DEPARTAMENTO DE ESTADÍSTICA E INVESTIGACIÓN OPERATIVA

ALGORITMOS HEURÍSTICOS Y EXACTOS PARA PROBLEMAS DE CORTE NO GUILLOTINA EN DOS DIMENSIONES

FRANCISCO PARREÑO TORRES

UNIVERSITAT DE VALENCIA
Servei de Publicacions
2004

Aquesta Tesi Doctoral va ser presentada a Valencia el día 05 de Març de 2004 davant un tribunal format per:

- D. Laureano Escudero Bueno
- D. Enrique Benavent López
- D. José Fernando Da Costa Oliveira
- D. José Antonio Gamez Martín
- D. Rafael Martí Cunquero

Va ser dirigida per:
D. Ramón Álvarez – Valdés Olaguibel

Depòsit legal:
I.S.B.N.:84-370-5474-5
Edita: Universitat de València
Servei de Publicacions
C/ Artes Gráficas, 13 bajo
46010 València
Spain
Telèfon: 963864115

VNIVERSITAT DE VALÈNCIA

Facultat de Ciències Matemàtiques

Departament d'Estadística i Investigació Operativa

Algoritmos heurísticos y exactos para problemas de corte no guillotina en dos dimensiones

Memoria presentada por

Francisco Parreño Torres

para optar al grado de Doctor en

Ciencias Matemáticas

Dirigida por:

Dr. Ramón Álvarez-Valdés Olaguíbel

Dr. José Manuel Tamarit Goerlich

II

Dr. Ramón Álvarez-Valdés Olaguíbel y Dr. José Manuel Tamarit Goerlich, profesores titulares de Estadística e Investigación Operativa del Departament d'Estadística i Investigació Operativa de la Universitat de València,

CERTIFICAN: Que la presente memoria de investigación:

" Algoritmos heurísticos y exactos para problemas de corte no guillotina en dos dimensiones"

ha sido realizado bajo su dirección en el Departament d'Estadística i Investigació Operativa por Dn. Francisco Parreño Torres, y constituye su tesis para optar al grado de Doctor en Ciencias Matemáticas.

Y para que así conste, en cumplimiento de la normativa vigente, autorizan su presentación ante la Facultat de Matemàtiques de la Universitat de València para que pueda ser tramitada su lectura y defensa pública.

Burjassot, 15 de Diciembre de 2003

LOS DIRECTORES

Ramón Álvarez-Valdés Olaguíbel José Manuel Tamarit Goerlich

A la memoria de mi padre.
Sin su esfuerzo, consejo y ejemplo
jamás habría hecho este trabajo.

Agradecimientos

Llegado este momento me acuerdo de muchas de las personas que han hecho posible este trabajo y por ello quiero darles las gracias:

A mis directores, Ramón Álvarez-Valdés Olaguíbel y José Manuel Tamarit Goerlich por iniciarme en la investigación científica, por enseñarme a ser mejor persona y por soportarme. Ha sido y seguirá siendo un orgullo trabajar con ellos.

A mi madre, por todos los sacrificios, esfuerzos y desvelos que ha hecho por mis hermanos y por mí, antes junto a mi padre y ahora en soledad. Todo lo que soy, o alguna vez seré, ha sido gracias a ellos.

A mis hermanos, Consuelito, Alfon y Jose por reír y llorar juntos, y por ser los más fieles espectadores de mis hazañas intelectuales.

A Pi, por esa sonrisa maravillosa que hace los momentos difíciles mucho más fáciles. Por su apoyo, cariño y comprensión en todo momento. `I.L.Y.`

A mis abuelos, tíos, primos y al resto de familia por su apoyo incondicional.

A mis amig@s, por esos ratos de cenas, comidas, cafés, bares, cervezas, discusiones, becerradas, bares, partidos, partidas, fiestas, bares, bares y más bares etc.. Y por estar siempre ahí, en otros momentos no tan placenteros.

A todo el Departamento de Estadística e Investigación Operativa de la Universidad de Valencia, si su nivel académico y científico es enorme, el personal está muy por encima. De una forma especial a Mario y Rafa.

A mis compañer@s de doctorado, por hacer el trabajo mucho más llevadero, muy especialmente a Antonio por compartir conmigo tantas y tantas horas de trabajo.

A todos, muchas gracias.

Francisco Parreño Torres

Diciembre 2003.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

En muchas aplicaciones de la vida real relacionadas con los procesos industriales surgen problemas que la Investigación Operativa engloba bajo el epígrafe de *Problemas de Corte y Empaquetamiento.* Por una parte, muchos procesos de fabricación producen tableros o láminas de grandes dimensiones de madera, metal, papel, plástico o vidrio que luego han de ser cortados en piezas más pequeñas para ajustarse a las necesidades de los clientes. Por otra parte, en muchas ocasiones unidades de almacenamiento grandes, tales como pallets o contenedores son utilizadas para transportar objetos más pequeños.

Estos problemas, a primera vista dispares, están conceptualmente muy relacionados debido a la dualidad entre el material y el espacio ocupado por éste (Dyckhoff (1988)[37]). En ambos casos, existen dos tipos de objetos, grandes y pequeños, y el espacio definido por el objeto grande ha de ser ocupado por los objetos pequeños siguiendo ciertas normas. En ambos tipos de problemas, el ajuste entre objetos grandes y pequeños ha de ser lo más eficiente posible, de acuerdo con la función objetivo establecida, que en muchos casos se reducirá a minimizar el espacio no utilizado.

Se trata de problemas de Optimización Combinatoria. Como ocurre con muchos otros problemas de esta clase, son problemas fáciles de definir intuitivamente, aunque no siempre fáciles de modelizar formalmente. A partir de unos modelos

básicos, existen gran cantidad de variantes, derivadas de la gran cantidad de aplicaciones prácticas existentes. Desde el punto de vista de la complejidad, como ocurre en muchos otros casos, existen algunos casos particulares fácilmente resolubles, pero los problemas importantes son difíciles y siguen atrayendo el interés de los investigadores, que continúan desarrollando algoritmos exactos y heurísticos cada vez más eficientes.

Dyckhoff (1990)[38] desarrolló un esquema de clasificación que generaliza la clasificación presentada por Hinxman (1977)[50]. Este esquema permite identificar las propiedades más comunes de los problemas de corte o empaquetamiento según las siguientes características:

1. *Dimensionalidad*
   (N) Número de dimensiones.

2. *Tipo de asignación*
   (B) Todos los tableros y una parte de las piezas demandadas.
   (V) Una parte de los tableros y todas las piezas demandadas.

3. *Surtido de tableros almacenados*
   (O) Un tablero.
   (I) Tableros idénticos.
   (D) Tableros diferentes.

4. *Surtido de piezas demandadas*
   (F) Pocas piezas de diferentes tamaños.
   (M) Muchas piezas de muchos tamaños.
   (R) Muchas piezas de relativamente pocas dimensiones.
   (C) Muchas piezas, pero idénticas.

En los últimos años se han publicado numerosos artículos sobre problemas de empaquetamiento y corte. Pueden consultarse las excelentes revisiones bibliográficas de Dowsland (1992)[34], Sweeney y Paternoster (1992)[82] y Wang y Wäscher (2002)[87].

La configuración de las piezas en el tablero en el que se cortan o se empaquetan constituye lo que denominaremos *patrón*. En general, los patrones serán *ortogonales*, indicando que las cajas están colocadas con sus lados paralelos a los lados del tablero. Patrones no-ortogonales surgen cuando se coloca una pieza de dimensiones mayores que el largo y ancho del tablero.

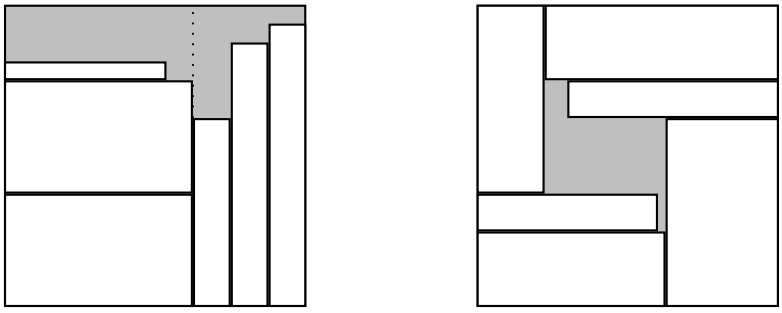
Morabito y Morales(1998)[62], [63] hacen la siguiente clasificación de los patrones de empaquetamiento:

- *Patrones guillotina y no guillotina*:

  Un corte es de tipo *guillotina* si cuando se aplica sobre un rectángulo produce dos nuevos rectángulos, es decir, si el corte va de un extremo a otro del rectángulo original; en otro caso se denomina de tipo *no guillotina*.
  Un patrón es de tipo guillotina si se puede obtener por sucesivos cortes de tipo guillotina (Figura 1.1(a), 1.2(a)). Un patrón es *no guillotina* si es obtenido por sucesivos cortes de guillotina y no guillotina (Figura 1.1(b), 1.2(b)).

- *Patrones de primer orden y de orden superior*:

  Un corte es de *primer orden no guillotina* si cuando lo aplicamos en un rectángulo produce cinco nuevos rectángulos de tal manera que no forma un patrón de guillotina (Figura 1.1(b)). Un patrón es llamado de primer orden no guillotina si es obtenido por sucesivos cortes guillotina y/o cortes de primer orden no guillotina (Figura 1.2(b)). Obviamente hay patrones no guillotina que no son de primer orden que son denominados de *orden superior* (Figura 1.2(c)).



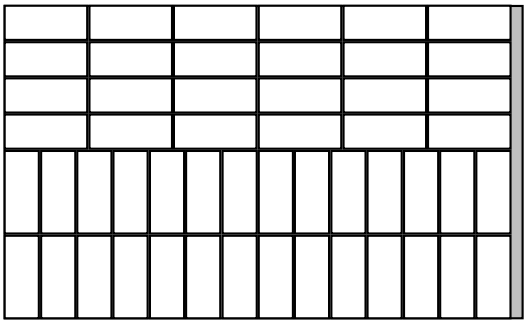
(a) Corte Guillotina (b) Corte no guillotina

Figura 1.1: *Tipos de Cortes.*

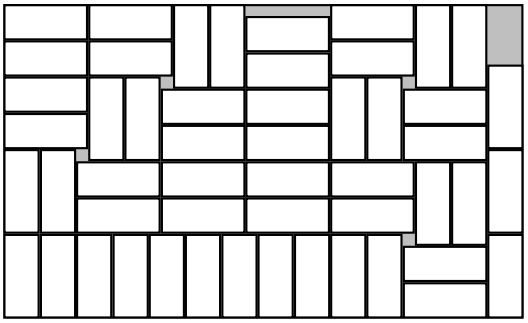
1.1. Objetivos y estructura de la memoria presentada

- En la primera parte de nuestro trabajo estudiaremos el problema del pallet (*Pallet Loading Problem (PLP)*). Se trata de un problema clásico de empaquetamiento en el que se ha de maximizar el número de cajas idénticas que pueden colocarse sobre un pallet, en la clasificación de Dyckhoff *(2/B/O/C)*. El problema del pallet aparece en el transporte desde una fábrica a las distribuidoras más pequeñas, por lo que también es conocido como el *manufacturer's pallet loading* (MPL), ya que en las fábricas generalmente se preparan pallets con un solo tipo de producto.

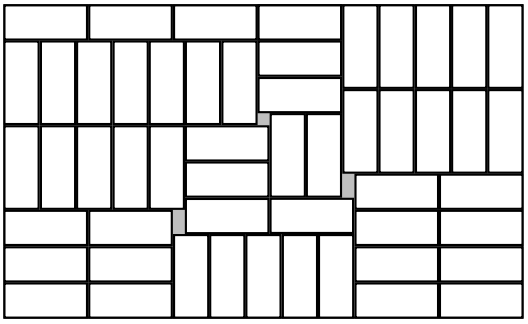
- En la segunda parte estudiaremos una generalización del problema de empaquetamiento anterior, en el que se tienen cajas de diferentes tamaños que se han de colocar sobre el mismo pallet, en la clasificación de Dyckhoff *(2/B/O/M)*. En la literatura, para distinguirlo del problema anterior, se le conoce como el problema del distribuidor, *distributor's pallet loading* (DPL), que combina sobre el mismo pallet diferentes tipos de productos.



(a) Patrón Guillotina



(b) Patrón de primer orden



(c) Patrón de orden superior

Figura 1.2: *Tipos de Patrones.*

Este problema también puede verse como un problema de corte no guillotina de piezas de diferentes tamaños. Se trata en este caso de un problema claramente en la frontera entre corte y empaquetamiento y ha sido estudiado desde ambos puntos de vista.

En ambos problemas, dependiendo de la escala del canal de distribución y producción, un pequeño incremento en el número de productos cargados en cada pallet puede ocasionar substanciales beneficios. Por esta razón tienen una considerable importancia económica.

La memoria se estructura de la forma siguiente:

- Iniciamos el **capítulo 2** con una descripción del *PLP* y sus propiedades. Después, ofrecemos una visión exhaustiva de los métodos de solución que se han seguido para resolverlo.

  Cuando describimos los métodos existentes para resolver el problema, explicaremos los principales procedimientos heurísticos de la literatura, así como las diferentes cotas superiores que han sido propuestas. Finalmente haremos una revisión de los métodos exactos existentes hasta este momento.

- En el **capítulo 3** desarrollamos un nuevo procedimiento exacto, basado en *Branch and Cut*, que permite la resolución de problemas más grandes que los resueltos óptimamente hasta el momento. Comenzamos describiendo brevemente los elementos de un *Branch and Cut* y a continuación los procedimientos usados para cada componente de nuestro algoritmo.

- En el **capítulo 4** concluimos el trabajo para el *PLP* desarrollando un procedimiento metaheurístico, basado en *Tabu Search*, con el que se obtienen resultados de gran calidad en tiempos moderados de computación para los problemas existentes en la literatura y un conjunto nuevo de problemas test con un mayor número de cajas.

- En el **capítulo 5** estudiamos el problema con cajas de diferentes tamaños, en el que aplicamos con éxito algunas de las ideas utilizadas en capítulos anteriores. Desarrollamos un algoritmo *Grasp (Greedy Randomized Adaptive Search Procedure)* de propósito general para cualquier tipo de problemas que produce excelentes resultados en tiempos de computación muy breves.
- En el **capítulo 6** presentamos las conclusiones y se plantean las futuras líneas de investigación.

Capítulo 2

El problema del Pallet

2.1. Descripción del problema

La primera parte de esta tesis está dedicada al problema de empaquetamiento conocido como problema del pallet, *pallet loading problem (PLP)*. El problema es el siguiente: tenemos un producto empaquetado en cajas de igual tamaño que se han de cargar en pallets rectangulares y el objetivo consiste en maximizar el número de cajas cargadas en el pallet, que es usado para el transporte y almacenamiento de productos.

Aunque a primera vista parece ser un problema de empaquetamiento en tres dimensiones, consideraciones prácticas (Dowsland (1985)[20], (1995)[36]) como:

- estabilidad,
- fácil manejo para la carga y posterior descarga de los productos del pallet

permiten pasar del problema de tres dimensiones a un problema de dos dimensiones, dado que la mayoría de pallets se cargan en capas con la misma configuración (Figura 2.1).

El problema del pallet se define formalmente de la siguiente manera: dado un rectángulo, llamado *pallet*, de longitud L y anchura W, con $L \geq W$, L y W enteros, y un rectángulo menor, llamado *caja*, con longitud l y anchura w, con

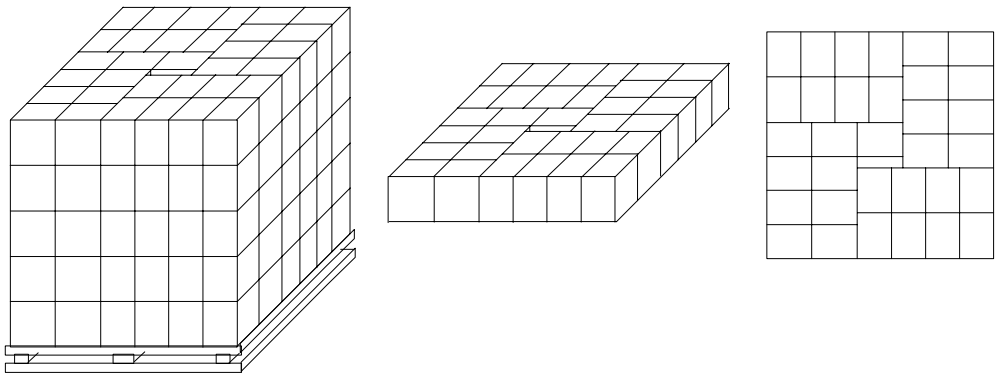


Figura 2.1: *Reducción del problema a dos dimensiones.*

$l \geq w$, $L \geq l$ y $W \geq w$, l y w enteros, se trata de *maximizar el número de cajas que pueden ser colocadas ortogonalmente en el pallet.*

A partir de ahora denotaremos una instancia del *PLP* como una cuádrupla *(L,W,l,w).*

Aunque aparentemente el problema parece muy fácil de resolver de manera óptima, no existe hasta la fecha ningún algoritmo exacto en tiempo polinómico y se piensa que el *PLP* es NP-completo, pero no se ha podido demostrar todavía (Nelißen (1993)[66]).

Un argumento a favor de la NP-completitud es el aportado por Baker et al. (1980)[5]. Supongamos el siguiente procedimiento para empaquetar el pallet con n cajas (suponiendo que existe un diseño con n cajas, al menos, para el pallet): para cada caja se decide primero si es posible colocarla de forma vertical u horizontal. Se coloca la caja lo más hacia abajo y hacia la izquierda del pallet. Este procedimiento es una adaptación del algoritmo de *bottom-left* para el $bin - packing$ en dos dimensiones. De esta forma hay 2^n posibles secuencias de colocaciones de las n cajas y el procedimiento estudiará 2^n diseños diferentes. Dado que, en la mayoría de los casos, las cajas podrán ser movidas en ambas dimensiones, ha-

brá más de $\Omega(2^n)$ posibles empaquetamientos. Sin embargo, este argumento no es en absoluto concluyente y el problema sigue abierto.

Una versión restringida del *PLP* es el problema del *PLP-guillotina* donde sólo son permitidos cortes de tipo guillotina. En 1994, Scheithauer et al.[83] demostraron que el *PLP-guillotina* es resoluble en tiempo polinómico. Sin embargo, la solución óptima del *PLP* coincide con la solución óptima del *PLP-guillotina* únicamente en el 80 % de los casos.

Matemáticamente, el estudio del *problema del pallet* se puede incluir dentro de la teoría de optimización combinatoria dado que tenemos una función objetivo a maximizar y un conjunto de soluciones finito, aunque considerablemente grande.

Como ocurre en otros problemas de optimización combinatoria el *PLP* tiene la particularidad de que en la mayoría de las instancias resulta muy sencillo encontrar una solución óptima. Pero existen algunas instancias, quizás menos del 1 %, en las que encontrar una solución óptima puede ser extremadamente difícil.

2.2. Clases de equivalencia

Un estudio de las propiedades del *PLP* necesariamente ha de comenzar describiendo el concepto de clases de equivalencia y sus implicaciones algorítmicas. El concepto y su desarrollo se deben a K. Dowsland (1984)[26]. Comenzamos con las definiciones básicas:

Sea el problema *(L,W,l,w)* y sea *(n,m)* un par ordenado de enteros no negativos satisfaciendo:

$$n * l + m * w \leq S, \tag{2.1}$$

para un lado del pallet *S (L ó W)*. Entonces *(n, m)* es una *partición posible* de *S*, es decir, es una combinación de segmentos de longitud l ó w que pueden ser acomodados en la dimensión de *S*. Si n y m satisfacen:

$$0 \leq S - n * l - m * w < w, \tag{2.2}$$

entonces *(n,m)* es una *partición eficiente* de S. Si además, n y m satisfacen:

$$0 = S - n * l - m * w, \tag{2.3}$$

entonces *(n,m)* es una *partición perfecta* de S.

Existe una partición eficiente para cada n satisfaciendo:

$$0 \leq n \leq \lfloor S/l \rfloor \tag{2.4}$$

El conjunto de todas las particiones posibles de L por l y w será denotado por *F(L,l,w)* y el correspondiente conjunto de particiones eficientes por *E(L,l,w)*. K. Dowsland (1984)[26] demostró el siguiente teorema y los lemas consecuentes:

Teorema 2.1 *El conjunto de diseños posibles (o soluciones posibles) para un pallet está totalmente definido por el conjunto de particiones eficientes.*

Lema 2.1 *Para cualquier diseño del pallet, hay un diseño correspondiente obtenido al colocar todas las cajas lo más a la izquierda posible, en el que la distancia del lado derecho de cualquier caja al lado izquierdo del pallet puede ser expresado como $n * l + m * w$ para enteros no negativos n, m.*

Lema 2.2 *Dada una instancia (L,W,l,w) y un diseño de las cajas como en el lema anterior, y una segunda instancia (V,Y,c,d) satisfaciendo F(L,l,w)=F(V,c,d) y F(W,l,w)=F(Y,c,d) entonces existe un diseño correspondiente para el conjunto (V,Y,c,d).*

Lema 2.3 *Si dos instancias del PLP son equivalentes, es decir, tienen el mismo conjunto de particiones posibles, entonces tienen el mismo conjunto de soluciones posibles y por lo tanto el mismo conjunto de soluciones óptimas.*

Lema 2.4 *Si E(L,l,w)=E(V,c,d) entonces F(L,l,w)=F(V,c,d). Cualquier par de problemas con el mismo conjunto de particiones eficientes tendrán el mismo conjunto de particiones posibles.*

En resumen, cada instancia *(L,W,l,w)* del *PLP* pertenece a una clase de equivalencia definida por los conjuntos de particiones eficientes *E(L,l,w)* y *E(W,l,w)*.

Una instancia puede tener más de una solución óptima. Para reducir el conjunto de soluciones posibles y soluciones óptimas para una instancia del *PLP*, podemos aplicar un procedimiento de *normalización*. Para cada solución S, existe una solución S' con el mismo número de cajas y la propiedad de que cada esquina inferior izquierda de cada caja (x_u, y_u) satisface las igualdades:

$$x_u = p * l + q * w$$
$$y_u = r * l + s * w$$

para enteros no negativos p, q, r y s, es decir, la posición de cada esquina de cualquier caja es una combinación entera de la anchura y longitud de la caja. Para conseguir S', simplemente se colocan todas las cajas recursivamente tan a la izquierda y abajo como sea posible sin solapamientos. Por lo tanto es suficiente considerar sólo tales empaquetamientos normalizados. Cada solución normalizada, especialmente las óptimas, pueden ser obtenidas por el procedimiento 'izquierda-abajo' mencionado anteriormente.

Dada una solución óptima para una instancia, se puede calcular la solución equivalente para cualquier otra instancia de su clase de equivalencia por un procedimiento sencillo, descrito en Dowsland (1984)[26]. Básicamente nos quedamos con la posición (expresada en forma de partición) de cada caja respecto al lado izquierdo e inferior del pallet y colocamos una caja en la misma posición para la otra instancia. Como consecuencia directa, es suficiente buscar una solución óptima o una cota superior para un miembro de la clase de equivalencia, ya que será válida para cualquier otra instancia.

K. Dowsland estableció ciertas condiciones para comprobar la equivalencia de dos instancias. Dos instancias *(L,W,l,w)* y *(L',W',l',w')* son equivalentes, si y

sólo si se satisfacen las siguientes condiciones:

$$n * l' + m * w' \leq L' \qquad \forall (n,m) \in E(L,l,w) \tag{2.5}$$

$$n * l' + (m+1) * w' > L' \qquad \forall (n,m) \in E(L,l,w) \tag{2.6}$$

$$(\lfloor \frac{L}{l} + 1 \rfloor) * l' > L' \tag{2.7}$$

junto con un conjunto similar de condiciones para W' y $E(W', l', w')$. Dado que, multiplicar todas las dimensiones por una constante c, obviamente produce un problema equivalente, cualquier conjunto de dimensiones se puede representar, salvo un factor escalar, por una terna *(L/w,W/w,l/w)*, es decir un punto en el espacio tridimensional. Las ecuaciones (2.5), (2.6) y (2.7) del conjunto de desigualdades serán consideradas como una serie de planos del espacio tridimensional cuya envoltura convexa de su intersección produce un poliedro semiabierto. Todas las instancias que puedan ser representadas como un punto dentro del mismo poliedro son equivalentes.

Ejemplo 2.1 *Tenemos las instancias (23099,18480,4620,4619) y (29,24,6,5), estudiemos cuáles son sus conjuntos de particiones eficientes:*

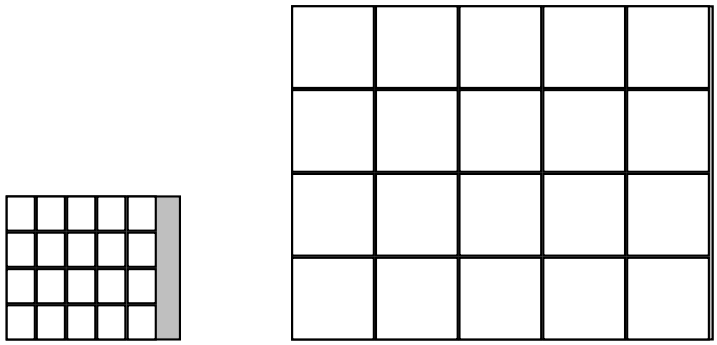
E(23099,4620,4619):={(0,5),(1,4),(2,3),(3,2),(4,1)}:=E(29,6,5)
E(18480,4620,4619):={(0,4),(1,3),(2,2),(3,1),(4,0)}:=E(24,6,5)

Como las dos tienen el mismo conjunto de particiones eficientes son equivalentes. La Figura 2.2 muestra la solución óptima de las dos instancias.

2.3. Instancias utilizadas

Las instancias que se han utilizado en la literatura para probar los distintos algoritmos son de dos tipos:
Tipo I:

$$1 \leq \frac{L}{W} \leq 2 \qquad 1 \leq \frac{l}{w} \leq 4 \qquad 1 \leq \frac{L * W}{l * w} < 51 \tag{2.8}$$



(a) (29,24,6,5) (b) (23099,18480,4620,4619)

Figura 2.2: *Instancias equivalentes.*

Tipo II:

$$1 \leq \frac{L}{W} \leq 2 \qquad 1 \leq \frac{l}{w} \leq 4 \qquad 51 \leq \frac{L * W}{l * w} < 101 \tag{2.9}$$

La definición de estos conjuntos se debe a que la mayoría de las aplicaciones reales del *PLP* están dentro de estos rangos.

Teniendo en cuenta la relación de equivalencia definida en la sección anterior, parece claro que cualquier estudio computacional debería incluir como máximo un representante de cada clase de equivalencia. Si simplemente generamos instancias aleatorias, podemos estar generando diversas instancias de una misma clase de equivalencia y resolviendo más de una vez el mismo problema, lo que distorsionaría las conclusiones sobre la eficiencia de los algoritmos.

K. Dowsland (1987)[29] desarrolló un procedimiento para generar las distintas clases de equivalencia entre unos rangos establecidos. El método es el siguiente: a partir de las ecuaciones (2.5), (2.6) y (2.7) tenemos que cada clase de equivalencia

está acotada por un conjunto de restricciones de la forma:

$$n * l + m * w \leq L \qquad \forall(n, m) \tag{2.10}$$

$$n * l + (m + 1) * w > L \qquad \forall(n, m) \tag{2.11}$$

$$(\lfloor n_{max} + 1 \rfloor) * l > L \qquad n_{max} = \lfloor L/l \rfloor \tag{2.12}$$

junto con unas similares para W. Cada región está acotada por al menos una expresión de la forma

$$L = n_x * l + m_x * w \tag{2.13}$$

$$W = j_y * l + k_y * w \tag{2.14}$$

y se pasa de una región a otra cuando:

$$n * l + m * w = n_x * l + m_x * w \tag{2.15}$$

ó

$$j * l + k * w = j_y * l + k_y * w \tag{2.16}$$

es decir, cuando el cociente de la caja l/w satisface:

$$l/w = (m - m_x)/(n_x - n) = (m_x - m)/(n - n_x) \tag{2.17}$$

ó

$$(k - k_y)/(j_y - j) = (k_y - k)/(j - j_y) \tag{2.18}$$

Como las ecuaciones (2.8)-(2.9) de cada tipo de problemas definen valores máximos para todos los problemas L', W', l', w' en la región definida, es posible determinar unos valores máximos para las variables en (2.13) y en (2.14), dados por:

$$m_x \leq L/w \qquad n_x \leq L/l \qquad k_y \leq W/w \qquad j_y \leq W/l \tag{2.19}$$

$$m \leq L/(w + 1) \qquad n \leq L/(l + 1) \qquad k \leq W/(w + 1) \qquad j \leq W/(l + 1) \tag{2.20}$$

Como todas las variables son enteras no negativas, estos límites definen un conjunto finito de cocientes l/w en los cuales pasamos de una región a otra (de una

clase de equivalencia a otra). Por tanto, podremos determinar una cota superior para el numerador *cmax* y el denominador *dmax* dentro de cada tipo de problemas, a partir de las ecuaciones (2.8) junto con (2.17) y (2.18). El máximo numerador ocurre cuando $m = [L/w] + 1$, $m_x = 0$. Entonces:

$$c_{max} = \text{máx}\{[L/w] + 1\} \tag{2.21}$$

$$c_{max} \leq \text{máx}\{\sqrt{(LW/lw)(L/W)(l/w)} + 1\} \tag{2.22}$$

$$c_{max} \overset{Tipo\ I}{\leq} \text{máx}\{[\sqrt{51 * 2 * 4}] + 1\} = \text{máx}\{\sqrt{408} + 1\} = 21 \tag{2.23}$$

$$c_{max} \overset{Tipo\ II}{\leq} \text{máx}\{[\sqrt{101 * 2 * 4}] + 1\} = \text{máx}\{\sqrt{808} + 1\} = 29 \tag{2.24}$$

Un cálculo similar para dmax

$$d_{max} = \text{máx}\{[L/l] + 1\} \tag{2.25}$$

$$d_{max} \leq \text{máx}\{\sqrt{(LW/lw)(L/W)(w/l)} + 1\} \tag{2.26}$$

$$d_{max} \overset{Tipo\ I}{\leq} \text{máx}\{[\sqrt{51 * 2 * 1}] + 1\} = \text{máx}\{\sqrt{102} + 1\} = 11 \tag{2.27}$$

$$d_{max} \overset{Tipo\ II}{\leq} \text{máx}\{[\sqrt{101 * 2 * 1}] + 1\} = \text{máx}\{\sqrt{202} + 1\} = 15 \tag{2.28}$$

Por lo tanto el conjunto finito de cocientes para las clases de equivalencia de *Tipo I*, donde nos movemos de una región a otra, son los siguientes:

$$\begin{aligned} c/d \qquad & s.t: \\ & c \leq 21 \\ & d \leq 11 \\ & 1 \leq c/d \leq 4 \end{aligned} \tag{2.29}$$

siendo c, d enteros no negativos. La condición c, d co-primos se añade para eliminar cocientes duplicados. De esta forma se obtienen los 78 valores que aparecen en la Tabla 2.1. El objetivo es cubrir cada región con un cociente. Puesto que los intervalos son abiertos puede ocurrir que no se cubran todas las regiones. Para evitar esto tomamos un conjunto de cocientes c'/d' obtenido ordenando los cocientes anteriores de forma ascendente y eligiendo un punto entre cada par

c	d	c	d	c	d	c	d	c	d
1	1	11	4	11	6	9	8	13	10
2	1	13	4	13	6	11	8	17	10
3	1	15	4	17	6	13	8	19	10
4	1	6	5	19	6	15	8	21	10
3	2	7	5	8	7	17	8	12	11
5	2	8	5	9	7	19	8	13	11
7	2	9	5	10	7	21	8	14	11
4	3	11	5	11	7	10	9	15	11
5	3	12	5	12	7	11	9	16	11
7	3	13	5	13	7	13	9	17	11
8	3	14	5	15	7	14	9	18	11
10	3	16	5	16	7	16	9	19	11
11	3	17	5	17	7	17	9	20	11
5	4	18	5	18	7	19	9	21	11
7	4	19	5	19	7	20	9		
9	4	7	6	20	7	11	10		

Tabla 2.1: *Soluciones del problema 2.29.*

adyacente. Se añaden los dos extremos 1/1 y 4/1 y se obtiene un total de 79 cocientes. Estos 79 cocientes dan un total de 79 planos en el espacio tridimensional, los cuales intersectan con las ecuaciones del *Tipo I* al menos una vez. El conjunto de todos los elementos de cada tipo se obtiene colocando las dimensiones de las cajas iguales a *c'*, *d'* y considerando todos los problemas con dimensiones de la forma:

$$\begin{aligned} L &= n * c' + m * d' \\ W &= j * c' + k * d' \end{aligned}$$

Para todos los enteros n, m, k, j para los cuales resulten L y W en el rango definido por las ecuaciones (2.8). K. Dowsland aplicó estas etapas y obtuvo 8565

clases de *Tipo I*. Sin embargo, nosotros repetimos el procedimiento y obtuvimos "únicamente" 7827 clases de equivalencia distintas. Mediante correo electrónico nos pusimos en contacto con K. Dowsland y nos comentó que, quizás en los 8565, había algunos que no cumplen exactamente esos rangos y que los 8565 no los podía volver a reproducir. Como además, el conjunto de 8565 clases de Dowsland, no ha podido ser reproducido por ningún autor posterior y, sin embargo, sí aparecen las 7827 clases que hemos generado, pensamos que estas 7827 clases son el conjunto completo para los problemas que cumplen estrictamente las desigualdades que definen el *Tipo I*. En estas 7827 estarán todas las clases de equivalencia, cuyo representante menor en relación a las dimensiones de la caja esté dentro del rango dado. Pueden existir clases de equivalencia que tengan representantes fuera y dentro de los rangos dados. Por ejemplo, el problema (24905,24905,3558,3557) está dentro del conjunto de problemas de *Tipo I*, pero el representante con menores dimensiones (55,55,8,7) no, ya que $(L*W)/(l*w) = (55*55)/(8*7) = 54,017$. En ese caso, no se incluye en el conjunto.

Para los problemas del *Tipo II* repetimos el proceso y obtuvimos 40609 clases de equivalencia distintas. Tenemos por tanto un conjunto variado de problemas test. Las diferentes clases de equivalencia serán las instancias donde validar los futuros algoritmos propuestos para el *PLP*.

2.4. Heurísticos

En esta sección describimos los diferentes métodos heurísticos propuestos en la literatura para resolver el problema del pallet.

2.4.1. Algoritmo de Steudel

El primer heurístico para el *PLP* es el propuesto por Steudel (1979)[80], basado en programación dinámica. Consiste en un procedimiento recursivo sencillo que determina la colocación de las cajas de forma que se maximize la utilización

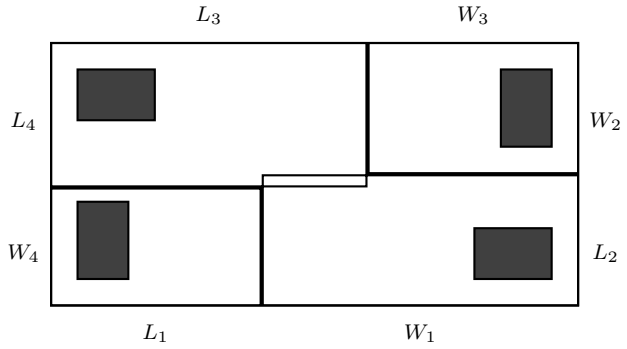


Figura 2.3: *Esquema del algoritmo de Steudel.*

del perímetro del pallet. Utiliza la siguiente ecuación recursiva:

$$F_n(S_n) = \text{máx}[L_n * l + W_n * w + F_{n-1}(S_{n-1})] \tag{2.30}$$

sujeto a:

$$L_n * l + W_n * w \leq D_n, \qquad n = 1, \ldots, 4, \tag{2.31}$$

donde:

- $F_n(S_n)$: el valor máximo de la suma de la longitud y anchura del lado n con estado S_n.
- L_n : el número de cajas de orientación horizontal colocadas en el lado n.
- W_n : el número de cajas de orientación vertical colocadas en el lado n.
- D_n : el tamaño del lado n (L ó W).
- S_n : el tipo de colocación para el lado n. S_n tiene tres valores posibles:

$$S_n = 1 : L_n = 0, \text{ (sólo cajas en vertical)}, \tag{2.32}$$

$$S_n = 2 : W_n = 0, \text{ (sólo cajas en horizontal)}, \tag{2.33}$$

$$S_n = 3 : L_n > 0, W_n > 0, \text{ (cajas en horizontal y en vertical)}. \tag{2.34}$$

Estos diseños de cajas a lo largo del pallet se extienden a todo el pallet mediante bloques. Entendemos por un *bloque*, un rectángulo que contiene un conjunto de cajas con la misma orientación. Una etapa posterior evita solapamientos o huecos en el centro del pallet. En la Figura 2.3 observamos la orientación de cada uno de los bloques.

2.4.2. Algoritmo de cuatro bloques

Un segundo procedimiento, propuesto por Smith y De Cani (1980)[79], es el llamado algoritmo de cuatro *bloques*. Este heurístico intenta colocar cuatro *bloques* con orientaciones como en la Figura 2.4 en las esquinas del rectángulo.

Es un algoritmo de tipo enumerativo y explora todos los posibles diseños de cuatro bloques. Es fácil ver que estas orientaciones no causan pérdida de generalidad debido a la simetría. Para explorar todos los posibles diseños de forma eficaz, utiliza las particiones *eficientes* en cuatro bucles encadenados. Un bucle externo determina el número de cajas horizontales en el bloque 1 y el número de columnas de las cajas verticales en el bloque 2. El primer bucle interno determina el número de filas de los bloques 2 y 3 a lo largo del lado derecho del rectángulo, el segundo bucle el número de columnas de los bloques 3 y 4 y el bucle más interno el número de filas de los bloques 4 y 1.

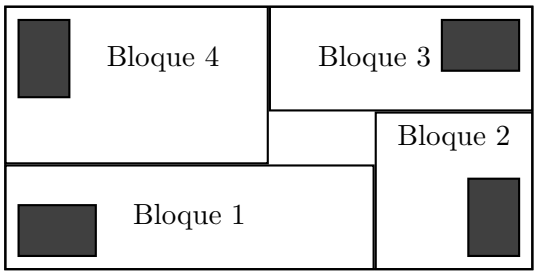


Figura 2.4: *Orientaciones de los cuatro bloques básicos.*

El heurístico de cuatro bloques sólo considera empaquetamientos donde no son posibles intersecciones entre los bloques. La Figura 2.6(a) muestra un diseño producido por este procedimiento. Los heurísticos de bloques permiten la posibilidad de que algunos de los bloques no aparezcan, produciendo soluciones que se denominan *degeneradas* (Figura 2.6(b)).

2.4.3. Algoritmo de cinco bloques

El heurístico de cinco bloques, desarrollado por Bischoff y Dowsland (1982)[16], intenta mejorar una solución con cuatro bloques rellenando el hueco que puede quedar en el centro del pallet con un quinto bloque (Figura 2.6(c)). Como en el de cuatro bloques la orientación de los bloques es fija, a excepción del bloque del centro (Figura 2.5).

El algoritmo de 5-bloques, al igual que el de 4-bloques, es un algoritmo de tipo enumerativo. Para reducir el esfuerzo computacional de estudiar todos los posibles diseños se utilizan las particiones eficientes, del mismo modo que en el algoritmo de 4-bloques.

Este método permite intersecciones entre los bloques por lo que no asegura una solución posible, pero una etapa posterior eliminaría los diseños que no producen soluciones posibles. El número de diseños examinados por el algoritmo depende del número de particiones eficientes para el pallet.

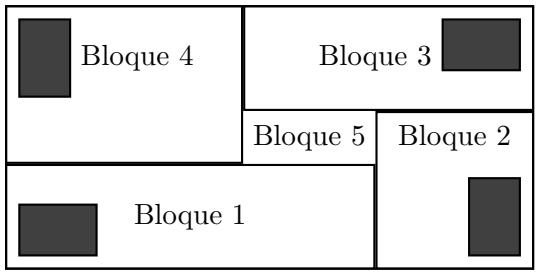
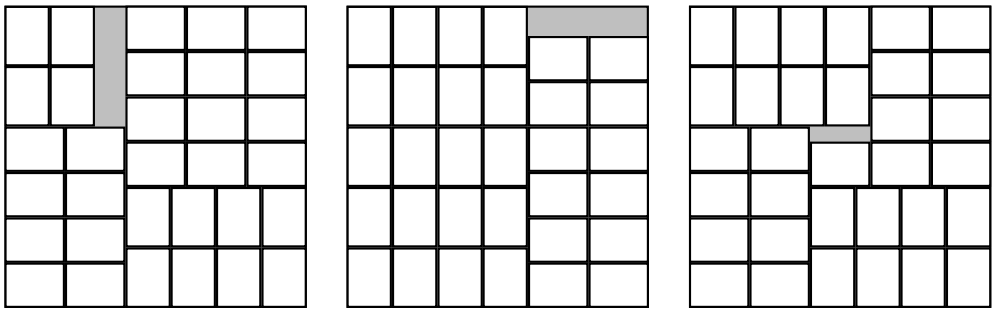


Figura 2.5: *Orientaciones de los cinco bloques básicos.*



(a) 4-bloques (b) 4-bloques degenerado (c) 5-bloques

Figura 2.6: *Diseños de bloques.*

2.4.4. Otros algoritmos de bloques

También han sido descritos en la literatura otros heurísticos con estructura de bloques. Intentando mejorar los huecos que podría ocasionar el algoritmo de 5-bloques, surgen el de siete bloques propuesto por Dowsland y Dowsland (1983)[33] y el de nueve bloques propuesto por Exeler (1991)[41].

2.4.5. Algoritmo G4

En 1996, Scheithauer y Terno [78] describen un nuevo heurístico llamado *Algoritmo G4*. Vamos a describir con detalle este algoritmo ya que se trata del mejor heurístico existente en la actualidad para resolver las instancias de *Tipo I* y *Tipo II*.

Introducimos en primer lugar la formulación y notación necesaria para la explicación de este heurístico.
Sea A=$\{((x_i, y_i, o_i) : i = 1, ..., n\}$ un patrón del PLP, donde tenemos la posición de n cajas (x,y) y su orientación o. Por simplicidad, en adelante definiremos l_i y

w_i de la i-ésima caja como:

$$l_i = \begin{cases} l & \text{si } o_i\text{=horizontal} \\ w & \text{si } o_i\text{=vertical} \end{cases} \qquad w_i = \begin{cases} w & \text{si } o_i\text{=horizontal} \\ l & \text{si } o_i\text{=vertical} \end{cases} \tag{2.35}$$

Definición 2.1 *Un subconjunto* $\tilde{A}$=$A(\tilde{I}) = \{(x_i, y_i, o_i) : i \in \tilde{I}\}$ *de* A *es un* patrón de bloque *si existe un subconjunto* $\tilde{I}$ *de* $I = \{1, ..., n\}$ *y un rectángulo* $R(\underline{x}, \underline{y}, \overline{x}, \overline{y}) := \{(x, y) \in \Re^2 : \underline{x} \leq x \leq \bar{x}, \underline{y} \leq y \leq \bar{y}\}$ *tal que:*
$\cup_{i \in \tilde{I}} (x_i, y_i, o_i) \subset R(\underline{x}, \underline{y}, \bar{x}, \bar{y})$ *y* $\cup_{i \in I \setminus \tilde{I}} (x_i, y_i, o_i) \cap \operatorname{int} R(\underline{x}, \underline{y}, \bar{x}, \bar{y}) = \emptyset$

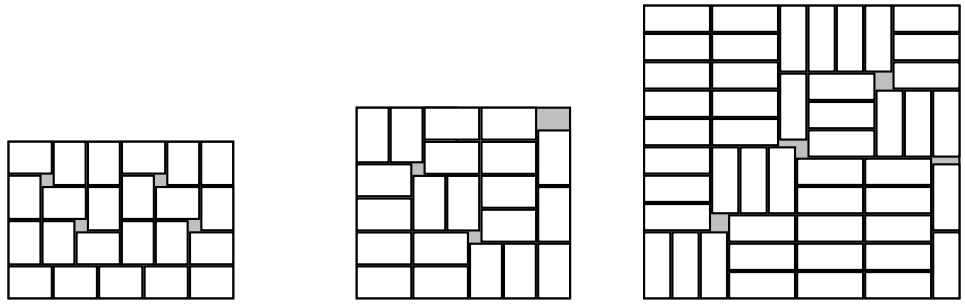
Cuando todos las cajas de un patrón tienen la misma orientación lo llamamos un *patrón homogéneo*. Para caracterizar las diferentes estructuras de patrones introducimos las siguientes definiciones.

