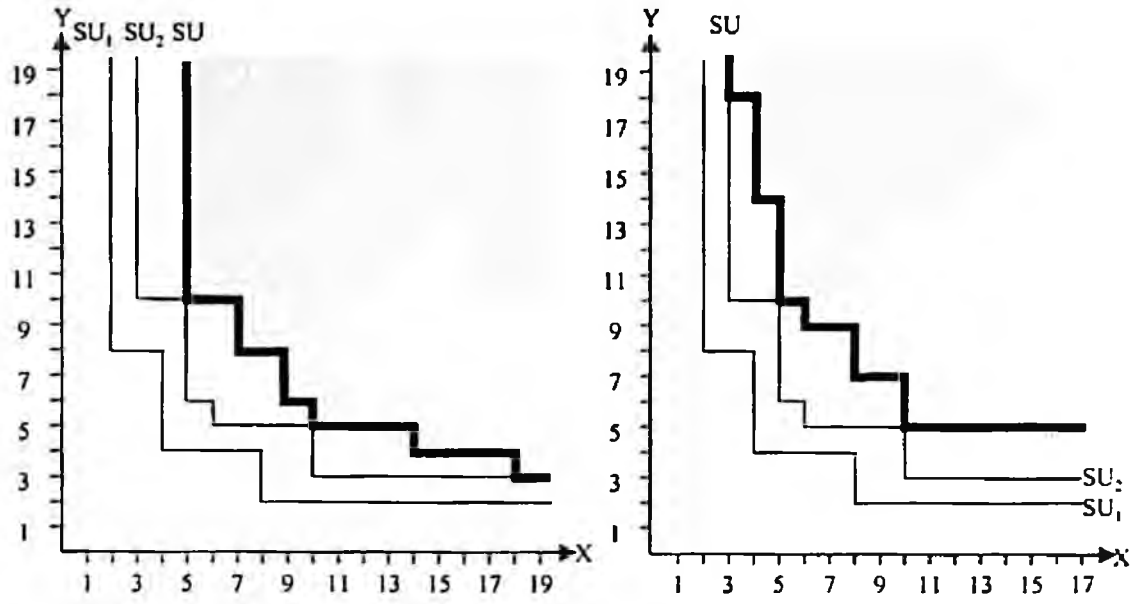


Figura 3-10 Suma de las funciones SU_1 y SU_2 de forma: a) horizontal y b) vertical.

Tras obtener la curva de configuración del nodo raíz hay que estudiar todos los ángulos de la misma y encontrar los que impliquen el área mínima (Figura 3-11). Una vez determinados todos los ángulos óptimos del floorplan, se retrocede para decidir la implementación o implementaciones óptimas de los módulos o rectángulos básicos.

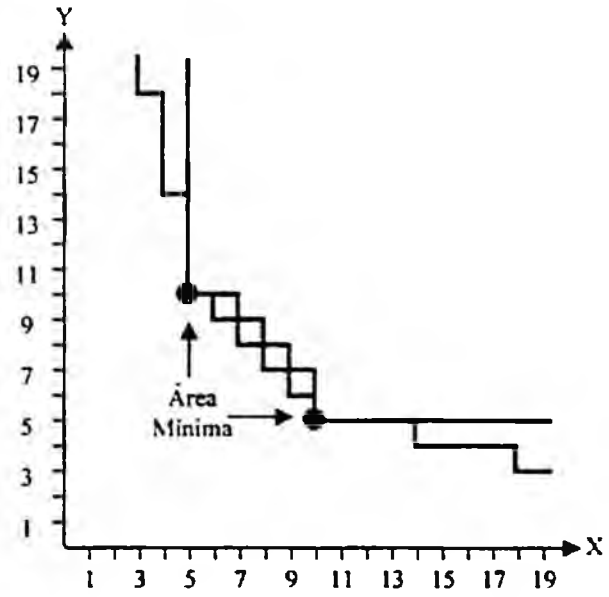


Figura 3-11 Superposición óptima.

3.3.2 Algoritmo de Stockmeyer con Módulos de Finitas Implementaciones

Stockmeyer presentó un algoritmo para resolver el problema de optimización del área en un tiempo polinomial, para módulos con un número finito de implementaciones.

El objetivo de este algoritmo es la localización de las implementaciones óptimas de los rectángulos básicos que componen el floorplan, teniendo en cuenta que para cada módulo, existe una lista-R con un conjunto de implementaciones de la forma $\{(w_1,h_1),(w_2,h_2)...(w_i,h_i)...(w_n,h_n)\}$ donde cada h_i representa la altura y cada w_i la anchura.

Mediante el método de síntesis bottom-up, se van uniendo los rectángulos de forma recursiva, hasta conseguir el floorplan deseado. Una vez conocida el área óptima, mediante un proceso de descomposición top-down, se obtiene el floorplan o floorplans con las implementaciones óptimas de cada rectángulo.

Si un floorplan jerárquico contiene m módulos y su árbol T posee h+1 niveles, entonces se puede decir que la raíz r del árbol T es h y los hijos v_1, v_2, ...v_q pertenecen al nodo interno v cuyas implementaciones estarán recogidas respectivamente en las listas L_v, L_{v1}, L_{v2}, ...L_{vq} correspondientes. Estas implementaciones están clasificadas en orden creciente de anchura y en orden decreciente de altura.

En la figura 3-12 se muestra dicho algoritmo. El paso 2 requiere un tiempo de proceso del orden de $O(|L_r|)$ que dependerá del número de implementaciones que contenga la lista L_r, y el paso 3 necesita $O(m)$, dado que hay $O(m)$ nodos en el árbol T. Sin embargo, el tiempo de proceso del paso 1 depende del método de cálculo empleado sobre L_v.

Aunque su algoritmo está enfocado solamente a floorplans particionados, su técnica se puede extender a la resolución de todo tipo de floorplans jerárquicos.

ALGORITMO DE STOCKMEYER

1. FOR i=1 TO h DO
   FOR cada NODO interno v del nivel i DO
   OBTENER la lista L_V de L_{V1}, L_{V2}, ... L_{Vq}
   END FOR
   END FOR
2. ELEGIR una implementación i con área mínima de la lista L_r
3. Basándonos en i ATRAVESAR T de forma top-down hasta la implementación óptima de cada hoja

Figura 3-12 Algoritmo de Stockmeyer.

4. ALGORITMO AWO

A continuación, exponemos el algoritmo aportación del presente trabajo de investigación que hemos bautizado como Algoritmo del Wheel Óptimo (AWO), y que resuelve la problemática descrita en el capítulo anterior.

4.1 DEFINICIÓN

Este algoritmo implementa un método no heurístico y válido para cualquier floorplan no particionado, utilizando nuevas técnicas de construcción. Para su desarrollo hemos tenido en cuenta los tres criterios que deben cumplirse siempre en la realización de los floorplans:

- Obligatoriedad del mantenimiento de la topología del floorplan de partida.
- Imposibilidad del solapamiento de los módulos.
- Exigencia de que cada módulo tenga siempre un número finito de implementaciones.

El algoritmo AWO calcula el floorplan no particionado óptimo en área mediante la ejecución de cinco fases:

- En la fase AB obtenemos un conjunto de bloques L2 no redundantes y sin zonas desaprovechadas al unir cada una de las implementaciones del módulo B con las del módulo A.
- En la fase ABC logramos un conjunto de bloques L3 no redundantes y sin zonas desaprovechadas al unir cada una de las implementaciones del bloque L2, AB, con las del módulo C.

- En la fase DE uniendo cada una de las implementaciones del módulo D con las del módulo E, conseguimos un conjunto de bloques 7 no redundantes y sin zonas desaprovechadas.

- En la fase ABCDE obtenemos los floorplans, es decir, un conjunto de bloques tanto redundantes como no redundantes con forma rectangular como resultado de unir cada una de las implementaciones del bloque L3, ABC, con las del bloque 7, DE.

- En la fase FINAL obtenemos los floorplans no particionados de orden cinco óptimos en área una vez eliminados todos los que son redundantes.

Todas las uniones producidas en estas cinco fases mediante el proceso de síntesis bottom-up las podemos agrupar en un único árbol donde se representa un floorplan no particionado F jerárquico de orden cinco. Cada hoja del árbol corresponde a un módulo rectangular cuyas implementaciones no redundantes se almacenan en una lista-R irreducible y cada nodo interno corresponde o a un bloque L2, L3 o 7.

En una estructura no particionada, cada uno de los nodos internos de F tiene dos padres que pueden ser o un módulo rectangular (A, B, C, D o E), un bloque L2 (AB), uno L3 (ABC) o un 7 (DE). Así mismo, cada nodo no interno sólo tiene un padre que es un bloque rectangular.

Nuestro algoritmo genera para cada nodo interno de F, mediante un proceso de síntesis bottom-up una lista irreducible correspondiente al conjunto de todas las implementaciones no redundantes del bloque. Una vez obtenida la lista-R asociada con el nodo raíz, la examinamos para localizar aquellas implementaciones óptimas del floorplan no particionado con el menor área posible. A continuación, realizamos un proceso de descomposición top-down hasta localizar las implementaciones de cada uno de los módulos rectangulares que constituyen el floorplan óptimo.

Para reducir el número de nodos visitados y por tanto el tiempo de proceso, eliminamos lo antes posible todas aquellas implementaciones redundantes con zonas desaprovechadas que se van generando en cada uno de las fases de las que consta nuestro algoritmo.

4.2 FASE AB

En esta fase combinamos cada una de las implementaciones del módulo B con las del módulo A logrando un conjunto de bloques L2 AB. En la figura 4-1 se representa dicho proceso.

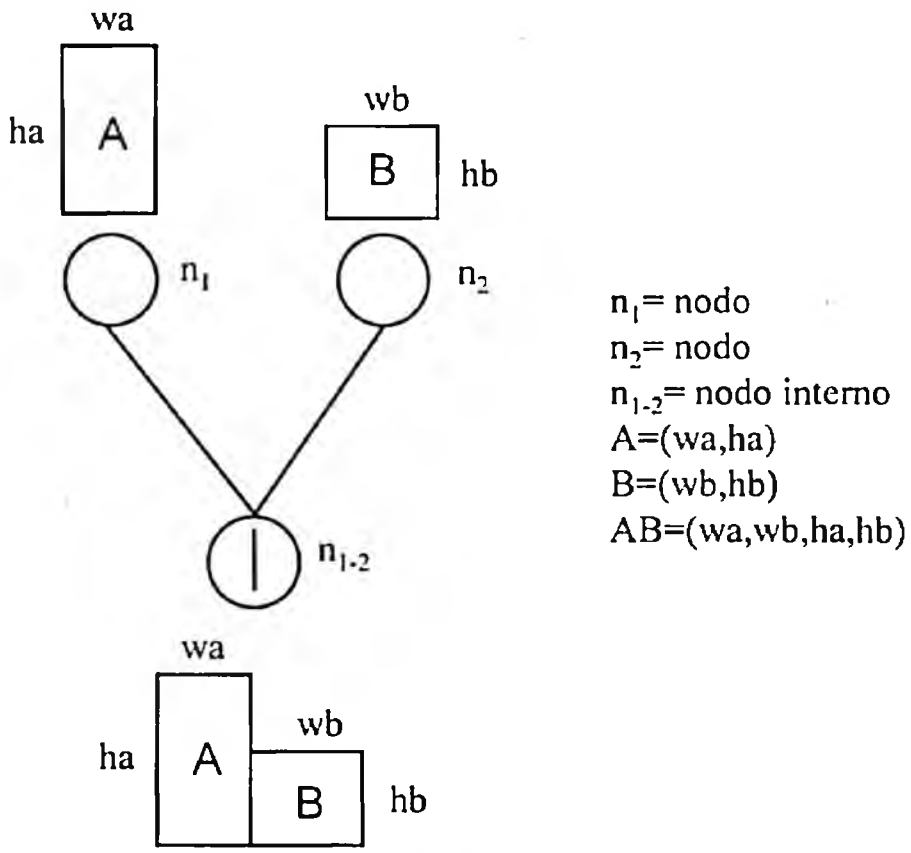


Figura 4-1 Árbol de unión AB.

T.-C.Wang y Wong, K.Wang y Chen, y Wimer, Koren y Cederbaum plantean con sus respectivos algoritmos unir las k_1 implementaciones del módulo A con las k_2 implementaciones del módulo B, obteniéndose siempre $k_1 \bullet k_2$ bloques L2. Sin embargo, este planteamiento puede provocar, en la mayoría de los casos, el arrastre de una o varias implementaciones redundantes con zonas desaprovechadas en cada una de las sub-listas-L2 generadas en AB.

Cada uno de estos elementos redundantes con zonas desaprovechadas generarían, a su vez en los pasos posteriores, otros bloques también redundantes que contendrían zonas desaprovechadas. Lo único que se conseguiría de esta manera, sería la ralentización del proceso de obtención de un floorplan no particionado óptimo en área. Por esta razón, nosotros eliminamos lo antes posible cualquier elemento redundante con zonas desaprovechadas. Toda implementación contenida en cada una de las sub-listas-L2 es redundante si cumple las siguientes propiedades:

P13: $wa_i + wb_i \geq wa_j + wb_j$

P14: $\max(ha_i, hb_i) \geq \max(ha_j, hb_j)$

Por ejemplo, supongamos que las implementaciones de los módulos A y B son: A={(1,6),(2,3),(3,2),(6,1)} y B={(1,6),(2,3),(3,2),(6,1)}. El proceso de unión se puede efectuar de dos formas:

1. Unión de cada una de las implementaciones del módulo A con todas las del módulo B.

   En este caso, se obtendrían los bloques L2 representados en la figura 4-6 y cuya lista-L2 L_{AB} resultante constará de $k_1 \bullet k_2 = 4 \bullet 4 = 16$ elementos:

   L_{AB}={sub-lista 1, sub-lista 2, sub-lista 3, sub-lista 4}

   sub-lista 1={(1,1,6,6),(1,2,6,3),(1,3,6,2),(1,6,6,1)}

   sub-lista 2={(2,1,3,6),(2,2,3,3),(2,3,3,2),(2,6,3,1)}

   sub-lista 3={(3,1,2,6),(3,2,2,3),(3,3,2,2),(3,6,2,1)}

   sub-lista 4={(6,1,1,6),(6,2,1,3),(6,3,1,2),(6,6,1,1)}

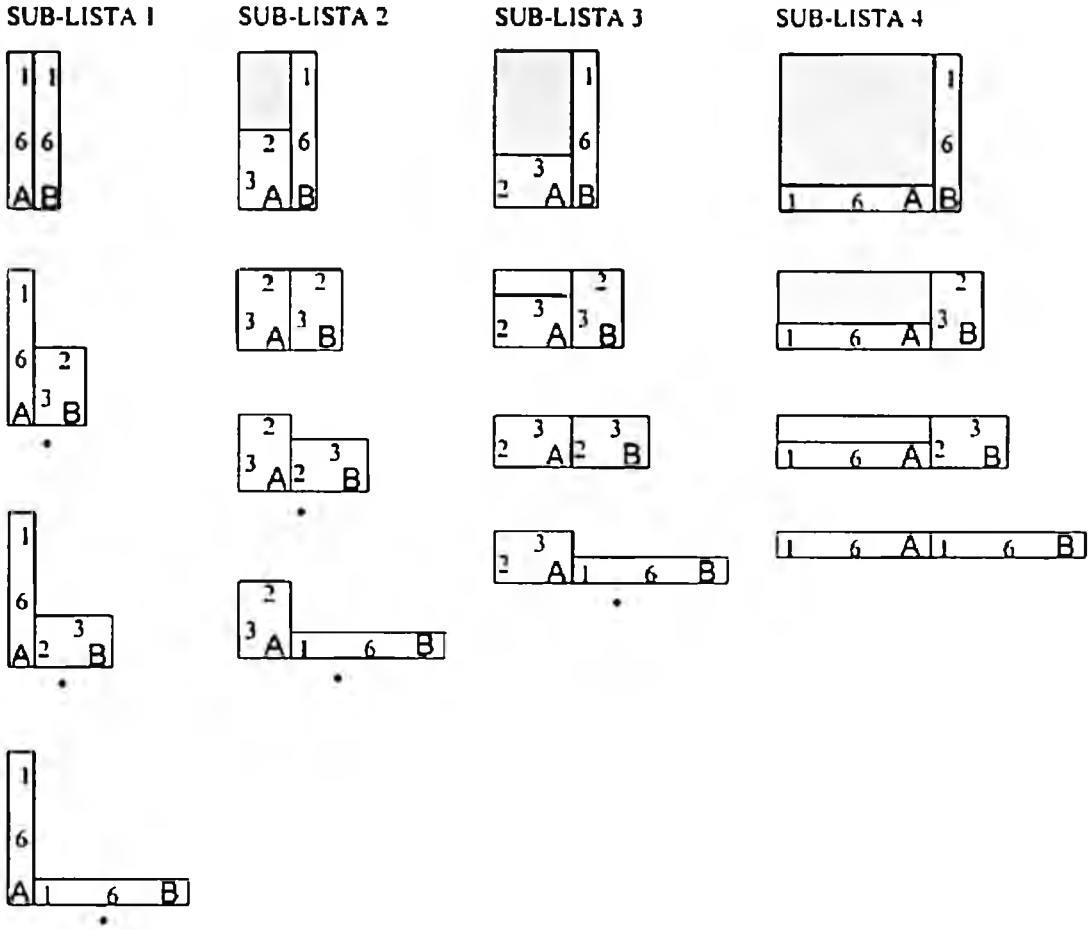


Figura 4-2 Implementaciones en la fase AB al unir A con B.

La única forma de comprobar si en cada una de las sub-listas-L2 hay implementaciones redundantes es ver si todas ellas cumplen las propiedades P13 y P14.

Todos los elementos marcados con un asterisco (*) en la figura 4-2 son implementaciones redundantes con respecto a otros elementos de la misma sub-lista. Sin embargo, dichas configuraciones, al no contener ninguna zona desaprovechada, no se pueden eliminar ya que podrían conducir a soluciones finales óptimas. Por lo tanto, habría que tener en cuenta todas las posibilidades incluidas las redundantes ($k_1 \bullet k_2$ bloques L2 AB).

Los algoritmos planteados por T.-C.Wang y Wong, K.Wang y Chen, y Wimer, Koren y Cederbaum, al llevar a cabo la unión de esta forma, tienen que acarrear todos los elementos (redundantes y no redundantes con zonas desaprovechadas o sin ellas), por lo que el proceso de obtención de la solución óptima es más largo.

2. Unión de cada una de las implementaciones del módulo B con todas las del módulo A.

   En este caso se obtendrían los bloques L2 representados en la figura 4-3 y cuya lista-L2 L_{AB} resultante constará de $k_1 \bullet k_2 = 4 \bullet 4 = 16$ elementos:

   L_{AB}={sub-lista 1, sub-lista 2, sub-lista 3, sub-lista 4}

   sub-lista 1={(1,1,6,6),(2,1,3,6),(3,1,2,6),(6,1,1,6)}

   sub-lista 2={(1,2,6,3),(2,2,3,3),(3,2,2,3),(6,2,1,3)}

   sub-lista 3={(1,3,6,2),(2,3,3,2),(3,3,2,2),(6,3,1,2)}

   sub-lista 4={(1,6,6,1),(2,6,3,1),(3,6,2,1),(6,6,1,1)}

   La única forma de comprobar si en cada una de las sub-listas-L2 hay implementaciones redundantes es ver si todas ellas cumplen las propiedades P13 y P14.

   Todos los elementos marcados con un asterisco (*) en la figura 4-3 son implementaciones redundantes y en todos ellos se cumple que la altura del módulo A es menor que la del módulo B por lo que contienen zonas desaprovechadas.

   Nosotros planteamos que dichas implementaciones redundantes obtenidas se pueden eliminar de la sub-lista correspondiente, o lo que es lo mismo no es necesario tenerlas en cuenta ya que nunca mejorarán las soluciones obtenidas al tener ya en esta fase, zonas desaprovechadas.

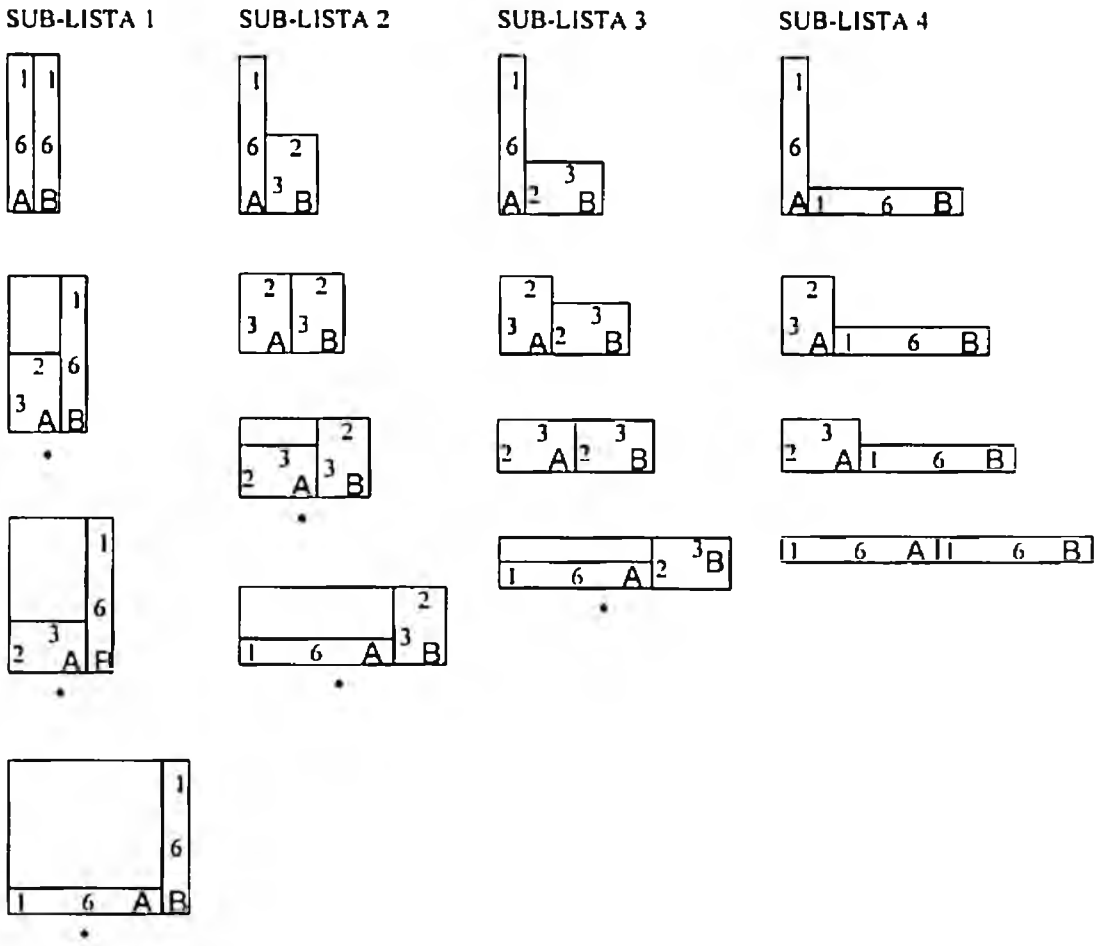


Figura 4-3 Implementaciones en la fase AB al unir B con A.

Como hemos visto, en esta fase se producen dos tipos de implementaciones redundantes L2, las que se deben tener en cuenta y las que se deben eliminar. El mayor problema estriba en determinar cuáles son las de un tipo y cuáles las del otro.

- Implementaciones redundantes que no se deben eliminar.

