UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE CIENCIAS MATEMÁTICAS

E.A.P. DE INVESTIGACIÓN OPERATIVA

Un estudio algorítmico del problema de corte y empaquetado 2D

TESIS

para optar el Título Profesional de Licenciado en Investigación Operativa

AUTOR

Rosa Sumactika Delgadillo Avila

Lima – Perú

2007

UN ESTUDIO ALGORÍTMICO DEL PROBLEMA DE CORTE Y EMPAQUETADO 2D

Rosa Sumactika Delgadillo Avila

Tesis presentada a consideración del Cuerpo Docente de la Escuela de Investigación Operativa de la Facultad de Ciencias Matemáticas, de la Universidad Nacional Mayor de San Marcos, como parte de los requisitos para optar el Título de Licenciado en Investigación Operativa.

Aprobada por:

_____

Dra. María del Pilar Álvarez Rivas
Presidente

_____

Mg. Luis Alberto Oré Luján
Miembro

_____

Mg. Esther Berger Vidal
Miembro Asesor

Lima – Perú
Marzo, 2007

A todas las personas que hacen Investigación de Operaciones.

A mi esposo e hijos, a mis padres y hermanos, por las horas de estímulo y amor incondicional que me han dado y me dan.

AGRADECIMIENTOS

Agradezco a Dios por haberme bendecido abundantemente en toda mi vida, por haberme capacitado intelectualmente para este trabajo, por haberme mostrado que en él encuentro vida.

A mis padres porque siempre me incentivaron para alcanzar mis metas.

A mi esposo porque en él me siento estimulada, a mis hijos por el amor que me dan.

A la profesora Esther Berger por confiar en mí para la realización de este trabajo y por su apoyo.

A los profesores María Álvarez y Luis Oré por su participación como jurados.

A todos los profesores de la Facultad de Matemáticas que aportaron en mi formación académica.

También a los distintos autores de los libros y artículos que he leído, que contribuyeron en mi conocimiento del tema y que forman parte de este trabajo.

INDICE

INTRODUCCIÓN 1
Relevancia del Problema 4
Objetivo 5

CAPÍTULO I
Conceptos y Notaciones Preliminares
1.1 Estructura de un Problema de Corte 7
1.2 Estructura de un Problema de Empaquetado 10
1.3 Dualidad del Problema de Corte y el Problema de Empaquetado 12
1.4 Definición del Problema de Corte en una Dimensión 13
1.5 Definición del Problema de Corte en dos Dimensiones 15
1.6 Definición del Problema de Empaquetado 17
1.7 Problemas de Corte por Guillotina 20
1.8 Problemas de Corte No-Guillotina 23
1.9 Relación con otros Problemas de Optimización 24
1.9.1 Problema de la Mochila 24
1.9.2 Problema de Programación de Tareas Independientes 26
1.10 Clasificación del Problema de Corte y Empaquetado 27

CAPÍTULO II
Métodos Exactos
2.1 Método Simplex 38
2.2 Método de Generación de Columnas 39
2.3 Método Programación Dinámica 40
2.4 Método de Ramificación y Acotación 43
2.5 Método Enumerativo de Wang 47
2.6 Método Informado Algoritmo AAO* 49

CAPÍTULO III
Métodos Heurísticos
3.1 Heurísticas para el Problema de Empaquetado 1D 53
3.1.1 Heurística NF 53
3.1.2 Heurística FF 54
3.1.3 Heurística BF 55
3.1.4 Heurística NFD 55
3.1.5 Heurística FFD 56
3.1.6 Heurística BFD 56
3.2 Heurísticas para el Problema de Empaquetado 2D 58
3.2.1 Algoritmo FBS_{OG} 58
3.2.2 Algoritmo FFF_{OG} 59
3.2.3 Algoritmo FBS_{RG} y FFF_{RG} 60
3.2.4 Algoritmo FC_{RG} 62
3.2.5 Algoritmo KP_{OG} 62
3.2.6 Algoritmo KP_{RG} 63
3.2.7 Algoritmo AD_{OF} 65
3.2.8 Algoritmo TP_{RF} 68
3.2.9 Algoritmo FFD-CUT-$2D_{RG}$ 70
3.2.10 Algoritmo BFD-CUT-$2D_{RG}$ 74

CAPÍTULO IV
Meta Heurísticas
4.1 GRASP 77

4.2 Búsqueda Tabú 83
4.3 Templado Simulado 89
4.4 Algoritmos Genéticos. 94

CAPÍTULO V
Conclusiones y Perspectivas Futuras
5.1 Conclusiones 100
5.2 Perspectivas 102

BIBLIOGRAFÍA 103

LISTA DE FIGURAS Y TABLAS

Fig. 1.1 Estructura general del problema de corte 1D.
Fig. 1.2 Estructura general del problema de corte 2D.
Fig. 1.3 Estructura general del problema de empaquetado 3D.
Fig. 1.4 Patrones de corte en una dimensión.
Fig. 1.5 Patrones de corte en dos dimensiones.
Fig. 1.6 Empaquetado en fajas.
Fig. 1.7 Modelo de corte 2D por guillotina.
Fig. 1.8 Modelo de corte por guillotina en dos periodos.
Fig. 1.9 Modelo de corte por guillotina en tres periodos.
Fig. 1.10 Modelo de corte por guillotina en *n* periodos.
Fig. 1.11 Patrón de corte Nested.
Fig. 1.12 Patrón de corte de contornos no-regulares.
Fig. 1.13 Problema de secuenciación de tareas.
Fig. 2.1 Proceso que sigue el método de programación dinámica.
Fig. 2.2 Forma como opera el método de Wang.
Fig. 2.3 (a) Grafo And/Or y (b) Grafo aditivo And/Or.
Fig. 2.4 Grafo Aditivo And/Or de Fig.2.2.
Fig. 3.1 Soluciones que consiguen las diferentes heurísticas para la instancia $l_1=3, l_2=4\ l_3=5, l_4=2, l_5=2, l_6=1, l_7=3, l_8=3$
Fig. 3.2 Una instancia de binpacking 2D con m=7, (A) Empaquetado en franjas producido por FBS_{OG} ; (B) Caja finita solución encontrada por FBS_{OG} y FFF_{OG}; (C) Empaquetado en tiras dado por $KP_{OG.}$
Fig. 3.3 Soluciones producidas por el algoritmo KP_{RG}.
Fig. 3.4 En la parte superior instancia con n=12, Abajo solución encontrada por el algoritmo AD_{OF} para el bin parking 2D.
Fig. 3.5 Solución encontrada por el algoritmo TP_{RF}
Fig. 3.6 Generación de una pieza residual fija. (a) lámina sin pieza residual fija. (b) lámina con pieza residual fija.
Fig. 3.7 Rotación de una pieza.
Fig. 3.8 Ejecución del algoritmo FFD-CUT-$2D_{RG,}$ donde ® denota rotación del ítem.
Fig. 4.1 Esquema de un algoritmo genético.
Fig. 4.2 Ilustración de Bottom-left.
Fig. 4.3 Representación del cromosoma para el problema de corte.
Fig. 4.4 Ilustración de operador crossover.
Tabla 1.1 Sistematización sobre las características principales.
Tabla 1.2 Problemas de C & E con sus correspondientes notaciones.
Tabla 4.1 Componentes de Búsqueda Tabú.
Tabla 4.2 Relación entre Simulación Termodinámica vs Optimización Combinatoria.

ABREVIATURAS

1D	Una Dimensión.
2D	Dos Dimensiones
3D	Tres Dimensiones
AAO*	Algorithm Additive And y Or
AND/OR	And y Or
AO*	Algorithm And y Or
AD_{OF}	Alternate Directions Oriented Free
AG	Algoritmo Genetico
BF	Best Fit
BFD	Best Fit Decreasing
BFD-CUT-$2D_{RG}$	Best Fit Decreasing Cutting Two Dimensional Rotation Guillotine
BP1D	Bin Packing One Dimensional
BP2D	Bin Packing Two Dimensional
$BP2D_{OG}$	Bin Packing Two Dimensional Oriented Guillotine
$BP2D_{OF}$	Bin Packing Two Dimensional Oriented Free
C&E 2D	Corte y Empaquetado en Dos Dimensiones
FF	First Fit
FFD	First Fit Decreasing
FBS_{OG}	Finite Best Strip Oriented Guillotine.
FFF_{OG}	Finite First Fit Oriented Guillotine.
FBS_{RG}	Finite Best Strip Rotation Guillotine.
FFF_{RG}	Finite First Fit Rotation Guillotine.
FFD-CUT-$2D_{RG}$	First Fit Decreasing Cutting Two Dimensional Rotation guillotine
GRASP	Greedy Ramdom Adaptative Search Procedure
KP2D	Knapsack Problem Two Dimensional
KP01	Knapsack Problem 0-1
KP_{OG}	Knapsack Problem Oriented Guillotine
KP_{RG}	Knapsack Problem Rotation Guillotine
NP-	No Polinomial
NF	Next Fit
NFD	Next Fit Decreasing
PCID	Problema de Corte en Una Dimensión
PC2D	Problema de Corte en Dos Dimensiones
PPL	Problema de Programación Lineal
PPE	Problema de Programación Entera
SA	Simulated Annealing
TP_{RF}	Touch Perimeter Rotation Free
TS	Tabu Search
VSC	Virtual Storage computer

RESUMEN

UN ESTUDIO ALGORÍTMICO DEL PROBLEMA DE CORTE Y EMPAQUETADO 2D

ROSA SUMACTIKA DELGADILLO AVILA

Marzo-2007

Orientadora: Mg. Esther Berger Vidal
Título Obtenido: Licenciado en Investigación Operativa

El problema de corte y empaquetado en dos dimensiones, es un problema NP- difícil perteneciente a la familia de problemas de la optimización combinatoria. El problema combinatorio estriba en la gran cantidad de patrones de corte que puede construirse a partir de un número determinado de requerimientos y un conjunto de objetos los cuales deben ser cortados para satisfacer estos. Este problema es muy importante debido a la gran cantidad de aplicaciones que tiene en la industria. En este trabajo presentamos un estudio de los diferentes métodos que resuelven el problema, clasificándolos por métodos exactos, heurísticas y meta heurísticas. También presentamos conceptos, modelos del problema y las relaciones con otros problemas combinatorios.

Palabras claves: Optimización Combinatoria
Algoritmos
Heurísticas
Meta Heurísticas

ABSTRACT

AN ALGORITHMS STUDY OF THE TWO DIMENSIONAL CUTTING AND PACKING PROBLEMS

ROSA SUMACTIKA DELGADILLO AVILA

March - 2007

Guide: Mg. Esther Berger Vidal
Title Obtain: Graduate in Operations Research

Two dimensional cutting and packing problems is NP-hard, it belong to the family of problems of the optimization combinatory. This problem is based in the great amount of cut patterns that can be constructed from a determined number of requirements and a set of objects which must be cut to satisfy these. This problem is very important because it presents enormous applicability in the industry.
In this work we presented a study of the different methods that solve the problem, classifying them by exact methods, heuristic and meta heuristic. Also we presented concepts, models and the relations with other combinatory problems.

Key words: Optimization Combinatory
Algorithms
Heuristic
Meta Heuristic

INTRODUCCIÓN

Una de las más claras aplicaciones del problema de corte y empaquetado 2D se puede observar en la industria del cuero y calzado nacional. Este es un sector industrial importante por su producción de bienes de consumo masivo, por la cantidad de pequeñas y medianas empresas que aglomera, por la cantidad de puestos de trabajo que genera; y por su efecto multiplicador respecto a otros sectores como el sector de curtiembres, cría de ganado, y sectores empresariales relacionados a la producción y comercialización.

Uno de los problemas inherentes que se observa como en todas las industrias, es el nivel competitivo a que está sometida, siendo crecientes las importaciones en este sector de cuero y calzado, debido a que se ofrecen productos de menores precios y no necesariamente de mejor calidad. Uno de los factores decisivos y necesarios para alcanzar niveles competitivos de ámbito internacional para este sector es la tecnología, la misma que debe ser aplicada en todo el proceso de producción y comercialización. Uno de los problemas, referente al proceso productivo se refiere al corte de las plantillas de calzado; estos cortes son efectuados considerando varios aspectos:

- El contorno irregular del cuero, es decir no se tiene un contorno geométrico definido.
- La textura del cuero o defectos inherentes, como son, posibles tajos, huecos, partes no deseadas, entre otros.
- El contorno no rectangular del corte a ser efectuado, esto debido al diseño del tipo de calzado a confeccionarse, y por tanto el tipo de corte puede tomar formas con contornos curvos, rectos y/o una combinación de estos.

Las tendencias a automatizar los procesos de producción en todos los ámbitos, exigen que los procesos de cortes de cuero también sigan esta tendencia. En la actualidad la tecnología existente permite que el diseño sea generado por computadora. Sin embargo los cortes son efectuados por un maestro cortador que asigna los diseños de corte a la manta de cuero, tratando de observar el menor desperdicio del cuero; no garantizando que la elección del

corte sea el mejor, además de generar un costo adicional por el tiempo gastado en tratar de economizar el material.

Un estudio nacional de Carazo y Hurtado [8] menciona que en micro, pequeñas, y medianas empresas los cortes son efectuados por el maestro cortador (es decir a mano), este estudio también menciona que el 24% de empresarios señalan problemas de desperdicio de materiales, un 45% tiene falta de uniformidad de su producto, un 30% reconoce fallas y errores, y un 18% no sabe sacar moldes o patrones, o en todo caso puede tener más de una de estas dificultades a la vez.

Ciertamente el problema de desperdicio por la no utilización óptima de la materia prima (cuero) coloca al producto nacional (calzado) en serios problemas para competir con sus similares importados, a pesar de su calidad y diseño, tanto en el mercado nacional como internacional.

Similares problemas se presentan en otras industrias de manufacturas con mayor o menor incidencia, como son las industrias de: confección de ropas, derivados del cartón, papel, vidrio, plásticos, entre otros. Vea por ejemplo: el problema de corte en la industria de la lona para la confección de carpas, toldos para jeep y otros (Farley A. [26]); el problema de corte en la industria del vidrio (Dyson R.G. y Gregory A. [20], Farley [24], Madsen [59]); el problema de corte en la industria de ropas (Farley A. [25]); el problema de la perdida residual en el corte de papel corrugado (Haessler R.W. y Talbot F.B. [42]); el problema de cortes en la industria de la madera (Morabito y Garcia [68], Venkateswarlu P. y Martyn C.W. [77]); el problema de corte en la industria del papel (Westernlund et alt [80]); el problema de corte en la industria de tapete (Liton [55]).

En estas industrias una de las grandes tareas es la de utilizar sus recursos de materiales (materia prima) lo más eficientemente; para estas empresas el material desperdiciado es dinero desperdiciado que contribuye al encarecimiento del producto final y por consiguiente a un producto no competitivo.

Otra aplicación del problema un tanto diferente se encuentra en el área de la ciencia de la computación Campello y Maculan [6]; por ejemplo en el cálculo de la memoria virtual en las computadoras (v*irtual storage computer*-VSC). La capacidad de memoria principal es un recurso limitado y extremadamente valioso en un computador digital, por tanto su utilización eficiente es indispensable. Los programas de computador debido a su tamaño, muchas veces no pueden ser almacenados totalmente en la memoria principal. Otras veces, grandes porciones de memoria son ocupadas por segmentos de programas que raramente son utilizados durante el procesamiento. De ahí que para mejorar la utilización de memoria, los programas deben ser estructurados de forma que diferentes segmentos puedan ser asignados a la misma área de memoria en instantes de tiempo definidos.

Una alternativa es utilizar la técnica de memoria virtual, cuando un computador ejecuta un programa virtual, así denominado porque no está necesariamente almacenado en su totalidad en la memoria principal; diversos segmentos automáticamente son traídos cada vez a la memoria principal, mapeando el espacio de dirección virtual en direcciones físicas. Estos segmentos reciben el nombre de páginas pues tienen un tamaño fijo de palabras; el número de palabras en una página es escogido normalmente como una potencia de 2, por ejemplo 2^m. Si el número de palabras en el programa es de 2^n con $n > m$, entonces será automáticamente particionado en 2^k páginas, donde $k = n - m$. La memoria principal es también dividida en segmentos del mismo tamaño llamados marcos de las páginas. La secuencia de operaciones de un VSC al cambiar las páginas es:

- El programa virtual se particiona en páginas. Algunos son colocados en la memoria principal, dependiendo de la disponibilidad y los restantes dejados en la memoria secundaria
- Si durante la ejecución se hace un acceso a una página que no se encuentra en la memoria principal (esto es posible consultando una tabla que indica si la página está o no en la memoria principal), entonces se genera una interrupción.
- Un algoritmo de paginación se activa. La transferencia de página solicitada se realiza substituyendo, si es necesario, una página de la memoria principal y se actualiza la tabla de paginación.

- Cuando la transferencia se completa, la interrupción se suspende y la ejecución del nuevo programa virtual obtenido del anterior por la introducción de una nueva página, prosigue con una nueva referencia a la página recién introducida y obviamente ahora ya no ocurrirá una interrupción.

El problema consiste en organizar los segmentos de un programa en las páginas. Estas páginas tienen tamaño fijo y los segmentos tienen tamaños menores que el tamaño de una página; un segmento de un programa puede ser conformado por datos, funciones, procedimientos, entre otros. Dicho de otra forma, el problema consiste en determinar el menor número posible de páginas y también la organización de los segmentos en las páginas; esto es utilizar al máximo la capacidad de memoria sin desperdiciar espacio ocioso de memoria disponible.

RELEVANCIA DEL PROBLEMA

La importancia del problema de corte en 2D radica en su aplicabilidad en varios sectores productivos, como tecnología clave para la optimización de desperdicios de los recursos incurridos en el proceso de producción que como consecuencia incrementa el nivel de competitividad de las industrias; además también radica en la dificultad de resolución, debido a su característica altamente combinatoria y en consecuencia compleja, que estimula a los investigadores para su estudio.

Las primeras investigaciones de este problema se inician como una extensión del problema de corte en una dimensión. El primer trabajo que aparece es de Kantorovitch [50] el cual fue traducido del ruso en 1960. Similares problemas fueron tratados por Paull y Walter [72], Metzger [67] y Eilon [21] con relativo éxito para problemas pequeños. Gilmore y Gomory [32, 33] resuelve el problema de corte de una dimensión utilizando la técnica de generación de columnas con gran éxito, esto estimula la investigación para el caso de dos dimensiones.

El problema de corte 2D es un problema de programación matemática combinatoria de la clase de problemas NP-difíciles, (problemas intratables). Un problema de programación

matemática es NP-difícil (también denominado NP-arduo o NP-duro o NP-completo) cuando no existe algoritmo exacto que lo resuelva en tiempo razonable para cualquier número de variables; esto es, el tiempo de resolución de estos problemas crece en forma no polinomial (por ejemplo exponencial, factorial) respecto al tamaño (número) de variables; por esto los métodos exactos para la resolución de estos problemas presentan elevados costos de computación (en tiempo y memoria) al extremo de tornarse inviables para su uso. Los investigadores, por años han estudiado y siguen estudiando esta clase de problemas intentando cada día encontrar nuevos algoritmos (heurísticas) para problemas de gran tamaño (gran número de variables) con el fin de encontrar buenos resultados (próximos al óptimo) en tiempo razonable.

OBJETIVO

El objetivo de este trabajo es hacer un análisis de los algorítmos del problema de corte y empaquetado en dos dimensiones (C&E 2D), presentando conceptos, formulaciones, relación existente con otros problemas de optimización, clasificaciones y metodologías existentes para su resolución; clasificando estas metodologías en las que alcanzan la solución óptima (métodos exactos) y las que consiguen soluciones aproximadas (heurísticas y meta heurísticas).

Se persigue con este estudio mostrar como las nuevas técnicas de resolución meta heurísticas son posibles de incorporarse en la resolución de problemas de optimización de la clase intratable con un buen desempeño; como por ejemplo, en el problema de corte y empaquetado 2D.

En el primer capítulo definimos el problema de corte, el problema de empaquetado, mostramos la estructura del problema, clasificación, dualidad del problema de corte y el problema de empaquetado y la relación con otros problemas de optimización.

En el segundo capítulo introducimos los métodos exactos para la resolución del problema de corte y empaquetado como son los de programación lineal, programación dinámica, ramificación y acotación y otros.

En el tercer capítulo presentamos los métodos heurísticos tales como los métodos NFD, FFD, BFD, FBS_{OG}, FFF_{OG}, FC_{RG}, KP_{OG} y otros.

En el cuarto capítulo mostramos las meta heurísticas tales como búsqueda tabú, templado simulado, algoritmos genéticos y GRASP.

En el quinto capítulo concluimos el estudio presentando alcances futuros de investigaciones que se están realizando, y otras perspectivas de resolución del problema de corte y empaquetado 2D.

CAPÍTULO I

CONCEPTOS Y NOTACIONES PRELIMINARES

En el presente capítulo definimos la estructura del problema de corte, la estructura del problema de empaquetado, la dualidad existente entre ambos problemas, definimos el problema de corte de una dimensión, el problema de corte de dos dimensiones, los problemas de la programación matemática combinatoria relacionados al problema de corte y la clasificación de este problema.

1.1 Estructura de un Problema de Corte

La estructura general de un problema de corte, Dyckoff [19] puede ser declarada como sigue:

a) Existen dos grupos de datos básicos cuyos elementos definen cuerpos geométricos de formas definidas en una o más dimensiones:
   - Un conjunto de órdenes solicitando la producción de ciertas piezas (ítems), en general de diferentes tamaños, formas (figuras) y en determinadas cantidades, y
   - Un stock de grandes objetos (bloques) de material (barras, varillas, láminas, planchas) de tamaños fijos o estándar que tienen que ser cortados según las órdenes de las piezas a fin de satisfacer las demandas de producción.

b) Un proceso de corte que determina patrones (esquemas) como resultado de combinaciones geométricas de las piezas en los objetos, siguiendo:
   - Algunas especificaciones del material y de la tecnología del corte, restringido al número de procesos de cortes factibles, y
   - Ciertos objetivos a alcanzarse.

Se obtienen pérdidas residuales cuando no es posible colocar ninguna pieza a cortarse en el objeto grande el cual aún tiene material sobrante.

Como ejemplo de la estructura de un problema de corte se ilustra un problema de corte de tubos para radiadores descrito en Dyckhoff H. [19], problema de corte en una dimensión, el cual considera:

a) Un stock ilimitado de tubos de 98 de longitud utilizados para producir tubos pequeños y una orden de pedido de tubos pequeños de longitudes entre 5 y 46 que deben ser producidos según la demanda. El stock de tubos grandes y la relación de los pequeños tubos, constituyen el dato básico del problema de corte.

b) Las órdenes de producción de los pequeños tubos son combinaciones de estas, de forma que constituyen patrones (esquemas) de corte que se asignan a los grandes tubos. El proceso de construir un patrón obedece a ciertos objetivos y restricciones, siendo específico para el problema. En este caso el objetivo es minimizar la pérdida residual total.

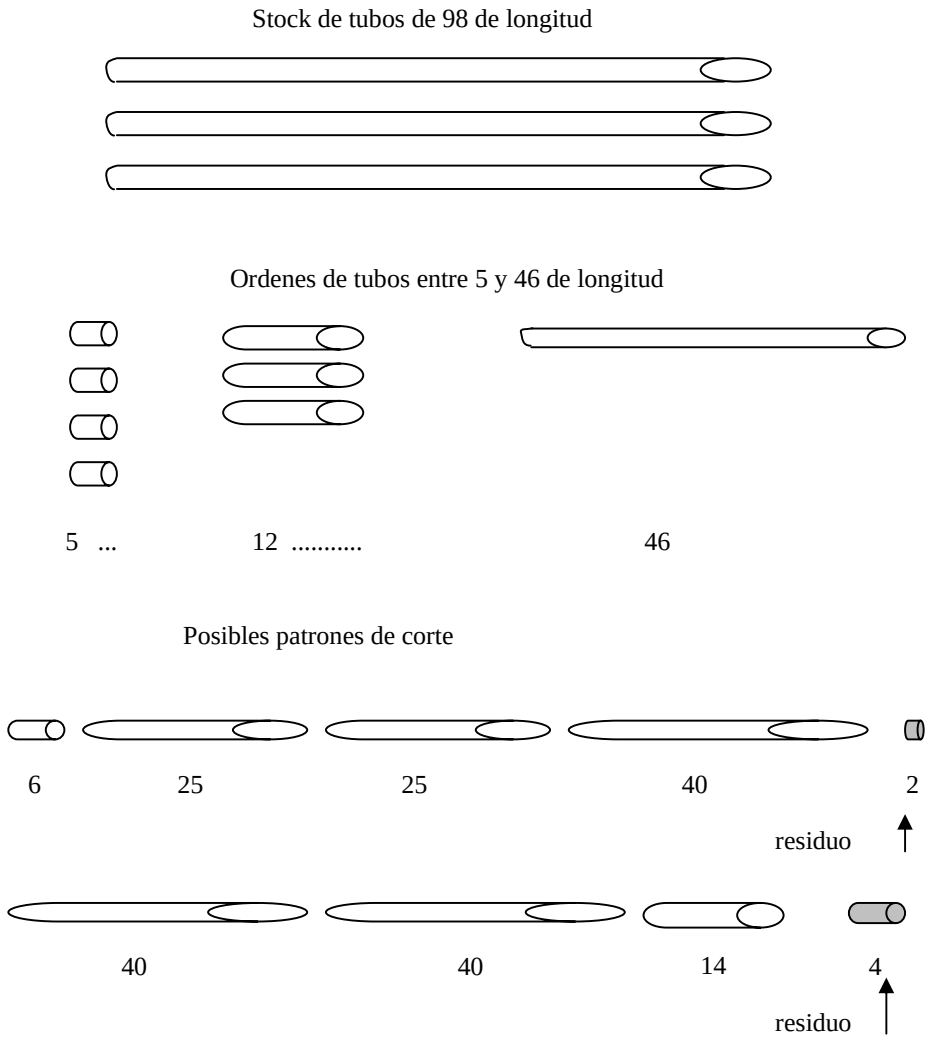


Fig. 1.1 Estructura general del problema de corte 1D

La misma estructura se mantiene para problemas de dos dimensiones, tres y *N*-dimensiones. Un ejemplo del problema de corte de dos dimensiones se consigue al tratar de cortar planchas de metal, vidrio, madera, cartón, etc. En este caso:

a) Existe un stock ilimitado de planchas de tamaño *L x A*, y una orden de pedido de *m* pequeñas piezas de dimensiones $l_1 x a_1, l_2 x a_2, ..., l_m x a_m$ tal que $l_i \leq L$, $a_i \leq A$ para todo *i= 1,2,...,m* que deberá ser satisfecha por el corte adecuado de las planchas.

b) El proceso de construcción de los patrones de corte (órdenes para la producción de las piezas) obedece a objetivos y restricciones, que en este caso consisten en utilizar la menor cantidad de planchas y satisfacer las órdenes de las piezas.

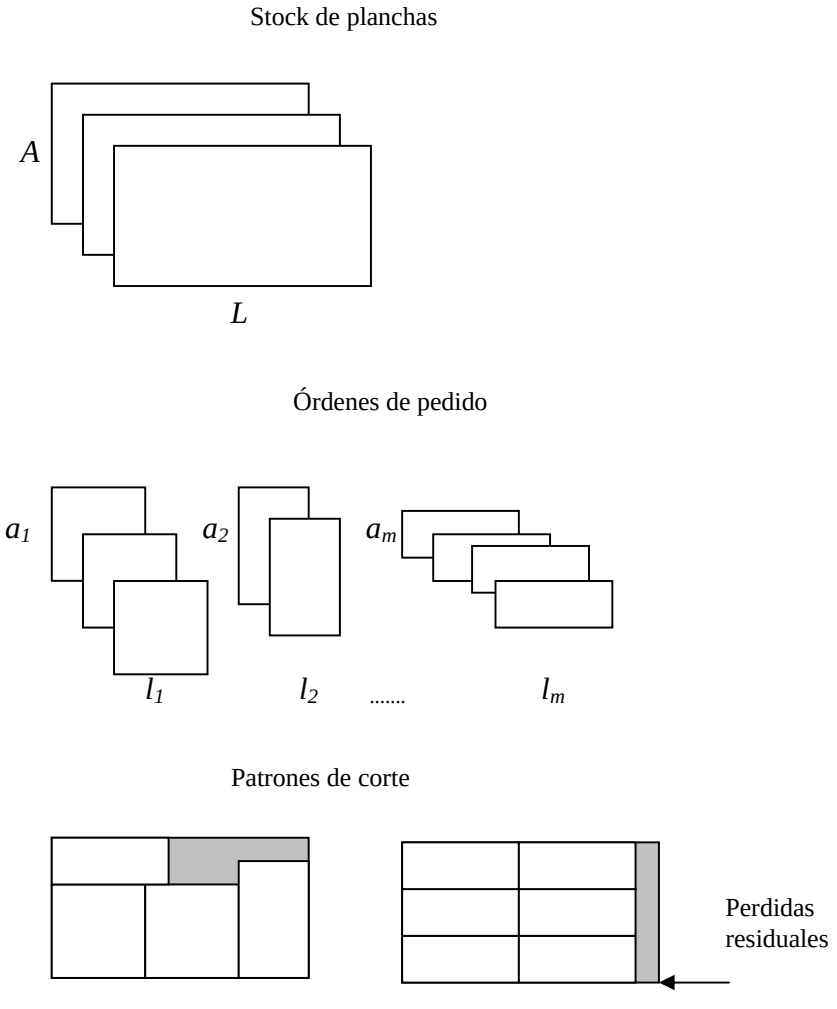


Fig. 1.2 Estructura general del problema de corte 2D.