Definición 2.2 *Un patrón de empaquetamiento* A=$\{(x_i, y_i, o_i$ *): i=1, ..., n*$\}$ *de un pallet (L,W,l,w) tiene estructura guillotina (estructura-G) si* $n \leq 3$ *o si existe una partición* I_1, $I_2 = I \setminus I_1$ *de* $I = \{1, ..., n\}$ *tal que ambos* $A(I_1)$ *y* $A(I_2)$ *forman patrones de bloque con estructura-G.*

Definición 2.3 *Un patrón homogéneo se dice que tiene estructura de 1-bloque. Para* $k \geq 2$*, un patrón de empaquetamiento* A=$\{(x_i, y_i, o_i)$*: i=1, ..., n*$\}$ *tiene estructura de k-bloque si existe una partición de* I=$\{1, ..., n\}$ *en q subconjuntos disjuntos* I_j*, j=1, ..., q, con* $q \leq k$*, tal que cada patrón de empaquetamiento* $A(I_j)$ *forman un patrón de estructura de* $p_j - bloque$ *con* $p_j \leq k$*.*

Nota 2.1 *Un patrón* A=$\{(x_i, y_i, o_i)$*:i=1,...,n*$\}$ *tiene estructura de k-bloque con* $k \leq 3$ *si y sólo si el patrón es de estructura guillotina (estructura G).*

Definición 2.4 *Un patrón* A=$\{(x_i, y_i, o_i)$*:i=1,...,n*$\}$ *de un PLP tiene estructura G4 si A tiene estructura de* $k - bloque$ *con* $k \leq 4$*.*



(a) Instancia (20,14,4,3) (b) Instancia (19,17,5,3) (c) Instancia (56,52,12,5)

Figura 2.7: *Ejemplos de diseños G4.*

Con estas definiciones tenemos que cualquier patrón de k-bloque con $k \leq 4$ tiene estructura-$G4$, pero además se puede demostrar que cualquier patrón de bloques también tiene estructura-$G4$. Por tanto, cualquier solución de los heurísticos de bloques anteriores posee estructura-$G4$, por lo que si se calcula el mejor patrón de estructura-$G4$ para una instancia del *PLP*, conseguiremos un patrón al menos tan bueno como el mejor patrón obtenido por cualquiera de los otros heurísticos de bloques.

Por la definición de estructura-$G4$, un patrón-$G4$ siempre se puede descomponer en dos estructuras-$G4$, si ocurre un corte guillotina (Figura 2.7(a), 2.7(b)), o en cuatro cuando no existe un corte guillotina (Figura 2.7(c)). Para calcular el patrón-$G4$ maximal necesitamos patrones-$G4$ máximos para rectángulos más pequeños. Sea *n(L',W')* el número de cajas (con longitud l y anchura w) que caben dentro de L' y W' con un patrón-$G4$ ($0 \leq L' \leq L$, $0 \leq W' \leq W$). Usando las notaciones de la Figura 2.8 con

$$e := L' - a, \quad f := W' - d, \quad g := L' - c, \quad h := W' - b$$

la siguiente fórmula de recurrencia es válida:

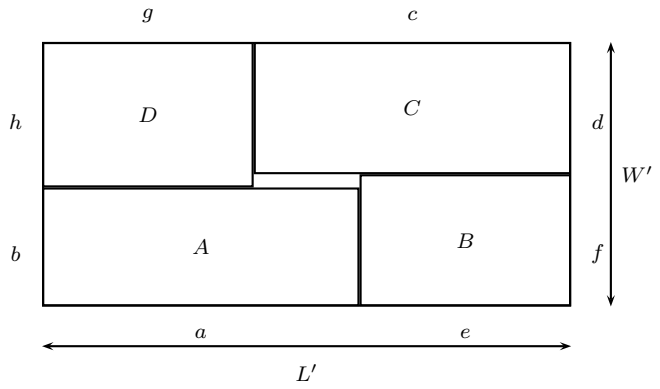


Figura 2.8: *Notación para las fórmulas de recurrencia.*

$$n(L',W') = \max\{\max_{1\le l\le L'/2} n_v(L',W',l), \max_{1\le w\le W'/2} n_H(L',W',w),$$
$$\max_{1\le l<L'}\ \max_{1\le w<W'}\ \max_{L'-l<c<L'}\ \max_{1\le d<W'-b}\{n(a,b)+n(e,f)+n(c,d)+n(g,h)\}\} \quad (2.37)$$

$$1 \le W' \le W, \quad 1 \le L' \le L$$
$$n_V(L',W',a) := n(a,W') + n(L'-a,W'),$$
$$n_H(L',W',b) := n(L',b) + n(L',W'-b),$$
$$n(0,b) := 0, \quad 0 \le b \le W, \quad n(a,0) := 0, \quad 1 \le a \le L,$$
$$n(l,w) := n(w,l) := 1.$$

La versión más simple consiste en calcular $n(L',W')$ para todo L' y W' con $0 \le W' \le W$ y $0 \le L' \le L$. Pero la fórmula de recurrencia y el algoritmo básico incluyen algunas mejoras para reducir el esfuerzo computacional. En primer lugar utiliza ideas de normalización y puntos interiores descritos por Herz (1972)[49]. Además el algoritmo explora todos los posibles diseños usando cuatro cotas distintas (que se describen en la sección siguiente): cota del cociente del área del

pallet/área de la caja $u_C(L,W)$, cota propuesta por Barnes $u_B(L,W)$, la mínima cota de la clase de equivalencia $u_A(L,W)$, y la cota de Isermann basada en la resolución de un problema de programación lineal $u_L(L,W)$. El algoritmo es pseudo-polinomial por ser un algoritmo de programación dinámica.

El esquema principal del algoritmo para calcular $n(L',W')$ para un pallet (L',W',l,w) con $L' > w$ y $W' > w$, es el siguiente:

Inicialización:

$$n(w,W') := \lfloor W'/l \rfloor, W' \in S(W),\ n(L',w) := \lfloor L'/l \rfloor, L' \in S(L).$$

i) Simetría: Si $W' < L'$ y $L' \leq W$ entonces $n(L',W') := n(W',L')$. *Salir*, ya que el cálculo de $n(L',W')$ ya está hecho.

ii) Calcular el mejor patrón homogéneo (con todas las cajas con la misma orientación). Sea n_1 el número de cajas de este patrón y la cota del área $u := u_C(L',W')$.
Si n_1=u, entonces *Salir*.

iii) Comparación con la mejores soluciones para pallets más pequeños:
Sea $n_2 := max\{n(L'-1,W'), n(L',W'-1), n_1\}$.
Si $n_2 = u$, entonces *Salir*.

iv) Mejora de la cota superior con la cota de Barnes:
Sea $u := min\{u, u_B(L',W')\}$.
Si $n_2 = u$, entonces *Salir*.

v) Estructura guillotina: Calcular
$n_3 := \text{máx}\left\{\{n_2, \underset{\{a \in S_0(L'/2)\}}{\text{máx}} n_V(L',W',a), \underset{\{b \in S_0(W'/2)\}}{\text{máx}} n_H(L',W',b)\}\right\}$.
Si $n_3 = u$, entonces *Salir*.

vi) Mejora de la cota superior con otras cotas:
Sea $u := min\{u, u_A(L',W')\}$. Si $n_3 = u$, entonces *Salir*.
Sea $u := min\{u, u_L(L',W')\}$. Si $n_3 = u$, entonces *Salir*.

vii) Estructura de 4 bloques:

Calcular n_4, el número máximo de cajas en patrones con estructura de 4-bloques mediante las fórmulas anteriores (2.37). Como calcular n_4 requiere la variación de cuatro parámetros, se incorporan algunos métodos para reducir el esfuerzo computacional.

Este heurístico proporciona todas las soluciones óptimas para los problemas test de *Tipo I* y no existe ningún problema de *Tipo II* en el que la diferencia entre la solución exacta y la solución del *G4* sea mayor que uno. Además, como se comentará en la sección 4.4, pág 133, nosotros hemos resuelto todos los problemas del *Tipo II* con este algoritmo y sólo en 45 casos no obtiene la solución óptima. Este algoritmo no consigue el óptimo para los problemas que tienen como solución óptima un diseño de *orden superior* (Figura 1.2(c)).

2.4.6. Métodos Metaheurísticos

En los últimos años también se ha intentado trabajar con algoritmos metaheurísticos para el problema, pero sin obtener por el momento buenos resultados. Dowsland (1993)[31] propone un *simulated annealing* para el problema del pallet y lo generaliza al problema con cajas diferentes. Dowsland (1996)[32] también desarrolla un *tabu thresholding*. Amaral y Wright (2001)[3] proponen otro método basado en oscilación estratégica. Herbert y Dowsland (1996)[48] presentan *algoritmos genéticos*. Sin embargo, los resultados obtenidos por los métodos metaheurísticos no son comparables con los heurísticos comentados anteriormente. En estos trabajos sólo se resuelven de manera óptima problemas con 50 cajas como máximo.

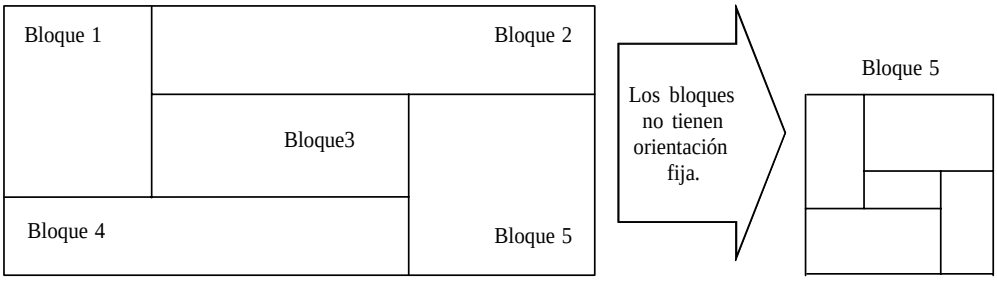


Figura 2.9: *Heurístico recursivo de Cinco bloques.*

2.4.7. Heurísticos más recientes

Un algoritmo recursivo similar al *G4* fue desarrollado por Arenales y Morabito (1998)[62]-[63] en 1998. El heurístico que proponen es hacer recursivo el algoritmo de 5-bloques de Bischoff y Dowsland (1992)[16]. Consiste en buscar una estructura de 5 rectángulos y en cada uno de estos rectángulos se vuelve a buscar el mejor diseño con 5 rectángulos (Figura 2.9). Para reducir el esfuerzo computacional utilizan como en el *algoritmo G4* los puntos interiores. Sin embargo, este procedimiento no consigue ninguna mejora respecto a la calidad de las soluciones obtenidas por el *G4*, ya que los patrones considerados como de orden superior tampoco pueden ser obtenidos por este algoritmo.

Una última publicación aparecida recientemente corresponde a Lins et al. (2003)[60]. Su algoritmo está basado en el procedimiento recursivo anterior, pero sofisticándolo un poco más, introduciendo la idea de *L-estructura* que es parecida a la idea de corte de guillotina pero con un corte definido en L. Un rectángulo se puede cortar de dos maneras para que las estructuras resultantes tengan *L-estructura* (Figura 2.10) y una vez que tenemos una *L-estructura* existen cinco posibles formas de cortar esa estructura para que vuelvan a quedar *L-estructuras*. El objetivo es que siempre tengamos estructuras en L (Figura 2.11).

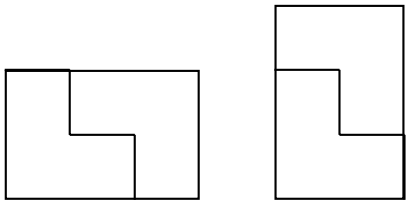


Figura 2.10: *L-estructura para rectángulos.*

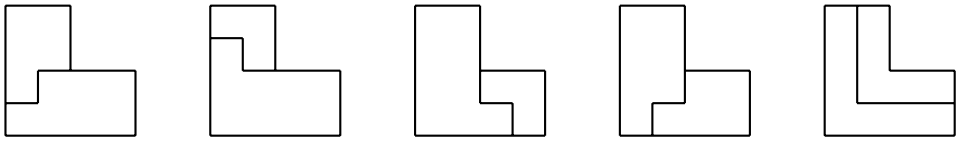
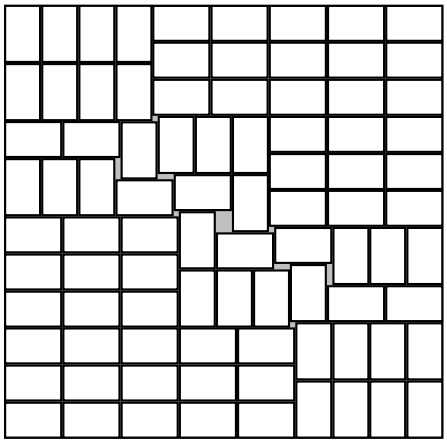
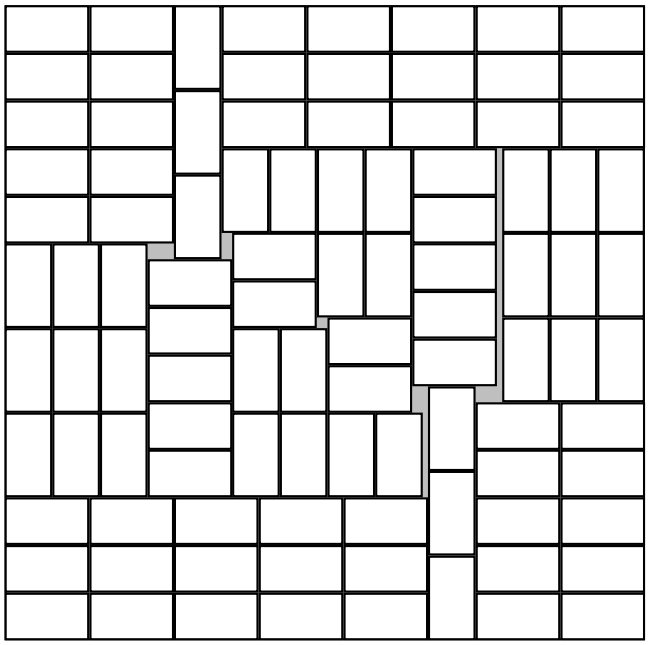


Figura 2.11: *Cortes posibles en L para l-estructuras.*

Los resultado computacionales de los dos trabajos anteriores no utilizan el conjunto completo de *Tipo II* sino un subconjunto de aproximadamente unas 20000 instancias. De estos problemas, con el primer algoritmo resuelven todos menos 18, que son los mismos resultados del *algoritmo G4*. Para el segundo algoritmo, como las soluciones de 5-rectángulos generadas por el anterior heurístico tienen *L-estructura* sólo necesitan resolver los 18 problemas para los que no encontraban solución óptima. Todos ellos son ahora resueltos óptimamente, pero los tiempos de computación son muy elevados, para algunas instancias superiores a 6 horas en un Pentium III a 700 MHz. Los autores conjeturan que su algoritmo podría resolver óptimamente todos los problemas del pallet al no encontrar ningún contraejemplo que no tenga *L-estructura*. Sin embargo, esta conjetura no puede ser comprobada ni siquiera para todos los problemas de *Tipo II*. Nosotros hemos encontrado soluciones óptimas que no poseen *L-estructura*, (Figura 2.12), pero no hemos podido demostrar que no exista alguna solución óptima que no se pueda obtener mediante *L-estructuras*.



(a) (83,82,11,7)



(b) (121,120,16,9)

Figura 2.12: *Soluciones que no se obtienen mediante un corte en L.*

2.4.8. Un heurístico para pallets con un gran número de cajas

En 2001 Young-Gun y Kang [92] proponen un algoritmo para pallets con un enorme número de cajas (hasta 6800 cajas), derivado de una aplicación práctica de un problema de carga de contenedores. En este caso no importa tanto que la solución sea óptima como proporcionar una buena solución en un tiempo muy reducido. Para ello construyen un algoritmo basado en el trabajo de Steudel (1979)[80], pero aplicado de forma recursiva. El algoritmo de Steudel podía originar un hueco en el centro, y la idea es que si este hueco no se rellena de manera óptima con un bloque, se elige otro rectángulo interior donde volver a aplicar el método. Este algoritmo se prueba en una batería de 30 problemas, con instancias de tamaño entre 1000 y 6800 cajas. Sin embargo, al no hacer referencia a la utilización de otros algoritmos para estos problemas, es difícil valorar la bondad del algoritmo en relación a otros heurísticos, por ejemplo heurísticos de bloques.

2.5. Cotas Superiores

Las cotas superiores son usadas para verificar la optimalidad de las soluciones heurísticas y para mejorar la eficiencia de los algoritmos exactos. A lo largo de la literatura se han desarrollado un amplio número de cotas para el problema. Una revisión de algunas de estas cotas se puede encontrar en el trabajo de Letchford y Amaral (2001)[57]. A continuación citamos las más importantes:

2.5.1. Cota del Área

La primera y más simple cota utilizada es la basada en el área:

$$UB = \lfloor (L * W) \,/\, (l * w) \rfloor \tag{2.38}$$

Únicamente entre el 10 y el 15 por ciento de los casos coinciden la cota del área y la solución óptima, como mostraron Smith y De Cani (1980)[79] y Dowsland (1985)[27] .

2.5.2. Cota de Barnes

Una mejor estimación se obtiene mediante la cota propuesta por Barnes (1979)[8]. El procedimiento está basado en la consideración de que un diseño con cajas $(l \times w)$ en un pallet $L \times W$ se puede descomponer en un diseño con cajas $(l \times 1)$ ó $(w \times 1)$. Por tanto, el desperdicio máximo de las soluciones óptimas de los problemas $(L, W, l, 1)$ y $(L, W, w, 1)$ será una cota inferior del desperdicio mínimo del problema original. Para estos problemas la solución óptima es conocida, como demostraron Brualdi y Foregger (1974)[18], mediante un procedimiento que conseguía los diseños óptimos para una pallet con dimensiones $(L, W, l = k * w, w)$ con $k \geq 1, k \in \mathcal{N}$.

Sea A el desperdicio en un diseño óptimo de cajas $(l \times 1)$ y B el desperdicio en un diseño óptimo de cajas $(w \times 1)$. El valor de A está dado por:

$$A = \text{mín}\left\{r * s, (l - r) * (l - s)\right\} \tag{2.39}$$

donde $r = (L \mod l)$ y $s = (W \mod w)$. B se obtiene mediante un cálculo similar. La cantidad de desperdicio $\mathcal{D}$ en cualquier diseño de cajas $(l \times w)$ satisface:

$$\mathcal{D} \geq \text{máx}\left\{A, B\right\} \tag{2.40}$$

$$\begin{aligned} &\mathcal{D} \equiv A \pmod{\text{l}} \\ &\mathcal{D} \equiv B \pmod{\text{w}} \end{aligned} \tag{2.41}$$

Dado que el desperdicio en un diseño de $(l \times w)$ cajas debe ser un elemento de la clase residual de $l*w$, obtenemos que:

$$\mathcal{D} \equiv L * W \pmod{\text{l} * \text{w}} \tag{2.42}$$

que es mejor que las anteriores. Por tanto, $\mathcal{D}$ se obtiene como el menor elemento de la clase residual de $l * w$ que satisface la condición (2.40). La cota de Barnes quedaría:

$$UB = \lfloor (L * W - \mathcal{D}) \,/\, (l * w) \rfloor \tag{2.43}$$

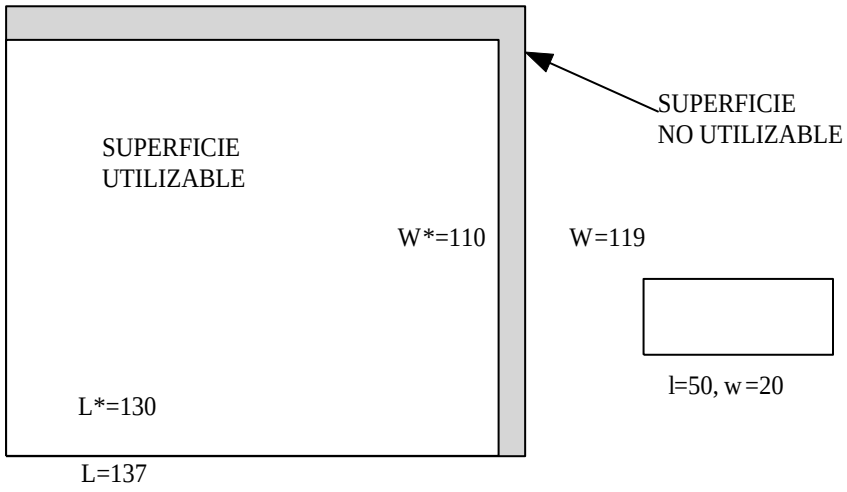


Figura 2.13: *Superficie utilizable en la instancia (137,119,50,20).*

2.5.3. Superficie utilizable del Pallet

Dowsland (1985)[27] planteó una cota basada en la observación de que en la mayoría de los casos el rectángulo usado para los diseños es menor que el pallet entero (Figura 2.13). Como hemos mencionado, cada diseño se puede normalizar de modo que cada esquina inferior izquierda de la caja es una combinación entera de las dimensiones de la caja. De este modo las dimensiones utilizables del pallet son:

$$L^* = \max_{(r,s)\in E(L,l,w)} \{r * l + s * w\} \tag{2.44}$$

y

$$W^* = \max_{(p,q)\in E(W,l,w)} \{p * l + q * w\} \tag{2.45}$$

Por tanto la cota sería:

$$UB = \left\lfloor \frac{L^* * W^*}{l * w} \right\rfloor \tag{2.46}$$

2.5.4. Cota de Keber

En 1985, Keber[54] observó que la cota de un problema puede mejorarse en algunos casos teniendo en cuenta la siguiente consideración: si $u_{c,d}$ es una cota superior para el problema *(L,W,c,d)*, entonces $u_{c,d}$ es válida para problemas *(L,W,l,w)* con $l \geq c$ y $w \geq d$ (para problemas con el mismo pallet y cajas más grandes). Es obvio que si en (L, W) sólo podemos colocar $u_{c,d}$ cajas $(c \times d)$ y ahora las cajas son de un tamaño mayor $(l \times w)$, entonces $u_{c,d} \geq u_{l,w}$. Sin embargo, un descenso en las dimensiones de la caja puede permitir un descenso mayor que el esperado en las dimensiones utilizables del pallet L^* y W^*.

Ejemplo 2.2 *Supongamos la instancia (1060,813,162,136) y la instancia con el mismo pallet (1060,813,136,136) pero con cajas más pequeñas. Para la primera instancia el pallet utilizable es (1056,810,162,136) y su cota es 38 mientras que para la segunda instancia el pallet utilizable es (952,816,136,136) y su cota es 35. Por tanto, la cota superior de 35 es válida para la primera instancia.*

Dada una instancia *(L,W,l,w)*, Keber propone examinar las cotas del pallet utilizable para algunas instancias (L, W, i, j) con cajas más pequeñas $(0{,}9 * l) \leq i \leq l$ y $(0{,}9 * w) \leq j \leq w$.

2.5.5. La menor "cota del área" de la clase de equivalencia

Debido a que todos los elementos de la clase de equivalencia tienen la misma cota, podemos encontrar el elemento con menor cota. Dowsland (1984)[26] utilizó la cota superior basada en el área y calculó su mínimo en la clase de equivalencia por medio de un nuevo procedimiento (cuyos detalles describió muy vagamente). La formulación que planteó para resolver esta cota fue la siguiente:

$$\text{Minimizar } \left\lfloor \frac{L*W}{l*w} \right\rfloor \tag{2.47}$$

$$\text{sujeto a:} \tag{2.48}$$

$$n*l+m*w \leq L \quad \forall (n,m) \in E(L,l,w) \tag{2.49}$$

$$n*l+(m+1)*w > L \quad \forall (n,m) \in E(L,l,w) \tag{2.50}$$

$$(\left\lfloor \frac{L}{l} \right\rfloor + 1)*w > L \tag{2.51}$$

$$p*l+q*w \leq W \quad \forall (p,q) \in E(W,l,w) \tag{2.52}$$

$$p*l+(q+1)*w > V \quad \forall (p,q) \in E(W,l,w) \tag{2.53}$$

$$(\left\lfloor \frac{W}{w} \right\rfloor + 1)*w > W \tag{2.54}$$

Los métodos existentes para resolver este sistema requieren que ninguna de las desigualdades sea estricta, y la forma estándar de hacer que no sean estrictas es añadiendo una pequeña constante a la desigualdad ϵ. Sin embargo en este caso las desigualdades $>$ pueden reemplazarse por $\geq$, ya que la función objetivo se puede expresar como la parte entera de una función continua. De este modo, si el mínimo ocurre en un extremo abierto, podemos encontrar un punto junto al extremo para el cual la parte entera de la función no cambie.

La rutina que empleó K. Dowsland para resolver el sistema consiste en elegir pares de restricciones en L y W por turno y reducir su holgura a 0. Estos valores son entonces sustituidos en las otras desigualdades y en la función objetivo. Reajustando las desigualdades, se puede hacer la sustitución $x = l/w$ dejando el problema en una variable. La función objetivo adquiere la forma de $A+B*x+C/x$ y las restricciones definen intervalos en la recta real que deben contener a x. La forma de la función objetivo implica que el mínimo para este par será el punto dentro del intervalo más cercano al óptimo global de la función, el cual ocurre en $\sqrt{(C/B)}$. El mínimo de todos los pares será el mínimo global.

Este método también puede adaptarse para encontrar el pallet de la clase de equivalencia de mínimas dimensiones.

Nelißen (1993)[66] propone otro método para resolver el sistema propuesto por Dowsland. En primer lugar, propone escalar el problema permitiendo pasar de *(L,W,l,w)* al problema equivalente $(L/w, W/w, l/w, 1)$ donde los valores de la instancia pueden no ser racionales. Entonces $(V, W, x, 1)$ será un miembro de la misma clase de equivalencia si y sólo si se satisfacen las siguientes condiciones:

$$n * x + m \leq V \quad \forall (n, m) \in E(L, l, w) \tag{2.55}$$

$$n * x + m + 1 > V \quad \forall (n, m) \in E(L, l, w) \tag{2.56}$$

$$\left(\max_{(n,m) \in E(L,l,w)} n + 1 \right) * x > V \tag{2.57}$$

junto con un conjunto similar para W y $E(W, l, w)$. La ecuación (2.55) podría ser reescrita como:

$$\max_{(n,m) \in E(L,l,w)} n * x + m \leq V \tag{2.58}$$

que junto con la condición (2.56) permite:

$$V^* = \max_{(n,m) \in E(L,l,w)} n * x + m \tag{2.59}$$

Para cada x y cada conjunto $E(L, l, w)$, $E(W, l, w)$ se puede definir un PLP $(\max_{(n,m) \in E(L,l,w)} n*x+m, \max_{(p,q) \in E(W,l,w)} p*x+q, x, 1)$, si y sólo si x está dentro de un intervalo abierto "suficientemente" pequeño conteniendo a l/w. Este PLP estará en la misma clase de equivalencia que *(L,W,l,w)*.

El intervalo posible donde se encuentra x se determina mediante las condiciones (2.55)-(2.57) y condiciones análogas para W. Sean r=$|E(L, l, w)|$, s=$|E(W, l, w)|$, $R = \{1, 2, ..., r\}$ y $S = \{1, 2, ..., s\}$, en otras palabras, número de particiones eficientes para L y W. Entonces el PLP correspondiente al cociente de longitud/anchura= x está en la misma clase de equivalencia de *(L,W,l,w)* si y sólo si:

$$n_i * x + m_i + 1 > n_j * x + m_j \quad \forall i \neq j \in R \tag{2.60}$$

$$\left(\max_{i \in R} n_i + 1\right) * x > n_j * x + m_j \quad \forall j \in R \tag{2.61}$$

$$p_i * x + q_i + 1 > p_j * x + q_j \quad \forall i \neq j \in S \tag{2.62}$$

$$\left(\max_{i \in S} p_i + 1\right) * x > p_j * x + q_j \quad \forall j \in S \tag{2.63}$$

Mediante transformaciones se llega a las siguientes condiciones necesarias y suficientes.

$$x > \frac{m_j - m_i - 1}{n_i - n_j} \quad \forall i \neq j \in R : n_i > n_j \tag{2.64}$$

$$x < \frac{m_j - m_i + 1}{n_i - n_j} \quad \forall i \neq j \in R : n_i > n_j \tag{2.65}$$

$$x > \frac{m_j}{\max_{i \in R} n_i + 1 - n_j} \quad \forall j \in R \tag{2.66}$$

$$x > \frac{q_j - q_i - 1}{p_i - p_j} \quad \forall i \neq j \in S : p_i > p_j \tag{2.67}$$

$$x < \frac{q_j - q_i + 1}{p_i - p_j} \quad \forall i \neq j \in S : p_i > p_j \tag{2.68}$$

$$x > \frac{q_j}{\max_{i \in R} p_i + 1 - p_j} \quad \forall j \in S \tag{2.69}$$

Como todas las desigualdades son estrictas y las fracciones están compuestas por valores enteros, el intervalo posible es abierto y acotado por números racionales. Las desigualdades (2.64) y (2.67) impiden que una partición eficiente (n_i, m_i) (p_i, q_i) pueda ser reemplazada por $(n_i, m_i + 1)$ $(p_i, q_i + 1)$ en el lado izquierdo del intervalo posible, mientras que las desigualdades (2.65) y (2.68) hacen lo mismo en el lado derecho. Las desigualdades (2.66) y (2.69) indican que $\left(\lfloor \frac{L}{a} \rfloor + 1{,}0\right)$ ó $\left(\lfloor \frac{W}{a} \rfloor + 1{,}0\right)$ respectivamente, no pueden ser una partición eficiente del *PLP* derivado de x. Por tanto, la longitud del problema correspondiente para una valor x del intervalo posible será $\max_{(n,m) \in E(L,l,w)} n * x + m$ y la anchura $\max_{(p,q) \in E(W,l,w)} p * x + q$. En otras palabras las dimensiones están determinadas por un conjunto de funciones lineales. La tarea es determinar para qué subin-

tervalo la función $n * x + m$ produce el máximo. Sea (lb, ub) con $lb, ub \in \mathbb{Q}$, el intervalo posible para x.

Entonces se calcula el par (n_1, m_1) para el que $\text{máx}_{(n,m)\in E(L,l,w)}\, n * x_1 + m$ con $x_1 = l$. A partir de (n_1, m_1), se calcula otro par (n_2, m_2) que cumpla $n_2 > n_1$ y la intersección de $n_1 * x + m_1$ y $n_2 * x + m_2$ debe estar dentro del intervalo, esto es $x_2 = \frac{m_2 - m_1}{n_1 - n_2} \in (l, u)$. Se deben explorar todos los pares $(n, m) \in E(L, l, w)$ con $n > n_1$ y elegir el mínimo valor obtenido para x_2. Este proceso se repite iterativamente hasta que $x_i = u$. De forma análoga se calculan los subintervalos para el ancho.

Si se unen los resultados, se consiguen una partición de (l, u) en subintervalos donde las particiones de longitud y anchura que dominan son constantes.

Para los subintervalos $(r_1, s_1], (r_2, s_2], ..., (r_k, s_k)$ y particiones dominantes para la longitud y anchura (n_i, m_i) y (p_i, q_i) respectivamente, el área del PLP viene definida por:

$$f(x) = \frac{(n_i * x + m_i) * (p_i * x + q_i)}{x} = n_i * p_i * x + n_i * q_i + m_i * p_i + \frac{m_i * q_i}{x} \quad (2.70)$$

Dowsland, como hemos mencionado anteriormente, presentaba una fórmula general con la misma forma que ésta: $A + B * x + C/x$. Se tiene una función racional en x cuyo mínimo se alcanza en:

$$f'(x) = n_i * p_i - \frac{m_i * q_i}{x^2} \overset{n_i, p_i \neq 0}{\Rightarrow} x_0 = \sqrt{\frac{m_i * q_i}{n_i * p_i}} \quad (2.71)$$

Si x_0 está dentro del intervalo $(r_i, s_i]$ ó (r_i, s_i) (esto implica $m_i * q_i > 0$, dado que $r_1 = 1$), entonces $f(x_0)$ es el mínimo de f en $(r_i, s_i]$ ó (r_i, s_i), dado que la segunda derivada es

$$f''(x) = \frac{2 * m_i * q_i}{x^3} \quad (2.72)$$

y $m_i * q_i > 0$ y $x > 0$. Se puede demostrar que f es convexa en cada subintervalo y a su vez en todo el intervalo (l, u). Esto permite usar el siguiente procedimiento para obtener el mínimo:

1. Sea i =1.

2. Calcular el subintervalo $(r_i, s_i]$ (ó (r_i, s_i), si $s_i = u$) y la partición dominante (n_i, m_i) y (p_i, q_i) respectivamente.

3. Si $x_0 = \sqrt{\frac{m_i * q_i}{n_i * p_i}}$ está dentro del intervalo $(r_i, s_i]$ ((r_i, s_i)) o si $f(r_i) < f(s_i)$ se sigue cumpliendo, entonces *parar* (debido a la convexidad de la función el óptimo ha sido encontrado).

4. En otro caso si $s_i = u$, *parar*. (Si el mínimo se alcanza en la cota superior). Hacer $i = i + 1$, ir a 2.

Se sabe que f es continua en cada subintervalo. Como el valor de la función es un número racional, y sólo interesa calcular la parte entera de f, siempre se puede buscar un ε tal que $\lfloor f(l + \varepsilon) \rfloor = \lfloor f(l) \rfloor$ ó $\lfloor f(u - \varepsilon) \rfloor = \lfloor f(u) \rfloor$. Si el mínimo es obtenido por un número racional dentro del intervalo (l, u), sea r/s, entonces el problema ($\max_{(n,m) \in E(L,l,w)} n * r + m * s, \max_{(p,q) \in E(W,l,w)} p * r + q * s, r, s$) es el de cota mínima de la clase de equivalencia.

Si el mínimo es obtenido en l ó u ó un número irracional dentro del intervalo posible (un x_0 de algún subintervalo), entonces este cociente puede ser aproximado sólo con cierta precisión. Es decir, se ha de buscar dentro del subintervalo un valor racional que pertenezca a la misma clase de equivalencia que el original y que esté lo más cercano posible al cociente mínimo.

2.5.6. Cotas basadas en programación lineal

Isermann (1987)[52] desarrolló un método de cota superior resolviendo un problema de programación lineal. Este procedimiento es otro ejemplo de la utilidad de la noción de particiones. Supongamos un empaquetamiento normalizado cubierto por un rejilla de longitud uno. Entonces cada fila o columna de la rejilla contiene una partición.

La Figura 2.14 muestra una solución de la instancia (8,5,3,2). De arriba a abajo, las filas contienen las particiones posibles (1,1), (2,1), (1,2), (2,1) y (2,1) del lado L (8), y de izquierda a derecha las columnas contienen las particiones (1,1), (1,1), (0,2), (1,1), (1,1), (0,2), (0,2) y (0,2) del lado W (5). Se asigna a cada partición posible (i, j) de L una variable entera x_{ij} y para cada partición posible (f, g) de W una variable entera y_{fg}. Entonces cada empaquetamiento posible de (L, W, l, w) determina valores para aquellas variables $x_{i,j}$ y $y_{f,g}$ por el número de filas/columnas que contienen las particiones (i, j) y (f, g).

El empaquetamiento de la Figura 2.14 corresponde a los valores de:
$x_{2,1} = 3$, $x_{1,2} = 1$, $x_{1,1} = 1$ y $x_{i,j} = 0$ $\forall (i, j) \in F(L, l, w)$,
$y_{1,1} = 4$, $y_{0,2} = 4$ y $y_{f,g} = 0$ $\forall (f, g) \in F(W, l, w)$.

Obviamente $F(L, l, w) = F(L^*, l, w)$ y $F(W, l, w) = F(W^*, l, w)$, con L^* y W^* como en las ecuaciones (2.44) y (2.45). Considerar ahora el siguiente conjunto de restricciones lineales para las variables $x_{i,j}$ y $y_{f,g}$:

$$\sum_{(i,j)\in F(L,l,w)} x_{i,j} \leq W^* \tag{2.73}$$

$$\sum_{(f,g)\in F(W,l,w)} y_{f,g} \leq L^* \tag{2.74}$$

$$\sum_{(i,j)\in F(L,l,w)} l * i * x_{i,j} - \sum_{(f,g)\in F(W,l,w)} w * g * y_{f,g} = 0 \tag{2.75}$$

$$\sum_{(i,j)\in F(L,l,w)} w * j * x_{i,j} - \sum_{(f,g)\in F(W,l,w)} l * f * y_{f,g} = 0 \tag{2.76}$$

$$x_{i,j} \geq 0 \ \forall (i, j) \in F(L, l, w) \tag{2.77}$$

$$y_{f,g} \geq 0 \ \forall (f, g) \in F(W, l, w) \tag{2.78}$$

La restricción (2.73) garantiza que el número de filas que son ocupadas por las particiones de W^* no sea mayor que la anchura posible del rectángulo, y la restricción (2.74) hace lo mismo para el número de columnas y la longitud posible del rectángulo L^*. Las ecuaciones (2.75) y (2.76) pueden ser consideradas como tiras de ancho unidad y longitud l ó w respectivamente. La primera suma de la

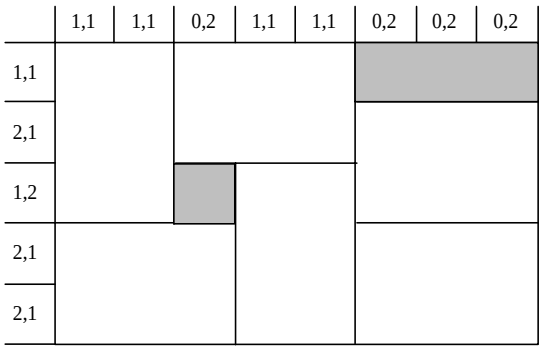


Figura 2.14: *Particiones en un diseño de la instancia (8,5,3,2).*

ecuación (2.75) cuenta el número de tiras unidad horizontales de anchura l en un empaquetamiento y suma el número de celdas unidad ocupadas por aquellas tiras. La segunda suma cuenta el número de tiras unidad verticales de anchura w y suma las celdas unidad ocupada por ellas. En cada posible empaquetamiento, la diferencia de aquellas sumas ha de ser cero. La ecuación (2.76) hace lo mismo que la anterior pero para las tiras unidad horizontales de anchura w y las ocupadas por tiras unidad verticales de anchura l.
Definimos:

$$H = \sum_{(i,j)\in F(L,l,w)} \frac{i * x_{i,j}}{w} \tag{2.79}$$

$$V = \sum_{(f,g)\in F(W,l,w)} \frac{f * y_{f,g}}{w} \tag{2.80}$$

Definimos una V-caja como una caja con orientación vertical y una H-caja como una caja con orientación horizontal. Cada V-caja se descompone en w tiras verticales de longitud l ó l tiras horizontales de anchura w. Por otro lado, cada H-caja se descompone en l tiras verticales de anchura w ó w tiras horizontales de anchura l. Entonces la parte entera de H es el número de H-cajas y la parte entera de V es el número de V-cajas, y el problema de programación lineal entero

dado por las seis restricciones (2.73)-(2.78) y la función objetivo:

$$\text{máx}\, Z = \lfloor H + V \rfloor = \left\lfloor \sum_{(i,j)\in F(L,l,w)} \frac{i * x_{i,j}}{w} + \sum_{(f,g)\in F(W,l,w)} \frac{f * y_{f,g}}{w} \right\rfloor \tag{2.81}$$

proporciona una cota superior válida para el PLP (L, W, l, w). Dado que el problema lineal entero es NP-Completo, resolvemos la relajación lineal del problema.

Naujoks (1991)[65] propone una mejora de este método: sea U_1 la cota superior obtenida por el LP dado anteriormente. Introduce una séptima restricción:

$$H + V = \sum_{(i,j)\in F(L,l,w)} \frac{i * x_{i,j}}{w} + \sum_{(f,g)\in F(W,l,w)} \frac{f * y_{f,g}}{w} \geq U_1 \tag{2.82}$$

que asegura que la suma de H-cajas y V-cajas no baje de U_1. Resuelve este LP con las siguientes funciones objetivo:

$$\text{máx}\, H = \sum_{(i,j)\in F(L,l,w)} \frac{i * x_{i,j}}{w} \quad \text{máx}\, V = \sum_{(f,g)\in F(W,l,w)} \frac{f * y_{f,g}}{w}$$

esto es, maximizando independientemente el número de H-cajas y V-cajas.
Sea:

$$U_2 = \text{mín}\left\{ \left\lfloor \hat{H} \right\rfloor + \left\lfloor \hat{V} \right\rfloor, U_1 \right\} \tag{2.84}$$

donde $\hat{H}$ y $\hat{V}$ respectivamente son los valores de las soluciones óptimas de los problemas lineales. Entonces U_2 es una nueva cota superior, que en algunos casos es inferior a U_1.

2.5.7. La cota de Nelißen

Nelißen (1995)[67] plantea un nuevo procedimiento para calcular una cota para el problema a partir de la formulación anterior. Partiendo del procedimiento para calcular $\hat{H}$ y $\hat{V}$, cotas para el número de H-cajas y V-cajas, Nelißen utiliza el mismo procedimiento para $x_{i,j}$ y $y_{f,g}$, es decir, resuelve el LP formado por las siete restricciones anteriores pero con la función objetivo

mín $z^L_{i,j} = x_{i,j}$ y max $z^L_{i,j} = x_{i,j}$ para todas las particiones posibles de L, $x_{i,j}$, y obtiene cotas inferiores y superiores $lx_{i,j}$ y $ux_{i,j}$ para el número de filas en una solución óptima (una solución con U_2 rectángulos). Se procede de forma análoga para las particiones de W.

A partir de aquí intenta mejorar la cota mediante el uso de implicaciones lógicas. El diseño del procedimiento es por contradicción: asume que existe una solución de valor igual a la cota superior $U = U_2$ y mediante el ajuste de diferentes argumentos se intenta llegar a una contradicción lo que produciría una rebaja de la cota. En primer lugar, intenta ajustar más los valores de H-cajas y V-cajas. Mediante un procedimiento de etiquetado obtiene:

$$H_{\text{mín}} = \text{máx}\left\{ \underset{2\le r\le \lfloor W/w\rfloor}{\text{máx}} r * \left\lceil \sum_{\substack{(c,d)\in E(W,l,w)\\ d\ge r}} ly_{c,d}/l \right\rceil , \underset{2\le r\le \lfloor L/l\rfloor}{\text{máx}} r * \left\lceil \sum_{\substack{(c,d)\in E(L,l,w)\\ c\ge r}} lx_{c,d}/w \right\rceil \right\} \quad (2.85)$$

y

$$V_{\text{mín}} = \text{máx}\left\{ \underset{2\le r\le \lfloor W/l\rfloor}{\text{máx}} r * \left\lceil \sum_{\substack{(c,d)\in E(W,l,w)\\ c\ge r}} ly_{c,d}/w \right\rceil , \underset{2\le r\le \lfloor L/w\rfloor}{\text{máx}} r * \left\lceil \sum_{\substack{(c,d)\in E(L,l,w)\\ d\ge r}} lx_{c,d}/l \right\rceil \right\} \quad (2.86)$$

Como resultado de este primer argumento se ajustan los valores de $\hat{H}$ y $\hat{V}$ a

$$\hat{H} = \text{mín}\left\{\hat{H}, U - V_{\text{mín}}\right\} \quad (2.87)$$

y

$$\hat{V} = \text{mín}\left\{\hat{V}, U - H_{\text{mín}}\right\} \quad (2.88)$$

También se obtiene la eliminación de ciertas particiones y una reducción para $\hat{H}$ y $\hat{V}$ mediante dos proposiciones. Dada una W-partición (f, g) con un cota inferior $ly_{f,g}$, si se cumple $L - ly_{f,g} < l$, entonces cada H-caja intersecta con una columna de tipo (f, g). Las siguientes dos proposiciones se cumplen en cualquier solución óptima:

i) No pueden existir columnas de tipo (p, q), donde $q > 2 * g$.

ii) El número de H-cajas es un múltiplo de g.