  En la figura 4-4, tenemos tres bloques L2 resultantes de la unión de los módulos A y B. Los bloques B1 y B2 son redundantes con respecto al B3, al cumplir las propiedades P13 y P14. Sin embargo, al no tener zonas desaprovechadas todos ellos deben tenerse en cuenta, ya que en fases posteriores al unirlos con el resto de módulos (C, D y E) no tienen porque dar lugar a floorplans redundantes.

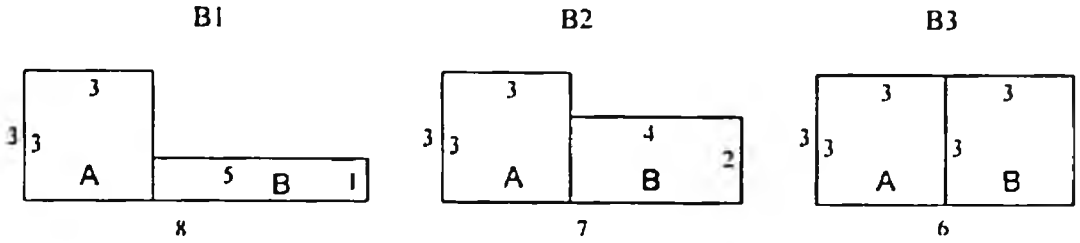


Figura 4-4 Implementaciones AB redundantes a tener en cuenta.

Así, como podemos observar en la figura 4-5, al unir los bloques B1, B2 y B3 con un mismo bloque que contiene los módulos C, D y E se generan tres floorplans B4, B5 y B6 que no son redundantes.

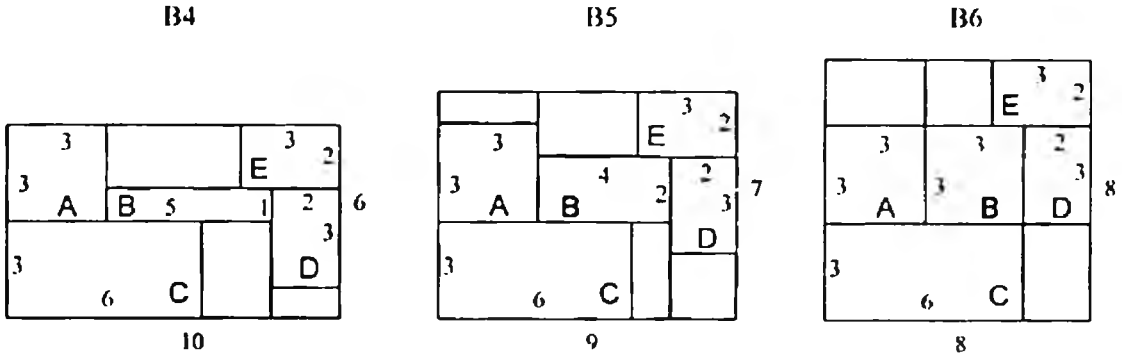


Figura 4-5 Implementaciones AB redundantes que dan lugar a floorplans no redundantes.

- Implementaciones redundantes que deben eliminarse.

  En la figura 4-6, tenemos tres bloques L2 resultantes de la unión de los módulos A y B. Los bloques B2 y B3 son redundantes con respecto al B1 y al tener zonas desaprovechadas hay que eliminarlos ya que nunca se obtendrán con ellos mejores soluciones que las que se puedan conseguir con el bloque B1; en fases posteriores se obtendrían siempre floorplans redundantes al unirlos con el resto de módulos (C, D y E).

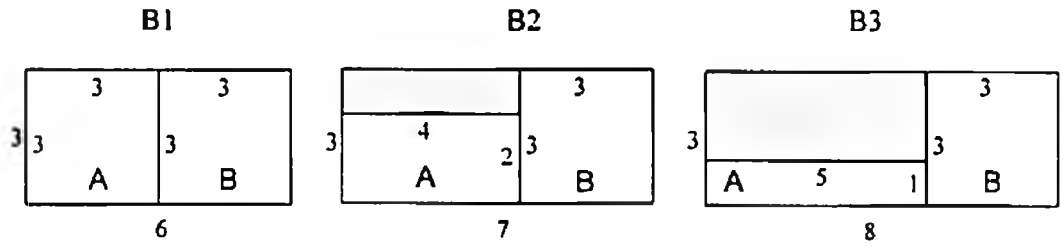


Figura 4-6 Implementaciones AB redundantes a eliminar.

Así, como podemos observar en la figura 4-7, al unir los bloques B1, B2 y B3 con un mismo bloque que contiene los módulos C, D y E se generan tres floorplans, B4, B5 y B6, de los cuales dos son redundantes con respecto al B4.

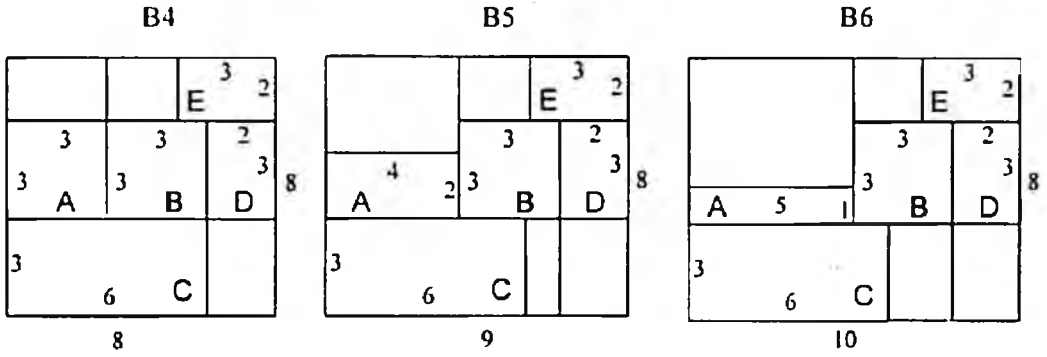


Figura 4-7 Implementaciones AB redundantes que dan lugar a floorplans redundantes.

Como la obtención del bloque L2 a través de la unión de un módulo A con uno B se tiene que realizar manteniendo la topología dada, establecemos una relación entre las alturas de los módulos A y B para evitar en lo posible zonas desaprovechadas:

- Cuando la altura del módulo A sea mayor que la del módulo B (ha>hb), siempre tendremos en cuenta la unión producida, aunque sea redundante, ya que no contiene zonas desaprovechadas.

Teóricamente ésta sería la solución óptima, aunque las uniones en las fases posteriores pueden dar lugar a bloques que no lo sean.

- Cuando la altura del módulo A sea igual que la del módulo B (ha=hb), tendremos en cuenta siempre la unión obtenida, aunque sea redundante, pero no así el resto de implementaciones que contengan zonas desaprovechadas. El bloque L2 resultante contendrá una zona desaprovechada cuya altura la determinará la altura del módulo E en una fase posterior.

- Cuando la altura del módulo A es menor que la del módulo B (ha<hb), eliminaremos la unión generada si es redundante, ya que el bloque L2 obtenido contendrá una zona desaprovechada mayor que la que se tiene con la que es redundante. La altura de la zona desaprovechada será la diferencia de alturas entre A y B más la altura del módulo E.

Al suprimir en las sub-listas del ejemplo anteriormente citado, los elementos redundantes seleccionados, la lista-L2 L_{AB} resultante sería:

L_{AB}={{(1,1,6,6)},{(1,2,6,3),(2,2,3,3)},{(1,3,6,2),(2,3,3,2),(3,3,2,2)},
{(1,6,6,1),(2,6,3,1),(3,6,2,1),(6,6,1,1)}

Como vemos en este ejemplo, de los 16 elementos resultantes hay 6 que se pueden eliminar, por tanto, el número de elementos n es de 10.

Sean dos módulos, A y B, cuyas implementaciones k_1 y k_2 están recogidas en las listas-R, $L_A=\{(wa_i,ha_i)|1 \le i \le k_1\}$ y $L_B=\{(wb_j,hb_j)|1 \le j \le k_2\}$ respectivamente. La fase AB de nuestro algoritmo genera una lista-L2 que contiene k_2 sub-listas como máximo y cada una de ellas un máximo de k_1 implementaciones.

Dado que las listas-R L_A y L_B satisfacen las propiedades siguientes:

P11: $w_i \le w_j$

P12: $h_i \ge h_j$

Todas las implementaciones de cada una de las sub-listas-L2 resultantes de la unión de A y B pertenecientes a la lista-L2 $L_{AB}=\{L_{AB_1}, L_{AB_2} ... L_{AB_{k_2}}\}$ cumplen las siguientes propiedades:

P1: $wa_i \leq wa_j$

P2: $wb_i \leq wb_j$

P3: $ha_i \geq ha_j$

P4: $hb_i \geq hb_j$

$L_A = \{(wa_1, ha_1), (wa_2, ha_2), ..., (wa_{k_1}, ha_{k_1})\}$

$L_B = \{(wb_1, hb_1), (wb_2, hb_2), ..., (wb_{k_2}, hb_{k_2})\}$

$(wb_1, hb_1) \oplus (wa_1, ha_1) = (wa_{11}, wb_{11}, ha_{11}, hb_{11})$

$(wb_1, hb_1) \oplus (wa_2, ha_2) = (wa_{12}, wb_{12}, ha_{12}, hb_{12})$

....

$(wb_1, hb_1) \oplus (wa_{k_1}, ha_{k_1}) = (wa_{1k_1}, wb_{1k_1}, ha_{1k_1}, hb_{1k_1})$

$(wb_2, hb_2) \oplus (wa_1, ha_1) = (wa_{21}, wb_{21}, ha_{21}, hb_{21})$

$(wb_2, hb_2) \oplus (wa_2, ha_2) = (wa_{22}, wb_{22}, ha_{22}, hb_{22})$

....

$(wb_2, hb_2) \oplus (wa_{k_1}, ha_{k_1}) = (wa_{2k_1}, wb_{2k_1}, ha_{2k_1}, hb_{2k_1})$

....

$(wb_{k_2}, hb_{k_2}) \oplus (wa_1, ha_1) = (wa_{k_2 1}, wb_{k_2 1}, ha_{k_2 1}, hb_{k_2 1})$

$(wb_{k_2}, hb_{k_2}) \oplus (wa_2, ha_2) = (wa_{k_2 2}, wb_{k_2 2}, ha_{k_2 2}, hb_{k_2 2})$

....

$(wb_{k_2}, hb_{k_2}) \oplus (wa_{k_1}, ha_{k_1}) = (wa_{k_2 k_1}, wb_{k_2 k_1}, ha_{k_2 k_1}, hb_{k_2 k_1})$

$L_{AB} = \{L_{AB_1}, L_{AB_2}, ... L_{AB_{k_2}}\} =$

$\{\{(wa_{11}, wb_{11}, ha_{11}, hb_{11}), (wa_{12}, wb_{12}, ha_{12}, hb_{12}), ..., (wa_{1k_1}, wb_{1k_1}, ha_{1k_1}, hb_{1k_1})\},$

$\{(wa_{21}, wb_{21}, ha_{21}, hb_{21}), (wa_{22}, wb_{22}, ha_{22}, hb_{22}), ..., (wa_{2k_1}, wb_{2k_1}, ha_{2k_1}, hb_{2k_1})\}, ...$

$\{(wa_{k_2 1}, wb_{k_2 1}, ha_{k_2 1}, hb_{k_2 1}), (wa_{k_2 2}, wb_{k_2 2}, ha_{k_2 2}, hb_{k_2 2}), ..., (wa_{k_2 k_1}, wb_{k_2 k_1}, ha_{k_2 k_1}, hb_{k_2 k_1})\}\}$

En consecuencia, las listas L_A, L_B, L_{AB} y cada una de las sub-listas contenidas en L_{AB}, están clasificadas en orden ascendente de anchura y descendente de altura.

El proceso de construcción de la lista L_{AB} (Figura 4-8) lo efectuamos de la siguiente forma:

- Creamos una lista vacía L_{AB}. A continuación, vamos tomando secuencialmente una a una las implementaciones (wb_i,hb_i) de la lista-R L_B. Por cada una creamos una sub-lista vacía L_{AB_i} y la combinamos con las implementaciones (wa_j,ha_j) de la lista-R L_A.

- El bloque L2 resultante de dicha combinación, lo insertamos en la sub-lista-L2 L_{AB_i} de la forma (wa_j,wb_i,ha_j,hb_i).

- Si la altura del módulo B, hb, es mayor o igual a la altura del módulo A, ha_j, no tendremos en cuenta las demás implementaciones de la lista-R L_A. Hacemos esto porque al estar clasificada en orden ascendente de anchura la lista-R L_B, el resto de elementos de dicha lista-R si los hubiere, serían redundantes con respecto a éste último.

- Si la altura del módulo B, hb_i, es menor que la altura del módulo A, ha_j, tomamos el siguiente elemento de la lista-R L_B si lo hubiere, y así sucesivamente.

De esta forma, la lista-L2 L_{AB} generada mediante el algoritmo AB no tendrá implementaciones redundantes y tendrá como máximo k_2 sub-listas-L2 y en cada una k_1 elementos como máximo.

```
ALGORITMO AB
  CREAR una lista vacía L_AB
  FOR i = 1 TO k_2 DO
      (wb_i, hb_i) = es el elemento i de L_B
      CREAR una sub - lista vacía L_ABi
      FOR j = 1 TO k_1 DO
          (wa_j, ha_j) = es el elemento j de L_A
          AÑADIR en L_ABi el elemento (wa_j, wb_i, ha_j, hb_i)
          IF hb_i < ha_j
            THEN j = j + 1
            ELSE j = k_1
          ENDIF
      ENDFOR
      i = i + 1
  ENDFOR
```

Figura 4-8 Algoritmo AB

4.3 FASE ABC

En la segunda fase de nuestro algoritmo AWO, combinamos cada una de las implementaciones del bloque L2 obtenidas en la fase anterior, con cada una de las implementaciones del módulo C. En la figura 4-9 se representa dicho proceso.

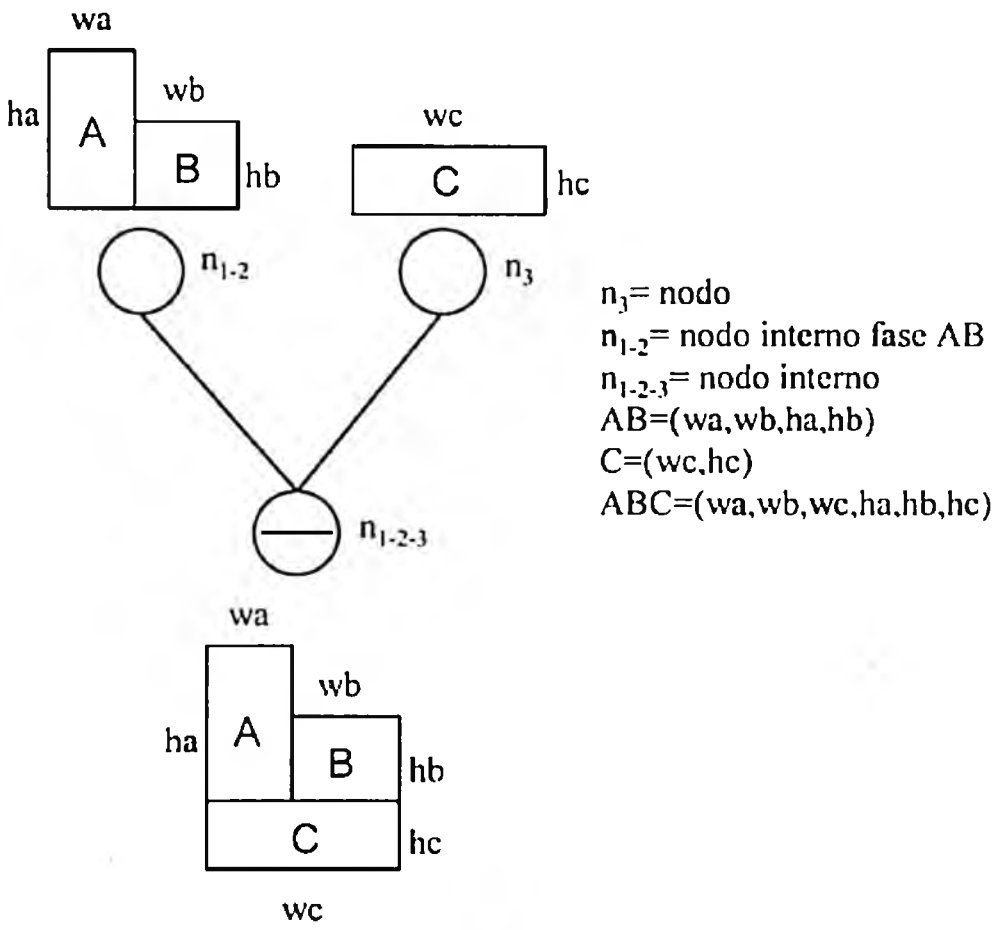


Figura 4-9 Árbol de unión ABC.

T.-C. Wang y Wong plantean en la fase β de su algoritmo OPT la unión de las $k_1 \bullet k_2$ implementaciones obtenidas en la fase anterior, α, con las k_3 implementaciones del módulo C, con lo que obtiene como máximo de $k_1 \bullet k_2 \bullet k_3$ bloques L3. Sin embargo, en la mayoría de los casos no hay que tener en cuenta todos los bloques L3 ya que muchos de ellos son redundantes.

La fase β es similar a nuestra fase ABC. La gran diferencia estriba en que la lista-L2 obtenida en la fase α, contiene en todas o algunas de las sub-listas elementos redundantes con zonas desaprovechadas. Por ello, una vez efectuadas las uniones de los bloques L2 con los correspondientes módulos C, tiene que aplicar otra sub-rutina cuya misión es eliminar todos los bloques L3 redundantes.

Nosotros, al no arrastrar ningún elemento redundante con zonas desaprovechadas, no tenemos que aplicar ninguna sub-rutina adicional. En consecuencia, con nuestro algoritmo AWO el número de nodos visitados así como el número de uniones a realizar es bastante menor y en consecuencia, el tiempo de proceso también es menor.

Toda implementación perteneciente a la lista-L3 que cumpla las siguientes propiedades:

$$wa_i \geq wa_j$$
$$wb_i \geq wb_j$$
$$wc_i \geq wc_j$$
$$ha_i \geq ha_j$$
$$hb_i \geq hb_j$$
$$hc_i \geq Hc_j$$

o lo que es lo mismo:

$$P15: \max(wa_i+wb_i,wc_i) \geq \max(wa_j+wb_j,wc_j)$$
$$P16: \max(ha_i+hc_i,hb_i+hc_i) \geq \max(ha_j+hc_j,hb_j+hc_j)$$

es redundante, y por lo tanto i domina a j.

Por ejemplo, supongamos que las implementaciones del bloque L2 AB son las de la lista siguiente: L_{AB}={{(1,1,6,6)},{(1,2,6,3),(2,2,3,3)}, {(1,3,6,2),(2,3,3,2),(3,3,2,2)},{(1,6,6,1),(2,6,3,1),(3,6,2,1),(6,6,1,1)} y las del módulo C son C={(1,6),(2,3),(3,2),(6,1)}. En este caso, los n elementos de la lista L_{AB} son 10 y los k_3 elementos de la lista L_C son 4. El proceso de unión se puede efectuar de dos formas:

1. Unión de cada una de las implementaciones del módulo C con todas las del bloque L2 AB.

En este caso, se obtendrán los bloques L3 representados en la figura 4-10 y cuya lista-L3 L_{ABC} resultante constará de $k_3 \bullet n = 4 \bullet 10 = 40$ elementos:

L_{ABC}={sub-lista 1, sub-lista 2, sub-lista 3, sub-lista 4}

sub-lista 1={(1,1,1,6,6,6),(1,2,1,6,3,6),(2,2,1,3,3,6),(1,3,1,6,2,6),
(2,3,1,3,2,6),(3,3,1,2,2,6),(1,6,1,6,1,6),(2,6,1,3,1,6),
(3,6,1,2,1,6),(6,6,1,1,1,6)}

sub-lista 2={(1,1,2,6,6,3),(1,2,2,6,3,3),(2,2,2,3,3,3),(1,3,2,6,2,3),
(2,3,2,3,2,3),(3,3,2,2,2,3),(1,6,2,6,1,3),(2,6,2,3,1,3),
(3,6,2,2,1,3),(6,6,2,1,1,3)}

sub-lista 3={(1,1,3,6,6,2),(1,2,3,6,3,2),(2,2,3,3,3,2),(1,3,3,6,2,2),
(2,3,3,3,2,2),(3,3,3,2,2,2),(1,6,3,6,1,2),(2,6,3,3,1,2),
(3,6,3,2,1,2),(6,6,3,1,1,2)}

sub-lista 4={(1,1,6,6,6,1),(1,2,6,6,3,1),(2,2,6,3,3,1),(1,3,6,6,2,1),
(2,3,6,3,2,1),(3,3,6,2,2,1),(1,6,6,6,1,1),(2,6,6,3,1,1),
(3,6,6,2,1,1),(6,6,6,1,1,1)}

La única forma de comprobar si en cada una de las sub-listas-L3 hay implementaciones redundantes es ver si todas ellas cumplen las propiedades P15 y P16.

Todos los elementos marcados con un asterisco (*) en la figura 4-10 son implementaciones redundantes con respecto a otros elementos de la misma sub-lista. Sin embargo, no se pueden eliminar todas las configuraciones ya que algunas de ellas podrían conducir a soluciones finales óptimas. Además todos los elementos no redundantes tienen implícita una zona desaprovechada al ser las alturas de los módulos A y B iguales (debido a la configuración de partida lo ideal es que la altura del módulo A sea mayor que la del B). Por lo tanto, habría que tener en cuenta todas las posibilidades incluidas las redundantes, dando lugar a $k_3 \bullet n$ bloques L3 ABC.