Los patrones de corte mostrados no son los únicos, las formas sombreadas muestran las pérdidas residuales.

1.2 Estructura de un Problema de Empaquetado

La estructura general de un problema de empaquetado, en forma análoga al problema de corte se puede declarar como sigue:

a) Existe dos grupos de datos básicos cuyos elementos definen cuerpos geométricos de formas fijas en una o mas dimensiones:
   - Un conjunto de órdenes solicitando el empaquetado de ciertas piezas (ítems, cajas), en general de diferentes tamaños y en determinadas cantidades.
   - Un stock de grandes objetos (cajas, container) de tamaños fijos o estándar que tienen que ser utilizados para empaquetar las piezas para satisfacer las demandas de empaque.

b) Un proceso de empaque que determina patrones como resultado de combinar las piezas en los grandes objetos, siguiendo:
   - Algunas especificaciones del material y de la forma del encaje.
   - Ciertos objetivos.

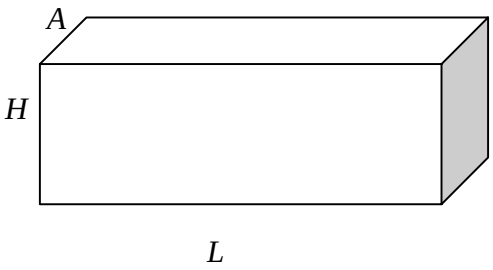
Los espacios residuales se obtienen cuando no es posible colocar ninguna pieza a empacar en el objeto grande en el cual aún se dispone de espacio.

Un ejemplo, se presenta en el problema de carga en un contenedor. En este problema de empaquetado en tres dimensiones se tiene:

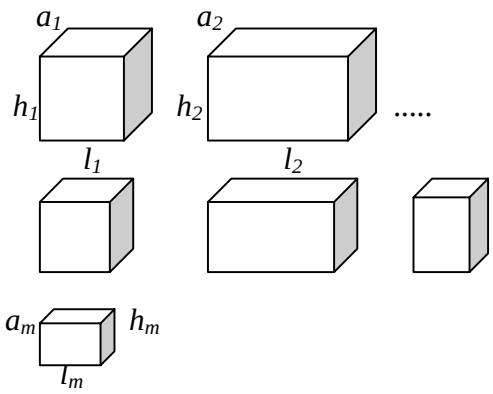
a) Los dos grupos básicos de datos. Por un lado uno o más contenedores (grandes objetos) de dimensiones $L \; x \; A \; x \; H$, y por otro lado una lista de m piezas de dimensiones $l_1 x a_1 x h_1, l_2 x a_2 x h_2, ..., l_m x a_m x h_m$ talque $l_i \leq L$, $a_i \leq A$, $h_i \leq H$ para todo $i = 1,2,...,m$ que deben de ser embalados en los contenedores.

b) Aparte de considerar los objetivos individuales y restricciones, el problema se interesa por la combinación geométrica de las piezas a embalarse que constituyen un patrón o esquema que debe ser asignado al contenedor (o contenedores).

Stock de contenedores



Lista de piezas con largo, ancho y altura dados



Un esquema de empaquetar las cajas en el contenedor

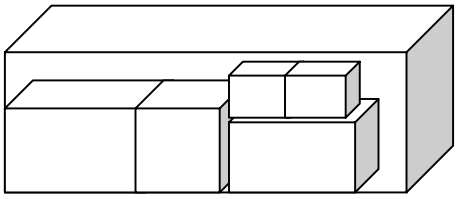


Fig. 1.3 Estructura general del problema de empaquetado 3D

Observe que tanto en el problema de corte como en el problema de empaquetado la estructura general es idéntica. El problema de empaquetado también puede ser colocado en una, dos o mas dimensiones.

1.3 Dualidad del Problema de Corte y el Problema de Empaquetado

En las gráficas mostradas anteriormente se observa una similitud entre el problema de corte y el problema de empaquetado, referente a la estructura de los datos básicos, y en consecuencia la presencia de una fuerte relación entre ellos.

En el problema de corte los objetos son materiales sólidos que deben cortarse en pequeñas piezas o ítems demandados. Ejemplos de ello son papel, metal, vidrio, madera, plásticos, cuero y tejidos o telas. El objetivo principal es la optimización de la pérdida residual (merma), así hablamos del problema de optimización del desperdicio.

En los problemas de empaquetado y carga, los objetos son los espacios vacíos de vehículos de carga, camiones, contenedores, pallets, cajas u otros; que tienen que ser llenados (cargados) con pequeños objetos o piezas. Uno de los objetivos es la minimización de los espacios no utilizados.

El problema de empaquetado puede ser visto como un problema de corte en el cual se trata de cortar los espacios vacíos de los objetos grandes en partes de espacios vacíos pequeños, algunos de los cuales deben ser ocupados con las piezas; los espacios sobrantes son considerados como pérdida residual. Similarmente, el problema de corte puede ser visto como problema de empaquetado en el cual se trata de empaquetar los espacios que ocupan las piezas dentro de los espacios que ocupan los grandes objetos. En otras palabras la fuerte relación que existe entre los problemas de corte y empaquetado resulta de la **dualidad de material y espacio**, esto es la dualidad de un material sólido y el espacio ocupado por este, Golden, [40].

Como consecuencia de esta relación:
1) Se denomina indistintamente problema de corte (*cutting stock problem*) o problema de empaquetado (*packing problem*); o más general problema de corte y empaquetado (*C&E*, *cutting and packing problem*) y
2) Los métodos de resolución, son aplicables a ambos problemas.

1.4 Definición del Problema de Corte de una Dimensión

Se conceptualiza a partir del siguiente enunciado, se tiene un stock de barras de tamaño *L* (objetos grandes) que deben ser cortadas en piezas de tamaño l_i (pequeños ítems) para atender la demanda $N_i, i = 1,...,m$ de las piezas l_i. Las demandas son atendidas decidiendo sobre los distintos patrones (modelos de cortes) de la barra de tamaño *L*. El objetivo es minimizar el número de los grandes objetos utilizados.

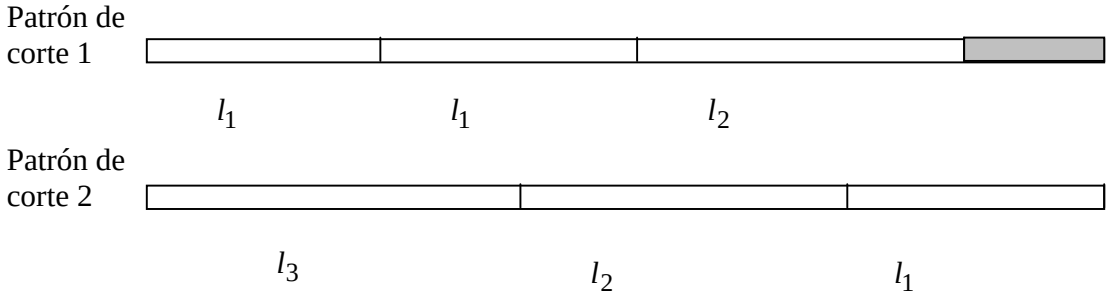


Fig. 1.4 Patrones de corte en una dimensión

El j-ésimo patrón (modelo de corte) es una manera de dividir la barra de tamaño *L* en piezas de tamaño l_i y x_j es el número de veces que se utiliza el j-ésimo patrón de corte.

En la formulación del problema de corte de una dimensión como un programa lineal no entero, Gilmore y Gomory [32], la matriz **A** del problema lineal tiene *m* filas y un gran número de columnas, uno por cada posible patrón de corte. Así, cada vector $(a_1, a_2,..., a_m)$ de enteros no negativos satisfaciendo $l_1 a_1 + l_2 a_2 + ... + l_m a_m \le L$ es una columna de la matriz.

Así, tenemos (PC1D):

$$Min \quad \sum_j x_j$$

$$s.a \quad \sum_j a_{ij} x_j \ge N_i, \quad i = 1,...,m$$

$$x_j \ge 0$$

y en la forma matricial :

$$
\begin{aligned}
&\textit{Min} \quad 1.\mathbf{x} \\
&\textit{s.a} \quad \mathbf{Ax} \geq \mathbf{N} \\
&\qquad \mathbf{x} \geq 0
\end{aligned}
$$

donde:

$\mathbf{x} = (x_1, x_2, ..., x_j, ...)$ es un vector columna de las variables x_j, una para cada columna de la matriz **A**, donde x_j es el número de veces que el patrón *j* se utiliza.

1 = (1,.....,1) es un vector fila de elementos todos iguales a uno.

$\mathbf{N} = (N_1, ..., N_m)$ es un vector columna de las demandas N_i.

A = Matriz de *m* filas, cuyas columnas de estructura $(a_1, a_2, ..., a_m)$ son los posibles patrones de corte

a_i = es el número de veces que la pieza l_i (i = 1,...,m) aparece en el patrón de corte.

Ejemplo, sí el número de piezas diferentes demandadas fueran 5 (m = 5), entonces el vector columna asociado al patrón de corte 1 de la figura 1.4 es (2 1 0 0 0), esto es, el patrón 1 presenta dos piezas de longitud l_1 y una pieza de longitud l_2; y x_1 = 1, 2,… es el número de veces que el patrón 1 se repite.

Si las restricciones son de igualdad, el problema de corte es menos general pues no permite una sobreproducción de las demandas, pero permite una solución de menor costo (más barata) Gilmore y Gomory [32].

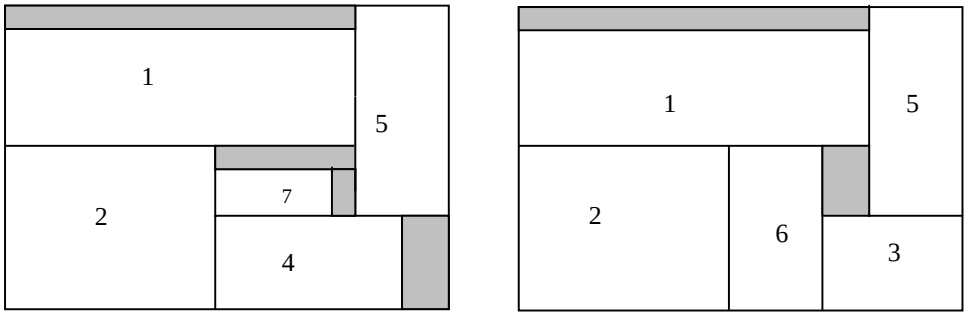
Considerando x_j y N_i enteros, tenemos un problema de programación entera. En la práctica se resuelve el problema de programación lineal (PPL) asociado y redondeando la solución de este, podemos tener una solución satisfactoria para el problema de programación entera (PPE).

1.5 Definición del Problema de Corte de dos Dimensiones

Se enuncia como: Un stock de láminas de dimensiones L x A, deben ser cortadas en piezas de tamaño l_i x a_i proporcionando N_i piezas. Las N_i piezas es la demanda que debe ser atendida por un número cualquiera de láminas.

El corte se ejecuta del siguiente modo: Se selecciona un cierto número de patrones de corte rectangular, donde cada patrón (modelo) es definido como la manera de ajustar rectángulos menores l_i x a_i dentro del rectángulo mayor L x A. El j-ésimo patrón de corte describe cómo una lámina deberá ser cortada para producir láminas menores de acuerdo a la demanda, y donde x_j es la cantidad de veces que el j-ésimo patrón de corte se utiliza.

El objetivo del problema es atender las demandas usando el menor número de láminas.



Patrón de corte 1 Patrón de corte 2

Fig. 1.5 Patrones de corte en dos dimensiones

El problema general de corte en dos dimensiones es formulado en forma semejante al de cortes en 1 dimensión

(PC2D)

$$\begin{aligned} &Min && 1.\mathbf{x} \\ &s.a && \mathbf{Ax} \geq \mathbf{N} \\ & && \mathbf{x} \geq 0 \end{aligned}$$

donde:

$\mathbf{x} = (x_1, x_2, ..., x_j, ...)$ es vector columna de las variables x_j, una para cada columna de la matriz **A**, donde x_j es la cantidad de veces que el j-ésimo patrón de corte es utilizado.

1 = (1,.....,1) es vector fila de elementos todos iguales a uno.

$\mathbf{N} = (N_1, ..., N_m)$ es vector columna de las demandas.

A = Matriz de *m* filas cuyas columnas con formato $(a_1, a_2, ..., a_m)$ son los posibles modelos de corte rectangular del rectángulo *L x A*.

a_i = es el número de rectángulos l_i x a_i (i = 1,..., *m*) que ocurre en el modelo.

En la figura 1.5, en el patrón de corte 1 se muestra que las piezas (rectángulos): 1, 2, 4, 5 y 7 con tamaños ($l_1 x a_1$), ($l_2 x a_2$), ($l_4 x w_4$), ($l_5 x a_5$) y ($l_7 x a_7$) respectivamente han sido encajadas en la lámina *L x A*. Sí *m* = 8, para un problema con este posible patrón de corte, una de las columnas de **A** será (1,1,0,1,1,0,1,0), y para el patrón de corte 2 como las piezas encajadas son: 1, 2, 3, 5 y 6, la columna de **A** asociada a este modelo será (1,1,1,0,1,1,0,0).

El problema de corte algunas veces es subdividido por el objetivo que persigue en dos sub-problemas:

1) **Problema de selección o agrupamiento** (***The assortment problem***) que tiene por objetivo determinar un subconjunto de láminas (barras) de las disponibles para satisfacer la demanda de piezas, cuya formulación es la que aparece en PC2D.

2) **Problema del desperdicio por corte** (***The trim-loss problem***) que tiene por objetivo determinar un patrón de corte adecuado para cumplir con la demanda a partir del stock de láminas (barras) minimizando las pérdidas o desperdicio. Este puede ser formulado como:

$$\begin{aligned} \min \quad & LxA - \sum_{i=1}^{m} x_i l_i a_i \\ s.a \quad & 0 \le x_i \le b_i, \quad 1 \le i \le m \\ & x_i \text{ entero} \end{aligned}$$

x_i, representa el número de veces que la pieza *i* es usada en el patrón de corte.

Se conoce como **problema de corte restricto (*constrained stock cutting problem*)** cuando el número de piezas de un determinado tipo solicitado es limitado superiormente para todo patrón de corte; en caso de no existir un límite superior del número de piezas producidos, el problema se conoce como **problema de corte no restricto** (***unconstrained stock cutting problem*).**

Algunos aspectos teóricos del problema de *cutting stock* son referenciados por Golden [40] y Hinxman [46].

1.6 Definición del Problema de Empaquetado

Como se observó el problema de empaquetado y el problema de corte son lo mismo, la diferencia estriba en la dualidad de espacio y material utilizado. Para efecto de formulación, Campello y Maculan [6] consideran ahora cada varilla como una caja B_j (*Bin*) de tamaño fijo T_0; cada pieza a ser empaquetada tiene tamaño t_i $i = 1,2,...m$. El objetivo es buscar una asignación (empaquetado) de todas las piezas $T = \{t_1, t_2, ..., t_m\}$ utilizando el menor número posible de cajas, tal que ninguna caja B_j sea llenada sobrepasando su capacidad T_0, así como ninguna pieza sea fraccionada en piezas menores (***Bin Packing*).**

La formulación del problema de empaquetado de una dimensión como un problema de programación entera 0-1 es la siguiente:

$$
\begin{aligned}
Min \quad & x_0 = \sum_{j=1}^{n} y_j \\
s.a. \quad & \sum_{i=1}^{m} t_i x_{ij} \le T_0 y_j, \qquad j = 1,2,...,n \\
& \sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,2,....,m \\
& y_j \in \{0,1\}, \qquad j = 1,2,...,n \\
& x_{ij} \in \{0,1\}, \qquad \forall i, j
\end{aligned}
$$

Donde:

$$
y_j = \begin{cases} 1, & \text{si } B_j \text{ es utilizada} \\ 0, & \text{caso contrário} \end{cases}
$$

$$x_{ij} = \begin{cases} 1, & \text{si el item } i \text{ es empaquetado en } B_j \\ 0, & \text{en caso contrário} \end{cases}$$

La primera restricción nos dice que las piezas empaquetadas no sobrepasan la capacidad de la caja y la segunda restricción nos asegura que cada pieza es empaquetada solamente en una caja.

De manera natural podemos extender esta formulación para un problema de empaquetado bi-dimensional; donde T_0 y L_0 representan la capacidad de la caja en términos de ancho y altura de la caja y, t_i y l_i representan el ancho y altura del ítem *i*.

$$Min \quad x_0 = \sum_{j=1}^{n} y_j$$

s.a.

$$\sum_{i=1}^{m} t_i x_{ij} \le T_0 y_j, \quad j = 1,2,...,n$$

$$\sum_{i=1}^{m} l_i x_{ij} \le L_0 y_j, \quad j = 1,2,...,n$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,2,....,m$$

$$y_j \in \{0,1\}, \qquad j = 1,2,...,n$$

$$x_{ij} \in \{0,1\}, \qquad \forall i, j$$

Una formulación alternativa del problema como programación entera dado en Toth [76] es la siguiente: Sea *M* el conjunto de todos los ítems, y *S* la familia de todos los conjuntos de ítems de inclusión máxima que encajan en una caja, esto es;

$$S = \left\{ s \subseteq M : \sum_{i \in s} a_i \le A, \sum_{i \in s} l_i \le L, \sum_{i \in s \cup \{k\}} a_i > L \text{ ó } \sum_{i \in s \cup \{k\}} l_i > L \text{ para todo } k \in M - s \right\}$$

Una formulación como cubrimiento de conjunto (*set covering*) del problema de empaquetado bidimensional es:

$$\min z = \sum_{s \in S} \sigma_s$$

s.a

$$\sum_{i \in s} \sigma_s \ge 1, \qquad i \in M$$

$$\sigma_s \in \{0,1\}, \qquad s \in S$$

donde, $\sigma_s = \begin{cases} 1 & \text{si } s \text{ es empaquetado en una caja} \\ 0 & \text{en otro caso} \end{cases}$

La desventaja de esta formulación es la posibilidad de un gran (exponencial) número de variables.

Otra versión del problema de empaquetado bidimensional es considerar un rectángulo de ancho fijo y longitud ilimitada, y un conjunto de piezas o ítems, con el objetivo de empaquetar todas las piezas minimizando la longitud del rectángulo **(*Strip Packing*).**

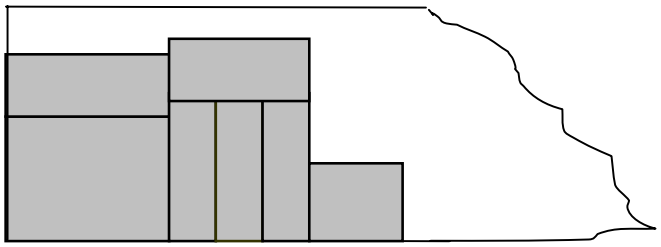


Fig. 1.6 Empaquetado en fajas

Cuando la lista de todos los ítems a empaquetar no se conoce en el momento de la programación, el problema y los algoritmos que lo resuelven son denominados ***on-line*.** En caso que la lista de todos los ítems esta disponible en el momento de la programación el término que se utiliza es ***off-line*.** En los problemas de empaquetado las piezas son empaquetadas ortogonalmente (es decir sus lados son paralelos a los lados de las cajas) y se puede permitir la rotación de las piezas, esto es, un giro de 90°. Una de las principales aplicaciones es en la programación de tareas de un computador donde los anchos representan los requerimientos de memoria y la longitud representa el tiempo; en esta aplicación la lista de tareas a ser programadas puede ser conocida o no.

Formulaciones alternativas son: 1) Fijar el número de cajas minimizando la capacidad de estas, 2) Fijar el número y capacidad de las cajas maximizando el valor total de las piezas empaquetadas.

1.7 El Problema de Corte por Guillotina

Existen problemas particulares del problema de corte en dos dimensiones resultantes de la utilización de ciertas máquinas que cortan el material en línea recta de un extremo de la lámina al otro extremo opuesto, como por ejemplo la guillotina, usada para cortar papel y debido fundamentalmente a que el contorno de los objetos (planchas, láminas) y de las piezas o ítems son rectangulares o cuadrangulares; los patrones de corte en este caso son denominados ortogonales. La complejidad de estos problemas depende del número de cambios de dirección de los cortes (**períodos**) y del número de cortes paralelos que se efectúa por período.

Un ejemplo de corte en periodos es mostrado en la figura 1.7, los cortes son enumerados en el orden en que deben ser efectuados.

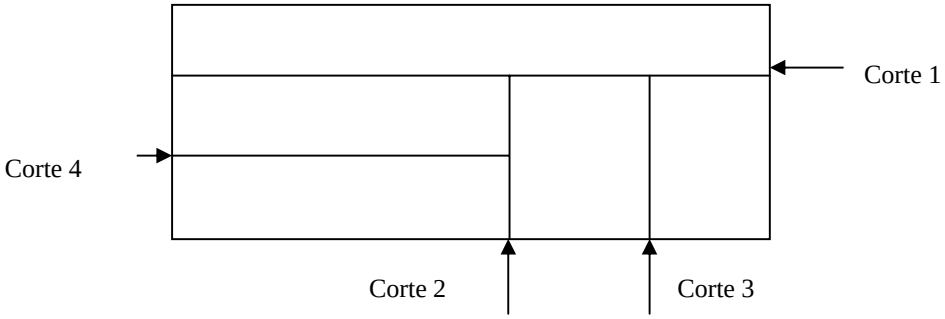


Fig. 1.7 Modelo de corte 2D por guillotina

El corte por guillotina limita mucho el modelo de corte en dos dimensiones, a pesar de esto, se pueden obtener modelos bastante generales.

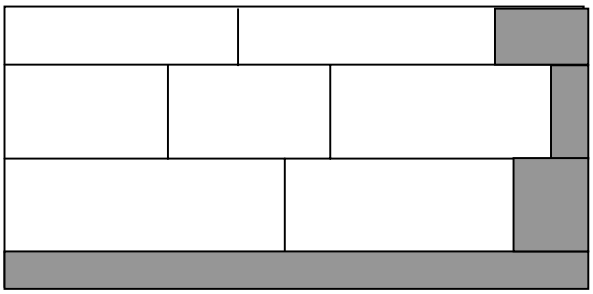


Fig. 1.8 Modelo de corte por guillotina en dos periodos

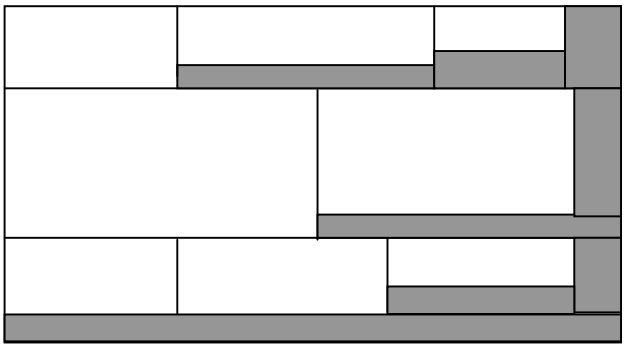


Fig. 1.9 Modelo de corte por guillotina en tres periodos

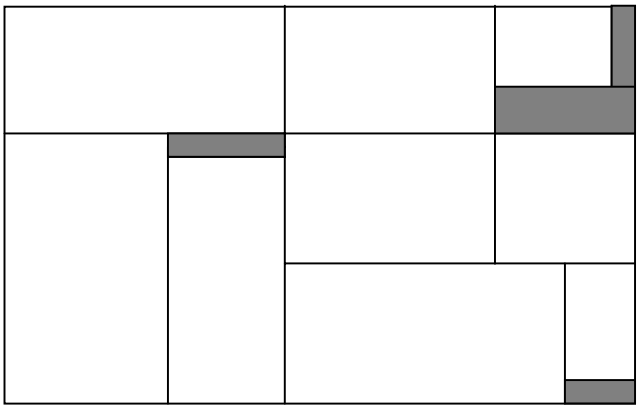


Fig. 1.10- Modelo de corte por guillotina en *n* periodos.

En la figura 1.8, se tiene un modelo de corte en dos periodos, el rectángulo *L* x *A* es cortado según el ancho en tiras y luego cada tira es cortada según el largo, en ocasiones es necesario un tercer periodo como se muestra en la figura 1.9.

De forma inductiva es posible definir cortes por guillotina en *n* periodos como un estilo de corte en el que se considera cada tira como una nueva lámina en la que es posible efectuar un corte por guillotina en *n*-1 periodos.

El problema de corte por guillotina en dos periodos es tratado como un PPL (problema de programación lineal) Gilmore y Gomory [34], donde el primer periodo corresponde al corte del rectángulo *LxA* en tiras de ancho igual al ancho de los rectángulos $l_i \; x \; a_i$ ($i = 1,\ldots,m$) pedidos. El segundo periodo corresponde al corte de las tiras en largos iguales a los largos de los rectángulos pedidos.

El PPL es:

$$\begin{aligned} Min \quad & \sum_j x_j \\ s.a. \quad & A\bar{x} \geq \bar{N} \\ & \bar{x} \geq 0 \end{aligned}$$

donde,

A es una matriz con 2*m* filas partida verticalmente en (*m+2*) matrices $\mathbf{A}_s$ (*s*=0,1,…*m*, *m*+1); de tal modo que cada columna de $\mathbf{A}_0$ corresponde a los modelos que cortan los rectángulos *L x A* en tiras, específicamente, en la *j*-ésima columna que corresponde al *j*-ésimo modelo de corte en tiras; los *m* primeros elementos son (b_1,b_2,…,b_m) enteros no negativos que satisfacen la desigualdad: $\sum_{i=1}^{m} b_i a_i \leq A$ y los últimos *m* elementos son todos ceros. Entonces para cada conjunto de enteros (b_1,b_2,…,b_m) que satisface la desigualdad, el rectángulo *L x A* es cortado en b_i tiras de ancho a_i. Las matrices $\mathbf{A}_s$, (*s* = 1, 2,…, *m*), son el conjunto de modelos que toma las tiras de ancho a_s y las corta en rectángulos. Las columnas de $\mathbf{A}_s$ contienen cero en las *m* primeras filas excepto en la *s*-ésima fila, donde aparece -1; en las *m*+*i* (*i*= 1,…,*s*) filas existen a_i enteros no negativos, que satisfacen la desigualdad: $\sum_{i=1}^{m} a_i l_i \leq L$ y en las últimas (*s*+1, …,2*m*) filas son completadas con ceros.

$\mathbf{A}_{m+1}$ es una matriz de 2*m x m* con las *m* primeras filas nulas y las últimas filas iguales a –I (matriz identidad), formada por los coeficientes de las variables de holgura de las últimas ecuaciones.

$\bar{\mathbf{x}}$ es un vector columna partido horizontalmente de acuerdo con la matriz **A** en $(\bar{x}_0, \bar{x}_1, ..., \bar{x}_m, \bar{u})$ donde $\bar{u}$ es el vector columna de las variables de holgura. Los $\bar{x}_0 = x_1^0, x_2^0, ..., ..$ son el número de veces que es utilizado cada uno de los varios modelos para cortar tiras de *L x A*. Los $\bar{x}_s = x_1^s, x_2^s, ..., \quad s = 1, ...m$ son el número de veces que cada uno de los varios modelos de corte de tiras de ancho a_s es utilizado.

Definiendo $\mathbf{A}_0^*$, como las *m* primeras filas de $\mathbf{A}_0$, $\mathbf{A}_s^*$ como siendo las filas *m*+1,…,*m*+s de $\mathbf{A}_s$, (*s* = 1,…,*m*) se puede escribir las restricciones del PPL para el problema de corte en dos periodos de la siguiente forma:

$$\left[\begin{array}{c|c|c|c|c|c} & -1\ldots-1 & \mathbf{0} & & \mathbf{0} & \\ \mathbf{A}_0^* & & -\mathbf{1}\ldots-\mathbf{1} & \ldots\ldots & & \mathbf{0} \\ & \mathbf{0} & \mathbf{0} & & -\mathbf{1}\ldots-\mathbf{1} & \\ \hline & \mathbf{A}_1^* & & & & \\ \mathbf{0} & & \mathbf{A}_2^* & \ldots\ldots & & -I \\ & \mathbf{0} & \mathbf{0} & & \mathbf{A}_m^* & \end{array}\right]\left[\begin{array}{c} x_0 \\ \bar{x}_1 \\ \ldots \\ \hline \ldots \\ x_m \\ \bar{u} \end{array}\right]=\left[\begin{array}{c} \mathbf{0} \\ \\ \\ \\ \overline{\mathbf{N}} \\ \\ \end{array}\right]$$

Las *m* primeras ecuaciones establecen que las tiras producidas en el primer periodo son utilizadas en el segundo periodo. El segundo conjunto de *m* ecuaciones establece que el número de rectángulos pedidos l_i *x* a_i debe ser atendido cortando las tiras.