Aplicando resultados similares obtiene:

$$L - ly_{f,g} < l \Rightarrow \begin{array}{l}(i)\ q > 2 * g \ \Rightarrow uy_{p,q} = 0 \\ (ii)\hat{H} = \left\lfloor \hat{H}/g \right\rfloor * g,\end{array} \tag{2.89}$$

$$L - ly_{f,g} < w \Rightarrow \begin{array}{l}(i)\ p > 2 * f \ \Rightarrow uy_{p,q} = 0 \\ (ii)\hat{V} = \left\lfloor \hat{V}/f \right\rfloor * f,\end{array} \tag{2.90}$$

$$W - lx_{i,j} < l \Rightarrow \begin{array}{l}(i)\ s > 2 * j \ \Rightarrow ux_{r,s} = 0 \\ (ii)\hat{V} = \left\lfloor \hat{V}/j \right\rfloor * j,\end{array} \tag{2.91}$$

$$W - lx_{i,j} < w \Rightarrow \begin{array}{l}(i)\ \mathrm{r} > 2 * i \ \Rightarrow ux_{r,s} = 0 \\ (ii)\hat{H} = \left\lfloor \hat{H}/i \right\rfloor * i,\end{array} \tag{2.92}$$

y también:

$$\left\lfloor \hat{H}/g \right\rfloor * g < U - \hat{V} \ \Rightarrow uy_{f,g} = \text{mín}\left\{uy_{f,g}, L - l\right\}, \tag{2.93}$$

$$\left\lfloor \hat{V}/f \right\rfloor * f < U - \hat{H} \Rightarrow uy_{f,g} = \text{mín}\left\{uy_{f,g}, L - w\right\}, \tag{2.94}$$

$$\left\lfloor \hat{V}/j \right\rfloor * j < U - \hat{H} \ \Rightarrow ux_{i,j} = \text{mín}\left\{ux_{i,j}, W - l\right\}, \tag{2.95}$$

$$\left\lfloor \hat{H}/i \right\rfloor * i < U - \hat{V} \ \Rightarrow ux_{i,j} = \text{mín}\left\{ux_{i,j}, W - w\right\}. \tag{2.96}$$

por último, el número de tiras *sueltas* sobrantes viene dado por:

$$n_{l\times 1} = \hat{H} * w - \sum_{(i,j)\in E(L,l,w)} i * lx_{i,j}, \tag{2.97}$$

$$n_{w\times 1} = \hat{V} * l - \sum_{(i,j)\in E(L,l,w)} j * lx_{i,j}, \tag{2.98}$$

$$n_{1\times l} = \hat{V} * w - \sum_{(f,g)\in E(W,l,w)} i * ly_{f,g}, \tag{2.99}$$

$$n_{1\times w} = \hat{H} * l - \sum_{(f,g)\in E(W,l,w)} g * lx_{f,g}. \tag{2.100}$$

de las que se obtiene:

$$ux_{i,j} = \text{mín}\left\{ux_{i,j}, lx_{i,j} + \text{mín}\left\{\left\lfloor n_{a\times 1}/i \right\rfloor, \left\lfloor n_{w\times 1}/j \right\rfloor\right\}\right\}, \tag{2.101}$$

$$uy_{f,g} = \text{mín}\left\{uy_{f,g}, ly_{f,g} + \text{mín}\left\{\left\lfloor n_{1\times a}/f \right\rfloor, \left\lfloor n_{1\times b}/g \right\rfloor\right\}\right\}. \tag{2.102}$$

Nelißen propone un último procedimiento si todos estos razonamientos no permiten llegar a una contradicción que disminuya la cota.

Sea $l_1 = l_{f_1,g_1}$, $l_2 = l_{f_2,g_2}$, ..., $l_n = l_{f_n,g_n}$, donde $n = |E(W,l,w)|$, la secuencia en orden de longitud decreciente de las particiones de W, y $l_{n+1} = W - w$ la longitud máxima de una partición no eficiente. Resolvemos el siguiente problema mochila:

$$\begin{aligned} \text{Maximizar } & \sum_{k=1}^{n+1} y_k * l_k = A \\ \text{s.a: } & \sum_{k=1}^{n+1} y_k \leq L \\ & y_k \leq u y_{f_k,g_k} \qquad k = 1, ..., n \\ & y_{n+1} \leq u y_{ne} \\ & y_k \geq 0 \text{ y entero}, \quad 1 \leq k \leq n+1 \end{aligned}$$

La solución óptima de este problema se obtiene por un simple procedimiento greedy. Esta solución es una cota superior para el problema. Si se cumple $A < U * l * w$, es decir, si la cota superior del área que se va a llenar es menor que U, llegamos a una contradicción, por lo que no existe una solución con U cajas. Además, si A cumple una determinadas condiciones se podría mejorar la cota inferior para ly_{f_1,g_1} de la partición dominante (f_1, g_1).

Debido a que algunos valores incluidos en los cálculos, por ejemplo los valores $lx_{i,j}$, pueden haber cambiado durante el procedimiento, puede ser útil repetir iterativamente el procedimiento completo siempre que se produzca algún cambio.

2.6. Algoritmos Exactos para el *PLP*

En este apartado mencionamos los algoritmos exactos que se han desarrollado para el problema del pallet. Estos algoritmos sólo se han probado en instancias de Tipo I. Podemos diferenciar dos tipos de algoritmos exactos: algoritmos de tipo

Branch and Bound y algoritmos para la reducción a un problema de conjunto máximo independiente.

2.6.1. Branch and Bound

En 1979 De Cani[24] fue el primero que propuso un algoritmo exacto para el *PLP*. Este algoritmo y los posteriores de Isermann (1987)[52] y Exeler (1988)[40] son de tipo *Branch and Bound*.

En este tipo de algoritmos se busca un diseño con un determinado número de cajas n. En primer lugar, buscan una solución con $n = n_{ub}$ (número de cajas de la cota superior). Si no la encuentran, se vuelve a aplicar el procedimiento, pero esta vez con $n = n_{ub} - 1$ cajas, hasta llegar a la solución del heurístico. Para cada valor n se construye un árbol de búsqueda donde cada nodo representa un diseño parcial de las cajas en el pallet. Si en un nodo la cantidad de desperdicio del diseño parcial excede la pérdida correspondiente a una solución con n cajas ($L*W - n*l*w$) se produce el corte de esa rama. La manera de ramificar elegida es la búsqueda en profundidad (*depth-first-search*).

El punto crucial de este tipo de algoritmos es en primer lugar que ningún diseño parcial sea considerado más de una vez (esta condición no es satisfecha, por ejemplo, por el algoritmo de De Cani (1979)[24]) y en segundo lugar un cálculo eficiente del desperdicio en cada nodo.

2.6.2. El algoritmo de Dowsland

En una serie de artículos (1985-1987)([28], [30], [29]). Dowsland desarrolló un algoritmo exacto que fue el primero basado en una correspondencia entre el *PLP* y la teoría de grafos, transformando el *PLP* en un problema equivalente de conjunto máximo independiente. Se considera el pallet como un rectángulo dividido en cuadrados unidad y G_{LWlw} es el grafo cuyos vértices representan la esquina inferior-izquierda de todos los posibles emplazamientos de una caja en el pallet.

El número de vértices de G_{LWlw} viene dado por:

$$|V| = (L + 1 - l) * (W + 1 - w) + (L + 1 - w) * (W + 1 - l) \tag{2.103}$$

donde el primer producto corresponde a las posiciones posibles de cajas horizontales y el segundo a las cajas verticales. Dos vértices son adyacentes si al colocar dos cajas en esos vértices se produce un solapamiento. Entonces cualquier conjunto independiente de vértices corresponde a una solución posible. En particular, cualquier conjunto máximo independiente representa una solución óptima para el *PLP* y viceversa. Para resolver este problema existen una amplia variedad de algoritmos. El inconveniente de este método es que el problema de buscar el conjunto máximo independiente se ha demostrado que es *NP-completo*.

El grafo G_{LWlw} para nuestro *PLP* será un grafo muy poco denso (entre 8 % y 12 % de densidad), aunque sí localmente denso, es decir cada vértice tiene muchos adyacentes entre sus cercanos, pero ninguno entre los que estén a una distancia mayor que l.

Es posible reducir el grafo utilizando diferentes procedimientos. En primer lugar, Dowsland resuelve un problema para buscar la instancia equivalente de menor dimensión. Además, puede reducirse el número de vértices al considerar sólo aquellas posiciones de las cajas que son combinaciones enteras de particiones eficientes desde la esquina inferior izquierda del pallet. Esto permitirá reducir el conjunto de vértices de forma que, por ejemplo, en una muestra aleatoria de problemas del *Tipo I* no se sobrepasaron los 600 vértices.

K. Dowsland resuelve el problema del conjunto máximo independiente con el algoritmo de Loukakis y Tsouros (1982)[61]. El algoritmo consiste básicamente en un árbol lexicográfico de búsqueda de los vértices. Loukakis y Tsouros recomiendan ordenar los vértices de forma descendente por el grado de los vértices. Sin embargo, según K. Dowsland funciona mejor el algoritmo si los vértices son ordenados en filas o en columnas, por ejemplo de izquierda a derecha del pallet y desde arriba hacia abajo, donde los empates son resueltos a favor de una H-caja o V-caja.

Durante la búsqueda es posible definir áreas de desperdicio que permanecen en la solución parcial correspondientes al conjunto actual de vértices en el algoritmo. Si la suma de estas áreas excede el desperdicio de la mejor solución conocida, entonces puede saturarse.

2.6.3. El algoritmo de Bhattacharya et al

Un método exacto más reciente es el de Bhattacharya et al. [15] de 1998 que también es de tipo *Branch and Bound*. Los nodos en el árbol de búsqueda representan diseños parciales. El nodo raíz corresponde a un diseño parcial con 0 piezas y a partir de este nodo se ramifica según las posibles particiones eficientes del diseño. Estas particiones están previamente ordenadas por su anchura, en primer lugar las cajas verticales y luego las horizontales. Notar que el pallet se enumera desde el lado (L) colocando piezas a lo largo del ancho del pallet (W).

El algoritmo incorpora dos métodos de acotación para no enumerar o explorar diseños que sean peores que los ya explorados. El primer método tiene en cuenta la máxima pérdida permitida, de forma que si el diseño parcial que estamos construyendo tiene una pérdida mayor que la mejor solución, entonces satura ese nodo. El segundo método que utilizan es considerar si existe un diseño parcial explorado con el mismo número de piezas colocadas que el actual pero no ha utilizado tanta región del pallet, entonces el anterior era mejor por lo que este nodo también se puede saturar. En la Figura 2.15 se muestra el árbol de búsqueda completa con este algoritmo para el problema (13,11,7,3).

Las instancias utilizadas para probar el algoritmo son 8 instancias de la literatura y dos muestras aleatorias de 500 instancias, todas del *Tipo I*. La elección de las muestras es por estratos: 100 instancias de 1 a 10 cajas, otras 100 de 11 a 20, etc. Para estas muestras obtienen una media de 20 segundos de tiempo de computación. Al no intentar resolver ningún subconjunto de *Tipo II* y no generar una muestra aleatoria del *Tipo I* es difícil valorar la calidad de este algoritmo.

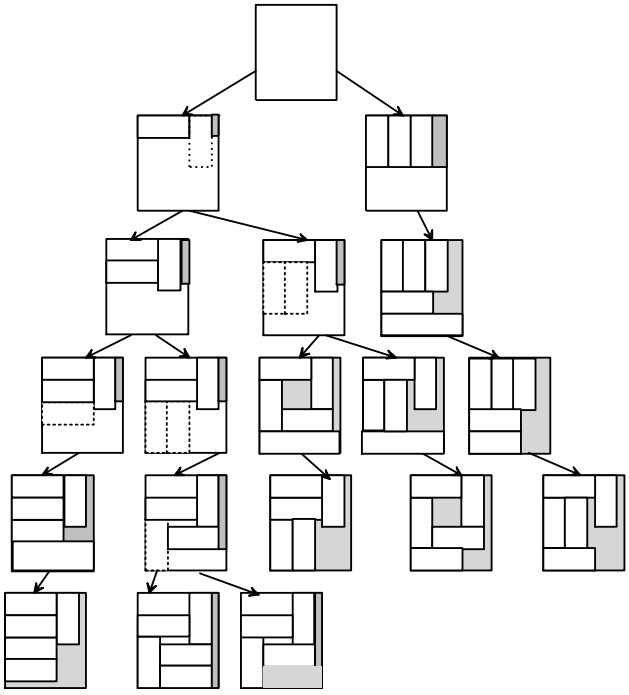


Figura 2.15: *Árbol del algoritmo de Bhattacharya para la instancia (13,11,7,3).*

Capítulo 3

Un nuevo algoritmo exacto para el *PLP* basado en branch and cut

3.1. Introducción

En este capítulo describimos el desarrollo de un nuevo algoritmo exacto para el *PLP*. Tal como se observó en el capítulo anterior, los algoritmos exactos desarrollados actualmente resuelven eficientemente los problemas de *Tipo I*, pero no se han planteado la resolución sistemática de problemas más grandes (*Tipo II*). Éste es el objetivo de este capítulo: diseñar e implementar un algoritmo que resuelva óptimamente todos los problemas de *Tipo II*.

Para ello proponemos un esquema *branch and cut* en lugar de utilizar una estructura *branch and bound* como las descritas anteriormente. En la siguiente sección se presenta una introducción a este tipo de procedimientos para posteriormente ir describiendo los componentes específicos de nuestro algoritmo. Al final del capítulo se presenta el estudio computacional que ilustra la eficiencia de nuestro método.

3.2. Algoritmos Branch and Cut

Un algoritmo *Branch and Cut* es un algoritmo *Branch and Bound* donde en cada nodo del árbol de búsqueda se puede llamar a un algoritmo de planos de corte. El uso de facetas definiendo planos de corte y la generación automática de planos de corte, en combinación con el *Branch and Bound*, fue formulado y aplicado con éxito por primera vez por Grötschel, Jünger y Reinelt [45] para el problema del ordenamiento lineal en 1984.

El término *Branch and Cut* fue introducido por Padberg y Rinaldi (1991)[70] en un algoritmo para el problema del viajante (TSP). La definición original de Padberg y Rinaldi fue hecha para un algoritmo *Branch and Cut* permitiendo únicamente planos de corte definidos por facetas, pero en la mayoría de los algoritmos no se exige que tengan que ser facetas.

Daremos una corta descripción de una versión básica de un algoritmo *Branch and Bound* y un algoritmo de planos de corte. Más detalles pueden encontrarse en el libro de Wolsey (1998)[90]. Consideremos el problema de determinar:

$$z_{OPT} = \text{máx}\,\{z(x) : x \in P, x \text{ entero}\} \tag{3.1}$$

donde z es una función lineal en x, y donde P es un poliedro. Nos referiremos a este problema como el problema π. El *Branch and Bound* hace uso de la relajación lineal

$$\tilde{\pi} : \tilde{z} = \text{máx}\,\{z(x) : x \in P\} \tag{3.2}$$

Es fácil ver que $\tilde{z} \geq z_{OPT}$. En el nivel superior o nodo raíz del árbol tenemos el problema $\tilde{\pi}$. En el nivel k del árbol tenemos una colección de problemas, que denotamos $\tilde{\pi}_1, \tilde{\pi}_2, ..., \tilde{\pi}_l$, tales que cada uno de los correspondientes poliedros $P_1, P_2, ..., P_l$ son disjuntos, y tal que todos los vectores enteros de P están contenidos en $P_1 \cup P_2 \cup \ldots \cup P_l$. Mantenemos un conjunto de *problemas abiertos*, el valor de la mejor solución posible $z^* = z(x^*)$ y la correspondiente solución x^*. Inicialmente $\tilde{\pi}$ es el único problema abierto. En la iteración i, elegimos un problema abierto $\tilde{\pi}^i$ y lo resolvemos. Si el problema $\tilde{\pi}^i$ es imposible o si el valor

$z(\bar{x}^i)$ de su solución óptima es menor que z^*, eliminamos $\tilde{\pi}^i$ de la lista de problemas abiertos y continuamos con la siguiente iteración. Si la solución óptima de $\tilde{\pi}^i$ es entera, es decir, $\bar{x}^i$ es una solución del problema π, hacemos $x^* := \bar{x}^i$ si $z(\bar{x}^i) > z^*$, eliminamos $\bar{\pi}^i$ del conjunto de problemas abiertos y continuamos con la siguiente iteración. En otro caso ramificamos.

Cuando usamos *Branch and Bound* para resolver problemas de programación entera es crucial que obtengamos buenas cotas, debido a que estas cotas son usadas para reducir el árbol de búsqueda. Como nuestro problema es de maximizar, necesitamos buenas cotas superiores, para lo cual podemos ajustar la relajación lineal añadiendo desigualdades válidas.

En un algoritmo de planos de corte mantenemos una relajación lineal de un problema π. Resolvemos $\tilde{\pi}$ dado P y si la solución óptima $\bar{x}$ es entera entonces paramos. En otro caso llamamos a algoritmos de separación basados en las diferentes familias de desigualdades válidas que consideremos. Si alguna desigualdad violada es identificada la añadimos a P. Si no encontramos desigualdades violadas, paramos.

El procedimiento *Branch and Cut* usa las dos herramientas, utiliza un algoritmo de planos de corte para hacer más fuerte la relajación lineal, y utiliza esta relajación en un algoritmo *Branch and Bound*, con la particularidad de que los cortes introducidos en un nodo son válidos para el problema original y pueden ser utilizados en los demás nodos del árbol. Por este motivo elegimos este procedimiento ya que observamos, como comentó Nelißen (1995)[67], que las dificultades del problema para los algoritmos exactos se dan en problemas en los que no existe una solución de valor igual a la cota superior, y los algoritmos exactos de tipo *Branch and Bound* tienen que explorar todos los posibles diseños hasta concluir que no existe tal solución. Un algoritmo *Branch and Cut* nos permitirá mejorar la cota mediante la inclusión de nuevas restricciones.

La Figura 3.1 muestra un diagrama general de un algoritmo *Branch and Cut*.

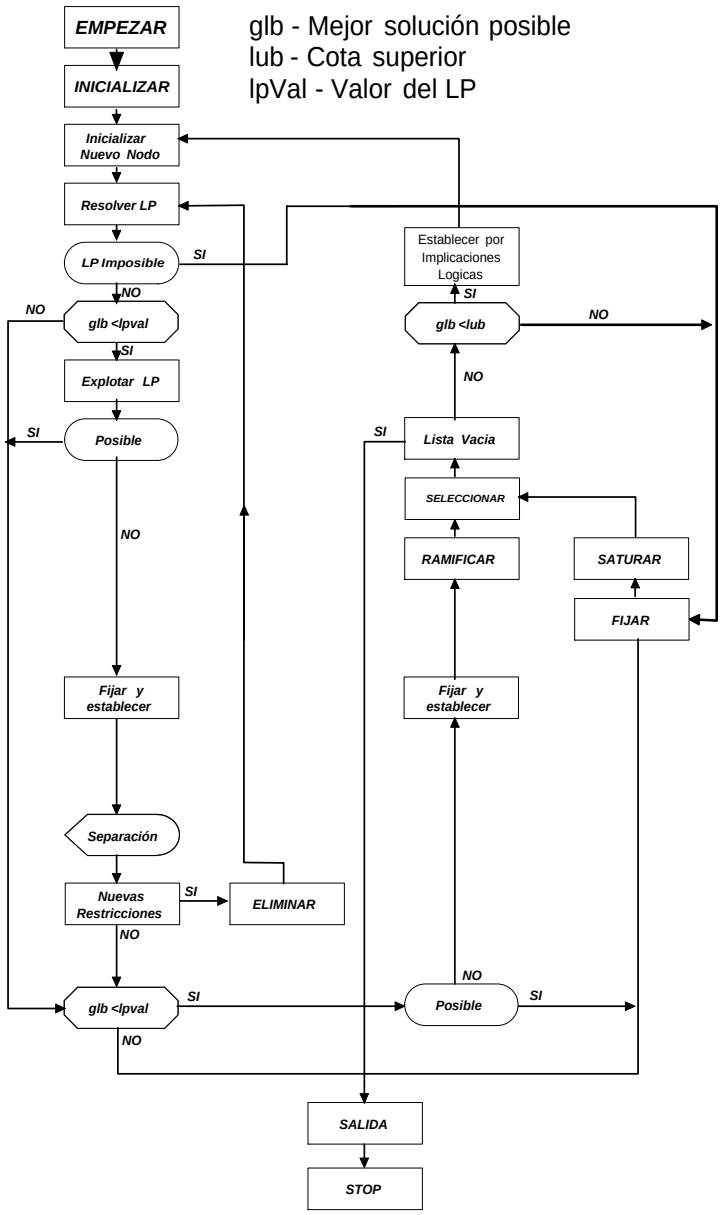


Figura 3.1: *Diagrama básico de un algoritmo* Branch and Cut.

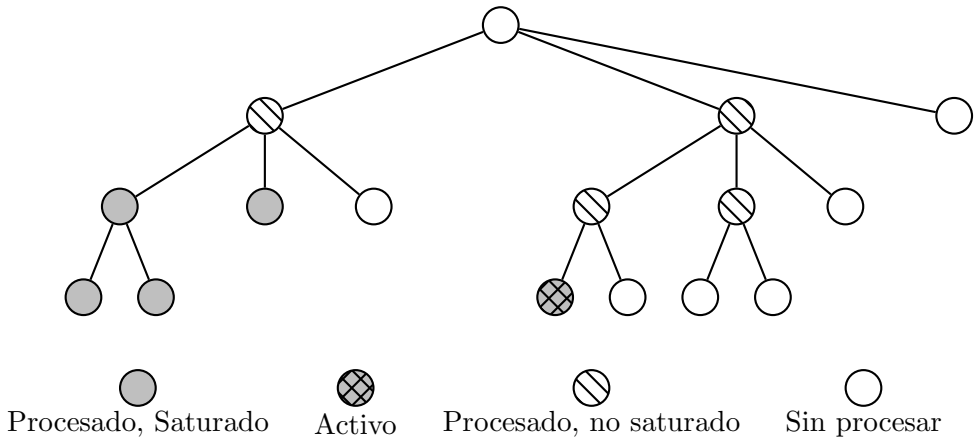


Figura 3.2: *Árbol de un algoritmo de ramificación y corte.*

3.2.1. Componentes de un *Branch and Cut*

En primer lugar explicaremos la terminología del árbol de *Branch and Cut* (Figura 3.2). En cada fase de ramificación, tal como sucede en un algoritmo de *Branch and Bound*, dos (o más) subproblemas son generados por lo que el conjunto de todos los subproblemas puede ser representado por un árbol en el que los subproblemas son los nodos. El nodo que está siendo procesado se llama nodo *activo*. Los nodos que han sido procesados pueden estar *saturados* (si ha sido procesado todo el sub-árbol restante cuya raíz es el mismo nodo) o *no saturados*. Los otros nodos no saturados en el árbol que tienen que ser procesados, se denominan *sin procesar*.

A continuación, describimos las principales componentes de un algoritmo *Branch and Cut*:

- *Formulación lineal.*
  Se parte de una formulación lineal del problema original a la que se irán añadiendo restricciones obtenidas en el proceso de planos de corte.

- *Explotar la solución del LP.*
  Si la solución obtenida al resolver la relajación lineal no es entera, se intenta obtener una buena solución entera mediante diferentes heurísticos de redondeo de la solución fraccionaria.

- *Separación.*
  La fase principal de un algoritmo *Branch and Cut* es la separación. Se intentan buscar restricciones violadas *globalmente válidas* (preferiblemente facetas), que son añadidas al LP. Se dice que una desigualdad es *globalmente válida*, si es válida para cada subproblema del problema original. Una restricción es *localmente válida*, si sólo es válida para un subproblema S y todos los subproblemas del sub-árbol cuya raíz es S.

  Puede no ser siempre una buena estrategia llamar a todos los algoritmos de separación existentes en cada iteración del proceso de separación. Diversos trabajos anteriores muestran que una jerarquía de los procedimientos de separación puede ser preferible. Ciertos métodos de separación serán probados únicamente si otros métodos no encuentran ninguna restricción.

  En algunos problemas se ha comprobado que es conveniente eliminar del LP aquellas restricciones obtenidas en las iteraciones anteriores que no sean activas. Mantener en todos los nodos del árbol de ramificación todas las restricciones obtenidas puede aumentar considerablemente el tamaño del problema lineal lo que implica un aumento en el tiempo de computación del LP.

- *Ramificar.*
  Se debe elegir la forma de ramificar en los diferentes nodos del árbol.

- *Fijar variables.*
  Al ramificar podemos fijar ciertas variables (o restricciones) por implicaciones lógicas.

3.3. Formulación entera

3.3.1. Formulación clásica para el Problema

El problema puede formularse como un caso especial de la formulación clásica para el problema de corte bidimensional no guillotina propuesto por Beasley (1985)[9] y que se ha tratado varias veces en la literatura (Dowsland (1985)[28], Arenales y Morabito (1998)[62]-[63]).

La formulación inicial de Beasley divide el pallet en cuadrados unidad y cada esquina inferior izquierda de un cuadrado es una posible localización para colocar la esquina inferior izquierda de una caja.

La formulación es la siguiente:

$$\text{máx} \sum_{k=0}^{L-l} \sum_{j=0}^{W-w} h_{kj} + \sum_{k=0}^{L-w} \sum_{j=0}^{W-l} v_{kj} \tag{3.3}$$

sujeto a:

$$\sum_{k=max\{0,r-l\}}^{min\{r,L-l\}} \sum_{j=max\{0,s-w\}}^{min\{s,W-w\}} h_{kj} + \sum_{k=max\{0,r-w\}}^{min\{r,L-w\}} \sum_{j=max\{0,s-l\}}^{min\{s,W-l\}} v_{kj} \leq 1$$
$$(r = 0, \ldots, L-1; s = 0, \ldots, W-1), \tag{3.4}$$

$$h_{kj} \in \{0,1\} \qquad (0 \leq k \leq L-l; 0 \leq j \leq W-w) \tag{3.5}$$

$$v_{kj} \in \{0,1\} \qquad (0 \leq k \leq L-w; 0 \leq j \leq W-l) \tag{3.6}$$

Para entender esta formulación es útil mirar el pallet como si estuviera formado por $L * W$ pequeños cuadrados unidad. Las coordenadas de las cuadrados unidad de las esquinas inferior-izquierda, inferior-derecha, superior-izquierda y superior-derecha del pallet son $(0,0)$, (L-1, 0), $(0, W$-1), (L-1,W-1) respectivamente. Las variables h identifican cajas colocadas de forma horizontal y las v de

forma vertical. Los subíndices indican el extremo inferior izquierdo de esa caja.

$$h_{kj} = \begin{cases} 1 & \text{si una caja horizontal está colocada en la posición (k,j),} \\ 0 & \text{en otro caso.} \end{cases} \tag{3.7}$$

$$v_{kj} = \begin{cases} 1 & \text{si una caja vertical está colocada en la posición (k,j),} \\ 0 & \text{en otro caso.} \end{cases} \tag{3.8}$$

Las restricciones (3.4) son las llamadas *restricciones de cubrimiento* que obligan a que no se solapen cajas. Cada restricción individual de la forma (3.4) asegura que cada cuadrado está cubierto como máximo por una caja.

Tendríamos:

- $(L-l)*(W-w)$ variables v
- $(L-w)*(W-l)$ variables h
- $(L-w)*(W-w)$ restricciones

Aunque parece que dependiendo del tamaño de L y W, el tamaño del modelo puede ser considerablemente grande, veremos que el tamaño se puede reducir drásticamente.

3.3.2. Reducción del tamaño del problema

En esta sección explicamos cómo se han construido el conjunto de posibles emplazamientos de variables y el conjunto de restricciones para que resulte un problema equivalente al original con el menor número tanto de variables como de restricciones. La reducción se realiza mediante el uso del concepto de particiones posibles y teniendo en cuenta diferentes consideraciones de dominancia como explicamos a continuación. Partiendo de la formulación clásica de Beasley los conjuntos de puntos donde colocar una variable se pueden reducir sin pérdida de generalidad. Vamos a explicar las diferentes reducciones para los posibles emplazamientos de las cajas:

- *Conjuntos normalizados.*
  Se puede reducir el conjunto de emplazamientos a los puntos de los conjuntos normalizados propuesto por Herz[49] y Christofides y Whitlock[22].

$$S(L) = S(L, l, w) = \{r : r = \alpha l + \beta w, r + w \leq L, \alpha, \beta \in \mathcal{Z}_+\} \tag{3.9}$$

$$S(W) = S(W, l, w) = \{r : r = \alpha l + \beta w, r + w \leq W, \alpha, \beta \in \mathcal{Z}_+\} \tag{3.10}$$

- *Puntos interiores.*
  Scheithauer y Terno (1996)[78] proponen una nueva reducción basada en ideas de dominancia. Los conjuntos normalizados se reducen a los que denominaremos *puntos interiores* utilizados en el *algoritmo G4*. Si denotamos:

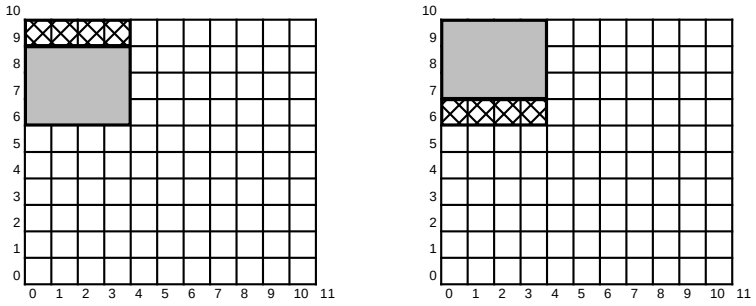
$$\langle s \rangle_L := max\{r \in S(L) : r \leq s\} \tag{3.11}$$

  el conjunto de puntos interiores se define como:

$$\widetilde{S}(L) = \widetilde{S}(L, l, w) = \{\langle L - r \rangle_L : r \in S(L)\} \tag{3.12}$$

  Análogamente puede definirse $\widetilde{S}(W) = \widetilde{S}(W, l, w)$.

- *Nueva reducción.*
  Finalmente nosotros proponemos reducir aún más el número de variables distinguiendo la posición de variables horizontales y variables verticales, como podemos ver en el ejemplo $(11, 10, 4, 3)$. Tenemos:
  $S(W, l, w) = S(10, 4, 3) = \{0, 3, 4, 6, 7\}$
  $\widetilde{S}(W, l, w) = \widetilde{S}(10, 4, 3) = \{0, 3, 4, 6, 7\}$.
  $S(L, l, w) = S(11, 4, 3) = \{0, 3, 4, 6, 7, 8\}$
  $\widetilde{S}(L, l, w) = \widetilde{S}(11, 4, 3) = \{0, 3, 4, 7, 8\}$.
  De acuerdo con $\widetilde{S}(L, l, w)$ y $\widetilde{S}(W, l, w)$, existiría la variable $h_{0,6}$. Sin embargo, estaría dominada por la variable $h_{0,7}$. La variable $h_{0,6}$ dejaría una región de (1×4) no utilizable, mientras que $h_{0,7}$ permitiría utilizar la región (1×4) por otra variable, con lo que en cualquier solución que aparezca $h_{0,6}$



(a) Variable $h_{0,6}$ (b) Variable $h_{0,7}$

Figura 3.3: *Reducción de variables. Instancia (11,10,4,3).*

ésta se puede sustituir por $h_{0,7}$ obteniendo una solución igual o mejor que la anterior, ver Figura 3.3.

Por tanto no tenemos los mismos puntos interiores para las variables h que para las variables v. Tendríamos:

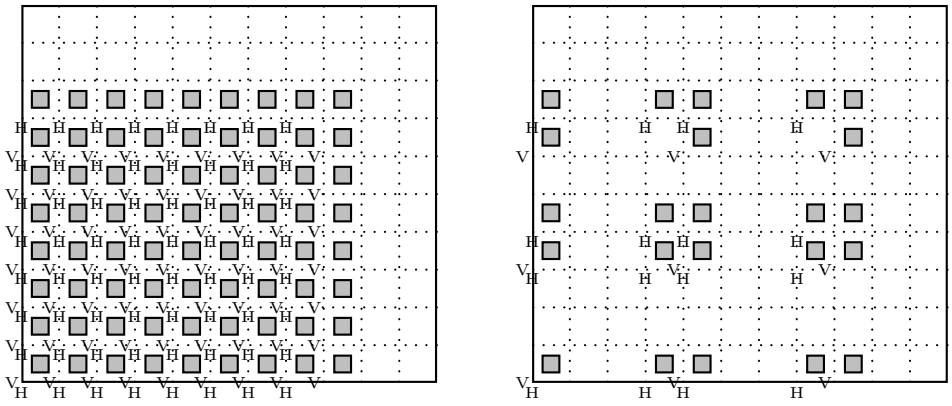
$$\begin{aligned}
S_V(L) &= S_V(L,l,w) = \{r : r = \alpha l + \beta w, r \leq L, \alpha \in \mathcal{Z}_+, \beta \in \mathcal{Z}_+ \setminus 0\} \\
S_V(W) &= S_V(W,l,w) = \{r : r = \alpha l + \beta w, r \leq W, \alpha \in \mathcal{Z}_+ \setminus 0, \beta \in \mathcal{Z}_+\} \\
S_H(L) &= S_H(L,l,w) = \{r : r = \alpha l + \beta w, r \leq L, \alpha \in \mathcal{Z}_+ \setminus 0, \beta \in \mathcal{Z}_+\} \\
S_H(W) &= S_H(W,l,w) = \{r : r = \alpha l + \beta w, r \leq W, \alpha \in \mathcal{Z}_+, \beta \in \mathcal{Z}_+ \setminus 0\}
\end{aligned}$$

A partir de estos conjuntos construiríamos los conjuntos normalizados:
$\widetilde{S}_V(L) = \{\langle L - r\rangle_L : r \in S_V(L)\}$, $\widetilde{S}_V(W) = \{\langle W - r\rangle_W : r \in S_V(W)\}$,
$\widetilde{S}_H(L) = \{\langle L - r\rangle_L : r \in S_H(L)\}$, $\widetilde{S}_V(W) = \{\langle W - r\rangle_W : r \in S_H(W)\}$.

Este proceso de reducción de variables puede también afectar al número de restricciones. Cuando consideramos la restricción correspondiente a un cuadrado (r, s), si todas las variables implicadas en esa restricción ya han aparecido en una restricción *anterior*, es decir, en una restricción correspondiente a un cuadrado (r', s') situado a la izquierda y/o por debajo del cuadrado (r, s), la restricción correspondiente al cuadrado (r, s) es redundante. En este caso la restricción que

impide el solapamiento en el cuadrado (r', s') es también responsable del no solapamiento en el cuadrado (r, s).

En la Figura 3.4 se muestra las variables (H y V) y para qué cuadrados había restricciones (cuadrados en gris) para el problema original y con la nueva reducción.



(a) Original (b) Nueva Reducción

Figura 3.4: *Reducción de la instancia (11,10,4,3).*

Formulación	Variables h	Variables v	Total	Restricciones
Beasley	49	48	97	56
Conjuntos normalizados	25	24	49	30
Puntos interiores	20	20	40	25
Nueva Reducción	16	9	25	23

Tabla 3.1: *Reducción de variables. Instancia (11,10,4,3).*

En la Tabla 3.1 se muestran las distintas reducciones para la instancia (11,10,4,3) y en la Tabla 3.2 observamos la comparación de los puntos interiores y la nueva reducción para la media de variables y restricciones de los dos conjuntos completos de problemas test.

	Puntos interiores		Nueva Reducción	
	Variables	Restricciones	Variables	Restricciones
Tipo I	373,46	212,34	266,37	211,89
Tipo II	1406,35	753,03	1041,25	752,49

Tabla 3.2: *Comparación entre los puntos interiores y la nueva reducción.*

Los conjuntos de variables y de restricciones dependen únicamente de las particiones eficientes y éstas a su vez definen las clases de equivalencia. Por tanto, existe una relación biunívoca entre las variables y restricciones de cualquier par de miembros de una clase de equivalencia. La formulación con estas nuevas reducciones (ver sección 3.3.4) es única para todos los miembros de una misma clase de equivalencia y el tamaño del LP no viene determinado por el orden de L y W. Por ejemplo, la instancia (42,28,8,5) tiene 161 variables y una equivalente, por ejemplo la instancia (25786,17727,4835,3223), las mismas 161 variables.

3.3.3. Una restricción de cota superior

Como se ha mencionado en el capítulo 2, existen buenos algoritmos heurísticos y buenas cotas superiores para el PLP. Como la función objetivo de este problema es el número de cajas que pueden colocarse en el pallet, el valor de la solución óptima tiene un estrecho rango de posibles valores. De hecho, dado n_{Heur}, el valor de una solución posible, y n_{Cota}, el valor de una cota superior, nosotros buscamos una solución con n_{Opt} cajas, donde $n_{Heur} < n_{Opt} \leq n_{Cota}$. Aprovechando este hecho, en el árbol de búsqueda definimos un primer nivel de ramificación en el que cada nodo tiene un número fijo de cajas, decreciente desde n_{Cota} hasta $n_{Heur}+1$. Comenzamos estudiando la rama con el mayor número de cajas y la exploramos hasta que encontremos una solución posible, y por tanto óptima, o la rama haya sido completamente estudiada. En este caso pasamos a la rama siguiente con una caja menos y así sucesivamente.

Por tanto, cuando resolvemos la relajación lineal de la formulación entera en cada nodo del árbol, añadimos la restricción de cota superior:

$$\sum_{k\in\widetilde{S}_{HL}}\sum_{j\in\widetilde{S}_{HW}} h_{kj} + \sum_{k\in\widetilde{S}_{VL}}\sum_{j\in\widetilde{S}_{VW}} v_{kj} \leq B \tag{3.13}$$

donde B es el número de cajas correspondientes a la rama a la que pertenece el nodo en estudio.

Como consecuencia de ello, la *pérdida*, es decir, la superficie no utilizada del pallet correspondiente a cada rama es también conocida. En una rama con n cajas, la pérdida es $\mathcal{P} = L*W - n*l*w$. Esta información puede utilizarse para mejorar la formulación. Cuando describimos la reducción de restricciones mencionamos que una restricción puede ser responsable de evitar el solapamiento de varios cuadrados unidad. Por tanto, si todas sus variables son 0 no uno sino varios cuadrados estarán vacíos. Si el número de cuadrados asociados a la restricción es mayor que la pérdida $\mathcal{P}$, está restricción tiene que fijarse a uno. Este cambio en el status de la restricción no es necesario en la formulación entera, pero es relevante en su relajación lineal.

La Figura 3.5 ilustra el primer nivel de ramificación sobre un ejemplo.

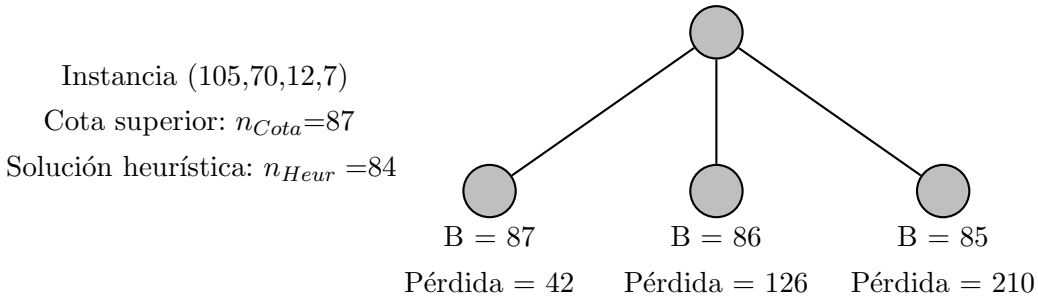


Figura 3.5: *Ramificación en el nodo raíz.*

Concretamente, utilizaremos la mejor de las siguientes cotas:

- La cota de Barnes (1979)[8].
- La menor cota de la clase de equivalencia (1985)[28].
- La cota de Nelißen (1995)[67].

3.3.4. Formulación para el problema

La formulación que utilizaremos será la siguiente:

$$\text{máx} \sum_{k\in\widetilde{S}_H(L)} \sum_{j\in\widetilde{S}_H(W)} h_{kj} + \sum_{k\in\widetilde{S}_V(L)} \sum_{j\in\widetilde{S}_V(W)} v_{kj} \tag{3.14}$$

sujeto a:

$$\sum_{\substack{k\in\widetilde{S}_H(L)\\ p-l<k\leq p}} \sum_{\substack{j\in\widetilde{S}_H(W)\\ p-w<j\leq p}} h_{kj} + \sum_{\substack{k\in\widetilde{S}_V(L)\\ q-w<k\leq q}} \sum_{\substack{j\in\widetilde{S}_V(W)\\ q-a<j\leq q}} v_{kj} \leq 1 \quad (p,q)\in\widetilde{S}_{Res} \tag{3.15}$$

$$\sum_{\substack{k\in\widetilde{S}_H(L)\\ p-l<k\leq p}} \sum_{\substack{j\in\widetilde{S}_H(W)\\ p-w<j\leq p}} h_{kj} + \sum_{\substack{k\in\widetilde{S}_V(L)\\ q-w<k\leq q}} \sum_{\substack{j\in\widetilde{S}_V(W)\\ q-a<j\leq q}} v_{kj} = 1 \quad (p,q)\in S'\subseteq\widetilde{S}_{Res} \tag{3.16}$$

$$\sum_{k\in\widetilde{S}_H(L)} \sum_{j\in\widetilde{S}_H(W)} h_{kj} + \sum_{k\in\widetilde{S}_V(L)} \sum_{j\in\widetilde{S}_V(W)} v_{kj} \leq Cota \tag{3.17}$$

$$h_{kj}\in\{0,1\} \qquad k\in\widetilde{S}_H(W), j\in\widetilde{S}_H(W) \tag{3.18}$$

$$v_{kj}\in\{0,1\} \qquad k\in\widetilde{S}_V(W), j\in\widetilde{S}_V(W) \tag{3.19}$$

$$\widetilde{S}_{Res} = \{(p,q)|p\in\widetilde{S}_H(L)\cup\widetilde{S}_V(L), q\in\widetilde{S}_H(W)\cup\widetilde{S}_V(W)\} \tag{3.20}$$

3.4. Reducción del problema del Pallet Loading a un problema de grafos

3.4.1. Definición del grafo original

Como describió K. Dowsland (1987)[30], es posible asociar un grafo con la formulación descrita anteriormente, definiendo un vértice por cada variable y un arco entre dos vértices si y sólo si las dos variables aparecen juntas en una restricción de cubrimiento (se solapan las dos cajas). A este grafo lo llamaremos G_{LWlw}. El problema del pallet se transforma en un problema de encontrar el conjunto máximo independiente en el grafo G_{LWlw}. Cada restricción de la forma (3.15) corresponde a un clique del grafo. En la Figura 3.6 observamos el grafo para la instancia (5,5,3,2).

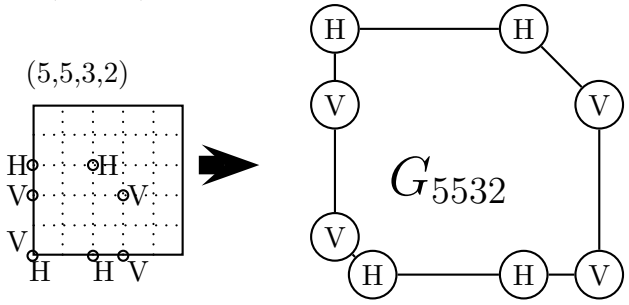


Figura 3.6: *Transformación en un problema de conjunto máximo independiente.*

3.4.2. Nuevas Relaciones entre Variables

Además de las relaciones de solapamiento podemos distinguir dos nuevos tipos de relaciones:

- *Relaciones debidas a la pérdida.*
  Dos cajas no pueden estar juntas en una solución con un número n de cajas si la pérdida que se produciría al colocar las dos cajas en la solución es mayor que la pérdida asociada a dicha solución.

- *Relaciones de dominancia.*
  Un par de cajas no se permite en una solución si existe otro par que produzca menor pérdida, sin provocar cambios en el resto de la solución.

Para definir las distintas relaciones, utilizamos la siguiente notación:

$$(x_i, y_i) = \textit{esquina inferior izquierda de la caja } i \tag{3.21}$$

$$o_i = \textit{Orientación de la caja } i \tag{3.22}$$

$$dh_i = \begin{cases} l & \textit{Si la caja } i \textit{ es horizontal}, \\ w & \textit{Si la caja } i \textit{ es vertical}. \end{cases} \tag{3.23}$$

$$dv_i = \begin{cases} w & \textit{Si la caja } i \textit{ es horizontal}, \\ l & \textit{Si la caja } i \textit{ es vertical}. \end{cases} \tag{3.24}$$

3.4.2.1. Relaciones debidas a la pérdida

A continuación definimos los diferentes tipos de relaciones debidas a la pérdida, describiendo las condiciones que se han de cumplir en cada caso. Suponemos que las cajas no se solapan:

1. *Huecos directos entre cajas.*
   Son huecos existentes entre las dos cajas cuando no cabe ninguna caja entre ellas.