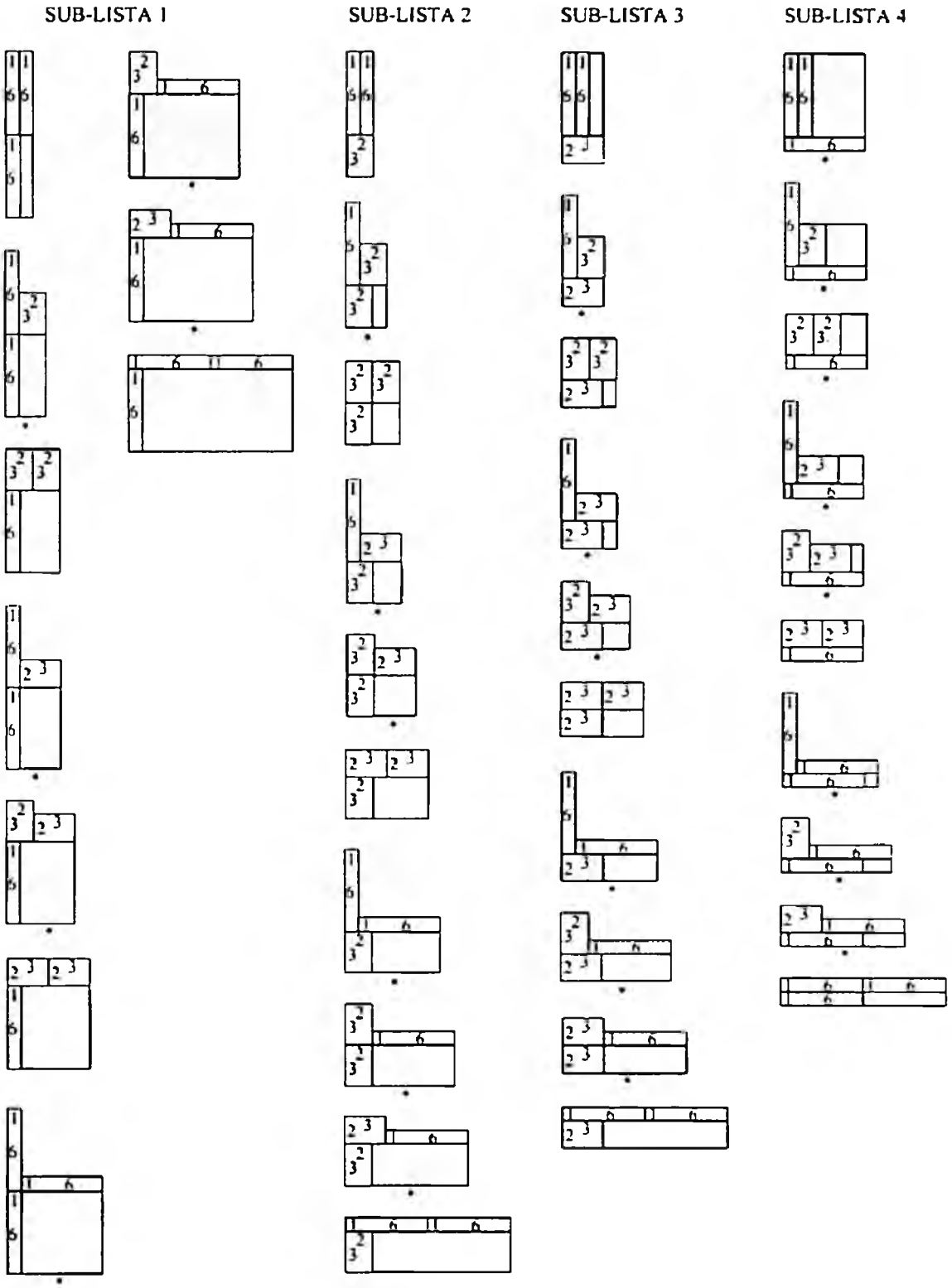


Figura 4-10 Implementaciones en la fase ABC al unir C con AB.

2. Unión de cada una de las implementaciones bloque L2 AB con todas las del módulo C.

En este caso, se obtendrán los bloques L3 representados en la figura 4-11 y la lista-L3 L_{ABC} resultante constará de $n \bullet k_3 = 10 \bullet 4 = 40$ elementos:

L_{ABC}={sub-lista 1, sub-lista 2, sub-lista 3, sub-lista 4}

sub-lista 1={(6,6,6,1,1,1),(6,6,3,1,1,2),(6,6,2,1,1,3),(6,6,1,1,1,6),
(3,6,6,2,1,1),(3,6,3,2,1,2),(3,6,2,2,1,3),(3,6,1,2,1,6),
(2,6,6,3,1,1),(2,6,3,3,1,2),(2,6,2,3,1,3),(2,6,1,3,1,6),
(1,6,6,6,1,1),(1,6,3,6,1,2),(1,6,2,6,1,3),(1,6,1,6,1,6)}

sub-lista 2={(3,3,6,2,2,1),(3,3,3,2,2,2),(3,3,2,2,2,3),(3,3,1,2,2,6),
(2,3,6,3,2,1),(2,3,3,3,2,2),(2,3,2,3,2,3),(2,3,1,3,2,6),
(1,3,6,6,2,1),(1,3,3,6,2,2),(1,3,2,6,2,3),(1,3,1,6,2,6)}

sub-lista 3={(2,2,6,3,3,1),(2,2,3,3,3,2),(2,2,2,3,3,3),(2,2,1,3,3,6),
(1,2,6,6,3,1),(1,2,3,6,3,2),(1,2,2,6,3,3),(1,2,1,6,3,6)}

sub-lista 4={(1,1,6,6,6,1),(1,1,3,6,6,2),(1,1,2,6,6,3),(1,1,1,6,6,6)}

La única forma de comprobar si en cada una de las sub-listas-L3 hay implementaciones redundantes es ver si todas ellas cumplen las propiedades P15 y P16.

Todos los elementos marcados con un asterisco (*) en la figura 4-11 son implementaciones redundantes y en la mayoría de ellos se cumple que la altura del módulo A es igual que la del módulo B y por tanto, en la fase ABCDE al unir los bloques L3 y 7 contendrán todos ellos zonas desaprovechadas.

Nosotros planteamos que dichas implementaciones redundantes se pueden eliminar de la sub-lista correspondiente o lo que es lo mismo no es necesario tenerlas en cuenta ya que nunca mejorarían las soluciones obtenidas al tener ya en esta fase, zonas desaprovechadas.

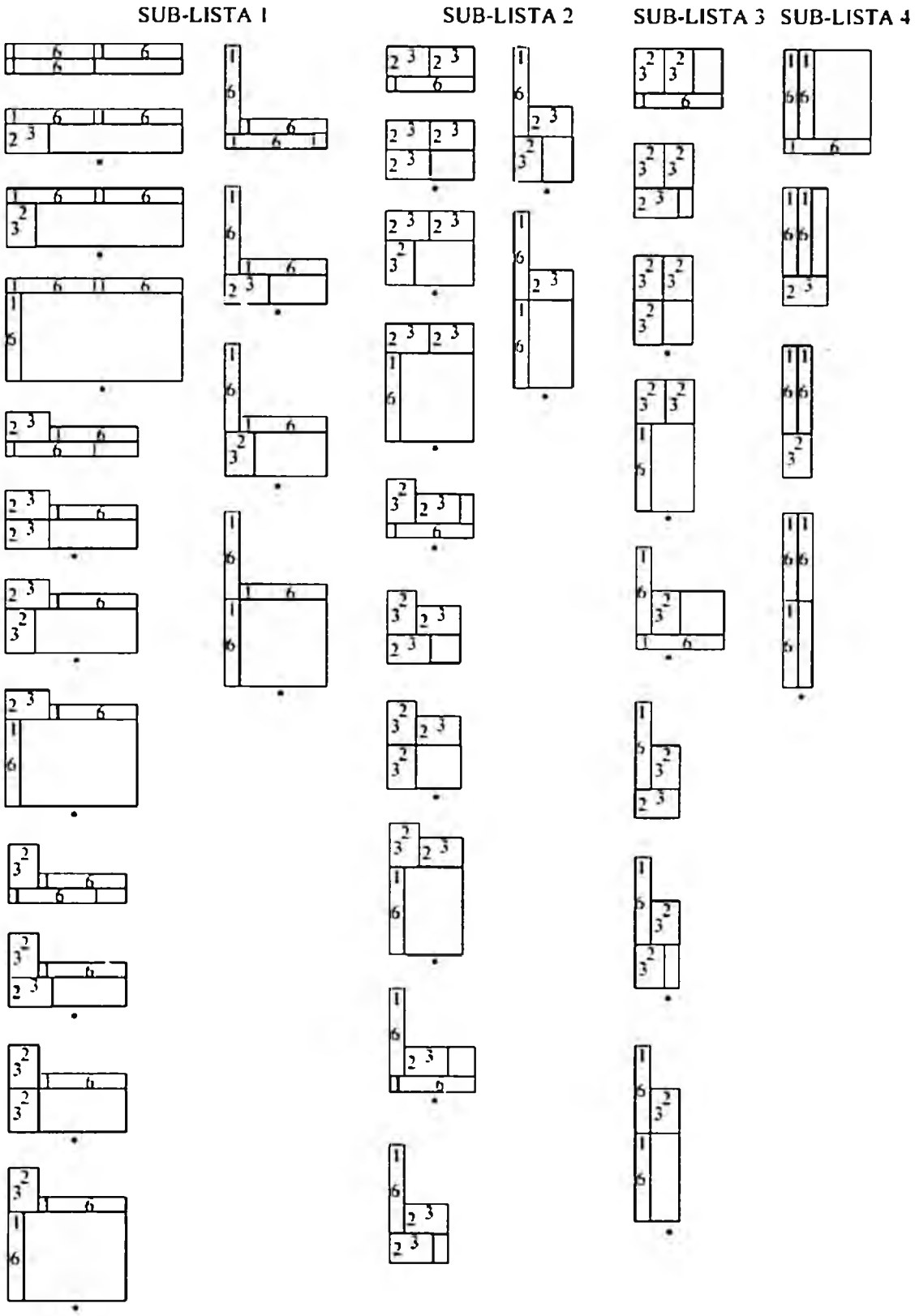


Figura 4-11 Implementaciones en la fase ABC al unir AB con C.

Como hemos visto, en esta fase se producen dos tipos de implementaciones redundantes L3, las que se deben tener en cuenta y las que se deben eliminar, por lo que tenemos que determinar cuáles son las de un tipo y cuáles las del otro.

- Implementaciones redundantes que no se deben eliminar.

  En la figura 4-12, tenemos tres bloques L3 resultantes de la unión de los módulos A, B y C. Los bloques B1 y B2 son redundantes con respecto al B3, al cumplir las propiedades P15 y P16. Sin embargo, aunque tengan zonas desaprovechadas deben tenerse en cuenta ya que, en fases posteriores al unirlos al resto de módulos (D y E) no tienen porque dar lugar a floorplans redundantes.

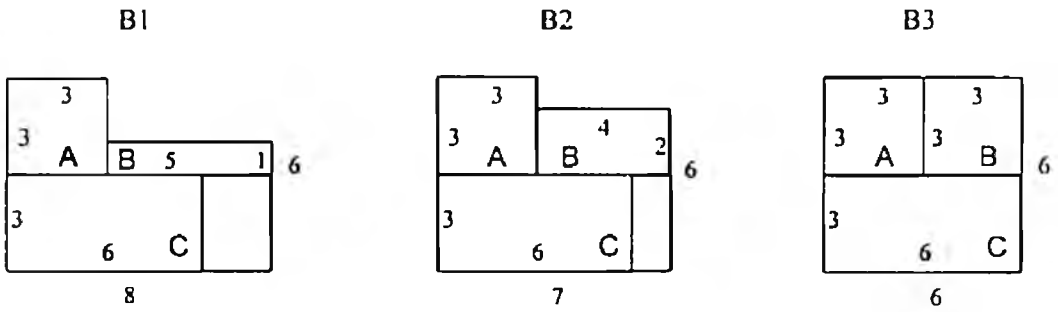


Figura 4-12 Implementaciones ABC redundantes a tener en cuenta.

Así, como podemos observar en la figura 4-13, al unir los bloques B1, B2 y B3 con un mismo bloque que contiene los módulos D y E se generan tres floorplans B4, B5 y B6 que no son redundantes.

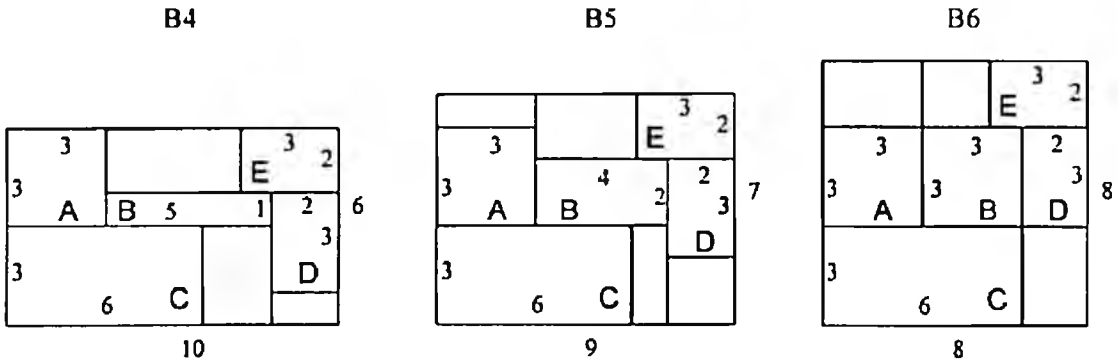


Figura 4-13 Implementaciones AB redundantes que dan lugar a floorplans no redundantes.

- Implementaciones redundantes que se deben eliminar.

En la figura 4-14, tenemos tres bloques L3 resultantes de la unión de los módulos A, B y C. Los bloques B1 y B2 son redundantes con respecto al B3 y al tener zonas desaprovechadas hay que eliminarlos ya que nunca se obtendrían con ellos mejores soluciones que con el bloque B3; en fases posteriores se obtendrían siempre floorplans redundantes al unirlos con el resto de módulos (D y E).

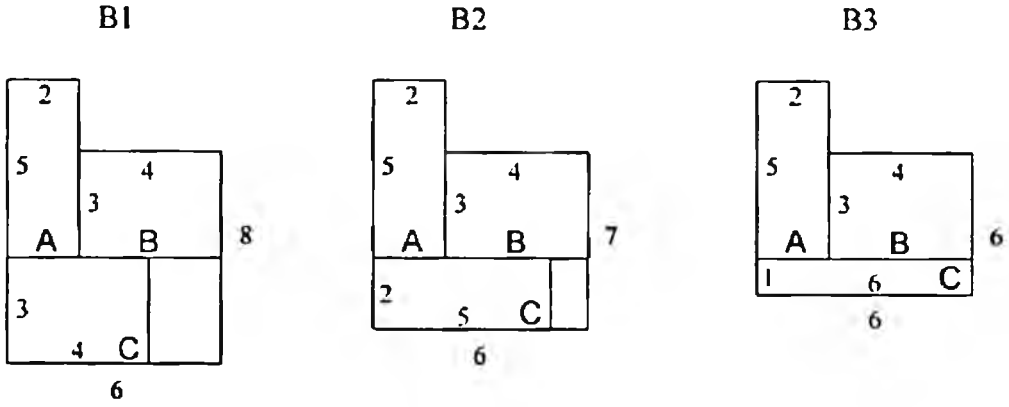


Figura 4-14 Implementaciones ABC redundantes a eliminar.

Así, como podemos observar en la figura 4-15, al unir los bloques B1, B2 y B3 con un mismo bloque que contiene los módulos D y E se generan tres floorplans, B4, B5 y B6, de los cuales dos son redundantes con respecto al B4.

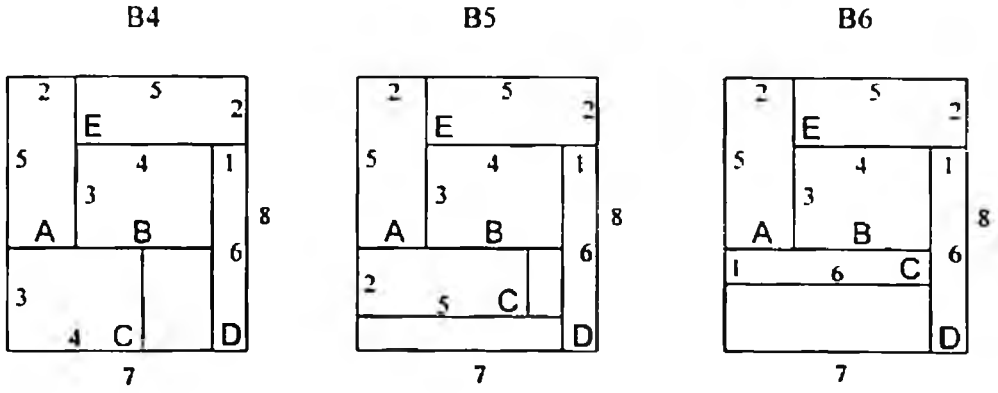


Figura 4-15 Implementaciones AB redundantes que dan lugar a floorplans redundantes.

Para evitar en lo posible elementos redundantes, establecemos en esta fase una relación entre las anchuras de los módulos A y B pertenecientes al bloque L2 AB y la anchura del módulo C:

- Cuando la anchura del módulo A más la anchura del módulo B es menor que la anchura del módulo C (wa+wb<wc), tendremos en cuenta siempre la unión producida. Al unir estos módulos obtenemos una solución que teóricamente no es la perfecta ya que contiene una zona desaprovechada. El bloque L3 resultante contendrá una zona desaprovechada cuya anchura la determinará la diferencia entre el módulo C y el bloque L2 AB (Figura 4-16.a y Figura 4-16.b).

- Cuando la anchura del módulo A más la anchura del módulo B es igual a la del módulo C (wa+wb=wc), tendremos en cuenta la unión producida. Teóricamente ésta sería la solución óptima ya que no hay zonas desaprovechadas (Figura 4-16.c). Sin embargo, esto no implica que en las futuras uniones este bloque L3 sea el óptimo.

- Cuando la anchura del módulo A más la anchura del módulo B es mayor que la anchura del módulo C (wa+wb>wc) eliminaremos la unión generada si es redundante, ya que el bloque L3 obtenido contendrá una zona desaprovechada mayor que la que se tiene con la que es redundante, y la tendremos en cuenta si no es redundante. La anchura de dicha zona la determinará la diferencia entre el bloque L2 AB y el módulo C (Figura 4-16.d, Figura 4-16.e y Figura 4-16.f).

Al suprimir en las sub-listas del ejemplo anteriormente citado, los elementos redundantes, que contiene zonas desaprovechadas la lista-L3 L_{ABC} resultante sería:

L_{ABC}={{(6,6,6,1,1,1),(3,6,6,2,1,1),(2,6,6,3,1,1),(1,6,6,6,1,1)},{(3,3,6,2,2,1),
(2,3,3,3,2,2),(1,3,3,6,2,2)},{(2,2,6,3,3,1),(2,2,3,3,3,2),(1,2,3,6,3,2)},
{(1,1,6,6,6,1),(1,1,3,6,6,2),(1,1,2,6,6,3)}

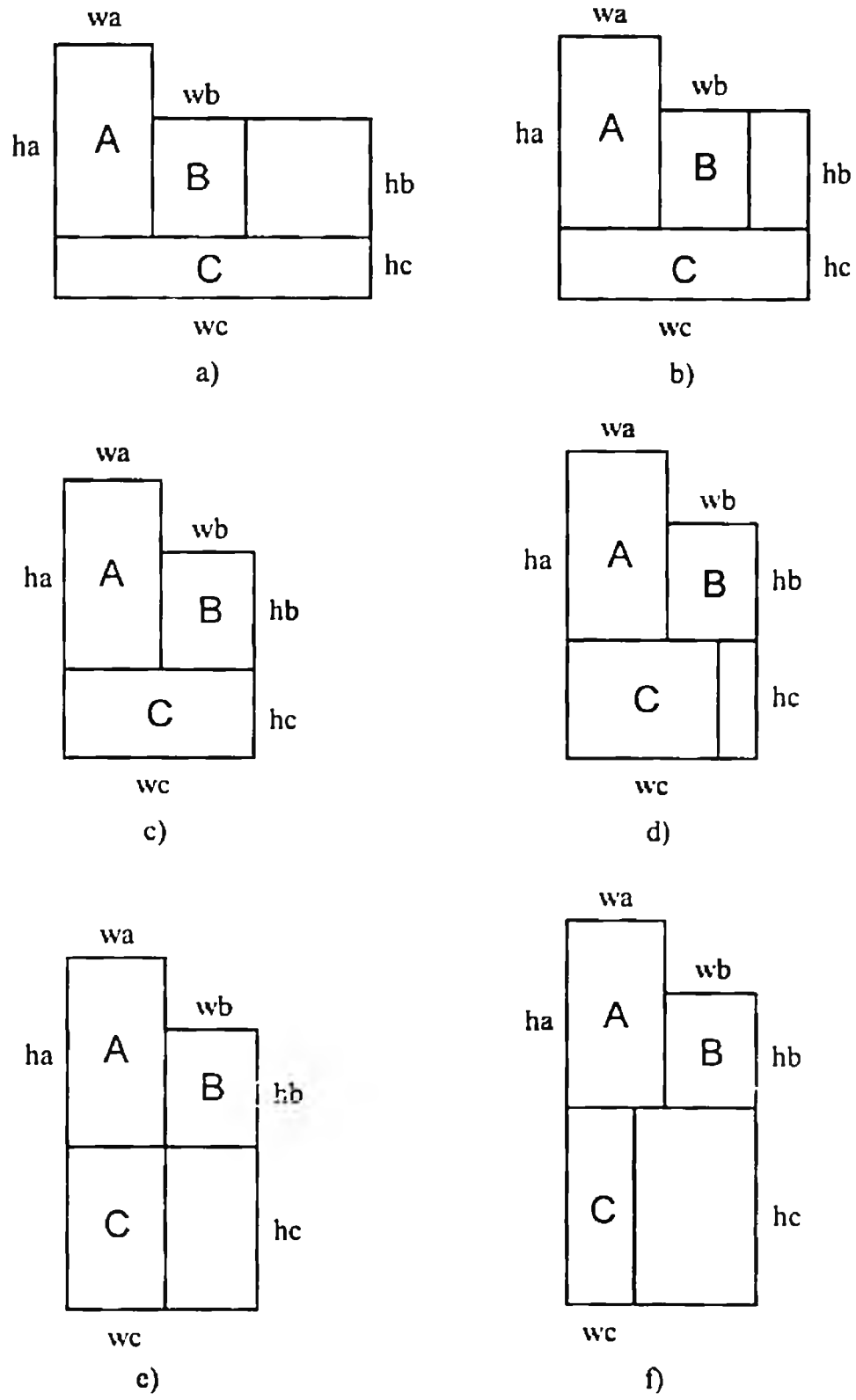


Figura 4-16 Implementaciones producidas en la fase ABC.

Nuestro algoritmo combina, cada una de las implementaciones de las sub-listas-L2, incluidas en la lista-L2 $L_{AB}=\{L_{AB_1},L_{AB_2}...L_{AB_{k_2}}\}=\{(wa_j,wb_j,ha_j,hb_j)\ |1 \le j \le k_1\}$ con cada una de las implementaciones de la lista-R, correspondientes al módulo C, $L_C=\{(wc_i,hc_i)|\ 1 \le i \le k_3\}$, de la siguiente forma:

$$L_{AB}=\{L_{AB_1},L_{AB_2},....L_{AB_{k_2}}\}=$$

$$\{\{(wa_{11},wb_{11},ha_{11},hb_{11}),(wa_{12},wb_{12},ha_{12},hb_{12}),....,(wa_{1k_1},wb_{1k_1},ha_{1k_1},hb_{1k_1})\},$$

$$\{(wa_{21},wb_{21},ha_{21},hb_{21}),(wa_{22},wb_{22},ha_{22},hb_{22}),....,(wa_{2k_1},wb_{2k_1},ha_{2k_1},hb_{2k_1})\},.....$$

$$....,\{(wa_{k_21},wb_{k_21},ha_{k_21},hb_{k_21}),(wa_{k_21},wb_{k_21},ha_{k_21},hb_{k_21}),....,(wa_{k_2k_1},wb_{k_2k_1},ha_{k_2k_1},hb_{k_2k_1})\}\}$$

$$L_C=\{(wc_1,hc_1),(wc_2,hc_2),....,(wc_{k_3},hc_{k_3})\}$$

$$(wa_{11},wb_{11},ha_{11},hb_{11})\Theta(wc_1,hc_1)=(wa_{111},wb_{111},wc_{111},ha_{111},hb_{111},hc_{111})$$

$$(wa_{11},wb_{11},ha_{11},hb_{11})\Theta(wc_2,hc_2)=(wa_{112},wb_{112},wc_{112},ha_{112},hb_{112},hc_{112})$$

....

$$(wa_{11},wb_{11},ha_{11},hb_{11})\Theta(wc_{k_3},hc_{k_3})=(wa_{11k_3},wb_{11k_3},wc_{11k_3},ha_{11k_3},hb_{11k_3},hc_{11k_3})$$

$$(wa_{12},wb_{12},ha_{12},hb_{12})\Theta(wc_1,hc_1)=(wa_{121},wb_{121},wc_{121},ha_{121},hb_{121},hc_{121})$$

$$(wa_{12},wb_{12},ha_{12},hb_{12})\Theta(wc_2,hc_2)=(wa_{122},wb_{122},wc_{122},ha_{122},hb_{122},hc_{122})$$

....

$$(wa_{12},wb_{12},ha_{12},hb_{12})\Theta(wc_{k_3},hc_{k_3})=(wa_{12k_3},wb_{12k_3},wc_{12k_3},ha_{12k_3},hb_{12k_3},hc_{12k_3})$$

....

$$(wa_{k_2k_1},wb_{k_2k_1},ha_{k_2k_1},hb_{k_2k_1})\Theta(wc_1,hc_1)=(wa_{k_2k_11},wb_{k_2k_11},wc_{k_2k_11},ha_{k_2k_11},hb_{k_2k_11},hc_{k_2k_11})$$

$$(wa_{k_2k_1},wb_{k_2k_1},ha_{k_2k_1},hb_{k_2k_1})\Theta(wc_2,hc_2)=(wa_{k_2k_12},wb_{k_2k_12},wc_{k_2k_12},ha_{k_2k_12},hb_{k_2k_12},hc_{k_2k_12})$$

....

$$(wa_{k_2k_1},wb_{k_2k_1},ha_{k_2k_1},hb_{k_2k_1})\Theta(wc_{k_3},hc_{k_3})=(wa_{k_2k_1k_3},wb_{k_2k_1k_3},wc_{k_2k_1k_3},ha_{k_2k_1k_3},hb_{k_2k_1k_3},hc_{k_2k_1k_3})$$

donde k_3 representa el número de implementaciones contenidas en la lista L_C, k_2 el número de sub-listas-L2 integradas en L_{AB} y k_1 el número de implementaciones en cada una de estas sub-listas.