1.8 Problema de Corte No-Guillotina

En el caso de que los cortes se efectúen utilizando otras máquinas diferentes a la guillotina que permite efectuar cortes no necesariamente lineales como por ejemplo sierra eléctrica, punzón eléctrico, u otra maquinaria, los patrones de corte se denominan de cortes no-guillotina; ejemplos de estos son los patrones de corte anidado (*Nested*) y los patrones de corte no-ortogonales. En particular, las demandas de ítems con contornos irregulares, no-convexos determinan patrones de cortes no-guillotina. El problema de corte no-guillotina es aún más difícil de resolver que el de corte por guillotina, pero presenta un mejor resultado respecto a la minimización de la pérdida del material. Los trabajos respecto a esta modalidad de corte son escasos en la literatura, Faina [23], Li [54], George [31], a pesar de su comprobada aplicabilidad en industrias como textiles, cuero, algunas industrias de la madera y de los metales.

El modelo matemático asociado a estos problemas se encuadra en rectángulos ajustados a los contornos de las formas y a algoritmos de separación y compactación, ver Li y Milenkovic [54].

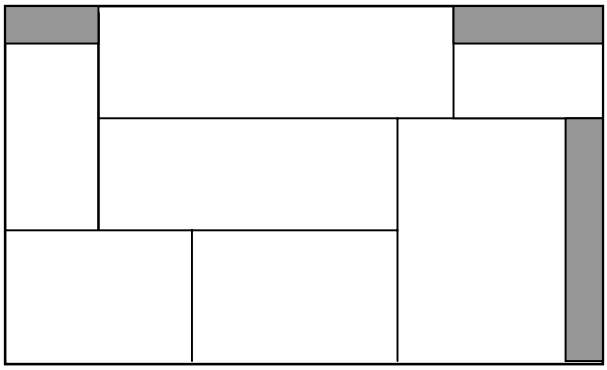


Fig. 1.11- Patrón de corte *Nested*

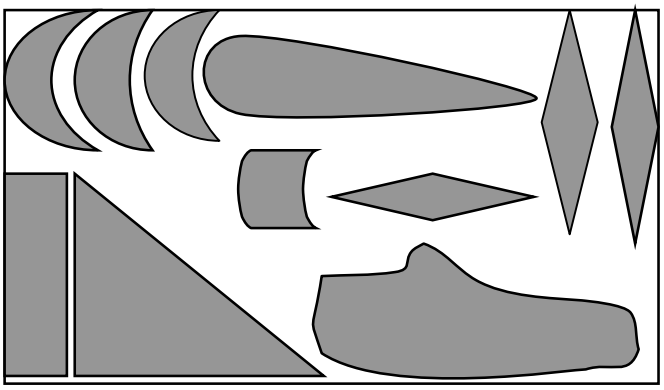


Fig. 1.12-Patrón de corte de contornos no-regulares

1.9 Relación con otros Problemas de Optimización

1.9.1 El problema de la mochila (*Knapsack*)

El problema de la mochila aparece con mucha frecuencia como sub-problema en la resolución de otros problemas como por ejemplo en el método de generación de columnas aplicado a los problemas de optimización combinatoria. La relación con el problema de corte y empaquetado (C&E), se deriva no sólo en la resolución de este utilizando la técnica de generación de columna, para el caso de un modelo lineal en el que cada columna de la matriz A se resuelve utilizando un problema de la mochila, sino también porque el

problema de la mochila puede ser visto como una caja y los productos que se colocan en ella pueden ser los ítems, esto es, un problema de C&E uni dimensional.

Se define, cuando se aplica a proyectos, a partir de que existe *m* proyectos, y para cada proyecto se tiene un costo a_i y un valor o peso c_i, $i = 1,2,...,m$ cada proyecto debe ser realizado o no, es decir no es posible realizar una fracción del proyecto. Existe un presupuesto *b* disponible para los proyectos. El problema es encontrar un subconjunto de proyectos que haga máxima la suma de los valores de los proyectos seleccionados de manera que no exceda el presupuesto (**problema de selección de proyectos)**. El problema es formulado como un problema de programación entera 0-1 de la siguiente manera:

$$\begin{aligned} Max \quad & \sum_{i=1}^{m} c_i x_i \\ s.a. \quad & \sum_{i=1}^{m} a_i x_i \le b \\ & x_i \in \{0,1\}, \quad i = 1,2,...,m \end{aligned}$$

donde:

$$x_i = \begin{cases} 1, & \text{si el proyecto } i \text{ es seleccionado} \\ 0, & \text{en caso contrario} \end{cases}$$

Este problema es más conocido como el problema de la mochila, por su semejanza al **problema del caminante**, en este problema el caminante debe decidir que llevar en su mochila dada una limitación del peso que él puede cargar o la capacidad o volumen de su mochila. La generalización de este problema es conocida como ***loading problem***, desde que el problema puede ser representado como el embarque de materiales de diferentes longitudes (tamaños, pesos, dimensiones) en navíos de una capacidad definida. En general, los problemas de este tipo pueden tener más de una restricción, adicional a la restricción de capacidad, se puede tener restricción de peso, o de volumen, entre otras restricciones, en este caso se trata del problema de la mochila multidimensional o *n*-dimensional.

1.9.2 El Problema de Programación de Tareas Independientes (*Scheduling independent task*)

Es descrito como un conjunto de *m* tareas independientes con duraciones $t_1, t_2, ..., t_m$ y *n* máquinas (procesadores) idénticas que funcionan en paralelo e inicialmente están ociosas. El objetivo es distribuir las *m* tareas en los *n* procesadores minimizando el tiempo de conclusión (*Completion Time / Makespan*) de la última tarea. La formulación del problema es la siguiente:

$$
\begin{aligned}
&Min \ \ x_0 \\
&s.a. \\
&x_0 \geq \sum_{i=1}^{m} t_i x_{ij}, \qquad j = 1,2,...,n \\
&\qquad \sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,2,...,m \\
&x_{ij} \in \{0,1\}, \qquad i = 1,2,...m; \qquad j = 1,2,...,n
\end{aligned}
$$

$$
x_{ij} = \begin{cases} 1 & \text{si la tarea } i \text{ es designa al procesador } j \\ 0 & \text{caso contrário} \end{cases}
$$

donde t_i es el tiempo de operación de la tarea *i*.

Las heurísticas para este problema adoptan en general la estrategia de organizar las tareas en una lista y en el momento en que uno de los procesadores queda ocioso una tarea aún no procesada es retirada de la lista y asignada a este procesador.

Otro problema más general se presenta cuando las tareas son precedidas de otras tareas, en este caso tenemos el **problema de programación de tareas dependientes** que como ejemplo aplicable se presenta cuando se ensambla una línea de producción.

1.10 Clasificación del Problema de Corte y Empaquetado

El problema de corte y empaquetado (C&E) puede ser clasificado por diferentes parámetros y/o características; a continuación mostraremos las clasificaciones más relevantes.

Clasificación por su dimensión

Debido a la importancia que tienen los patrones de corte y a su naturaleza de combinaciones geométricas, se puede decir que los problemas de C&E pertenecen al campo de la Geometría Combinatoria.

Dimensión espacial: En un sentido estricto, estos problemas se relacionan con objetos y piezas definidos por una, dos o tres dimensiones del espacio euclideano.

Dimensión no espacial: En un sentido general, amplio o abstracto los problemas de C&E se sitúan en dimensiones no espaciales. Algunos ejemplos al respecto de dimensiones de otra naturaleza distinta al espacio euclideano son:

- El problema de la mochila (Dantzing [14]) y el problema de carga en vehículos (Eilon y Christofides [22]) donde la dimensión es el peso.
- El problema de balance de una línea de producción (Wee y Magazine, [79]) y la programación de multiprocesadores (Coffman et al., [9]) donde la dimensión es el tiempo.
- El problema de presupuesto del capital (Lorie y Savage, [58]) y cambio monetario (*change making*) (Martello y Toth [60]), donde la dimensión que debe de tomarse en cuenta es la unidad monetaria.
- El problema de asignación de memoria de un computador (Garey y Johnson [30]), en el que la dimensión son los datos almacenados.

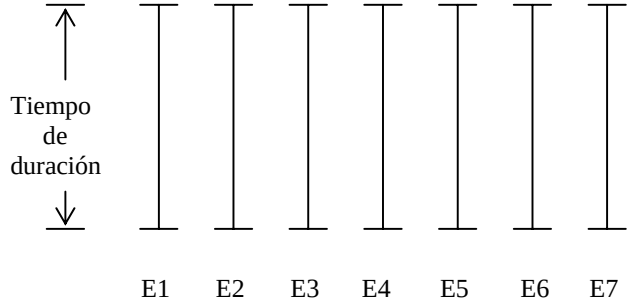
Como ejemplo, en el balance de una línea (Dyckhoff [19]) muestra la estructura lógica del problema de C&E de dimensiones abstractas.

- El stock de objetos grandes se define como las estaciones de trabajo con una capacidad de tiempo fijo, todas con la misma capacidad.

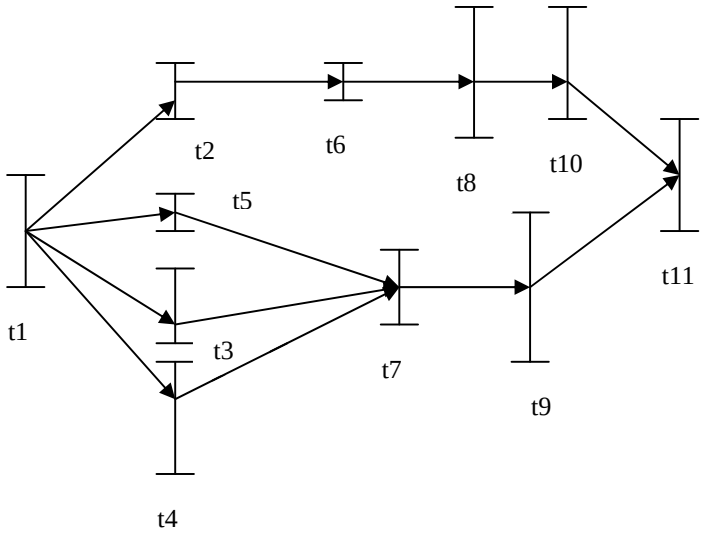
- La lista de piezas (pequeños ítems) se define por las tareas de duraciones específicas el cual tienen que ser ejecutadas.
- El balance de la línea se alcanza a través de patrones de tareas asignados a las estaciones.

Existe además una restricción adicional dada por la secuencia de las tareas (las tareas son precedidas de otras tareas). A excepción de esta restricción su estructura es igual a la del problema de empaquetado, de ahí que el problema de balance de una línea haya sido considerado como un problema general de empaquetado (Wee y Magazine [79]).

Estaciones de trabajo con tiempos iguales (Stock de grandes objetos)



Secuencia de tareas de una determinada duración (Ordenes de tareas)



Proceso de balance con patrones de tareas asignados a las estaciones

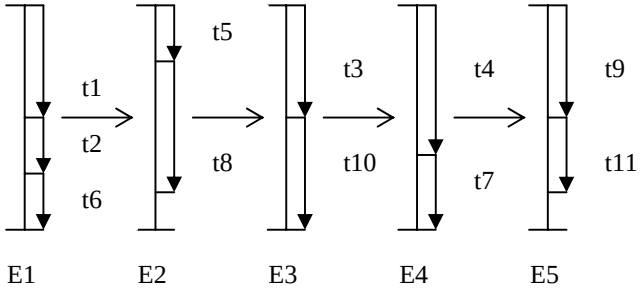


Fig. 1.13 Problema de secuenciación de tareas.

Clasificación por la herramienta de corte

Un factor que influye en la manera de tratar los problemas de corte 2D y por consiguiente en diferenciarlos es la herramienta o maquinaria que se utiliza en el corte de las piezas demandadas.

Problemas de corte guillotina: Se denomina así, cuando la herramienta de corte es una guillotina. Los cortes son ortogonales de lado a lado, (esto es, paralelos a los lados de la lámina).

Problemas de corte no-guillotina: Se denominan así, cuando la herramienta para el corte es cualquier otra herramienta como broca, tijeras, sierras, etc, los cortes no necesariamente son rectos de lado a lado, pueden tomar diversas formas geométricas.

Los métodos existentes para su resolución también son diferenciados por esta clasificación, debido a que la forma diferenciada de abordar cada uno de estos tipos de problemas.

Clasificación por el contorno de las piezas

Otra diferencia sustancial que se observa en los problemas de cortes 2D es el contorno o bordes de los cortes (piezas) demandadas que deben ser atendida.

Problemas de cortes rectangulares: Estas pueden ser cortes regulares o rectangulares propios de la industria de papel, vidrio, metal.

Problemas de corte no rectangulares: Los cortes son de contornos irregulares, esto son, contornos no simétricos, no convexos que son típicos de ciertas industrias textiles, de calzado y/o cuero, y de algunas industrias de la madera.

Clasificación por características y tipos

Una clasificación más sistematizada referente a los componentes de la estructura del problema de corte y sus características geométricas y combinatorias, entre otras; es dada por Dyckoff [19].

Dimensionalidad: La característica más importante es la dimensión, en vez de considerarse separadamente tanto para los grandes objetos como para los ítems esta es atribuida al problema, o más precisamente a los patrones (esquemas de corte). La dimensión es el mínimo número real que describe la geometría del patrón. Los tipos son: una dimensión, dos dimensiones, tres dimensiones y múltiples dimensiones.

Se puede obtener problemas de cuatro dimensiones cuando los problemas de empaquetado de tres dimensiones en el espacio tienen una cuarta dimensión, tiempo. Un ejemplo de cuatro dimensiones es considerar que las cajas que se almacenan en un contenedor pueden permanecer por periodos de tiempo fijos sin interrupción, o cuando consideramos diferentes tiempos para cocinar diferentes tipos de panes que son colocados en hornos industriales.

No siempre es fácil clasificar los problemas de C&E por su dimensión, así por ejemplo, problemas de carga en palletes espacialmente es un problema 3D, sin embargo usualmente se considera como 2D, pues la altura no se considera, pero si la colocación de los ítems es en camadas entonces se debe considerar una tercera dimensión. Similarmente, los contenedores son a menudo cargados construyendo primero pilas verticales y luego localizando las pilas horizontales en la base del contenedor. En ambos casos, se puede hablar de dimensionalidad “2 + 1” en vez de 3. De igual manera, cuando las planchas de vidrio son cortadas solamente por guillotina puede caracterizarse el problema de corte como de “dimensión 1+1” y no de 2 dimensiones.

Tabla 1.1 Sistematización sobre las características principales

Características de	**Características Geométricas**	**Características Combinatorias**	**Otras características**
Objetos grandes	Dimensionalidad Forma o contornos	Cantidad, Agrupamiento, Disponibilidad	Objetivos, Estado de la información, Variabilidad
Piezas	Dimensionalidad, Forma o contornos	Cantidad, Agrupamiento, Disponibilidad	Objetivos, Estado de la información, Variabilidad
Combinaciones Geométricas	Dimensionalidad, Restricción de patrones (figuras admisibles, clase de cortes, distancias, orientación)	Restricciones de patrones (número de cortes, clase, número y combinación de formas)	Objetivos, Estado de la información, Variabilidad
Asignación	----	Restricciones de número de periodos, orden o frecuencia de los patrones.	Objetivos, Estado de la información, Variabilidad

Medidas de Cantidad: Otra característica principal es la medición de la cantidad de los objetos grandes y de los pequeños ítems respectivamente. Dos casos pueden ser distinguidos (Gilmore [35]).

- Medidas discretas o enteras, esto es hecho por números naturales, y
- Medidas continuas o fraccionales, esto es hecho por números reales.

En el primer caso se refiere a la frecuencia o número de objetos o ítems de un cierto contorno (figura), en el otro caso cantidades fraccionales miden la longitud total de los objetos o ítems de contornos iguales con respecto a la dimensión relevante. Ej. en bobinas de papel la longitud o el ancho son dimensiones relevantes.

Contornos o figuras: Otra característica principal de los problemas de corte y empaquetado directamente relacionada a la dimensionalidad es el contorno de los grandes objetos y de los pequeños ítems o piezas.

El contorno de un objeto o un ítem es definido como su representación geométrica en el espacio de dimensiones relevantes. Un contorno es únicamente determinado por su forma, tamaño y orientación. En problemas de más de una dimensión una importante pregunta es si la forma de las figuras es regular o irregular.

Formas regulares pueden ser descritas por pocos parámetros. La mayoría de problemas considerados en la literatura se ocupan de problemas de corte con formas regulares especialmente rectangulares o en bloques, las formas irregulares con contornos no convexos, no simétricos, son menos tratadas porque exhiben mayor dificultad. Dependiendo de su dimensionalidad, el tamaño de una forma puede ser medido por su longitud, área o volumen.

Figuras de igual forma y tamaño difieren a lo más con respecto a su orientación (o posición) esto es, son congruentes. Se pueden distinguir tres casos importantes que consideran la orientación.

- Si, **"cualquier orientación es permitida"** entonces objetos e ítems con figuras congruentes no pueden ser distinguidos.
- Si, **"solamente 90° de rotación es permitida"** entonces sólo objetos e ítems con figuras respectivas son considerados idénticos.
- Si, la **"orientación es fija"** entonces objetos e ítems con figuras congruentes son diferenciados excepto aquellos que pueden ser hechos idénticos por translación.

Se explica así, que figuras de una sola dimensión todas tienen igual forma y orientación, ellas solo difieren en su tamaño o longitud. Ejemplos de idénticas (regulares) formas en dos dimensiones son círculos y cuadrados, mientras que rectángulos con distintos razones de largo y ancho constituyen formas regulares diferentes.

Agrupamiento: No sólo las formas de las figuras principalmente permitidas, como también su particular agrupamiento son importantes para la caracterización de los problemas de C&E. El agrupamiento es dado por las formas y número de las figuras permitidas. Se puede distinguir si, diferentes formas aparecen o todos los objetos e ítems tienen la misma forma. En ambos casos las figuras pueden ser diferentes; pueden ser pocas o muchas figuras diferentes. En el caso de formas idénticas de los objetos e ítems, la diferencia entre figuras resulta del tamaño y orientación de los ítems. Casos especiales surgen cuando las figuras son congruentes o idénticas. Por ejemplo, carga en palletes y en cajas generalmente se refieren a problemas con todos los grandes objetos de igual figura, mientras los pequeños ítems tienen figuras congruentes en el caso de carga en palletes, y muchas figuras diferentes en el caso de carga en cajas.

Disponibilidad: El agrupamiento no dice nada acerca de la cantidad de objetos e ítems a ser considerados. Esto es indicado por la disponibilidad de objetos o ítems y se refiere a

- Límites inferiores y superiores sobre sus cantidades
- Su secuencia u orden
- La fecha cuando un objeto o ítem puede estar o tiene que ser cortado o empaquetado.

Podemos distinguir entre un número finito o infinito de objetos o ítems. En el primer caso pueden existir muchos o pocos objetos o ítems. Los problemas de corte y empaquetado clásicos se relacionan con un número infinito de objetos todos de igual forma, pero con pocos ítems para cada pieza (de muchas diferentes); este es el caso del empaquetado en cajas; mientras que en el problema de corte del stock (inventario) hay muchos ítems para cada pieza (relativamente pocas diferentes). Problemas de cargamento en palletes o de la mochila son usualmente caracterizados por sólo un gran objeto.

Restricciones del patrón: Las características anteriores se refieren a propiedades de los grandes objetos y también de los pequeños ítems. Muchos de ellos sin embargo tienen impacto sobre los patrones como combinaciones geométricas, así como a las asignaciones de los ítems a los objetos. En el primer caso la construcción de los patrones, resulta en restricciones geométricas o combinatorias. En el segundo caso resulta en restricciones que consideran el número, la secuencia o las combinaciones de patrones.

Cuatro importantes grupos sobre restricciones del patrón se pueden distinguir, los cuales deben ser considerados para la construcción de patrones:

- Mínima o máxima distancia entre piezas o entre cortes dividiendo los grandes objetos. Ejemplo, cuando cortamos vidrio o cargamos contenedores.
- La orientación de las piezas relativa a cada uno de las otras y/o al objeto grande debe ser considerada; por ejemplo cuando cortamos tejidos o telas, o cuando cargamos productos frágiles en palletes.
- Restricciones con respecto a la frecuencia de las piezas dentro del patrón, especialmente considerando las combinaciones o el número de piezas diferentes o el número de piezas en total.
- El tipo y el número de cortes permitidos son importantes, particularmente si el objeto y la pieza tienen forma rectangular o de bloque. Siguiendo esta línea se distinguen patrones ortogonales y no ortogonales. Los patrones ortogonales pueden ser patrones por guillotinas o en nido. La complejidad de los patrones por guillotina depende del número de cambios de dirección de los cortes (períodos) así como también del número de cortes paralelos por período. Podrían existir patrones no ortogonales como resultado de los cortes por guillotina.

Restricciones de Asignación: La asignación de las piezas a los objetos grandes puede ser visualizada ficticiamente en dos pasos: El primero, ordenando los ítems dentro de patrones y el segundo asignando los patrones a objetos grandes. Las restricciones pueden considerar:

- La clase de asignación
- Número de periodos (asignaciones)
- Dinámica de la asignación

En la clase de asignación es importante saber tanto para los objetos grandes como para las piezas, si sólo una selección o todos ellos tienen que ser asignados al correspondiente patrón. En el problema de la mochila una selección de los ítems dados tiene que ser combinada en patrones tales que para cada uno de los objetos dados un patrón sea asignado. En el problema de corte y empaquetado, todos los ítems tienen que ser asignados a una selección conveniente de los objetos.

El número de (asignaciones) periodos se refiere a la pregunta si, las piezas son supuestamente cortadas o empaquetadas en los grandes objetos simultáneamente en un paso o sucesivamente en varios pasos. En el primer caso sólo los objetos grandes del problema original son datos admitidos. En el segundo caso, los residuales de algunos patrones vienen a ser objetos grandes de otros patrones. Así, el proceso de asignar en múltiples periodos puede tener a priori un limitado o ilimitado número de periodos.

La asignación de los ítems a los objetos puede ser de naturaleza dinámica o estática. Asignaciones estáticas pueden ser, procesos on line donde en contraste con procesos off line, los objetos o ítems son asignados sucesivamente sin posible reasignación y sin conocimiento de los sucesivos objetos o ítems. En procesos dinámicos, diferentes periodos de tiempo juegan un rol explicito.

Objetivos: Los objetivos de los problemas de C&E a menudo tienen aspectos geométricos como combinatorios. Estos pueden ser atribuidos a los objetos, piezas, patrones o al proceso de asignación. Las distintas clases dependen de:

- Las cantidades de objetos grandes o piezas y residuales asignadas a los patrones.
- La geometría de los patrones (optimización del esquema)
- La secuencia, combinación o número de patrones.

Las funciones objetivo lineales respecto a las cantidades pueden valorar los objetos o las piezas por su tamaño, por ejemplo, el tamaño de la merma (corte perdido) o mínimo número de objetos; otros casos de minimización de costos, valoran objetos y piezas por sus precios. Los objetivos no-lineales aparecen por ejemplo si el “relativo corte perdido” es minimizado o si se deben tomar en cuenta cargas fijas para los patrones. Además, es típico en muchos problemas de C&E que se considere mas de un objetivo.

Estado de la información y variabilidad: Estas características son generales para los problemas de programación matemática. Es necesario saber si los datos del problema son determinísticos, aleatorios o inciertos y si estos son estrictos o pueden ser variables en un cierto intervalo. Por ejemplo, los rollos de plástico para películas tomados desde un almacén pueden tener bordes defectuosos de tamaños fluctuantemente aleatorios. Otro

ejemplo es la producción continua de planchas de metal que son cortadas en tamaños fijos pero que no son producidas exactamente con los tamaños predeterminados. Las demandas de las piezas de las listas de órdenes usualmente se asumen determinísticas, pudiendo ser variables siempre que ciertas desviaciones sean aceptadas por los clientes. La inexactitud de las medidas en la práctica es también una razón para la variabilidad de los datos.

Tipos de combinaciones: Las características consideradas ciertamente no dan una lista completa de todas las posibles propiedades. Además algunas de ellas se superponen, por ejemplo, hay una fuerte relación entre los límites de disponibilidad, la clase de asignación y aquellos objetivos que se refieren a cantidades de objetos y piezas. Dyckhoff [19] ha propuesto una clasificación de los problemas de C&E en base a cuatro importantes características: dimensionalidad, asignación y agrupamiento de objetos y de piezas, dado que ellos tienen un decisivo impacto sobre la selección y la complejidad de los métodos de solución.

1- Dimensionalidad: (1) una-dimensión, (2) dos-dimensiones, (3) tres-dimensiones, (N) N-dimensiones con N > 3.
2- Clase de asignación: (B) Todos los objetos y una selección de ítems, (V) Una selección de objetos y todos los ítems.
3- Agrupamiento de los grandes objetos: (O) Un objeto, (I) Figuras idénticas, (D) Diferentes figuras.
4- Agrupamiento de las piezas: (F) Pocas piezas (de formas diferentes), (M) Muchas piezas de muchas formas diferentes, (R) Muchas piezas de relativamente pocas formas diferentes, (C) Figuras congruentes.

En la tabla 1.2, la falta de un símbolo para una característica significa que todas las respectivas propiedades son posibles. Por ejemplo, el problema de mochila clásico pertenece a un tipo caracterizado por 1-dimensión con un objeto grande que tiene que ser empacado con una selección del conjunto de piezas. El problema de carga en palletes difiere del anterior en que es bi-dimensional. Como se observa por la notación el clásico problema de carga de vehículos, empaquetado en cajas y corte clásico pertenecen al mismo tipo de clasificación considerando las tres primeras características, ellos difieren en el agrupamiento de las piezas (ítems).

Tabla 1.2: Problemas de C& E con sus correspondientes notaciones

Conceptos	Pertenece al tipo
Problema de la mochila (clásico)	1/B/O/
Problema de carga en palletes	2/B/O/C
Problema de la mochila de más dimensiones	/B/O/
Problema de empaquetado en caja dual	1/B/O/M
Problema de carga en vehículos	1/V/I/F ó 1/V/I/M
Problema de carga en contenedores	3/V/I/ ó 3/B/O/
Problema de empaquetado en cajas (clásico)	1/V/I/M
Problema de corte clásico	1/V/I/R
Problema de empaquetado en cajas 2-dimensiones	2/V/D/M
Problema de corte 2-dimensiones	2/V/I/R
Problema general de corte o prob. del desperdicio	1/ / / ó 2/ / / ó 3/ / /
Problema de balance de una línea de producción	1/V/I/M
Problema de programación de multiprocesadores	1/V/I/M
Problema de asignación de memoria	1/V/I/M
Problema de cambio monetario	1/B/O/R
Problema de presupuesto en múltiples periodos	N/B/O/

Por otro lado los problemas de balance de una línea, programación de multiprocesadores y asignación de memoria pertenecen al mismo tipo que el problema de empaquetado en cajas; las diferencias entre ellas se refieren a características no incluidas en esta tipología, de ahí que esta notación determina tipos generales de problemas de C&E, cada una de ellas capturando una variedad de distintos problemas. Por ejemplo nada se dice de las formas de los objetos e ítems de dos o más dimensiones, si ellos son regulares o irregularmente formados; o la tecnología u herramienta del corte asociada que como se describe líneas arriba generan patrones de corte muy particulares. Esta tipología puede ser extendida considerándose otras características adicionales como las mencionadas.

CAPÍTULO II

MÉTODOS EXACTOS

El problema de corte y empaquetado en general es un problema de optimización combinatoria NP-difícil o NP-arduo, lo que indica la complejidad inherente del problema referente al número de variables (número de patrones). Las técnicas exactas aplicadas a este problema tienen relación con el modelo del problema; dado que el problema de C&E puede ser modelado como programa lineal y también como programa entero; se utilizan métodos lineales como Simplex y generación de columnas en el primer caso, y método de ramificación y acotación, programación dinámica y enumeración implícita entre otros, en el segundo caso.

Cuando se utiliza programación dinámica se define una familia de subproblemas similares al problema original y los estados en una fase (o periodo) se calculan en una forma iterativa aplicando métodos recursivos sobre el estado de la fase anterior. La solución óptima corresponde a la mejor solución de la última fase.

En los métodos basados en búsqueda en árboles, como el de ramificación y acotación, el problema original es subdividido en problemas más simples y cada uno de ellos es resuelto independientemente, la construcción del árbol de decisión se genera a partir de un esquema de ramificación; y para evitar la enumeración completa de todas las posible soluciones factibles, se calcula un límite (*bound*) a partir de la relajación del problema actual (nodo actual); si el límite no es mejor que el valor de la mejor solución encontrada hasta el momento, se deja el nodo. La mejor solución se actualiza cuando se detecta otra mejor solución factible. Los métodos de búsqueda en árboles difieren en el tipo de relajación utilizada para calcular el límite en cada nodo.

2.1 Método Simplex

Desde que el problema de corte puede ser formulado como un problema lineal, este es satisfactoriamente resuelto por métodos exactos (*Simple*x) cuando el número de variables es muy pequeño (recordemos que las variables x_j son el número de veces que un determinado patrón j aparece); pues es posible enumerar todos los posibles patrones o esquemas de corte y de esta forma hacer uso del método *simplex* para resolver el problema.

Este método es ampliamente conocido por los que utilizamos las técnicas de Investigación de Operaciones.