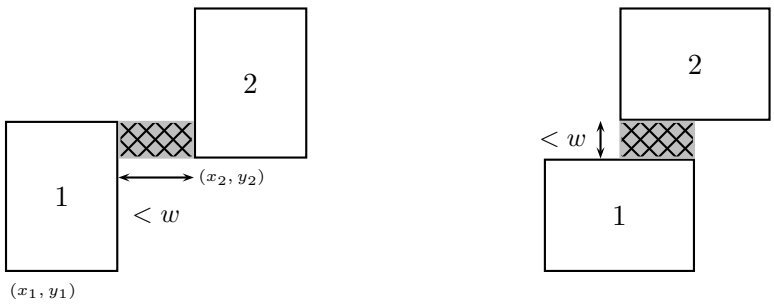
   a) Hueco en horizontal (Figura 3.7(a)).

$$x_2 > x_1 + dh_1 \tag{3.25}$$

$$y_2 < y_1 + dv_1 \tag{3.26}$$

$$y_2 + dv_2 > y_1 \tag{3.27}$$

$$x_2 - (x_1 + dh_1) < w \tag{3.28}$$



(a) Horizontales (b) Verticales

Figura 3.7: *Huecos directos.*

El hueco ocasionado es:

$$[x_2 - (x_1 + dh_1)] \times [\text{mín}\{y_1 + dv_1, y_2 + dv_2\} - \text{máx}\{y_1, y_2\}] \quad (3.29)$$

b) Hueco en Vertical (Figura 3.7(b)).

$$y_2 > y_1 + dv_1 \quad (3.30)$$

$$x_2 < x_1 + dh_1 \quad (3.31)$$

$$x_2 + dh_2 > x_1 \quad (3.32)$$

$$y_2 - (y_1 + dv_1) < w \quad (3.33)$$

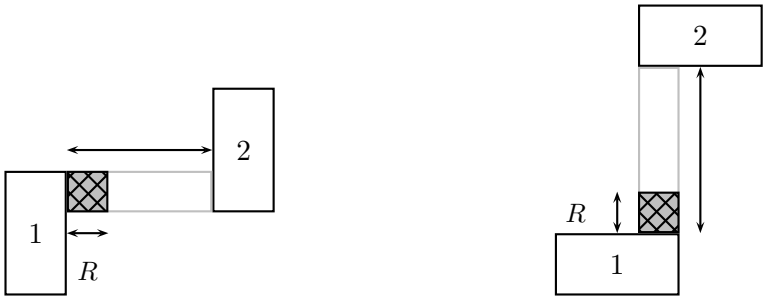
El hueco ocasionado es:

$$[y_2 - (y_1 + dv_1)] \times [\text{mín}\{x_1 + dh_1, x_2 + dh_2\} - \text{máx}\{x_1, x_2\}] \quad (3.34)$$

2. *Huecos indirectos entre cajas.*

   En el espacio entre las dos cajas cabe alguna otra caja, pero aunque obtengamos la utilización más eficiente de este espacio aún seguiría quedando un hueco. Por notación definimos:

$$Resto(x) = \{x - \{\text{máx}\, s | s = \alpha * l + \beta * w < x : \alpha \in Z^+, \beta \in Z^+\}\} \quad (3.35)$$



(a) Horizontales (b) Verticales

Figura 3.8: *Huecos indirectos.*

a) Hueco en horizontal (Figura 3.8(a).)

$$x_2 > x_1 + dh_1 \quad (3.36)$$

$$y_2 < y_1 + dv_1 \quad (3.37)$$

$$y_2 + dv_2 > y_1 \quad (3.38)$$

$$Resto(x_2 - (x_1 + dh_1)) > 0 \quad (3.39)$$

El hueco ocasionado es:

$$[Resto(x_2-(x_1+dh_1))]\times[\text{mín}\{y_1+dv_1, y_2+dv_2\}-\text{máx}\{y_1, y_2\}] \quad (3.40)$$

b) Hueco en Vertical (Figura 3.8(b)).

$$y_2 > y_1 + dv_1 \quad (3.41)$$

$$x_2 < x_1 + dh_1 \quad (3.42)$$

$$x_2 + dh_2 > x_1 \quad (3.43)$$

$$Resto(y_2 - (y_1 + dv_1)) > 0 \quad (3.44)$$

El hueco ocasionado es:

$$[Resto(y_2-(y_1+dv_1))]\times[\text{mín}\{x_1+dh_1, x_2+dh_2\}-\text{máx}\{x_1, x_2\}] \quad (3.45)$$

3. *Huecos en las esquinas del pallet H1* (Figura 3.9).
   Las cajas están a una distancia de los lados del pallet menor que el lado grande de la caja, es decir:

$$x_2 < l \qquad y_1 < l \tag{3.46}$$

$$x_1 < x_2 \qquad y_2 < y_1 \tag{3.47}$$

   Definimos:

$$d_1 = x_1 \tag{3.48}$$

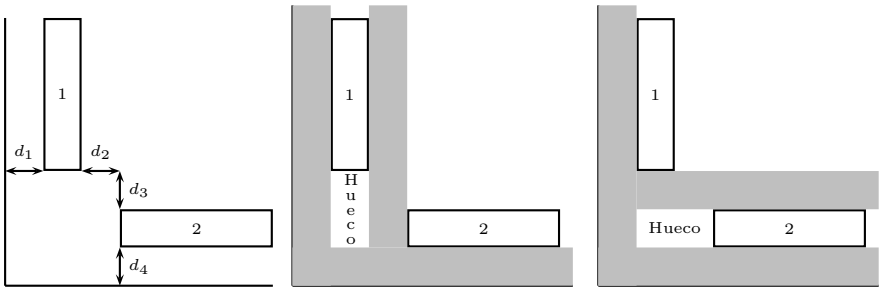
$$d_2 = \text{máx}\{0, x_2 - (x_1 + dh_1)\} \tag{3.49}$$

$$d_3 = \text{máx}\{0, y_1 - (y_2 + dv_2)\} \tag{3.50}$$

$$d_4 = y_2 \tag{3.51}$$

   El hueco ocasionado es:

$$\text{mín}\{(x_2 - \left\lfloor \frac{d_1}{w} \right\rfloor w - \left\lfloor \frac{d_2}{w} \right\rfloor w) * (y_1 - \left\lfloor \frac{d_4}{w} \right\rfloor w), \\ (y_1 - \left\lfloor \frac{d_3}{w} \right\rfloor w - \left\lfloor \frac{d_4}{w} \right\rfloor w) * (x_2 - \left\lfloor \frac{d_1}{w} \right\rfloor w)\} \tag{3.52}$$



(a) Original (b) Cómo se puede rellenar

Figura 3.9: *Huecos en las esquinas* H1.

4. *Huecos en las esquinas del pallet H2* (Figura 3.10).
   Las cajas están a una distancia de los lados del pallet superior al lado grande de la caja, pero cumpliendo las condiciones:

$$x_1 < x_2 < l + w \qquad x_1 < w \tag{3.53}$$

$$y_2 < y_1 < l + w \qquad y_2 < w \tag{3.54}$$

$$\text{máx}\{0, x_2 - (x_1 + dh_1)\} < w \quad \text{máx}\{0, y_1 - (y_2 + dv_2)\} < w \tag{3.55}$$

   definimos:

$$d_2 = \text{máx}\{0, x_2 - (x_1 + dh_1)\} < w \tag{3.56}$$

$$d_3 = \text{máx}\{0, y_1 - (y_2 + dv_2)\} < w \tag{3.57}$$

   El hueco ocasionado es:

$$(x_2 * y_1) - (\left\lfloor \frac{x_2 * y_1 - (d_2 * d_3)}{l * w} \right\rfloor * (l * w)) - (d_2 * d_3) \tag{3.58}$$

Estos huecos en las esquinas se pueden generalizar a cualquier esquina del pallet.

Por tanto, en todos los casos de huecos directos, indirectos o en las esquinas, si la pérdida producida por la posición de las cajas 1 y 2 supera la pérdida asociada con la solución, esas dos cajas no pueden aparecer a la vez en esa solución y podemos poner una arista entre ellas.

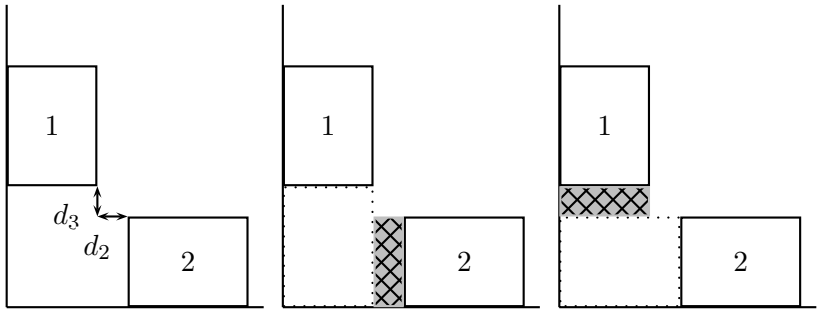


Figura 3.10: *Huecos en las esquinas* H2.

3.4.2.2. Relaciones de dominancia

Decimos que un par de cajas está *dominado* por otro par si este segundo par utiliza una región estrictamente menor del pallet, dejando el resto del diseño inalterado. En este caso entre los vértices correspondientes al par de cajas dominado introducimos una arista para impedir que aparezcan juntas en la solución. Este tipo de relaciones las definimos como *relaciones de dominancia*.

De acuerdo con la definición anterior, pueden darse situaciones en las que un par de cajas esté dominado por varios pares de cajas diferentes. Siempre optaremos por una dominancia que desplace la pérdida hacia el centro del pallet. Como se observa en la Figura 3.11, la dominancia desplaza las cajas hacia la esquina más cercana y las pérdidas hacia el centro.

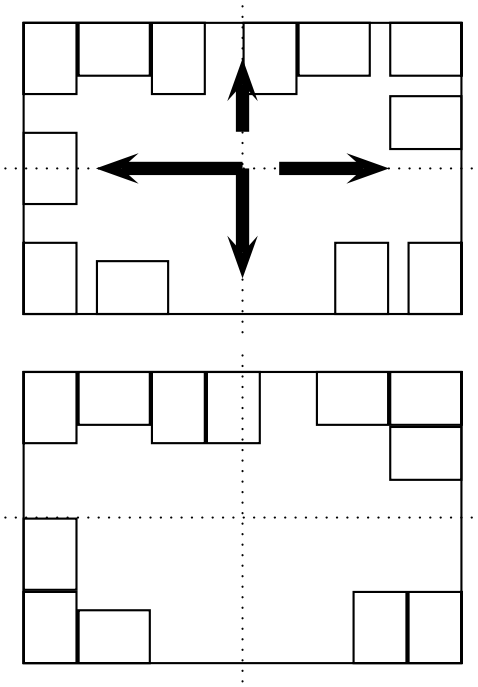


Figura 3.11: *Sentido de aplicación de la dominancia.*

Pasamos a continuación a definir los diferentes tipos de relaciones de dominancia:

1. *Dominancia entre cajas de tipo 1.*
   Vamos a estudiarlas de forma horizontal, pero fácilmente se pueden transformar a vertical por simetría. Tenemos una caja horizontal y otra caja en vertical, o las dos cajas en vertical. Suponemos que la caja 1 está a la izquierda de la caja 2 (Figura 3.12).

   a) Una caja vertical u horizontal, Caja 1, y otra horizontal, Caja 2.
   Condiciones:

   1) Mitad izquierda del pallet.

$$y_1 \leq y_2 \leq y_2 + dv_2 \leq y_1 + dv1 \tag{3.59}$$
$$0 < x_2 - (x_1 + dh_1) < w \tag{3.60}$$
$$x_2 \leq \frac{L}{2} \tag{3.61}$$
$$\exists\ Var_{o_2} \text{ en } ([x_1 + dh_1, x_2 - 1], y_2) \tag{3.62}$$

   2) Mitad derecha del pallet.

$$y_2 \leq y_1 \leq y_1 + dv_1 \leq y_2 + dv2 \tag{3.63}$$
$$0 < x_2 - (x1 + dh_1) < b \tag{3.64}$$
$$x_1 > \frac{L}{2} \tag{3.65}$$
$$\exists\ Var_{o_1} \text{ en } ([x_1 + dh_1, x_2 - 1], y_1) \tag{3.66}$$

b) Las dos cajas verticales.

Condiciones:

1) Mitad izquierda del pallet.

$$y_1 = y_2 \tag{3.67}$$

$$0 < x_2 - (x_1 + dh_1) < w \tag{3.68}$$

$$x_2 \leq \frac{L}{2} \tag{3.69}$$

$$\exists\ Var_V \text{ en } ([x_1 + dh_1, x_2 - 1], y_1) \tag{3.70}$$

2) Mitad derecha del pallet.

$$y_1 = y_2 \tag{3.71}$$

$$0 < x_2 - (x_1 + dh_1) < b \tag{3.72}$$

$$x_1 > \frac{L}{2} \tag{3.73}$$

$$\exists\ Var_V \text{ en } ([x_1 + dh_1, x_2 - 1], y_1) \tag{3.74}$$

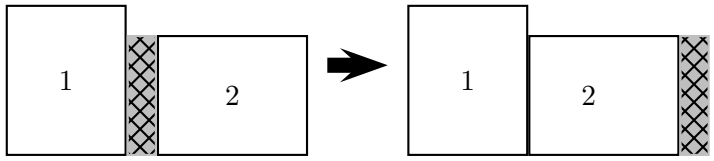


Figura 3.12: *Dominancia entre cajas de tipo 1.*

2. *Dominancia en las esquinas D1.*

Se basa en las relaciones existentes de huecos en las esquinas *H1* (Figura 3.9).

$$x_2 < l \qquad y_1 < l \tag{3.75}$$

$$x_1 < x_2 \qquad y_2 < y_1 \tag{3.76}$$

$$\exists\ Var_{o_1} \text{ en } (x_1, [y_2, y_1 - 1]) \tag{3.77}$$

$$\exists\ Var_{o_2} \text{ en } ([x_1 + dh_1, x_2 - 1], y_2) \tag{3.78}$$

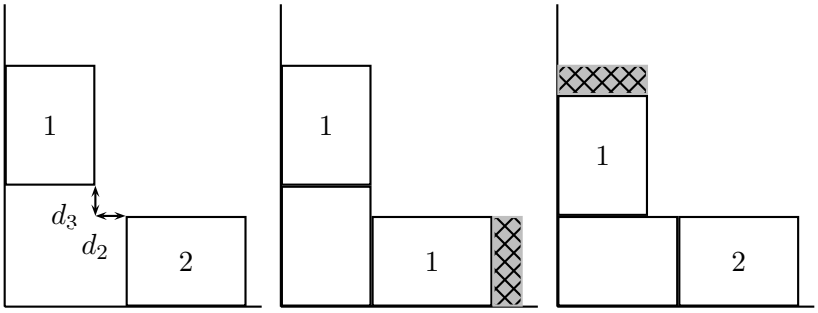


Figura 3.13: *Dominancia* D2.

3. *Dominancia en las esquinas D2.*
   Basado en las relaciones existentes de huecos en las esquinas *H2* (Figura 3.13).

$$x_1 < x_2 < l + w \qquad x_1 < w \tag{3.79}$$

$$y_2 < y_1 < l + w \qquad y_2 < w \tag{3.80}$$

$$\text{máx}\{0, x_2 - (x_1 + dh_1)\} < w \qquad \text{máx}\{0, y_1 - (y_2 + dv_2)\} < w \tag{3.81}$$

$$\exists\ Var_{o_1} \text{ en } (x_1, [y_2, y_1 - 1]) \tag{3.82}$$

$$\exists\ Var_{o_2} \text{ en } ([x_1 + dh_1, x_2 - 1], y_2) \tag{3.83}$$

4. *Dominancia D3.*
   Trata de evitar diseños con cajas como las que vemos en la Figura 3.14. Debemos tener dos cajas con la misma orientación, con la condición $l < 2{*}w$. Vamos a ver el caso horizontal, pues el vertical sería equivalente por simetría (Figura 3.14(a)).

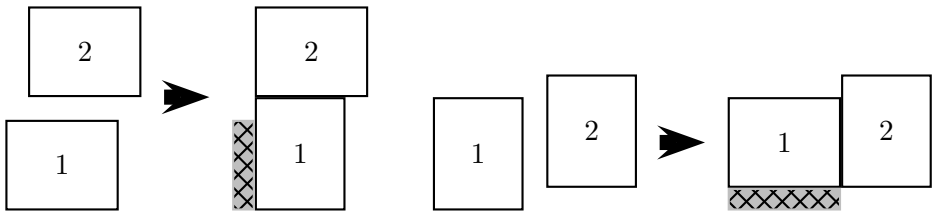
$$l \leq y_1 - y_2 \tag{3.84}$$

$$w \leq (x_1 + dh_1) - x_2 \tag{3.85}$$

$$\exists\ Var_{o_2} \text{ en } ([x_1, (x_2 + dh_2) - w], [y_2, y_1 - l]) \tag{3.86}$$

ó

$$\exists\ Var_{o_2} \text{ en } ([x_1, (x_2 + dh_2) - w], [y_1 + dv_1, y_2 + dv_2 - l]) \tag{3.87}$$



(a) Horizontal (b) Vertical

Figura 3.14: *Dominancia* D3.

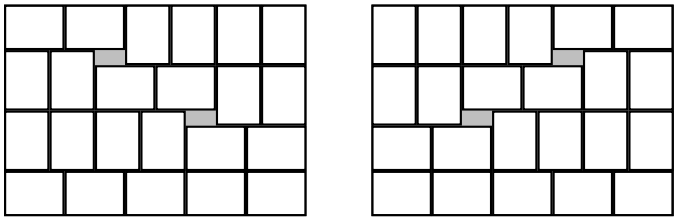
3.4.3. Soluciones simétricas

Es muy frecuente tener soluciones que son simétricas a otras, respecto a un eje vertical que pase por el centro del pallet, respecto a un eje horizontal o respecto ambos simultáneamente (Figura 3.15).

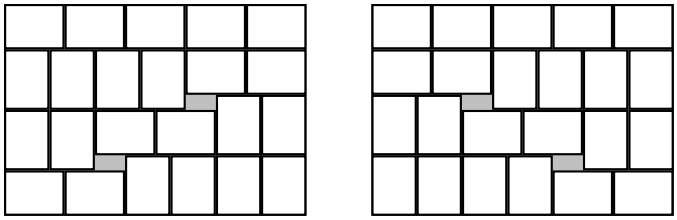
Esta simetría multiplica el conjunto de soluciones posibles y es una dificultad añadida para los métodos exactos de resolución. Por ello, vamos a desarrollar algunas estrategias que permitan no considerar soluciones simétricas de otras ya estudiadas.

Tal como comentamos en la sección 3.3.3 en cada rama del árbol de búsqueda tendremos fijado el número de cajas de la solución buscada n y con ello la pérdida asociada $\mathcal{P} = (L * W) - (n * l * w)$.

Si consideramos el pallet dividido verticalmente por la mitad, es obvio que podremos considerar únicamente soluciones para las que la pérdida en el lado izquierdo, A, sea menor que $\mathcal{P}/2$ ya que si no se da en A tendríamos que se daría en el lado derecho, B, y por simetría podríamos conseguirla en A (Figura 3.16(a)). De forma similar, si consideramos el pallet dividido horizontalmente sólo necesitamos estudiar soluciones en las que la pérdida en la mitad inferior sea menor que $\mathcal{P}/2$ (Figura 3.16(b)).

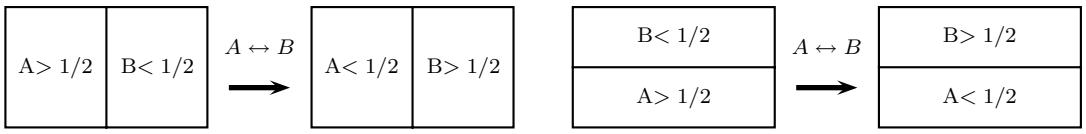


(a) Original (b) Sobre eje vertical



(c) Sobre eje horizontal (d) Sobre los dos ejes

Figura 3.15: *Simetrías sobre los ejes para la instancia (20,14,4,3).*



(a) Simetría vertical (b) Simetría horizontal

Figura 3.16: *Simetrías respecto de los ejes.*

Además estas dos condiciones se pueden dar conjuntamente ya que lo único que ocurriría es que habría que hacer dos simetrías respecto a los ejes.

B	**D**
A	**C**

Figura 3.17: *División del pallet en cuadrantes.*

En un segundo paso, buscamos zonas del pallet donde podamos ajustar más la pérdida. Para ello dividimos el pallet en 4 cuadrantes (Figura 3.17):

1) La pérdida en el cuadrante A, $P_A \leq \frac{\mathcal{P}}{4}$.

   Demostración:

   Supongamos que existe una solución con $P_A > \frac{\mathcal{P}}{4}$

   Entonces $\mathcal{P} - P_A < \frac{3\mathcal{P}}{4} \Rightarrow \frac{\mathcal{P} - P_A}{3} < \frac{\mathcal{P}}{4} \Rightarrow \exists$ cuadrante con pérdida $< \frac{\mathcal{P}}{4}$

   Por simetría, A sería ese cuadrante.

   1.1) Si L par y W par: $P_A \leq \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$

   Demostración:

   Supongamos que $P_A > \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$

   Como L par y W par, P_A es entero y entonces $P_A \geq \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor + 1$

   Por lo tanto, $\mathcal{P} - P_A \leq \mathcal{P} - \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor - 1 < \mathcal{P} - \frac{\mathcal{P}}{4} = \frac{3\mathcal{P}}{4}$

   como $\mathcal{P} - P_A$ es entero, $\mathcal{P} - P_A \leq \left\lfloor \frac{3\mathcal{P}}{4} \right\rfloor \rightarrow \frac{\mathcal{P} - P_A}{3} \leq \left\lfloor \frac{3\mathcal{P}}{4} \right\rfloor \Big/ 3 \Rightarrow$

   podemos escribir: $\mathcal{P} = 4\text{a} + \text{b}$, $\text{b} < 4$ donde $\text{a} = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$

Luego, $3\mathcal{P} = 12a + 3b$

Dividiendo por 4 y redondeando al entero inferior más próximo:

$\left\lfloor \frac{3\mathcal{P}}{4} \right\rfloor = 3a + \left\lfloor \frac{3b}{4} \right\rfloor$

Si dividimos por 3:

$$\left\lfloor \frac{3\mathcal{P}}{4} \right\rfloor \Big/ 3 = a + \left\lfloor \frac{3b}{4} \right\rfloor \Big/ 3 = \begin{cases} a & \text{si } b = 0 \\ a & \text{si } b = 1 \\ a + \frac{1}{3} & \text{si } b = 2 \\ a + \frac{2}{3} & \text{si } b = 3 \end{cases}$$

Por tanto $\left\lfloor \left\lfloor \frac{3\mathcal{P}}{4} \right\rfloor \Big/ 3 \right\rfloor = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$ y $\frac{\mathcal{P} - P_A}{3} \leq \left\lfloor \frac{3\mathcal{P}}{4} \right\rfloor \Big/ 3$

Hay al menos 1 cuadrante con $\left\lfloor \frac{\mathcal{P} - P_A}{3} \right\rfloor \leq \left\lfloor \left\lfloor \frac{3\mathcal{P}}{4} \right\rfloor \Big/ 3 \right\rfloor = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$

Por simetría, este cuadrante sería A.

1.2) Si L ó W par: $P_A = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor_{0,5}$

(donde $\lfloor \rfloor_{0,5}$ denota redondear al múltiplo de 0.5 inferior más próximo)

Demostración:

Si $P_A > \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor_{0,5} \rightarrow P_A \geq \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor_{0,5} + \frac{1}{2}$

$\mathcal{P} - P_A \leq \mathcal{P} - \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor_{0,5} - \frac{1}{2} < \mathcal{P} - \frac{\mathcal{P}}{4} = \frac{3\mathcal{P}}{4}$

como $\mathcal{P} \in Z$ y $P_A = \lfloor P_A \rfloor_{0,5}$ porque L ó W par $\rightarrow \mathcal{P} - P_A \leq \left\lfloor \frac{3\mathcal{P}}{4} \right\rfloor_{0,5}$

$\frac{\mathcal{P} - P_A}{3} \leq \left\lfloor \frac{3\mathcal{P}}{4} \right\rfloor_{0,5} \Big/ 3$

Al menos un cuadrante con $\left\lfloor \frac{\mathcal{P} - P_A}{3} \right\rfloor_{0,5} \leq \left\lfloor \left\lfloor \frac{3\mathcal{P}}{4} \right\rfloor_{0,5} \Big/ 3 \right\rfloor_{0,5} = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor_{0,5}$

2) La pérdida en el segundo cuadrante B, $P_B \leq \frac{\mathcal{P}}{2}$.

Esta pérdida corresponde al lado izquierdo del pallet.

3) La pérdida en el tercer cuadrante C, $P_C \leq \left\lceil \frac{3}{4}\mathcal{P} - 1 \right\rceil + \frac{1}{4}$

Demostración:

Si $P_C > \left\lceil \frac{3}{4}\mathcal{P} - 1 \right\rceil + \frac{1}{4} \rightarrow P_C \geq \left\lceil \frac{3}{4}\mathcal{P} - 1 \right\rceil + \frac{1}{2}$

Luego, $\mathcal{P} - P_C \leq \mathcal{P} - \left\lceil \frac{3}{4}\mathcal{P} - 1 \right\rceil - \frac{1}{2} = \Delta$

Veamos el valor de Δ:

Sea $\mathcal{P} = 4a + \text{b}$ (b < 4), donde $a = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$

$3\mathcal{P} = 12a + 3b$

$\frac{3\mathcal{P}}{4} = 3a + \frac{3}{4}b \rightarrow \frac{3\mathcal{P}}{4} - 1 = 3a - 1 + \frac{3}{4}b \rightarrow \left\lceil \frac{3\mathcal{P}}{4} - 1 \right\rceil = 3a - 1 + \left\lceil \frac{3}{4}b \right\rceil$

$\left\lceil \frac{3\mathcal{P}}{4} - 1 \right\rceil + \frac{1}{2} = 3a - \frac{1}{2} + \left\lceil \frac{3}{4}b \right\rceil$

$\Delta = \mathcal{P} - \left(\left\lceil \frac{3\mathcal{P}}{4} - 1 \right\rceil + \frac{1}{2} \right) = (4a + b) - \left(3a - \frac{1}{2} + \left\lceil \frac{3}{4}b \right\rceil \right) =$

$$= a + \frac{1}{2} + \left(b - \left\lceil \frac{3}{4}b \right\rceil \right) \begin{cases} b - \left\lceil \frac{3}{4}b \right\rceil = 0 \rightarrow \text{si } b = 0 \\ b - \left\lceil \frac{3}{4}b \right\rceil = 0 \rightarrow \text{si } b = 1 \\ b - \left\lceil \frac{3}{4}b \right\rceil = 0 \rightarrow \text{si } b = 2 \\ b - \left\lceil \frac{3}{4}b \right\rceil = 0 \rightarrow \text{si } b = 3 \end{cases}$$

Luego, $\Delta = a + \frac{1}{2} = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor + \frac{1}{2}$ y $\mathcal{P} - P_C \leq \Delta = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor + \frac{1}{2}$

La pérdida en los otros 3 cuadrantes es menor o igual a $\left\lfloor \frac{\mathcal{P}}{4} \right\rfloor + \frac{1}{2}$

3.1) Si L y W par, la pérdida en C debe ser entera.

Por tanto, $\mathcal{P} - P_C \leq \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$

la pérdida en los otros 3 cuadrantes será como máximo $\left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$

Por simetría horizontal: C $\leftrightarrow$ D y A $\leftrightarrow$ B,

podemos obtener una solución válida con $P_A \leq \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$ y $P_C \leq \left\lceil \frac{3}{4}\mathcal{P} - 1 \right\rceil + \frac{1}{4}$

3.2) L impar, W par (puede haber una pérdida fraccionaria de 0.5 en C).
Si $P_C > \left\lceil \frac{3}{4}\mathcal{P} - 1 \right\rceil + \frac{1}{4}$, entonces $\mathcal{P} - P_C \leq \Delta = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor + \frac{1}{2}$
La pérdida en los otros 3 cuadrantes es menor o igual que $\left\lfloor \frac{\mathcal{P}}{4} \right\rfloor + \frac{1}{2}$.
Si $\mathcal{P} - P_C = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$ estamos en el caso (3.1).
Si $\mathcal{P} - P_C = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor + \frac{1}{2}$ hay al menos una pérdida de $\frac{1}{2}$
en el cuadrante A.
Por tanto, $P_B \leq \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$. Con una simetría horizontal obtenemos
una solución con $P_C \leq \left\lfloor \frac{3}{4}\mathcal{P} - 1 \right\rfloor + \frac{1}{4}$ satisfaciendo $P_A \leq \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$.

3.3) L par, W impar.
El mismo argumento del caso anterior, con una pérdida de $\frac{1}{2}$
en el cuadrante D.

3.4) L, W impar (puede haber una pérdida fraccionaria de 0.25 en C)
Si $P_C > \left\lceil \frac{3}{4}\mathcal{P} - 1 \right\rceil + \frac{1}{4}$, entonces $\mathcal{P} - P_C \leq \Delta = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor + \frac{1}{2}$
La pérdida en los otros 3 cuadrantes es menor o igual a $\left\lfloor \frac{\mathcal{P}}{4} \right\rfloor + \frac{1}{2}$
Si $\mathcal{P} - P_C = \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor + \frac{1}{2}$,
existe un cuadrante con una pérdida de $\frac{1}{2}$ en el cuadrante A ó D
o una pérdida de $\frac{1}{4}$ en el cuadrante A y $\frac{1}{4}$ en el cuadrante D.
En ambos casos, $P_B \leq \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$. Con una simetría horizontal, obtenemos
una solución con $P_C \leq \left\lceil \frac{3}{4}\mathcal{P} - 1 \right\rceil + \frac{1}{4}$ satisfaciendo $P_A \leq \left\lfloor \frac{\mathcal{P}}{4} \right\rfloor$

4) Obviamente en el último cuadrante D, $P_D \leq \mathcal{P}$.

Nos quedaría por lo tanto:

$P_B \leq \frac{\mathcal{P}}{2}$	$P_D \leq \mathcal{P}$
$P_A \leq \frac{\mathcal{P}}{4}$	$P_C \leq \left\lceil \frac{3}{4}\mathcal{P} - 1 \right\rceil + \frac{1}{4}$

Figura 3.18: *Límites de las pérdidas por cuadrante.*

Estas cotas sobre la pérdida en cada cuadrante se aplican cuando se añaden las aristas relativas a la pérdida que se han introducido en la subsección 3.4.2.1. Por ejemplo, si estamos considerando una pérdida directa entre cajas en el primer cuadrante A, basta que esta pérdida sea mayor que $\mathcal{P}/4$ para incluir una arista entre ellas.

Además, se ha de tener en cuenta que este tipo de relaciones no son las mismas en todos los miembros de una clase de equivalencia. El elemento de la clase de equivalencia en el que se alcance la menor cota del área va a tener más relaciones, ya que la pérdida en relación a la caja puede ser menor. Observemos el siguiente ejemplo:

a) (27,17,5,3)
   Cota: 30
   Solución Heurística: 29
   Pérdida: 9
   Pérdida/Área de la caja: $\frac{9}{15} = 0,6$

b) El mínimo de la clase de equivalencia es: (40007,24005,8001,4001)
   Cota: 30
   Solución Heurística: 29

Pérdida: 8005
Pérdida/Área de la caja: $\frac{8005}{8001*40001} = 0,00025$

En la instancia (27,17,5,3) tenemos que en la posición:

$(1*l+4*w, 1*l+2*w) = (17,11)$ puede colocarse una caja horizontal
$(3*l+3*w, 0*l+4*w) = (24,12)$ puede colocarse una caja vertical

Análogamente, en la instancia (40007,24005,8001,4001) en la posición

$(1*l+4*w, 1*l+2*w) = (24005,16003)$ puede colocarse una caja horizontal
$(3*l+3*w, 0*l+4*w) = (36006,16004)$ puede colocarse una caja vertical

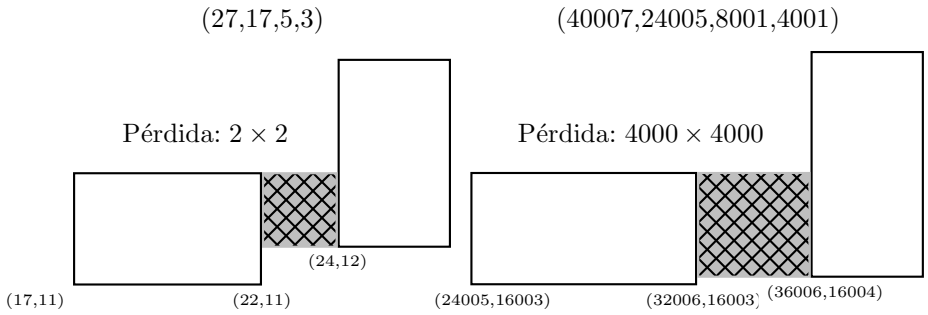
En el primer caso el par de cajas produce una pérdida de 4 que sería admisible en el cuadrante inferior-izquierda (Figura 3.19(a)). Pero en el segundo caso produce una pérdida de 4000*4000=16000000 que no sería admisible (Figura 3.19(b)).

En este ejemplo la solución del problema lineal de la primera instancia tenía 75 variables fraccionarias, y a partir de ella se podían definir 279 aristas, mientras que a partir de la solución de la segunda instancia se podían definir 341 aristas.

La instancia (40007,24005,8001,4001) exhibe estas buenas propiedades debido a que es el elemento de la clase con la menor cota superior. Su pérdida relativa es muy pequeña (0.0025) y por ello es posible detectar muchas parejas de cajas que están *mal colocadas* respecto a una solución óptima y añadir muchas aristas para eliminarlas. Las diferencias entre las pérdidas relativas de instancias de una misma clase de equivalencia pueden ser muy grandes. Por ello, cuando construimos el grafo de la solución, usamos la instancia con la menor cota superior, obtenida utilizando el procedimiento propuesto por Nelißen [66].

3.5. Desigualdades válidas

Dado el grafo $G_{LWlw} = (V, E)$, vamos a introducir brevemente las desigualdades válidas más importantes que se han estudiado para el problema del conjunto



(a) Menor proporción (b) Mayor proporción

Figura 3.19: *Las relaciones debidas a la pérdida dependen de la instancia.*

máximo independiente. Denotamos por P_{IS} al poliedro del conjunto máximo independiente.

3.5.1. Cliques

Dado $C \subseteq V$ un clique en G, cualquier conjunto independiente puede tener como máximo un vértice de C. Por lo tanto, la *desigualdad de clique*:

$$x(C) \leq 1 \tag{3.88}$$

es una desigualdad válida para P_{IS}. Padberg (1972)[69] demostró que cada desigualdad (3.88) es una faceta para P_{IS} si y sólo si C es un clique.

Puede demostrarse que si en el grafo G_{LWlw} sólo consideramos las aristas correspondientes a las relaciones de solapamiento, los únicos cliques son las restricciones de cubrimiento. Sin embargo, si al grafo se añaden las aristas correspondientes a la pérdida y dominancia, pueden existir otros cliques.

3.5.2. Ciclos impares

Un ciclo impar en G es un ciclo que contiene un número impar de vértices. Si H es un ciclo impar en G, entonces la *desigualdad del ciclo impar* para H es:

$$x(V(H)) \leq \lfloor |V(H)|/2 \rfloor. \tag{3.89}$$

Padberg (1972)[69] demostró que la desigualdad anterior es válida para P_{IS}, pero generalmente no define facetas. Para construir facetas parte de un *ciclo impar sin cuerdas (hole)*, en el que dos vértices no consecutivos del ciclo no son adyacentes. La desigualdad sobre el ciclo impar sin cuerdas (3.89) define una faceta para $P_{IS} \cap \{x | x_{V \setminus V(H)} = 0\}$.

Para conseguir una faceta para P_{IS}, Padberg propone un procedimiento de lifting secuencial añadiendo variables x_v de una en una con coeficientes β_v que se obtienen resolviendo el siguiente problema:

Sea A_G la matriz de incidencia del grafo G_{LWlw} y a_j la columna j de A_G. Definimos el conjunto:

$$T = \{j \in N \setminus S : a^j a^k > 0 \text{ para algún } k \in S\}, \tag{3.90}$$

es decir, T es el conjunto de todos los nodos que no son de H adyacentes con uno (o más) nodo(s) de H. Definimos:

$$T^q = T^{q-1} \cup \{j_q\} \text{ para } j_q \in T \setminus T^{q-1}, \tag{3.91}$$

y para $q = 1, \ldots, Q = |T|$, con la convención que $T^0 = \emptyset$. El conjunto T no requiere ningún tipo de orden.

Entonces, sea (Z_q) el siguiente problema:

$$(Z_q) \quad max \quad z_q = \sum_{k \in H} x_k + \sum_{k \in T^{q-1}} \beta_k x_k, \tag{3.92}$$

$$\text{s.t} \tag{3.93}$$

$$\sum_{k \in H \cup T^{q-1}} a^k x_k \leq e^G - a^{jq}, \tag{3.94}$$

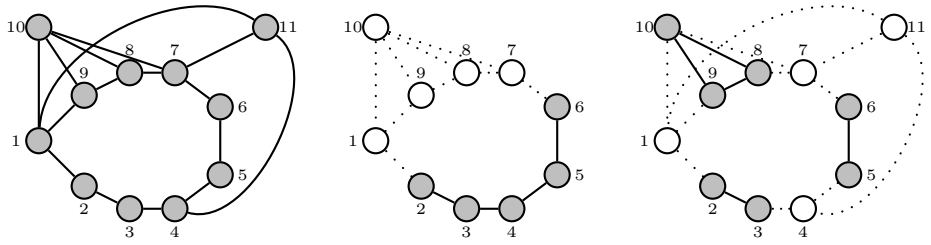
$$x_k = 0, 1 \; k \in H \cup T^{q-1} \tag{3.95}$$

donde los β_k son definidos recursivamente por: $\beta_{j_q} = s - \overline{z}_q$, y $s = \frac{1}{2}(|H| - 1)$, y $\overline{z}_q$ es el valor óptimo del problema (Z_q). Estos problemas son generalmente fáciles de resolver, pero pueden complicarse ya que en el fondo lo que hacen es resolver un problema de conjunto máximo independiente. Lo vemos mediante un ejemplo.

Ejemplo 3.1 *Supongamos el grafo G con 11 nodos como el de la Figura 3.20(a). Supongamos que elegimos el ciclo impar sin cuerdas formado por los nodos $H = \{1, \ldots, 9\}$. Por tanto, partiendo de la desigualdad $\sum_{1}^{11} x_i \leq 4$, para determinar los coeficientes de x_{10} y x_{11}, resolvemos $(Z_q), q = 1, 2$, con $T = \{10, 11\}$. El problema (Z_1) es elegir el coeficiente de lifting para la variable 10. Para ello resolvemos el problema (3.92)-(3.95) que no es otra cosa que eliminar los vértices adyacentes a la variable 10 y calcular el conjunto máximo independiente de los que quedan (Figura 3.20(b)). El resultado es $\overline{z}_1 = 3$ y por lo tanto $\beta_1 = 1$. De la misma forma para (Z_2) con $\overline{z}_2 = 3$ (Figura 3.20(c)) obtenemos $\beta_2 = 1$. Por lo tanto la desigualdad que se obtendría sería:*

$$\sum_{j \in S} x_j + x_{10} + x_{11} \leq 4 \tag{3.96}$$

En este ejemplo, una permutación de los elementos de T no produce una desigualdad diferente. Sin embargo, esto *no* es cierto en general.



(a) Grafo Original (b) Primera Variable (c) Segunda Variable

Figura 3.20: *Lifting de Padberg.*

3.5.3. Ruedas

Otra clase de desigualdades más recientes son las llamadas *ruedas* de Cheng y Cunningham (1997)[21]. Sea k un entero positivo y sea $G_1 = (V_1, E_1)$ un grafo con $V_1 = \{v_0, v_1, v_2, \ldots, v_{2k+1}\}$ y $E_1 = \{(v_0, v_i), (v_i, v_{i+1}) : 1 \leq i \leq 2k+1\}$. Tomamos $v_{2k+2} = v_1$. Considerar una subdivisión de G_1. Sean $P_{0,i}$ y $P_{i,i+1}$ los caminos desde (v_0, v_i) y (v_i, v_{i+1}) respectivamente a través de la subdivisión. Este grafo es una *1-rueda simple* de tamaño $2k+1$ si el ciclo C_i formado por $P_{0,i}, P_{i,i+1}, P_{0,i+1}$ es impar para cada i. Denotamos esta *1-rueda simple* por $W = W(v_0; v_1, v_2, \ldots, v_{2k+1})$. A v_0 le llamamos *centro (hub)*, a $P_{0,1}, P_{0,2}, \ldots, P_{0,2k+1}$ los *radios (spokes)*, a $P_{1,2}, P_{2,3}, \ldots, P_{2k,2k+1}$ los *segmentos de llanta (rim-paths)*, $v_1, v_2, \ldots, v_{2k+1}$ las *terminaciones de los radios (spoke-ends)* y el ciclo formado por $P_{1,2}, P_{2,3}, \ldots, P_{2k+1,1}$ la *llanta (rim)*. Si se particionan los vértices en dos conjuntos $\xi = \xi(W)$ y $O = O(W)$ donde $v_i \in \xi(O)$ si $P_{0,i}$ es un camino par (impar), se define también $S = S(W)$ el conjunto de vértices internos de los radios y $R = R(W)$ el conjunto de vértices internos de los segmentos de la llanta. Con este tipo de grafos conseguimos dos tipos de restricciones diferentes:

$$kx_0 + \sum_{i=1}^{2k+1} x_i + \sum_{v \in \xi} x_v + \sum_{v \in S \cup R} x_v \leq k + \frac{1}{2}(|S| + |R| + |\xi|) \quad (3.97)$$

$$(k+1)x_0 + \sum_{i=1}^{2k+1} x_i + \sum_{v \in O} x_v + \sum_{v \in S \cup R} x_v \leq k + \frac{1}{2}(|S| + |R| + |O| + 1) \quad (3.98)$$

En la Figura 3.21 tenemos un ejemplo de una 1-rueda simple de tamaño 13, donde a es el centro y $\{b, c, d, g, i\}$ es el conjunto de terminaciones de radios. En este subgrafo nos quedarán las siguientes restricciones:

$$2x_a + 2x_b + 2x_c + 2x_i + \sum_{v \notin \{a,b,c,i\}} x_v \leq 7$$

$$3x_a + 2x_d + 2x_g + \sum_{v \notin \{a,d,g\}} \leq 7$$

Para el cálculo de las desigualdades de rueda existen procedimientos basados en caminos más cortos entre pares de vértices, que aún siendo polinómicos son

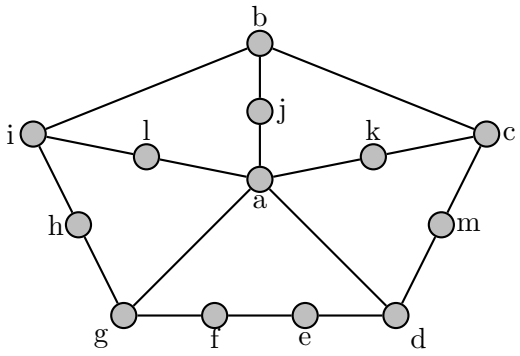


Figura 3.21: *Ejemplo de 1-rueda simple.*

bastante lentos. No utilizamos este tipo de algoritmos porque en la práctica no aparecen un gran número de grafos como el de la figura y el esfuerzo computacional es excesivo para el resultado obtenido.

3.5.4. Otras desigualdades válidas

Otras clases de desigualdades son los *antiholes* y *webs* de Balas y Padberg (1976)[6]), *abanicos* y *enrejados* de Cánovas et al. (2000)[23]. Sin embargo, no se conocen procedimientos eficientes de separación por lo que no se han incorporado al algoritmo.

3.5.5. Implementación del grafo

Cuando al resolver la relajación lineal en un nodo obtenemos una solución fraccionaria, procedemos a crear el grafo. Por razones de eficiencia computacional, no utilizamos todas las variables tales que $0 < x < 1$, sino sólo aquellas por encima de un umbral, es decir, aquellas tales que $0{,}005 < x < 0{,}995$. En cuanto a las aristas, se utilizan los 3 tipos de relaciones: solapamiento, pérdida y dominancia. Sobre este grafo se aplican los procedimientos de separación que se describen en los apartados siguientes.