Las listas-L2 L_{AB} satisfacen las propiedades siguientes:

P1 : $wa_i \le wa_j$

P2 : $wb_i \le wb_j$

P3 : $ha_i \ge ha_j$

P4 : $hb_i \ge hb_j$

y la lista-R L_C las siguientes:

$$P5 : wc_i \leq wc_j$$

$$P6 : hc_i \geq hc_j$$

Por tanto, todas las implementaciones de cada una de las sub-listas-L3 resultantes de la unión de AB y C pertenecientes a la lista-L3 $L_{ABC}=\{L_{ABC_1}, L_{ABC_2}, \ldots, L_{ABC_{k_2}}\}$ cumplen las propiedades siguientes:

$$P1 : wa_i \leq wa_j$$

$$P2 : wb_i \leq wb_j$$

$$P5 : wc_i \leq wc_j$$

$$P3 : ha_i \geq ha_j$$

$$P4 : hb_i \geq hb_j$$

$$P6 : hc_i \geq hc_j$$

con lo que la lista-L3 L_{ABC} estará clasificada en orden creciente de anchuras y en orden decreciente de alturas.

Esta lista tendrá un conjunto de k_2 sub-listas-L3 como máximo y cada una de ellas contendrá un máximo de k_3 elementos.

$$L_{ABC} = \{L_{ABC_1}, L_{ABC_2}, \ldots, L_{ABC_{k_2}}\} =$$
$$\{\{(wa_{111}, wb_{111}, wc_{111}, ha_{111}, hb_{111}, hc_{111}), (wa_{112}, wb_{112}, wc_{112}, ha_{112}, hb_{112}, hc_{112}) \ldots$$
$$(wa_{11k_3}, wb_{11k_3}, wc_{11k_3}, ha_{11k_3}, hb_{11k_3}, hc_{11k_3})\}, \{(wa_{211}, wb_{211}, wc_{211}, ha_{211}, hb_{211}, hc_{211}),$$
$$(wa_{212}, wb_{212}, wc_{212}, ha_{212}, hb_{212}, hc_{212}) \ldots (wa_{21k_3}, wb_{21k_3}, wc_{21k_3}, ha_{21k_3}, hb_{21k_3}, hc_{21k_3})\} \ldots$$
$$\{(wa_{k_2k_31}, wb_{k_2k_31}, wc_{k_2k_31}, ha_{k_2k_31}, hb_{k_2k_31}, hc_{k_2k_31}), (wa_{k_2k_32}, wb_{k_2k_32}, wc_{k_2k_32}, ha_{k_2k_32}, hb_{k_2k_32}, hc_{k_2k_32}),$$
$$\ldots (wa_{k_2k_3k_3}, wb_{k_2k_3k_3}, wc_{k_2k_3k_3}, ha_{k_2k_3k_3}, hb_{k_2k_3k_3}, hc_{k_2k_3k_3})\}\}$$

Representamos las implementaciones por un sexteto de la forma (wa,wb,wc,ha,hb,hc), donde wa, wb y wc son las anchuras y ha, hb y hc son las alturas de los módulos A, B y C respectivamente. Esta información de los distintos módulos, se utiliza para poder eliminar las implementaciones que pueden dar lugar a floorplans redundantes con zonas desaprovechadas en futuras uniones.

Por lo tanto, planteamos el proceso de unión de los bloques L_{ABC} (Figura 4-17) de la siguiente forma:

- Creamos una lista vacía L_{ABC} que contendrá como máximo tantas sub-listas-L3 como sub-listas-L2 tenga la lista-L2 AB. A continuación, por cada elemento de la sub-lista L_{AB} creamos una sub-lista vacía L_{ABC_k} y realizamos la unión de los bloques de forma secuencial en orden decreciente de anchura y en orden creciente de altura. Es decir, efectuamos la unión entre el bloque L2 y el módulo C comenzando desde la última sub-lista-L2 y dentro de ella desde la última de las implementaciones, hasta la primera y desde el último elemento hasta el primero.

- El bloque L3 resultante de dicha combinación, lo insertamos en la sub-lista-L3 L_{ABC_k} de la forma $(wa_i, wb_i, wc_j, ha_i, hb_i, hc_j)$.

- Si la anchura del módulo A más la del módulo B es menor que la anchura del módulo C ($wa_i+wb_i<wc_j$), nunca tendrá en esta lista o sub-lista otra implementación redundante. A continuación, debemos efectuar la unión del mismo bloque L2 AB con el siguiente elemento del módulo C. Las implementaciones del módulo C al cumplir las propiedades P5 y P6 están clasificadas en orden creciente de anchuras y en orden decreciente de alturas. Por tanto, al unir el siguiente elemento de C con el mismo bloque L2 AB puede dar lugar a un bloque L3 ABC sin zonas desaprovechadas y teóricamente dar lugar a una solución óptima.

- Cuando la anchura del módulo A más la del módulo B es igual a la del módulo C ($wa_i+wb_i=wc_j$), tenemos un hipotético floorplan óptimo ya que al producirse la unión perfecta entre los distintos módulos no se obtienen zonas desaprovechadas. Siempre que efectuamos una unión de un bloque L2 AB con un módulo rectangular C y se cumple que $wa_i+wb_i=wc_j$, el bloque L3 obtenido nunca tendrá en esta sub-lista otro bloque L3 que sea redundante ya que siempre cumplirán las propiedades P1, P2, P5, P3, P4 y P6. Sin embargo, nunca consideraremos la unión del mismo bloque L2 AB con la siguiente

implementación del bloque rectangular C ya que la unión producida, al cumplir las propiedades P15 y P16, será siempre redundante (Figura 4-16.d, Figura 4-16.e y Figura 4-16.f). Por tanto, la próxima unión será entre la siguiente implementación del módulo L2 AB y la posterior implementación del módulo C.

- Cuando la anchura del módulo A más la del módulo B es mayor que la anchura del módulo C ($wa_i+wb_i>wc_j$) obtenemos una solución que tiene una zona desaprovechada. Este caso nunca se puede producir si anteriormente se produjo una unión en la que $wa_i+wb_i=wc_j$, pero si si se produjo una unión en la que $wa_i+wb_i<wc_j$. Por ello, siempre que efectuemos una unión de un bloque L2 AB con un módulo rectangular C, si $wa_i+wb_i>wc_j$, el bloque L3 obtenido nunca tendrá en esta sub-lista otra implementación que sea redundante ya que siempre cumplirán las propiedades P1, P2, P5, P3, P4 y P6.

  Como la unión del mismo bloque L2 AB con la siguiente implementación del bloque rectangular C cumple las propiedades P15 y P16 será siempre redundante al tener la misma anchura y una altura mayor nunca la consideramos. Los módulos de la figura 4-16.e y de la figura 4-16.f son redundantes con respecto al de la figura 4-16.d. Por tanto, la siguiente unión será la siguiente implementación del módulo L2 AB con la última implementación del módulo C.

De esta forma, la lista-L3 L_{ABC} generada mediante el algoritmo ABC no tendrá implementaciones redundantes; tendrá k_2 sub-listas-L3 como máximo y en cada una k_3 elementos como máximo.

ALGORITMO ABC

CREAR una lista vacía L_{ABC}
FOR k = 1 TO k_2 DO
  ACCEDER a la sub-lista L_{AB_k} de la lista L_{AB}
  CREAR una sub-lista vacía L_{ABC_k}
  FOR i = n TO 1 DO
    FOR j = k_3 TO 1 DO
      (wa_i, wb_i, ha_i, hb_i) = es el elemento i de L_{AB_k}
      (wc_j, hc_j) = es el elemento j de L_C
      AÑADIR en L_{ABC_k} (wa_i, wb_i, wc_j, ha_i, hb_i, hc_j)
      IF $wa_i + wb_i \geq wc_j$
        THEN IF i = 1
          THEN j = 1
          ELSE i = i - 1
        ENDIF
      ENDIF
      IF $wa_i + wb_i \leq wc_j$
        THEN j = j - 1
      ENDIF
    ENDFOR
    i = 1
  ENDFOR
  k = k + 1
ENDFOR

Figura 4-17 Algoritmo ABC.

4.4 FASE DE

En esta fase combinamos cada una de las implementaciones del módulo D con las del módulo E obteniendo un conjunto de bloques 7 DE. En la figura 4-18 se representa dicho proceso.

K.Wang y Chen plantean en su algoritmo la unión de las k_4 implementaciones del módulo D con las k_5 implementaciones del módulo E, obteniendo siempre $k_4 \bullet k_5$ bloques 7. Sin embargo, este planteamiento puede provocar, en la mayoría de los casos, el arrastre de una o varias implementaciones redundantes con zonas desaprovechadas en cada una de las sub-listas-7 generadas en DE.

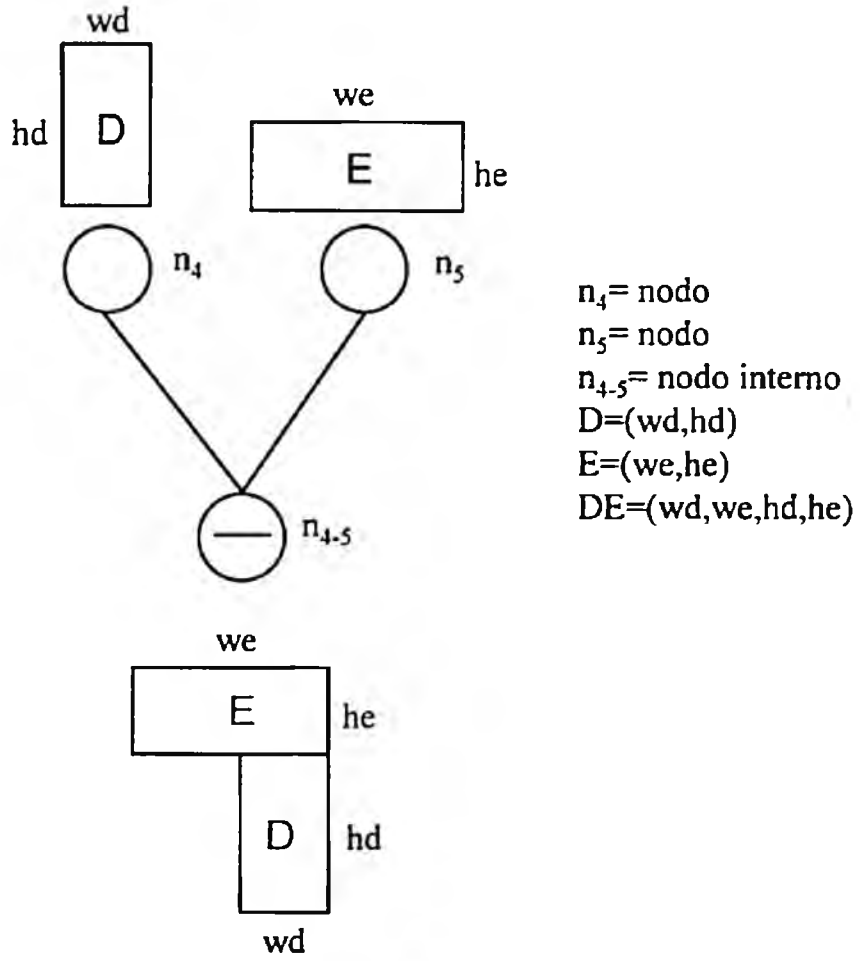


Figura 4-18 Árbol de unión DE.

Cada uno de estos elementos redundantes con zonas desaprovechadas generarían, a su vez en los pasos posteriores, otros bloques también redundantes que contendrían zonas desaprovechadas. Lo único que se conseguiría de esta manera, sería la ralentización del proceso de obtención de un floorplan no particionado óptimo en área. Por esta razón, nosotros eliminamos lo antes posible cualquier elemento redundante con zonas desaprovechadas.

Toda implementación contenida en cada una de las sub-listas-7 es redundante si cumple las siguientes propiedades:

P17: $\max(wd_i, we_i) \geq \max(wd_j, we_j)$

P18: $hd_i + he_i \geq hd_j + he_j$

Por ejemplo, supongamos que las implementaciones de los módulos D y E son: D={(1,6),(2,3),(3,2),(6,1)} y E={(1,6),(2,3),(3,2),(6,1)}. El proceso de unión se puede efectuar de dos formas:

1. Unión de cada una de las implementaciones del módulo E con todas las del módulo D.

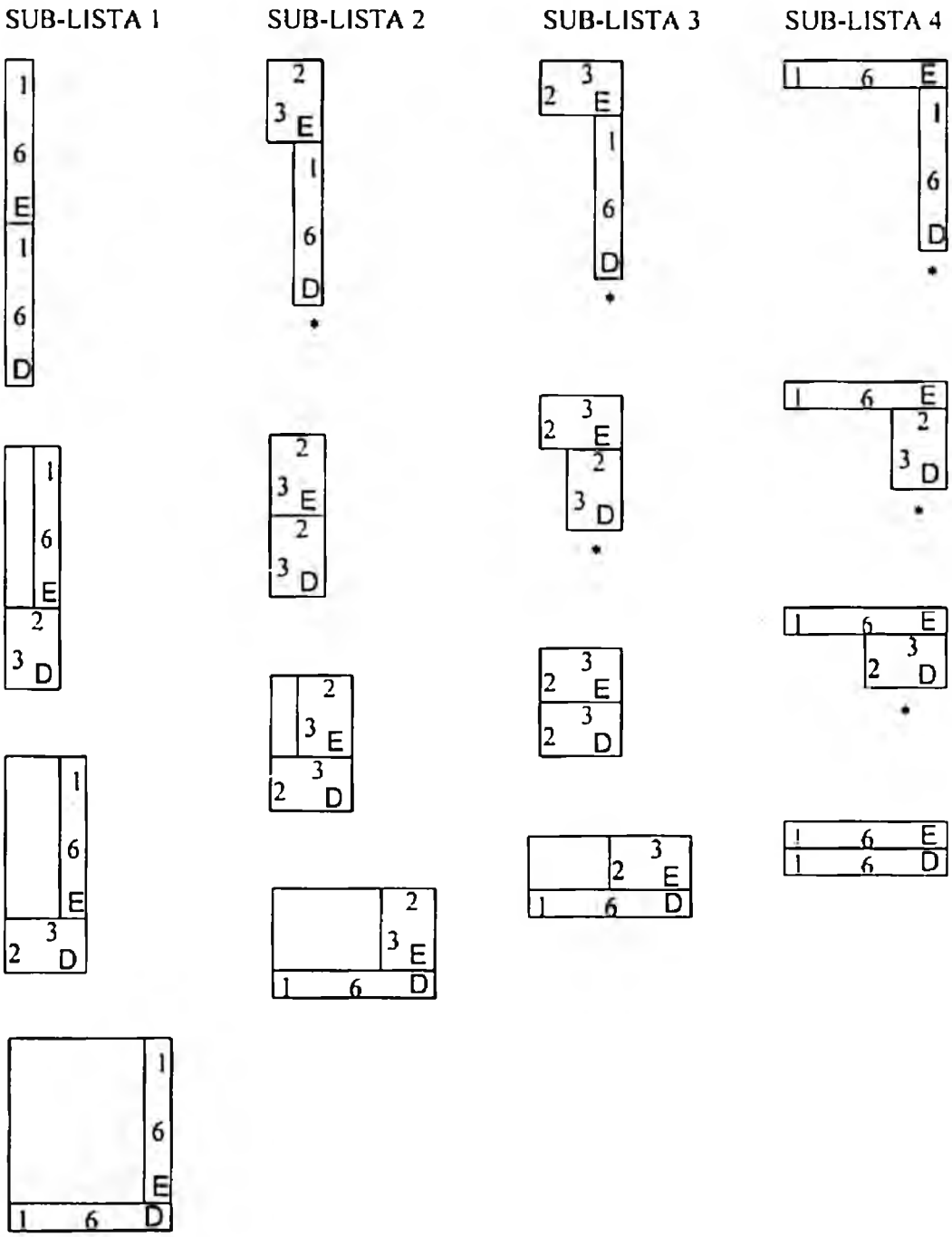


Figura 4-19 Implementaciones en la fase DE al unir E con D.

En este caso se obtendrían los bloques 7 representados en la figura 4-14 y la lista-7 L_{DE} resultante constarían de $k_4 \bullet k_5 = 4 \bullet 4 = 16$ elementos:

L_{DE}={sub-lista 1, sub-lista 2, sub-lista 3, sub-lista 4}

sub-lista 1={(1,1,6,6),(2,1,3,6),(3,1,2,6),(6,1,1,6)}

sub-lista 2={(1,2,6,3),(2,2,3,3),(3,2,2,3),(6,2,1,3)}

sub-lista 3={(1,3,6,2),(2,3,3,2),(3,3,2,2),(6,3,1,2)}

sub-lista 4={(1,6,6,1),(2,6,3,1),(3,6,2,1),(6,6,1,1)}

La única forma de comprobar si en cada una de las sub-listas-7 hay implementaciones redundantes es ver si todas ellas cumplen las propiedades P17 y P18. Todos los elementos marcados con un asterisco (*) en la figura 4-19 son implementaciones redundantes con respecto a otros elementos de la misma sub-lista. Sin embargo, dichas configuraciones, al no contener ninguna zona desaprovechada, no se pueden eliminar ya que podrían conducir a a soluciones finales óptimas. Por lo tanto, habría que tener en cuenta todas las posibilidades incluidas las redundantes, dando lugar a $k_4 \bullet k_5$ bloques 7 DE.

El algoritmo planteado por K.Wang y Chen lleva a cabo la unión de esta forma, acarreando todos los elementos (redundantes y no redundantes con zonas desaprovechadas o sin ellas), por lo que el proceso de obtención de la solución óptima es más largo.

2. Unión de cada una de las implementaciones del módulo D con todas las del módulo E.

   En este caso se obtendrían los bloques 7 representados en la figura 4-15 y la lista-7 L_{DE} resultante constarían de $k_4 \bullet k_5 = 4 \bullet 4 = 16$ elementos:

L_{DE}={sub-lista 1, sub-lista 2, sub-lista 3, sub-lista 4}

sub-lista 1={(1,1,6,6),(1,2,6,3),(1,3,6,2),(1,6,6,1)}

sub-lista 2={(2,1,3,6),(2,2,3,3),(2,3,3,2),(2,6,3,1)}

sub-lista 3={(3,1,2,6),(3,2,2,3),(3,3,2,2),(3,6,2,1)}

sub-lista 4={(6,1,1,6),(6,2,1,3),(6,3,1,2),(6,6,1,1)}

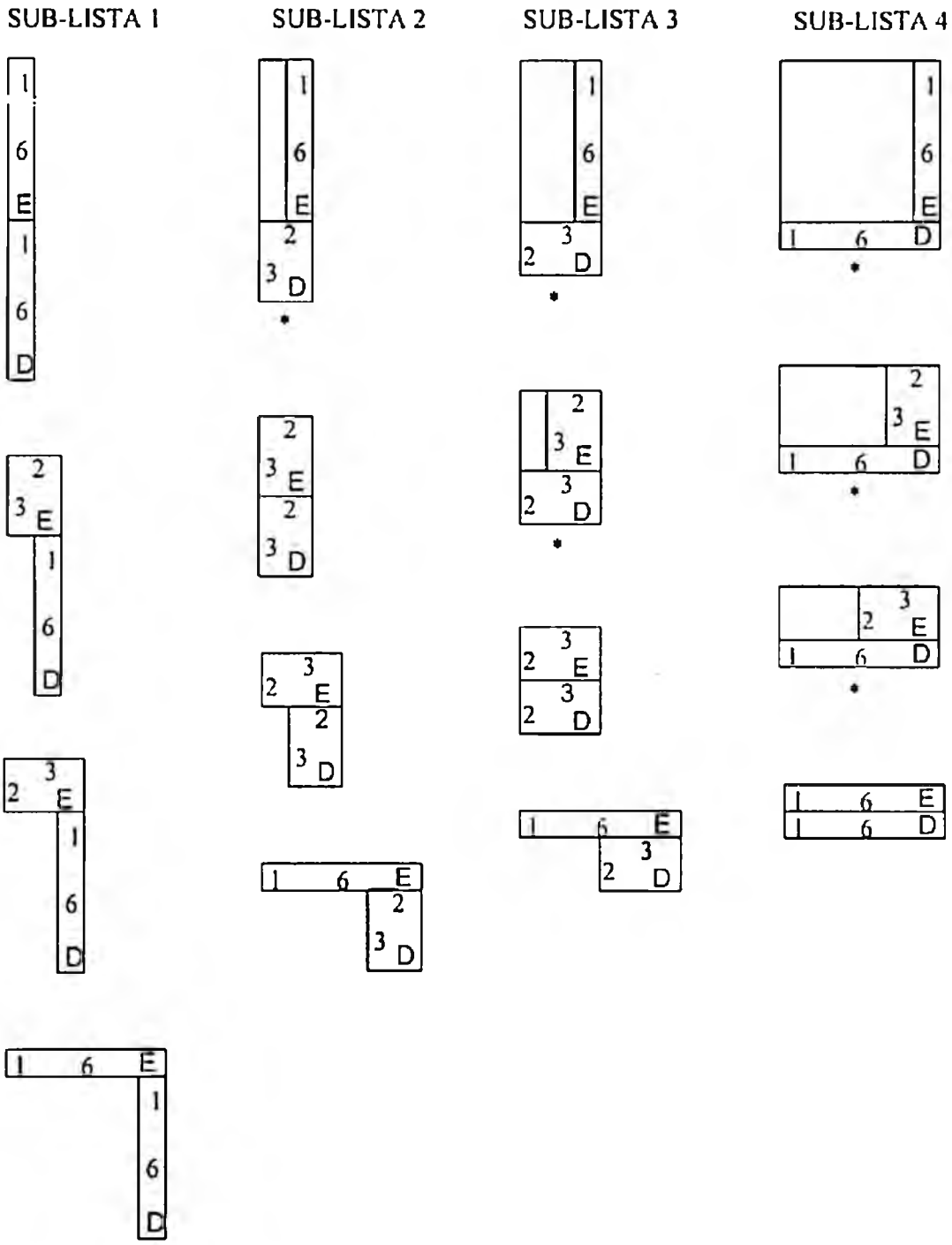


Figura 4-20 Implementaciones en la fase DE al unir D con E.