2.2 Método de Generación de Columnas

La dificultad de resolver el problema de corte aparece cuando el tamaño de las variables es grande como ocurre en sectores industriales (papel, cuero y otros) al determinar todos los posibles patrones de corte, lo cual puede ser una tarea simplemente imposible. Una forma de abordar esta dificultad fue dada por Gilmore y Gomory [32]; la idea es trabajar con sólo pocos patrones a la vez, una matriz *mxm* (base factible de dimensión igual al número de ítems diferentes) e ir generando nuevos patrones (nuevas columnas para la base) sólo cuando ellos fueran realmente necesarios. El método de generación de columnas para este tipo de problema, consiste en que a cada iteración del método simplex (paso de pivoteamento, variable entrante, en este caso patrón de corte entrante) se resuelve un problema de la mochila asociada al problema.

Veamos, el problema dual del problema de corte (PC2D) es:

$$
\begin{aligned}
&Max \quad \sum_{i}^{m} \pi_i \mathrm{N}_i \\
&s.a \qquad \sum_{i} \mathrm{a}_{ij} \pi_i \le 1, \quad j = 1,.. \\
&\qquad \pi \ge 0
\end{aligned}
$$

De donde el costo reducido de la variable x_j con respecto a la solución del problema dual π_i^* es dado por $1 - \sum_i \pi_i^* \mathrm{a}_i$, entonces para encontrar una variable x_j con costo reducido negativo se debe resolver un problema de la mochila 2D.

(PK2D)

$$
\begin{aligned}
&\max \ \pi_1 \mathrm{a}_1 + \pi_2 \mathrm{a}_2 + ... + \pi_m \mathrm{a}_m \\
&s.a. \quad (\mathrm{a}_1, \mathrm{a}_2, ..., \mathrm{a}_m) \text{ corresponde a un patrón de corte de la lámina } LxA
\end{aligned}
$$

siendo $\pi_i, (i = 1,..., m)$ el multiplicador de langrange o precio sombra correspondiente a la i-ésima ecuación de una solución básica factible de PC2D.

Así, en cada iteración deseamos encontrar un patrón de corte factible que haga máximo el valor dual total del conjunto de piezas pertenecientes al patrón. Sí, el valor máximo (Z_m*) del problema de la mochila asociada es mayor que el costo *C* de la lámina de dimensiones

LxA una columna (patrón) mejor ha sido encontrada, en caso contrario ($Z_m^* < C$ ó $Z_m^* = C$) no es posible mejora alguna y el problema del corte ha sido resuelto. Si hay muchas láminas $L_i x A_i$ con costos C_i, muchos problemas parecidos a (PK2D) deben ser resueltos. Ejemplo de cómo determinar los valores a_i, es dado en Chvátal [13].

Algoritmo GG-GC:

Inicio
Leer *L,A*
Leer l_i, a_i, $i = 1,2\ldots m$
Determine una solución básica factible B_{mxm} seleccionando un conjunto de *m* patrones, en general un patrón *j* se puede construir produciendo solo piezas *j*

Proceso
repetir
Determine el precio sombra resolviendo $\pi B = C$
Determine una columna entrante a la base resolviendo el problema de la mochila asociada KP2D
Sea $a = (a_1, a_2, ..., a_m)$ la columna hallada
Determine $d = (d_1, d_2, ..., d_m)$ resolviendo el sistema $Bd = a$
Determine la columna que sale de la base asociado a $t = \min\{X_B^* / d : d > 0\}$
Si $Z_m^* > C$ se ha conseguido mejorar la solución básica, por el ingreso de la nueva columna,
actualice la Base, haga el valor de la variable (columna) entrante igual a t y $X_B^* = X_B^* - td$
en caso contrario el proceso termina, la solución del problema es la base factible actual
fin repetir
Fin.

La dificultad de este método es la resolución del problema general de la mochila a cada iteración. Resolver el problema de la mochila no es económico en tiempo (es un problema NP-difícil); algunas propuestas han sido dadas como programación dinámica para resolver el problema de la mochila, o el de ordenamiento lexicográfico.
Por tanto la complejidad de este algoritmo es la complejidad del algoritmo que resuelve el problema de la mochila.

2.3 Método de Programación Dinámica

El problema de corte en general es resuelto con éxito cuando el número de variables no es muy grande, por el método de generación de columnas. Un problema de corte 2D también puede ser reducido a un problema de corte 1D desde que una de las dimensiones se fije; el método de generación de columna también resuelve el problema de corte 2D cuando el tipo de corte es por guillotina. Sin embargo el método que se sugiere por la naturaleza del corte guillotina y el número ilimitado de piezas de cada tipo es la

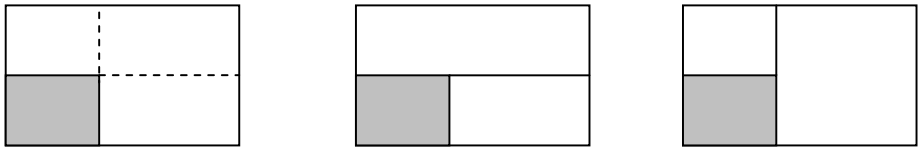
programación dinámica (Dowsland [17], Gilmore y Gomory [34]); la fórmula de recurrencia es la siguiente:

$$G(x,y) = \max_{\substack{0 \le x_0 < x/2 \\ 0 \le y_0 < y/2}} \left\{ g(x,y),\ G(x_0,y) + G(x-x_0,y),\ G(x,y_0) + G(x,y-y_0) \right\}$$

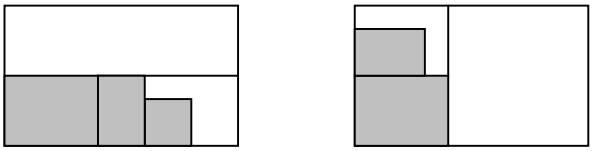
donde *G(x,y)* es el máximo valor que se puede obtener desde un rectángulo (lámina) *x,y* usando piezas l_i, a_i a un precio dual π_i y cualquier sucesión de cortes guillotina, y $g(x,y) = \max_{\substack{l_i \le x \\ a_i \le y}} \pi_i$. Así, la solución óptima se construye considerando soluciones dentro de otros rectángulos menores donde las piezas se pueden encajar. El valor máximo al cortar un rectángulo de dimensión *L* por *A* podrá ser obtenido de tres maneras:

1) Si existe una pieza de dimensión (*L*, *A*) igual a la del objeto entonces el valor óptimo es el valor de la pieza. 2) En otro caso el rectángulo se puede cortar en forma horizontal. 3) O en forma vertical. La posición óptima se determina respecto a la máxima utilización de cada tipo de corte.

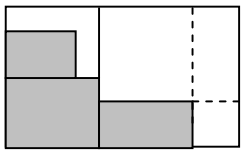
El proceso se aplica a rectángulos de mayores dimensiones hasta que la solución para el rectángulo (lámina) requerido se alcanza.



Forma como se puede hacer el corte dado una pieza, horizontalmente o verticalmente.



Representación de Máxima utilización de cada tipo de corte horizontal o vertical.



El proceso se repite.

Fig. 2.1- Proceso que sigue el método de programación dinámica.

El algoritmo que se muestra es tomado de Hifi y Zissimopoulos [44].

Algoritmo GG-PD:

Inicio
Leer $L,A,$
Leer l_i,a_i, $i= 1,2…m$
Sea $G^*(x,y) = Go(x,y)$
Sea $L^S(x,y)=x$; $A^S(x,y)=y$ para $0 \le x \le L,\ 0 \le y \le A$
Sea $x_1 = x\text{-}x_o = 1,\ y_1= y\text{-}y_0 = 1$

Proceso
Paso 1
Sea $x_o = 1,$
repetir
$V = G^*(x_0, y_1) + G^*(x_1, y_1)$
Si $V > G^*(x_0 + x_1, y_1)$ **entonces**
Sea $G^*(x_0 + x_1, y_1) = V$; $L^S(x_0 + x_1, y_1) = x_0$; $A^S(x_0 + x_1, y_1) = y_1$;
Caso contrario
Si $V = G^*(x_0 + x_1, y_1)$ **entonces** $L^S(x_0 + x_1, y_1) = x_0$;
$x_0 = x_0 + 1;$
Hasta que $(x_0 > x_1)$ **ó** $(x_0 + x_1 > L)$;
Paso 2
Sea $y_o = 1,$
repetir
$V = G^*(x_1, y_0) + G^*(x_1, y_1)$
Si $V > G^*(x_1, y_0 + y_1)$ **entonces**
Sea $G^*(x_1, y_0 + y_1) = V$; $L^S(x_1, y_0 + y_1) = x_1$; $A^S(x_1, y_0 + y_1) = y_0$;
Caso contrario
Si $V = G^*(x_1, y_0 + y_1)$ **entonces** $A^S(x_1, y_0 + y_1) = y_0$;
$y_0 = y_0 + 1;$
Hasta que $(y_0 > y_1)$ **ó** $(y_0 + y_1 > W)$;

Paso 3
Si $x_1 < L$ **entonces** $x_1 = x_1 + 1$ **ir al paso 1**
Si $y_1 < A$ **entonces** $y_1 = y_1 + 1$ **hacer** $x_1 = 1$ **ir al paso 1**

Fin
Solución óptima $G^*(L, A)$ **y** L^S, A^S
Fin.

Este algoritmo es uno de los mejores algoritmos exactos para resolver problemas de corte por guillotina no restringido, y ha sido mejorado en una variedad de maneras, por ejemplo:

Herz [43] sugiere el uso de análisis (cortes) canónicos el cual restringe los sub-problemas a rectángulos cuyas dimensiones son una combinación entera de las dimensiones largo, ancho de las piezas. Esto es, discretiza las variables largo, ancho de las piezas y las láminas, pero es solo aplicable a problemas no ponderados (*unweigth*). Beasley [3] ha mostrado cómo el proceso de discretización dado por Herz [3] y Christofides y Whitlock [11] puede ser usado para mejorar el desempeño del algoritmo de Gilmore y Gomory [35] y usa este para dar una heurística para problemas de gran tamaño.

Una generalización del algoritmo de Herz [43] es dada por M. Hifi y V. Zissimopoulos [44] para problemas de corte ponderados y no ponderados. Un problema de corte se dice que es ponderado cuando considera la prioridad o el valor (precio dual) de las piezas a ser cortadas.
Sin embargo, los algoritmos recursivos son caros computacionalmente (en tiempo y memoria), debido a esto muchos de los métodos se basan en búsqueda en árboles en los que límites inferiores y superiores puedan acortar el tiempo de resolución.

2.4 Método de Ramificación y Acotación

Cuando el problema de corte es formulado como un problema de programación entera lineal, los métodos que se pueden emplear son programación dinámica y ramificación y acotación, ó ramificación y valoración (*branch and price*).
Algoritmos exactos de ramificación y acotación han sido propuestos por Spieksma [75] y Caprara y Toth [7] entre otros.

Son necesarios límites superiores e inferiores (*Upper Bound and lower Bound*) en métodos basados en búsqueda en árboles, tanto para algoritmos enumerativos como en algoritmos de ramificación y acotación puesto que limitan la búsqueda eliminando caminos o ramas por los cuales no se llega a mejores soluciones. Para el problema de corte y empaquetado 2D los límites mas utilizados son:

$$LB1 = \max\left\{\left\lceil \sum_{i\in M} a_i / A \right\rceil, \left\lceil \sum_{i\in M} l_i / L \right\rceil\right\}$$

Este es el primer límite inferior (mayor número de cajas necesarias) para el empaquetamiento en cajas 2D dado por Spieksma [75] que puede ser calculado en tiempo $O(m)$.Caprara [7] demuestra que LB1 es el valor de la solución óptima redondeada superiormente del programa entero relajado del problema de corte.

El segundo límite inferior del problema se basa en el concepto de conjunto estable de un grafo asociado al problema de corte y empaquetado. Un grafo compatible asociado al problema de corte y empaquetado 2D se define como sigue: Dos ítems $i,k \in M$ son compatibles sí $a_i + a_k \leq A$ y $l_i + l_k \leq L$, esto es si i,k pueden ser empacados en la misma caja e incompatibles en caso contrario. Un grafo compatible es un grafo no dirigido $G = (M, E)$ en donde los nodos representan a los ítems y las aristas $(i,k) \in E$ representan cada par (i,k) de ítems compatibles. Un conjunto estable de *G* es un conjunto de nodos $S \subseteq M$ tal que $(i, j) \notin E$ para todo $i, j \in S$. Un conjunto estable *S* corresponde a un conjunto de ítems que deben ser empaquetados en $|S|$ cajas diferentes. De ahí que, un límite inferior es:

$$LB2 = \max\{|S| : S \text{ es un conjunto estable de } G\}$$

Spieksma [75], muestra que esto puede ser calculado en tiempo $O(m^2)$.

Otro límite inferior alternativo se obtiene como sigue: Claramente, sí el máximo número de ítems que encajan en una caja es 2, entonces una solución optima para el problema C&E 2D puede ser eficientemente determinado hallando el matching de máxima cardinalidad de *G*. Mas precisamente, sea $\overline{M} \subseteq E$ un matching de *G*, esto es, para cada $i \in M$ existe al menos una arista en $\overline{M}$ incidente a *i*. Para cada arista $(i, j) \in \overline{M}$, los nodos *i,j* son llamados emparejados (*matched*) y (i, j) es llamado par emparejado. Una solución factible para el problema de C&E 2D, se obtiene utilizando una caja para empaquetar los ítems correspondientes a cada par emparejado y una caja para cada ítem cuyo nodo correspondiente no ha sido emparejado. De ahí:

$$LB3 = \{m - |\overline{M}| : \overline{M} \text{ es un matching de } G\}$$

es el número de cajas a utilizarse. El cálculo de *matching* de máxima cardinalidad puede ser hecho en tiempo $O(m^2)$ (Caprara y Toth [7]). Otros límites puede encontrarse en Martello y Toth [61], Hifi y Zissimopoulos [44].

Ramificación y acotación es una técnica muy conocida para resolver problemas combinatorios, su esquema se basa en reducir el espacio de estado del problema (conjunto factible) podando áreas las cuales no podrían producir mejores resultados que los ya encontrados. Este método busca en un espacio finito de estados *S* un estado $s^* \in S$ que hace óptima la función objetivo. Generalmente este método procede desarrollando un árbol en el cual cada nodo representa una parte del espacio de estado *S*. El nodo raíz representa el espacio de estado total *S*. Los nodos son ramificados en nuevos nodos, esto significa que una parte del espacio de estado S' es dividido en un número de subconjuntos (estados), tal que la unión de estos es igual a S'. De ahí que la solución óptima de S' es la solución óptima de uno de los subconjuntos y el valor óptimo de S' es el mínimo (o máximo) de el valor óptimo de uno de los subconjuntos. El proceso de descomposición se repite hasta que la solución óptima se alcanza en uno de los estados del árbol.

En el método de ramificación y acotación propuesto por Caprara y Toth [7]; en el nodo raíz del árbol de decisión a ramificar, se calcula un límite inferior (LBA) y un límite superior (UBA). Si ambos límites son iguales (LBA = UBA), entonces la solución correspondiente al límite superior (UBA) es la solución óptima. De otro lado, se ejecuta una enumeración primero en profundidad basada en un esquema de ramificación caja por caja. Esto es, se considera una caja por vez y los ítems son colocados en esta siguiendo una ordenación, hasta que no se pueda colocar un ítem más en la caja. Entonces, se calcula un límite inferior para la instancia reducida (ítems restantes), si la solución actual no se puede mejorar se ejecuta un retroceso, en otro caso, se considera una nueva caja. El algoritmo intenta explorar primero las soluciones donde las cajas han sido llenadas en los pasos anteriores tanto como sea posible con respecto a los tamaños, más exactamente, en cada nodo α de ramificación, se considera los ítems $R \subseteq M$ que aún no han sido empacados. Se calcula el límite inferior $k+\max\{LB1, LB2, LB3\}$, donde *k* es el nivel de el nodo α en el árbol y se aplica el algoritmo goloso (*greedy*). Si el nodo α no es agotado, una ramificación se

ejecuta considerando todos los posibles subconjuntos de ítems de inclusión máxima $S \subseteq R$ que encajan en una caja, así, para cada subconjunto *S*, los ítems de *S* se colocan en una caja y se considera el correspondiente subproblema $R=R\backslash S$ (los ítems que faltan encajar). Se da prioridad a la exploración de aquellos subproblemas en los que se tiene un mayor $\sum_{i \in S} s_i$, donde s_i es el tamaño del ítem i. Con este fin una secuencia de instancias del PK2D (problema de la mochila de dos dimensiones) se debe resolver para alcanzar la optimalidad.

Para ser efectiva esta aproximación un nodo α debe ser agotado después de un número limitado de retrocesos (backtracking). Un test de dominancia adecuado se aplica para tal fin (por ejemplo número de retrocesos).

Algoritmo CT-BB:

Inicio
Leer *L,A,* (dimensiones de las cajas)
Leer l_i,a_i, $i= 1,2…m$
R:= m (*R* conjunto de item a ser empacados)
S:=0 (*S* subconjunto de items de inclusión maxima que encaja en una caja)
LBA=max {LB1, LB2, LB3} (un límite inferior)
UBA= Solución de una buena heurística para el problema por ejemplo tabú search. (límite superior)

Proceso

Condición de parada
Si *LBA = UBA* **entonces** la solución que corresponde a *UBA* es la solución óptima, pare;

Ramifica y acota
Si el nodo α no esta agotado
Entonces ramificar considerando todos los posibles subconjuntos de inclusión máxima $S \subseteq R$ que encajan en una caja,
Para cada subconjunto de *S,*
Los ítems en *S* son empaquetados en una caja siguiendo un orden (esquema goloso)
Guardar los subproblemas correspondientes (i.e. los ítems que no han sido considerados R=R \ S;
(1) Resolver una secuencia de problemas de la mochila 2D para seleccionar primero el nodo (subconjuntos) a explorar según se cumpla que $\sum_{i \in S} s_i$ sean los mayores;
(2) Calcular para el nodo seleccionado *LBA = k+max{LB1,LB2,LB3}* para la instancia(reducida) R; (*k* es el Nivel del nodo en el árbol)
Si la solución actual no puede ser mejorada
Entonces el nodo es agotado y un retroceso (*backtracking*) al nodo anterior se efectúa
Seleccione el siguiente mayor de (1) y regrese a (2)
En caso contrario una nueva caja se considera, regrese a condición de parada;

Fin

La complejidad de este algoritmo depende de la resolución del problema de la mochila 2D, del cálculo de los subconjuntos *S* de inclusión máxima y del cálculo de los límites (*bound*); el problema de la mochila 2D puede ser resuelto en tiempo pseudo polinomial utilizando programación dinámica. De igual forma el problema de encontrar los subconjuntos *S* de inclusión maximal se resuelve en tiempo pseudo polinomial, Toth [76], el cálculo de los límites puede ser resuelto en $O(m^2)$, por tanto el algoritmo CT-BB presenta una complejidad $\max\{O(m^2), O(\text{algoritmo tabú})\}$.

Otros algoritmos de ramificación y acotación han sido dados por Mhand Hifi [45] para el problema corte por guillotina (*strip cutting problem*) el cual basa su estrategia en definir los cortes guillotina según el método de Wang [78]. Un algoritmo *Branch and Price* es propuesto por Caprara y Toth [7], similar al algoritmo anterior y se basa en la relajación lineal del problema de corte.

2.5 Método Enumerativo de Wang

Una alternativa para construir un árbol de búsqueda basado en corte por guillotina se da en Wang [78]. El problema de corte que resuelve Wang es el problema de la menor merma (pérdida, desperdicio) dada una sola lámina y un conjunto de ítems con distintas demandas cada una. El método consiste en construir esquemas de corte juntando pares de piezas rectangulares para producir un conjunto de sub-soluciones factibles. Cuatro criterios de factibilidad son considerados a cada nueva generación:

- Las dimensiones de los sub-rectángulos deben ser menores o iguales que la lámina considerada.
- Las pérdidas (mermas) internas (de los sub-rectángulos) deben ser menores o iguales que un cierto porcentaje de la lámina.
- El número de piezas de un tipo especificado en la lámina debe ser menor o igual al límite superior.
- Restricciones del número de piezas de cada tipo deben considerarse para cada sub-solución (problema de corte restringido)

Almacenando los rectángulos que satisfacen el criterio en cada estado, el algoritmo selecciona el mejor rectángulo producido, esto es, el que tiene menor pérdida residual.

El algoritmo de Wang [78] presenta en forma ingeniosa la manera de construir rectángulos y unirlos asegurando así cortes guillotinables, pero en algunos casos también desperdicio, el algoritmo genera rectángulos considerando piezas originales y nuevos rectángulos generados en cada estado.

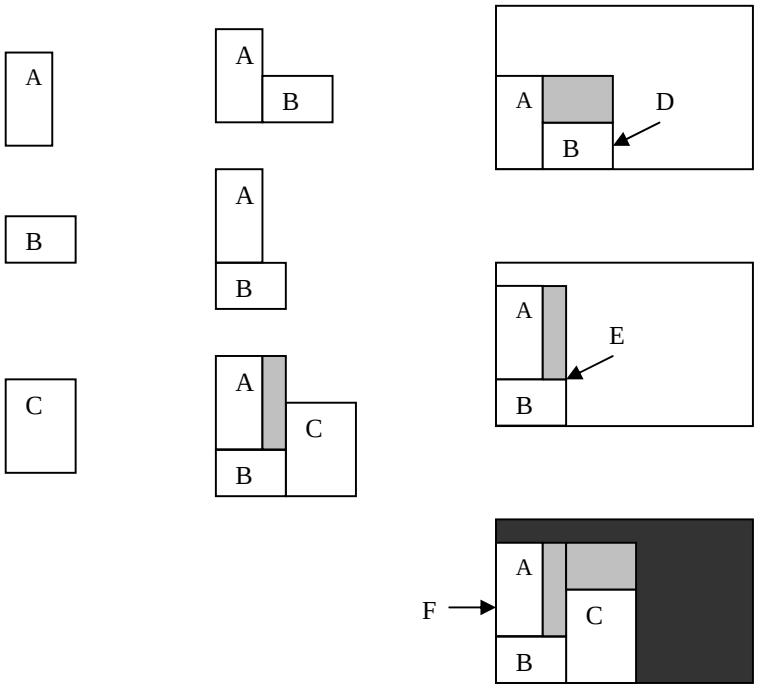


Fig. 2.2 Forma como opera el método de Wang

La figura 2.2 muestra cómo opera el método de Wang, las áreas sombreadas de gris son las pérdidas internas (T_s) generadas en cada sub-solución y el área sombreada de oscuro es la perdida externa (T), la suma de las dos dan la perdida total (T_s+T).

Algoritmo W-E:

Inicio
Leer L, A, β (β es un parámetro)
Leer l_i , a_i , p_i, $d_{,i}$ $i= 1,2…m$ (p_i es la pieza i y d_i su demanda)
$k:=0;$
$F^0 := \{p_1, p_2 ....., p_m\}$; Lista que contiene todas las soluciones parciales generadas en la iteración k
$L^0 := F^0$; Lista que contiene que contiene todas las soluciones parciales generadas hasta la iteración k.

Proceso
Mientras F^k no es vacío **hacer**

k **:=** $k + 1$;
F^k:={ };
Generar todas las soluciones parciales S_k adicionando todos los elementos de F^{k-1} todos los elementos de L^{k-1};
Para cada S_k **hacer**
Si S_k encaja en la lámina y el número de veces que la pieza *i* aparece en $S_k \leq d_i$ y la pérdida interna $\leq \beta LA$
Entonces $F^k := F^k \cup S_k$ **;**
Fin-para
$L^k := L^{k-1} \cup F^K$ **;**
Fin- mientras
$M:=k$;
Seleccionar el elemento de L^M con el menor desperdicio total;

Fin.

El parámetro β es crítico, pues otorga un balance entre un tiempo de ejecución (puede ser muy grande) y el hallazgo de la solución óptima (el mejor patrón seleccionado teniendo en cuenta el menor desperdicio total).

Este método puede ser considerado como un método de enumeración completa (cuando β es próximo de cero) y consecuentemente caro en almacenamiento y tiempo de solución. Muchos investigadores han reportado éxito con este tipo de algoritmo especialmente cuando es utilizado como una heurística antes que como un método exacto.

El número de sub-soluciones puede ser reducido por el uso de un valor de entrada sobre el área de utilización (o desperdicio total), esta mejora ha sido propuesta por Oliveira y Ferreira [69]; ellos sugieren que en cada solución parcial (patrón embrionario) se evalúe el desperdicio total que el mejor patrón a partir de la solución parcial tendría, de esta forma se rechaza o acepta la solución parcial; esto es, si $T_S + T \leq \beta LA$, donde $T = \min(T_m(L - l_S, A) + T_m(l_S, A - a_S), T_m(L, A - a_S) + T_m(L - l_S, a_S))$

Utilizando la metodología de Wang [78] y usando técnicas de ramificación y acotación y programación dinámica, Hifi [44, 45] también propone otros dos algoritmos exactos.

2.6 Método Informado Algoritmo AAO* (*Aditive And/Or*)

Una generalización de los métodos de Wang [78] y Oliveira y Ferreira [69] es dada por V. Parada A. y otros [73]. El método de Wang es visto como un caso particular del método informado (utiliza la información asociada a un estado para la búsqueda de la solución) que utiliza la Inteligencia Artificial; y dada una adecuada función heurística es posible alcanzar

la mejor solución (óptima). Primero se construye grafos *And/Or*, la idea básica de este grafo es que ciertos problemas pueden ser descompuestos en otros problemas más simples (método de reducción).

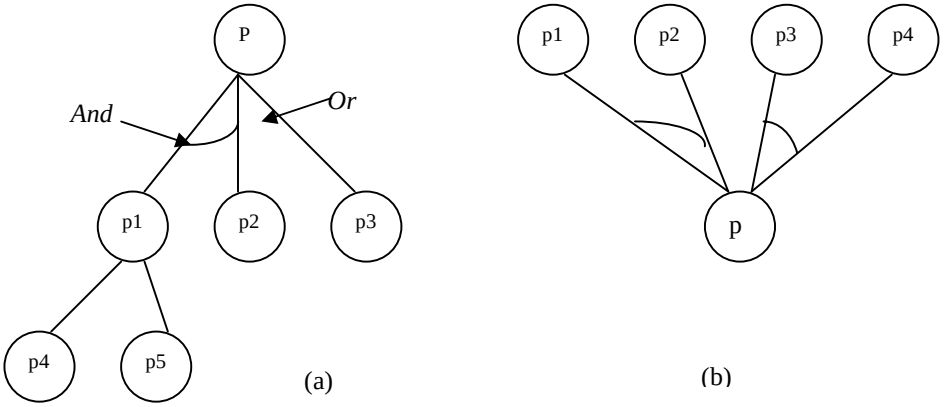


Fig. 2.3 (a) Grafo *And/Or* y (b) Grafo aditivo *And/Or*

En la Figura 2.3 (a), el problema P es descompuesto en p_1, p_2 y p_3. Para resolver P se requiere resolver p_1 y p_2 (representado por un arco *And)* ó p_3 (representado por un arco *Or*). Dado que el grafo tiene todas las posibles divisiones de P y al mismo tiempo estos (subproblemas) son divididos en otros más simples; encontrar una solución para P significa encontrar un subgrafo para el cual todos los subproblemas son resueltos en el último estado (esto es, no es posible seguir dividiendo más). De esta forma una solución de P es generada componiendo las soluciones parciales obtenidas. Desde que las soluciones (subgrafos, que van desde el nodo raíz hasta un nodo final) tienen asociadas a ellas distintos costos es necesario utilizar un método de búsqueda. Un método de búsqueda basado en grafos *And/O*r es el algoritmo AO^*. En el caso del problema de corte utilizando el método de Wang [78] una solución se obtiene combinando soluciones parciales, esto es, la generación de una solución es compuesta por el incremento de las partes de esta; de ahí que un grafo Aditivo *And/Or* se utiliza como en la figura 2.3 (b). Un método de búsqueda sobre este tipo de árbol es llamado de AAO^*.

Para hacer una búsqueda informada (con información) es necesario definir una función de evaluación asociada a cada nodo del grafo. Sea $f(n) = g(n) + h(n)$ la función de evaluación, donde $g(n)$ es una medida del costo de la generación parcial hasta el nodo *n* y $h(n)$ es la función heurística que da la estimación del costo de generación futura para alcanzar el nodo meta. Por tanto $f(n)$ es un costo aproximado de la generación del subgrafo con raíz *n*. El algoritmo AAO* utiliza dos listas de nodos, llamados lista de abiertos (que almacena los nodos que aún no han sido explorados) y lista de cerrados (los nodos que ya han sido explorados). El algoritmo en cada iteración selecciona un nodo de la lista de abiertos con el mínimo valor de *f*, colocando esta en la lista de cerrados y actualizando la búsqueda en el subgrafo (camino que describe).

El algoritmo de Wang puede ser representado por un grafo Aditivo *And/Or* donde cada nodo representa un rectángulo (ítem) y cada arco *And/Or* una unión entre ellos. Los nodos son unidos por pares considerando ambas combinaciones horizontales y verticales, así un par de nodos es generado por otros dos nodos. El grafo que representa todas las combinaciones es finito. Los nodos iniciales son los ítems iniciales y los nodos metas son aquellos que no pueden generar un nuevo rectángulo pues violan la condición de factibilidad (esto es, sobrepasan el tamaño de la lámina). Un grafo Aditivo *And/Or* para los rectángulos de la figura 2.2 se observa en la figura 2.4. Se puede observar que hay un patrón que corresponde a cada solución subgráfica, pero al menos una solución subgráfica correspondiente a cada patrón.