3.5.6. Algoritmos de Separación

En la fase de separación, se intentan encontrar desigualdades globalmente válidas para el problema con dos objetivos fundamentales: intentar mejorar la cota superior para el problema y obtener soluciones enteras.

El procedimiento consta de dos ingredientes básicos:

i) Algoritmos para la detección de cliques violados.

ii) Algoritmos para la detección de ciclos violados y su correspondiente lifting.

Algoritmos similares han sido utilizados en la literatura por Nemhauser (1992)[68] y Hoffman y Padberg (1993)[51].

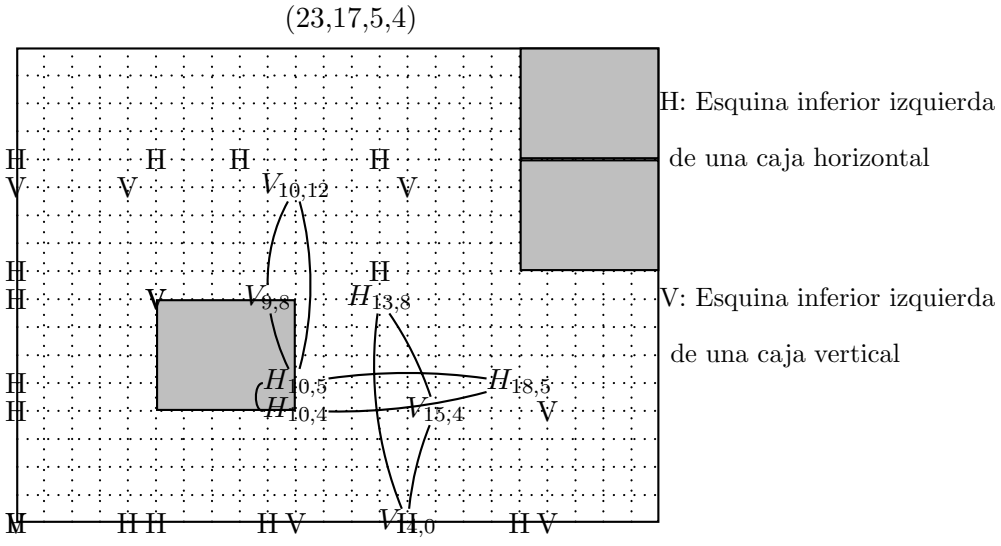


Figura 3.22: *Cliques con las nuevas relaciones.*

3.5.7. Cliques violados

Tal como se comentó anteriormente la existencia de nuevas relaciones debidas a la pérdida y la dominancia producen la aparición de nuevos cliques que no están incluidos en las restricciones de cubrimiento. La Figura 3.22 muestra una solución lineal del problema (23,17,5,4) con 32 variables fraccionarias de valor 0.5 y 3 enteras (de las 94 variables originales del problema), donde aparecen dibujados tres cliques violados. El primer clique es el formado por las variables $H_{10,5}$, $V_{9,8}$, $V_{10,12}$ donde las variables $H_{10,5} - V_{9,8}$ y $V_{9,8} - V_{10,12}$ están relacionadas por solapamiento y las variables $H_{10,5} - V_{10,12}$ están relacionadas por dominancia. El segundo clique violado es el $H_{10,4}$, $H_{10,5}$, $H_{18,5}$ formado por las variables $H_{10,4} - H_{10,5}$ relacionadas por solapamiento, las variables $H_{10,5} - H_{18,5}$ relacionadas por dominancia y $H_{10,4} - H_{18,5}$ relacionadas por pérdida, ya que la pérdida entre estas dos variables es 9, $\mathcal{P} = 11$ y en ese cuadrante la pérdida máxima $\lceil \frac{3}{4}\mathcal{P} - 1 \rceil + \frac{1}{4} = 8{,}25$. El tercero es $V_{14,0}$, $V_{15,4}$, $H_{13,8}$ con $V_{14,0} - V_{15,4}$, $V_{15,4} - H_{13,8}$ relacionadas por solapamiento y $V_{14,0} - H_{13,8}$ relacionadas por dominancia.

Para encontrar cliques violados, hemos trabajado con dos algoritmos de detección de cliques:

- *Algoritmo de Pardalos* (1994)[71].
  Construye un árbol de enumeración para encontrar el conjunto independiente de máxima cardinalidad en un grafo con m vértices. El procedimiento comienza ordenando los vértices de manera descendente de acuerdo a su grado de izquierda a derecha. La idea principal es el concepto de *profundidad*. En la profundidad 1 tenemos todos los vértices. Supongamos que consideramos el vértice x_k. En la profundidad 2, consideramos todos los vértices adyacentes a x_k y que se encuentren a la derecha de x_k en la profundidad 1 y así sucesivamente. Se recorren los vértices bajando en profundidad, teniendo en cuenta la siguiente consideración: sea x_{di}, el vértice i estudiado en la profundidad d. Si $d + (m - i) \leq Best$ (tamaño del mayor clique encontrado) entonces parar, dado que el tamaño del mayor clique en el que

aparezca x_{di} no superará $Best$. Si estamos en el conjunto de profundidad 1 y esta desigualdad se cumple, entonces paramos: hemos encontrado el clique maximal.

- *Algoritmo de Bron y Kerbosch* (versión extendida (2001)[17]).
  Está basado en un *Branch and Bound* simple enumerativo. En cada paso k un conjunto independiente, S_k, es aumentado por otro vértice elegido convenientemente para producir un S_{k+1}. Cuando no sea posible añadir más vértices, S_{k+1} será maximal. En el paso k sea Q_k el mayor conjunto de vértices para el que $S_k \cap Q_k = \emptyset$; esto es, cualquier vértice de Q_k añadido a S_k produce un $S_k + 1$ que es independiente. Q_k estará formado por dos tipos de vértices:

  a) Los vértices que ya han sido usados antes en la búsqueda para aumentar S_k que llamaremos Q_k^-.

  b) Los vértices que no han sido usados todavía en dicha búsqueda que llamaremos Q_k^+.

  En el paso $k+1$ elegimos un vértice $x_{ik} \in Q_k^+$. Dicho vértice se añade a S_k para producir $S_k + 1 = S_k \cup \{x_i k\}$, creando los conjuntos:

$$Q_{k+1}^- = Q_k^- - \Gamma(x_{ik}) \text{ y } Q_{k+1}^- = Q_k^- - \Gamma(x_{ik}) - \{x_{ik}\} \tag{3.99}$$

  Se cumple entonces el siguiente teorema: "La condición necesaria y suficiente para que S_k sea un conjunto independiente maximal es que $Q_k^+ = Q_k^- = \emptyset$." Por otro lado, la condición $\exists x \in Q_k^- / \Gamma(x) \cup Q_k^+ = \emptyset$ es suficiente para afirmar que x no puede generar un conjunto independiente maximal a partir de S_k. En estas condiciones el algoritmo que genera todos los conjuntos independientes maximales de un grafo G puede observarse en la Figura 3.23.

De los dos algoritmos que hemos probado nos quedamos con la versión extendida de Bron y Kerbosch que consigue todos los cliques de un grafo. El algoritmo de Pardalos computacionalmente es más rápido si sólo queremos encontrar los cliques de cardinalidad máxima, pero debido a la estructura particular del grafo elegimos el algoritmo de Bron y Kerbosch.

El esquema que seguimos, a partir de la propuesta por Hoffman y Padberg (1993)[51], es el siguiente, en el que para vértices con orden muy alto (en nuestro caso, superior a 25) se opta por un procedimiento greedy rápido:

Paso 1. Elegir $v \in V$ de menor orden $d(v)$.

Paso 2. Si $d(v) = 0$, *parar*.

Paso 3.
- Si $v \cup \{star(v)\}$ forman un clique:
  Si la suma de sus variables es mayor que 1,
  añadir la restricción. Borrar v.
- Si no, ir a *Paso 4*.

Paso 4.
- Si tenemos $d(v) \geq 25$. Procedimiento constructivo *greedy* que intenta encontrar el conjunto máximo independiente de máximo peso.
  Si la suma de las variables es mayor que 1,
  incluir la restricción.
  Borrar v e ir a *Paso 1*.
- Si $d(v) < 25$, utilizar el procedimiento que busca todos los conjuntos independientes con el algoritmo de Bron y Kerbosch.
  Cada conjunto independiente con valor mayor que 1 define una restricción.
  Borrar v e ir al *Paso 1*.

Paso 1. *Iniciación*, $S_0 = Q_0^- = \emptyset : Q_0^+ = X : k = 0$

Paso 2. *Aumentar conjunto*:
Sea $x_{ik} \in Q_k^+$
$S_{k+1} = S_k \cup \{x_{ik}\}$
$Q_{k+1}^- = Q_k^- - \Gamma(x_{ik})$
$Q_{k+1}^+ = Q_k^+ - \Gamma(x_{ik}) - \{x_i k\}$
Guardamos Q_k^+ y Q_k^-. Hacer $k = k + 1$

Paso 3. *Test*.
Si $\exists x \in Q_k^- / \Gamma(x) \cup Q_k^+ = \emptyset$ ir al *Paso 5*.
En otro caso ir al *Paso 4*.

Paso 4. Si $Q_k^+ = Q_k^- = \emptyset$ se imprime S_k
(conjunto independiente maximal).
Ir a *Paso 5*.
Si $Q_k^+ = \emptyset$ pero $Q_k^- \neq \emptyset$ ir a *Paso 5*.
En caso contrario, ir a *Paso 2*.

Paso 5. *Backtrack*. Si $k = 0$, *parar*.
Hacer $k = k - 1$. Se elimina x_{ik} de S_{k+1} y se obtiene S_k.
Recuperamos Q_k^+ y Q_k^-.
Eliminamos x_{ik} de Q_k^+ y lo añadimos a Q_k^-.
Si $k = 0$ y $Q_k^+ = \emptyset$, *Parar*.
En otro caso ir al *Paso 3*.

Figura 3.23: *Algoritmo de Bron y Kerbosch.*

3.5.8. Ciclos violados y su correspondiente lifting

Como hemos mencionado anteriormente, la separación de los ciclos impares consta de dos partes. La primera parte está formada por algoritmos para encontrar ciclos impares violados o "casi violados" (ciclos en los que el valor de las variables está muy próximo al valor del término independiente) a partir de la solución fraccionaria. La segunda parte consiste en algoritmos de lifting para incorporar nuevas variables a los ciclos impares que hemos encontrado en la primera parte y de esta manera hacer más fuerte las nuevas restricciones.

Una característica de este problema es que, excepto en las primeras iteraciones, no aparecen ciclos que se violen directamente y puedan originar una nueva restricción. Es muy frecuente que los ciclos estén "casi violados" y sólo originan nuevas restricciones después de ser liftados. Por ello no se han implementado algoritmos de identificación de ciclos violados, como los que utilizan Nemhauser y Sigismondi (1992)[68].

3.5.8.1. Ciclos violados

En primer lugar, notar que nuestro grafo puede ser no conexo, por lo que previamente separamos el grafo en sus componentes conexas y para cada componente buscamos ciclos violados.
Hemos probado dos tipos diferentes de algoritmos:

- *Algoritmo del árbol generador de mínimo peso.*
  En primer lugar, creamos el árbol generador de mínimo peso del grafo G_{LWlw} y el co-árbol. Posteriormente los ciclos son generados incluyendo aristas del co-árbol al árbol generador de mínimo peso y nos quedamos los de cardinalidad impar.

- *Algoritmo de búsqueda de ciclos sin cuerdas* (Hoffman y Padberg (1993)[51]).
  El procedimientos elige un vértice $v \in V$ que llamamos nodo raíz y construye un grafo "por niveles" empezando por el raíz. Cada nivel del nuevo grafo

es definido por el número de arcos de distancia al raíz. Por tanto, todos los vecinos de v están en el nivel 1, los vecinos de los vecinos excepto v y los que estuvieran en el nivel anterior estarán en el nivel 2, y así en adelante. Cualquier camino más corto del nivel k al raíz v contiene exactamente k arcos. Cualquier par de nodos adyacentes del nivel k y que tengan caminos más cortos disjuntos al raíz, forman un ciclo impar sin cuerdas que contiene a v.

Para construir el "grafo por niveles" se construye el grafo iterativamente nivel por nivel asignando pesos de $1 - x_v - x_{v'}$ a los arcos (v, v') que estén en el grafo por niveles. Supongamos que estamos en el nivel $k \geq 2$ de la construcción del grafo y sea u y z son cualquier par de nodos en dicho nivel. Encontramos el camino más corto de u al nodo raíz v y "bloqueamos" todos los vecinos del camino más corto asignándoles a los arcos un peso muy grande M. En el grafo restante buscamos el camino más corto de z a v que utiliza sólo nodos que estén en niveles más pequeños que k. Si existe un camino más corto de longitud menor que M, tenemos por tanto un ciclo impar sin cuerdas.

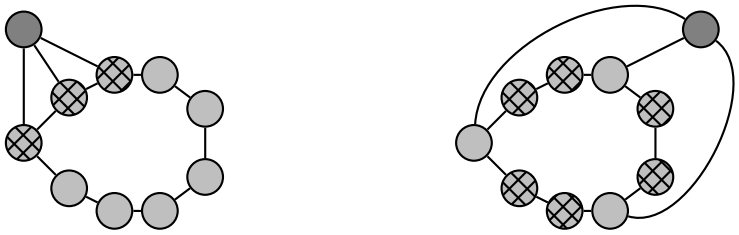
3.5.8.2. Procedimiento de lifting secuencial

En la literatura se han creado diferentes algoritmos de lifting secuencial para liftar los ciclos impares (Padberg (1972)[69] y Wolsey (1975)[89],[88]) basados en árboles de lifting. Nuestro procedimiento también está basado en un árbol de lifting secuencial, sólo para variables que estén en el grafo de la solución actual. Posteriormente las variables que no estén en el grafo se toman por orden y se van liftando en la restricción de acuerdo al procedimiento que explicamos a continuación.

En la mayoría de ocasiones en la literatura se ha trabajado con algoritmos de conjunto máximo independiente para liftar las variables. Nosotros buscábamos un procedimiento rápido, por lo que hemos evitado el uso de este tipo de algoritmos.

Comenzamos definiendo el conjunto de variables candidatas al lifting, es decir, variables que individualmente pueden ser incorporadas a la restricción del ciclo, aunque no tienen por qué poder entrar posteriormente en la restricción debido a la incorporación anterior de otras variables.

El conjunto de variables candidatas a liftar estará formado por variables adyacentes a tres o más vértices consecutivos del ciclo (Figura 3.24(a)) y variables que dejen *cadenas* pares (Figura 3.24(b)). Definimos como *cadenas* a los segmentos conexos de ciclo si eliminamos los vértices adyacentes a la variable y no tenemos en cuenta las cuerdas.



(a) Tres o más adyacentes consecutivos (b) Cadenas pares

Figura 3.24: *Vértices candidatos para liftar.*

Estudiamos dos posibilidades: elegir como candidatas los dos tipos de variables o sólo quedarnos con las que fueran consecutivas con tres o más vértices, que serán mucho más fáciles de liftar posteriormente. Implementamos los dos casos, pero el esfuerzo computacional para liftar los vértices candidatos del segundo tipo era mucho mayor, por lo que nos quedamos sólo con las variables que tienen tres o más vértices consecutivos adyacentes al ciclo.

Una vez determinado el conjunto de variables candidatas, comienza el proceso de lifting secuencial, en el que denotamos por $N_C(v)$ el conjunto de los vértices de C adyacentes a v. El algoritmo para liftar las diferentes variables fraccionarias e incluirlas en el ciclo aparece a continuación:

Algoritmo de lifting secuencial

Paso 1. *Inicialización.*

$p = 0$.

$E = \emptyset$. Conjunto de variables liftadas en el nodo.

L=Lista de candidatas a liftar.

Aumento=0.

Paso 2. *Elección de la variable a liftar.*

Si $L = \emptyset$, ir a *Paso* 4 (*backtrack*).

Si no, tomar la primera variable $x_k \in L$, hacer $L = L - \{x_k\}$

Paso 3. *Test de la variable* x_k.

$$\text{Si, para todo } x_v \in E \begin{cases} x_v \text{ y } x_k \text{ son adyacentes} \\ \text{ó} \\ |N_C(k) \setminus N_C(v)| \geq 2 \end{cases}$$

Ir a *Paso* 3 (*branching*), *Aumento*=1.

Si no, ir a *Paso* 1.

Paso 4. *Branching* (la variable x_k entra en la restricción).

$p = p + 1$

$E = E \cup \{x_k\}$

Ir a *Paso* 2.

Paso 5. *Backtrack.*

Si *Aumento* es igual a 1, guardar la restricción (conjunto E).

Hacer *Aumento* $= 0$.

$p = p - 1$

Si $p < 0 \rightarrow$ parar (enumeración completa).

Si no, $E = E - \{x_r\}$, x_r es la última variable de E.

$L = \{x_{r+1}, \ldots, x_n\}$

Ir a *Paso* 1.

La Figura 3.25 muestra las dos posibilidades para liftar un vértice. Si las variables x_1 y x_2 ya han sido incluidos en la restricción y la variable x_3 está siendo estudiada, será incluida también ya que $|N_C(x_3) \setminus N_C(x_1)| \geq 2$ y x_3 es adyacente a x_2.

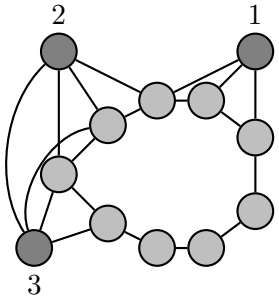


Figura 3.25: *Liftado secuencial de vértices.*

3.6. Utilizar la solución del LP para obtener soluciones posibles

Durante el procedimiento *branch and cut* resolvemos una gran cantidad de problemas lineales y obtenemos soluciones fraccionarias. Cada vez que tenemos una solución fraccionaria intentamos conseguir una solución entera mediante la utilización de alguna técnica de redondeo. En algunas ocasiones este proceso mejora la mejor solución posible conocida. Suponemos que tenemos una solución $x \in P_{IS}$ con alguna componente fraccionaria. Hemos estudiado varias estrategias de redondeo, que describimos a continuación.

3.6.1. Redondeo simple determinista y no determinista

Sea S el conjunto de nodos que corresponden a variables con valor mayor que un medio, $S = \{v \in V | x_v > \frac{1}{2}\}$. La forma más sencilla para conseguir una solución sería hacer x_S a 1, y $x_{V \setminus S}$ a 0, obteniendo una solución entera. La calidad de la solución de este redondeo puede ser muy pobre. Por ejemplo, si tenemos

una solución fraccionaria donde todas las variables distintas de 0 tienen valor $\frac{1}{2}$ con este redondeo tendríamos una solución nula.

Otro tipo de redondeo es redondear a 1 la variable de mayor valor fraccionario. Se ordena el conjunto de variables con valor fraccionario S' en orden decreciente. La de mayor valor fraccionario se hace 1 y se eliminan todas sus vecinos de S'. El proceso continúa hasta que no quedan variables por redondear a 1. Este método tiene el inconveniente de que si la solución fraccionaria tiene valores muy parecidos la elección de hacer variables iguales a 1 ó 0 es casi aleatoria.

Los métodos de redondeo anteriores se probaron también de forma no determinista.

- Crear una lista de candidatos a redondear a 1.
- Elegir de forma aleatoria uno de la lista de candidatos.
- Actualizar el conjunto de variables a redondear.

Dado que los procedimientos de redondeo simples necesitan muy poco tiempo de calculo, se repite el proceso un número determinado de veces.

Otro método no determinista es el propuesto por Bertsimas y Vohra (1998)[14], en el que se da a cada variable no fraccionaria una probabilidad de ser redondeada a 1:

$$P\{x_j = 1\} = f(x_j^*) = 1 - (1 - x_j^*)^k, k = log|V| \tag{3.100}$$

3.6.2. Redondeo del mínimo pesar

Strijk et al. (2000)[81] describen un procedimiento de redondeo para el problema del etiquetado de mapas que reducen a un problema de conjunto máximo independiente. Proponen un nuevo procedimiento de redondeo llamado *minimum regret rounding*. Sea $\mathcal{I}$ la colección de todos los conjuntos independientes en G_{LWlw}. El algoritmo redondea eligiendo repetidamente un conjunto independiente I$\in \mathcal{I}$ con $0 < x_I < 1$, redondeando x_I a 1 y $x_{N(I)}$ a 0, hasta que x sea entera.

$N(I)$ denota el conjunto de vecinos de I. Esta operación de redondeo define una función f que transforma x en un vector $x' \in P_E$ usando I:

$$f : P_E \times I \to P_E : (x, I) \to x', \qquad \text{donde } x'_u = \begin{cases} 1, & \text{si } u \in I; \\ 0, & \text{si } u \in N(I); \\ x_u, & \text{en otro caso.} \end{cases} \tag{3.101}$$

Definimos una función que llamamos *regret* que es $r(x, I) = x(N_1(I)) - |I|$. Esta función tiene la propiedad siguiente: sea $x \in P_E, I \in I$ con $|I| > 1$ y elegimos dos conjuntos no vacíos $I_1, I_2 \subset I$ donde $I_1 = I \setminus I_2$. Entonces:

$$\begin{aligned} r(x, I) &= x(N_1(I)) - |I| \\ &= (x(N_1(I_2)) + x(N_1(I_2))) - (x(N_1(I_1)) \cap x(N_1(I_2))) - |I_1| - |I_2| \\ &= r(x, I_1) + r(x, I_2) - x(N_1(I_1) \cap N_1(I_2)) \end{aligned}$$

Por lo que si:

$$x(N_1(I_1) \cap N_1(I_2)) = 0$$

el "regret"(pesar) de redondear x_I a 1 es el mismo que el combinado de redondear x_{I_1} y x_{I_2} a 1. Esta condición se satisface siempre que tengamos un grafo conexo. Como hemos mencionado los grafos producidos por la solución fraccionaria pueden no ser conexos, con lo que deberíamos hacer la función para cada componente conexa de la solución fraccionaria.

A partir de la función "regret"se define un algoritmo greedy de redondeo. El algoritmo tiene como entrada una solución fraccionaria $x \in P_E$ y un entero $t > 0$ y reemplazamos x por $f(x, I)$ para algún conjunto I, redondeando x_I a 1. Lo hacemos en t fases, numeradas $t, t-1, \ldots, 1$. En la fase k, se trabaja con conjuntos $I \in I$ satisfaciendo:

$$|I| = k, \ 0 < x_I < 1, \ \text{y } G_{supp[x_{N_1(I)}]} \text{ es conexo} \tag{3.103}$$

Durante la fase k, el conjunto $I \in \mathcal{I}$ es el que minimice el "regret"$r(x, I)$ dentro de estas condiciones. La fase k termina cuando no hay más conjuntos $I \in \mathcal{I}$ satisfaciendo estas condiciones. Hemos implementado el algoritmo del "minimum regret" para t=1 y t=2.

Una modificación del algoritmo para nuestro problema consiste en tener en cuenta también la pérdida, es decir, no fijar dos variables a 1 que produzcan una pérdida mayor de la que consideremos "*aceptable*" para intentar conseguir una solución óptima. Definimos una pérdida como aceptable si es menor que la mitad de la pérdida permitida en una solución entera de valor igual a la cota superior.

3.7. Ramificación

En esta sección, estudiamos las diferentes formas de ramificación para el algoritmo.

3.7.1. Ramificación por Variables

Se elige la variable fraccionaria que esté más cercana a 0.5 como *variable de ramificación* y se crean dos nuevos nodos. En la rama de la izquierda la variable de ramificación se fija a 1, en la de la derecha a 0.

3.7.2. Ramificación por Restricciones

Se elige la restricción de cubrimiento cuya holgura sea más cercana a 0.5 como *restricción de ramificación*. Se crean dos nuevos nodos. En la rama de la derecha la restricción de ramificación se fija a 0, en la de la izquierda se fija a 1.

Puede ocurrir que no se pueda elegir ninguna restricción para ramificar, ya que las restricciones que elegimos para ramificar deben tener holgura mayor que 0 y holgura menor que 1. Esto es debido a que existen soluciones fraccionarias con todos las holguras iguales a 1 ó 0. Si ocurriera este caso seguiríamos a lo largo de esa rama ramificando por variables.

3.7.3. Otras estrategias de ramificación

Hemos considerado también el esquema de ramificación propuesto por Rossi y Smriglio (2001)[75] en su trabajo para el conjunto máximo independiente. En un nodo t del árbol de búsqueda las restricciones de ramificación fijan un conjunto de $U_t \subseteq V$ ($L_t \subseteq V$) de variables a 1 (a 0). Claramente, $N(U_t) \subseteq L$. Por tanto nos queda un problema en el subgrafo G_t inducido por el conjunto $V_t = V \setminus (U_t \cup L_t)$. Sea $\bar{\alpha}$ la mejor solución conocida hasta el momento, sea $\alpha(G)$ la cardinalidad del conjunto máximo independiente del grafo G y sea $\bar{\alpha}_t = \bar{\alpha} - |U_t|$. Si podemos certificar que $\alpha(G_t) \leq \bar{\alpha}_t$ el nodo puede ser saturado. La regla de ramificación es la siguiente: consideremos un conjunto $W_t \subseteq V_t$ de vértices para los cuales es posible probar que

$$\alpha(G_t[W_t]) \leq \bar{\alpha}_t \tag{3.104}$$

y sea $Z_t = V_t \setminus W_t$. Entonces si $\alpha(G_t) > \bar{\alpha}_t$, cada conjunto de máxima cardinalidad de G_t debe contener, al menos, un vértice en Z_t. Cada $v_i \in Z_t$ define una rama. Por lo tanto, la idea es tener un conjunto W_t lo mayor posible. Rossi y Smriglio proponen un procedimiento para construir W_t y dan referencias de otras propuestas para obtener Z_t como la de Balas y Yu (1986)[7].

Hemos intentado diversos procedimientos para adaptar la propuesta de Rossi y Smriglio a nuestro problema, pero en todos los casos los árboles de búsqueda resultaron excesivamente grandes en comparación con los obtenidos con las estrategias de ramificación por variables o por restricciones. Por tanto, en la implementación de nuestros algoritmos no hemos considerado esta alternativa de ramificación.

3.8. Fijar variables por implicaciones lógicas

Dependiendo de la estrategia de ramificación que hayamos elegido, podemos fijar algunas variables o restricciones de manera local para el resto del sub-árbol de ramificación.

3.8.1. Ramificación por variables

Supongamos que elegimos la variable j para ramificar. En la rama en la que la variable j se fija a 1, podemos fijar todas las variables adyacentes a j en el grafo al valor 0. Debido a las nuevas relaciones definidas entre variables se pueden fijar un gran número de variables a 0.

3.8.2. Ramificación por restricciones de cubrimiento

Supongamos que la solución que buscamos tiene una pérdida de como máximo $\mathcal{P}$ y supongamos un nodo cuya restricción de cubrimiento se fija a 0, por lo que tenemos ya una pérdida p en los descendientes. Entonces, todas las restricciones que produzcan una pérdida mayor que $\mathcal{P} - p$ podemos igualarlas a 1 ya que si ramificamos posteriormente sobre ellas en esa rama nos daría una solución imposible. Esta idea puede ser refinada al aplicarla a los distintos cuadrantes del pallet ya que sabemos cuál es la máxima pérdida admisible en cada cuadrante.

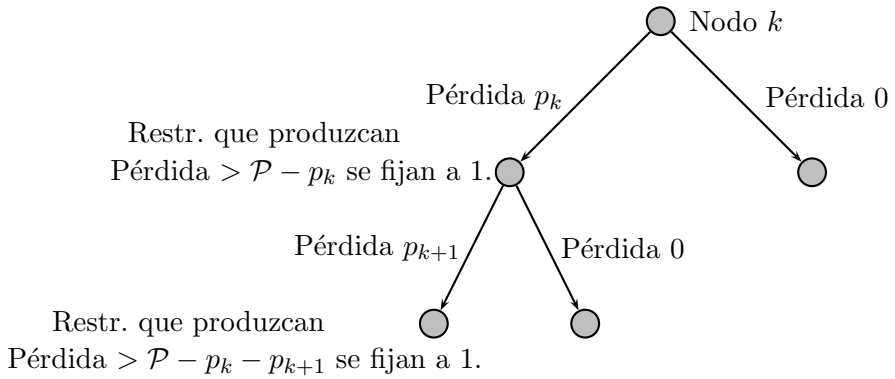


Figura 3.26: *Añadimos información a la ramificación por restricciones.*

3.9. Implementación y resultados computacionales

Para la implementación de nuestro procedimiento *Branch and Cut* hemos utilizado un software basado en teoría de combinatoria poliédrica. Este programa denominado *"ABACUS (A Branch And Cut System)"* fue desarrollado por Stefan Thienel (1995)[84] para la implementación de algoritmos *Branch and Bound* que usan la relajación lineal de problemas enteros y que se pueden complementar con generación dinámica de planos de corte (*Branch and Cut*), generación de columnas (*Branch and Price*) o la combinación de ambas (*Branch and Cut and Price*). Para una revisión de algunos trabajos realizados con este software consultar Elf et al. (2001)[39]. ABACUS proporciona un sistema de clases bases abstractas en *C++* de las cuales se pueden derivar las clases para la implementación de un problema específico.

Los algoritmos han sido codificados en *C++* y ejecutados sobre un ordenador personal con un procesador *Pentium III* a *800Mhz*. Esta sección describe los problemas test, el valor de los parámetros y estrategias usadas en cada algoritmo. Todos los problemas test están disponibles y pueden ser obtenidos solicitándolos al autor de este trabajo.

Para comparar el *Branch and Cut* hemos utilizado las soluciones ofrecidas por el *CPLEX 7.0*, uno de los más potentes paquetes de software para resolver problemas lineales y enteros. No se ha podido comparar con otros algoritmos debido a que el resto de los algoritmos exactos no se habían probado para los problemas de *Tipo II*.

Los problemas test con los que vamos a trabajar son las distintas clases de equivalencia de *Tipo I* y de *Tipo II*. Hemos dividido los resultados computacionales en dos conjuntos dependiendo del Tipo.

3.9.1. Problemas hasta 50 cajas. *Tipo I*

El conjunto de problemas de *Tipo I* contiene 7827 clases de equivalencia diferentes, como hemos mencionado en la sección 2.2. Hemos trabajado con estos problemas que no son especialmente difíciles (ya que han sido resueltos previamente por otros procedimientos) para tener una idea de qué estrategias eran mejores para luego abordar los problemas más grandes. No era el propósito de este estudio mejorar los métodos existentes con estos problemas, aunque los resultados computacionales muestran que nuestro método también es eficaz para estos problemas.

Para este conjunto de problemas hemos utilizado el heurístico de cinco bloques y la cota del área. Para las tablas sólo nos hemos quedado con 499 clases de equivalencia, ya que para el resto:

- En 6376 clases de equivalencia, la solución del heurístico era igual a la cota, con lo que no había que resolver ningún lineal.
- De los restantes, en 592 la primera solución lineal permitía obtener la solución óptima, bien porque el valor redondeado de la solución era igual al del heurístico o bien porque la solución era entera.

De los procedimientos de redondeo hemos elegido el "minimum regret rounding" ya que fue el que experimentalmente obtuvo mejores resultados. La forma de ramificar elegida es por variables. No hay diferencias significativas con la ramificación por restricciones debido a que el tamaño de los árboles no es muy grande.

En las tablas se muestra la comparación en términos de número de nodos evaluados (# nodos) y en tiempo total de CPU (Tiempo).

La Tabla 3.3 muestra el comportamiento de los diferentes heurísticos de separación, utilizados únicamente en el nodo raíz:

- *Cliques*: sólo buscando desigualdades definidas por cliques.

- *Ciclos*: sólo buscando ciclos mediante el árbol generador de mínimo peso y liftándolos.

- *Holes*: buscando ciclos mediante el algoritmo para buscar ciclos impares sin cuerdas y liftándolos.

- *Fast*: buscando desigualdades definidas por cliques y por ciclos con el árbol generador de mínimo peso, pero sólo liftando los ciclos *prometedores*, ciclos en los que el valor de las variables antes de liftar es el 95 % del valor del término independiente.

		Cliques	*Ciclos*	*Holes*	*Fast*
Tiempo	*Máximo*	31,04	835,03	15188,22	104,9
	Des.Típica	4,2	71,74	905,61	12,85
	Media	3,14	28,34	179,91	8,45
#Nodos	*Máximo*	33	17	165	35
	Des.Típica		3,56 2,27	7,82	3,04
	Media	3,77	1,93	2,17	2,45
Resueltos en nodo raíz		209	399	422	349

Tabla 3.3: Tipo I. *Diferentes desigualdades.*

En la Tabla 3.3 se observa que buscar todos los ciclos parece no ser una estrategia muy aconsejable debido al gran esfuerzo computacional que conlleva. De la misma forma buscar ciclos con el algoritmo de búsqueda de *holes* requiere un esfuerzo computacional muy grande en relación al número de problemas que se resuelven en el nodo raíz. Buscar cliques es muy rápido, por lo que nos quedamos con una estrategia mixta (*Fast*) que consiste en buscar todos los cliques y buscar ciclos con el árbol generador de mínimo peso, pero liftando sólo los prometedores como hemos mencionado anteriormente.

Con la mejor estrategia para los algoritmos de separación, que parece ser la *Fast*, estudiamos la influencia de los diferentes tipos de relaciones, que se muestra en la Tabla 3.4:

- *Solapamiento*: sólo utilizamos las relaciones de solapamiento.
- *Dominancia*: utilizamos las relaciones de solapamiento más las relaciones de dominancia.
- *Pérdida*: utilizamos las relaciones de solapamiento más las relaciones debidas a la pérdida.
- *Todas*: utilizamos los tres tipos de relaciones.

		Solapamiento	*Dominancia*	*Pérdida*	*Todas*
Tiempo	*Máximo*	16,09	98,7	71,4	104,9
	Des.Típica	1,63	8,92	6,37	12,85
	Media	1,38	5,48	4,48	8,45
#Nodos	*Máximo*	25	25	27	35
	Des.Típica	3,7	3,6	3,6	3,04
	Media	4,18	3,41	3,38	2,45
Resueltos en nodo raíz		185	271	257	349

Tabla 3.4: Tipo I. *Diferentes relaciones.*

Como observamos en la Tabla 3.4, el esfuerzo computacional se incrementa cuando añadimos nuevos tipos de aristas. Si estuviéramos buscando un algoritmo eficiente para los problemas de *Tipo I* no deberíamos incluir las nuevas aristas. Sin embargo, la segunda mitad de la Tabla, y en particular su última línea, muestra que utilizando estas nuevas aristas se resuelven más problemas sin necesidad de ramificar o con árboles de búsqueda que tienden a ser menores por término medio, lo que pensamos que será muy útil para los problemas más grandes.

La Tabla 3.5 presenta los resultados para diferentes estrategias algorítmicas, utilizando todas las relaciones y sólo liftando los ciclos prometedores y los cliques:

- *Branch&Cut*: los heurísticos de separación son invocados en cada nodo del árbol de búsqueda mientras encuentren desigualdades violadas.

- *B&Cut2Iter*: los heurísticos de separación son utilizados en cada nodo del árbol de búsqueda, pero un máximo de dos iteraciones por nodo.

- *CuttingPlanes*: es un algoritmo de planos de corte, sólo se buscan desigualdades violadas en el nodo raíz.

		Branch&Cut	*B&Cut2iter*	*CuttingPlanes*
Tiempo	*Máximo*	344,83	137,1	104,9
	Des.Típica	27,16	8,3	12,85
	Media	12,07	4,73	8,45
#Nodos	*Máximo*	27	29	35
	Des.Típica	2,09	2,19	3,04
	Media	1,95	2,51	2,45
Resueltos en nodo raíz		349	259	349

Tabla 3.5: Tipo I. *Diferentes algoritmos.*

Parece eficiente limitar el número de veces que llamamos a los procedimientos de separación en cada nodo, o limitar su uso al nodo raíz. Según los resultados obtenidos parece ser aconsejable probar *CuttingPlanes* (Planos de corte) y *B&Cut2Iter* (*Branch and Cut* limitando el número máximo de iteraciones por nodo a 2) en los problemas más grandes, ya que son las técnicas con las que hemos obtenido mejores resultados.

3.9.2. Problemas hasta 100 cajas. *Tipo II*

El conjunto de *Tipo II* formado por 40609 clases de equivalencia no había sido anteriormente estudiado de manera completa, ni por técnicas heurísticas, ni por métodos exactos. Tal como hicimos para el conjunto de *Tipo I*, en primer lugar separamos los problemas fáciles, aquellos para los que el algoritmo de cinco bloques coincide con la cota superior del área, lo que nos deja con 10764 problemas. En un segundo paso, resolvemos estos problemas con un heurístico más complejo, el algoritmo *G4*, y la cota superior de Nelißen y nos quedan 2433 problemas donde la solución proporcionada por el heurístico no puede probarse que sea la solución óptima.

Entonces aplicamos nuestro algoritmo a estos 2433 problemas más difíciles. De ellos 2192 son resueltos por la formulación lineal, bien porque el valor redondeado de la solución lineal es igual al proporcionado por el heurístico, bien porque obtiene una solución entera. Quedan, por tanto, 241 problemas realmente difíciles para los que las fases de planos de corte y/o ramificación son necesarias.

El algoritmo de redondeo para estos problemas es el mismo que para los de *Tipo I*, el "minimum regret rounding". Para la estrategia de ramificación hemos utilizado la ramificación por restricciones, ya que los primeros resultados sobre un subconjunto de problemas mostraron que se reducían de forma considerable los árboles de ramificación.

La Tabla 3.6 contiene los tiempos de computación requeridos para los 241 problemas difíciles por los algoritmos *B&C2iter*, *CuttingPlanes* y *CPLEX 7.0*. La Figura 3.27 muestra una comparación directa entre *B&C2iter* y *CPLEX 7.0*. Ambos algoritmos comparten la misma formulación descrita en la sección 3.3.4 y la diferencia corresponde al efecto de los procedimientos específicos de separación de nuestro algoritmo con respecto a las estrategias, potentes pero de propósito general, incluidas en el CPLEX. La última fila de la Tabla 3.6 muestra que quedan todavía algunos problemas muy difíciles que necesitan más de 54000 segundos de tiempo de computación. Ha sido necesario imponer un límite porque pruebas

computacionales preliminares mostraron que en casos aislados el tiempo requerido puede ser extremadamente largo. Por ejemplo, CPLEX necesitó 1423508 segundos para resolver la instancia (89,75,10,7). No es posible calcular medias para los tiempos de computación y número de nodos ya que no se han resuelto todos los problemas. Además, estas medias quedarían muy distorsionadas por la existencia de unos cuantos valores extremadamente altos. En cambio, una medida que sí puede ser calculada y proporciona una estimación robusta de los valores centrales del tiempo de computación y el número de nodos es la mediana, que aparece en la Tabla 3.7.

	B&C2Iter	*CuttingPlanes*	*CPLEX*
Menor que 10 Minutos	137	127	85
Entre 10 m y 1 hr	57	77	84
Entre 1 hr y 2 hr	18	10	24
Entre 2 hr y 5 hr	15	15	16
Entre 5 hr y 10 hr	8	5	12
Entre 10 hr y 15 hr	1	0	1
Más de 15 hr.	5	7	19

Tabla 3.6: Tipo II. *Comparación de algoritmos.*

	B&C2iter	*CuttingPlanes*	*CPLEX*
Tiempo	436,44	506,91	1268,62
Nodos	15	21	82

Tabla 3.7: Tipo II. *Medianas para los algoritmos.*

Si el límite de 54000 segundos de CPU (15 horas) parece excesivamente alto para aplicaciones prácticas la Tabla 3.8 muestra el número de problemas no resueltos para algunos límites superiores. En general, *no resueltos* significa que la optimalidad de la mejor solución conocida no ha podido ser probada, ya que el

proceso de búsqueda no ha sido capaz de descartar completamente la existencia de una solución con una caja más que la mejor solución conocida.

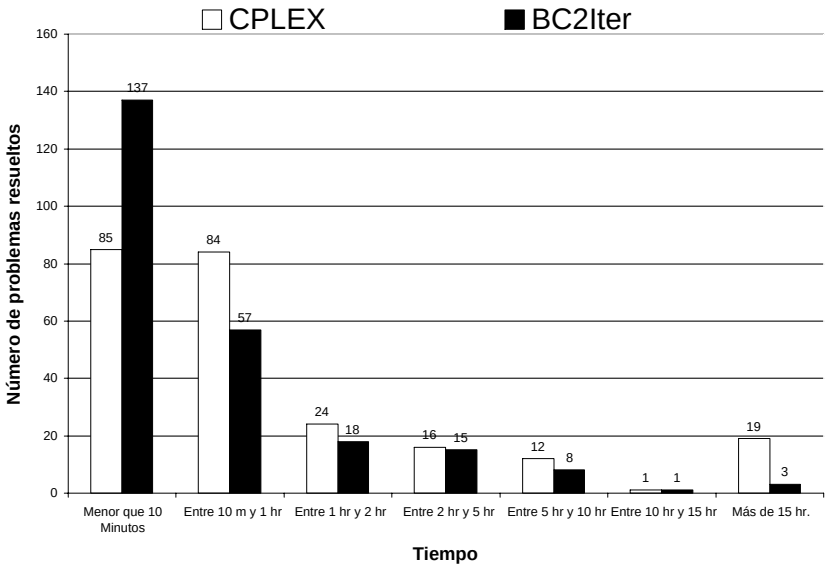


Figura 3.27: *Tipo II. Comparación de algoritmos.*

	B&C2Iter	*CuttingPlanes*	*CPLEX*
En 1 hora	47	37	72
En 2 horas	29	27	48
En 5 horas	14	12	32
En 10 horas	6	7	20
En 15 horas	5	7	19

Tabla 3.8: *Tipo II. Problemas no resueltos.*

Capítulo 4

Un algoritmo Tabu Search para el *PLP*

Tabu Search es un *metaheurístico* de optimización ya establecido (para una introducción ver Glover y Laguna (1997)[44]). Los elementos básicos del algoritmo se describen en las siguientes subsecciones.

4.1. Definición de movimiento

La primera etapa al diseñar un procedimiento *Tabu Search* es la definición de movimiento. Los movimientos nos permitirán explorar el espacio de soluciones pasando de una solución a otra vecina. En nuestro algoritmo, el espacio de soluciones estará formado únicamente por soluciones posibles, es decir, soluciones sin solapamiento.

Analizando las diferentes soluciones del problema, observamos que más que cajas individuales lo que aparecen son *bloques de cajas* (rectángulos formados por conjuntos de cajas con la misma orientación). La aparición de cajas individuales no incluidas en un bloque es muy poco frecuente, a no ser que sean cajas incluidas en alguna *estructura G4* (ver sección, 2.4.5, pág. 23). Los movimientos van a estar basados en la idea de bloques y la idea de *estructura G4*. Esta idea es totalmente distinta a los anteriores metaheurísticos utilizados hasta ahora, en los que los

movimientos consistían en cambiar cajas individuales de la solución, (ver sección, 2.4.6, pág. 28).

El movimiento consiste en cambiar de tamaño los diferentes bloques que aparecen en la solución. Distinguimos dos tipos de movimientos, definidos por la forma de cambiar el tamaño del bloque:

- ***Movimiento de reducción de bloque.***

  El movimiento consiste en reducir el tamaño de un bloque de la solución de forma horizontal (vertical) eliminando filas (columnas) del bloque. La disminución del número de filas o columnas de un bloque incluye la posibilidad de la desaparición completa del bloque.

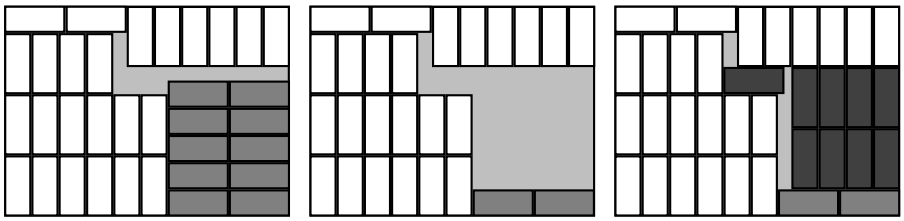
  El movimiento comienza eliminando de la solución actual algunas filas (columnas) del bloque, con lo que aparecerán nuevas zonas de pérdida, no ocupadas por cajas.

  En segundo lugar, procedemos a *desplazar los bloques hacia las esquinas* más cercanas.