La única forma de comprobar si en cada una de las sub-listas 7 hay implementaciones redundantes es ver si todas ellas cumplen las propiedades P17 y P18.

Todos los elementos marcados con un asterisco (*) en la figura 4-20 son implementaciones redundantes y en todos ellos se cumple que la anchura del módulo E es menor que la del módulo D, por lo que contienen zonas desaprovechadas.

Nosotros planteamos que dichas implementaciones redundantes obtenidas se pueden eliminar de la sub-lista correspondiente, o lo que es lo mismo no es necesario tenerlas en cuenta ya que nunca mejorarán las soluciones obtenidas al tener ya en esta fase, zonas desaprovechadas.

Como hemos visto, en esta fase se producen dos tipos de implementaciones redundantes 7, las que se deben tener en cuenta y las que se deben eliminar. El mayor problema estriba en determinar cuáles son las de un tipo y cuáles las del otro.

- Implementaciones redundantes que no se deben eliminar.

  En la figura 4-21, se muestran tres bloques 7 resultantes de la unión de los módulos D y E. Los bloques B2 y B3 son redundantes con respecto al B1, al cumplir las propiedades P17 y P18. Sin embargo, al no tener zonas desaprovechadas todos ellos deben tenerse en cuenta, ya que en fases posteriores al unirlos con el resto de módulos (A, B y C) no tienen por qué dar lugar a floorplans redundantes.

  Así, como podemos observar en la figura 4-22, al unir los bloques B1, B2 y B3 con un mismo bloque que contiene los módulos A, B y C se generan tres floorplans B4, B5 y B6 que no son redundantes.

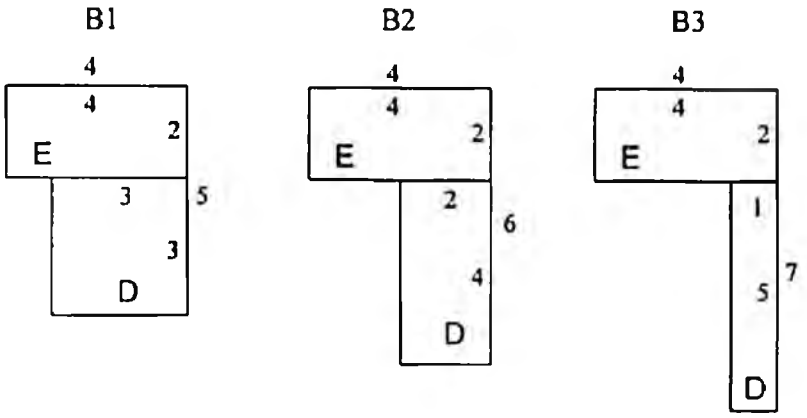


Figura 4-21 Implementaciones DE redundantes a tener en cuenta.

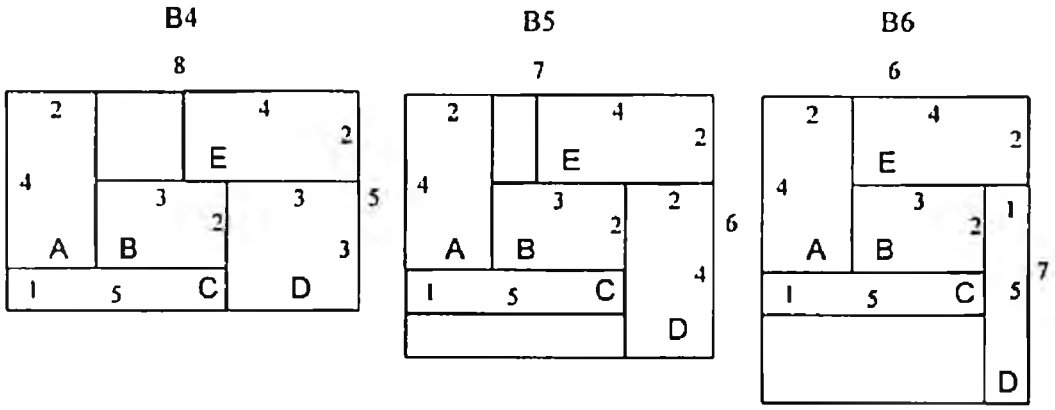


Figura 4-22 Implementaciones DE redundantes que dan lugar a floorplans no redundantes.

- Implementaciones redundantes que deben eliminarse.

  En la figura 4-23, tenemos tres bloques 7 resultantes de la unión de los módulos D y E. Los bloques B2 y B3 son redundantes con respecto al B4 y al tener zonas desaprovechadas hay que eliminarlos ya que nunca se obtendrán con ellos mejores soluciones que las que se puedan conseguir con el bloque B1; en fases posteriores se obtendrían siempre floorplans redundantes al unirlos con el resto de módulos (A, B y C).

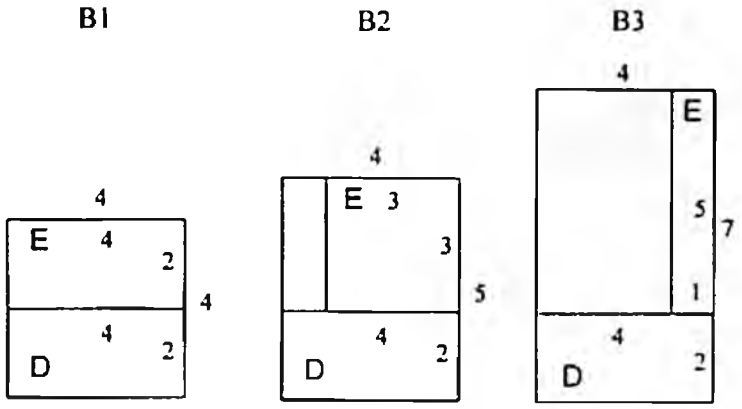


Figura 4-23 Implementaciones DE redundantes a eliminar.

Así, como podemos observar en la figura 4-24, al unir los bloques B1, B2 y B3 con un mismo bloque que contiene los módulos A, B y C se generan tres floorplans B4, B5 y B6, de los cuales dos son redundantes con respecto al B4.

Como la obtención del bloque 7 a través de la unión de un módulo D con uno E se tiene que realizar manteniendo la topología dada, establecemos una relación entre las anchuras de los módulos D y E para evitar en lo posible zonas desaprovechadas:

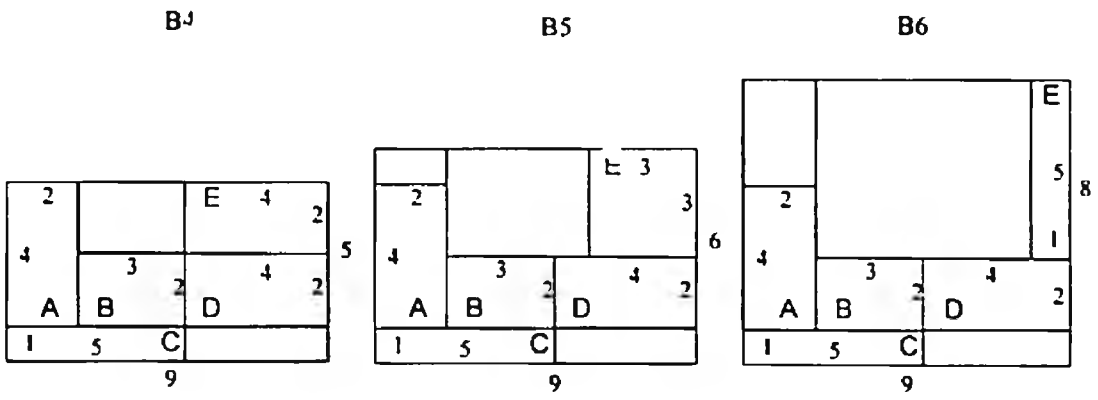


Figura 4-24 Implementaciones DE redundantes que dan lugar a floorplans redundantes.

- Cuando la anchura del módulo E sea mayor que la del módulo D (we>wd), siempre tendremos en cuenta la unión producida, aunque sea redundante, ya que no contiene zonas desaprovechadas. Teóricamente, ésta sería la solución óptima, aunque la unión en la fase posterior puede dar lugar a un bloque que no lo sea.

- Cuando la anchura del módulo E sea igual que la del módulo D (we=wd), tendremos en cuenta siempre la unión obtenida aunque sea redundante, pero no así, el resto de implementaciones que contengan zonas desaprovechadas. El bloque 7 resultante contendrá una zona desaprovechada cuya anchura es la del módulo B.

- Cuando la anchura del módulo E sea menor que la del módulo D (we<wd), eliminaremos la unión generada si es redundante, ya que el bloque 7 generado contendrá una zona desaprovechada mayor que la que se obtiene con el redundante. La anchura de la zona desaprovechada será de la diferencia de anchuras entre D y E más la anchura del módulo B.

Al suprimir de las sub-listas los elementos redundantes seleccionados, la lista-7 L_{DE} resultante quedaría:

$$L_{DE}=\{\{(1,1,6,6),(1,2,6,3),(1,3,6,2),(1,6,6,1)\},\{(2,2,3,3),(2,3,3,2),(2,6,3,1)\},\{(3,3,2,2),(3,6,2,1)\},\{(6,6,1,1)\}\}$$

Como vemos en este ejemplo, de los 16 elementos resultantes hay 6 que se pueden eliminar.

Sean dos módulos, D y E, cuyas implementaciones k_4 y k_5 están recogidas en las listas-R, $L_D=\{(wd_i,hd_i) \mid 1 \le i \le k_4\}$ y $L_E=\{(we_j,he_j) \mid 1 \le j \le k_5\}$.

$$L_D = \{(wd_1,hd_1),(wd_2,hd_2),\ldots,(wd_{k_4},hd_{k_4})\}$$
$$L_E = \{(we_1,he_1),(we_2,he_2),\ldots,(we_{k_5},he_{k_5})\}$$

$(we_1, he_1) \Theta (wd_1, hd_1) = (wd_{11}, we_{11}, hd_{11}, he_{11})$
$(we_1, he_1) \Theta (wd_2, hd_2) = (wd_{12}, we_{12}, hd_{12}, he_{12})$
....
$(we_1, he_1) \Theta (wd_{k_4}, hd_{k_4}) = (wd_{1k_4}, we_{1k_4}, hd_{1k_4}, he_{1k_4})$

$(we_2, he_2) \Theta (wd_1, hd_1) = (wd_{21}, we_{21}, hd_{21}, he_{21})$
$(we_2, he_2) \Theta (wd_2, hd_2) = (wd_{22}, we_{22}, hd_{22}, he_{22})$
....
$(we_2, he_2) \Theta (wd_{k_4}, hd_{k_4}) = (wd_{2k_4}, we_{2k_4}, hd_{2k_4}, he_{2k_4})$

....
$(we_{k_3}, he_{k_3}) \Theta (wd_1, hd_1) = (wd_{k_3,1}, we_{k_3,1}, hd_{k_3,1}, he_{k_3,1})$
$(we_{k_3}, he_{k_3}) \Theta (wd_2, hd_2) = (wd_{k_3,2}, we_{k_3,2}, hd_{k_3,2}, he_{k_3,2})$
....
$(we_{k_3}, he_{k_3}) \Theta (wd_{k_4}, hd_{k_4}) = (wd_{k_3,k_4}, we_{k_3,k_4}, hd_{k_3,k_4}, he_{k_3,k_4})$

$L_{DE} = \{L_{DE_1}, L_{DE_2}, \ldots L_{DE_{k_3}}\} =$
$\{\{(wd_{11}, we_{11}, hd_{11}, he_{11}), (wd_{12}, we_{12}, hd_{12}, he_{12}), \ldots (wd_{1k_4}, we_{1k_4}, hd_{1k_4}, he_{1k_4})\},$
$\{(wd_{21}, we_{21}, hd_{21}, he_{21}), (wd_{22}, we_{22}, hd_{22}, he_{22}), \ldots (wd_{2k_4}, we_{2k_4}, hd_{2k_4}, he_{2k_4})\}, \ldots$
$\{(wd_{k_3,1}, we_{k_3,1}, hd_{k_3,1}, he_{k_3,1}), (wd_{k_3,2}, we_{k_3,2}, hd_{k_3,2}, he_{k_3,2}), \ldots (wd_{k_3,k_4}, we_{k_3,k_4}, hd_{k_3,k_4}, he_{k_3,k_4})\}\}$

La fase DE genera una lista-7 que contiene k_4 sub-listas como máximo y cada una de ellas un máximo de k_5 implementaciones. Las listas-R L_D y L_E satisfacen las propiedades siguientes:

P11 : $w_i \leq w_j$

P12 : $h_i \geq h_j$

Por tanto, todas las implementaciones de cada una de las sub-listas-7 resultantes de la unión de D y E pertenecientes a la lista-7 $L_{DE} = \{L_{DE_1}, L_{DE_2} \ldots L_{DE_{k_4}}\}$ cumplen las siguientes propiedades:

P7 : $wd_i \leq wd_j$

P8 : $we_i \leq we_j$

P9 : $hd_i \geq hd_j$

P10: $he_i \geq he_j$

En consecuencia, las listas L_D, L_E, L_{DE} y cada una de las sub-listas contenidas en L_{DE}, están clasificadas en orden ascendente de anchura y descendente de altura.

El proceso de construcción de la lista L_{DE} (Figura 4-25) lo efectuamos de la siguiente forma:

- Creamos una lista vacía L_{DE}. A continuación, tomamos secuencialmente una a una las implementaciones (wd_i,hd_i) de la lista-R L_D. Por cada una, creamos una sub-lista vacía L_{DE_i} y la combinamos con las implementaciones (we_j,he_j) de la lista-R L_E.

- El bloque 7 resultante de dicha combinación, lo insertamos en la sublista-7 L_{DE_i} de la forma (wd_i,we_j,hd_i,he_j).

- Si la anchura del módulo D, wd_i, es mayor que la anchura del módulo E, we_j, no tendremos en cuenta las demás implementaciones de la lista-R, ya que como L_E está clasificada en orden ascendente de anchura y la recorremos empezando desde el último elemento al primero, el resto de elementos de dicha lista-R, si los hubiere, serían redundantes con respecto a éste.

- Si la anchura del módulo D, wd_i, es menor o igual a la anchura del módulo E, we_j, tomamos el siguiente elemento de la lista-R L_D si lo hubiere, y así sucesivamente.

```
ALGORITMO DE
  CREAR una lista vacía L_DE
  FOR i = 1 TO k_4 DO
      (wd_i,hd_i) = es el elemento i de L_D
      CREAR una sub - lista vacía L_DE_i
      FOR j = k_1 TO 1 DO
          (we_j,he_j) = es el elemento j de L_E
          AÑADIR en L_DE_i el elemento (wd_i,we_j,hd_j,he_j)
          IF we_j > wd_i
             THEN j = j - 1
             ELSE j = 1
          ENDIF
      ENDFOR
      i = i + 1
  ENDFOR
```

Figura 4-25 Algoritmo DE.

4.5 FASE ABCDE

En la cuarta fase de nuestro algoritmo AWO, combinamos cada una de las implementaciones L3 obtenidas en la fase ABC, con cada una de las implementaciones 7 generadas en la fase DE logrando un conjunto de bloques rectangulares ABCDE. En la figura 4-26 se representa dicho proceso.

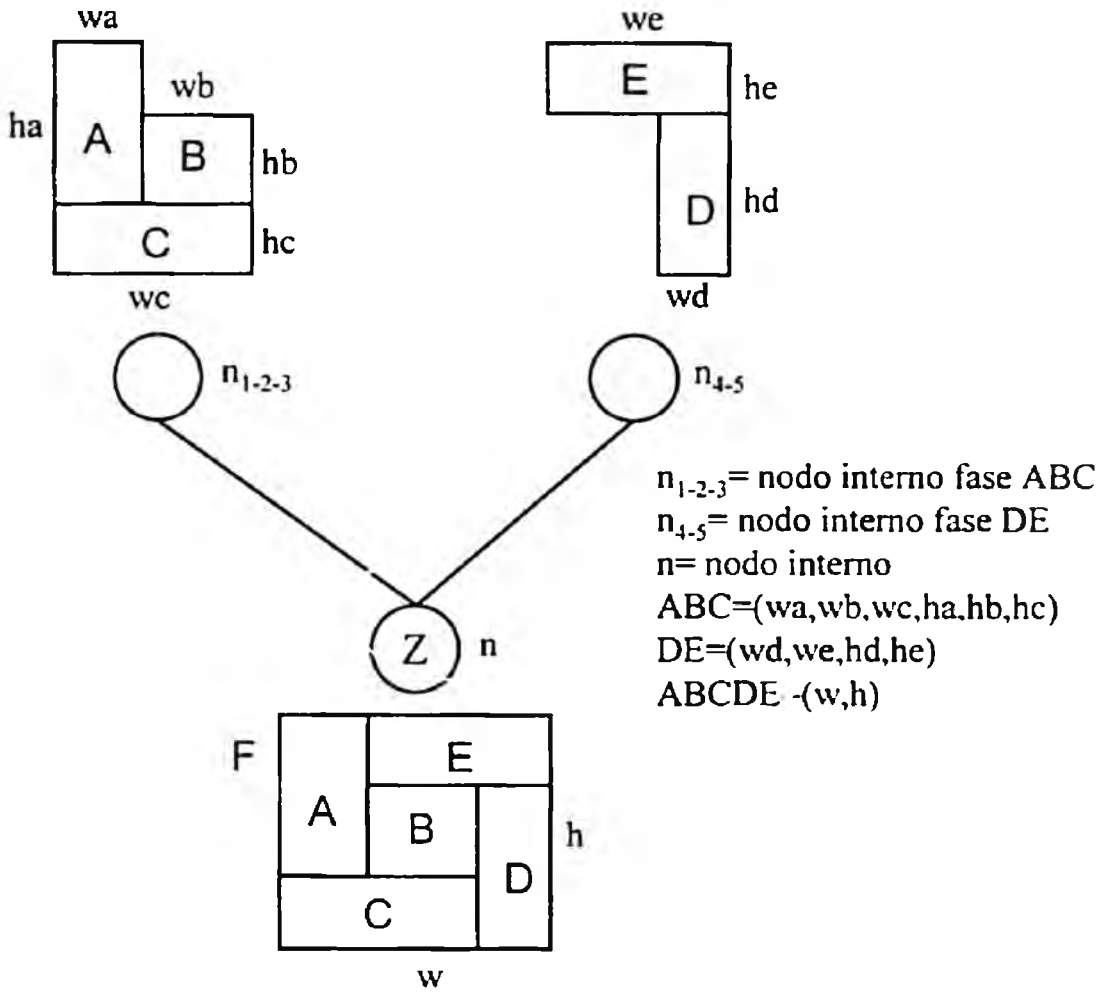


Figura 4-26 Árbol de unión ABCDE.

Sean dos bloques, ABC y DE, cuyas implementaciones $k_2 \bullet k_1 \bullet k_3$ y $k_5 \bullet k_4$ están recogidas en la lista $L_{ABC} = \{L_{ABC_1}, L_{ABC_2}, \ldots, L_{ABC_{k_2}}\} = \{(wa_i, wb_i, wc_i, ha_i, hb_i, hc_i) \mid 1 \le i \le k_2\}$

y la lista $L_{DE} = \{L_{DE_1}, L_{DE_2}, \ldots, L_{DE_{k_5}}\} = \{(wd_j, we_j, hd_j, he_j) \mid 1 \le j \le k_5\}$. La fase ABCDE genera una lista-R que contiene como máximo z implementaciones.

$L_{ABC} = \{L_{ABC_1}, L_{ABC_2}, \ldots, L_{ABC_{k_2}}\} =$

$\{\{(wa_{111}, wb_{111}, wc_{111}, ha_{111}, hb_{111}, hc_{111}), (wa_{112}, wb_{112}, wc_{112}, ha_{112}, hb_{112}, hc_{112}), \ldots$
$(wa_{11k_3}, wb_{11k_3}, wc_{11k_3}, ha_{11k_3}, hb_{11k_3}, hc_{11k_3})\}, \{(wa_{211}, wb_{211}, wc_{211}, ha_{211}, hb_{211}, hc_{211}),$
$(wa_{212}, wb_{212}, wc_{212}, ha_{212}, hb_{212}, hc_{212}), \ldots, (wa_{21k_3}, wb_{21k_3}, wc_{21k_3}, ha_{21k_3}, hb_{21k_3}, hc_{21k_3})\}, \ldots$
$\{(wa_{k_2k_11}, wb_{k_2k_11}, wc_{k_2k_11}, ha_{k_2k_11}, hb_{k_2k_11}, hc_{k_2k_11}), (wa_{k_2k_12}, wb_{k_2k_12}, wc_{k_2k_12}, ha_{k_2k_12}, hb_{k_2k_12}, hc_{k_2k_12}), \ldots$
$(wa_{k_2k_1k_3}, wb_{k_2k_1k_3}, wc_{k_2k_1k_3}, ha_{k_2k_1k_3}, hb_{k_2k_1k_3}, hc_{k_2k_1k_3})\}\}$

$L_{DE} = \{L_{DE_1}, L_{DE_2}, \ldots, L_{DE_{k_3}}\} =$

$\{\{(wd_{11}, we_{11}, hd_{11}, he_{11}), (wd_{12}, we_{12}, hd_{12}, he_{12}), \ldots, (wd_{1k_4}, we_{1k_4}, hd_{1k_4}, he_{1k_4})\},$
$\{(wd_{21}, we_{21}, hd_{21}, he_{21}), (wd_{22}, we_{22}, hd_{22}, he_{22}), \ldots, (wd_{2k_4}, we_{2k_4}, hd_{2k_4}, he_{2k_4})\},$
$\{(wd_{k_31}, we_{k_31}, hd_{k_31}, he_{k_31}), (wd_{k_32}, we_{k_32}, hd_{k_32}, he_{k_32}), \ldots (wd_{k_3k_4}, we_{k_3k_4}, hd_{k_3k_4}, he_{k_3k_4})\}\}$

$(wa_{111}, wb_{111}, wc_{111}, ha_{111}, hb_{111}, hc_{111}) \oslash (wd_{11}, we_{11}, hd_{11}, he_{11}) = (w_1, h_1)$
$(wa_{111}, wb_{111}, wc_{111}, ha_{111}, hb_{111}, hc_{111}) \oslash (wd_{12}, we_{12}, hd_{12}, he_{12}) = (w_2, h_2)$
.....
$(wa_{111}, wb_{111}, wc_{111}, ha_{111}, hb_{111}, hc_{111}) \oslash (wd_{1k_4}, we_{1k_4}, hd_{1k_4}, he_{1k_4}) = (w_t, h_{k_4})$

$(wa_{k_2k_1k_3}, wb_{k_2k_1k_3}, wc_{k_2k_1k_3}, ha_{k_2k_1k_3}, hb_{k_2k_1k_3}, hc_{k_2k_1k_3}) \oslash (wd_{11}, we_{11}, hd_{11}, he_{11}) = (w_u, h_t)$
$(wa_{k_2k_1k_3}, wb_{k_2k_1k_3}, wc_{k_2k_1k_3}, ha_{k_2k_1k_3}, hb_{k_2k_1k_3}, hc_{k_2k_1k_3}) \oslash (wd_{12}, we_{12}, hd_{12}, he_{12}) = (w_j, h_j)$
.....
$(wa_{k_2k_1k_3}, wb_{k_2k_1k_3}, wc_{k_2k_1k_3}, ha_{k_2k_1k_3}, hb_{k_2k_1k_3}, hc_{k_2k_1k_3}) \oslash (wd_{1k_4}, we_{1k_4}, hd_{1k_4}, he_{1k_4}) = (w_m, h_m)$

$(wa_{k_2k_1k_3}, wb_{k_2k_1k_3}, wc_{k_2k_1k_3}, ha_{k_2k_1k_3}, hb_{k_2k_1k_3}, hc_{k_2k_1k_3}) \oslash (wd_{k_31}, we_{k_31}, hd_{k_31}, he_{k_31}) = (w_r, h_r)$
$(wa_{k_2k_1k_3}, wb_{k_2k_1k_3}, wc_{k_2k_1k_3}, ha_{k_2k_1k_3}, hb_{k_2k_1k_3}, hc_{k_2k_1k_3}) \oslash (wd_{k_32}, we_{k_32}, hd_{k_32}, he_{k_32}) = (w_s, h_s)$
.....
$(wa_{k_2k_1k_3}, wb_{k_2k_1k_3}, wc_{k_2k_1k_3}, ha_{k_2k_1k_3}, hb_{k_2k_1k_3}, hc_{k_2k_1k_3}) \oslash (wd_{k_3k_4}, we_{k_3k_4}, hd_{k_3k_4}, he_{k_3k_4}) = (w_z, h_z)$

$L_{ABCDE} = \{(w_1, h_1), (w_2, h_2), (w_3, h_3), \ldots, (w_z, h_z)\}$

La lista-7 satisface las propiedades siguientes:

P7 : $wd_i \le wd_j$

P8 : $we_i \le we_j$

P9 : $hd_i \ge hd_j$

P10: $he_i \ge he_j$

y la lista-L3 L_{ABC} las siguientes:

$$P1 : wa_i \leq wa_j$$
$$P2 : wb_i \leq wb_j$$
$$P5 : wc_i \leq wc_j$$
$$P3 : ha_i \geq ha_j$$
$$P4 : hb_i \geq hb_j$$
$$P6 : hc_i \geq hc_j$$

Por tanto, todas las implementaciones de la lista-R resultante de la unión de ABC y DE pertenecientes a la lista-R $L_{ABCDE}=\{(w_1,h_1),(w_2,h_2),...,(w_z,h_z)\}$ no tienen por qué cumplir las siguientes propiedades:

$$P11 : w_i \leq w_j$$
$$P12 : h_i \geq h_j$$

Esto es debido a que en las fases anteriores obtenemos las listas L_{AB}, L_{ABC} y L_{DE} con implementaciones no redundantes en cada una de sus sub-listas. Sin embargo, podemos llegar a generar bloques L2, L3 o 7 que en unión con los de otras sub-listas son redundantes. Por ello, al unir las implementaciones de cada una de las sub-listas de la lista-L3 L_{ABC} y de la lista-7 L_{DE} en una única lista-R L_{ABCDE}, normalmente obtenemos elementos redundantes. Las listas L_{ABC}, L_{DE} así como cada una de las sub-listas contenidas en ellas están clasificadas en orden ascendente de anchura y en orden descendente de altura, mientras que la lista L_{ABCDE} al no contener ninguna sub-lista no tiene porqué estarlo.