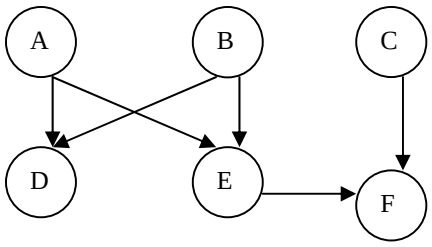


Fig. 2.4- Grafo Aditivo *And/Or* de la figura 2.2.

Una función de evaluación adecuada asociada a cada nodo del grafo para el problema de corte es: $f(n) = g(n) + h(n)$

donde $f(n)$ representa la pérdida total en el rectángulo n (nodo n), como la suma de las pérdidas interna $g(n)$ y externa $h(n)$.

$g(n) = g(n_i) + g(n_j) + c(n_i, n_j)$, donde el rectángulo n es generado uniendo n_i y n_j, las pérdidas internas en los rectángulos n_i y n_j son $g(n_i)$ y $g(n_j)$, y $c(n_i, n_j)$ es la pérdida interna asociada al nuevo rectángulo. La función heurística $h(n)$ se asocia con la pérdida externa del rectángulo cuando este es colocado en la lámina.

Algoritmo AAO*

Inicio
Leer L, A,
Leer l_i, a_i, p_i, $i = 1,2\ldots m$ (p_i es la pieza i)
$Abierto := \{p_1, p_2 ....., p_m\}$; Lista que Abiertos
$Cerrado := \{\ \}$; Lista de Cerrados

Proceso
Calcular $f(n) \quad \forall n \in Abierto$
termina := falso;
Mientras *termina* = falso y $Abierto \neq \phi$ **hacer**
Determine n tal que $f(n) = \min\{f(m); m \in Abierto\}$;
Remueva n de *Abierto* y coloque en *Cerrado;*
Si n es el estado final **entonces** *termina*:= verdadero;
De lo contrario S := *Sucesores (n);*
Calcule $f(j) \quad \forall j \in S$;
Coloque j en *Abierto* $\forall j \in S$; actualizando la información del nodo si es necesario;
Fin Mientras
Si Termina = verdadero **entonces** C^* := *f(n)*;
Caso contrario solución no encontrada
Fin.

Sucesores(n) es el conjunto de sucesores del nodo n (esto es, todas las posibles generaciones de nuevos rectángulos siguiendo el criterio de Wang [78] a partir de n). La complejidad de este algoritmo radica en la determinación de la función de evaluación; esto es, en la determinación de *h(n)* y en el tamaño (número de niveles) que tiene el árbol de decisión. Distintas versiones de cómo determinar *h(n)* son dados en Parada y Gomes [73];

CAPÍTULO III

MÉTODOS HEURÍSTICOS

Las heurísticas son métodos exploratorios para la resolución aproximada del problema. La palabra heurística se deriva del griego *Heuriskein*, que significa *encontrar o descubrir reglas prácticas* utilizadas por los expertos para obtener soluciones aceptables sin tener que realizar búsquedas exhaustivas. La solución obtenida puede o no ser óptima, dependiendo del criterio aplicado y de la estructura particular del problema.

La utilización de heurísticas en la resolución del problema de corte y empaquetado se justifica por la naturaleza combinatoria del problema, por esta complejidad intrínseca se le clasifica como problema de la clase NP-difícil, esto es, el costo computacional (tiempo y memoria) se eleva a medida que se incrementa el número de variables, esto generalmente ocurre en situaciones reales en donde los problemas son de mediano y gran tamaño.

Muchos métodos heurísticos para problemas de corte por guillotina y no guillotina consisten de un ordenamiento y de reglas de colocación de las piezas, generalmente estos algoritmos han sido desarrollado para la solución del problema general de empaquetado en cajas (*Bin packing*), y utilizan heurísticas para el problema de una dimensión extendiendo la idea al problema de dos dimensiones. Primero es resuelto un problema de empaquetado en franjas (*strip packing*) y entonces las franjas (tiras) son tratadas como piezas de una dimensión las cuales son fijadas en las cajas. Han sido propuestas, reglas tales como empaquetar en la primera caja, empaquetar en la caja con mayor capacidad excedente ó en la caja con la mínima capacidad excedente.

En este capítulo describimos primero las heurísticas mas conocidas para *bin packing* 1D (Campello y Maculan [6]) y después describimos las extensiones de estas para el problema de *bin packing* 2D; además de describir algunas otras heurísticas que son una mezcla de las otras.

3.1 Heurística para el problema de empaquetado 1D

3.1.1 Heurística NF (*next fit*- encaje en el siguiente)

Consiste en empacar los ítems en las cajas en cuanto haya espacio para ser colocados, en caso de que no exista espacio se cierra la caja y se utiliza la siguiente caja; los objetos no

están ordenados, su complejidad es $O(m)$. Se utiliza cuando los ítems no son conocidos previamente y el empaquetado se realiza en el momento de llegada de cada ítem (empaquetado *on-line*).

Algoritmo NF
Inicio:
Sea $l_1, l_2, ..., l_m$ los largos de los ítems *1, 2,...,m* ,
Capacidad de *caja p:= L* , (*p= 1,…..)* (largo de la caja)
p:= 1;
Proceso
Para *j* := 1 a *m* **Hacer**
Si Capacidad de *caja p* $\geq$ *lj* **entonces**
Empaquetar *ítem j* en la *caja p*
Caso contrario
Abrir una nueva caja, *p:= p +1*
Empaquetar *ítem j* en la *caja p;*
Fin Si
Capacidad de *caja p*:= Capacidad de *caja p* – l_j ;
Fin Para
Fin
Escribir *p*

3.1.2 Heurística FF (*first fit* – en el primero que ajusta)

También de tipo *on-line*, sin embargo busca mejorar el empaquetado por el aprovechamiento del espacio libre en las cajas ya utilizadas. Cada ítem es empacado usando la primera caja ya utilizada en la que aún cabe, si no es posible se utiliza una nueva caja. Su complejidad es $O(m^2)$.

Algoritmo FF
Inicio:
Sea $l_1, l_2, ..., l_m$ los largos de los ítems *1, 2,...,m* ,
Capacidad de *caja p:= L*, (*p= 1,…..)* (largo ó altura de la caja)
p:= 1;
Proceso
Para *j* := 1 a *m* **Hacer**
Ítem j-ya –empacado:= false**,** *i:=1*,
Mientras (*i* $\leq$ *p*) ó *ítem j-ya-empacado*:= false **Hacer**
Si Capacidad de *caja i* $\geq$ l_j **entonces**
Empaquetar *ítem j* en la *caja i*
Capacidad de *caja i*:= Capacidad de *caja i* – l_j
Ítem j-ya-empacado:= true
Caso contrario
i:= i + 1;
Fin Mientras
Si *ítem j- ya –empacado* = false **entonces**
Abrir una nueva caja, *p:= p +1*
Empaquetar *ítem j* en la *caja p*
Capacidad de *caja p*:= Capacidad de *caja p* – l_j*;*
Fin Para
Fin

Escribir *p*

3.1.3 Heurística BF (*best fit* – el mejor ajuste)

El criterio aquí es minimizar los espacios ociosos de las cajas ya utilizadas, esto es cada ítem es empacado en la caja ya utilizada cuyo espacio vacío se ajusta o es más próxima al tamaño del ítem. Los ítems son conocidos previamente (modalidad *off- line*). También de complejidad $O(m^2)$.

Algoritmo BF
Inicio:
Sea $l_1, l_2, ..., l_m$ los largos de los ítems *1, 2,...,m* ,
Capacidad de *caja p:= L*, (*p= 1,…..)* (largo ó altura de la caja)
p:= 1;
Proceso
Para *j*:= 1 a *m* **Hacer**
Sobra i := - ∞ ;
Sobra i = $Min_{1\le i\le p}$ {sobra k = (capacidad de *caja k – lj*) ≥ 0, $1 \le k \le p$ };
Si sobra i ≥ 0 **entonces**
Empaquetar *ítem j* en la *caja i*
Capacidad de *caja i:*= Capacidad de *caja i* - l_j ;
Caso contrario
Abrir una nueva caja, *p:= p +1*
Empaquetar *ítem j* en la *caja p*
Capacidad de *caja p:*= Capacidad de *caja p* - l_j ;
Fin Para
Fin
Escribir *p*

3.14 Heurística NFD (*next fit decresing*)

Es idéntica a la heurística NF con la diferencia que los ítems son previamente conocidos y ordenados de forma decreciente (empaquetado *off-line*).

Algoritmo NFD
Inicio:
Ordenar los ítems en forma nocreciente, por sus largos l_m.
Sea $l_1, l_2, ..., l_m$ los largos de los ítems *1, 2,...,m* , ya ordenados
Capacidad de *caja p:= L*, (*p= 1,…..)* (largo ó altura de la caja)
p:= 1;
Proceso
Para *j := 1 a m* **Hacer**
Si Capacidad de *caja p* ≥ *lj* **entonces**
Empaquetar *ítem j* en la *caja p*
Caso contrario
Abrir una nueva caja, *p:= p +1*
Empaquetar *ítem j* en la *caja p*;
Fin Si
Capacidad de *caja p:*= Capacidad de *caja p* - l_j ;
Fin Para
Fin
Escribir *p*

3.1.5 Heuristica FFD (*first fit decresing*)

También de tipo *off- line*, idéntica a la heurística FF, con la diferencia de que los ítems son ordenados en forma decreciente.

Algoritmo FFD

Inicio:
Ordenar los *ítems* en forma nocreciente, por sus largos (altura) l_m.
Sea $l_1, l_2, ..., l_m$ los largos de los *ítems 1, 2,...,m* , ya ordenados
Capacidad de *caja p:= L*, (*p= 1,…..)* (largo ó altura de la caja)
p:= 1;
Proceso
Para *j := 1 a m* **Hacer**
Ítem j-ya –empacado:= false**,** *i:=1*,
Mientras ($i \leq p$) ó *ítem j-ya-empacado* := false **Hacer**
Si Capacidad restante de la *caja i* $\geq lj$ **entonces**
Empaquetar *ítem j* en la *caja i*
Capacidad de *caja i*:= Capacidad de *caja i* – l_j
Ítem j-ya-empacado:= true
Caso contrario
i:= i + 1;
Fin Mientras
Si *ítem j- ya –empacado* = false **entonces**
Abrir una nueva caja, *p:= p +1*
Empaquetar *ítem j* en la *caja p;*
Capacidad de *caja p*:= Capacidad de *caja p* – l_j
Fin Para
Fin
Escribir *p*

3.1.6 Heurística BFD (*best fit decresing*)

Difiere de la heurística BF por el hecho de que los ítems son ordenados en forma decreciente.

Algoritmo BFD

Inicio:
Ordenar los *ítems* en forma nocreciente, por sus largos l_m.
Sea $l_1, l_2, ..., l_m$ los largos de los *ítems 1, 2,...,m* , ya ordenados
Capacidad de *caja p:= L*, (*p= 1,…..)* (largo ó altura de la caja)
p:= 1;
Proceso
Para *j := 1 a m* **Hacer**
Sobra i := $-\infty$
Sobra i = $Min_{1\leq i\leq p}$ {sobra k = (capacidad de *caja k* – *lj*) ≥ 0, $1 \leq k \leq p$ };
Si sobra i ≥ 0 **entonces**
Empaquetar *ítem j* en la *caja i*
Capacidad de *caja i:*= Capacidad de *caja i* - l_j ;
Caso contrario
Abrir una nueva caja, *p:= p +1*
Empaquetar *ítem j* en la *caja p*
Capacidad de *caja p:*= Capacidad de *caja p* - l_j ;

Fin Para
Fin.
Escribir *p*

Los algoritmos aproximativos decrecientes, FFD y BFD son los que presentan comportamientos semejantes y muestran mejores resultados, su complejidad de tiempo es $O(m \log m)$ donde m es el número total de ítems a ser empaquetados (m no es el número de ítems diferentes). En la figura 3.1 se observa que para la instancia $l_1=3$, $l_2=4$, $l_3=5$, $l_4=2$ $l_5=2$, $l_6=1$, $l_7=3$, $l_8=3$ los algoritmos FFD y BFD consiguen mejores resultados (empaquetan en 4 cajas) que los otros algoritmos. Aunque los algoritmos FF y BF presentan la misma solución, su comportamiento en tiempo es mucho más caro.

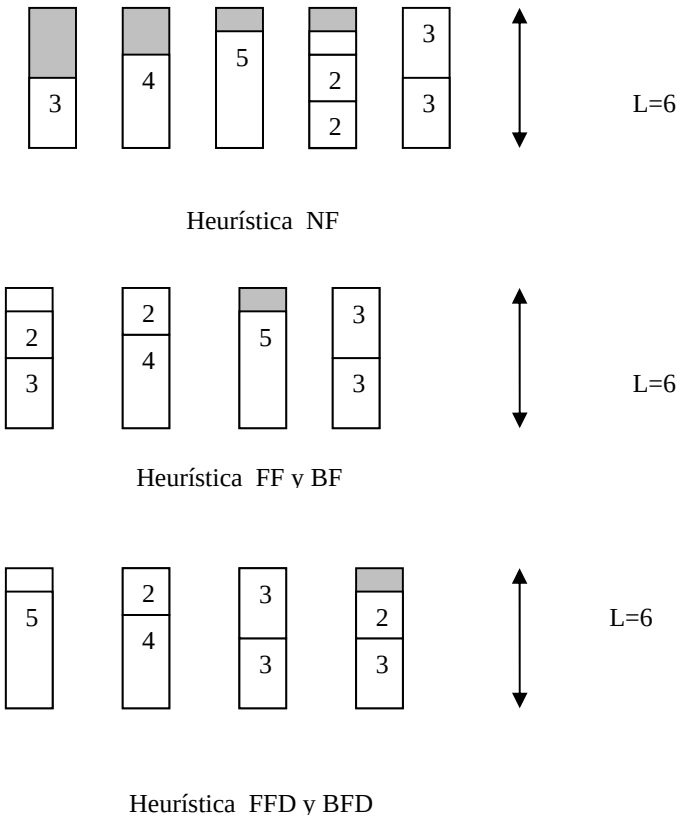


Fig. 3.1 Soluciones que consiguen las diferentes heurísticas para la instancia $l_1=3$, $l_2=4$ $l_3=5$, $l_4=2$, $l_5=2$, $l_6=1$, $l_7=3$, $l_8=3$

3.2 Heurísticas para el problema de empaquetado 2D

Las extensiones de las heurísticas antes mencionadas para el problema *bin packing* 2D, en muchos casos consisten de dos fases (ver Chung, Garey y Jhonson [12]):

La primera fase: Consiste en resolver un problema asociado conocido como *strip packing* 2D (empaquetar en franjas ó tiras); en este caso, dada una caja abierta de ancho *A* y altura (ó largo) infinita, el objetivo es empacar todos los ítems tal que la altura de la caja (llena con los ítems) es minimizada.

La segunda fase: Consiste en que la franja (caja) empacada es dividida en cajas de altura finita *H*.

Un algoritmo para el problema *strip packing* 2D usualmente se obtiene a través de un algoritmo estante (*shelf*), esto es, se empaca los ítems en filas formando estantes (Coffman, Garey y otros [10]). El primer estante de una caja coincide con su fondo, nuevos estantes dentro de la misma caja son creados a lo largo de la línea horizontal que coincide con el tope del ítem más grande empaquetado sobre el último estante. La segunda fase, consiste en resolver un problema *bin packing* 1D en el cual cada estante es visto como un elemento que tiene un valor h_j (j = 1, 2,...) igual a la altura ítem del mas grande que este contiene.

La primera aproximación de este tipo ha sido propuesta por Chung, Garey y Jhonson [12]. Heurísticas efectivas han sido desarrolladas por Berkey y Wang [5] para $BP2D_{OG}$ (*Bin packing* 2D en donde el corte es por guillotina y no se permite rotación de las piezas); estos algoritmos son FBS_{OG} y FFF_{OG}; ambos inician ordenando por altura los ítems en forma no creciente.

3.2.1 Algoritmo FBS_{OG} (*Finite Best Strip* $_{Oriented\ Guillotine}$)

Inicia en la primera fase empaquetando en estantes (franjas ó tiras), con altura igual a la altura del ítem a empaquetar, de acuerdo a la política BFD*;* esto es, sí el actual ítem no encaja en cualquier estante existente, entonces un nuevo estante se inicia para este. De lo contrario el ítem es empacado dentro del estante que minimiza el espacio residual horizontal.

En la segunda fase, los estantes resultantes son empaquetados en cajas finitas utilizando la heurística de BFD para *bin packing* 1D, esto es, el siguiente (más alto) estante es empacado

dentro la caja que minimiza la capacidad residual vertical o dentro de uno nuevo, si ninguna caja se le acomoda.

Algoritmo FBS$_{OG}$

Inicio
Ordenar los ítems de acuerdo al valor *hj* en forma no creciente;
Abrir un estante, con altura h_1
Empacar primer ítem
Proceso
Fase1;
J:= 2;
Repetir
Si *ítem j* no quepa en ningún *estante* ya existente **entonces**
Abrir un nuevo *estante* con tamaño la altura del *ítem, hj*
Empacar *ítem j* en *nuevo estante*
Caso contrario
Empacar el *ítem j* dentro del *estante k*, tal que
sobra k = min { sobra i = (ancho del estante i – (ancho utilizado en *estante i* + *aj*)) $\geq$ 0, para todo estante, donde *aj* es el ancho del ítem};

Hasta que todos los ítems son empaquetados;

Sea $\bar{h}_1,...,\bar{h}_s$ las alturas de los estantes resultantes,
Fase2;
Determine la solución finita para la caja resolviendo la instancia del BP1D

el cual tiene *s* elementos, con los valores asociados $\bar{h}_1,...,\bar{h}_s$, y capacidad *H*
Según algoritmo BFD,
Fin.

El algoritmo puede ser implementado con complejidad de tiempo $O(m\log m)$. Como un ejemplo, considere la instancia mostrada en la figura 3.2. El empaque en franjas producido en la primera fase es representado en la figura 3.2 (A), y su correspondiente solución en caja finita se da en la figura 3.2 (B).

3.2.2 Algoritmo FFF$_{OG}$ (*Finite First Fit* *Oriented Guillotine*)

Comienza empaquetando directamente los ítems dentro una caja finita (saltando la fase de empaque en franjas), de acuerdo a la política de FFD, esto es, el ítem actual es empacado en la primera caja que puede acomodar a este, o en el fondo de una caja nueva, si no hay cajas que acomodan a este. En el primer caso, el ítem es empacado sobre el primer

(más bajo) estante existente en la caja, que pueda acomodar a este o iniciando uno nuevo si tal estante no existe.

Algoritmo FFF_{OG}

Inicio
Ordenar los ítems de acuerdo al valor *hj* en forma no creciente;
Empacar primer ítem en la primera caja
Proceso
Repetir
Encontrar primera caja donde quepa ítem j,
Colocar en el estante más bajo que lo contenga o iniciar un nuevo estante
En caso que ninguna caja ya existente pueda acomodar al ítem j, abrir una nueva caja y empacar ítem j en esta.
Hasta que todos los ítems son empaquetados;
Fin.

Este algoritmo tiene complejidad $O(m^2)$.

En la figura 3.2 el algoritmo FFF_{OG} produce la misma solución que FBS_{OG}. Es necesario remarcar que estos algoritmos son de tipo *off-line* y corte guillotina.

3.2.3 Algoritmos FBS_{RG} y FFF_{RG} (*Finite Best Strip Rotation Guillotine y Finite First Fit Rotation Guillotine*)

Estos algoritmos son adaptaciones de los algoritmos FBS_{OG} y FFF_{OG} respectivamente cuando la rotación del ítem se permite (Lodi, Martello, y Vigo[56]). Ambos algoritmos se denotan como FBS_{RG} y FFF_{RG}. Se puede decir que un ítem tiene una orientación horizontal (respectivamente vertical) si su lado más grande (respectivamente más pequeño) es paralelo al fondo de la caja o de la franja. Ambos algoritmos inician ordenando los ítems en forma no creciente respecto al valor de su lado más pequeño y orientándolos horizontalmente.

Difiriendo en su fase iterativa respecto de los algoritmos FBS_{OG} y FFF_{OG} en dos aspectos: 1) Cuando el actual ítem inicializa un nuevo estante, este es siempre empacado horizontalmente de forma que minimice la altura del estante.

2) Cuando un estante existente se considera como posible empaquetamiento del ítem actual si ambas orientaciones son factibles, entonces la orientación vertical siempre se considera de forma que favorece futuros empaquetamientos sobre el mismo estante.

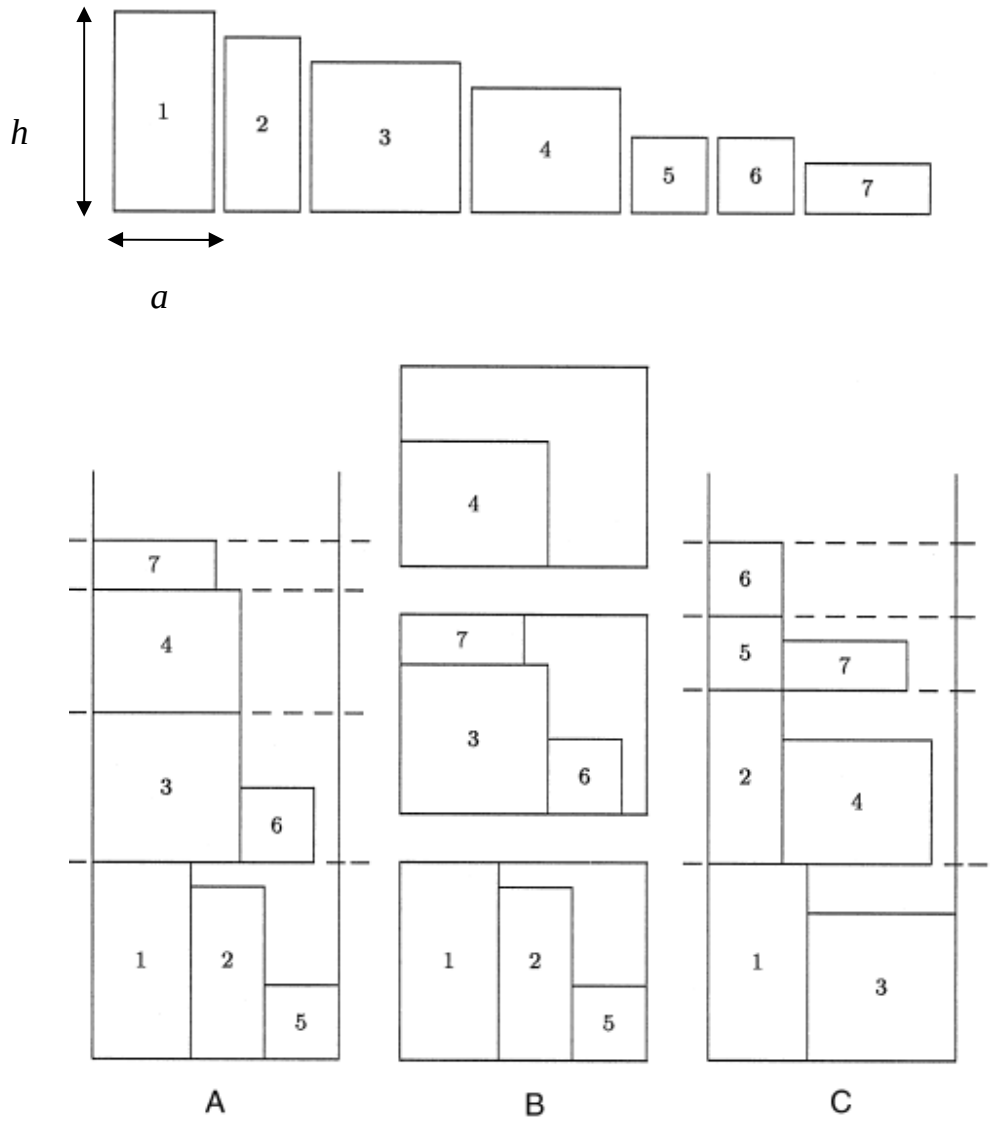


Fig. 3.2 - Una instancia de *binpacking* 2D con m=7, (A) Empaquetado en franjas producido por FBS_{OG}; **(B)** Solución en cajas encontrada por FBS_{OG} y FFF_{OG}; **(C)** Empaquetado en tiras dado por KP_{OG}

3.2.4 Algoritmo FC_{RG} (*Floor-Ceiling* *Rotation Guillotine*)

Otra evolución diferente del algoritmo en estantes de dos fases es el que propone Lodi, Martello, y Vigo[56] para el problema *bin packing* 2D donde la rotación de las piezas se permite y el corte es guillotina, este algoritmo se conoce como algoritmo *Floor-ceiling* (piso-techo).

Inicia ordenando los ítems respecto al lado más pequeño del ítem en orden no creciente y orientándolos horizontalmente. La principal peculiaridad esta en la forma en que los estantes en tiras (*strip shelves*) son empacados en la primera fase. El algoritmo ubica los ítems de izquierda a derecha, con el lado base sobre la línea de base del estante (esto es el piso). Adicionalmente, el algoritmo puede ubicar los ítems de derecha a izquierda con su lado tope sobre el techo del estante (esto es la línea horizontal definida por el borde (lado tope) del ítem mas grande empaquetado en el estante).
Cuando un ítem se empaqueta se consideran: 1) La evaluación de la posición de rotación y 2) La restricción guillotina; por consiguiente la posición (rotación) de un ítem posiblemente se modifique.
En la segunda fase, los estantes son empaquetados en cajas usando un algoritmo exacto para el problema de empaquetamiento en cajas de una dimensión. Por consiguiente su complejidad es no polinomial.

Podrían hacerse otras aproximaciones a partir de este algoritmo; sí no se considera 2), se puede construir patrones no guillotinables; sí no se considera 1), entonces se puede construir patrones en los que rotación de las piezas no son permitidas; y no considerando 1) y 2) se consigue patrones no guillotinables donde la rotación de las piezas no se permite.

3.2.5 Algoritmo KP_{OG} (*Knapsack Problem* *Oriented Guillotine*)

Este algoritmo ha sido propuesto por Lodi, Martello y Vigo [57]. En el algoritmo el empaquetamiento en los estantes son determinados resolviendo una serie de problemas de la mochila 0-1 (KP01), en el cual un elemento es cada ítem no empaquetado, con utilidad

igual al área del ítem, costo igual al ancho del ítem y la capacidad de la mochila igual al ancho del estante menos los anchos de los ítems ya empacados en ella.

En cada iteración de la primera fase empaqueta en franjas, inicializando un nuevo estante con el ítem más alto no empaquetado, entonces el empaquetado de los ítems en el estante se completa resolviendo un problema KP01.

En la segunda fase resuelve el problema BP1D para los S elementos (estantes generados).

Algoritmo KP_{OG}

Inicio
Ordenar los ítems de acuerdo al valor *hj* en forma nocreciente;
Proceso
Fase1
repetir
Abrir un nuevo estante dentro de la franja empacando el primer ítem más alto no empacado;
Resolver la instancia del KP01 asociado con el estante;
Empacar los ítems seleccionados sobre el estante
Hasta que todos los ítems son empaquetados;

Sea $\bar{h}_1,...,\bar{h}_s$ las alturas de los estantes resultantes,
Fase2
Determine la solución finita para la caja resolviendo la instancia del BP1D

el cual tiene *s* elementos, con los valores asociados $\bar{h}_1,...,\bar{h}_s$, y capacidad H
Fin.

En el ejemplo de la figura 3.2, el empaque en franjas (*strip packing*) producido por el algoritmo KP_{OG} se da en la figura 3.2 C; la solución en cajas finitas se da cuando los dos primeros estantes se empaquetan en dos cajas distintas y los estantes restantes en una caja.

El algoritmo también presenta complejidad no polinomial, la complejidad de la mochila 0-1 Lodi [57], propone una implementación eficiente interrumpiendo las rutinas no polinomiales asociados a los problemas PK01 y BP1D en un número (pequeño) fijo de iteraciones.

3.2.6 Algoritmo KP_{RG} (*Knapsack Problem* $_{Rotation\ Guillotine}$)

Este algoritmo es una variación del algoritmo KP_{OG} en el sentido de explotar la posible rotación de los ítems.

Inicia ordenando los ítems de acuerdo a su arista más pequeña y orientándolos horizontalmente. Esta orientación siempre se usa cuando se inicia un estante.

Para cada estante de altura h^*, la instancia del problema KP01 incluye todos los ítems no empacados con orientación vertical si el tamaño del ítem no supera h^* u horizontal en otro caso. Una solución en cajas finita se obtiene resolviendo el problema BP1D a partir de los estantes resultantes. Otra solución alternativa se determina si se construyen instancias de un problema $BP2D_{OG}$, considerando los estantes como pseudo-ítems de dimensiones la altura del estante y el espacio que ocupan horizontalmente, entonces, para cada pseudo-ítem se escoge una orientación vertical si el lado más grande no excede la altura de la caja. La solución para esta instancia resultante se obtiene al ejecutar el algoritmo KP_{OG}. Se selecciona la mejor solución final.