  Para rellenar las nuevas pérdidas, el primer paso es *juntar las pérdidas*, que consiste básicamente en unir los diferentes rectángulos de pérdida de manera que se puedan utilizar para colocar en ellos el mayor número de cajas con la orientación que buscamos.

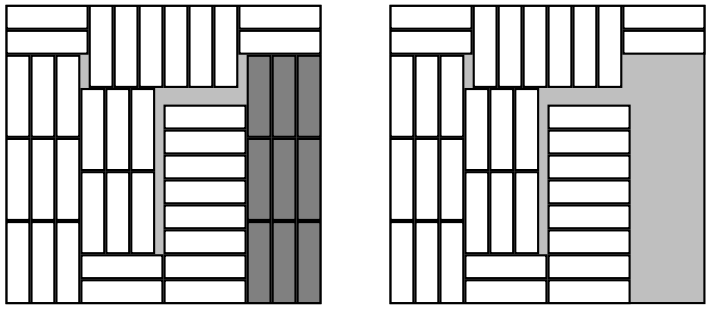
  Para *rellenar los posibles rectángulos* de pérdida, operación que denominaremos *Paletizar*, utilizamos una función que debe ser muy rápida, por lo que sólo consideramos cajas con orientación horizontal o vertical o una *estructura G4*.

  En último lugar tendríamos un mecanismo para *compactar bloques*, con el que intentamos reducir el número de bloques de la solución mediante la unión de bloques contiguos. Algunos ejemplos de movimientos de reducción aparecen en las Figuras 4.1, 4.2.

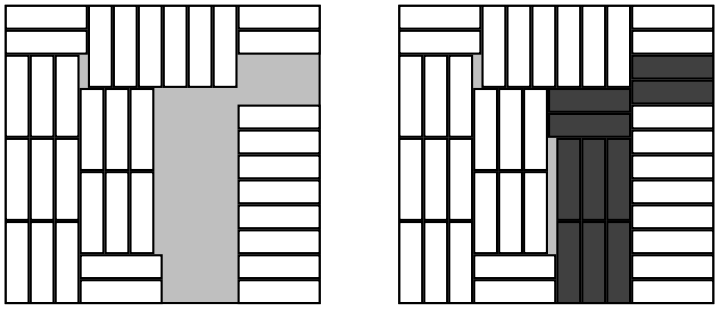


(a) Selección del bloque (b) Disminución (c) Rellenar

Figura 4.1: *Reducción de un bloque. Instancia (42,31,9,4)*



(a) Selección del bloque (b) Reducción



(c) Desplazar (d) Rellenar

Figura 4.2: *Reducción de un bloque. Instancia (38,36,10,3)*

- ***Movimiento de aumento de bloque.***

  El movimiento consiste en aumentar un bloque de la solución de forma horizontal (vertical) al aumentar el número de filas (columnas) del bloque.

  El bloque aumentado puede intersectar con otros bloques, por lo que eliminamos de la solución las cajas del resto de bloques que se solapen con el que hemos aumentado.

  Una vez aumentado el bloque y eliminados los solapamientos, *juntamos las pérdidas* que nos quedan y *paletizamos*. Posteriormente *desplazamos los bloques hacia las esquinas* y volvemos a *juntar pérdidas*. Finalmente, *compactamos los bloques* de la solución. Algunos ejemplos de movimientos de aumento aparecen en las Figuras 4.3, 4.4, 4.5.

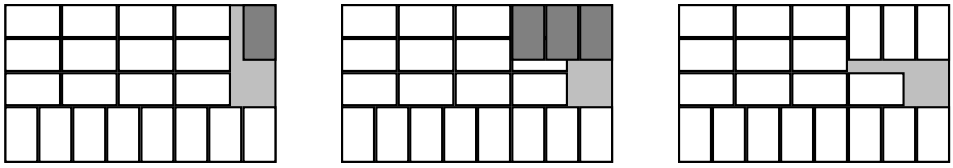
Vamos a explicar más detalladamente los diferentes procedimientos mencionados anteriormente que forman parte del movimiento:

4.1.1. Juntar Rectángulos

Necesitamos un procedimiento para unir los diferentes rectángulos de pérdida que se van produciendo. Este procedimiento es *esencial* para el movimiento. Una unión no eficiente de las pérdidas ocasionará que el procedimiento posterior de rellenar estos huecos no pueda conseguir mejorar las soluciones.

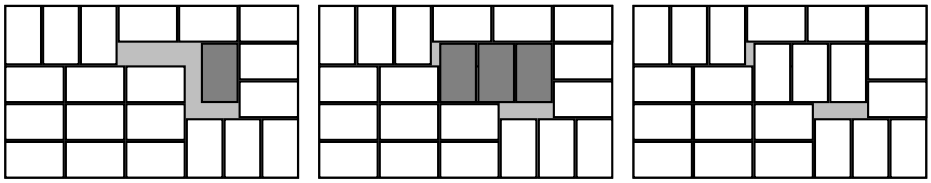
Al unir dos rectángulos, se puede ocasionar la aparición de, como máximo, tres nuevos rectángulos: uno que tendrá una parte de cada uno de los originales, que normalmente es el de mayor tamaño, al que definimos como *Grande* y dos más pequeños que son partes de los antiguos, como se observa en la Figura 4.6.

Intentamos unir los rectángulos para que el resultado sea un rectángulo en el que quepan el mayor número de cajas del tipo que queremos colocar: horizontales o verticales. Para ello imponemos ciertas condiciones a esta unión.



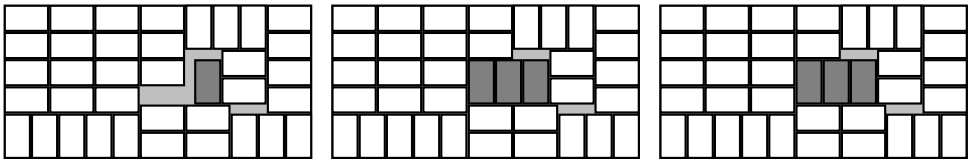
(a) Selección del bloque (b) Aumento Bloque (c) Eliminar solapadas

Figura 4.3: *Aumento de un bloque. Instancia (24,14,5,3)*



(a) Selección del bloque (b) Aumento Bloque (c) Eliminar solapadas

Figura 4.4: *Aumento de un bloque. Instancia (39,23,8,3)*



(a) Selección del bloque (b) Aumento Bloque (c) Eliminar solapadas

Figura 4.5: *Aumento de un bloque. Instancia (34,17,5,3)*

La primera condición es que tengan un segmento en común y sea mayor o igual que w para conseguir rectángulos en los que al menos quepa una caja.

También debe cumplirse que en los nuevos rectángulos se puedan colocar más cajas que en los antiguos con la orientación que buscamos, o más cajas con una *estructura G4* que las que cabían en los anteriores.

Si no se da ninguna de estas condiciones tratamos de conseguir que el rectángulo *Grande* tenga más área que cada uno de los antiguos por separado.

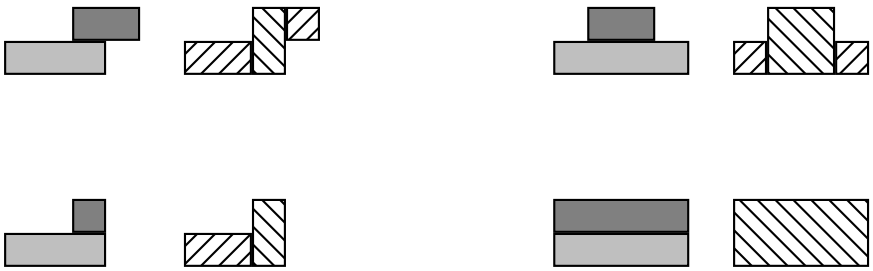


Figura 4.6: *Juntar dos rectángulos*

4.1.2. Desplazar los bloques hacia las esquinas más cercanas

Procuramos que los bloques de las diferentes soluciones estén lo más próximos a las esquinas del pallet, para intentar dejar la pérdida en el centro, donde suponemos que existirán más posibilidades para colocar alguna caja. Los bloques son movidos con la idea de concentrar los espacios vacíos en una sola zona (Figura 4.7). Generalmente las soluciones tienen un número de cajas muy cercano al óptimo (normalmente una caja menos) y unir pequeñas zonas vacías aumenta la oportunidad de la inclusión de una caja adicional en el diseño. Además, este mecanismo será útil para normalizar la posición de los diferentes bloques de la solución.

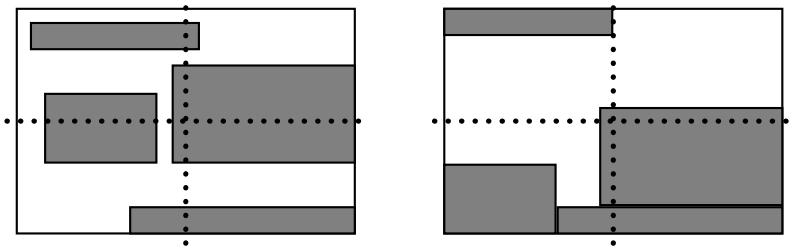


Figura 4.7: *Desplazar los bloques hacia las esquinas*

4.1.3. Rellenar rectángulos (*Paletizar*).

Para rellenar los diferentes rectángulos producidos por la disminución de un bloque o la eliminación de cajas que se solapen necesitamos un procedimiento bastante rápido, por lo que no debe ser muy complejo.

Inicialmente intentamos colocar cajas en una orientación distinta a la de las cajas que se han eliminado. Por ejemplo, si hemos eliminado cajas horizontales intentaremos colocar el mayor número de cajas verticales posibles y sólo cuando no se puedan colocar más cajas verticales intentaremos colocar cajas horizontales. El algoritmo de rellenar rectángulos es iterativo y va colocando cajas mientras que exista alguna pérdida mayor que una caja en la lista de rectángulos vacíos.

Para cada rectángulo se estudian, por tanto, tres posibilidades: colocar un bloque horizontal, un bloque vertical o una estructura G4.

Elegimos la mejor de estas posibilidades y si se da un empate rellenamos con un bloque horizontal o vertical. Al rellenar los rectángulos colocamos los bloques o la *estructura G4* lo más próximo posible a la esquina del pallet más cercana.

El *algoritmo G4* es el mejor heurístico para el problema conocido hasta la fecha. En nuestra opinión, su gran aportación es la idea de *estructura G4* y por ello intentamos introducir este tipo de estructuras en nuestro algoritmo, pero de una manera sencilla. En cada iteración solamente se considera un tipo de *estructura G4* que es la que definimos como estructura *G4-natural*, en el sentido de que es

la más apropiada dada las dimensiones de la caja y el pallet. Será una estructura con el mismo número de cajas horizontales que verticales y que produzca el menor hueco interno. Por ejemplo, para una caja de 7×3, la estructura G4 natural sería la formada por 2 cajas horizontales y 2 verticales como se observa en la Figura 4.8(a), que dejaría un hueco interno de 1×1. Sin embargo, pueden haber otras estructuras alternativas, no tan buenas considerando la pérdida interna producida, pero que quizá encajen mejor en la solución completa del pallet. Se podría considerar otra estructura, Figura 4.8(b), que producirá un hueco interno de 2×2. Por ello, después de un cierto número de iteraciones sin mejorar, cambiamos la que hemos llamado *G4-natural* por la siguiente que consideramos más natural.

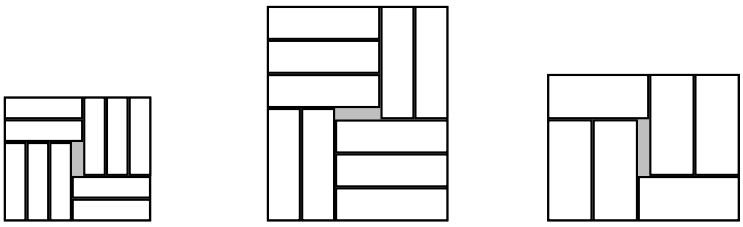
Figura 4.8: *Estructuras G4-naturales para una caja 7 por 3*

Para elegir la estructura *G4-natural* tenemos en cuenta también consideraciones debidas a particiones eficientes, por lo que si el tamaño horizontal o vertical de la estructura G4 no pertenece a una *partición perfecta* no la consideramos.

De esta forma podemos llegar a no quedarnos con ninguna *estructura G4* con igual número de cajas horizontales y verticales. En este caso, estudiamos estructuras con distinto número de cajas horizontales y verticales. Algunos ejemplos se muestran en la Figura 4.9.

Por ejemplo, en la Figura 4.9(a) tenemos la instancia (26, 18, 7, 2). La primera estructura natural sería la que deje un hueco de 1×1 con tres cajas horizontales y tres verticales pero esta estructura de tamaño total (13×13) al colocarse en el pallet de ancho 18 sólo permitiría rellenar con dos piezas horizontales, lo que

produciría una pérdida mínima de 13 unidades. Por esta causa encaja mucho mejor en el pallet la que vemos en la Figura 4.9(a).

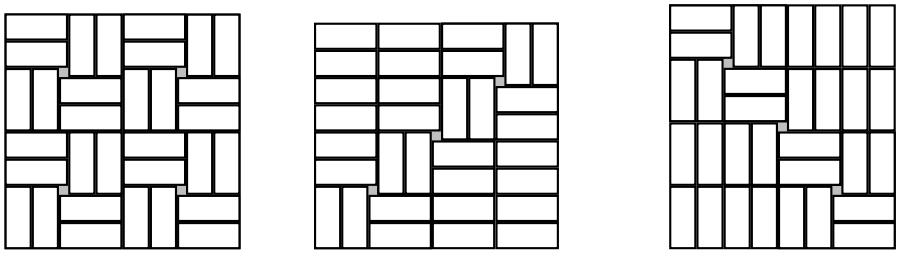


(a) (26,18,7,2) (b) (36,28,10,3) (c) (58,31,9,4)

Figura 4.9: *Otros tipos de estructuras G4-naturales*

Cuando ya tenemos seleccionado un tipo de estructura G4, a partir de ésta fácilmente podemos incluir otras estructuras derivadas:

- Debidas a repetir varias veces la estructura G4 natural (Figura 4.10(a)).
- *G4-encadenados*: las estructuras se encadenan hasta formar estructuras como las de las Figuras 4.10(b), 4.10(c).



(a) (b) (c)

Figura 4.10: *Tipos de estructuras G4 repetidas y encadenadas*

4.1.4. Compactar bloques

El número de movimientos a explorar es mayor cuanto mayor es el número de bloques. Por esta causa creamos un mecanismo para unir (*compactar*) bloques. Dados dos bloques con la misma orientación y con la misma longitud o anchura, se pueden unir en un único bloque:

- Si tienen la misma orientación y comparten un lado completo (Figura 4.11).
- Si no comparten un lado, pero entre ellos sólo existe pérdida, por lo que podemos unirlos en un único bloque (Figura 4.12).

(a) Antes de compactar (b) Bloque Compactado

Figura 4.11: *Compactar Bloques*

(a) Antes de compactar (b) Bloque Compactado

Figura 4.12: *Compactar Bloques*

4.1.5. Reducción del entorno de vecinos

Si tenemos un bloque de dimensión $(n \times m)$, n cajas horizontales y m verticales, podemos tener para este bloque $(n + m - 1)$ reducciones y, dependiendo

del tamaño del pallet, el correspondiente número de aumentos. De todos estos posibles movimientos vamos a descartar los que consideramos poco prometedores. Mediante diversas consideraciones lógicas podemos reducir el entorno sin una disminución importante de la calidad:

- En primer lugar, si un bloque no tiene rectángulos de pérdida contiguos, no se explora. Mover un bloque situado en una región del pallet totalmente ocupada parece menos eficiente que intentar movimientos que involucren directamente a los rectángulos de pérdida.

  Si el bloque tiene alguna pérdida contigua a alguno de sus lados sólo realizaremos movimientos en la dirección en la que aparezca la pérdida. Por ejemplo, si un bloque sólo tiene pérdida a su derecha sólo se permite aumentar o disminuir el bloque hacia la derecha.

- Dependiendo del tamaño de las cajas y de su orientación:

  Necesitamos la siguiente notación:

  $$B_l = \text{Longitud del bloque B}$$
  $$B_w = \text{Anchura del bloque B}$$
  $$l_p = \text{Longitud de la pérdida p}$$
  $$w_p = \text{Anchura de la pérdida p}$$
  $$R_{ori} = \text{filas o columnas reducidas en sentido } ori$$
  $$A_{ori} = \text{filas o columnas aumentadas en sentido } ori$$
  $$ori = \{der, arr, aba, izq\}$$

  No exploraremos un movimiento en las siguientes condiciones:

  - Si *disminuimos* el tamaño de un bloque con orientación horizontal (vertical) por arriba (derecha) o abajo (izquierda):
    - Si $l_p + R_{der} * w < l$, es decir, si en el hueco que se va a originar no va a caber una caja en orientación diferente a la actual, ese

movimiento no se explora, pues en principio no se va a poder colocar ninguna caja nueva.

- Si $((l_p + R_{der} * w) \bmod l) \geq w$ este movimiento de reducción de R_{der} columnas será muy probablemente peor que un movimiento que elimine $R_{der} - 1$ columnas.
- Si $B_l = L$ ó $B_w = W$, sólo permitimos que se reduzca a una *partición perfecta* de ese lado.

- Si *aumentamos* el tamaño de un bloque horizontal (vertical) por arriba (derecha) o abajo (izquierda):
  - Si $((l_p + A_{der} * w) \bmod l) \geq w$, el movimiento de aumento de A_{der} columnas no se explora, ya que siempre será mejor aumentar una columna más.
  - Si el bloque aumentado provoca una pérdida que no permite mejorar la mejor solución conocida.
  - Si aumentar el bloque produce un nuevo bloque con más cajas de un tipo que el número mayor de cajas de ese tipo en una partición perfecta.

- *Puntos interiores.* Si al reducir o aumentar el bloque va a quedar un bloque que no tiene su extremo inferior izquierda en un punto interior.
- Tanto al reducir como aumentar un bloque en una dirección arriba o abajo (o derecha e izquierda) sólo exploramos el movimiento si el ancho (largo) del nuevo bloque puede formar parte de una *partición eficiente* para ese lado del pallet.

4.1.6. Selección de movimientos

En cada iteración estudiamos los movimientos a realizar siguiendo tres etapas.

1. Seleccionar un bloque de la lista de bloques de la solución actual que comparten un lado con alguna zona de pérdida.

2. Considerar todos los posibles tipos de reducciones o aumentos del bloque, en las direcciones en las que el bloque comparta un lado con algún rectángulo de pérdida.

3. Para cada uno de los tipos posibles de movimiento considerar los posibles números de filas (columnas) que se reducen o aumentan.

Estas tres etapas se repiten para cada bloque de la solución actual. El orden para estudiar los distintos movimientos es aleatorio.

4.1.7. Función objetivo

La función objetivo inicial de este problema consiste únicamente en maximizar el número de cajas. Sin embargo si la evaluación del movimiento se hace únicamente con este criterio pueden existir una gran cantidad de movimientos de igual valor.

Por ello hemos incluido en la evaluación de los movimientos algunas propiedades que consideramos *deseables* para una solución. Las enumeramos en orden de importancia en la función objetivo:

- *Simetría*. Intentamos no explorar soluciones simétricas. Preferimos soluciones en las que la pérdida esté concentrada hacia la derecha y hacia la parte superior del pallet.

- *Eficiencia*. Si es posible, preferimos soluciones en las que todos los bloques sean *eficientes*. Decimos que un bloque es *eficiente* si su altura y anchura pueden formar parte de una *partición perfecta*.

  Para intentar conseguir este tipo de soluciones preferimos mover bloques que no sean eficientes a otros que sí lo sean. Con ello intentamos guiar al procedimiento hacia soluciones que tengan todos los bloques eficientes.

- *Pérdidas unidas y centradas*. Intentamos que las zonas de pérdida estén lo más concentradas posible y a su vez lo más centradas posible. Es de esperar

que sea más probable colocar un mayor número de cajas si hacemos que se junten todas las pérdidas que si éstas están dispersas. Asimismo, que las pérdidas estén centradas facilitan los movimientos de cajas.

- *Pérdida en lados*. Preferimos soluciones que no tengan ninguna pérdida en los bordes del pallet.

- *Óptimo*. No permitir soluciones con bloques que no puedan aparecer en una solución mejor que la mejor solución conocida.

- *Nuevas*. Preferimos realizar movimientos en los que aparezca alguna caja nueva, en lugar de quedarnos con una solución que sea igual que la anterior pero eliminando alguna caja.

  Estos criterios se incorporan a la función de evaluación con unos ciertos pesos que reflejan su importancia relativa. Hemos elegido estos pesos a la vista de los resultado obtenidos en un estudio preliminar sobre un subconjunto de problemas.

4.1.8. Puntos interiores

Como observamos en la sección 3.3.2 (pág. 58) se puede reducir el conjunto de posibles puntos donde colocar una caja. Buscamos soluciones normalizadas y para ello sólo tenemos en cuenta soluciones en las que todas las esquinas inferiores de los bloques estén colocadas en variables posibles con la nueva reducción (sección 3.3.2). Solamente admitimos soluciones que no cumplan este requisito si mejoran la mejor solución obtenida hasta el momento.

4.1.9. Lista Tabu

Hemos probado tres tipos diferentes de lista tabu:

- *Lista de no regreso de bloques.*

  Incluimos en la lista tabu el bloque elegido para hacer el movimiento. Un movimiento será tabu si la nueva solución producida contiene algún bloque que esté en la lista tabu. Es decir, durante un cierto número de iteraciones la lista tabu evita que volvamos a una solución con el bloque en la misma posición y del mismo tamaño que el que hemos modificado.

- *Lista de permanencia de bloques.*

  Incorporamos en la lista los bloques que aparecen en la solución al efectuar el movimiento. Un movimiento será tabu si el bloque elegido para reducir o aumentar pertenece a la lista tabu. Intentamos que durante un cierto número de iteraciones el nuevo bloque permanezca en la solución. Esta lista puede producir que no exista ningún bloque que no sea tabu.

- *Lista de no repetición de pérdidas.*

  De cada solución guardamos en la lista tabu el conjunto de sus rectángulos de pérdida. Un movimiento será tabu si las pérdidas que ocasionan están en la lista tabu. Intentamos no volver a una solución con el mismo conjunto de pérdidas.

La longitud de la lista cambia dinámicamente después de un número dado de iteraciones sin mejorar la solución conocida. Si m^* es el número de cajas de la cota del área (ver sección 2.5.1, pág. 32), la longitud se selecciona aleatoriamente en el intervalo $[0{,}5m^*, 1{,}5m^*]$.

El *criterio de aspiración* permite movernos a una solución *tabu*, sólo si este movimiento mejora el número de cajas de la mejor solución obtenida hasta el momento.

4.1.10. Mejora local

Hacemos una mejora local después de elegir cada movimiento, de la siguiente forma:

Examinamos si mejoraríamos la solución incluyendo una *estructura G4-natural* como hemos explicado en la sección 4.1.3 en alguna de las cuatro esquinas y quitando todas las cajas que se solapen con esta nueva estructura.

4.1.11. Procedimiento básico Tabu Search

Sea S la solución actual con un conjunto asociado de bloques B, pérdidas P y valor v_s.

Sea S^* la mejor solución conocida con valor v^*.

Sea $S^* = S = S_I$, la solución inicial obtenida usando el algoritmo de un solo bloque (todas las cajas con la misma orientación).

Sea $niter = 0$.

Iteración: Mientras ($niter < maxiter$).

{

Etapa 1. Elección del tipo de movimiento

Seleccionar aleatoriamente un bloque $b_i \in B$ que comparta algún lado con alguna pérdida de P.

Mientras se pueda seleccionar algún bloque:

Seleccionar aleatoriamente tipo de movimiento: aumentar o reducir.

Seleccionar aleatoriamente la dirección del movimiento.

Seleccionar aleatoriamente la cantidad de filas o columnas a aumentar o reducir.

Etapa 2. Efectuar movimiento

Evaluar si el movimiento pertenece al entorno de vecinos reducido.
Si pertenece al entorno, seguir.
Si no, volver a **Etapa 1**.

Si el movimiento es de aumentar:

Aumentar las filas o columnas del bloque b_i.

Quitar las cajas de los restantes bloques que se solapen con el nuevo bloque b_i.

Rellenar los posibles rectángulos de pérdidas.

Mover los bloques hacia las esquinas.

Compactar bloques.

Calcular el número de cajas que se han introducido.

Si el movimiento es de disminuir:

Quitar las filas o columnas del bloque b_i.

Desplazar los bloques hacia las esquinas.

Unir las diferentes pérdidas intentado colocar el mayor número de cajas con orientación diferente a b_i.

Rellenar la nueva lista de rectángulos.

Compactar bloques.

Mover los bloques hacia las esquinas.

Calcular el número de cajas que se han introducido.

Etapa 3. Elegir el mejor

Si se dan alguna de las condiciones:

- el movimiento no es *tabu*, la solución tiene todos las esquinas inferiores de sus bloques en puntos interiores y el número de cajas es mayor o igual que el del mejor movimiento estudiado hasta el momento
  ó

- el movimiento proporciona una solución con un número de cajas mayor que v_{best}.

  Evaluar con la función objetivo explicada anteriormente.

  Elegir el mejor movimiento posible S_{best} y v_{best}.

Si todos los elementos estudiados son *tabu* y no cumplen el criterio de aspiración, se elimina de la lista tabu el elemento más antiguo y se vuelve a considerar el status *tabu* de los movimientos. Este proceso se repite hasta encontrar algún movimiento no *tabu*.

Etapa 4. Actualizar el movimiento

Actualizar la solución actual S sustituyendo los nuevos rectángulos.

Actualizar la lista tabu.

Si $v_{best} > v*$, actualizar la mejor solución conocida $S^* = S_{best}$.

$niter = niter + 1$

Etapa 5. Mejorar la solución

Si $v^* = v_{best}$ intentar realizar una mejora local de la solución.

}

4.2. Estrategias de intensificación y diversificación

La definición del movimiento supone un importante grado de diversificación en el proceso de búsqueda, pero a nuestro juicio necesitamos algunas estrategias complementarias. Hemos probado dos técnicas diferentes:

4.2.1. Oscilación Estratégica

Dado que tenemos varios pesos en la función objetivo, una primera posibilidad es la oscilación de estos pesos, ya que los elegidos inicialmente pueden producir buenos resultados en la mayoría de problemas, pero en algunos casos pueden no ser los adecuados para obtener la solución óptima del problema.

Por ello, realizamos la oscilación de la forma siguiente:
Utilizamos un coeficiente $\beta_i,\ i = 1, \ldots 6$ que multiplica a cada peso. Inicialmente cada $\beta_i = 1,\ i = 1, \ldots 6$. Cuando llevemos un cierto número de iteraciones sin mejorar hacemos oscilar estos coeficientes entre los valores 0 y 1.

4.2.2. Uso de la memoria a largo plazo

Utilizamos la memoria a largo plazo como una estrategia más de diversificación. A lo largo de la búsqueda vamos anotando el número de veces que aparece un bloque de un determinado tamaño y orientación en cada solución.

Posteriormente esta información será utilizada de la siguiente forma: permitiremos mover más fácilmente un bloque que haya aparecido muchas veces a lo largo de la búsqueda que otros que hayan aparecido en menos ocasiones. De esta forma se favorece el estudio de soluciones que contengan estos bloques menos frecuentes.

Para implementar esta estrategia creamos dos matrices, una para cada orientación posible de los bloques. En cada una de ellas vamos apuntando las veces que aparece un bloque horizontal o vertical y su tamaño. Por ejemplo si tenemos una solución con un bloque horizontal de 2 filas por 3 columnas de cajas en la matriz de orientación horizontal, el elemento [2][3] es incrementado en uno.

Esta técnica también se utilizó de manera inversa, es decir, como estrategia de intensificación, anotando sólo aquellos bloques que aparecen en soluciones *buenas*, para posteriormente guiar al algoritmo para que se mueva por soluciones con bloques muy frecuentes en las soluciones *buenas*.

4.3. Path Relinking

Si se tiene un conjunto de soluciones de *referencia*, que es típicamente un conjunto de soluciones de gran calidad, entonces *Path Relinking* genera nuevas soluciones explorando los caminos que las conectan. Comenzando desde soluciones

iniciales, una serie de movimientos definen un camino en el espacio de soluciones que llevan a otras soluciones, llamadas *soluciones guías*.

Esta técnica se denomina *Path Relinking* ya que en *Tabu Search* dos soluciones cualesquiera se enlazan por una serie de movimientos que se ejecutan durante la búsqueda (Glover y Laguna (1997)[44]). Ahora, estas soluciones se unen otra vez pero siguiendo un camino diferente en el cual los movimientos no son seleccionados según el mejor valor de la función objetivo, sino buscando movimientos que lleven directamente a la solución guía.

Consideremos dos soluciones élite, tomando una de ellas A como solución inicial y la otra B como solución guía. La idea es llegar a la solución B partiendo de la solución A.

Intentamos que no coincidan las pérdidas de las dos soluciones para favorecer que el proceso produzca soluciones nuevas. Dado que el algoritmo *tabu search* prima las soluciones con pérdidas predominantemente en la parte superior derecha del pallet, ambas soluciones tenderán a tener las pérdidas en esa zona. Por ello, a la solución inicial la sometemos a una simetría vertical antes de comenzar el *path relinking*.

Seguimos una estrategia constructiva, introduciendo uno por uno los bloques de la solución B en la solución A. Cada vez que introducimos un bloque de B en A, procedemos a quitar de A los bloques que solapen con el que hemos introducido de B, desplazamos los bloques que queden en la solución A que no son de B, hacia las esquinas y rellenamos los posibles huecos que hayan quedado. Al final del proceso habremos reproducido totalmente la solución guía B.

4.3.1. Procedimiento Path Relinking

Sea $\mathbf{S}$ el conjunto de soluciones de referencia.

Para cada par $(S_1,\ S_2)$ con $S_1,\ S_2 \in \mathbf{S}$, repetir el siguiente procedimiento.

Sea B_i la lista de bloques de la solución $S_i,\ i = 1, 2$.

Si $|B_1| > |B_2|$ tomar como guía $A = S_2$ y solución de referencia $B = S_1$, en otro caso: $A = S_1$ y $B = S_2$.

Inicialización:

Sea B_A (B_B) la lista de bloques de la solución A (B).

Transformar A mediante una simetría vertical.

Sea $v = 0$, el valor de la solución construida.

Mientras existan bloques de B_B que no están en B_A y $A \neq B$.

Para cada bloque $b_k \in B_B$, incluir el bloque en la solución A.

Quitar todos los bloques de B_A que solapen o corten con b_k.

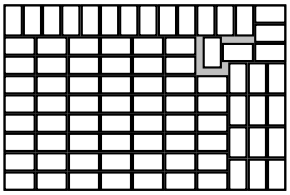
Mover los bloques de la solución A que no sean de B hacia las esquinas.

Juntar los rectángulos de pérdida.

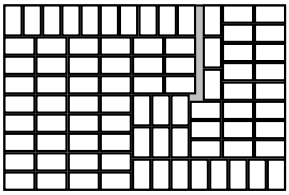
Rellenar los posibles rectángulos de pérdida.

Ejemplo 4.1 *Problema (44,29,5,3). Denotamos un rectángulo por una cuádrupla (x_1, y_1, x_2, y_2), donde (x_1, y_1) son las coordenadas de la esquina inferior izquierda y (x_2, y_2) de la esquina superior derecha. Partimos de dos soluciones de valor 84. En primer lugar, tomamos como solución guía la que tiene más bloques que en este caso es la segunda solución (Figura 4.13(b)).*

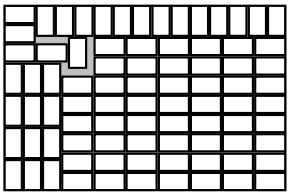
Sometemos la primera solución (Figura 4.13(a)) a una simetría vertical para que las pérdidas de ambas soluciones no se solapen (Figura 4.13(c)). Empezamos a insertar bloques de la solución guía en la solución de referencia ordenándolos por bloques próximos a las esquinas. El primer bloque es el (29,0,44,5) (gris oscuro). Quitamos todos los bloques que solapen con él, en este caso sólo el (9,0,44,30), desplazamos los bloques hacia las esquinas y rellenamos las pérdidas, con lo que incorporamos los dos bloques en gris más claro.



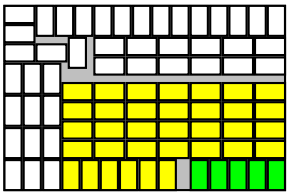
(a) Solución Inicial (A)



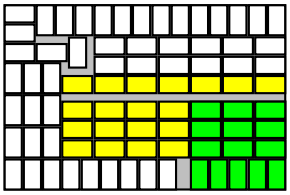
(b) Solución guía (B)



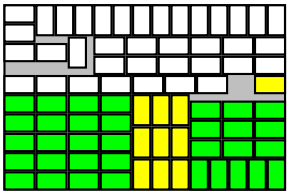
(c) Solución simétrica (A)



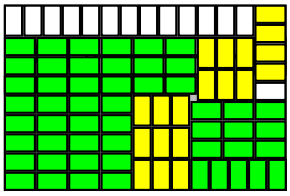
(d) Primer bloque



(e) Segundo bloque



(f) Tercer bloque



(g) Cuarto Bloque

Figura 4.13: *Path Relinking*

En segundo lugar incorporamos el bloque (29,5,44,14), volviendo a repetir el proceso, seguidamente el bloque (0,0,20,15) y finalmente el bloque (0,15,30,24). Obtenemos una solución óptima de valor 85.

4.4. Resultados computacionales

Los algoritmos han sido codificados en *C++* y ejecutados en un ordenador personal con un procesador *Pentium IV* a *2 Ghz*.

El criterio de parada para el algoritmo *Tabu Search* es un número máximo de 100000 iteraciones. Este límite permite mantener el tiempo de computación por debajo de 10 minutos.

Cada 100 iteraciones se intenta cambiar de *estructura G4-natural* si existen varias posibilidades de este tipo de estructuras. Asimismo, la longitud de la lista tabu también varía cada 100 iteraciones sin mejorar.

El tipo de lista tabu que proporcionó mejores resultados fue la lista de no regreso de bloques.

Cada 10000 iteraciones sin mejorar se procede a la oscilación estratégica de los parámetros y cada 9000 iteraciones se procede a una fase de diversificación durante 250 iteraciones seguida de una fase de intensificación de otras 250 iteraciones.

El uso del path relinking fue descartado debido a que el esfuerzo computacional no era proporcional a la mejora obtenida en los resultados.

Para el conjunto *Tipo I* el algoritmo resuelve todas las instancias en menos de 250 iteraciones. Esto supone un tiempo de computación inferior a 1 segundo. Los resultados computacionales para el conjunto *Tipo II* se observan en la Tabla 4.1 en la que se muestra el número de instancias no resueltas óptimamente en función del número de iteraciones.

El algoritmo *Tabu Search* proporciona muy buenos resultados, dado que de las 7827 instancias de *Tipo I* y 40609 de *Tipo II* sólo quedan 15 problemas sin resolver óptimamente.

Iteración	No resueltos
500	229
1000	137
10000	38
50000	23
100000	15

Tabla 4.1: *Problemas de Tipo II (40609 Instancias).*

El mejor heurístico conocido hasta el momento es el *G4*. Sin embargo, para 45 problemas del *Tipo II* este algoritmo no consigue el óptimo debido a que sólo examina diseños con *estructura G4*, por lo que cuando no existe una solución óptima con este tipo de estructura, no alcanzará jamás el óptimo, al ser un algoritmo constructivo. De estos 45 problemas considerados como *difíciles*, sólo han quedado 15 por resolver, dado que todos los que conseguía el *algoritmo G4* también son resueltos con el *algoritmo tabu*.

Por otro lado hemos comparado con el último heurístico publicado debido a Lins et al. (2003)[60] que resuelven óptimamente todos los problemas de un subconjunto del *Tipo II* aunque necesitando en algunos casos tiempos de computación muy elevados. Únicamente en una de esas instancias nuestro algoritmo no encuentra la solución óptima.

Hemos generado un nuevo conjunto de instancias, *Tipo III*, para evaluar la eficiencia del algoritmo en problemas más grandes. El conjunto contiene 98016 instancias que cumplen:

$$1 \leq \frac{L}{W} \leq 2 \qquad 1 \leq \frac{l}{w} \leq 4 \qquad 101 \leq \frac{L * W}{l * w} < 151 \tag{4.1}$$

Los resultados para este conjunto se muestran en la Tabla 4.2 que sigue el mismo esquema de la Tabla 4.1, pero comparando con las cotas superiores, en vez de con las soluciones óptimas que en este caso no son conocidas. Los 1122 problemas que quedan por resolver son problemas en que ninguna de las cotas superiores utilizadas coincide con el valor proporcionado por nuestro heurístico. Si el comportamiento para este nuevo conjunto de instancias es parecido al obtenido sobre el conjunto de *Tipo II* es de suponer que en muchos de estos problemas la solución óptima se encuentra en la solución del heurístico. En ninguna instancia del *Tipo III* la diferencia entre la solución proporcionada por nuestro heurístico y la cota superior es mayor de una caja.

Iteración	No resueltos
1000	1770
10000	1263
25000	1202
50000	1173
100000	1122

Tabla 4.2: *Problemas de Tipo III (98016 Instancias).*

Finalmente mostramos los diseños óptimos conseguidos por nuestro algoritmo para dos instancias del Tipo II (Figura 4.14, 4.15) no resueltas previamente por ningún heurístico. También el diseño óptimo para dos instancias del Tipo III (Figura 4.16, 4.17).

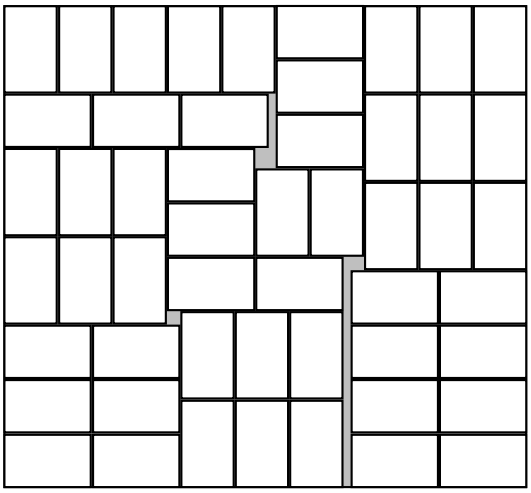


Figura 4.14: *Instancia (77, 71, 13, 8)*

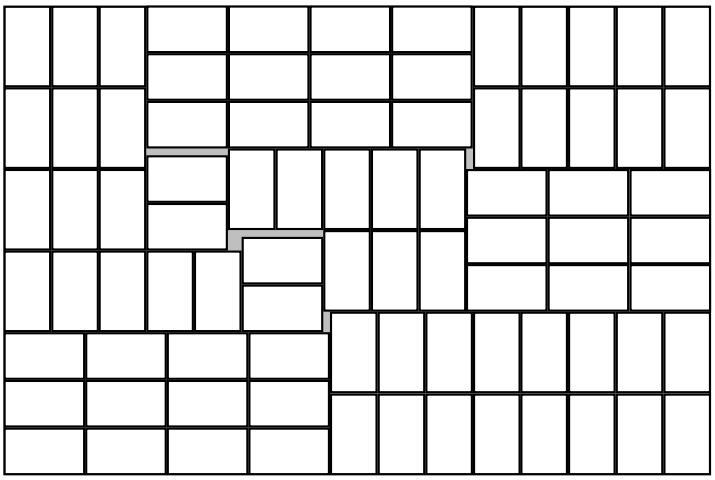


Figura 4.15: *Instancia (104, 69, 12, 7)*

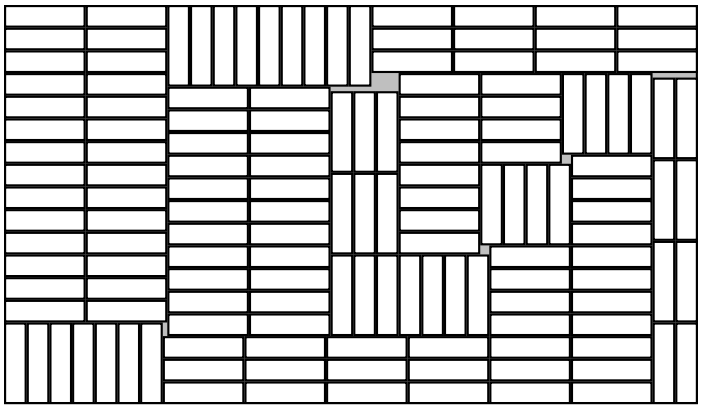


Figura 4.16: *Instancia (153, 88, 18, 5). Tipo III*

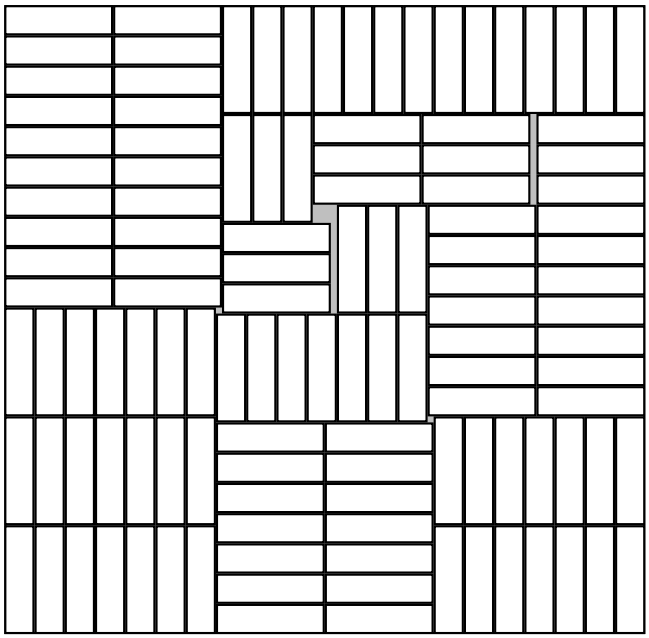


Figura 4.17: *Instancia (106, 104, 18, 5). Tipo III*

Capítulo 5

Un algoritmo GRASP para el problema con cajas de diferente tamaño

En los capítulos anteriores hemos trabajado con el problema del pallet (*manufacturer's pallet loading*). El problema se daba en las fábricas donde es habitual empaquetar pallets de un único producto para el envío a los distribuidores. En este capítulo tratamos el problema del *distributor's pallet loading*, en el que un distribuidor recibe productos de varias fábricas embalados en cajas de diferentes dimensiones. Estas cajas son cargadas sobre pallets para su posterior distribución a las tiendas. En algunos casos no todas las cajas pueden ser cargadas, por lo que se le da un determinado peso o prioridad a cada caja. El peso o prioridad de cada caja no tiene por qué estar relacionado con sus dimensiones. Se puede preferir cargar una caja pequeña a una grande, dado que es más urgente o conlleva un mayor beneficio cargar la primera. Por tanto, el objetivo del problema en muchas ocasiones no es utilizar la mayor área del pallet posible, sino maximizar el valor o prioridad de los paquetes cargados en el pallet.

Los problemas de corte y empaquetamiento, como hemos mencionado en la introducción de esta tesis, están estrechamente relacionados. Concretamente nuestro problema también puede ser visto como un problema de corte. Supongamos un tablero del que deseamos cortar una serie de piezas, donde el valor de cortar

cada pieza no tiene por qué ser exactamente su área. El objetivo es obtener la mayor ganancia al cortar ese tablero. También debemos tener en cuenta que, en algunos casos, las piezas no pueden ser rotadas ya que si el material a cortar tiene alguna veta o dibujo una pieza (l, w) no es igual a una pieza (w, l) con $l \neq w$.

Este problema se conoce comúnmente como el *problema restringido bi-dimensional de corte no-guillotina*.

Definido formalmente, el problema consiste en cortar de un rectángulo de dimensiones (L, W) que llamamos *tablero*, de longitud L y anchura W, un conjunto de rectángulos (l_i, w_i), que llamamos *piezas*, de longitud l_i y anchura w_i $(i = 1, \ldots, m)$. Cada pieza tiene orientación fija, es decir, no puede ser rotada. Los cortes deben ser paralelos a los lados del tablero. El número de piezas de cada tipo i debe estar dentro de unos límites P_i, Q_i con $(0 \leq P_i \leq Q_i)$. Por notación definimos $M = \sum_{i=1}^{m} Q_i$ y se puede suponer sin pérdida de generalidad que todas las dimensiones son enteras. Cada tipo de pieza i tiene asociado un *valor* v_i y el objetivo es *maximizar el valor total de las piezas cortadas*.