El proceso de construcción de la lista L_{ABCDE} (Figura 4-28) lo efectúa de la siguiente forma:

- Creamos una lista vacía L_{ABCDE}. A continuación, accedemos secuencialmente a cada una de las sub-listas-L3 ABC y dentro de ellas a cada implementación $(wa_i,wb_i,wc_i,ha_i,hb_i,hc_i)$ de la sub-lista-L3 L_{ABC_k}.

- Para efectuar las uniones de los distintos elementos del bloque L3 con las del bloque 7 distinguimos cuatro casos según la configuración de partida del bloque L3. Cada uno de ellos (Figura 4-27) se producen cuando se establecen las siguientes condiciones:
  - CASO 1: $hb_i \geq ha_i$ y $wc_i > wa_i + wb_i$
  - CASO 2: $hb_i \geq ha_i$ y $wc_i \leq wa_i + wb_i$
  - CASO 3: $hb_i < ha_i$ y $wc_i > wa_i + wb_i$
  - CASO 4: $hb_i < ha_i$ y $wc_i \leq wa_i + wb_i$

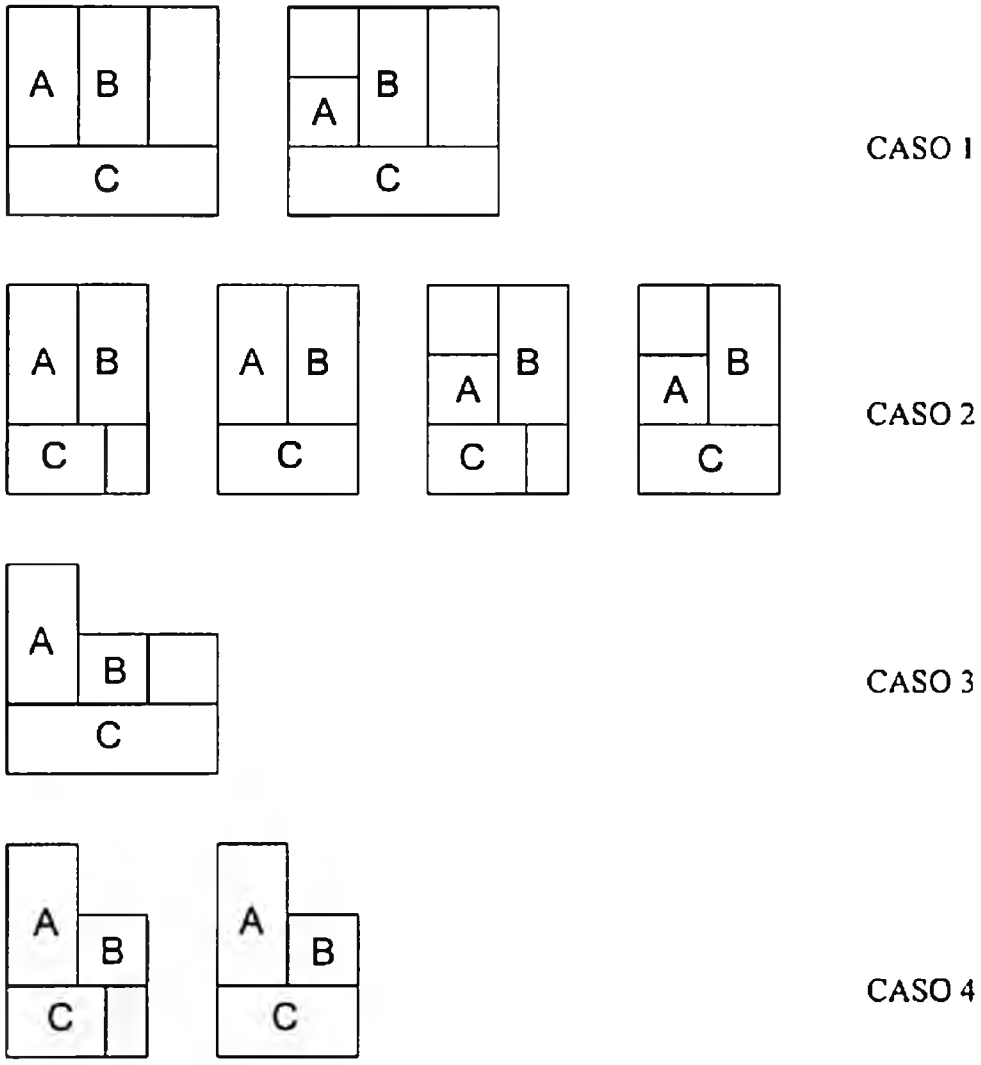


Figura 4-27 Distintas configuraciones producidas en la fase ABCDE.

```
ALGORITMO ABCDE
  CREAR una lista vacía L_ABCDE
  FOR k = 1 TO k_2 DO
      ACCEDER a la sub-lista L_ABC_k de la lista L_ABC
      FOR i = 1 TO n DO
          (wa_i, wb_i, wc_i, ha_i, hb_i, hc_i) es el elemento i de L_ABC_k
          IF hb_i ≥ ha_i AND wc_i > wa_i + wb_i
            THEN CASO 1
            ELSE IF hb_i ≥ ha_i AND wc_i ≤ wa_i + wb_i
                    THEN CASO 2
                    ELSE IF hb_i < ha_i AND wc_i > wa_i + wb_i
                            THEN CASO 3
                            ELSE CASO 4
                         ENDIF
                 ENDIF
          ENDIF
          i = i + 1
      ENDFOR
      k = k + 1
  ENDFOR
```

Figura 4-28 Algoritmo ABCDE.

4.5.1 Caso 1

Este caso se produce cuando la implementación del bloque L3 ABC perteneciente a la sub-lista-L3 de la lista-L3 tiene las dos características siguientes (Figura 4-29):

- La altura del módulo B es mayor o igual que la del módulo A. Al cumplirse esto, la unión del módulo A con el módulo B producirá siempre una zona desaprovechada que tendrá una anchura idéntica a la del módulo A y una altura equivalente a la que tenga el módulo E más la diferencia de alturas entre el módulo B y el módulo A.

- La anchura del módulo C es mayor que la correspondiente al bloque obtenido al unir los módulos A y B. Por tanto, producirá una zona desaprovechada en la consecución del floorplan, que tendrá una anchura equivalente a la diferencia de anchuras del módulo C con la del bloque AB y una altura idéntica a la altura del módulo B.

ALGORITMO CASO 1

```
FOR L = 1 TO k4 DO
    ACCEDER a la sub-lista LDEL de la lista LDE
    FOR j = 1 TO p DO
        (wdj,wej,hdj,hej) es el elemento j de LDEL
        IF L = k4
          THEN IF hdj ≤ hbi + hci
                 THEN IF wej ≤ wci − wai + wdj
                        THEN AÑADIR (wci + wdj,hbi + hci + hej)
                        ELSE AÑADIR (wai + wej,hbi + hci + hej)
                      ENDIF
                 ELSE IF wej ≤ wci − wai + wdj
                        THEN AÑADIR (wci + wdj,hdj + hej)
                        ELSE AÑADIR (wai + wej,hdj + hej)
                      ENDIF
               ENDIF
               j = j+1
          ELSE IF hdj ≤ hbi + hci
                 THEN IF wej ≤ wci − wai + wdj
                        THEN AÑADIR (wci + wdj,hbi + hei + hej)
                             j = p
                             L = k4
                        ELSE j = j+1
                      ENDIF
                 ELSE j = p
               ENDIF
        ENDIF
    ENDFOR
    L = L+1
ENDFOR
```

Figura 4-29 Algoritmo del CASO 1.

La unión de un bloque L3 ABC con otro bloque 7 DE da lugar a distintas configuraciones de floorplans, las cuales se añadirán a la lista-R del bloque final ABCDE. Este bloque es rectangular y por lo tanto las implementaciones son de la forma (w,h) donde la anchura w y la altura h dependerá de la unión de los bloques ABC y DE.

Las posibles configuraciones de los floorplans que se pueden producir son las siguientes (Figura 4-30):

- (wc_i+wd_j,hd_j+he_j)
- $(wc_i+wd_j,hb_i+hc_i+he_j)$

- (wa_i+we_j, hd_j+he_j)
- $(wa_i+we_j, hb_i+hc_i+he_j)$

es decir, $(w,h)=(\max(wc_i+wd_j, wa_i+we_j), \max(hd_j+he_j, hb_i+hc_i+he_j))$.

Los bloques ABCDE obtenidos por el algoritmo para este caso (Figura 4-30) siempre tendrán zonas desaprovechadas provocadas por una parte por la propia configuración del bloque L3 y por otra por la unión con el bloque 7. Sin embargo, esto no quiere decir que alguno de estos bloques no sea el floorplan no particionado óptimo en área.

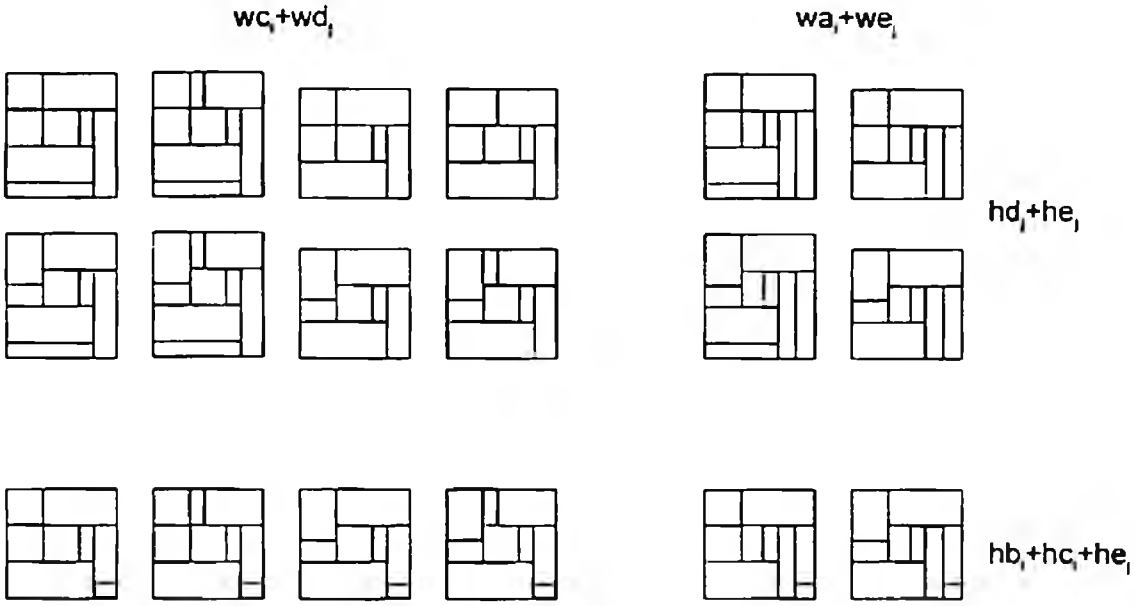


Figura 4-30 Configuraciones del CASO 1 en altura y anchura.

4.5.2 Caso 2

Este caso se produce cuando la implementación del bloque L3 ABC perteneciente a la sub-lista-L3 de la lista-L3 tiene las dos características siguientes (Figura 4-31):

- La altura del módulo B es mayor o igual que la del módulo A. Al cumplirse esto, la unión del módulo A con el módulo B producirá siempre una zona desaprovechada que tendrá una anchura idéntica a la del módulo A y una altura equivalente a la que tenga el módulo E mas la diferencia de alturas entre el módulo B y el módulo A.
- La anchura del módulo C siempre es menor o igual que la correspondiente al bloque obtenido al unir los módulos A y B. Este módulo C producirá en el peor de los casos ($wc_i<wa_i+wb_i$) una zona desaprovechada en la consecución del floorplan que tendrá una anchura equivalente a la diferencia de anchuras del módulo C con la del bloque AB y una altura idéntica a la altura del módulo C.

La unión de un bloque L3 ABC con otro bloque 7 DE da lugar a distintas configuraciones de floorplans, las cuales se añadirán a la lista-R del bloque final ABCDE. La implementación del bloque rectangular ABCDE obtenido es de la forma (w,h) donde la anchura w y la altura h dependerá de la unión de los bloques ABC y DE.

Las posibles configuraciones de los floorplans resultantes son las siguientes (Figura 4-32):

- $(wa_i+wb_i+wd_j, hd_j+he_j)$
- $(wa_i+wb_i+wd_j, hb_i+hc_i+he_j)$
- (wa_i+we_j, hd_j+he_j)
- $(wa_i+we_j, hb_i+hc_i+he_j)$

es decir, $(w,h)=(\max(wa_i+wb_i+wd_j, wa_i+we_j), \max(hd_j+he_j, hb_i+hc_i+he_j))$.

ALGORITMO CASO 2

```
FOR L = 1 TO k4 DO
    ACCEDER a la sub-lista LDEL de la lista LDE
    FOR j = 1 TO p DO
        (wdj, wej, hdj, hej) es el elemento j de LDEL
        IF L = k4
          THEN IF hdj ≤ hbi + hci
                  THEN IF wej ≤ wbi + wdj
                          THEN AÑADIR (wai + wbi + wdj, hbi + hci + hej)
                          ELSE AÑADIR (wai + wej, hbi + hci + hei)
                       ENDIF
                  ELSE IF wej ≤ wbi + wdj
                          THEN AÑADIR (wai + wbi + wdj, hdj + hei)
                          ELSE AÑADIR (wai + wej, hdj + hej)
                       ENDIF
               ENDIF
               j = j + 1
          ELSE IF hdj ≤ hbi + hci
                  THEN IF wej ≤ wbi + wdj
                          THEN AÑADIR (wai + wbi + wdj, hbi + hci + hej)
                               j = p
                               L = k4
                          ELSE j = j + 1
                       ENDIF
                  ELSE j = p
               ENDIF
        ENDIF
    ENDFOR
    L = L + 1
ENDFOR
```

Figura 4-31 Algoritmo del CASO 2.

Los bloques ABCDE obtenidos por el algoritmo para este caso siempre tendrán zonas desaprovechadas provocadas por una parte por la propia configuración del bloque L3 y por otra por la unión con el bloque 7. Esto no quiere decir que alguno de estos bloques no sea el floorplan no particionado óptimo en área.

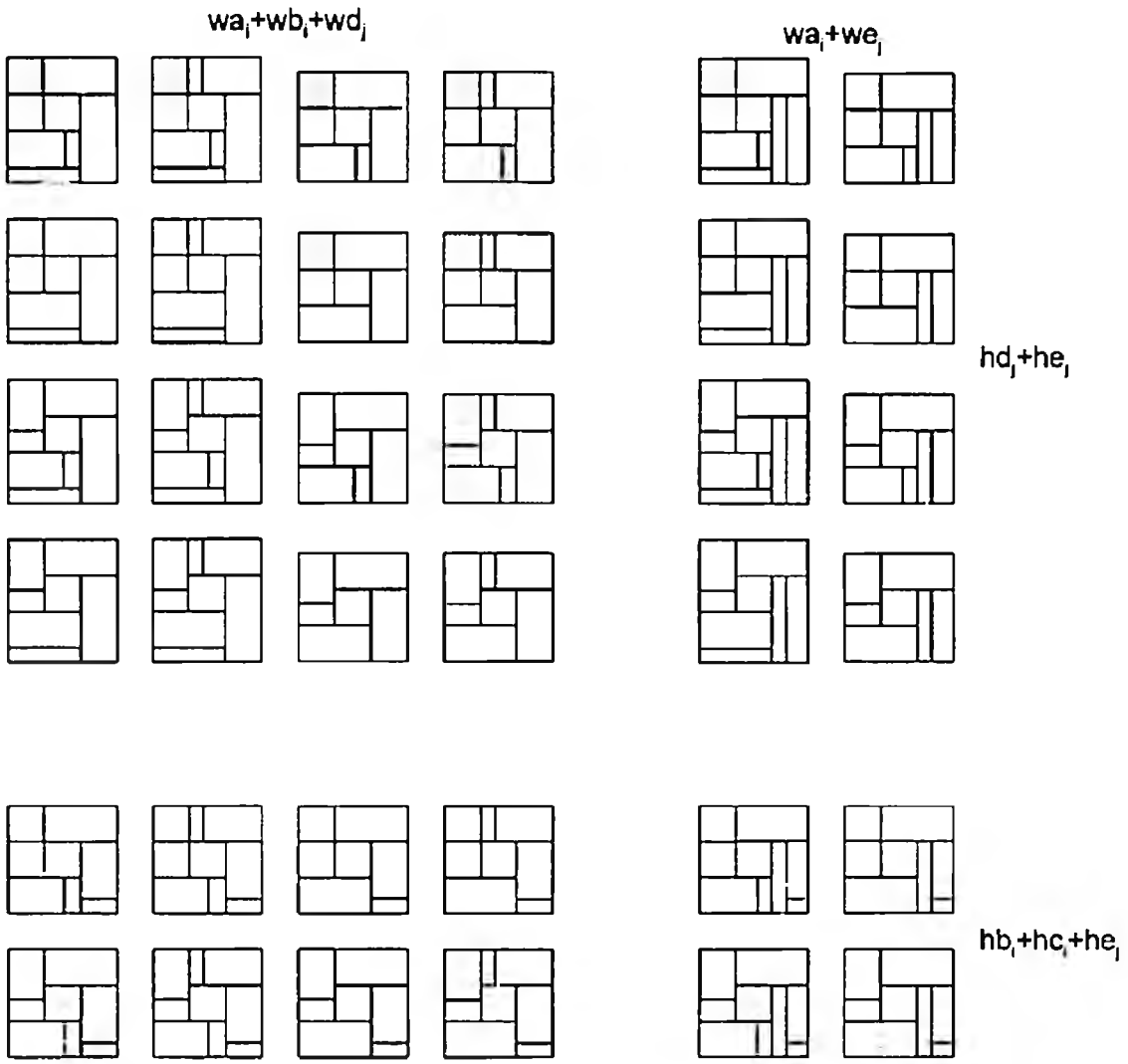


Figura 4-32 Configuraciones del CASO 2 en altura y anchura.

4.5.3 Caso 3

Este caso se produce cuando la implementación del bloque L3 ABC perteneciente a la sub-lista-L3 de la lista-L3 tiene las dos características siguientes:

- La altura del módulo B es menor que la del módulo A. Al cumplirse esto, la unión del módulo A con el módulo B producirá en el mejor de los casos un bloque que no tenga zonas desaprovechadas.

- La anchura del módulo C es mayor que la correspondiente al bloque obtenido al unir los módulos A y B. Este módulo produce una zona desaprovechada en la consecución del floorplan que tendrá una anchura equivalente a la diferencia de anchuras del módulo C y la del bloque AB, y una altura idéntica a la altura del módulo B.

Las posibles configuraciones de los floorplans que pueden producirse en este caso, son las siguientes (Figura 4-33):

- (wa_i+we_j, hd_j+he_j)
- (wa_i+we_j, ha_i+hc_i)
- $(wa_i+we_j, hb_i+hc_i+he_j)$
- (wa_i+we_j, hd_j+he_j) o (wa_i+we_j, ha_i+hc_i)
- (wc_i+wd_j, hd_j+he_j)
- (wc_i+wd_j, ha_i+hc_i)
- $(wc_i+wd_j, hb_i+hc_i+he_j)$
- (wc_i+wd_j, hd_j+he_j) o (wc_i+wd_j, ha_i+hc_i)

es decir, $(w,h)=(\max(wa_i+we_j, wc_i+wd_j), \max(hd_j+he_j, ha_i+hc_i, hb_i+hc_i+he_j))$.

La unión de un bloque L3 ABC con otro bloque 7 DE da lugar a distintas configuraciones de floorplans, las cuales se añadirán en la lista-R del bloque final ABCDE. La implementación del bloque rectangular ABCDE obtenido es de la forma (w,h) donde la anchura w y la altura h que dependerán de la unión de los bloques ABC y DE.

Los bloques ABCDE obtenidos por el algoritmo para este caso (Figura 4-34) siempre tendrán zonas desaprovechadas provocadas por una parte por la propia configuración del bloque L3 y por otra por la unión con el bloque 7. Esto

no quiere decir que alguno de estos bloques no sea el floorplan no particionado óptimo en área.