Algoritmo KP_{RG}

Inicio
Ordenar los ítems por su valor min{*wj*, *hj*} en forma no creciente, y orientarlos horizontalmente;

Proceso
Fase 1;
Repetir
Abrir un nuevo estante dentro de la franja por empaquetando horizontalmente el primer ítem no empacado;
Orientar apropiadamente cada ítem no empacado;
Resolver la instancia KP01 resultante, y empacar los ítems seleccionados
Hasta que todos los ítems son empacados;
Sea $\bar{w}_i$ y $\bar{h}_i$ $(i = 1,..., s)$ el tamaño de los *s* resultantes estantes;
Fase 2;
Determine una solución para la caja finita, resolviendo la asociada instancia del BP1D
Sea *z*1 el valor de la solución obtenida para la instancia (inicial) del $BP2D_{RG}$;
Fase 3;
Defina *s* pseudo-ítems teniendo tamaños $\bar{w}_i$ y $\bar{h}_i$ $(i = 1,..., s)$;
Oriente verticalmente cada pseudo-ítem *i* tal que max { $\bar{w}_i$, $\bar{h}_i$ } = *H*;
Ejecute el algoritmo KP_{OG} para la instancia $BP2D_{OG}$ producida por los pseudo-ítems;
Sea *z*2 el valor de la solución resultante;
Sea *z* := min{*z*1, *z*2}
Fin.

En el ejemplo de la figura 3.3, considere que los ítems pueden ser rotados. Los ítems son ordenados por KP_{RG} como (3, 4, 1, 2, 5, 6, 7), con los ítems 1 y 2 rotados en 90°. La figura 3.3 muestra la solución encontrada en cada fase de KP_{RG}. Note que en la fase 2 la solución es de tres cajas mientras que en la fase 3 produce 2 cajas (solución óptima).
La complejidad de este algoritmo, también no es polinomial.

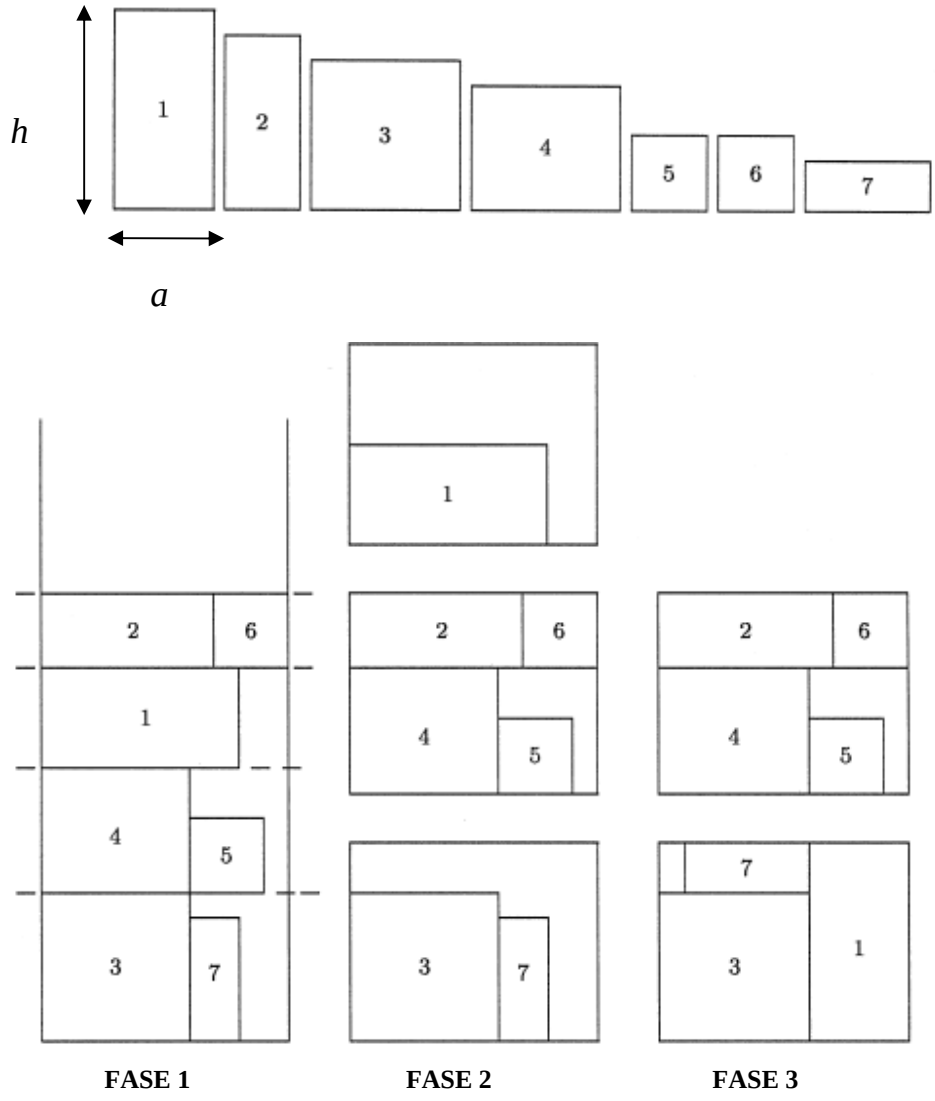


Fig. 3.3 Soluciones producidas por el algoritmo KP_{RG}

3.2.7 Algoritmo AD_{OF} (*Alternate Direction* *Oriented Free*)

Muy pocas heurísticas han sido presentadas para el problema $BP2D_{OF}$ (*bin packing* de dos dimensiones con objetos orientados y corte no guillotina); ésta es una de ellas y utiliza la forma peculiar de empacar del algoritmo FC_{RG} haciendo estos empaques en ambas direcciones, propuesta por Lodi y otros [56]. El algoritmo no usa estantes pero explota la factibilidad de patrones no guillotinables, empacando los ítems dentro de la caja en direcciones alternantes, esto es, de izquierda a derecha y de derecha a izquierda en la posición mas baja posible y así, en la adelante.

Inicia ordenando los ítems por altura en orden no creciente y calculando un límite inferior L_0 sobre el valor de la solución óptima. Un límite trivial es $L_0 = \left\lceil \sum_{j=1,m} a_j h_j / AH \right\rceil$; mejores límites pueden ser encontrados en Martello y Vigo [62]. Entonces, L_0 cajas son abiertas y un subconjunto de ítems son empacados (al fondo) siguiendo una política BFD. Los restantes ítems son empacados dentro de los subsiguientes niveles (fajas) de acuerdo a la dirección actual asociada con la caja. A saber, si la dirección es de "izquierda a derecha" (respectivamente de "derecha a izquierda"): 1) El primer ítem de la faja es empacado con su lado izquierdo (respectivamente derecho) tocando el lado izquierdo (respectivamente derecho) de la caja (objeto grande), en la mas baja posición; 2) Cada subsiguiente ítem es empacado con su lado izquierdo (respectivamente derecho) tocando el lado derecho (respectivamente izquierdo) de la caja anteriormente empacada en la faja, en la posición más baja posible.

Observe que los ítems empacados (de izquierda a derecha) en el paso inicial constituyen la primera faja de cada caja. La dirección asociada a todas las cajas después de este paso es de derecha a izquierda. En la fase iterativa del algoritmo, para cada caja se examina todos los ítems no empacados, los posibles empaques de ellos en la dirección actual y cambio de dirección cuando ningún ítem puede ser empacado en la faja actual. Tan pronto como ningún ítem puede ser empacado en cualquier dirección, se mueve a la siguiente caja, o se inicializa una nueva caja vacía.

Algoritmo AD_{OF}
Inicio
Ordenar los ítems de acuerdo al valor de h_j en forma no creciente

Proceso
Fase 1;
Calcule un limite inferior L_0 sobre el valor de la solución óptima,
y abra L_0 cajas vacías;
Para *j:* 1 a *m* **hacer**
  Si ítem *j* puede ser empacado en el fondo de alguna caja
  entonces
    empaquete *j*, alineado a la izquierda, sobre el fondo de la caja cuyo
    espacio residual horizontal es el mínimo;
Fase 2;
i := 0;
Repetir
 i := *i* + 1, *Derecha-izquierda* := *verdadero*, *n-fracasos* := 0;
  Repetir
    sea *j* el primer ítem no empacado el cual puede ser empacado en la caja *i* de acuerdo al actual *derecha-*

```
    izquierda;
   sino alguno;
    if i = vacio entonces
          n-fracasos := n-fracasos+ 1,
          derecha-izquierda := not derecha-izquierda
       caso contrario
          empaque j dentro de la caja i de acuerdo  a derecha-izquierda
          n-fracasos := 0;
  hasta n-fracasos= 2
hasta todos los ítems son empacados
Fin.
```

Como ejemplo considere la siguiente instancia de 12 ítems mostrado en la parte superior de la figura 3.4, las cajas son de tamaño $A = 10$ y $H = 8$ y los ítems con los siguientes tamaños $(a_1,h_1) = (4,6)$; $(a_2,h_2) = (4,4)$; $(a_3, h_3) = (8,3)$; $(a_4, h_4) = (4,3)$; $(a_5, h_5) = (4,3)$; $(a_6, h_6) = (4,3)$; $(a_7, h_7) = (1,3)$; $(a_8, h_8) = (6,2)$; $(a_9, h_9) = (2, 2)$; $(a_{10}, h_{10}) = (9,2)$; $(a_{11}, h_{11}) = (9, 2)$; $(a_{12}, h_{12})=(3, 1)$.

El límite superior da como resultado 2 cajas. En la parte inferior de la figura 3.4 se muestra el resultado; en la fase 1 los ítems 1, 2, 3, 7 y 9 son empaquetados en las dos cajas, en la fase 2 una faja es adicionada a la caja 1 ítems 4 y 8, y dos fajas a la caja 2, (5, 6) y 10. Una tercera caja es entonces abierta y empaquetado los restantes ítems en dos fajas. Este algoritmo puede ser implementado en complejidad de tiempo $O(m^3)$.

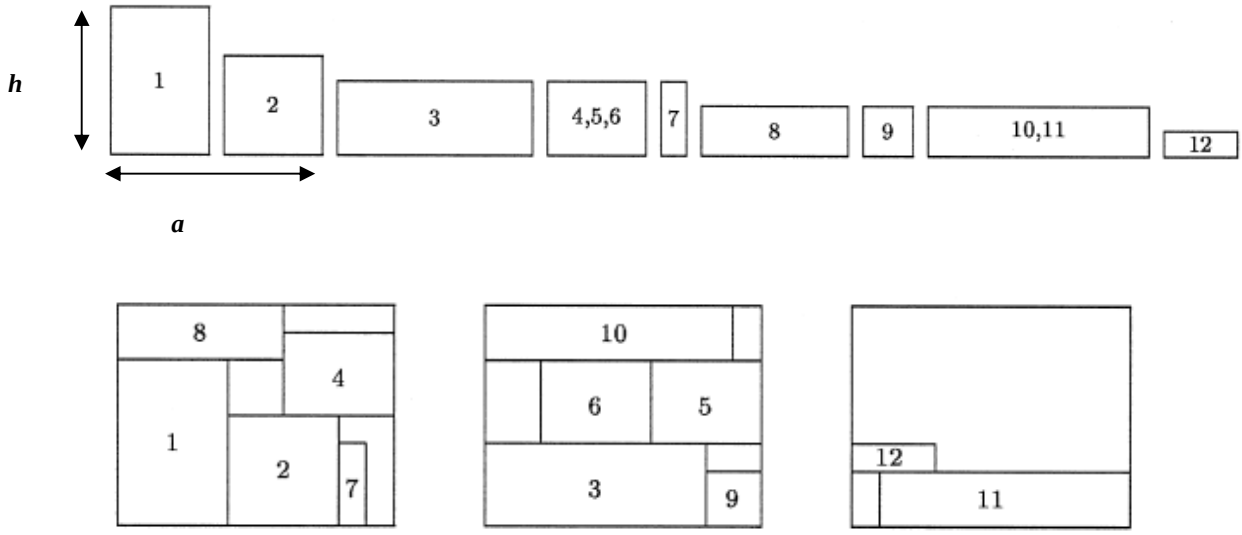


Fig. 3.4 En la parte superior instancia con n=12. Abajo solución encontrada por el algoritmo AD_{OF} para el *binpacking* 2D

3.2.8 Algoritmo TP_{RF} (*Touch Perimeter* $_{Rotation\ Free}$)

También propuesta por Lodi y otros [57]. Inicia ordenando los ítems de acuerdo a su área por orden no creciente (rompiendo empates al ordenar no crecientemente por min{a_j, h_j}), y orientándolos horizontalmente. Un límite inferior L_0 respecto al valor de la solución optima se calcula y L_0 cajas vacías son abiertas. Por ejemplo $L_0 = \left\lceil \sum_{j=1,m} a_j h_j / AH \right\rceil$.

El algoritmo empaca un ítem cada vez, en una caja ya abierta o abriendo una nueva. El primer ítem empacado en una caja es colocado siempre en la esquina izquierda. Cada subsiguiente ítem es empacado con su base coincidiendo con la base de la caja o con el tope de cualquier otro ítem y con su lado izquierdo coincidiendo con el lado izquierdo de la caja o el lado derecho de otro ítem, (posición normal).

La selección de la caja y de la posición del ítem a empaquetar se efectúa evaluando un puntaje definido como el porcentaje del perímetro del ítem que toca la caja y a otro(s) ítem(s) ya empacado(s). Esta estrategia favorece patrones donde el ítem empacado no genera pequeñas áreas libres, los cuales pueden ser difíciles de usar en futuras ubicaciones.

Para cada empaque de un ítem el puntaje referente a las dos orientaciones del ítem (si ambas son factibles) se evalúa y se selecciona el de más alto valor. Los puntajes empatados son rotos escogiendo la caja con mayor área ya ocupada.

Algoritmo TP_{RF}

Inicio
Ordene los ítems por el valor $a_j h_j$ en forma no creciente, y orientándolos *horizontalmente*;

Proceso
Fase 1;
  Calcule un límite inferior L_0 del valor de la solución óptima,
  y abra L_0 cajas vacías;
Fase 2;
  Para *j* := 1 a *m* **hacer**
    puntaje := 0;
    Para cada item *j* a empaquetar en *posición normal* en una caja abierta **hacer**
      Sea *puntaje*1 y *puntaje*2 los puntajes asociados con las dos orientaciones del item *j*;
      puntaje := max{*puntaje*, *puntaje*1, *puntaje*2}
    Si *puntaje* > 0 **entonces**
      Empaquetar item *j* en la caja, según posición y orientación correspondiente a *puntaje*
    caso contrario
      Abrir una nueva caja, y empaquetar *horizontalmente* ítem *j*
  Fin de para
Fin.

En la parte superior de la figura 3.5, se considera una instancia de $BP2D_{RF}$ (*bin packing* con rotación de ítems y corte no guillotina); las cajas son de tamaño $A = 10$ y $H = 8$ y los ítems tienen los siguientes tamaños $(a_1,h_1) = (4,6)$; $(a_2,h_2) = (4,4)$; $(a_3, h_3) = (8,3)$; $(a_4, h_4) = (4,3)$; $(a_5, h_5) = (4,3)$; $(a_6, h_6) = (4,3)$; $(a_7, h_7) = (1,3)$; $(a_8, h_8) = (6,2)$; $(a_9, h_9) = (2, 2)$; $(a_{10}, h_{10}) = (9,2)$; $(a_{11}, h_{11}) = (9, 2)$; $(a_{12}, h_{12})=(3, 1)$.

El algoritmo TP_{RF} ordena los ítems por el área como (1, 3, 10, 11, 2, 4, 5, 6, 8, 9, 7, 12), con ítems 1 y 7 rotados en 90° (posición horizontal). Luego encaja cada uno de los ítems según la ordenación en las 2 cajas abierta (valor de L_0), considerando la posición (horizontal o vertical) y los porcentaje de perímetro que toca el ítem con respecto a la caja u otros ítems. La solución se muestra en la parte inferior de la figura 3.5.

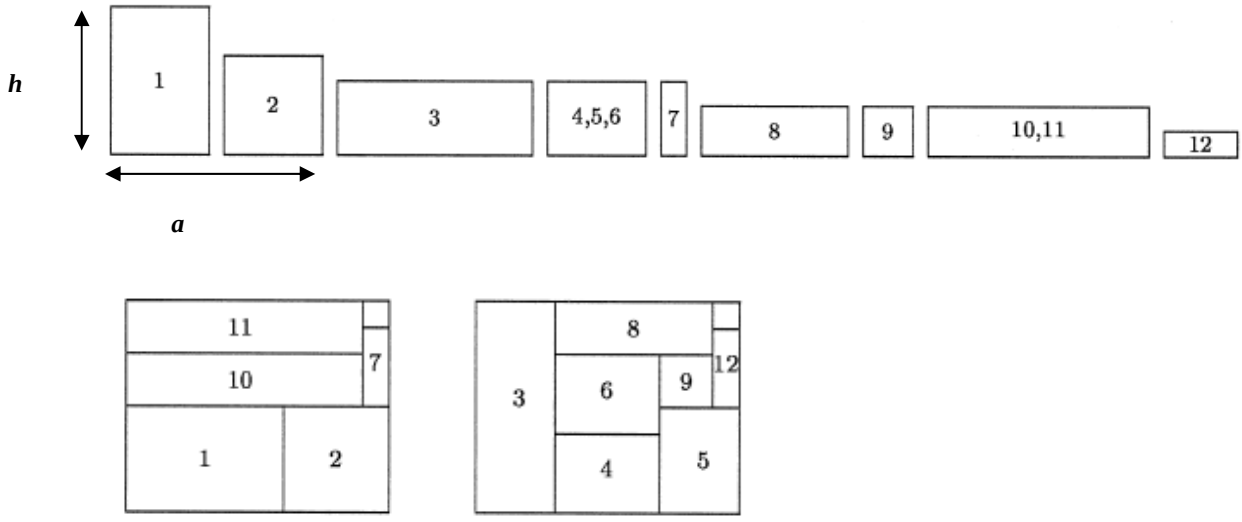


Fig. 3.5 Solución encontrada por el algoritmo TP_{RF}

El patrón de la segunda caja es un corte no guillotina. La complejidad del algoritmo es $O(m^3)$.

3.2.9 **Algoritmo** FFD-CUT-2D$_{RG}$ (*First Fit Decresing-Cutting-2D$_{Rotation\ Guillotine}$*)

Un abordaje un tanto diferente es el que presenta (Mauricio y Delgadillo [64]). No sigue la forma usual de resolver el problema en dos fases, utiliza el concepto goloso para empaquetar los ítems, esto es, se selecciona primero los ítems que utilizan mayor capacidad de la caja o láminas, por tanto la ordenación de los ítems es no creciente por su área. Supone que los ítems admiten rotación; entonces los ítems pueden ser colocados en forma vertical u horizontal, la elección de sí el ítem es colocado en forma horizontal o vertical depende del mayor número de ítems del mismo tamaño que encajarían en la lámina en la posición que se evalúa, esto es: $Max\left\{\left\lfloor \frac{L}{l_k} \right\rfloor \left\lfloor \frac{A}{a_k} \right\rfloor, \left\lfloor \frac{L}{a_k} \right\rfloor \left\lfloor \frac{A}{l_k} \right\rfloor\right\}$. Debido a que el tipo de corte es por guillotina, cuando se coloca un ítem en la lámina se genera dos láminas residuales potenciales, horizontal o vertical la selección del corte se hace evaluando si el siguiente ítem encaja mejor (aprovecha mas) la lámina horizontal o vertical; una vez efectuado el corte, se obtiene una lamina residual fija; esta lámina es considerada como una nueva lámina de tamaño diferente pero con prioridad para ser utilizada en los subsiguientes ítems a encajar.

El algoritmo es de tipo recursivo; esto es en cada iteración se selecciona cual es la mejor lámina en la que encaja el ítem siguiente, en la lámina horizontal potencial o vertical potencial o en lamina residual fija.

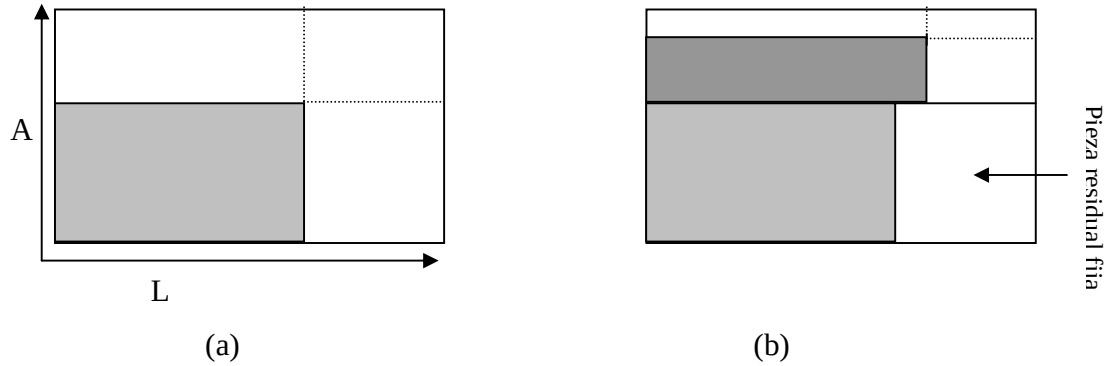


Fig. 3.6 Generación de una pieza residual fija. (a) lámina sin pieza residual fija. (b) lámina con pieza residual fija.

Un concepto, necesario que se debe especificar es que una lámina cualquiera puede ser identificada por las coordenadas de sus vértices inferior izquierdo y superior derecho, como se indica en la figura 3.7.

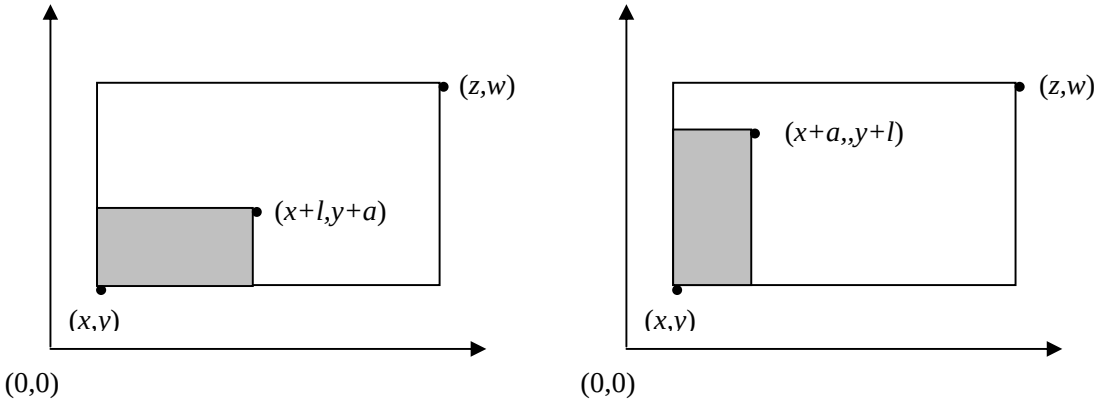


Fig. 3.7. Rotación de una pieza.

Algoritmo FFD-CUT-2D$_{RG}$

Inicio
Leer la instancia($m,l_1,a_1,...,l_m,a_m,L,A$)

Ordenar los ítems por sus áreas, en forma nocreciente, esto es, $l_1a_1 \geq l_2a_2 \geq ... \geq l_ma_m$

Sea la primera lámina horizontal $H^1 := \varnothing;$ la primera lámina vertical $V^1 := \varnothing;$

Sea la primera lámina fija, $F^1 := \{((0,0),(H,W))\};$ la lámina a cortar $n := 0;$

Proceso
Para *k:=1,..,m* **hacer**
$B_k := (l_k, a_k);$ el ítem *k*
$i := Min\{j \in \{1,2,...,n+1\} : H^j \cup V^j \cup F^j \supseteq B_k\};$
Sí *i* = *n*+ 1 **entonces**
n:=n+ 1; $F^{n+1} := \{((0,0),(L,A))\};$
Corte-Lámina-Nueva(*B$_k$*,*n*);
Caso contrario
Corte_Lámina_Usada(*B$_k$*,*i*);
Fin-Si
Fin-Para

Corte-Lámina-Nueva(*ítem$_k$*,*i*);

Rotación (rotar, si ítem k colocado en la forma horizontal entra en la lámina un número mayor de veces que si colocado en la forma vertical)

Sí $Max\left\{\left\lfloor \frac{L}{l_k} \right\rfloor \left\lfloor \frac{A}{a_k} \right\rfloor, \left\lfloor \frac{L}{a_k} \right\rfloor \left\lfloor \frac{A}{l_k} \right\rfloor\right\} = \left\lfloor \frac{L}{a_k} \right\rfloor \left\lfloor \frac{A}{l_k} \right\rfloor$ **entonces** $(l_k, a_k) := (a_k, l_k)$;

Proceso de Corte (determina cortes horizontales y verticales)

$B_k := ((0,0),(l_k, a_k))$; $B^i := \{B_k\}$; B^i denota el conjunto de items atendidos desde la lámina "*i*".

Sí $(a_k < A)$ **entonces** $H_k^i := ((0, a_k),(L, A))$, $H^i := \{H_k^i\}$;

Sí $(l_k < L)$ **entonces** $V_k^i := ((l_k, 0),(L, A))$, $V^i := \{V_k^i\}$;

$F^i := \varnothing$;

Corte_Lámina_Usada(B_k,*i*);

Estrategia de selección (Selecciona una lámina de las láminas residuales, horizontales, verticales y fijas en la que mejor encaje el ítem k, según formula anterior

Seleccione una lámina $P = (\bar{l}, \bar{a})$ de $H^i \cup V^i \cup F^i$, tal que verifique:

$$Max\left\{\left\lfloor \frac{\bar{l}}{l_k} \right\rfloor \left\lfloor \frac{\bar{a}}{a_k} \right\rfloor, \left\lfloor \frac{\bar{l}}{a_k} \right\rfloor \left\lfloor \frac{\bar{a}}{l_k} \right\rfloor\right\} = Max_{\tilde{B}=(\tilde{l},\tilde{a})\in H^i \cup V^i \cup F^i} \left\{ Max\left\{\left\lfloor \frac{\tilde{l}}{l_k} \right\rfloor \left\lfloor \frac{\tilde{a}}{a_k} \right\rfloor, \left\lfloor \frac{\tilde{l}}{a_k} \right\rfloor \left\lfloor \frac{\tilde{a}}{l_k} \right\rfloor\right\}\right\}$$

Sea $P := ((p_1, p_2),(p_3, p_4))$;

Rotación

Hacer la rotación del ítem, dentro de la lámina seleccionada, si mejor se aprovecha según formula.

Sí $Max\left\{\left\lfloor \frac{\bar{l}}{l_k} \right\rfloor \left\lfloor \frac{\bar{a}}{a_k} \right\rfloor, \left\lfloor \frac{\bar{l}}{a_k} \right\rfloor \left\lfloor \frac{\bar{a}}{l_k} \right\rfloor\right\} = \left\lfloor \frac{\bar{l}}{a_k} \right\rfloor \left\lfloor \frac{\bar{a}}{l_k} \right\rfloor$ **entonces** $B_k := (l_k, a_k) := (a_k, l_k)$;

Proceso de Corte (Cortar sobre la lámina seleccionada)

Corte sobre lámina residual horizontal,

Sí $P \in H^i$, **entonces**

Sea $H_j^i = P$ para algún $j \in \{1,..., k-1\}$

$H^i := H^i - \{H_j^i\}$;

Si $\exists\ V_j^i$, **entonces** sea $V_j^i = ((v_1, v_2),(v_3, v_4))$,

$F_k^i := ((v_1, v_2),(p_3, p_2))$; $F^i = F_j^i \cup \{F_k^i\}$; $V^i := V^i - \{V_j^i\}$;

* **Corte sobre lámina residual vertical**

Sí $P \in V^i$, **entonces**

sea $V_j^i = P$, para algún $j \in \{1,..., k-1\}$

$V^i := V^i - \{V_j^i\}$;

Si $\exists\ H_j^i$, **entonces** sea $H_j^i = ((h_1, h_2),(h_3, h_4))$,

$F_k^i := ((h_1, h_2),(p_1, p_4))$; $F^i = F_j^i \cup \{F_k^i\}$; $H^i := H^i - \{H_j^i\}$;

* **Corte sobre lámina residual fija**

Sí $P \in F^i$, **entonces**

sea $F_j^i = P$, para algún $j \in \{1,...,k-1\}$

$F^i = F^i - \{F_j^i\};$

* **Cortes y actualización: lámina horizontal y vertical.**

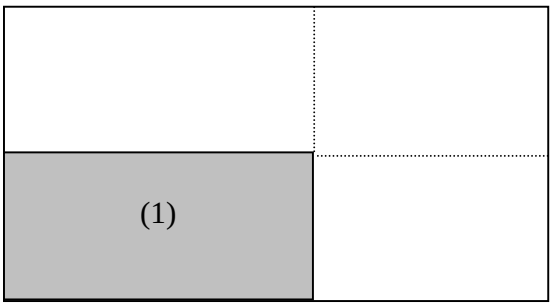
$B_k^i := ((p_1, p_2),(p_1 + l_k, p_2 + a_k));\ B^i := B^i \cup \{B_k^i\};$

Sí $(p_2 + a_k < p_4)$ **entonces** $H_k^i := ((p_1, p_2 + a_k),(p_3, p_4)),\ H^i := H^i \cup \{H_k^i\};$

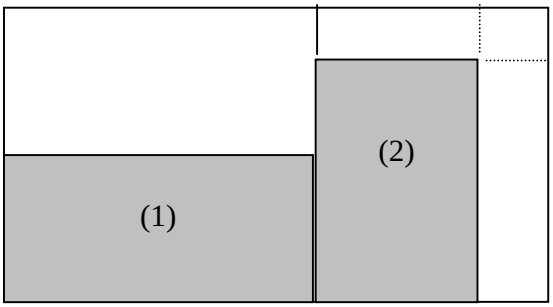
Sí $(p_1 + l_k < p_3)$ **entonces** $V_k^i := ((p_1 + l_k, p_2),(p_3, p_4)),\ V^i := V^i \cup \{V_k^i\};$

La complejidad del algoritmo radica en el procedimiento Corte_Lámina_Usada, esta tiene complejidad $O(s_i)$, donde s_i es el número de items atendidos desde la lámina "*i*"; por tanto el algoritmo es de baja complejidad.