Podemos hablar de tres tipos de problemas:

- El problema *no restringido*, donde no existen límites, ni superiores, ni inferiores, para el número de piezas a cortar de cada tipo, excepto la cota superior trivial: $Q_i = \lfloor \frac{L*W}{l_i*w_i} \rfloor \, \forall i = 1, \ldots, m$.
- El problema *restringido*, donde está acotado superiormente el número de piezas de cada tipo que se pueden utilizar: $\exists \; i \mid Q_i < \lfloor \frac{L*W}{l_i*w_i} \rfloor$.
- El problema *doblemente restringido*, donde además de estar acotado el número máximo de algunos tipos de piezas a cortar, se exige cortar un número mínimo de piezas de algunos tipos: $\exists \; i \mid P_i > 0$.

Definimos la $eficiencia$ de un tipo de pieza i como $e_i = v_i/(l_i * w_i)$, es decir, el beneficio por unidad de superficie de la pieza de tipo i. Algunos autores diferencian dos tipos de problemas, dependiendo de la eficiencia de las piezas: el

problema sin pesos (*un-weighted*) donde el valor de cada pieza es el área de la pieza ($e_i = 1\ \forall i$) y el problema con pesos (*weighted*) donde el valor de la pieza no tiene por qué depender del área de la pieza ($\exists i\,|e_i \neq 1$), por lo que pueden existir piezas pequeñas con valores superiores a otras más grandes. Nuestro algoritmo pretende ser de carácter general, por lo que no distinguimos entre estos dos tipos de problemas.

En la Figura 5.1 observamos un ejemplo donde tenemos un tablero (10, 10), el número de tipos de piezas es m=10 y el número de piezas posibles a cortar es M=21. En la Figura 5.1(a) se observa la solución óptima para el problema restringido con $P_i = 0\ \forall i = 1, \ldots, m$ y en la Figura 5.1(b) la solución para el problema doblemente restringido, donde se obliga a cortar algunas piezas, concretamente para la piezas de tipo: 1, 2, 3, 7 debemos cortar al menos una pieza.

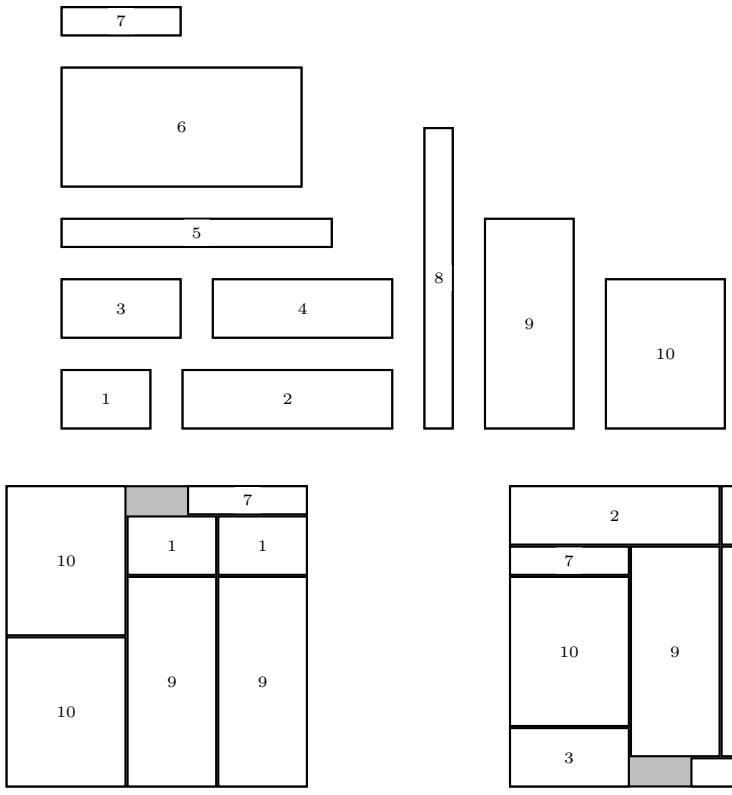
5.1. Métodos de solución

En esta sección hacemos una descripción general de algunos de los métodos existentes en la literatura con los cuales se ha intentado resolver el problema general. No pretendemos hacer una exposición exhaustiva de los mismos, sino solamente presentar una visión panorámica de las aportaciones que nos parecen más significativas y relacionadas con nuestro trabajo.

5.1.1. Métodos exactos

Existen algunos métodos exactos propuestos para el problema: Beasley (1985) [9], Tsai et al. (1988)[85] y Scheithauer y Terno (1993)[77] presentan formulaciones enteras. Hadjiconstantinou y Christofides (1995)[46] también proponen otra formulación, aunque su formulación es incorrecta (Amaral y Letchford (1999)[1]). Arenales y Morabito (1995)[4] presentan un branch and bound basado en un tipo especial de grafo (AND/OR graph). Fekete y Schepers (1997)[42] crean un procedimiento basado en la representación mediante grafos de la posición relati-

$Tablero: L = 10\ W = 10$						
$Pieza$	l_i	w_i	P_i	Q_i	v_i	e_i
1	3	2	1	2	7	1,166
2	7	2	1	3	20	1,428
3	4	2	1	2	11	1,375
4	6	2	0	3	13	1,083
5	9	1	0	2	21	2,333
6	8	4	0	1	79	2,468
7	4	1	1	2	9	2,25
8	1	10	0	1	14	1,4
9	3	7	0	3	52	2,476
10	4	5	0	2	60	3



(b) Restringido. Óptimo 247 (c) Doblemente restringido. Óptimo 220

Figura 5.1: *Instancia 3 de la Tabla 5.8.*

va de las piezas en un patrón posible. Finalmente, Caprara y Monaci (2004)[19] presentan un algoritmo con algunas mejoras respecto a la propuesta de Fekete y Schepers.

5.1.2. Cotas superiores

Una cota natural es la obtenida mediante la transformación del problema bidimensional en un problema unidimensional. Sea r_i el número de piezas de tipo i cortadas del tablero (L, W) donde $r_i \geq 0$ y entero. Consideramos la formulación entera:

$$\text{maximizar} \quad \sum_{i=1}^{m} v_i r_i \tag{5.1}$$

$$\text{sujeto a :} \quad \sum_{i=1}^{m} (l_i w_i) r_i \leq LW \tag{5.2}$$

$$P_i \leq r_i \leq Q_i, \ i = 1, \ldots, m \tag{5.3}$$

$$r_i \geq 0 \text{ y entero}, \ i = 1, \ldots, m. \tag{5.4}$$

La ecuación (5.2) asegura que el área total de las piezas cortadas no exceda la superficie del tablero y la ecuación (5.3) asegura que el número de piezas de cada tipo caiga dentro de los límites apropiados. La función objetivo es maximizar el valor de las piezas cortadas y la solución de esta formulación será una cota superior para el problema de corte. La formulación anterior puede ser transformada fácilmente en un problema mochila acotado (5.5)-(5.8) y resolverse en tiempo pseudo-polinómico usando programación dinámica, (por ejemplo el propuesto por Pisinger (1994)[72] que se encuentra disponible en la siguiente dirección: *http://www.diku.dk/ pisinger/bouknap.c.*)

$$\text{maximizar} \quad \sum_{i=1}^{m} v_i r_i + \sum_{i=1} v_i * P_i \tag{5.5}$$

$$\text{sujeto a :} \quad \sum_{i=1}^{m} (l_i w_i) r_i \leq \left(LW - \sum_{i=1}^{m} P_i (l_i w_i) \right) \tag{5.6}$$

$$r_i \leq Q_i - P_i, \ i = 1, \ldots, m \tag{5.7}$$

$$r_i \geq 0 \text{ y entero}, \ i = 1, \ldots, m. \tag{5.8}$$

Otra cota se origina mediante la relajación lineal de la formulación entera dada por Beasley (1985)[9], ya utilizada para el problema del pallet. Vamos a describir la formulación para el problema general.

En primer lugar definimos unas variables para definir la adyacencia de dos piezas:

$$a_{ipqrs} = \begin{cases} 1 & \text{si una pieza de tipo } i \text{ cortada desde } (p,\, q) \text{ cubre el punto } (r,\, s), \\ 0 & \text{en otro caso.} \end{cases}$$

por tanto:

$$a_{ipqrs} = \begin{cases} 1 & \text{si } 0 \leq p \leq r \leq p + l_i - 1 \leq L - 1 \text{ y } 0 \leq q \leq s \leq q + w_i - 1 \leq W - 1 \\ 0 & \text{en otro caso.} \end{cases}$$

Definimos $L_0 = \{0, \ldots, L-1\}$ el conjunto de localizaciones posibles para el extremo inferior de una pieza. De la misma forma $W_0 = \{0, \ldots, W-1\}$ conjunto de localizaciones posibles para el extremo izquierdo de una pieza. Inicialmente $L_0 = \{0, 1, 2, \ldots, L-2, L-1\}$ y $W_0 = \{0, 1, 2, \ldots, W-1\}$ pero como ya observamos en el problema del pallet estos conjuntos pueden reducirse mediante diversas consideraciones. Se define:

$$x_{ipq} = \begin{cases} 1 & \text{si una pieza de tipo } i \text{ es cortada desde } (p,\, q), \\ 0 & \text{en otro caso.} \end{cases}$$

La formulación que se obtiene es la siguiente:

$$\text{máx} \sum_{i=1}^{m} \sum_{p \in L} \sum_{q \in W} v_i x_{ipq} \tag{5.9}$$

sujeto a:

$$\sum_{i=1}^{m} \sum_{p \in L_0} \sum_{q \in W_0} a_{ipqrs} x_{ipq} \leq 1 \ \forall r \in L_0, \ \forall s \in W_0 \tag{5.10}$$

$$P_i \leq \sum_{p \in L_0} \sum_{q \in W_0} x_{ipq} \leq Q_i, \ i = 1, \ldots, m \tag{5.11}$$

$$x_{ipq} \in (0,1) \ i = 1, \ldots, m \ \forall p \in L_0, \ \forall q \in W_0. \tag{5.12}$$

La ecuación (5.10) evita que haya solapamientos entre las piezas. La ecuación (5.11) garantiza que el número de piezas de cada tipo se encuentre entre los límites permitidos. El procedimiento de cota propuesto por Beasley consistía en relajación lagrangiana.

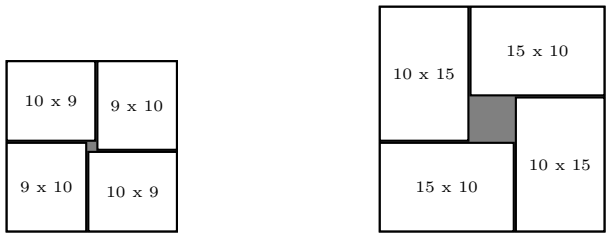
Otros procedimientos de cota más complejos son presentados por Hadjiconstantinou y Christofides (1995)[46] que usan relajación lagrangiana para derivar una cota superior. Scheithauer (1999)[76] propone otra cota basada en generación de columnas, mejorada por Amaral y Letchford (2003)[2]. Fekete y Schepers (1997)[42] estudian otros métodos de cota basados en un grupo de funciones llamadas *funciones duales posibles*.

5.1.3. Algoritmos Heurísticos

Explicamos a continuación los métodos heurísticos propuestos para el problema. En primer lugar hablaremos de los métodos constructivos y posteriormente de los metaheurísticos.

5.1.3.1. Métodos constructivos

Podemos distinguir los siguientes métodos constructivos empleados para el problema:



(a) No con BL (b) No con DP

Figura 5.2: *Patrones que no puede ser obtenidos con otras técnicas.*

- El procedimiento más habitual empleado para colocar piezas es el algoritmo *Bottom-Left* (BL). A partir de un orden de la piezas a cortar, el algoritmo consiste en colocar cada pieza en el tablero lo más abajo posible y una vez hecho esto mover la pieza todo lo que se pueda hacia la izquierda. Al aplicar este método pueden quedar áreas no aprovechables (desperdicios). Una modificación de esta estrategia para intentar aprovechar las áreas no aprovechables es la estrategia *Bottom-left-fill*(BLF). Este procedimiento antes de colocar una pieza según la estrategia BL comprueba si se puede colocar en alguna de las áreas no aprovechables que se han generado hasta el momento. En cualquier caso, existen patrones de corte que son imposibles de reproducir (Figura 5.2(a)).

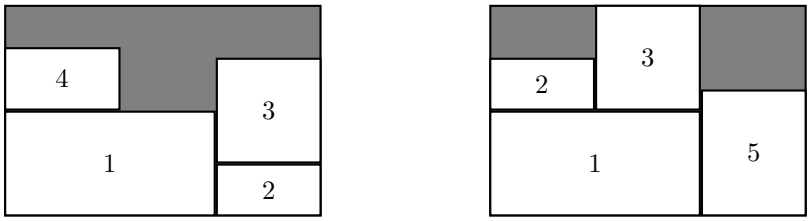
  Ejemplo 5.1 *Supongamos la instancia de la Figura 5.3 y suponer el orden inicial (1, 2, 3, 4, 5). En primer lugar colocamos la pieza 1 en la posición (0,0), posteriormente la 2 en (20,0), la 3 en (20,10) y finalmente la 4 en la posición (0,20). El patrón final quedaría como se observa en la Figura 5.3(a).*

- Otro método utilizado en la literatura para construir una solución a partir de un orden de las piezas, es el mecanismo *Difference Process* (DP) de Lai y Chan[55]. La estrategia *Difference Process* intenta colocar cada pieza

en la esquina posible del resto del diseño más cercana a la esquina inferior-izquierda del tablero. Para medir cuál es esta posición se emplea la distancia euclídea. Al igual que con el método anterior existen patrones que no puede alcanzarse con esta estrategia (5.2(b)).

Ejemplo 5.2 *Tenemos la instancia de la Figura 5.3 y supongamos el orden inicial (1, 2, 3, 4, 5). En primer lugar colocamos la pieza 1 en la posición (0,0) que está a una distancia 0 de la esquina inferior. Una vez colocada la primera pieza tenemos dos posibles esquinas para colocar la segunda pieza: (0,10) y (20,0) si tomamos la distancia euclídea, la posición más cercana es la (0,10). Por tanto colocamos la pieza 2 en la posición (0,10). Para la pieza 3 tenemos las siguientes localizaciones (esquinas) posibles: (0, 15), (10, 10),(20, 0). La posición de mínima distancia es (10, 10). Por tanto,*

$Tablero: L = 30\ W = 20$						
$Pieza$	l_i	w_i	P_i	Q_i	v_i	e_i
1	20	10	0	1	200	1
2	10	5	0	1	50	1
3	10	10	0	1	100	1
4	11	6	0	1	66	1
5	10	12	0	1	120	1



(a) Constructivo BL (b) Constructivo DP

Figura 5.3: *Algoritmos constructivos.*

colocamos la pieza 3 en (10,10). Si continuamos este procedimiento, para la pieza 4 no existiría ninguna posición posible y estudiamos la pieza 5 que colocaríamos en la posición (20,0). El diseño final para este procedimiento puede observarse en la Figura 5.3(b).

- Una método diferente para acomodar las cajas es propuesto por Wu et al. (2002)[91]. Es un procedimiento determinista en el que se intenta colocar las cajas en primer lugar en las esquinas y posteriormente en el centro. El algoritmo original es para el problema especial, donde $P_i = Q_i \ \forall \, i$ y se permite que las piezas sean rotadas, pero puede transformarse para el problema general que estamos considerando. El algoritmo va cortando las piezas de una en una, en una esquina del espacio vacío que queda en cada instante. Para elegir la pieza a cortar y la esquina en la cual cortarla desarrolla una función de evaluación que estima el beneficio que se obtendría al completar el corte a partir de dicha pieza.

5.1.3.2. Metaheurísticos

De los métodos metaheurísticos para el problema sólo uno es para el problema general, el propuesto por Beasley (2003)[12], mientras que el resto de métodos son para el caso restringido $P_i = 0 \ \forall i$ y sin pesos.

En 1997, Lai y Chan [56] presentan un procedimiento basado en *simulated annealing*. El procedimiento consiste en una lista con el orden de corte de las diferentes piezas, de tamaño M. Por ejemplo, supongamos $M = 5$. Una solución se representa por la lista $\{2, 3, 5, 4, 1\}$ lo que indicaría que la pieza 2 se corta en primer lugar, después la pieza 3, ..., y finalmente la pieza 1. En esta representación si una pieza puede cortarse más de una vez se crean Q_i copias de esa pieza. Posteriormente estas piezas son cortadas según el procedimiento *Difference Process*. Las soluciones se generan mediante intercambios de piezas en la lista. Para los resultados computacionales presentan un conjunto de problemas generados

aleatoriamente, en los cuales el objetivo es minimizar el desperdicio del tablero y se resuelven problemas de hasta $m \leq 35$.

También en 1997, Lai y Chan [55] describen un procedimiento evolutivo. Usan la misma representación que en el anterior trabajo. Su algoritmo incluye un procedimiento de mejora basado en dividir la lista ordenada en piezas activas (cortadas) e inactivas (no cortadas) y examinar si alguna de las piezas, actualmente, inactivas puede ser cortada. Se presenta resultados computacionales para un conjunto de problemas generados aleatoriamente, donde el objetivo es, como en el trabajo anterior, minimizar el desperdicio del tablero. El problema más grande que resuelven tiene m=10.

En 2001 Leung et al.[59] utilizan un algoritmo evolutivo, en el que usan la representación de las soluciones de Lai y Chan [56]. Muestran que existen patrones que no pueden alcanzarse por los procedimientos BL y DP. Presentan un simulated annealing con un movimiento que corresponde a intercambiar dos piezas en la lista o mover una pieza a una nueva posición de la lista. También presentan un algoritmo genético que incluye cinco tipos diferentes de operadores de cruce. Resuelven 8 problemas test, pero no presentan detalles de los resultados. Resuelven problemas con $m \leq 30$.

En 2003 Leung et al.[58] comparan un algoritmo genético frente a un método mixto formado por un algoritmo genético y el simulated annealing [59] del 2001 para el problema. La utilización de las dos técnicas de forma conjunta evita la convergencia prematura del método. El mayor problema que resuelven es con $m = 97$.

En 2003 Beasley[12] presenta un algoritmo genético para el caso general. El algoritmo está basado en una nueva formulación no lineal para el problema. La formulación también admite extensiones para el problema con más de un tablero, el problema con algunas zonas del tablero que no se pueden utilizar y el problema donde las piezas se pueden rotar. Los resultados computacionales son los mejores hasta el momento. Resuelve problemas con hasta 1000 tipos de piezas y recopila

la mayoría de instancias test utilizados para el problema. Además crea una nueva batería de problemas basados en los estudios de Fekete y Schepers[42].

5.2. Un algoritmo Constructivo

En esta sección, describimos un algoritmo *constructivo* que será usado como una etapa inicial en procedimientos más complejos.

El método que definimos para cortar las piezas está basado en la forma de colocar cajas que utilizábamos en el algoritmo Tabu del capítulo 4. Al igual que en el algoritmo Tabu vamos a trabajar con bloques de piezas, rectángulos formados por piezas del mismo tipo.

Seguimos un proceso iterativo en el cual combinamos dos elementos: una lista $\mathcal{P}$ de piezas que deben ser cortadas, inicialmente la lista completa de piezas, y una lista $\mathcal{L}$ de rectángulos vacíos en los cuales una pieza puede ser cortada, inicialmente el tablero completo. En cada paso, se elige un rectángulo de $\mathcal{L}$ y un tipo de pieza de $\mathcal{P}$ que quepa en él para ser cortada. Por lo general, el corte produce nuevos rectángulos que se añaden a $\mathcal{L}$ y el proceso continua hasta que $\mathcal{L} = \emptyset$ o ninguna de las restantes piezas de $\mathcal{P}$ cabe en alguno de los rectángulos de $\mathcal{L}$. Un esquema del algoritmo puede observarse a continuación.

Paso 1. *Inicialización:*
$\mathcal{L} = \{R\}$, el conjunto de rectángulos.
$\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$, el conjunto de piezas a cortar.
$\mathcal{C} = \emptyset$, el conjunto de piezas cortadas.

Paso 2. *Elección del rectángulo:*
Tomar R^*, el menor rectángulo de $\mathcal{L}$ en el cual puede cortarse una pieza de $\mathcal{P}$.
Si tal R^* no existe, *parar*.
En otro caso ir a *Paso 3*.

Paso 3. *Elección de las piezas a cortar:*

Elegir una pieza p_i y una cantidad $n_i \leq Q_i$, formando el bloque b^*, para cortarlo en R^*.

Elegir una posición en R^* para cortar b^*.

Modificar $\mathcal{P}$, $\mathcal{C}$ y Q_i que indica el número de piezas restantes por cortar.

Mover el bloque b^* hacia la esquina más cercana del tablero.

Paso 4. *Modificar la lista de rectángulos $\mathcal{L}$:*

Añadir a $\mathcal{L}$ los posibles rectángulos producidos cuando cortamos b^* en R^*.

Tener en cuenta los posibles cambios en $\mathcal{L}$ cuando movemos el bloque b^*.

Unir los rectángulos para intentar favorecer la posibilidad de cortar nuevas piezas de $\mathcal{P}$.

Ir a *Paso 2.*

Vamos a explicar con más detalle los procedimientos básicos del algoritmo.

Paso 1. Inicialización.

La lista $\mathcal{P}$ de piezas posibles se ordena desde el inicio de la siguiente forma, para facilitar los restantes procedimientos:

1. Ordenar por $P_i * l_i * w_i$, dando prioridad a las piezas que deben ser cortadas obligatoriamente.
2. Si hay un empate en (1) (por ejemplo, si $P_i = 0$, $\forall i$), ordenar por e_i, otorgando mayor prioridad a las piezas más eficientes.
3. Si hay un empate en (2) (por ejemplo, si $e_i = 1$ $\forall i$), ordenar por $l_i * w_i$, dando prioridad a las piezas grandes sobre las pequeñas.

Paso 2. Elección de un rectángulo de $\mathcal{L}$

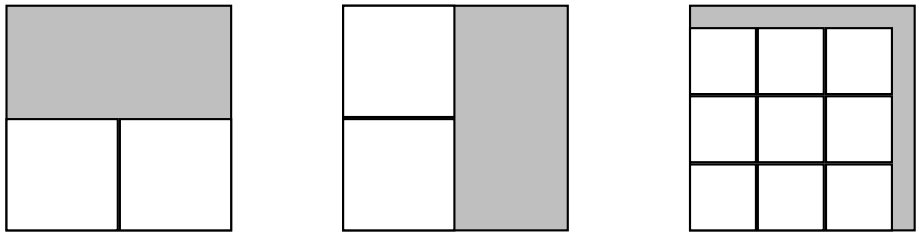
Dado que tenemos varios rectángulos para rellenar, debemos elegir cuál de ellos es el primero a rellenar. Elegimos el rectángulo de menor área, ya que se

intenta satisfacer la demanda de piezas pequeñas con rectángulos pequeños, dejando los rectángulos más grandes para piezas grandes. Si tomamos un rectángulo grande al principio y lo usamos para colocar piezas pequeñas, los rectángulos vacíos resultantes pueden ser menos útiles para las piezas grandes que quedan por colocar.

Si se produce un empate elegimos el rectángulo que esté más cercano a alguna de las cuatro esquinas del tablero. Se supone que los rectángulos del centro tendrán más posibilidades para juntarse con otros rectángulos vacíos.

Paso 3. Elección de las piezas a cortar

a) Elegir el tipo de pieza i y la cantidad $n_i \leq Q_i$, formando el bloque b^*.
Una vez elegido un rectángulo R^*, intentamos cortar la pieza i de $\mathcal{P}$ en R^*. Si $Q_i > 1$, consideramos la posibilidad de cortar más de una pieza del mismo tipo, en un *bloque*, un rectángulo conteniendo algunas copias de la pieza colocadas en filas o columnas, tal que el número de piezas del bloque no exceda Q_i. La forma más habitual será en una fila (Figura 5.4(a)) o en una columna (Figura 5.4(b)) pero también podemos colocarlas en bloques de más de una fila o columna (Figura 5.4(c)). Por notación diremos que un bloque es de tipo i, si contiene piezas del tipo i.



(a) Pieza 5 x 5, Q=2 (b) Pieza 5 x 5, Q=2 (c) Pieza 3 x 3, Q=9

Figura 5.4: *Diferentes alternativas para cortar piezas. Tablero 10 × 10.*

Se han considerado dos criterios para seleccionar el tipo de pieza y a su vez la cantidad de estas piezas:

- Elegir el primer tipo de pieza en la lista ordenada $\mathcal{P}$ y colocar tantas piezas como sea posible.
  Si $P_i = 0\ \forall i$, dado un rectángulo elegimos el tipo de pieza más eficiente que pueda colocarse dentro y colocamos tantas piezas de este tipo como sea posible.
- Elegir el bloque que proporcione un mayor incremento en la función objetivo, independientemente de su eficiencia.

b) Elegir una posición en R^* para colocar b^*.

Normalmente el bloque b^* no ocupa completamente el rectángulo R^* (ver Figura 5.4). Colocamos el bloque en la esquina del rectángulo R^* que se encuentre más cerca, según la distancia euclídea, de alguna de las esquinas del tablero. Por la manera en que colocamos el bloque dentro del rectángulo, pueden aparecer dos nuevos rectángulos vacíos como máximo.

c) Actualizar $\mathcal{P}$, $\mathcal{C}$ y Q_i.

Incluir en $\mathcal{C}$ las piezas cortadas de tipo i y la cantidad n_i.
Actualizar el número máximo de piezas que quedan por cortar $Q_i = Q_i - n_i$.
Si $Q_i = 0$, eliminar el tipo de pieza i de la lista $\mathcal{P}$.

d) Desplazar el bloque b^* hacia la esquina del tablero más cercana.

Este procedimiento permite que el bloque salga del rectángulo R^* en el cual fue colocado y entre de forma total o parcial en otro rectángulo vacío, localizado más cerca de la esquina del tablero. Al desplazar los bloques hacia las esquinas se intenta dejar las zonas vacías concentradas en el centro, para que puedan ser más fácilmente juntadas en el *Paso 4*.

Paso 4. Modificar la lista de rectángulos $\mathcal{L}$

a) Añadir a $\mathcal{L}$ los nuevos rectángulos.
Normalmente, la colocación de b^* en el rectángulo R^* y su posterior movimiento pueden producir nuevos rectángulos vacíos, que son añadidos a la lista $\mathcal{L}$.

b) Juntar los rectángulos vacíos $\mathcal{L}$, para favorecer cortar nuevas piezas de $\mathcal{P}$.
Aunque guardamos una lista de rectángulos vacíos, realmente tenemos una zona vacía poligonal, donde intentar acomodar las piezas restantes por cortar. Una forma de adaptar la flexibilidad del corte no-guillotina a la lista $\mathcal{L}$ es la posibilidad de juntar algunos rectángulos de la lista para producir otros rectángulos en los cuales se puedan cortar mejor las piezas que quedan pendientes.

Al igual que ocurría en el procedimiento Tabu el mecanismo de juntar rectángulos es fundamental para el algoritmo, ya que de él depende qué piezas podremos cortar posteriormente.

La unión de dos rectángulos pueden ocasionar, como mencionamos en la sección 4.1.1 (pág. 114), la aparición de tres nuevos rectángulos: uno que llamamos grande y dos más pequeños. Existen varias alternativas de unión, por lo que intentamos seleccionar la mejor alternativa, esto es, la que permita cortar las piezas mejor situadas en la lista ordenada $\mathcal{P}$. Con este objetivo, imponemos una serie de condiciones a la unión:

1. Si el menor orden de las piezas que quepan en el rectángulo grande es estrictamente menor que el menor orden de las piezas que cabían en los rectángulos originales, unimos estos rectángulos.
2. Si el menor orden de las piezas que quepan en el rectángulo grande es estrictamente más grande que el menor orden de las piezas que quepan en los rectángulos originales, no la juntamos.

3. Si el menor orden de las piezas que quepan en el rectángulo grande es igual al menor orden de alguna de las piezas que caben en los rectángulos originales, juntaremos estos rectángulos si el área del rectángulo grande es mayor que el área de cada uno de los rectángulos originales.

Debido al orden de piezas que hemos definido, la función de juntar rectángulos es idónea en todo tipo de problemas. Si tenemos un problema doblemente restringido, intentará juntar las pérdidas para colocar piezas que obligatoriamente deben estar en una solución. Si el problema no está restringido inferiormente o ya están cortadas todas las piezas obligatorias, se juntan las pérdidas para intentar cortar el tipo de pieza más eficiente.

Para los dos métodos de elección del tipo de pieza y cantidad del *Paso 3*(a), el mecanismo de juntar las pérdidas es el mismo. Esto provoca que cuando se utiliza el método de selección de mayor incremento en la función objetivo también se tiene en cuenta la eficiencia de las piezas.

Ejemplo 5.3 *Tenemos la instancia de la Figura 5.1 restringida con* $P_i = 0$. *Denotamos un rectángulo por una cuádrupla* (x_1, y_1, x_2, y_2), *donde* (x_1, y_1) *son las coordenadas de la esquina inferior izquierda y* (x_2, y_2) *de la esquina superior derecha. El algoritmo constructivo puede observarse en la Figura 5.5.*
Inicialmente, $\mathcal{L}$=$\{R\}$=$\{(0,0,10,10)\}$ *y* $\mathcal{P}$=$\{10,9,6,5,7,2,8,3,1,4\}$, *ordenada por eficiencia. Consideramos la primera pieza en la lista, la pieza 10, con* Q_{10}=*2 e intentamos colocar el mayor número de piezas en el primer rectángulo, dos en este caso. Hemos cortado un bloque compuesto por dos copias de la pieza 10 en la esquina inferior izquierda del rectángulo. Modificamos* $\mathcal{P}$=$\{9,6,5,7,2,8,3,1,4\}$ *y* $\mathcal{L}$=$\{(4,0,10,10)\}$. *Intentamos cortar la pieza 9, con* Q_9=*3, pero sólo 2 copias de ella caben en el rectángulo. Cortamos el bloque en la esquina inferior derecha. Modificamos* $\mathcal{P}$=$\{9,6,5,7,2,8,3,1,4\}$, Q_9=*1,* $\mathcal{L}$=$\{(4,7,10,10)\}$. *Consideramos las piezas 9, 6, 5 por turno, pero no caben en el rectángulo hasta que exploramos la pieza 7, con* Q_7=*2, que es la elegida. Colocamos un bloque con 2 copias en la esquina superior derecha. Modificamos* $\mathcal{P}$=$\{9,6,5,2,8,3,1,4\}$, $\mathcal{L}$=$\{(4,7,6,10),$

(6,7,10,8)}. Ninguna de las restantes piezas cabe en los rectángulos de $\mathcal{L}$. Antes de terminar el proceso, intentamos juntar los rectángulos de otra manera para producir nuevos rectángulos en los cuales se puedan acomodar alguna pieza restante de $\mathcal{P}$. En este caso, la única alternativa posible sería producir el par de rectángulos (4,7,10,8) y (5,7,6,10), que tampoco permiten cortar alguna pieza y el proceso termina.

Figura 5.5: *Instancia 3 de la Tabla 5.8. Constructivo más eficiente.*

5.3. Un algoritmo GRASP

El algoritmo *GRASP (Greedy Randomized Adaptive Search Procedure)* fue desarrollado a finales de 1980 por Feo y Resende (1989)[43] para resolver problemas de cubrimiento de conjuntos. Una introducción actualizada puede hallarse

en Ribeiro y Resende (2003)[74]. Cada iteración *GRASP* consiste en una fase *constructiva* y un procedimiento de *mejora*. La fase *constructiva* es iterativa ya que se construye la solución considerando los elementos de uno en uno, *greedy* porque la selección del elemento que se añade es guiada por una función *greedy*, con una cierta aleatorización, y adaptativa debido a que la función *greedy* toma en cuenta los elementos seleccionados previamente. La fase de *mejora* es usualmente un procedimiento de búsqueda local basado en intercambios.

5.3.1. Fase constructiva

En nuestro caso, la fase *constructiva* se sigue del algoritmo *constructivo* de la sección anterior, introduciendo aleatorización cuando seleccionamos el bloque a colocar. Sea s_i la puntuación del bloque con el tipo de pieza i con el criterio de selección que estemos usando y sea δ un parámetro a determinar ($0 < \delta < 1$). En lugar de seleccionar el bloque j de puntuación máxima elegimos al azar uno de un subconjunto de las mejores. Este subconjunto se denomina comúnmente *lista restringida de candidatos (LRC)*. Para este procedimiento estudiamos tres alternativas diferentes:

- *LRC-Porcentaje.*
  Seleccionar un bloque i de forma aleatoria del subconjunto formado por los $100(1-\delta)\,\%$ bloques mejores, independientemente de su puntuación s_i. Esta forma de crear el subconjunto se le conoce por lista restringida de candidatos por cardinalidad.

- *LRC-Valor.*
  Seleccionar un bloque i de forma aleatoria del subconjunto:

$$S = \{j | s_j \geq \delta s_{max}\}$$

- *Probabilidad.*
  Una estrategia en la cual todos los bloques de $\mathcal{P}$ pueden ser seleccionados pero con probabilidad p_i proporcional a su puntuación s_i ($p_i = s_i / \sum s_j$).

5.3.2. Elección del parámetro δ

Experimentos preliminares muestran que no existe un valor particular para el cual el parámetro δ sea el mejor para todas las instancias. Por tanto, hemos considerado estrategias en las que el parámetro δ varía de forma aleatoria o sistemática en algún conjunto de valores. Concretamente hemos estudiado las siguientes alternativas:

- En cada iteración, elegir δ de forma aleatoria del intervalo [0.4,0.9].
- En cada iteración, elegir δ aleatorio del intervalo [0.25,0.75].
- Cada iteración δ toma por turno uno de estos cinco valores: {0.5, 0.6, 0.7, 0.8, 0.9}.
- Fijar δ al valor 0.75.
- También examinamos otra estrategia, llamada *Reactive Grasp*, en la que el parámetro δ es ajustado de acuerdo a la calidad de las soluciones obtenidas con un conjunto inicial de posibles valores. El algoritmo que proponemos (Figura 5.6) es una modificación del algoritmo de Prais y Ribeiro (2000)[73] descrita por Delorme et al. (2003)[25].

  Partiendo de una distribución uniforme discreta del conjunto $\mathcal{D}$ de posibles valores, la probabilidad de escoger un valor δ, $prob_\delta$, es modificada periódicamente, cada cierto número de iteraciones, 100 en este caso. Las nuevas probabilidades se calculan a partir del valor medio de las soluciones obtenidas para cada valor de δ. La probabilidad de cada valor depende de la calidad de las soluciones obtenidas con ese valor. Se introduce un parámetro α para atenuar las probabilidades de los nuevos valores. El valor de α, como indican Prais y Ribeiro [73], lo establecemos a 10.

Inicialmente:

$\mathcal{D}$= {0.1, 0.2, 0.3 ,. . . , 0.8, 0.9}.

$S_{best} = 0$, $S_{worst} = \infty$.

n_{δ^*} = número de iteraciones con δ^*, $\forall \delta^* \in \mathcal{D}$.

$sum_{\delta^*} = 0$, suma del valor de la soluciones con δ^*, $\forall \delta^* \in \mathcal{D}$.

$P(\delta = \delta^*) = prob_{\delta^*} = (1/|\mathcal{D}|),\ \forall\, \delta^* \in \mathcal{D}$.

Repetir mientras $(numIter < maxIter)$

{

Elegir aleatoriamente δ^* de $\mathcal{D}$ con $prob_\delta$.

$n_{\delta^*} \leftarrow n_{\delta^*} + 1$.

$numIter \leftarrow numIter + 1$

$S \leftarrow$ Construir Solución Heurístico Greedy con δ^*.

$S' \leftarrow$ Búsqueda Local a partir de S.

Si $S' > S_{best}$ entonces $S_{best} = S'$.

Si $S' < S_{worst}$ entonces $S_{worst} = S'$.

$sum_{\delta^*} \leftarrow sum_{\delta^*} + S'$.

Si $mod(numIter, 100) == 0$ (cada 100 iteraciones):

$$eval_\delta \leftarrow \left(\frac{media_\delta - S_{worst}}{S_{best} - S_{worst}} \right)^\alpha, \forall \delta \in \mathcal{D}$$

$$prob_\delta \leftarrow \frac{eval_\delta}{\left(\sum_{\delta' \in \mathcal{D}} eval_{\delta'} \right)}, \forall \delta \in \mathcal{D}$$

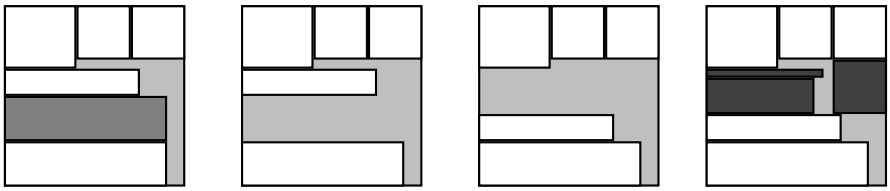
}

Figura 5.6: *Algoritmo Reactive Grasp.*

5.3.3. Fase de mejora

Cada solución de la fase constructiva es el punto de partida para una búsqueda local en la cual intentamos mejorar la solución. Hemos estudiado tres alternativas:

I) El primer método de mejora está basado en un procedimiento de intercambio. Consiste en considerar todos los bloques de piezas que tengan al lado una pérdida y estudiar si es posible mejorar la solución actual reduciendo un bloque o eliminándolo completamente (Figura 5.7). Al eliminar estas piezas de la solución, desplazamos las restantes piezas hacia las esquinas, intentamos juntar las pérdidas y las rellenamos con el algoritmo constructivo anterior. Si el movimiento mejora la solución se realiza el cambio y se continúan examinando las restantes piezas. Sólo intentamos reducir bloques que permitan cumplir las restricciones de demanda.

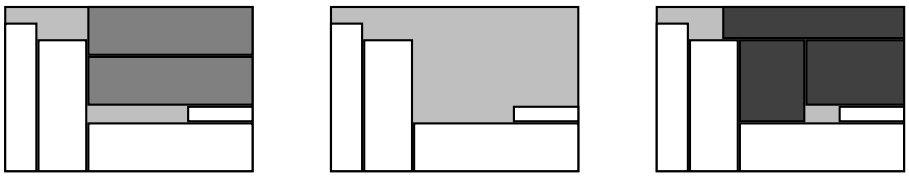


(a) Selección del bloque (b) Disminución (c) Desplazar esquinas (d) Rellenar

Figura 5.7: *Procedimiento de mejora I. Instancia 8 de la Tabla 5.8.*

II) El segundo método está basado en el procedimiento de intercambio del apartado anterior, pero en éste se simplifican algunos pasos del procedimiento anterior para que sea más rápido, aunque obviamente perdiendo algo de calidad. No desplazamos los bloques hacia las esquinas cuando quitamos una pieza, solamente juntamos el nuevo rectángulo con las pérdidas colindantes. Asimismo se intenta rellenar el nuevo hueco solamente en el caso en que podamos colocar una pieza de igual o mayor orden (con el orden

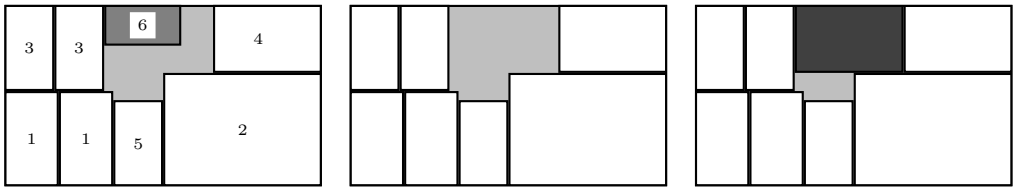
definido al inicio) en la nueva zona de pérdida. Un ejemplo de este método puede observarse en la Figura 5.8.



(a) Selección del bloque (b) Disminución (c) Rellenar

Figura 5.8: *Procedimiento de mejora II. Instancia 6 de la Tabla 5.8.*

III) El tercer método consiste en extraer los últimos k por ciento bloques de la solución (donde k=10) e intentar llenar los nuevos huecos con el procedimiento constructivo sin aleatoriedad. Esta técnica fue propuesta por Beltrán et al.(2002)[13]. Una vez hemos eliminado las últimas piezas de la solución, desplazamos los bloques hacia las esquinas, se unen los rectángulos de pérdidas y se rellenan las pérdidas con el algoritmo constructivo (ver Figura 5.9, en la que los números en las piezas reflejan el orden en el que se han incluido en la solución).



(a) Selección del bloque (b) Disminución (c) Rellenar

Figura 5.9: *Procedimiento de mejora III. Instancia 15 de la Tabla 5.8.*

5.4. Ajustando las cotas de cada pieza

A lo largo de las iteraciones del algoritmo tenemos almacenada la mejor solución encontrada hasta el momento, de valor v_{best}. Esta información la podemos utilizar en dos direcciones, para aumentar las cotas inferiores P_i de algunos tipos de piezas que deben aparecer necesariamente si deseamos conseguir una solución mejor y para reducir las cotas superiores Q_i de algunos tipos de piezas cuya inclusión no permite producir una solución mejor.

5.4.1. Aumentar cotas inferiores

Supongamos que la mejor solución obtenida hasta el momento tiene valor v_{best} y que $\exists\, i | P_i < Q_i$. Definimos $total_{piezas} = \sum_{i}^{m} v_i * Q_i$ como el valor de todas las piezas posibles para colocar en el tablero. Si tenemos un tipo de pieza i que cumple: $P_i < Q_i$ y $total_{piezas} - (Q_i - P_i) * v_i <= v_{best}$, de este tipo de pieza alguna debe aparecer obligatoriamente en la solución para poder alcanzar una solución de mayor valor.

Podemos actualizar el valor de P_i de la siguiente forma:

$$max\, t: \quad total_{piezas} - t * v_i > v_{best}, \quad t \geq 0,\, t \leq Q_i - P_i \tag{5.13}$$

Ajustamos el valor de $P_i = Q_i - t$. La mejora en la cota inferior puede ser útil para la fase constructiva, en la que las piezas con $P_i > 0$ son cortadas primero, y en la fase de mejora, en la que las piezas en su cota inferior no son consideradas para eliminarlas de la solución.

Ejemplo 5.4 *Tenemos la instancia de la Tabla 5.1. Supongamos que tenemos una solución de valor 84. Tenemos* $total_{piezas} = 102$ *y por tanto, del tipo 1 podemos ajustar* $P_1 = 3$ *y para las de tipo 2 se puede ajustar al valor* $P_2 = 1$.

$Tablero: L = 10\ W = 10$						
$Pieza$	l_i	w_i	P_i	Q_i	v_i	e_i
1	5	5	0	3	25	1
2	3	3	0	3	9	1

Tabla 5.1: *Ejemplo 5.1*

5.4.2. Disminuir cotas superiores

Si tenemos una instancia donde la eficiencia de todas las piezas no es la misma, podemos disminuir el número máximo Q_i de algunos tipos de piezas i que no pueden producir una solución de valor mayor a v_{best}.

Sea $R = \sum_{P_i>0} P_i * l_i * w_i$ el área de las piezas que deben estar en cualquier solución posible, $R_v = \sum_{P_i>0} P_i * l_i * v_i$ el valor de esas piezas y e_{max} la máxima eficiencia de las piezas. Si pudiéramos llenar todo el tablero con piezas de eficiencia máxima tendríamos una cota superior $e_{max} * (L * W - R) + R_v$. Si tenemos un tipo de pieza i con $Q_i > P_i$ y $e_i < e_{max}$ que cumple:

$$(Q_i * l_i * w_i * (e_{max} - e_i) \geq e_{max} * (L * W - R) + R_v - v_{best} \tag{5.14}$$

entonces una solución con Q_i piezas de tipo i no puede producir una solución de valor mayor que la mejor obtenida hasta ese momento. Por tanto, podemos reducir el valor de Q_i, de la siguiente forma:

$$max\, t: \quad t*l_i*w_i*(e_{max}-e_i) < e_{max}*(L*W-R)+R_v-v_{best} \quad t \geq 0,\, t \leq Q_i - P_i \tag{5.15}$$

Podemos ajustar $Q_i = P_i + t$. Esta rebaja en la cota superior es muy útil tanto en la fase constructiva como en la fase de mejora. En algunos casos podremos hacer $Q_i = 0$, con lo que dejaríamos de considerar este tipo de pieza para futuras iteraciones.

$Tablero: L = 10\ W = 10$						
$Pieza$	l_i	w_i	P_i	Q_i	v_i	e_i
1	5	5	0	4	75	3
2	5	5	0	3	50	2
3	5	5	0	3	25	1

Tabla 5.2: *Ejemplo 5.2*

Ejemplo 5.5 *Tenemos la instancia de la Tabla 5.2. Supongamos que tenemos una solución de valor 250. Entonces podemos ajustar* $Q_2 = 1$ *y* $Q_3 = 0$.