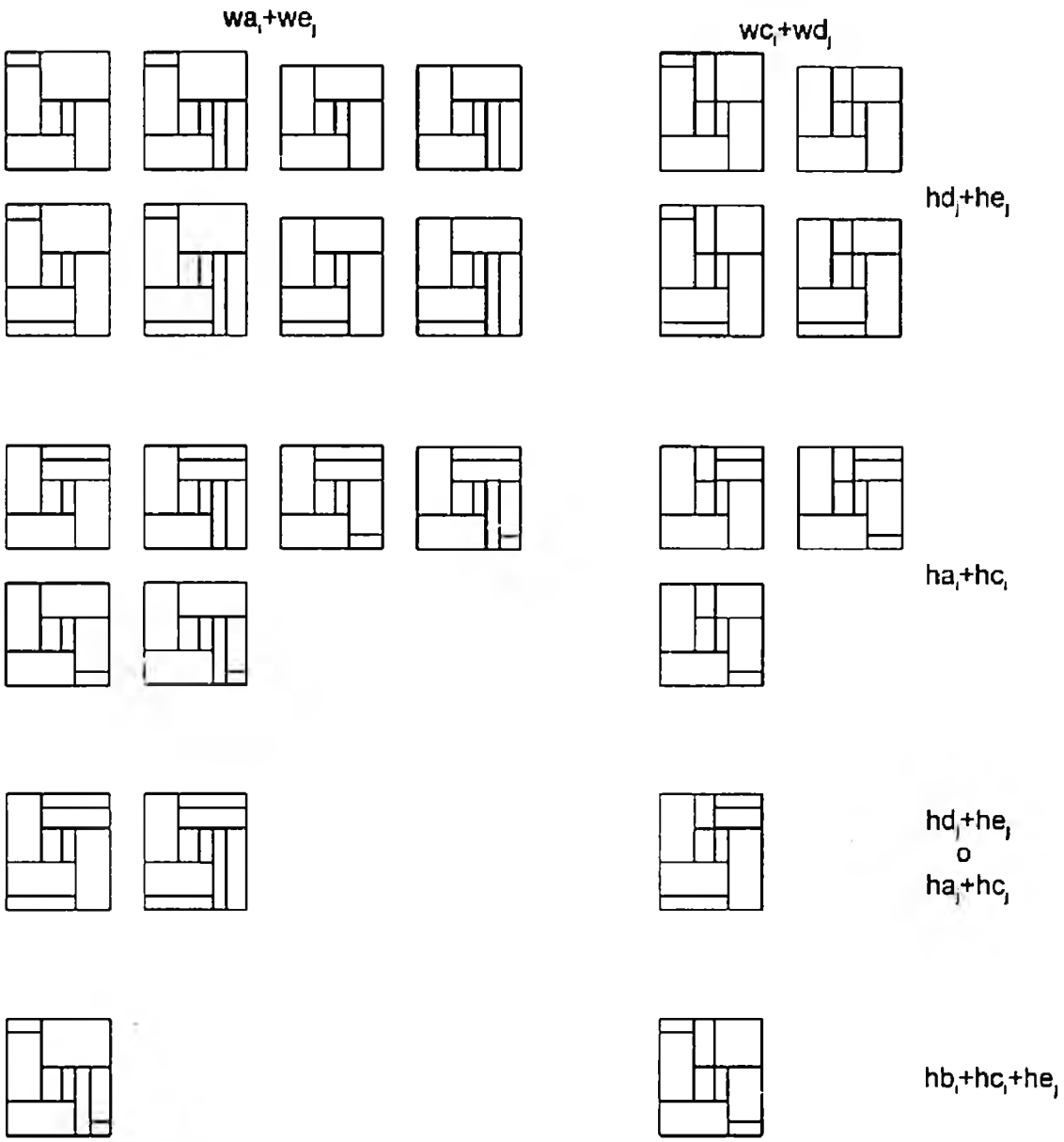


Figura 4-33 Configuraciones del CASO 3 en altura y anchura.

ALGORITMO CASO 3

```
FOR L = 1 TO k4 DO
    ACCEDER a la sub-lista L_DE_L de la lista L_DE
    FOR j = 1 TO p DO
        (wdj, wej, hdj, hej) es el elemento j de L_DE_L
        IF L = k4
          THEN IF hdj ≤ hbi + hci
                 THEN IF wej ≤ wci − wai + wdj
                        THEN IF hej > hai − hbi
                               THEN AÑADIR (wci + wdj, hbi + hci + hej)
                               ELSE AÑADIR (wci + wdj, hai + hci)
                             ENDIF
                        ELSE IF hej > hai − hbi
                               THEN AÑADIR (wai + wej, hbi + hci + hej)
                               ELSE AÑADIR (wai + wej, hai + hci)
                             ENDIF
                      ENDIF
                 ELSE IF wej ≤ wci − wai + wdj
                        THEN IF hej > hai − hbi
                               THEN AÑADIR (wci + wdj, hdj + hej)
                               ELSE IF hai + hci > hdj + hej
                                      THEN AÑADIR (wci + wdj, hai + hci)
                                      ELSE AÑADIR (wci + wdj, hdj + hej)
                                    ENDIF
                             ENDIF
                        ELSE IF hej > hai − hbi
                               THEN AÑADIR (wai + wej, hdj + hej)
                               ELSE IF hai + hci > hdj + hej
                                      THEN AÑADIR (wai + wej, hai + hci)
                                      ELSE AÑADIR (wai + wej, hdj + hej)
                                    ENDIF
                             ENDIF
                      ENDIF
                 ENDIF
                 j = j + 1
          ELSE IF wej ≤ wci − wai + wdj
                 THEN IF hdj ≤ hbi + hci OR hdj + hej ≤ hai + hci
                        THEN IF hej ≤ hai − hbi
                               THEN AÑADIR (wci + wdj, hai + hci)
                               ELSE AÑADIR (wci + wdj, hbi + hci + hej)
                             ENDIF
                             j = p
                             L = k4
                        ELSE j = p
                      ENDIF
                 ELSE IF hdj ≤ hbi + hci AND hdj + hej > hai + hci
                        THEN j = p
                        ELSE j = j + 1
                      ENDIF
               ENDIF
      ENDIF
    ENDFOR
    L = L + 1
ENDFOR
```

Figura 4-34 Algoritmo del CASO 3.

4.5.4 Caso 4

Este caso se da cuando la implementación del bloque L3 ABC perteneciente a la sub-lista-L3 de la lista-L3 tiene las dos características siguientes:

- La altura del módulo B es menor a la del módulo A. Al cumplirse esto, la unión del módulo A con el B producirá en el mejor de los casos, un bloque sin zonas desaprovechadas.
- La anchura del módulo C es menor o igual que la correspondiente al bloque obtenido al unir los módulos A y B. El módulo C producirá en el peor de los casos ($wc_i<wa_i+wb_i$) una zona desaprovechada en la consecución del floorplan, que tendrá una anchura equivalente a la diferencia de anchuras del módulo C y la del bloque AB, y una altura idéntica a la altura del módulo C. En el mejor de los casos obtendremos un floorplan óptimo en área y sin zonas desaprovechadas.

La unión de un bloque L3 ABC con otro bloque 7 DE da lugar a distintas configuraciones de floorplans, las cuales se añadirán en la lista-R del bloque final ABCDE. La implementación del bloque rectangular ABCDE obtenido es de la forma (w,h) donde la anchura w y la altura h que dependerán de la unión de los bloques ABC y DE.

Los bloques ABCDE obtenidos por el algoritmo para este caso (Figura 4-35) no siempre tendrán zonas desaprovechadas y si las tienen son provocadas por una parte por la propia configuración del bloque L3 y por otra, por la unión con el bloque 7. Esto significa que alguno de estos bloques puede dar lugar al floorplan no particionado óptimo en área.

ALGORITMO CASO 4

```
FOR L = 1 TO k4 DO
    ACCEDER a la sub-lista L_DEL de la lista L_DE
    FOR j = 1 TO p DO
        (wdj.wej.hdj.hej) es el elemento j de L_DEL
        IF L = k4
          THEN IF hdj ≤ hbi + hci
                 THEN IF wej ≤ wbi + wdj
                        THEN IF hej > hai - hbi
                               THEN AÑADIR (wai + wbi + wdj.hbi + hci + hej)
                               ELSE AÑADIR (wai + wbi + wdj.hai + hci)
                             ENDIF
                        ELSE IF hej > hai - hbi
                               THEN AÑADIR (wai + wej.hbi + hci + hej)
                               ELSE AÑADIR (wai + wej.hai + hci)
                             ENDIF
                      ENDIF
                 ELSE IF wej ≤ wbi + wdj
                        THEN IF hej > hai - hbi
                               THEN AÑADIR (wai + wbi + wdj.hdi + hei)
                               ELSE IF hai + hci > hdj + hej
                                      THEN AÑADIR (wai + wbi + wdj.hai + hci)
                                      ELSE AÑADIR (wai + wbi + wdj.hdj + hej)
                                    ENDIF
                             ENDIF
                        ELSE IF hej > hai - hbi
                               THEN AÑADIR (wai + wej.hdj + hej)
                               ELSE IF hai + hci > hdj + hej
                                      THEN AÑADIR (wai + wej.hai + hci)
                                      ELSE AÑADIR (wai + wej.hdj + hej)
                                    ENDIF
                             ENDIF
                      ENDIF
                 ENDIF
                 j = j+
          ELSE IF wej ≤ wbi + wdj
                 THEN IF hdj ≤ hbi + hci OR hdj + hej ≤ hai + hci
                        THEN IF hej ≤ hai - hbi
                               THEN AÑADIR (wai + wbi + wdj.hai + hci)
                               ELSE AÑADIR (wai + wbi + wdj.hbi + hci + hej)
                             ENDIF
                             j = p
                             L = k4
                        ELSE j = p
                      ENDIF
                 ELSE IF hdj ≥ hbi + hci AND hdj + hej > hai + hci
                        THEN j = p
                        ELSE j = j+1
                      ENDIF
               ENDIF
      ENDIF
    ENDFOR
    L = L+1
ENDFOR
```

Figura 4-35 Algoritmo del CASO 4.

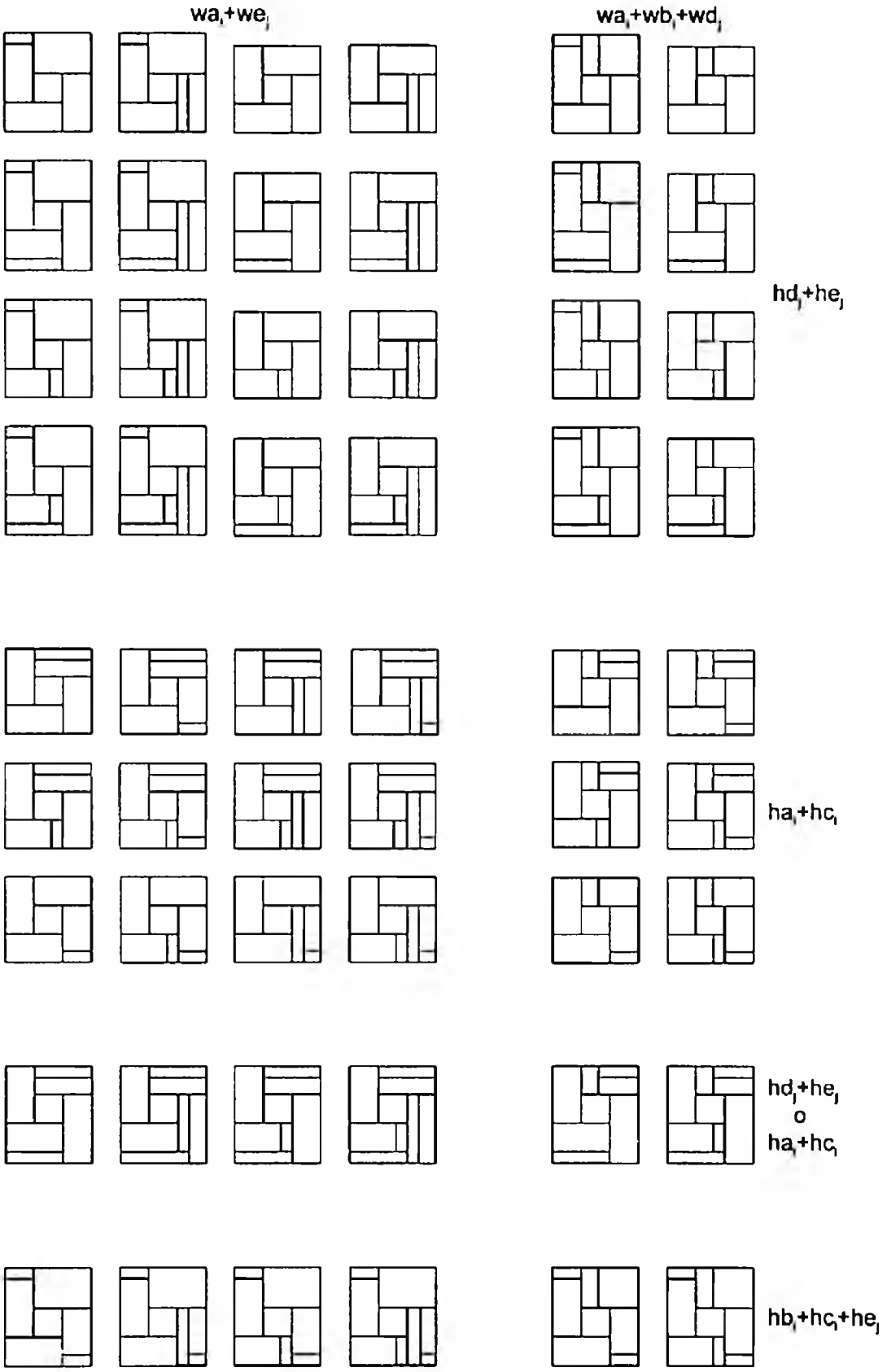


Figura 4-36 Configuraciones del CASO 4 en altura y anchura.

Las posibles configuraciones de los floorplans que se pueden producir son los siguientes (Figura 4-36):

- (wa_i+we_j, hd_j+he_j)
- (wa_i+we_j, ha_i+hc_i)
- $(wa_i+we_j, hb_i+hc_i+he_j)$
- (wa_i+we_j, hd_j+he_j) o (wa_i+we_j, ha_i+hc_i)
- $(wa_i+wb_i+wd_j, hd_j+he_j)$
- $(wa_i+wb_i+wd_j, ha_i+hc_i)$
- $(wa_i+wb_i+wd_j, hb_i+hc_i+he_j)$
- $(wa_i+wb_i+wd_j, hd_j+he_j)$ o (wc_i+wd_j, ha_i+hc_i)

es decir, $(w,h)=(\max(wa_i+we_j, wa_i+wb_i+wd_j), \max(hd_j+he_j, ha_i+hc_i, hb_i+hc_i+he_j))$.

4.6 FASE FINAL

La misión de esta última fase es la de eliminar aquellas implementaciones redundantes que puedan estar presentes en la lista-R $L_{ABCDE}=\{(w_1,h_1),(w_2,h_2),\ldots,(w_i,h_i),\ldots,(w_z,h_z)\}$.

El proceso de construcción de la nueva lista L_{ABCDE} (Figura 4-37) es el siguiente:

- En primer lugar, clasificamos la lista-R L_{ABCDE} en orden creciente de anchura w y en orden decreciente de altura h. Una vez clasificada comprobamos si todas las implementaciones cumplen las propiedades P19 y P20. La que no las cumpla es redundante y por tanto la eliminamos.

- Si la anchura y la altura de la implementación que estamos comparando en un momento determinado (w_a,h_a) es menor que la anchura y la altura de la siguiente implementación (w_i,h_i), entonces esta última implementación es redundante y por tanto la eliminamos de la lista-R L_{ABCDE}.

- En caso contrario, la implementación (w_a,h_a) es válida y por tanto, la mantenemos como solución final del floorplan en la lista-R L_{ABCDE}.

Una vez terminado el proceso, obtenemos en la lista-R final L_{ABCDE} todas las implementaciones no redundantes de los floorplans no particionados de orden cinco de la forma: $L_{ABCDE}=\{(w_1,h_1),(w_2,h_2),\ldots,(w_n,h_n)\}$.

ALGORITMO FINAL
CLASIFICAR la lista-R L_{ABCDE} en orden ascendente de w y descendente de h
i=1
$(w_a,h_a)=(w_i,h_i)$ es el primer elemento de L_{ABCDE}
FOR i=2 TO z DO
  (w_i,h_i) es el elemento i de L_{ABCDE}
  IF $w_i \geq w_a$ AND $h_i \geq h_a$
    THEN BORRAR en L_{ABCDE} el elemento (w_i,h_i)
    ELSE $(w_a,h_a)=(w_i,h_i)$
  ENDIF
  i=i+1
ENDFOR

Figura 4-37 Algoritmo FINAL.

Así, con el algoritmo AWO que hemos desarrollado, siempre obtenemos todas las implementaciones óptimas en área del floorplan no particionado y esto lo conseguimos en un tiempo de proceso menor gracias a un número mínimo de nodos visitados y de implementaciones generadas con respecto a los algoritmos desarrollados por los diferentes investigadores que han abordado este problema. Además, podemos manejar floorplans más complejos ya que en su resolución necesitamos menos memoria.

5. CONCLUSIONES

El presente trabajo de investigación aporta una solución factible a la problemática del área óptima del floorplan no particionado con el menor tiempo de proceso posible. Dicha solución viene dada por el algoritmo AWO que hemos desarrollado. En la figura 5-1 está representada la estructura de dicho algoritmo.

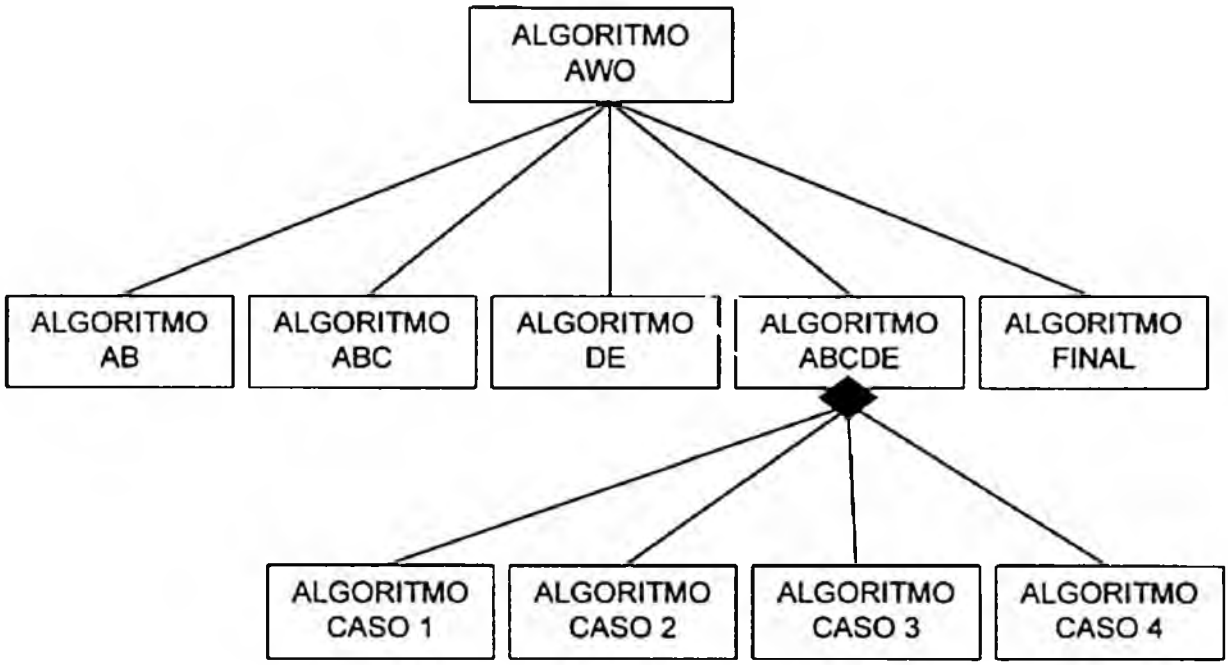


Figura 5-1 Diagrama del algoritmo AWO.

Hemos implementado nuestro algoritmo, en lenguaje JAVA en un ordenador Pentium II 400 Mhz y con 384 MB de RAM bajo Windows NT versión 4.0 (en el Apéndice B se incluyen las rutinas más relevantes de dicho algoritmo y al completo en el CD adjunto). A diferencia de los métodos desarrollados por otros autores, el nuestro es no heurístico y válido para cualquier tipo de floorplan no particionado de orden cinco. Incluso se puede hacer extensible a los floorplans generales, utilizando la técnica de Stockmeyer

[STOC83] cuando contengan estructuras particionadas y por lo tanto sean de orden dos.

Para probar la validez, robustez y fiabilidad de nuestro algoritmo lo hemos sometido a una batería de pruebas. Éstas son las mismas que las realizadas para los algoritmos de otros autores, que se plantearon resolver el difícil problema de los floorplans no particionados, como son Wimer, Koren y Cederbaum [WIME89], T.-C. Wang y Wong [WANG92b], K. Wang y Chen [WANG93a] y, Pan y Liu [PAN95]. En cada una de estas pruebas utilizamos distinto número de módulos así como distinta cantidad de implementaciones por cada uno de ellos. Hay que indicar que el número de nodos visitados por nuestro algoritmo es el número total de implementaciones generadas por todos los nodos internos en el árbol binario del floorplan, de tal manera que cada implementación sólo corresponde a un floorplan parcial. Los resultados experimentales indican que nuestro algoritmo elimina un gran número de implementaciones redundantes. Por ello, el número de nodos visitados y por tanto, el número de implementaciones investigadas es menor con respecto a los demás algoritmos que logran un floorplan no particionado óptimo en área. Los resultados obtenidos tanto por nuestro algoritmo como por los algoritmos propuestos por dichos autores están resumidos en las distintas tablas del Apéndice A. En ellas representamos el algoritmo rama-y-límite de Wimer, Koren y Cederbaum por BB, el algoritmo de T.-C.Wang y Wong por OPT, el algoritmo de K.Wang y Chen por ES, el algoritmo de Pan y Liu por AreaMin y el nuestro por AWO. El algoritmo BB que hemos utilizado es la implementación hecha por Arvindam, Kumar y Rao [ARVI89].

De estos resultados se desprenden las conclusiones siguientes:

- Con los algoritmos AWO, BB y OPT siempre se obtiene el área óptima del floorplan no particionado de orden cinco y no con los algoritmos ES y AreaMin en ciertas pruebas, como se puede observar en la tabla A-16.

- El algoritmo BB o rama-y-límite es siempre el que peor se comporta, de forma que para casos sencillos, con 25 módulos, el problema puede llegar a ser implanteable debido a tiempos de proceso excesivamente elevados. Así mismo, como con él se tienen que visitar una gran cantidad de nodos la memoria disponible no es suficiente para la consecución de ciertas soluciones (Tabla A-4 y A-5). Por ello, hemos descartado las comparaciones con floorplans más complejos de 125 y 625 módulos.

- El algoritmo OPT siempre obtiene áreas óptimas, pero tiene el inconveniente que para 625 módulos, el tiempo de proceso necesario para la obtención de la solución óptima empieza a ser excesivo.

- Las mejoras que hemos obtenido en el número de nodos visitados en comparación con el algoritmo BB rondan entre un 93% y un 99%, con el algoritmo OPT entre un 61% y un 70%, con el algoritmo ES entre un 51% y un 77% y por último con el algoritmo AreaMin entre un 10% y un 48%.