Un ejemplo del funcionamiento de este algoritmo se muestra en las figuras 3.8 para la instancia L= 74, A = 40, y los ítems ya ordenados con los siguientes tamaños (l_1,a_1) = (42,20); (l_2,a_2) = (33,22); (l_3, a_3) = (30,13); (l_4, a_4) = (14,10); (l_5, a_5) = (14,10); (l_6, a_6) = (17,7); (l_7, a_7) = (17,7); (l_8, a_8) = (17,7). Los ítems son enumerados del 1 al 8 respectivamente.

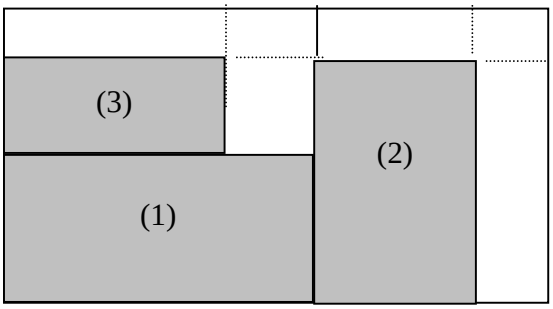


$$Max\left\{\left\lfloor \frac{74}{42} \right\rfloor \left\lfloor \frac{40}{20} \right\rfloor, \left\lfloor \frac{74}{20} \right\rfloor \left\lfloor \frac{40}{42} \right\rfloor\right\} = 2$$



$$Max\left\{\left\lfloor \frac{74}{33} \right\rfloor \left\lfloor \frac{20}{22} \right\rfloor, \left\lfloor \frac{74}{22} \right\rfloor \left\lfloor \frac{20}{33} \right\rfloor\right\} = 0$$

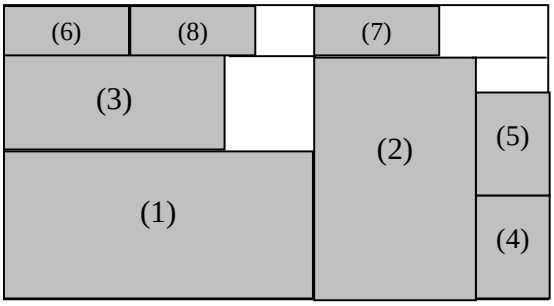
$$Max\left\{\left\lfloor \frac{32}{33} \right\rfloor \left\lfloor \frac{40}{22} \right\rfloor, \left\lfloor \frac{32}{22} \right\rfloor \left\lfloor \frac{40}{33} \right\rfloor\right\} = 1 \quad ®$$



$$Max\left\{\left\lfloor\frac{42}{30}\right\rfloor\left\lfloor\frac{20}{13}\right\rfloor,\left\lfloor\frac{42}{13}\right\rfloor\left\lfloor\frac{20}{30}\right\rfloor\right\}=1$$

$$Max\left\{\left\lfloor\frac{32}{30}\right\rfloor\left\lfloor\frac{7}{13}\right\rfloor,\left\lfloor\frac{32}{13}\right\rfloor\left\lfloor\frac{7}{30}\right\rfloor\right\}=0$$

$$Max\left\{\left\lfloor\frac{10}{30}\right\rfloor\left\lfloor\frac{40}{13}\right\rfloor,\left\lfloor\frac{10}{13}\right\rfloor\left\lfloor\frac{40}{30}\right\rfloor\right\}=0$$



$$Max\left\{\left\lfloor\frac{25}{17}\right\rfloor\left\lfloor\frac{7}{7}\right\rfloor,\left\lfloor\frac{25}{7}\right\rfloor\left\lfloor\frac{7}{17}\right\rfloor\right\}=1$$

$$Max\left\{\left\lfloor\frac{12}{17}\right\rfloor\left\lfloor\frac{13}{7}\right\rfloor,\left\lfloor\frac{12}{7}\right\rfloor\left\lfloor\frac{13}{17}\right\rfloor\right\}=0$$

$$Max\left\{\left\lfloor\frac{15}{17}\right\rfloor\left\lfloor\frac{7}{7}\right\rfloor,\left\lfloor\frac{15}{7}\right\rfloor\left\lfloor\frac{7}{17}\right\rfloor\right\}=0$$

$$Max\left\{\left\lfloor\frac{10}{17}\right\rfloor\left\lfloor\frac{5}{7}\right\rfloor,\left\lfloor\frac{10}{7}\right\rfloor\left\lfloor\frac{5}{17}\right\rfloor\right\}=0$$

Fig.3.8 Ejecución del algoritmo FFD-CUT-2D$_{RG}$, donde ® denota rotación del item.

3.2.10 Algoritmo BFD-CUT-2D$_{RG}$ (*Best First Decreasing-Cutting-2D$_{Rotation\ Guillotine}$*)

Otro algoritmo presentado por Mauricio y Delgadillo [64], es la adaptación del clásico algoritmo BFD para el problema de corte 1D; utilizando los conceptos, procedimientos y recursividad mostrada en el algoritmo FFF-CUT-2D$_{RG}$.

Difiere del anterior en la selección de la lámina usada, esta debe ser aquella lámina que presente el mejor valor para la función objetivo y en el que pueda encajar el item requerido; en caso que no exista tal lámina usada, se selecciona una nueva lámina. Este proceso se repite para cada item, hasta atender a todos ellos.

La función objetivo a optimizar es el área residual de la lámina. Denotemos el área residual (esto es, área no usada) de una lámina "*i*" por r_i.

Algoritmo BFD-CUT-2D$_{RG}$

Inicio

Leer la instancia($m,l_1,a_1,...,l_m,a_m,L,A$)

Ordenar los ítems por sus áreas, en forma nocreciente, esto es, $l_1a_1 \geq l_2a_2 \geq ... \geq l_ma_m$

Sea la primera lámina horizontal $H^1 := \varnothing;$ la primera lámina vertical $V^1 := \varnothing;$

Sea la primera lámina fija, $F^1 := \{((0,0),(H,W))\};$ la lámina a cortar $n := 0;$

Proceso

Para *k:=1,..,m* **hacer**

$B_k := (l_k, a_k);$ item k

$i := ArgMin\{r_j : H^j \cup V^j \cup F^j \supseteq B_k,\ j \in \{1,2,...,n+1\}\};$

Sí *i* = *n*+ 1 **entonces**

n:=n+1 ; $F^{n+1} := \{((0,0),(L,A))\};$

Corte-Lámina-Nueva(*B$_k$*,*n*);

Caso contrario

Corte_Lámina_Usada(*B$_k$*,*i*);

Fin-Si

Fin-Para

La complejidad del algoritmo es la misma que presenta el algoritmo FFD-CUT-2D$_{RG}$. Estos dos últimos algoritmos pueden aplicarse a problemas de corte donde la rotación no es permitida desde que no se considere rotación en los procedimientos de corte-lámina-nueva y corte-lámina-usada.

CAPÍTULO IV

META HEURÍSTICAS

El término meta heurístico fue introducido en 1986 por Fred Glover [37]; el prefijo meta significa "por encima" o "mas allá", en este sentido los procedimientos meta heurísticos se sitúan conceptualmente "por encima" de los heurísticos en el sentido que guían el diseño de estos. Según Osman y Kelly [71] meta heurísticas se define como: *"Una clase de métodos aproximados que están diseñados para resolver problemas difíciles de la optimización combinatoria, en los que los heurísticos clásicos no son efectivos"*.

Las meta heurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos.

Debido a que las meta heurísticas son estrategias para diseñar procedimientos heurísticos, una clasificación natural de meta heurísticas se establece en primer lugar en función del tipo de procedimiento a los que se refiere. Algunos de estos tipos fundamentales son:

(Belén Melián, José A. Moreno Pérez, J. Marcos Moreno Vega [66])

- **Meta heurísticas de relajación**: Se refieren a los procedimientos de resolución de problemas que utilizan relajación del modelo original, y cuya solución facilita la solución del problema original.
- **Meta heurísticas constructivas**: Se orientan a los procedimientos que tratan de la obtención de una solución a partir del análisis y selección paulatina de las componentes que la forman.
- **Meta heurísticas de búsqueda**: Guían los procedimientos que usan transformaciones o movimientos para recorrer el espacio de soluciones alternativas y explotar las estructuras de entornos asociados.
- **Meta heurísticas evolutivas**: Se enfocan a los procedimientos basados en conjuntos de soluciones que evolucionan sobre el espacio de soluciones.

Algunas meta heurísticas surgen de la combinación de meta heurísticas de distinto tipo, como la meta heurística GRASP, que combina una fase constructiva con una fase de

búsqueda de mejora. Otras meta heurísticas se centran en el uso de algún tipo de recurso computacional o formal especial como las redes neuronales, los sistemas de hormigas o la programación por restricciones y no se incluyen claramente en ninguno de los cuatro tipos anteriores.

En general todas las meta heurísticas se pueden *concebir* como estrategias aplicadas a procesos de búsqueda, donde todas las situaciones intermedias en el proceso de resolución del problema se interpretan como elementos de un espacio de búsqueda, que se van modificando a medida que se aplican las distintas operaciones diseñadas para llegar a la resolución definitiva. Por ello, y porque los procesos de búsqueda heurística constituyen el paradigma central de las meta heurísticas, es frecuente interpretar que el término meta heurística es aplicable esencialmente a los procedimientos de búsqueda sobre un espacio de soluciones alternativas.

En el presente capitulo, mencionaremos las meta heurísticas más conocidas que se han utilizado para resolver el problema de C&E 2D, como: GRASP, búsqueda tabú, *Simulated Annealing*, algoritmo genético; previamente hacemos una introducción de estos métodos donde definimos los conceptos fundamentales que explican el mecanismo de acción de estas meta heurísticas.

4.1 GRASP (*Greedy Randomized Adaptive Search Procedures*)

Los métodos GRASP (Procedimientos de Búsqueda Adaptativa Aleatoria y Golosa) fueron desarrollados al final de la década de los 80 con el objetivo inicial de resolver problemas de cubrimientos de conjuntos (Feo y Resende, [27]; posteriormente GRASP fue introducido como una nueva técnica meta heurística de propósito general (Feo y Resende, [28]).

GRASP es un procedimiento de multi-arranque (también llamado *multi-start* ó *re-start*) en donde cada iteración consiste de una fase de construcción y una de mejora. En la fase de construcción se aplica un procedimiento heurístico constructivo para obtener una buena solución inicial. Esta solución se mejora en la segunda fase mediante un algoritmo de búsqueda local. La solución encontrada en cada fase es guardada. La mejor de todas las soluciones examinadas se da como resultado final.

En la fase de construcción se construye iterativamente una solución posible, considerando un elemento en cada paso según una función *greedy* (golosa ó voraz). Esta función mide el beneficio de añadir cada uno de los elementos según la función objetivo y elegir la mejor. Esta medida es miope pues no toma en cuenta qué ocurrirá en iteraciones sucesivas al realizar una elección en la presente iteración.

Se dice que el heurístico *greedy* se *adapta* porque en cada iteración se actualizan los beneficios obtenidos al añadir el elemento seleccionado a la solución parcial. Es decir, la evaluación que se tenga de añadir un determinado elemento a la solución en la iteración j, no coincidirá necesariamente con la que se tenga en la iteración $j + 1$.

Se dice que es aleatoria porque no selecciona el mejor candidato según la función *greedy* adaptada sino que, con el objetivo de diversificar y no repetir soluciones en dos construcciones diferentes, se construye una lista con los mejores candidatos de entre los que se toma uno al azar. Esta fase no garantiza el óptimo local.

En la fase de mejora se suele emplear un procedimiento de intercambio simple con el objeto de no emplear mucho tiempo en esta mejora; dado que GRASP se basa en realizar múltiples iteraciones y quedarse con la mejor, no es beneficioso detenerse demasiado en mejorar una solución dada.

Un esquema del funcionamiento global del algoritmo es:

Algoritmo GRASP
Inicio
Leer la instancia del problema
Proceso
Mientras (Condición de parada no sea satisfecha)
Fase Constructiva
Seleccionar una lista de elementos candidatos.
Considerar una Lista Restringida de los mejores Candidatos.
Seleccionar un elemento aleatorio de la Lista Restringida.
Fase de Mejora
Realizar un proceso de búsqueda local a partir de la
solución construida hasta que no se pueda mejorar más.
Actualización
Si la solución obtenida mejora a la mejor almacenada, **entonces** actualizarla.
Fin Mientras
Retorna mejor solución encontrada
Fin.

El algoritmo GRASP termina cuando algún criterio de parada tal como un número máximo de iteraciones haya ocurrido o alguna solución encontrada es satisfactoria.

Las implementaciones GRASP generalmente son robustas en el sentido de que es difícil el encontrar ejemplos patológicos en donde el método funcione arbitrariamente mal.
Algunas de las sugerencias de los autores para mejorar el procedimiento son:
- Se puede incluir una fase previa a la de construcción: una fase determinista con el objetivo de ahorrar esfuerzo a la fase siguiente.
- Si se conoce que ciertas subestructuras forman parte de una solución óptima, estas pueden ser el punto de partida de la fase constructiva.

Tal y como señalan Feo y Resende [28], una de las características más relevantes de GRASP es su sencillez y facilidad de implementación. Basta con fijar el tamaño de la lista de candidatos y el número de iteraciones para determinar completamente el procedimiento. De esta forma se pueden concentrar los esfuerzos en diseñar estructuras de datos para optimizar la eficiencia del código y proporcionar una gran rapidez al algoritmo, dado que este es uno de los objetivos principales del método.

El enorme éxito de este método se puede constatar en la gran cantidad de aplicaciones que han aparecido en los últimos años. Festa y Resende [29] comentan cerca de 200 trabajos en los que se aplica o desarrolla GRASP.

A seguir se muestra la meta heurística GRASP para el problema de C&P 2D presentada por Mauricio [65]. Mauricio, resalta la diferencia que existe entre la construcción de una solución utilizando criterio goloso y el criterio GRASP; en el primer caso la elección del siguiente ítem a empaquetar está dada por el ítem de mayor área ($l_k a_k := Max\{l_h a_h : h \in M\}$) como visto en las heurísticas FFD-CUT-2D$_{RG}$ y BFD-CUT-2D$_{RG}$.
En el segundo caso el procedimiento GRASP, construye una solución usando un criterio de selección golosa-aleatoria; el cual consiste de dos pasos: 1) Primero, construye un conjunto de ítems candidatos para ser parte de la solución, llamado de lista de candidatos restrictos (RCL), esto se hace mediante una relajación del criterio goloso; 2) Segundo, el ítem seleccionado para ser parte de la solución es obtenido desde RCL a través de una selección aleatoria. La lista RCL estará formada por los ítems que presentan un área próxima a la mayor área de los ítems por atender.

Criterio GRASP de selección del ítem

$\bar{s} := Max\{l_h a_h : h \in M\};$
$\underline{s} := Min\{l_h a_h : h \in M\};$
$RCL := \{h \in M : \bar{s} - \alpha(\bar{s} - \underline{s}) \le l_h a_h \le \bar{s}\};$
$k := Random(RCL);$

Donde, M =1, 2,…m es el conjunto de todos los ítems.
La lista RCL está formada por todos los ítems no atendidos cuyas áreas se encuentran entre la mayor área $\bar{s}$ de los ítems no atendidos (criterio goloso) y una relajación de este $\bar{s} - \alpha(\bar{s} - \underline{s})$. El parámetro α es llamado parámetro de relajación, cuyo valor varia entre 0 y 1, y debe ser introducido como un dato de entrada. Observe que, cuando $\alpha = 0$ el criterio de selección GRASP es exactamente el mismo criterio goloso; y cuando $\alpha = 1$ el criterio de selección GRASP se torna un criterio de selección totalmente aleatorio. Esto significa que la calidad de la solución dependerá del parámetro de relajación. Cuando se use valores para α próximos a cero se deberá obtener soluciones próximas a la solución golosa; sin embargo para valores próximos a uno, se deberá obtener soluciones muy diversas posiblemente indeseable (contraria al goloso).

En el problema de corte 2D, en donde el objetivo es determinar el menor número de láminas que satisfacen la producción de los ítems; el problema no es sólo saber en qué orden (o criterio) se debe tomar cada uno de los ítems para encajar (cortar); sino también el problema es determinar qué lámina se debe seleccionar para poder efectuar el corte. Los algoritmos FFD-CUT-2D$_{RG}$ y BFD-CUT-2D$_{RG}$, proporcionan criterios distintos de cómo hacer esta selección; el primero selecciona la primera lámina usada en que encaja el ítem a ser cortado; y el segundo selecciona la lámina usada que presente menor pérdida y en la que encaja el ítem. En ambos casos, si tal lámina usada no existe, se usará una lámina nueva. Si "k" es el ítem a ser atendido (B_k), entonces los criterios según FFD-CUT-2D$_{RG}$ y BFD-CUT-2D$_{RG}$, respectivamente son:

$i := Min\{j \in \{1,2,...,n+1\} : H^j \cup V^j \cup F^j \supseteq B_k\};$ (4.1)

$i := ArgMin\{r_j : H^j \cup V^j \cup F^j \supseteq B_k,\ j \in \{1,2,...,n+1\}\};$ (4.2)

donde r_j es el resto asociado a la lámina "j" y "n" es el número de láminas usadas.

A partir de estos criterios Mauricio [65] construye dos Algoritmos de construcción GRASP, no considera el literal "D" (*decreasing*), pues los ítems son atendidos en forma aleatoria.

Algoritmo FF-ConstrutionGRASP

Leer la instancia $(m,l_1,a_1,...,l_m,a_m,L,A)$

$M := \{1,2,...,m\}; \quad n := 0;$

$H^1 := V^1 := \varnothing; \quad F^1 := \{((0,0),(L,A))\};$

Mientras $M \neq \varnothing$

Criterio GRASP de selección del ítem

$B_k := (l_k, a_k)$

$i := Min\{j \in \{1,2,...,m+1\} : H^j \cup V^j \cup F^j \supseteq B_k\};$ (4.1)

Si $i = n+1$

entonces ***Corte_Lámina_Nueva(*** $B_k, n+1$***)***, $n:=n+1$, $F^{n+1} := \{((0,0),(L,A))\}$

caso contrario ***Corte_Lámina_Usada(*** B_k***,i)***;

$M := M - \{k\};$

Fin Mientras

Regresa$(n, B^i \ \ \forall i)$

El algoritmo BF-Construcción GRASP, difiere en que en la instrucción (4.1) es reemplazada por (4.2).

Algoritmo BF-ConstrutionGRASP

Leer la instancia $(m,l_1,a_1,...,l_m,a_m,L,A)$

$M := \{1,2,...,m\}; \quad n := 0;$

$H^1 := V^1 := \varnothing; \quad F^1 := \{((0,0),(L,A))\};$

Mientras $M \neq \varnothing$

Criterio GRASP de selección del ítem

$B_k := (l_k, a_k)$

$i := ArgMin\{r_j : H^j \cup V^j \cup F^j \supseteq B_k, \ j \in \{1,2,...,n+1\}\};$ (4.2)

Si $i = n+1$

entonces ***Corte_Lámina_Nueva(*** $B_k, n+1$***)***, $n:=n+1$, $F^{n+1} := \{((0,0),(L,A))\}$

caso contrario ***Corte_Lámina_Usada(*** B_k***,i)***;

$M := M - \{k\};$

Fin Mientras

Regresa$(n, B^i \ \ \forall i)$

La segunda componente de un algoritmo GRASP es un algoritmo de búsqueda local (*Local Search*), el cual tiene por objetivo mejorar la solución encontrada por el algoritmo de construcción GRASP. Existen diversas técnicas para construir una mejor solución a partir de una solución dada, entre ellas cabe destacar: Búsqueda tabú y Templado Simulado.

Mauricio [65] propone que el proceso de mejorar la solución encontrada en la fase de construcción del GRASP, sea; dado $S = \{B^1, B^2, ..., B^m\}$ una solución, una mejor solución a partir de *S*, es una solución que presente menor número de láminas. El procedimiento de mejoría consiste de tres pasos: 1) Primero, se determina la lámina que presenta mayor resto, el cual se denota por B_{r1}; 2) Segundo se determina la lámina que presenta mayor resto sin considerar B_{r1}, el cual se denota por B_{r2}; 3) Tercero, se realizan los intercambios de los ítems entre la lámina B_{r2} y los ítems uno a uno de las otras láminas restantes de *S*, de forma que el resto de B_{r2} o de las otras lámina restantes permita encajar un ítem de B_{r1}. Los pasos 2 y 3 se repetirán hasta que todos los ítems de B_{r1} hayan encajado en las otras láminas, esto es, hasta que no sea necesario usar la lámina B_{r1}. Si no es necesario usar B_{r1}, se habrá conseguido reducir el número de láminas en una unidad, esto es se habrá conseguido mejorar la solución *S*, y el procedimiento podrá repetirse a fin de seguir mejorando la solución, de lo contrario el procedimiento deberá terminar. No siempre es posible mejorar una solución dada, pues el algoritmo GRASP puede haber alcanzado la solución óptima.

Los criterios de parada más comunes en los algoritmos GRASP son: tiempo de procesamiento, número máximo de iteraciones, condición de optimalidad, condición de óptimo local y una combinación de las anteriores. Estos criterios debe ser introducidos como datos de entrada para el algoritmo GRASP; verificado el criterio de parada el algoritmo deberá terminar y reportar la mejor solución encontrada.

Otros, algoritmos GRASP para este problema son presentado por Beltrán Cano y otros [4].

4.2 BÚSQUEDA TABÚ (*Tabú Search*)

Los orígenes de la Búsqueda Tabú (TS) se sitúan en diversos trabajos publicados a finales de los 70 (Glover [36]). El nombre y la metodología fueron introducidos posteriormente en 1989 por Fred Glover [38]. Numerosas aplicaciones han aparecido en la literatura, así como artículos y libros para difundir el conocimiento teórico del procedimiento (Glover y Laguna [39]).

TS es una meta heurística que se utiliza para evitar que en el proceso de búsqueda de la solución, ésta se quede atrapada en un óptimo local, conduciendo la búsqueda local a una búsqueda del óptimo global; para esto utiliza algunos conceptos y principios de Inteligencia Artificial (IA), como es el concepto de memoria. La memoria proporciona información de lo sucedido no permitiendo que en la búsqueda se considere soluciones que ya han sido antes investigadas. En este sentido puede decirse que hay un cierto aprendizaje y que la búsqueda es inteligente. El principio de TS se resume (Rafael Martí [63]):

"Es mejor una mala decisión basada en información que una buena decisión al azar, ya que, en un sistema que emplea memoria*, una mala elección basada en una estrategia proporcionará claves útiles para continuar la búsqueda. Una buena elección fruto del azar no proporcionará ninguna información para posteriores acciones".*

Para entender el procedimiento de TS definimos los siguientes conceptos:

Sean *x, y;* dos soluciones cualesquiera del problema.

N(x) es la vecindad de *x,* definida como todas las soluciones *y* que se encuentran en el entorno de *x.*

N(x)* es la vecindad reducida de *x,* esto es $N^*(x) \subseteq N(x)$. Una reducción de *N(x)* se consigue no considerando en la vecindad algunas soluciones que se etiquetan como tabú, es decir $N^*(x) = N(x) \setminus T$.

Tabú: Es una etiqueta que se coloca a una solución visitada recientemente. El objetivo principal de etiquetar las soluciones visitadas como tabú es el de evitar que la búsqueda cicle.

Nivel de Aspiración: Se define como aquellas condiciones que, de satisfacerse, permitirían alcanzar una solución aunque tenga estatus tabú. Una implementación sencilla consiste en permitir alcanzar una solución siempre que mejore a la mejor almacenada, aunque esté

etiquetada tabú. De esta forma se introduce cierta flexibilidad en la búsqueda y se mantiene el propósito de alcanzar el óptimo global.

Atributo: es una característica, una propiedad o un concepto asociado a una solución o a un conjunto de soluciones.

Memoria a corto Plazo: Es la memoria de las soluciones visitadas recientemente, estas se guardan en una lista tabú T y su objetivo es explorar a fondo una región dada, del espacio de soluciones. La memoria puede ser expresada mediante atributos, esta forma de representar produce un efecto más sutil y efectivo en la búsqueda, ya que un atributo o grupo de atributos identifica a un conjunto de soluciones, de igual forma que los hiperplanos. En ocasiones se utilizan estrategias de listas de candidatos para restringir el número de soluciones examinadas en una iteración dada o para mantener un carácter agresivo en la búsqueda

Memoria a largo plazo: Almacena las frecuencias u ocurrencias de atributos en las soluciones visitadas tratando de identificar o diferenciar regiones. La memoria a largo plazo tiene dos estrategias asociadas: Intensificar y Diversificar la búsqueda

Intensificar: consiste en regresar a regiones ya exploradas para estudiarlas más a fondo. Para ello se favorece la aparición de aquellos atributos asociados a buenas soluciones encontradas.

Diversificar: consiste en visitar nuevas áreas no exploradas del espacio de soluciones. Para ello se modifican las reglas de elección para incorporar a las soluciones atributos que no han sido usados frecuentemente.

TS interactúa entre la memoria a corto plazo y la memoria a largo plazo de forma a alcanzar su objetivo. La siguiente tabla muestra los elementos de los dos tipos de memoria utilizado en TS.

Memoria	**Atributos**	**Estrategias**	**Ámbito**
Corto Plazo	Reciente	Tabú - Aspiración Listas de Candidatos	Local
Largo Plazo	Frecuente	Intensif. - Diversif.	Global

Tabla 4.1 Componentes de la Búsqueda Tabú.

El diseño de un algoritmo básico de TS para un problema de optimización cualquiera se da abajo, sin embargo existen otros elementos que deben ser considerados para construir algoritmos realmente potentes y eficaces, como estrategias de reinicio de la búsqueda. La introducción de estos es uno de los retos para los investigadores del área.

Algoritmo Tabú
K= 1
Mientras (no fin)
Identificar $N(s) \subseteq S$; (*S* = Espacio de soluciones, *N(s)* = Conjunto de vecinos de s)
Identificar $T(s) \subseteq N(s)$; (*T(s)* = conjunto tabú)
Identificar $A(s) \subseteq T(s)$; (*A(s)* = conjunto aspirante)
Seleccionar $s' \in (N(s_j) - T(s_j)) \cup A(s_j)$, para el cual
$F(s')$ es máxima
$s = s'$
$k = k + 1$
Fin mientras;

Una de las meta heurísticas que muestra ser efectiva para el problema de corte es TS, afirma Lodi, Martello y Vigo [57]; ellos presentan una esquema general de búsqueda tabú que resuelve el problema *bin packing* 2D; es general porque utiliza una heurística subordinada al algoritmo TS para resolver el problema BP2D.

La principal característica de este método es el uso de un parámetro *k* unido a la determinación del vecindario, cuyo tamaño y estructura son dinámicamente variados durante la búsqueda. El esquema y la vecindad son independientes del problema específico a resolverse. Las peculiaridades del problema son tomados en cuenta sólo en la selección de un especifico algoritmo a ser usado en la búsqueda de la vecindad. Se denota por *A'* tal algoritmo y por *A'(s)* el valor de la solución que este produce cuando aplicado a las (sub)instancias de BP2D determinado por ítems en *S*.

Dada una solución, la vecindad se busca a través de movimientos que consisten en modificar la solución cambiando el empaquetado de un subconjunto de ítems *S*, en un intento por vaciar una especifica "caja-objetivo". Para este fin, el subconjunto *S* siempre incluye un ítem *j*, de la caja-objetivo y el contenido actual de las otras *k* cajas. El nuevo empaquetado para *S* se obtiene ejecutando el algoritmo *A'* sobre *S*. De esta forma, el parámetro *k* define el tamaño y la estructura de la vecindad actual. Este valor es

automáticamente incrementado o disminuido durante la búsqueda y el algoritmo mantiene *k* distintas listas tabú.

La caja-objetivo se selecciona como uno de los más probables a vaciarse por el movimiento. El criterio que se sigue es: La caja-objetivo *t*, que minimiza la función de llenado $\varphi(S_i)$ sobre todas las cajas actuales.

$$\varphi(S_i) = \alpha \frac{\sum_{j \in S_i} w_j h_j}{WH} - \frac{|S_i|}{m}$$

Donde, S_i denota el conjunto de ítems actualmente empaquetados dentro de la caja *i*, y α es un pre-especificado peso positivo (Lodi y otros [57], reporta α igual a 20). Este criterio favorece la elección de una caja–objetivo con una pequeña área empaquetada, rompiéndose empates de las cajas por el menor número de ítems empaquetados.