5.5. Resultados computacionales

Nuestro algoritmo ha sido codificado en *C++* y ejecutado sobre un ordenador personal con un procesador *Pentium III* a *800 Mhz*.

5.5.1. Instancias Utilizadas

Para verificar la eficiencia del algoritmo hemos utilizado varios conjuntos de problemas test. Podemos diferenciar cuatro grupos de problemas.

- Un primer conjunto de 21 problemas test formado por instancias de diferentes autores y que son clásicas para el problema, compuesto por doce problemas de Beasley (1985)[9], dos de Hadjiconstantinou y Christofides (1995)[46], uno de Wang (1983)[86], uno de Christofides y Whitlock (1977)[22] y cinco más de Fekete y Schepers (1997)[42].

- Un conjunto de problemas con un gran número de cajas. Está formado por 630 problemas generados por Beasley [12] según los estudios de Fekete y Schepers [42] para los que el óptimo no es conocido. Hemos obtenido

una cota mediante la transformación a un problema mochila acotado, que explicamos en la subsección 5.1.2 (pág. 143).

Los 630 problemas incluyen un tablero (L, W) de tamaño (100,100). Para cada valor de m considerado (donde $m = 40$; 50; 100; 150; 250; 500; 1000) 10 problemas generados aleatoriamente con $P_i = 0$, $Q_i = Q^*$, $\forall i = 1, \ldots, m$ donde se considera $Q^* = 1; 3; 4$. Estos 630 problemas se dividen en tres tipos de problemas basados en los porcentajes de piezas de cada clase:

Clase	*Descripción*	*Longitud*	*Anchura*
1	Alargada a lo ancho	[1,50]	[75,100]
2	Alargada a lo largo	[75,100]	[1,50]
3	Grandes	[50,100]	[50,100]
4	Pequeñas	[1,50]	[1,50]

Tipo	Porcentaje de piezas de cada clase			
	1	2	3	4
1	20	20	20	40
2	15	15	15	55
3	10	10	10	70

Por ejemplo, un problema del tipo I tiene un 20 % de piezas de clase 1, un 20 % de piezas de clase 2, un 20 % de piezas de clase 3 y el 40 % restante de la clase 4.

Para estas 630 instancias el valor asignado a cada pieza es igual a su área multiplicada por un número entero elegido aleatoriamente del conjunto $\{1, 2, 3\}$.

- Además, Beasley transforma los 21 problemas del primer conjunto en problemas doblemente restringidos obligando a colocar algunas piezas. Para cada tipo de pieza, desde $i = 1, \ldots, m$ que cumple:

$$\sum_{j=1, j\neq i}^{m} (l_j w_j) P_j + l_i w_i \leq (LW)/3 \quad \text{fija } P_i = 1. \tag{5.16}$$

Se obliga a colocar una pieza de cada tipo mientras que la suma del área de las piezas obligatorias no exceda de un tercio del área total del tablero.

- Otro conjunto de problemas test tomados del trabajo de Leung et al. [58] consistentes en tres instancias de Lai y Chan[55] (en la segunda instancia hay un errata en el artículo), cinco de Jakobs[53], y dos más de los propios Leung et al.[58]. En estos problemas el valor de cada pieza es su área, son instancias sin pesos y el objetivo es minimizar el desperdicio del tablero. Estas instancias se han creado cortando el tablero aleatoriamente en rectángulos y la solución óptima se obtiene con un diseño de desperdicio cero.

Hemos incluido el conjunto de problemas de Leung et al. [58] porque sus características pueden ser consideradas complementarias de los dos primeros conjuntos usados por Beasley, como podemos observar en la Tabla 5.3 donde se muestran los ratios del total de piezas disponibles para ser cortadas y la cota superior de piezas que se pueden cortar del tablero. Podemos ver que los problemas del segundo conjunto, Tipos I, II, III pueden ser considerados *problemas de selección* porque existen muchas piezas disponibles y sólo una pequeña fracción de ellas puede formar parte de la solución. Sin embargo, los problemas de Leung son *problemas rompecabezas.* Casi todas las piezas disponibles deben formar parte de la solución y la dificultad es buscar una posición correcta en el patrón de corte. Un método que funciones bien en los dos tipos de problemas puede considerarse una algoritmo de propósito general.

Los tres primeros conjuntos de problemas test están disponibles públicamente en la *OR-library* en la siguiente dirección:

$$http://mscmga.ms.ic.ac.uk/jeb/orlib/ngcutinfo.html$$

Conjunto de Problemas	Media	
	Total valor piezas/ Cota superior	Total superficie piezas/ Cota Superior superficie
Problemas de literatura	3,13	3,61
Tipo I	123,69	185,60
Tipo II	101,69	152,71
Tipo III	79,67	119,20
Leung	1,01	1,01

Tabla 5.3: *Problemas Test – Propiedades.*

5.5.2. Ajuste de parámetros

Debemos en primer lugar ajustar los parámetros y las elecciones correspondientes al heurístico *GRASP*. Para estudiar el empleo de una u otra opción hemos trabajado con los conjuntos de problemas, primero, segundo y cuarto que hemos denotado como *Literatura, Grandes y Leung et al.*, respectivamente.

5.5.2.1. Criterio de selección de las piezas

En el algoritmo constructivo (*Paso 2*), hemos considerado dos estrategias para elegir las piezas a colocar en cada rectángulo de los dos tipos posibles.

- $Eficiencia$: Elegir el bloque del tipo de pieza más eficiente.
- $Valor$: Elegir el bloque que produzca mayor incremento en la función objetivo.

Los resultados de cada criterio se observan en la Tabla 5.4. Aparte de los problemas de Leung et al., en los que obviamente los resultados coinciden porque la eficiencia de todas las piezas es 1, los resultados para el criterio de *eficiencia* son ligeramente mejores que para el criterio de *valor*.

	Literatura		Grandes		Leung et al.	
	% Media desviación óptimo	Número óptimos	% Media desviación cota	Número óptimos	% Media desviación óptimo	Número óptimos
Eficiencia	8,850	6	2,929	28	8,080	1
Valor	10,172	7	3,837	28	8,080	1

Tabla 5.4: *Resultados Computacionales – Métodos constructivos.*

5.5.2.2. Métodos de aleatorización

Al no existir grandes diferencias entre los dos criterios de selección de la pieza, estudiamos los dos dentro de un constructivo aleatorizado con 1000 iteraciones y fijando un valor de δ=0.5 (Tabla 5.5).

Estudiamos tres posibilidades para la selección aleatoria de las piezas:

- Una lista en la cual cada elemento tiene una probabilidad proporcional a su valor de ser elegida (*Probabilidad*).
- Lista restringida de candidatos formada por los $(1-\delta)100\,\%$ de los mejores (*LRC-Porcentaje*).
- Lista restringida de candidatos formada por los candidatos que estén δ veces por encima del mejor valor (*LRC-Valor*).

En el algoritmo constructivo el rectángulo a cortar se elegía de forma determinista. Concretamente siempre tomábamos el menor de los rectángulos existentes. En el algoritmo *Grasp* vamos a estudiar la posibilidad de aleatorizar también la elección del rectángulo a cortar, introduciendo una mayor flexibilidad en el procedimiento. En la Tabla 5.5 se muestran los resultados para todas las alternativas consideradas.

La combinación que mejores resultados proporcionó (Tabla 5.5) fue elegir el rectángulo de forma aleatoria, elegir la pieza por valor y usar como método de aleatorización *LRC-Valor*.

		Literatura		Grandes		Leung et al.	
		% Media desviación óptimo	Número óptimos	% Media desviación cota	Número óptimos	% Media desviación óptimo	Número óptimos
Elegir el rectángulo de menor área							
Eficiencia	Probabilidad	0,795	11	2,533	37	5,296	3
	LRC-Porcentaje	0,782	13	1,987	19	2,880	0
	LRC-Valor	1,594	13	2,649	14	5,809	0
Valor	Probabilidad	0,800	14	1,900	47	4,460	2
	LRC-Porcentaje	0,835	15	3,076	26	3,430	2
	LRC-Valor	0,874	12	1,368	142	4,154	1
Elegir el rectángulo de forma aleatoria							
Eficiencia	Probabilidad	1,027	15	2,445	15	3,727	1
	LRC-Porcentaje	0,823	11	1,882	11	5,748	1
	LRC-Valor	1,212	13	2,492	13	6,282	1
Valor	Probabilidad	0,591	15	1,678	58	3,949	2
	LRC-Porcentaje	1,163	15	2,992	16	3,561	2
	LRC-Valor	0,597	14	1,223	157	3,999	2

Tabla 5.5: *Resultados Computacionales – Elección del rectángulo, pieza y aleatorización.*

5.5.2.3. Estudio del parámetro δ

Para la elección del delta, probamos las siguientes alternativas:

- Elegir un δ aleatorio entre [0.4, 0.9].
- Elegir un δ aleatorio entre [0.25, 0.75].
- Mover el δ de manera determinista entre: 0,5, 0,6, 0,7, 0,8, 0,9.
- Un valor $\delta = 0{,}75$ fijo para todo el algoritmo.
- La solución propuesta por el Reactive Grasp.

Los mejores resultados se obtuvieron con la estrategia del Reactive Grasp, aunque no muy lejos de la alternativa de elegir delta aleatorio entre [0.25 y 0.75], como observamos en la Tabla 5.6. Estas dos estrategias son usadas para las restantes pruebas.

	Literatura		Grandes		Leung et al.	
	% Media desviación óptimo	Número óptimos	% Media desviación cota	Número óptimos	% Media desviación óptimo	Número óptimos
Aleatorio [0.4,0,9]	0,428	15	1,195	177	3,207	2
Aleatorio [0.25,0.75]	0,334	15	1,166	179	3,618	2
Determinista de 0.4 a 0.9	0,498	14	1,254	162	3,019	3
Fijo 0,75	1,647	11	1,658	168	3,214	2
Reactive	0,216	17	1,194	194	3,061	2

Tabla 5.6: *Resultados Computacionales – Estudio del parámetro delta.*

5.5.2.4. Fase de mejora

Probamos los tres procedimientos de mejora:

- Método de mejora I: se basa en un intercambio de manera exhaustiva.

- Método de mejora II: se realizan intercambios pero simplificando todo lo posible el método de mejora anterior.
- Método de mejora III: eliminamos el 10 % de los últimos bloques colocados y rellenamos con el procedimiento constructivo sin aleatorizar.

El método de mejora I requiere mucho más tiempo que los otros dos métodos. Por tanto, para los otros dos métodos se han realizado más iteraciones (10000) para que los tiempos sean equivalentes. Los mejores resultados son los obtenidos por la método de mejora III, como observamos en la Tabla 5.7.

	Literatura		Grandes		Leung et al.	
	% Media desviación óptimo	Número óptimos	% Media desviación cota	Número óptimos	% Media desviación óptimo	Número óptimos
Aleatorio [0.25,0.75]						
Método de mejora I.	0,279	15	1,098	194	2,801	2
Método de mejora II.	0,205	17	1,055	220	2,235	2
Método de mejora III.	0,205	17	1,054	224	2,173	2
Reactive						
Método de mejora I.	0,239	17	1,118	197	2,247	2
Método de mejora II.	0,230	17	1,077	217	2,077	2
Método de mejora III.	0,188	18	1,072	210	2,054	2

Tabla 5.7: *Resultados Computacionales – Procedimientos de mejora.*

5.5.3. Resultados computacionales finales

Como consecuencia de los resultados obtenidos en las subsecciones anteriores, los parámetros utilizados para el algoritmo GRASP definitivo quedarían de la siguiente forma:

- Selección de la pieza: La de mayor incremento en la función objetivo.
- Selección del rectángulo: Aleatorio.
- Procedimiento de aleatorización: Lista restringida de candidatos por valor.
- Selección de δ: Reactive GRASP.
- Procedimiento de mejora: Método III.
- Numero de iteraciones: 10000 iteraciones.

Los resultados computacionales finales aparecen en las Tablas 5.8, 5.9, 5.10. Las primeras dos tablas incluyen una comparación directa con los resultados de Beasley[12] en términos de la calidad de las soluciones. Los tiempos no pueden ser comparados directamente. Beasley utilizaba un Silicon Graphics O2 workstation (R10000 chip, 225MHz, 128 MB). Para poder comparar los tiempos computacionales hemos utilizado la comparación de rendimientos de la página: *http* : *//www.spec.org*, que muestra que nuestro ordenador es aproximadamente dos veces más lento que el utilizado por Beasley. Por tanto, podemos decir que nuestro algoritmo mejora los resultados de Beasley en cada tipo de problema con menores tiempos de computación. En ambos algoritmos el criterio de parada es alcanzado cuando hemos conseguido la solución óptima, o un límite de iteraciones (10000 iteraciones en nuestro algoritmo, 75000 individuos en el algoritmo genético de Beasley).

La comparación directa con Leung et al. [58] no es posible. Por una parte, no proporcionan los tiempos de computación de su algoritmo. Por otra parte, proponen dos versiones de su algoritmo, cada una de ellas con un ratio de mutación,y muestran el mínimo y la media de desperdicio en 15 ejecuciones del algoritmo con 30000 iteraciones. Lo único que podemos decir es que las distancias medias al óptimo de nuestro algoritmo son comparables a las de los suyos. También podemos hacer notar que algunos patrones son imposibles para su algoritmo, situación que no sucede con nuestro procedimiento.

Finalmente, en la Tabla 5.11 se muestran los resultados del algoritmo GRASP para el conjunto de problemas test doblemente restringidos. La cota superior corresponde a la solución del problema restringido. Los problemas para los que el algoritmo no encuentra solución son problemas no posibles, pero Beasley los mantiene en el conjunto de problemas test y por lo tanto se han incluido.

El ajuste de las cotas inferiores no tiene efectos significativos para el funcionamiento del algoritmo, pero el ajuste de las cotas superiores tiene un gran efecto, especialmente en los problemas *Grandes* ya que existen grandes diferencias para la eficiencia de las piezas. Por ejemplo, para problemas con $m = 1000$ tipos de piezas, más del 65 % de las piezas son descartadas tan pronto encontramos buenas soluciones.

En último lugar, mostramos los diseños óptimos para un problema del primer conjunto (Figura 5.10), un problema del cuarto conjunto (Figura 5.11), y para dos de los considerados grandes de Tipo III (Figura 5.12, 5.13) con $m = 1000$ tipos de piezas y Q^*=4.

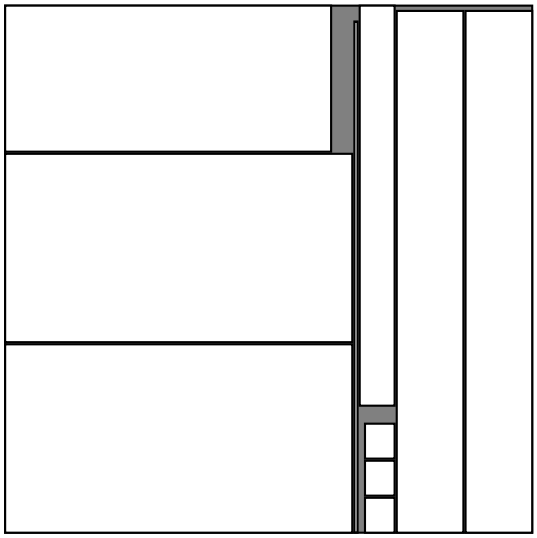


Figura 5.10: *Solución óptima de la instancia 20 de la Tabla 5.8.*

Origen del problema	I	Tamaño del problema (L,W)	m	M	Constructivo eficiencia	Constructivo aleatorizado	Grasp	Solución Beasley	Solución óptima	Tiempo (segundos) Grasp	Tiempo (segundos) Beasley
Beasley [9]	1	(10, 10)	5	10	146	164	164	164	164	0,00	0,02
	2	(10, 10)	7	17	213	230	230	230	230	0,00	0,16
	3	(10, 10)	10	21	220	247	247	247	247	0,00	0,53
	4	(15, 10)	5	7	268	268	268	268	268	0,00	0,01
	5	(15, 10)	7	14	358	358	358	358	358	0,00	0,11
	6	(15, 10)	10	15	268	289	289	289	289	0,00	0,43
	7	(20, 20)	5	8	430	430	430	430	430	0,00	0,01
	8	(20, 20)	7	13	753	831	834	834	834	0,77	3,25
	9	(20, 20)	10	18	863	924	924	924	924	0,00	2,18
	10	(30, 30)	5	13	1452	1452	1452	1452	1452	0,00	0,03
	11	(30, 30)	7	15	1524	1688	1688	1688	1688	0,05	0,6
	12	(30, 30)	10	22	1389	1865	1865	1801	1865	0,05	3,48
Hadjiconstantinou	1	(30, 30)	7	7	1178	1178	1178	1178	1178	0,00	0,03
y Christofides [46]	2	(30, 30)	15	15	1270	1270	1270	1270	1270	0,00	0,04
Wang[86]	1	(70, 40)	19	42	2277	2726	2726	2721	2726	0,77	6,86
Christofides [22]	12	(40, 70)	20	62	1560	1860	1860	1720	1860	0,39	8,63
y Whitlock											
Fekete y Schepers [42]	1	(100, 100)	15	50	18384	27589	27589	27486	27718	2,31	19,71
	2	(100, 100)	30	30	19790	21976	21976	21976	22502	4,17	13,19
	3	(100, 100)	30	30	23282	23743	23743	23743	24019	3,68	11,46
	4	(100, 100)	33	61	30197	32893	32893	31269	32893	0,00	32,08
	5	(100, 100)	29	97	25650	27923	27923	26332	27923	0,00	83,44
Media del porcentaje de desviación del óptimo					8,85 %	0,22 %	0,19 %	1,21 %		0,58	8,87
Número de óptimos					6	17	18	13			

Tabla 5.8: *Resultados Computacionales – Problemas de la Literatura.*

Media del porcentaje de desviación de la cota superior

m	Q*	M	Constructivo eficiencia	Constructivo aleatorizado	Grasp	Solución Beasley	Tiempo (segundos) Grasp	Tiempo (segundos) Beasley
40	1	40	10,81	7,27	6,97	7,77	2,33	13,57
	3	120	5,47	2,55	2,22	3,54	6,62	47,43
	4	160	4,38	1,99	1,81	3,24	4,44	63,30
50	1	50	8,90	4,91	4,80	5,48	4,71	14,60
	3	150	3,69	1,69	1,50	2,35	7,05	59,27
	4	200	4,04	1,41	1,18	2,63	5,34	80,07
100	1	100	4,44	1,75	1,51	2,26	5,36	27,20
	3	300	2,24	0,60	0,47	1,27	9,41	119,47
	4	400	1,96	0,34	0,26	1,06	6,99	175,10
150	1	150	3,66	1,13	0,89	1,31	5,53	40,60
	3	450	1,61	0,21	0,14	0,60	11,71	190,53
	4	600	1,48	0,22	0,11	0,92	6,75	323,83
250	1	250	2,41	0,66	0,51	0,88	5,27	76,70
	3	750	1,07	0,12	0,04	0,57	13,89	439,47
	4	1000	1,01	0,04	0,03	0,39	6,65	693,67
500	1	500	0,91	0,09	0,05	0,26	3,24	203,10
	3	1500	0,62	0,01	0,00	0,18	12,24	1210,80
	4	2000	0,60	0,01	0,00	0,18	1,15	1790,83
1000	1	1000	0,85	0,03	0,00	0,09	1,01	667,23
	3	3000	0,74	0,00	0,00	0,07	6,53	3318,47
	4	4000	0,51	0,00	0,00	0,07	0,29	4840,57
Tipo 1			3,01	1,15	1,04	1,64	5,13	558,11
Tipo 2			2,88	1,25	1,14	1,70	5,90	668,41
Tipo 3			2,90	1,18	1,03	1,66	7,28	830,02
Todos			2,93	1,19	1,07	1,67	5,91	685,51

Tabla 5.9: *Resultados Computacionales – Problemas Grandes.*

Origen del Problema	I	Tamaño del problema (L,W)	m	M	Constructivo eficiencia	Constructivo aleatorizado	Grasp	Solución óptima	Tiempo Grasp
Lai y Chan[55]	1	(400,200)	9	10	80000	80000	80000	80000	0,00
	2	(400,200)	7	15	75000	79000	79000	79000	0,00
	3	(400,400)	5	20	143500	149800	154600	160000	4,12
Jakobs[53]	1	(70,80)	14	20	4695	5400	5447	5600	10,16
	2	(70,80)	16	25	5055	5295	5455	5600	15,44
	3	(120,45)	22	25	4968	5310	5328	5400	12,57
	4	(90,45)	16	30	3717	3978	3978	4050	10,28
	5	(65,45)	18	30	2649	2844	2871	2925	14,94
Leung et al.[58]	1	(150,110)	40	40	15400	15668	15856	16500	90,52
	2	(160,120)	50	50	17816	18440	18628	19200	132,26
Media del porcentaje de desviación del óptimo					8,08	3,06	2,05		29,03

Tabla 5.10: *Resultados Computacionales – Problemas de Leung et al.*

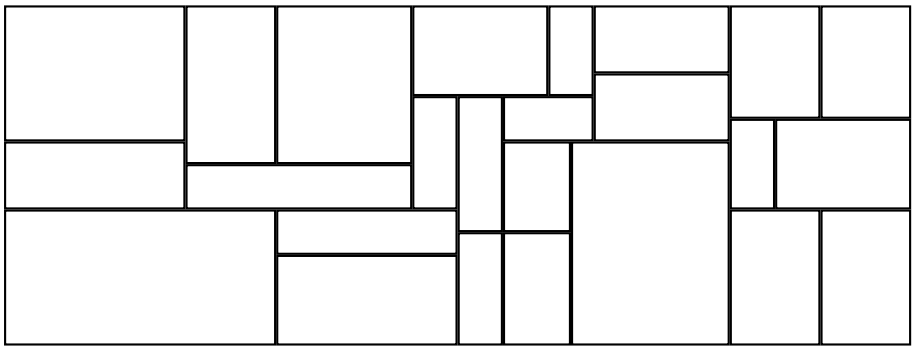


Figura 5.11: *Solución óptima de la instancia 6 de la Tabla 5.10.*

Origen del problema	I	Tamaño del problema			Constructivo	Constructivo	Grasp	Solución	Cota	Tiempo (segundos)	
		(L,W)	m	M	eficiencia	aleatorizado		Beasley	superior	Grasp	Beasley
Beasley [9]	1	(10, 10)	5	10	156	164	164	164	164	0,00	0,02
	2	(10, 10)	7	17	n/p	225	225	225	230	0,71	5,53
	3	(10, 10)	10	21	176	220	220	220	247	1,21	7,85
	4	(15, 10)	5	7	268	268	268	268	268	0,00	0,01
	5	(15, 10)	7	14	301	301	301	301	358	0,72	5,05
	6	(15, 10)	10	15	229	249	252	265	289	1,81	6,81
	7	(20, 20)	5	8	430	430	430	430	430	0,00	0,01
	8	(20, 20)	7	13	712	819	819	819	834	1,32	6,54
	9	(20, 20)	10	18	552	924	924	924	924	0,00	5,64
	10	(30, 30)	5	13	n/p	n/p	n/p	n/p	n/p	0,22	2,38
	11	(30, 30)	7	15	1132	1518	1518	1505	1688	1,59	2,96
	12	(30, 30)	10	22	1443	1648	1648	1666	1865	1,65	3,78
Hadjiconstantinou	1	(30, 30)	7	7	1178	1178	1178	1178	1178	0,00	0,25
y Christofides [46]	2	(30, 30)	15	15	1216	1216	1216	1216	1270	2,08	2,6
Wang[86]	1	(70, 40)	19	42	2180	2587	2700	2499	2726	1,48	6,36
Christofides [22]	12	(40, 70)	20	62	1340	1720	1720	1600	1860	0,88	6,81
y Whitlock											
Fekete y Schepers [42]	1	(100, 100)	15	50	n/p	24869	24869	25373	27718	3,73	11,86
	2	(100, 100)	30	30	n/p	18078	19083	17789	22502	3,02	5,8
	3	(100, 100)	30	30	n/p	n/p	n/p	n/p	n/p	0,66	4,03
	4	(100, 100)	33	61	25973	27665	27898	27556	32893	2,80	20,42
	5	(100, 100)	29	97	n/p	21899	22011	21997	27923	3,30	18,41
Media del porcentaje de desviación de la cota superior						7,93	7,36	8,11			

n/p: No encuentra solución posible

Tabla 5.11: *Resultados Computacionales – Problemas doblemente restringidos.*

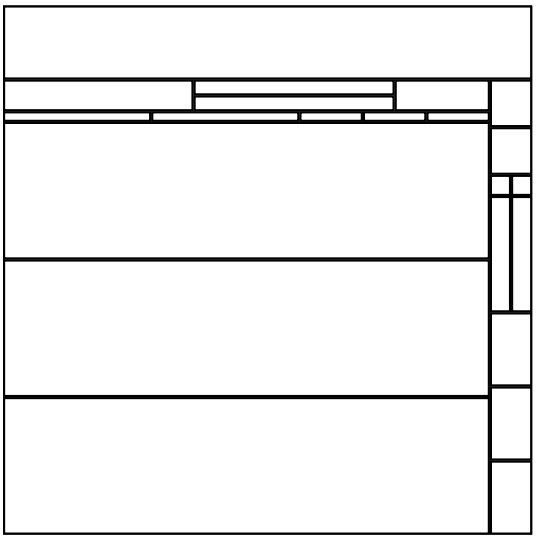


Figura 5.12: *Soluciones óptimas de la instancia tfs3200.*

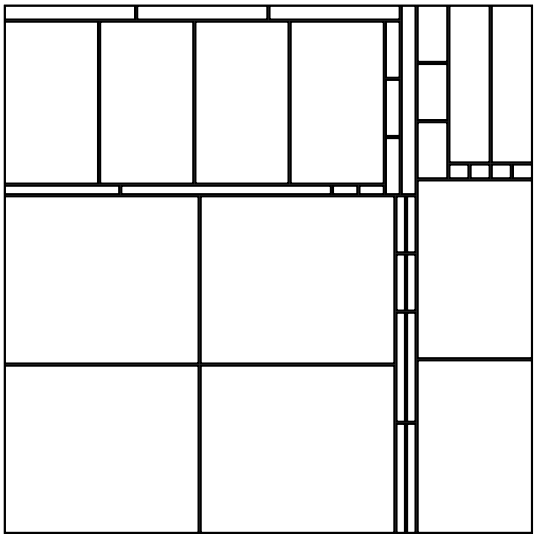


Figura 5.13: *Solución óptima de la instancia tfs3208.*

Capítulo 6

Conclusiones y nuevas líneas de investigación

Cuando iniciamos el trabajo de investigación descrito en esta memoria, nuestros objetivos eran básicamente tres:

- Desarrollar un algoritmo exacto para el problema del pallet para resolver los problemas de *Tipo II* en un tiempo razonable, dado que en la literatura no se habían resuelto de manera óptima todas las instancias de *Tipo II*, utilizando para ello un nuevo procedimiento como es un *Branch and Cut*.

- Crear un algoritmo heurístico, también para el problema del pallet, que pueda encontrar soluciones rápidas para instancias que no habían sido resueltas por otros métodos heurísticos, utilizando técnicas distintas a las empleadas más habitualmente como son los algoritmos metaheurísticos.

- Utilizar los estudios hechos para el pallet para otros problemas más complejos, como el problema con cajas de diferente tamaño.

En el **capítulo 2** hemos hecho una revisión exhaustiva de las publicaciones relacionadas con el problema del pallet, y hemos realizado la generación de todas las instancias para el problema, clasificadas en diferentes tipos, que se utilizarán a lo largo del trabajo.

En el **capítulo 3** proponemos un algoritmo exacto basado en *Branch and Cut* para el problema del pallet. Mejoramos la formulación clásica del problema, disminuyendo el número de restricciones y de variables. Creamos un procedimiento novedoso para el problema basado en *Branch and Cut*. Los algoritmos de separación están basados en la detección de desigualdades válidas para el conjunto máximo independiente del grafo de la solución, pero la definición del grafo incluye nuevos tipos de relaciones, característicos del *PLP*, basados en ideas de pérdidas, dominancia y simetría. Asimismo, adaptamos los procedimientos clásicos para la identificación de ciclos impares y el correspondiente lifting a las características especiales del problema. Hemos obtenido muy buenos resultados para instancias con hasta 100 cajas.

En el **capítulo 4** hemos aplicado con éxito un algoritmo metaheurístico basado en *Tabu Search* para el problema del pallet, para el que sólo algoritmos de tipo recursivo habían proporcionado buenos resultados. El algoritmo proporciona excelentes resultados para los conjuntos de problemas de *Tipo I* y *Tipo II*, que pueden extenderse a un nuevo conjunto de problemas *Tipo III* de hasta 150 cajas. En nuestra opinión el buen funcionamiento del algoritmo se debe en gran parte a la definición del movimiento, basado en el aumento y disminución de bloques de cajas y en el uso de estructuras *G4*.

En el **capítulo 5** hemos empezado a abordar el problema de empaquetamiento general con cajas de diferente tamaño. Hemos aplicado con éxito un algoritmo *Grasp*. El algoritmo está basado en las ideas ya desarrolladas para el problema del pallet. El *Grasp* consigue muy buenos resultados tanto para los problemas *restringidos*, *doblemente restringidos* y problemas *rompecabezas*.

Los resultados que hemos obtenido nos animan a trabajar en un futuro inmediato en nuevas líneas de investigación que extiendan el trabajo realizado en esta memoria para ampliar su ámbito de aplicación. Algunos aspectos en los que se tiene previsto trabajar son los siguientes:

- Aplicar la metodología e ideas utilizadas para el *Branch and Cut* en el problema del pallet al problema de empaquetamiento con cajas de diferente tamaño, para intentar realizar un algoritmo exacto y mejorar los mecanismos de cota para dicho problema.
- Estudiar la posibilidad de realizar otros algoritmos de tipo metaheurístico para el problema general de corte, basándonos en el movimiento que hemos definido, a la vez que intentar abordar otros problemas de corte siguiendo la misma metodología, como el *strip packing*.

Bibliografía

[1] Amaral, A. y Letchford, A. (1999). Comment on 'An exact algorithm for general, orthogonal, two-dimensional knapsack problems'. Working paper available from the second author at Department of Management Science, Management School, Lancaster University, Lancaster LA1 4YW, England, 1999.

[2] Amaral, A. y Letchford, A. (2003). An improved upper bound for the two-dimensional non-guillotine cutting problem. Working paper available from the second author at Department of Management Science, Management School, Lancaster University, Lancaster LA1 4YW, England, 1999.

[3] Amaral, A. y Wright, M. (2001). Experiments with a strategic oscillation algorithm for the pallet loading problem. *International Journal of Production Research*, **39-11**:2341–2351.

[4] Arenales, M. y Morabito, R. (1995). An AND/OR-graph approach to the solution of two-dimensional non-guillotine cutting problems. *European Journal of Operational Research*, **84**:599–617.

[5] Baker, B. S., Coffman, E. G., y Rivest, R. L. (1980). Orthogonal packing in two dimensions. *SIAM Journal on Computing*, **9**(4):846–855.

[6] Balas, E. y Padberg, M. (1976). Set partitioning: a survey. *SIAM Review*, **18**:710–760.

[7] Balas, E. y Yu, C. S. (1986). Finding maximum clique in an arbitrary graph. *SIAM Journal on Computing*, **15**:1054–1068.

[8] Barnes, F. W. (1979). Packing the maximum number of m x n tiles in a large p x q rectangle. *Discrete Mathematics*, **26**:93–100.

[9] Beasley, J. E. (1985). An exact two dimensional non-guillotine cutting tree search procedure. *Operations Research*, **33**:49–64.

[10] Beasley, J. E. (1990). OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, **41**:1069–1072.

[11] Beasley, J. E. (1996). Obtaining test problems via internet. *Journal of Global Optimization*, **8**:429–433.

[12] Beasley, J. E. (2003). A population heuristic for constrained two-dimensional non-guillotine cutting. *European Journal of Operational Research, In press*.

[13] Beltrán, J. C., Calderón, J. E., Cabrera, R. J., y Moreno, J. M. (2002). Procedimientos constructivos adaptativos (GRASP) para el problema del empaquetado bidimensional. *Revista Iberoamericana de Inteligencia Artificial*, **15**:26–33.

[14] Bertsimas, D. y Vohra, R. (1998). Rounding algorithms for covering problems. *Mathematical Programming*, **80**:63–89.

[15] Bhattacharya, S., Roy, R., y Bhattacharya, S. (1998). An exact depth-first algorithm for the pallet loading problem. *European Journal of Operational Research*, **110**:610–625.

[16] Bischoff, E. E. y Dowsland, W. B. (1982). An Application of the Micro to Product Design and Distribution. *Journal of the Operational Research Society*, **33**:271–280.

[17] Bron, C. y Kerbosch, J. (1973). Finding all cliques of an undirected graph. *Communications of the ACM*, **16-9**:575–577.

[18] Brualdi, R. A. y Foregger, T. H. (1974). Packing boxes with harmonic bricks. *Journal of Combinatory Theory*, **17(B)**:81–114.

[19] Caprara, A. y Monaci, M. (2004). On the two-dimensional knapsack problem. *Operations Research Letters*, **32 (1)**:5–14.

[20] Carpenter, H. y Dowsland, W. B. (1985). Practical Considerations of the Pallet-Loading Problem. *Journal of the Operational Research Society*, **36**:489–497.

[21] Cheng, E. y Cunningham, W. H. (1997). Wheel inequalities for stable set polytopes. *Mathematical Programming*, **77**:389–421.

[22] Christofides, N. y Whitlock, C. (1977). An algorithm for two-dimensional cutting problems. *Operations Research*, **25**:30–44.

[23] Cánovas, L., Landete, M., y Marín, A. (2000). New facets for the set packing polytope. *Operations Research Letters*, **27-4**:153–161.

[24] De Cani, P. (1979). *Packing problems in theory and practice*. Ph.D. thesis, Department of Engineering Production, University of Birmingham.

[25] Delorme, X., Gandibleux, X., y Rodriguez, J. (2003). GRASP for set packing problems. *European Journal of Operational Research*, **153 (3)**:564–580.

[26] Dowsland, K. A. (1984). The three-dimensional pallet chart: an analysis of the factors affecting the set of feasible layouts for a class of two-dimensional packing problems. *Journal of the Operational Research Society*, **35**:895–905.

[27] Dowsland, K. A. (1985). Determining an upper bound for a class of rectangular packing problems. *Computers and Operations Research*, **12**:201–205.

[28] Dowsland, K. A. (1985). *Graph Theory and OR–An exact solution to a pallet loading problem*. Ph.D. thesis, University of Wales.

[29] Dowsland, K. A. (1987). A combined data-base and algoritmic approach to the pallet-loading problem. *Journal of the Operational Research Society*, **38**:341–345.

[30] Dowsland, K. A. (1987). An exact algorithm for the pallet loading problem. *European Journal of Operational Research*, **31**:78–84.

[31] Dowsland, K. A. (1993). Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research*, **68**:389–399.

[32] Dowsland, K. A. (1996). Simple tabu thresholding and the pallet loading problem. En I.H Osman, y J.P Kelly (Eds). *Meta-heuristics: theory & applications, Kluwer Academic Publishers*, páginas 379–406.

[33] Dowsland, K. A. y Dowsland, W. B. (1983). A comparative analysis of heuristics for the two-dimensional packing problem. EURO VI.

[34] Dowsland, K. A. y Dowsland, W. B. (1992). Packing problems. *European Journal of Operational Research*, **56**:2–14.

[35] Dowsland, W. B. (1985). Two and three dimensional packing problems and solution methods. *New Zealand Operations Research*, **13**:1–18.

[36] Dowsland, W. B. (1995). Improving palletization efficiency–the theorical basis and practical applications. *International Journal of Production Research*, **33**:213–222.

[37] Dyckhoff, H. (1990). Classification of real world trim loss problems. *En: G. Fandel et al.(Eds), Essays on Production Theory and Planning, Springer-Verlag, Berlin*, páginas 191–208.

[38] Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research*, **44**:145–159.

[39] Elf, M., Gütwenger, C., Jünger, M., y Rinaldi, G. (2001). Branch-and-Cut Algorithms for Combinatorial Optimization and Their Implementation in ABACUS. En M. Jünger y D. Naddef (Eds). *Computational Combinatorial Optimization: Optimal or Provably Near-Optimal Solutions, Springer, Lectures Notes in Computer Science*, **2241**:157–223.

[40] Exeler, H. (1988). *Das homogene Packproblem in der betriebswirtschaftlichen Logistik*. Master's thesis, Physica, Heidelberg.

[41] Exeler, H. (1991). Upper bounds for the homogenous case of a two-dimensional packing problem. *New Zealand Operations Research*, **35**:45–58.

[42] Fekete, S. P. y Schepers, J. (1997). On more-dimensional packing III: Exact algorithms. *submitted to: Discrete Applied Mathematics*.

[43] Feo, T. y Resende, M. G. C. (1989). A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, **8**:67–71.

[44] Glover, F. y Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers. Boston.

[45] Grötschel, M., Jünger, M., y Reinelt, G. (1984). A cutting plane algorithm for the linear ordering problem. *Operations Research*, **32**:1195–1220.

[46] Hadjiconstantinou, E. y Christofides, N. (1995). An exact algorithm for general, orthogonal, two-dimensional knapsack problems. *European Journal of Operational Research*, **83**:39–56.

[47] Haessler, R. W. y Sweeney, P. E. (1991). Cutting stock problems and solution procedures. *European Journal of Operational Research*, **54**:141–150.

[48] Herbert, E. A. y Dowsland, K. A. (1996). A family of genetic algorithms for the pallet loading problem. *Annals of Operations Researchs*, **63**:415–436.

[49] Herz, J. C. (1972). A recursive computational procedure for two-dimensional stock cutting. *IBM Journal of Research Development*, **16**:462–469.

[50] Hinxman, A. I. (1977). A two dimensional trim-loss problem with sequencing constraints. *Advanced Paper of IJCAI-77 MTI*, páginas 859–864.

[51] Hoffman, K. L. y Padberg, M. W. (1993). Solving airline crew scheduling problems by branch-and-cut. *Management Science*, **39-6**:657–682.

[52] Isermann, H. (1987). Ein planungssystem zur optimierung der palettenbelandung mit kongruenten rechteckigen versandgebinden. *OR Spektrum*, **9**:235–249.

[53] Jakobs, S. (1996). On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, **88**:165–181.

[54] Keber, R. (1985). Stauraumprobleme Bei Stückguttransporten, Wissenschaftliche Berichte des Institutes für Fördertechnik der Universität Karlsruhe. *Heft 17*, **49**:819–828.

[55] Lai, K. K. y Chan, J. W. M. (1997). Developing a simulated annealing algorithm for the cutting stock problem. *Computers and Industrial Engineering*, **32**:115–127.

[56] Lai, K. K. y Chan, J. W. M. (1997). A evolutionary algorithm for the rectangular cutting stock problem. *International Journal of Industrial Engineering*, **4**:130–139.

[57] Letchford, A. y Amaral, A. (2001). Analysis of upper bounds for the pallet loading problem. *European Journal of Operational Research*, **132**:582–593.

[58] Leung, T. W., Chan, C. K., y Troutt, M. D. (2003). Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. *European Journal of Operational Research*, **145**:530–542.

[59] Leung, T. W., Yung, C. H., y Troutt, M. D. (2001). Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem. *Computers and Industrial Engineering*, **40**:201–214.

[60] Lins, L., Lins, S., y Morabito, R. (2003). An L-approach for packing (l,w)-rectangles into rectangular and L-shaped pieces. *Journal of the Operational Research Society*, **54**:777–789.

[61] Loukakis, E. y Tsouros, C. (1982). Determining the number of internal stability of a graph. *International Journal of Computer Mathematics*, **11**:207–220.

[62] Morabito, R. y Morales, A. (1998). A simple and effective recursive procedure for the manufacturer's pallet loading problem. *Journal of the Operational Research Society*, **49**:819–828.

[63] Morabito, R. y Morales, A. (1999). Errata 'A simple and effective recursive procedure for the manufacturer's pallet loading problem'. *Journal of the Operational Research Society*, **50**:876.

[64] Murray, A. T. y Church, R. L. (1997). Facets for node packing. *European Journal of Operational Research*, **101**:598–608.

[65] Naujoks, G. (1991). Ein neuer ansatz zur bestimmung theoretischer obergrenzen für das zweidimensionale homogene Packproblem. *OR Spektrum*, **13**:224–228.

[66] Neliβen, J. (August 1993). New approaches to the pallet loading problem. Technical report, Dept. of Computer Science, University of Aachen.

[67] Neliβen, J. (1995). How to use structural constraints to compute an upper bound for the pallet loading problem. *European Journal of Operational Research*, **84**:662–680.

[68] Nemhauser, G. L. y Sigismondi, G. (1992). A strong cutting plane / branch-and-bound algorithm for node packing. *Journal of the Operations Research Society*, **43**:444–457.

[69] Padberg, M. W. (1973). On the facial structure of set packing polyhedra. *Mathematical Programming*, **5**:199–215.

[70] Padberg, M. W. y Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, **33**:60–100.

[71] Pardalos, P. M. y Xue, J. (1994). The maximal clique problem. *Journal of Global Optimization*, **4**:301–328.

[72] Pisinger, D. (September 1994). A minimal algorithm for the bounded knapsack problem. Technical Report 94/27, Dept. of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark.

[73] Prais, M. y Ribeiro, C. C. (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, **12**:164–176.

[74] Ribeiro, C. C. y Resende, M. G. C. (2003). Greedy Randomized Adaptive Search Procedures. En F. Glover and G. Kochenberger (Eds). *Handbook of Metaheuristics, Kluwer Academic Publishers*, páginas 219–249.

[75] Rossi, F. y Smriglio, S. (2001). A branch-and-cut algorithm for the maximum cardinality stable set problem. *Operations Research Letters*, **28**:63–74.

[76] Scheithauer, G. (1999). LP-based bounds for the container and multi-container loading problem. *International Transactions in Operational Research*, **6**:199–213.

[77] Scheithauer, G. y Terno, J. (1993). Modeling of packing problems. *Optimization*, **28**:63–84.

[78] Scheithauer, G. y Terno, J. (1996). The G4-Heuristic for the pallet loading problem. *Journal of the Operational Research Society*, **46**:511–522.

[79] Smith, A. y De Cani, P. (1980). An algorithm to optimise the layout of boxes in pallets. *Journal of the Operational Research Society*, **31**:573–578.

[80] Steudel, H. J. (1979). Generating pallet loading patterns: A special case of the two-dimensional Cutting Stock problem. *Management Science*, **25**:997–1004.

[81] Strijk, T., Verweij, B., y Aardal, K. (2000). Algorithms for maximum independent set applied to map labelling. Technical Report UU-CS-2000-22.

[82] Sweeney, P. E. y Paternoster, E. R. (1992). Cutting and Packing Problems: A Categorized, Application Orientated Research Bibliography. *Journal of the Operational Research Society*, **43**:691–706.

[83] Tarnowski, A., Terno, J., y Scheithauer, G. (1994). A polynomial time algorithm for the guillotine pallet loading problem. *INFORMS*, **32**:275–287.

[84] Thienel, S. (1995). *ABACUS A Branch And CUt System, Version 2.3*. Optimization Research And Software, OREAS.

[85] Tsai, R. D., Malstrom, E. M., y Meeks, H. D. (1988). A two-dimensional palletizing procedure for warehouse loading operations. *IIE Transactions*, **20**:418–425.

[86] Wang, P. Y. (1983). Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, **31**:573–586.

[87] Wang, P. Y. y Wäscher, G. (2002). Cutting and packing. *European Journal of Operational Research*, **141**:239–469.

[88] Wolsey, L. A. (1975). Facets and strong valid inequalities for integer programs. *Operations Research*, **24**:367–372.

[89] Wolsey, L. A. (1975). Facets for a linear inequality in 0-1 variables. *Mathematical Programming*, **8**:165–178.

[90] Wolsey, L. A. (1998). *Integer Programming*. John Wiley and Sons.

[91] Wu, Y. L., Huang, W., Lau, S. C., Wong, C. K., y Young, G. H. (2002). An effective quasi-human based heuristic for solving rectangle packing problem. *European Journal of Operational Research*, **141**:341–358.

[92] Young-Gun, G. y Maing-Kyu, K. (2001). A fast algorithm for two-dimensional pallet loading problems of large size. *European Journal of Operational Research*, **134**:193–202.