- En cuanto al tiempo de proceso las mejoras que hemos conseguido a la hora de obtener las implementaciones no redundantes óptimas en área son en comparación con el algoritmo BB de un 99%, con el algoritmo OPT entre un 93% y un 99%, con el algoritmo ES entre un 13% y un 63% y por último con el algoritmo AreaMin entre un 26% y un 87%.

En resumen, nuestro algoritmo AWO siempre obtiene el área óptima, es más rápido y puede manejar floorplans más complejos ya que el consumo de memoria es mucho menor. Así mismo, debemos indicar que si la ejecución se realizara en paralelo, el tiempo de proceso será aún menor. Las fases que se pueden ejecutar en paralelo son por una parte DE y por otra parte la secuencia AB, ABC.

5.1 FUTURAS LÍNEAS DE INVESTIGACIÓN

La continuación del trabajo expuesto en esta memoria deja abiertas diversas líneas de investigación, entre las que destacamos las siguientes:

En primer lugar, sería interesante abordar la unión de los procesos del floorplanning y del conexionado. Los floorplans obtenidos podrían no cumplir ciertas características prefijadas o inclusive volverse irrealizables.

Por otro lado, los compiladores de silicio actuales carecen de "inteligencia" para determinar la forma, tamaño y posición de cada uno de los módulos que compone el floorplan, tanto particionado como no particionado, así como las conexiones que determinan el área activa de un CI. En muchos casos, se hace imprescindible la intervención del diseñador para lograr resultados óptimos. Por ello, otra vía de investigación sería estudiar la viabilidad y mejoras potenciales de sustituir la experiencia y el conocimiento del diseñador por un sistema experto. Esto permitiría una mayor velocidad de proceso al eliminarse errores de diseño sólo detectables al final del mismo, y un progresivo aumento de la calidad del producto resultante.

Otra línea de investigación abierta en la optimización del área de floorplans no particionados, sería el planteamiento de restricciones en la ubicación, la orientación y la forma definitiva de alguno o algunos módulos. Esto simplificaría significativamente por una parte, la labor del diseñador y por otra, las problemáticas del tiempo de proceso y la memoria necesarias para la resolución del área óptima del floorplan, debido a la eliminación precoz de soluciones inválidas.

Así mismo, sería interesante abordar las implicaciones del establecimiento de una jerarquía de preferencias utilizando técnicas de lógica difusa. Ponderando cada uno de los criterios de diseño, se lograría una mayor

flexibilidad del proceso de obtención de resultados y de la elección de entre estos, el más óptimo en base a los criterios establecidos.

También proponemos investigar nuevas soluciones al problema de emplazamiento de módulos y bloques dentro de un floorplan general, que contengan estructuras tanto particionadas como no particionadas. La metodología a realizar parte de un conjunto totalmente arbitrario de bloques rectilíneos, que pueden ser a su vez rectangulares, convexos y cóncavos, de forma que el objetivo es obtener el floorplan correspondiente óptimo en área y en el menor tiempo de proceso posible. Un bloque rectangular es aquel que puede ser particionado mediante cortes horizontales y verticales hasta la obtención de los módulos básicos. Por otro lado, un bloque es convexo cuando dos puntos cualesquiera dentro del bloque pueden ser unidos totalmente por su interior recorriendo el menor espacio posible; este tipo de bloque puede dar lugar a su vez a otras formas del tipo L y 7. Por último, un bloque es cóncavo cuando para unir dos puntos dentro de él, el camino óptimo entre ellos obliga salirse fuera de la estructura del bloque.

Por último, conseguir un algoritmo polinómico (no NP-completo) sería una aportación vital en este campo. Con dicho algoritmo se resolverían múltiples problemas que se generan en el diseño del floorplan, para los que únicamente existen soluciones parciales.

BIBLIOGRAFÍA

[ARVI89] Arvindam, S., Kumar, V. y Rao, V.N., **"Floorplan Optimization on Multiprocessors"**, *Proceedings of the International Conference on Computer Design*, 1989, pp. 109-114.

[BALA99] Balasa, F. y Lampaert, K., **"Module Placement for Analog Layout Using the Sequence-Pair Representation"**, *Proceedings of the 36th Design Automation Conference*, 1999, pp. 274-279.

[BAZA98] Bazargan, K., Kim, S. y Sarrafzadeh, M., **"Nostradamus: a Floorplanner of Uncertain Design"**, *Proceedings of the International Symposium on Physical Design*, 1998, pp. 18-23.

[BRAY90] Brayton, R.K., Hachtel, G.D., McMullen, C.T. y Sangiovanni-Vincentelli, A.L., ***Logic Minimization Algorithms for VLSI Synthesis***, Kluwer Academic Publishers, 1990.

[BROO40] Brooks, R.L., Smith, C.A.B., Stone, A.H. y Tuttle, W.T., **"The Dissection of Rectangles into Squares"**, *Duke Mathematics Journal*, vol. 7, 1940, pp. 312-340.

[BUI87] Bui, T.N., Chauduru, S. , Leighton, F.T. y Sipser, M., **"Graph Bisection Algorithms with Good Average Case Behavior"**, *Combinatorica*, 7(2), 1987, pp. 171-191.

[CAI88] Cai, H. y Hegge, J.J.A., **"Comparison of Floor Planning Algorithms for Full Custom IC's"**, *Proceedings of the Custom Integrated Circuit Conference*, 1988, Sect. 7.2.

[CHEN93] Chen, C.-H.J., ***Area Optimization of Floorplan Designs***, Tesis Doctoral, Universidad de Texas en Dallas, 1993.

[CHEN96] Chen, D.Z. y Hu, X, **"Efficient Aproximation Algorithms for Floorplan Area Minimization"**, *Proceedings of the 33th Design Automation Conference*, 1996, pp. 483-486.

[CHEN98] Chen, T. y Fan, M.K.H., **"On Convex Formulation of the Floorplan Area Minimization Problem"**, *Proceedings of the International Symposium on Physical Design*, 1998, pp. 124-128.

[CHEN98] Chen, C.-S., Hwang, T.-T. y Liu, C.L., **"Architecture Driven Circuit Partitioning"**, *IEEE/ACM International Conference on Computer Aided Design*, 1998, pp. 408-411.

[CHON93] Chong, K. y Sahni, S., **"Optimal Realizations of Floorplans"**, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, nº 6, 1993, pp. 793-801.

[COHO86] Cohoon, J.P. y Paris, W.D., **"Genetic Placement"**, *Proceedings of IEEE International Conference on Computer Aided Design*, 1986, pp. 483-492.

[CONG99] Cong, J., Kong, T., Xu, D., Liang, F., Liu, J.S. y Wong, W.H., **"Simulated Tempering for VLSI Floorplan Designs"**, *Proceedings of the Asia and South Pacific Design Automation Conference*, 1999, pp. 13-16.

[DAI87] Dai, W.M. y Kuh, E.S., **"Simultaneous Floor Planning and Global Routing for Hierarchical Building-Block Layout"**, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, nº 5, 1987, pp. 828-837.

[DASG95] Dasgupta, P.S., Sur-Kolay, S. y Bhattacharya, B.B., **"A Unified Approach to Topology Generation and Area Optimization of General Floorplans"**, *IEEE/ACM International Conference on Computer-Aided Design*, 1995, pp. 712-715.

[DASG97] Dasgupta, P.S. y Sur-Kolay, S., **"Slicibility of Rectangular Graphs and Floorplan Optimization"**, *Proceedings of the International Symposium on Physical Design*, 1997, pp. 150-155.

[EISE98] Eisenmann, H. y Johannes, F.M., **"Generic Global Placement and Floorplanning"**, *Proceedings of the 35th Design Automation Conference*, 1998, pp. 269-274.

[ENOS99] Enos, M., Hauck, S. y Sarrafzadeh, M., **"Evaluation and Optimization of Replication Algorithms for Logic Bipartitioning"**, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, nº 9, 1999, pp. 1237-1248.

[FIDU82] Fiduccia, C.M. y Mattheyses, R.M., **"A Linear Time Heuristic for Improving Network Partitions"**, *Proceedings of the 19th Design Automation Conference*, 1982, pp. 175-181.

[GAJS85] Gajski, D.D., **"Silicon Compilation"**, *VLSI Systems Design*, 1985, pp. 48-64.

[GAO89] Gao, S., Kaufman, M. y Maley, F.M., **"Advances in Homotopic Layout Compaction"**, *Proceedings of the 1989 Symposium on Parallel Algorithms and Architectures*, 1989, pp. 273-282.

[GARE79] Garey, M.R. y Johnson, D.S., ***Computers and Intractability. A Guide to the Theory of NP-Completeness***, Freeman, 1979.

[GUO99] Guo, P.-N., Cheng, C.-K. y Yoshimura, T., **"An O-Tree Representation of Non-Slicing Floorplan and its Applications"**, *Proceedings of the 36th Design Automation Conference*, 1999, pp. 268-273.

[HAKI88] Hakimi, S.L., **"A Problem on Rectangular Floorplans"**, *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1988, pp. 1533-1536.

[HELL82] Heller, W.R., Sorkin, G. y Maling, K., **"The Planar Package Planner for System Designers"**, *Proceedings of the 19th Design Automation Conference*, 1982, pp. 253-260.

[HILL89] Hill, D., Shugard, D., Fishburn, J. y Keutzer, K., ***Algorithms and Techniques for VLSI Layout Synthesis***, Kluwer Academic Publishers, 1989.

[HOLL73] Holland, J., **"Genetic Algorithms and the Optimal Allocations of Trials"**, *SIAM Journal of Computing*, vol. 2, nº 2, 1973, pp. 88-105.

[IZUM98] Izumi, T., Takahashi, A. y Kajitani, Y., **"Air-Pressure-Model-Based Fast Algorithms for General Floorplan"**, *Proceedings of the Asia and South Pacific Design Automation Conference*, 1998, pp. 563-570.

[KANG97] Kang, M.Z.-W. y Dai, W.W.-M., **"General Floorplanning with L-shaped, T-shaped and Soft Blocks Based on Bounded Slicing Grid Structure"**, *Proceedings of the Asia and South Pacific Design Automation Conference*, 1997, pp. 265-270.

[KANG98] Kang, M.Z.-W. y Dai, W.W.-M., **"Arbitrary Rectilinear Block Packing Based on Sequence Pair"**, *Proceedings of the IEEE Transactions on Computer-Aided Design*, 1998, pp. 259-266.

[KERN70] Kernighan, B.W. y Lin, S., **"An Efficient Heuristic Procedure for Partitioning Graphs"**, *Bell Systems Technical Journal*, 49(2), 1970, pp. 291-307.

[KIRK83] Kirkpatrick, S., Gelatt, C.D. y Vecchi, M.P., **"Optimization by Simulated Annealing"**, *Science*, vol. 220, nº 4598, 1983, pp. 671-680.

[KORE97] Koren, Z. y Koren, I., **"On the Effect of Floorplanning on the Large Area Integrated Circuits"**, *IEEE Transactions on VLSI Systems*, vol. 5, 1997, pp. 3-14.

[KOZM84] Kozminski, K.A. y Kinnen, E., **"An Algorithm for Finding a Rectangular Dual of a Planar Graph for Use in Area Planning for VLSI Integrated Circuits"**, *Proceedings of the 21st Design Automation Conference*, 1984, pp. 655-656.

[KUH90] Kuh, E.S. y Ohtsuki, T., **"Recent Advances in VLSI Layout"**, *Proceedings of the IEEE*, vol. 78, nº 2, 1990, pp. 237-263.

[KUO96] Kuo, M.-T.K., Liu, L.-T. y Cheng, C.-K., **"Network Partitioning into Tree Hierarchie"**, *ACM/IEEE Design Automation Conference*, 1996, pp. 477-482.

[KUO97] Kuo, M.-T.K. y Cheng, C.-K., **"A New Network Flow Approach for Hierarchical Tree Partitioning"**, *ACM/IEEE Design Automation Conference*, 1997, pp. 512-517.

[LAI88] Lai, Y.T. y Leinwand, S.M., **"Algorithms for Floor-plan Design Via Rectangular Dualization"**, *IEEE Transactions on Computer-Aided Design*, vol. 7, 1988, pp. 1278-1289.

[LAPO86] LaPotin, D.P. y Director, S.W., **"MASON: A Global Floor-planning Approach to VLSI Design"**, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 5, 1986, pp. 477-489.

[LAUT80] Lauther, U., **"A Min-Cut Placement Algorith for General Cell Assemblies Based on a Graph Representation"**, *Journal of Digital Systems*, vol. IV, 1980, pp. 21-34.

[LEIN84] Leinwand, S.M. y Lai, Y.-T., **"An Algorithm for Building Rectangular Floorplans"**, *Proceedings of the 21st. Design Automation Conference*, 1984, pp. 663-664.

[LENG82] Lengauer, T., **"The Complexity of Compacting Hierarchically Specified Layouts of Integrated Circuits"**, *Proceedings of the 23th Symposium on Foundatios in Computer Science*, 1982, pp. 358-368.

[LENG90] Lengauer, T., ***Combinatorial Algorithms for Integrated Circuit Layout***, John Wiley & Sons, 1990.

[LIAO83] Liao, Y.-Z. y Wong, C.K., **"An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraints"**, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 2, 1983, pp. 62-69.

[LOKA89] Lokanathan, B. y Kinnen, E., **"Performance Optimized Floor Planning by Graph Planarization"**, *Proceedings of the 26th Design Automation Conference*, 1989, pp. 116-121.

[LOPE96] López, M.A. y Mehta, D.P., **"Efficient Decomposition of Polygons into L-Shapes with Applications to VLSI Layouts"**, *ACM Transactions on Design Automation of Electronic Systems*, vol. 1, nº 3, 1996, pp. 371-395.

[MEHT98] Mehta, D.P., **"Estimating the Storage Requirements of the Rectangular and L-shaped Corner Stitching Data Structures"**, *ACM Transactions on Design Automation of Electronic Systems*, vol. 2, 1998, pp. 272-284.

[MOH93] Moh, T.-S., ***Efficient Algorithms for Area Minimization in Floorplanning of VLSI Circuits,*** Tesis Doctoral, Universidad de California, Davis, 1993.

[OTTE82] Otten, R.H.J.M., **"Automatic Floorplan Design"**, *Proceedings of the 19th ACM/IEEE Design Automation Conference*, 1982, pp. 261-267.

[OTTE83] Otten, R.H.J.M., **"Efficient Floorplan Optimization"**, *Proceedings of the IEEE International Conference on Computer Design*, 1983, pp. 499-502.

[OTTE84] Otten, R.H.J.M. y van Ginneken, L.P.P.P., **"Floor-plan Design using Simulated Annealing"**, *Proceedings of the International Conference on Computer-Aided Design*, 1984, pp. 96-98.

[PAN95] Pan, P. y Liu, C.L., **"Area Minimization for Floorplans"**, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, nº 1, 1995, pp. 123-132.

[PAN96] Pan, P., Shi, W. y Liu, C.L., **"Area Minimization for Hierarchical Floorplans"**, *Algorithmica*, vol. 15, nº 6, 1996, pp. 550-571.

[PEDR90] Pedram, M. y Preas, B., **"A Hierarchical Floorplanning Approach"**, *Proceedings of International Conference on Computer Design*, 1990, pp. 332-338.

[PUCK94] Pucknell, D.A. y Eshraghian, K., ***Basic VLSI Design (3ª edición),*** Prentice Hall Silicon Systems Engineering Series, 1994.

[PRIC94] Price, T.E., ***Introduction to VLSI Technology***, Prentice Hall International Books, 1994.

[SAHE96a] Saheb-Zamani, M. y Hellestrand, G., **"A New Neural Network Approach to the Floorplanning of Hierarchical VLSI Designs"**, *International Conference on Neural, Parallel and Scientific Computation*, 1996, pp. 399-402.

[SAHE96b] Saheb-Zamani, M. y Hellestrand, G., **"A Stepwise Refinement Algorithm for Integrated Floorplanning, Placement and Routing of Hierarchical Designs"**, *IEEE International Symposium on Circuits and Systems*, 1996, pp. 49-52.

[SAIT99] Sait, S.M. y Youssef, H., ***Physical Design Automation: Theory and Practice***, Lecture Notes Series on Computing, World Scientific, vol. 6, 1999.

[SAKA98] Sakanushi, K., Nakatake, S. y Kajitani, Y., **"The Multi-BSG: Stochastic Approach to an Optimum Packing of Convex-Rectilinear Blocks"**, *IEEE/ACM International Conference on Computer Aided Design*, 1998, pp. 267-274.

[SARR89] Sarrafzadeh, M. y Lee, K.W., **"A New Aproach to Topological Via Minimization"**, *IEEE Transactions on Computer Aided Design*, 1989, pp. 890-900.

[SCHR86] Schrage, L., ***Linear, Integer and Cuadratic Programming with LINDO***, Scientific Press, 1986.

[SHAH90] Shahookar, K. y Mazmumder, P., **"A Genetic Approach to Standard Cell Placement Using Meta-genetic Parameter Optimization"**, *IEEE Transactions on Computer Aided Design*, 1990, pp. 500-511.

[SHER95] Sherwani, N.A., ***Algorithms for VLSI Physical Design Automation***, Kluwer Academic Publishers, 1995.

[SHI95] Shi, W., **"An Optimal Algorithm for Area Minimization of Slicing Floorplans"**, *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 1995, pp. 480-484.

[SHI96] Shi, W., **"A Fast Algorithm for Area Minimization of Slicing Floorplans"**, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1996, pp. 1525-1532.

[STOC83] Stockmeyer, L., **"Optimal Orientations of Cells in Slicing Floorplan Designs"**, *Information and Control*, vol. 59, 1983, pp. 91-101.

[SU99] Su, H.-P., Wu, A.C.-H. y Lin, Y.-L., **"A Timing-Driven Soft-Marco Resynthesis Method in Interaction with Chip Floorplanning"**, *Proceedings of the 36th Design Automation Conference*, 1999, pp. 262-267.

[SUTA91] Sutanthavibul, S., Shragowitz, E. y Rosen, J.B., **"An Analytical Approach to Floorplan Design and Optimization"**, *IEEE Transactions on Computer-Aided Design*, vol. 10, n° 6, 1991, pp. 761-769.

[TARA98] Tarafdar, S., Leeser, M. y Yin, Z., **"Integrating Floorplanning in Data-Transfer Based High-Level Synthesis"**, *IEEE/ACM International Conference on Computer Aided Design*, 1998, pp. 412-417.

[TSAY91] Tsay, R.-S. y Kuh, E.S., **"A Unified Approach to Partitioning and Placement"**, *IEEE Transactions on Circuits and Systems*, vol. 38, 1991, n° 5, pp. 521-533.

[WANG90] Wang, T.-C. y Wong, D.F., **"An Optimal Algorithm for Floorplan Area Optimization"**, *Proceedings of the 27th ACM/IEEE Design Automation Conference*, 1990, pp. 180-186.

[WANG92a] Wang, T.-C. y Wong, D.F., **"On Stockmeyer's Floorplan Optimization Technique"**, *Proceedings of the International Symposium on Circuits and Systems*, 1992, pp. 1989-1992.

[WANG92b] Wang, T.-C. y Wong, D.F., **"Optimal Floorplan Area Optimization"**, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, n° 8, 1992, pp. 992-1002.

[WANG93a] Wang, K. y Chen, W.K., **"Optimum Area Minimization for Nonslicing Floorplan"**, *IEEE Proceedings of the 36th Midwest Symposium on Circuits and Systems*, 1993, pp. 522-525.

[WANG93b] Wang, K. y Chen, W.K., **"A Class of Zero Wasted Floorplan for VLSI Design"**, *IEEE Proceedings of the International Symposium on Circuits and Systems*, 1993, pp. 1762-1765.

[WANG93c] Wang, K., ***Floorplan and Placement Algorithms in Very Large Scale Integrated (VLSI) Circuit Designs***, Tesis Doctoral, Universidad de Illinois en Chicago, 1993.

[WANG93d] Wang, T.-C., ***Efficient Algorithms for VLSI Layout Design***, Tesis Doctoral, Universidad de Texas en Austin, 1993.

[WANG95] Wang, K. y Chen, C.K., **"Floorplan Area Optimization Using Network Analogous Approach"**, *Proceedings of the International Symposium on Circuits and Systems*, vol. 5, 1995, pp. 167-170.

[WEI91] Wei, Y.C. y Cheng, C.K., **"Radio Cut Partitioning for Hierarchical Designs"**, *IEEE Transactions on Computer Aided Design*, vol. 40, n° 7, 1991, pp. 911-921.

[WEST92] Weste, N.H.E. y Eshraghian, K., ***Principles of CMOS VLSI Design: A Systems Perspective***, Addison-Wesley VLSI Systems Series, 1992.

[WIME88] Wimer, S., Koren, Y. y Cederbaum, I., **"Floorplans, Planar Graphs and Layouts"**, *IEEE Transactions on Circuits and Systems*, vol. 35, n° 3, 1988, pp. 267-278.

[WIME89] Wimer, S., Koren, Y. y Cederbaum, I., **"Optimal Aspect Ratios of Building Blocs in VLSI"**, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, 1989, pp. 139-145.

[WOLF94] Wolf, W., ***Modern VLSI Design a System Approach***, Prentice-Hall International Books, 1994.

[WONG86] Wong, D.F. y Liu, C.L., **"A New Algorithm for Floorplan Design"**, *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, 1986, pp. 101-107.

[WONG87] Wong, D.F. y Liu, C.L., **"Floorplan Design for Rectangular and L-shaped Modules"**, *Proceedings of the IEEE International Conference on Computer-Aided Design*, 1987, pp. 520-523.

[WONG89a] Wong, D.F. y Liu, C.L., **"Floorplan Design of VLSI Circuits"**, *Algorithmica*, vol. 4, nº 2, 1989, pp. 263-291.

[WONG89b] Wong, D.F. y The, K.-S., **"An Algorithm for Hierarchical Floorplan Design"**, *Proceedings of the International Conference on Computer-Aided Design*, 1989, pp. 484-487.

[WONG89c] Wong, D.F. y Sakhamuri, P.S., **"Efficient Floorplan Area Optimization"**, *Proceedings of 26th Design Automation Conference*, 1989, pp. 586-589.

[XU97] Xu, J., Guo, P.-N. y Cheng, C.-K., **"Cluster Refinement for Block Placement"**, *Proceedings of the 34th Design Automation Conference*, 1997, pp. 762-765.

[YAMA96] Yamanouchi, T., Tamakashi, K. y Kambe, T., **"Hybrid Floorplanning Based on Partial Clustering and Module Restructuring"**, *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 1996, pp. 478-483.

[YEAP93] Yeap, K.-H., ***High Level Physical Design of Very Large Scale Integrated Circuits***, Tesis Doctoral, Universidad de Northwestern en Illinois, 1993.

[YOUN97] Young, F.Y. y Wong, D.F., **"How Good are Slicing Floorplans?"**, *Integration the VLSI journal*, 1997, pp. 61-73.

[YOUN98] Young, F.Y. y Wong, D.F., **"Slicing Floorplans with Pre-placed Modules"**, *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 1998, pp. 252-258.

[YOUN99] Young, F.Y. y Wong, D.F., **"Slicing Floorplans with Range Constraint"**, *Proceedings of the International Symposium on Physical Design*, 1999, pp. 97-102.

[ZHAN91] Zhang, C.X., Vogt, A. y Mlynski, D.A., **"Floor-plan Design Using a Hierarchical Neural Learning Algorithm"**, *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1991, pp. 2060-2063.

[ZOBR94] Zobrist, G.W., ***Routing, Placement and Partitioning***, VLSI Design Automation Series, Ablex Publishing Corp., 1994.