En el algoritmo, una solución inicial conveniente se obtiene ejecutando el algoritmo *A'*, sobre la instancia completa. La solución inicial de búsqueda tabú consiste en empaquetar un ítem por caja. A cada iteración, una caja–objetivo se selecciona, y una secuencia de movimientos, realizado cada vez dentro de un procedimiento de SEARCH, intenta vaciar este. El procedimiento SEARCH también actualiza el valor del parámetro *k* y en situaciones especiales puede ejecutar acciones de diversificación. La ejecución termina tan pronto como se encuentra una solución óptima o se alcanza un tiempo límite.

Algoritmo BP2D-TABU:
z^* := *A'*({1, . . ., *m*}), (valor de la solución conveniente);
Calcule un lower bound L_0 sobre el valor óptimo;
Si $z^* = L_0$ **entonces pare**;
Inicialice todas las listas en vacío;
Empaquete cada ítem dentro de una caja diferente;
z := *m* (Valor de la solución tabú search);
Mientras tiempo limite no se alcanza **hacer**
determine la caja-objetivo *t*;
diversificado := falso, *k* := 1;
Mientras *diversificado* = falso y $z^* > L_0$
hacer
k_{im} := *k*;
Llamar SEARCH(*t*,*k*,*diversificado*,*z*);
z^* := min{z^*, *z*};
Si $k \leq k_{im}$ **entonces** determine una nueva caja-objetivo *t*
Fin Mientras;
Si $z^* = L_0$ **entonces parar**
De lo contrario ejecutar una acción de diversificación
Fin Mientras
Fin.

Dada la caja objetivo *t,* el procedimiento SEARCH explora la vecindad definida por el valor actual del parámetro *k.* Para cada ítem *j* en la caja *t* se evalúa el movimiento candidato ejecutando el algoritmo *A'* sobre las subinstancias producidas por todos los posibles conjuntos *S* definidos por el ítem *j* y por todos ítems de las otras *k*-tuples cajas.

En dos situaciones especiales el movimiento es inmediatamente ejecutado y el control retorna al programa principal: 1) Cuando un movimiento disminuye el número de cajas usadas; 2) Cuando un movimiento no-tabú remueve el ítem *j* desde la caja *t* por empaquetando las subinstancias en exactamente *k* cajas. En estos casos, la vecindad es cambiada disminuyendo el valor del parámetro *k* en una unidad. Cuando ni 1) ni 2) se consigue, una penalidad se asocia con el movimiento. La penalidad es infinita si el movimiento es tabú, o si el algoritmo *A'* usado da como resultado en al menos 2 cajas más (i.e., $A'(S) > k + 1$), o si, $k = 1$. En otro caso, la penalidad se calcula, así: se determina entre las $k + 1$ cajas producidas por *A'*, una caja $\bar{t}$, que minimiza la función de llenado, y se ejecuta el algoritmo *A'* sobre las subinstancias inducidas por los ítems en la caja $\bar{t}$ más los ítems residuales de la caja-objetivo. Si la solución es solamente una caja, la penalidad del movimiento global es el mínimo de los valores de la función de llenado calculado para las *k-1* resultantes cajas; de otro lado, la penalidad es infinita (esto es, un valor muy grande).

Cuando una vecindad ha sido completamente alcanzada sin encontrarse en los casos 1 o 2, el movimiento asociado a la mínima penalidad finita (o cualquiera) se ejecuta y el control regresa al programa principal.

Si en cambio, la penalidad mínima es infinita, esto es no se encontrado movidas aceptables, la vecindad se cambia por el incremento de valor del parámetro *k* en una unidad, o, si *k* ya alcanzó un máximo valor prefijado *kmax*, una acción de diversificación se ejecuta.

Procedimiento SEARCH (*t,k,diversificado,z*):
*penalidad** : = + ∞ ;
Para cada $j \in S_t$ **hacer**
Para cada *k*-tuple *k* de cajas no incluyendo *t* **hacer**
$S := \{j\} \cup \left(\cup_{i \in K} S_i\right)$;
penalidad := + ∞ ;
En caso de
$A'(S) < k$:
ejecute la movida y actualice el valor
de la solución actual *z*;
k := max{1, *k* - 1};

retornar;
A'(*S*) = *k*:
Si el movimiento no es tabú **o** $S_t \equiv \{j\}$ **entonces**
ejecute el movimiento y actualice el valor
de la solución actual *z*;
Si $S_t \equiv \{j\}$ **entonces** *k* := max{1, *k* - 1};
retornar
fin Si;
A'(*S*) = *k* + 1 **y** *k* > 1:
Sea *I* el conjunto de *k* + 1 cajas usadas por *A'*;
$\bar{t} := \arg\ \min_{i \in I} \{\varphi(S_i)\}$, $T := (S_t \setminus \{j\}) \cup S_{\bar{t}}$;
Si *A'*(*T*) = 1 **y** el movimiento no es tabú **entonces**
penalidad := $\min\{\varphi(T), \min_{i \in I \setminus \bar{t}} \{\varphi(S_i)\}\}$
fin En caso;
*penaltidad** := min{*penalidad**, *penalidad*};
Fin para;
Fin para;
Si *penalidad** $\neq +\infty$ **entonces** ejecute el movimiento correspondiente
a *penalidad**
de lo contrario Si *k*= *kmax* **entonces** *diversicado*: = true **en caso contrario** *k* := *k* + 1
retorne.

El algoritmo ejecuta dos clases de diversificación, controlado por un contador *d*, inicializado en uno. Cada vez que una diversificación es impuesta *d* es incrementado en uno y la caja–objetivo es determinada como aquella que tiene el *d*-ésimo valor mas pequeño de la función de llenado. Sin embargo, si $d > z$ ó $d = dmax$ (donde *dmax* es un limite prefijado), una diversificación más fuerte se ejecuta, a saber: 1) Las $\lfloor z/2 \rfloor$ cajas con valor más pequeño de la función de llenado son removidas desde la solución de la búsqueda tabú; 2) Una nueva solución se obtiene empaquetando sólo en una caja separada cada ítem actualmente empacado dentro de la caja removida; 3) Todas las listas tabú son inicializadas en vacío, y el valor de *d* es inicializado en uno.

Existe una lista tabú y una pertenencia tabú τ_k ($k = 1, \ldots, kmax$) para cada vecindad; la lista tabú almacena los valores de la función de llenado $\varphi(S)$ correspondiente a los últimos τ_1 conjuntos S para el cual una movida ha sido ejecutada.

El algoritmo termina porque la solución hallada es óptima o porque el tiempo límite se ha agotado.

Otro algoritmo basado en TS es dado por Alvarez, A. Parajón, y Jose Manuel Tamarit [1].

4.3 TEMPLADO SIMULADO (*Simulated Annealing*)

Esta meta heurística es propuesta por Kirpatrick, Gelatt y Vecchi [51] como un procedimiento para obtener soluciones aproximadas en problemas de optimización; el nombre viene de un algoritmo diseñado en los años 50 para simular el enfriamiento de materiales (un proceso denominado annealing). Kirpatrick y otros [51] trabajando en el diseño de circuitos electrónicos consideraron aplicar el algoritmo de Metrópolis en algunos de los problemas de optimización combinatoria que aparecen en este tipo de diseños. El algoritmo de Metrópolis simula el cambio energético de un sistema de partículas a medida que la temperatura decrece hasta llegar a un estado estable (congelado). Las leyes de la termodinámica establecen que, a una temperatura *t*, la probabilidad de un aumento de energía de magnitud ∂E viene dada por la expresión siguiente:

$$P[\partial E] = \exp(-\partial E / kt) \; ; \qquad (4.3)$$

Donde, *k* es la constante de Boltzmann.

En la simulación de Metrópolis se genera una perturbación aleatoria en el sistema y se calcula los cambios de energía resultante. Si hay una caída energética el cambio se acepta automáticamente, en caso contrario se acepta con una probabilidad dada en la ecuación (4.3). El proceso se repite en un número predefinido de iteraciones con temperaturas mas bajas hasta que el sistema este "frío".

Kirpatrick y otros [51], pensaron que era posible establecer una analogía entre los parámetros que intervienen en la simulación termodinámica de Metrópolis y los que aparecen en los métodos de optimización local, tal y como muestra la tabla adjunta.

Simulación Termodinámica	Optimización Combinatoria
Estados del sistema	Soluciones factibles
Energía	Costos
Cambio de estado	Solución en el entorno
Temperatura	Parámetros de control
Estado congelado	Solución heurística

Tabla 4.2 Relación entre Simulación Termodinámica vs Optimización Combinatoria

Así pues, Templado Simulado (SA) es un procedimiento basado en búsqueda local en donde los elementos son elegidos del entorno de forma aleatoria y en donde todo movimiento de mejora se acepta y se permiten movimientos de no mejora de acuerdo con unas probabilidades. Dichas probabilidades están basadas en la analogía con el proceso físico de enfriamiento y se obtienen como función de la temperatura del sistema.

SA reemplaza la energía por una función de costo *f*, la temperatura por un parámetro de control *c*, y el cambio de estado por un mecanismo de generación, esto es una simple regla para generar transición desde una configuración del problema de optimización combinatoria. El mecanismo de generación define una vecindad *N(s)* para cada configuración (solución) *s*, que consiste en todas las configuraciones que pueden ser alcanzados desde *s* en una transición.

Inicialmente, el parámetro de control tiene un valor muy alto; una secuencia de configuraciones del problema de optimización combinatoria se genera como sigue: Dada una configuración s_i, otra configuración s_j se puede obtener seleccionándose aleatoriamente un elemento desde $N(s_i)$. Sea $\partial = f(s_j)-f(s_i)$, entonces si $\partial \leq 0$, s_i es la siguiente configuración en la secuencia con probabilidad 1, en otro caso, $\partial > 0$ y s_i es la siguiente configuración en la secuencia con probabilidad $e^{-\partial / c}$. Por tanto existe una probabilidad mayor que cero de continuar con una configuración con mayor costo que la actual configuración. Este proceso continúa hasta que el equilibrio se alcanza, esto es, hasta que la distribución de probabilidad de la configuración se aproxime a la distribución de Boltzmann.

El valor del parámetro de control desciende en pequeños pasos, permitiendo al sistema aproximarse al equilibrio en cada paso, generando una secuencia de configuraciones en la forma ya descrita. El algoritmo termina para algún valor pequeño de *c*, para el cual las transiciones virtuales no son aceptadas. La configuración final de “congelado” es tomada como la solución del problema.

El diagrama siguiente muestra un esquema general del algoritmo para un problema de minimización:

Algoritmo *Templado Simulado*
Sea *f(s)* el costo de la solución *s* y sea *N(s)* su entorno
Seleccionar una solución inicial s_0
Seleccionar una temperatura inicial $c_0>0$
Seleccionar una función de reducción de la temperatura α
Dar un número de iteraciones *nrep*
Seleccionar un *criterio-de-parada*
Repetir
  Repetir
    Seleccionar aleatoriamente $s \in N(s_0)$.
    Sea $\partial = f(s) - f(s_0)$
    Si $\partial \leq 0$ **entonces** $s_0=s$
    Caso contrario generar aleatoriamente $u \in U(0,1)$
      Si $u < e^{-\partial / c}$ **entonces** $s_0=s$;
  Hasta que *cuenta-iteraciones = nrep*
  $c=\alpha(c)$
Hasta que *criterio-parada = cierto;*
La mejor solución visitada será la solución heurística dada por el algoritmo.

Para diseñar un algoritmo a partir del esquema general anterior hay que determinar los parámetros del sistema:

-La temperatura inicial se suele determinar realizando una serie de pruebas para alcanzar una determinada fracción de movimientos aceptados.

- La función de velocidad de enfriamiento α .

- El criterio de parada para finalizar la secuencia de enfriamiento (estado de congelación).

- La estimación del número de iteraciones.

La clave de una implementación de SA es el manejo de la cola o secuencia de enfriamiento: ¿Cuál es la temperatura inicial? y ¿cómo disminuye la temperatura en cada iteración?, son las dos preguntas a responder al diseñar un algoritmo. Podemos encontrar numerosos estudios en donde se comparan colas lentas (aquellas en las que la temperatura disminuye poco a poco) con colas rápidas, que vienen a ser métodos de descenso simple con pequeñas perturbaciones.

Johnson y otros [49] sugieren realizar mejoras y especializaciones aunque no se basen en la analogía física. Además, aconsejan utilizar soluciones posibles y no posibles cuando la estructura del problema lo permita, penalizando en la función de evaluación las soluciones imposibles. El objetivo es diversificar y considerar más soluciones (notar que con bajas temperaturas la solución tenderá a ser posible).

Las últimas implementaciones de SA apuntan a olvidarse de la analogía física y manejar la cola de enfriamiento como una estructura de memoria. Es decir, la probabilidad de aceptar o rechazar un movimiento de no mejora no depende de la iteración (tiempo transcurrido) sino de lo sucedido en la búsqueda. En este sentido, la probabilidad será función de algunas variables de estado del proceso. En la actualidad se está diseñando numerosos algoritmos híbridos en donde la búsqueda local se realiza con un procedimiento basado en SA, en ocasiones combinado con búsqueda Tabú.

Un algoritmo de optimización general, produce una solución óptima global; Templado Simulado es asintóticamente un algoritmo de optimización general (Loris Faina [23]), a pesar de esto cualquier implementación de Templado Simulado resulta en un algoritmo de aproximación. La ventaja teórica de utilizar Templado Simulado deriva del hecho que es mejor utilizar un algoritmo aproximativo derivado de un algoritmo de optimización, al uso de algoritmos puramente heurísticos.

Templado Simulado ha sido aplicado e implementado para el problema de corte 2D por guillotina y no guillotina por Loris Faina [23], en primer lugar define un mecanismo de generación para conseguir un esquema de corte que no depende de la ordenación de los ítems y luego es definido un mecanismo de transición que consiste en cambiar un poco el ordenamiento del esquema invirtiendo los ítems entre dos posiciones aleatoriamente escogidas, de esta forma son generados nuevos esquemas de corte.

El mecanismo de generación consiste en construir un patrón de corte como sigue: cada ítem es localizado dentro de un objeto apropiado, uno por uno siguiendo un orden, que podría ser como los ítems aparecen en el pedido (esto no es importante, pues SA, no depende de la configuración inicial). El primer ítem es localizado dentro del primer objeto con la esquina izquierda inferior en el origen del objeto (el origen del objeto es la esquina izquierda inferior); el segundo ítem es colocado adyacente al primer ítem (esto es con la esquina izquierda inferior del segundo ítem en la esquina derecha inferior o en la esquina izquierda superior del primer objeto; la elección de dónde colocar el ítem es aleatoria), si hay espacio suficiente para este; de lo contrario los otros objetos son examinados en orden por sus etiquetas (número que identifica al objeto ó lámina; obviamente, para describir un patrón de corte genérico, a lo más es necesario un número de objetos igual al número de ítems).

Luego que un patrón de corte inicial está disponible, el mecanismo de transición consiste simplemente en cambiar el orden de los ítems, invirtiendo el orden de dos ítems aleatoriamente seleccionados y construyendo un nuevo patrón de corte como se indica arriba. Faina [23], muestra que el algoritmo asintóticamente converge, esto es alcanza el óptimo.

Algoritmo *Templado Simulado No-Guillotina*
Sea *f(s)* el costo de la solución *s* y sea *N(s)* su entorno
Genere una solución inicial s_0 siguiendo una ordenación de los ítems
Seleccionar una temperatura inicial $c_0>0$
Seleccionar una función de reducción de la temperatura α
Dar un número de iteraciones *nrep*
Seleccionar un *criterio-de-parada*
Repetir
Repetir
Seleccionar aleatoriamente dos ítems e intercambiarlos de posición en la lista de ítems.
Para cada ítem de la lista
Sí hay espacio suficiente para encajar en la lámina actual
Entonces encaje el ítem dentro de la lámina
Caso contrario coloque el ítem dentro de la siguiente lámina según las etiquetas;
Fin para
Sea *s* la nueva solución,
Sea $\partial = f(s) - f(s_0)$
Si $\partial \leq 0$ **entonces** $s_0=s$
Caso contrario generar aleatoriamente $u \in U(0,1)$
Si $u < e^{-\partial / c}$ **entonces** $s_0=s$;
Hasta que *cuenta-iteraciones = nrep*
$c=\alpha(c)$
Hasta que *criterio-parada = cierto;*
La mejor solución hallada será la solución heurística dada por el algoritmo.

Otra variación del algoritmo SA para corte guillotina, presenta Faina [23], el cual difiere solamente en la generación de los patrones de tipo guillotina, y consiste en que la colocación de un ítem genera dos nuevos rectángulos (sub-láminas) como visto en capítulos anteriores, la selección del corte horizontal o vertical es aleatorio, y las nuevas láminas pasan a lista de láminas (ahora con diferentes tamaños, y con actualización de las etiquetas). Es necesario guardar la numeración de las sub-láminas que pertenecen a una misma lámina.

Otros, algoritmos basados en SA son propuestos por V. Parada y otros [74], Dowsland [16], Dowsland y Diaz [18] y Oliveira y Ferreira [70].

4.4. **ALGORITMOS GENETICOS**

Los Algoritmos Genéticos (AG) fueron introducidos por John Holland [47] en 1970 inspirándose en el proceso observado en la evolución natural de los seres vivos, posteriormente fue popularizado por David Goldberg [41]. Los AG siguen el principio de selección natural y sobrevivencia del más apto, declarado por el naturalista y fisiólogo ingles Charles Darwing, quien declara:

> "*Cuanto mejor un individuo se adapta a su medio ambiente, mayor será su chance de sobrevivir y generar descendientes"*.

De manera muy general podemos decir que en la evolución de los seres vivos el problema al que cada individuo se enfrenta cada día es la supervivencia. Para ello cuenta con las habilidades innatas provistas en su material genético. A nivel de los genes, el problema es el de buscar aquellas adaptaciones beneficiosas en un medio hostil y cambiante. Debido en parte a la selección natural, cada especie gana una cierta cantidad de "conocimiento", el cual es incorporado a la información de sus cromosomas. Así pues, la evolución tiene lugar en los cromosomas, en donde está codificada la información del ser vivo. La información almacenada en el cromosoma varía de unas generaciones a otras. En el proceso de formación de un nuevo individuo, se combina la información cromosómica de los progenitores aunque la forma exacta en que se realiza es aún desconocida.

Aunque muchos aspectos están todavía por discernir, existen unos principios generales de la evolución biológica ampliamente aceptados por la comunidad científica. Algunos de éstos son:

-La evolución opera en los cromosomas en lugar de en los individuos a los que representan.

- La selección natural es el proceso por el que los cromosomas con "buenas estructuras" se reproducen más a menudo que los demás.

- En el proceso de reproducción tiene lugar la evolución mediante la combinación de los cromosomas de los progenitores. Llamamos recombinación a este proceso en el que se forma el cromosoma del descendiente.

- También son de tener en cuenta las mutaciones que pueden alterar dichos códigos.

- La evolución biológica no tiene memoria en el sentido de que en la formación de los cromosomas únicamente se considera la información del periodo anterior.

Los algoritmos genéticos establecen una analogía entre el conjunto de soluciones de un problema y el conjunto de individuos de una población natural, codificando la información de cada solución en un string (vector binario) a modo de cromosoma. Según Holland [47]:

"Se pueden encontrar soluciones aproximadas a problemas de gran complejidad computacional mediante un proceso de evolución simulada".

Para este efecto se introduce una función de evaluación de los cromosomas, "fitness" (calidad) y que está basada en la función objetivo del problema. Igualmente se introduce un mecanismo de selección de manera que los cromosomas con mejor evaluación sean escogidos para "reproducirse" más a menudo que los que tienen peor evaluación.

Los algoritmos desarrollados por Holland [48] inicialmente eran sencillos y dieron buenos resultados en problemas considerados difíciles.

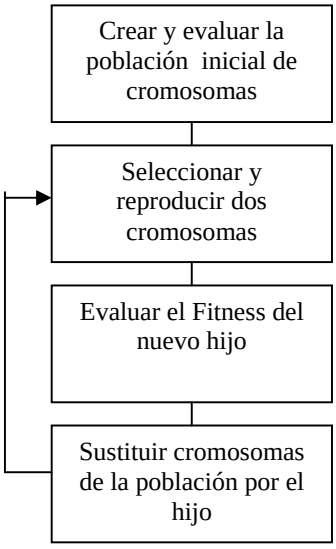


Fig. 4.1 Esquema de un algoritmo genético

Los AG están basados en integrar e implementar eficientemente dos ideas fundamentales: Las representaciones simples como strings (binario, enteros, cadenas de números y/o caracteres) de las soluciones del problema y la realización de transformaciones simples para modificar y mejorar estas representaciones.

Para tal fin se debe especificar los siguientes elementos:

Una representación cromosómica: En los trabajos originales las soluciones se representaban por strings binarios, es decir, listas de 1s y 0s. Este tipo de representaciones ha sido ampliamente utilizado incluso en problemas en donde no es muy natural. En 1985, De Jong [15] introduce la siguiente inquietud: ¿Qué se debe hacer cuando los elementos del espacio de búsqueda se representan de modo natural por estructuras complejas como vectores, árboles o grafos?, ¿Se debe intentar linealizar en un string o trabajar directamente con estas estructuras?. En la actualidad podemos distinguir dos escuelas, la primera se limita a strings binarios, mientras que la segunda utiliza todo tipo de configuraciones. Hemos de notar que las operaciones genéticas dependen del tipo de representación, por lo que la elección de una condiciona la otra.

La ventaja de la primera es que permite definir fácilmente operaciones de recombinación, además los resultados sobre convergencia están probados para el caso de strings binarios. Sin embargo, en algunos problemas puede ser poco natural y eficiente el utilizarlas. Por ejemplo en el problema del agente viajero sobre 5 ciudades y 20 aristas, el string 01000100001000100010 representa una solución sobre las aristas ordenadas. Sin embargo, dicha representación no es muy natural y además, no todos los strings con cinco 1s representan soluciones lo cual complica substancialmente la definición de una operación de sobrecruzamiento. Es más natural la ruta de ciudades: (2,3,1,5,4), lo cual permite definir naturalmente diferentes operaciones estables.

La población inicial: Suele ser generada aleatoriamente. Sin embargo, últimamente se están utilizando métodos heurísticos para generar soluciones iniciales de buena calidad. En este caso, es importante garantizar la diversidad estructural de estas soluciones para tener una representación de la mayor parte de población posible o al menos evitar la convergencia prematura.

Una medida de evaluación de los cromosomas: Se utiliza como medida de calidad (fitness); ésta mide la bondad del cromosoma (individuo, solución) según el valor de la función objetivo en el que se puede añadir un factor de penalización para controlar la infactibilidad. Este factor puede ser estático o ajustarse dinámicamente, lo cual produciría un efecto similar al de la oscilación estratégica en Tabu Search.

*calidad = Valor Objetivo Normalizado - Penalización * Medida Infactibilidad*

Un criterio de selección: La eliminación de los cromosomas padres viene dado habitualmente mediante probabilidades según su *fitness*. Uno de los procedimientos más utilizado es el denominado de la ruleta (clásico método de Montecarlo) en donde cada individuo tiene una sección circular de una ruleta que es directamente proporcional a su calidad. Para realizar una selección se realiza una tirada de bola en la ruleta, tomando al individuo asociado a la casilla donde cayó la bola.

Operadores de Cruzamiento: los más utilizados son:

De un punto: Se elige aleatoriamente un punto de ruptura en los padres y se intercambian sus bits.

De dos puntos: Se eligen dos puntos de ruptura al azar para intercambiar.

Uniforme: En cada bit se elige al azar un padre para que contribuya con su bit al del hijo, mientras que el segundo hijo recibe el bit del otro padre.

PMX, SEX: Son operadores más sofisticados fruto de mezclar y aleatorizar los anteriores.

Operación de Mutación: Es la más sencilla, y una de las más utilizadas consiste en reemplazar con cierta probabilidad el valor de un bit. Note que el papel que juega la mutación es el de introducir un factor de diversificación ya que, en ocasiones, la convergencia del procedimiento a buenas soluciones puede ser prematura y quedarse atrapado en óptimos locales. Otra forma obvia de introducir nuevos elementos en una población es recombinar elementos tomados al azar sin considerar su fitness.

A continuación mostramos un algoritmo genético híbrido para el problema de corte 2D, no guillotinable, presentado por Aparecido Constantino y Gomes Junior [2].

Algoritmo genético

Sea S(t) la población de cromosomas en la generación t
t → 0
inicializar S(t)
evaluar S(t)
En cuanto el criterio de parada no es satisfecho **hacer**
  t → t+1
  seleccionar S(t) a partir de S(t-1)
  aplicar *crossover* sobre S(t)
  aplicar mutación sobre S(t)
  evaluar S(t) //aplicación de *Bottom-Left*
Fin En cuanto

La estructura del algoritmo sigue el patrón bastante conocido, pero la evaluación se ejecuta por un algoritmo de encaje *Bottom-left*, por este motivo el algoritmo es denominado de híbrido. El algoritmo *Bottom-left*, consiste en encajar la primera pieza en la esquina inferior, y las demás son encajadas siempre que sea posible en la parte inferior con la siguiente prioridad: primero se intenta encajar la pieza lo más próximo de la base y después es movida a la izquierda lo más posible, sin sobreponerse a ninguna otra pieza. Después que todas las piezas han sido encajadas es posible determinar el desperdicio.

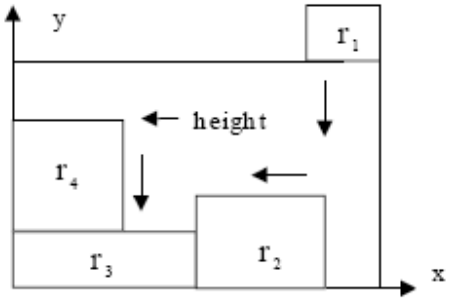


Fig. 4.2. Ilustración de Bottom-left

Para el problema de corte 2D la **representación de los cromosomas** que da Aparecido [2], es un cromosoma representado por un vector que contiene el orden en que las piezas serán encajadas; un ejemplo se da en la siguiente figura;

3	7	2	5	8	1	6	4

Fig. 4.3 Representación del cromosoma para el problema de corte

Cada número del vector corresponde al número de la pieza que será encajada. Las informaciones de cada pieza, como son ancho y altura, se guardan también en una lista.

La selección de los cromosomas para el cruzamiento se realiza sobre un porcentaje de los mejores cromosomas. En este caso se utilizó una variación de 30% a 60% una técnica conocida como Elitismo (Lacerda e Carvalho [53]).

En este problema de corte, el **operador *crossover*** genera un número aleatorio entre 1 y la mitad del primer cromosoma padre C1. Este número generado es llamado de "punto de corte". Este será el índice que indicará en qué punto del cromosoma C1 comenzará el cruzamiento. A partir de este punto de corte, se toma la mitad del cromosoma C1 y se

coloca al inicio del cromosoma hijo F1 que será generado. Después se recorre el segundo cromosoma padre C2, verificando si la pieza ya esta en el cromosoma F1 o no. Si no está, esta se coloca en la primera posición vacía del cromosoma F1. Si la pieza ya está se verifica la próxima pieza del cromosoma C2 si está o no está en el cromosoma F1. El proceso continúa hasta que el cromosoma F1 este completo. La figura 4.4 ilustra la aplicación de este tipo de *crossover*; donde los números 5, 8, 1, 2 del cromosoma C1 y 7, 6, 4 y 3 del cromosoma C2 fueron utilizados para la generación del cromosoma F1.

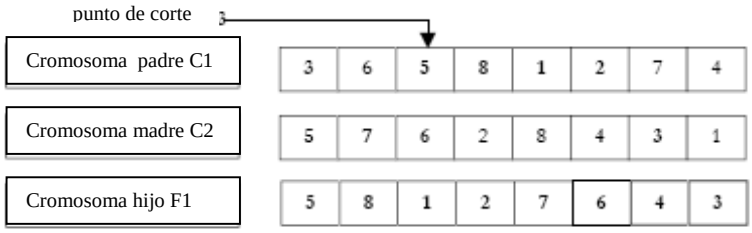


Fig. 4.4 Ilustración de operador crossover

El operador **Mutación** que funciona invirtiendo el valor de algún gen del cromosoma, en este problema de corte, invierte las dimensiones de una pieza, esto es, invierte la altura y el ancho de una pieza. La mutación mejora la diversidad de los cromosomas de una población, pero por otro lado, destruye la información contenida en el cromosoma. Por este motivo, debe ser usada en pequeña proporción, generalmente entre 0,5% y 5%.

La evaluación es efectuada a través de una técnica de encaje de las piezas en la lámina rectangular (*Bottom-Left*), así es posible cuantificar el desperdicio generado por un cromosoma dado. Ese valor es utilizado para definir la aptitud del cromosoma.

Aparecido y otros [2], reportan éxito en la implementación de su algoritmo genético.

Otra implementación del algoritmo genético para este problema es dado por Kroger [52].

CAPITULO V

CONCLUSIONES Y PERSPECTIVAS FUTURAS

5.1 Conclusiones

1. En este estudio hemos visto el problema de corte y empaquetado 2D, sus diferentes variantes respecto a los otros problemas de optimización combinatoria; aprendido a reconocer algunos problemas clásicos de la optimización combinatoria como el problema de la mochila, el problema de carga de vehículos, de carga en pallets, programación de tareas, etc., como perteneciente a esta gran familia de problemas algunos con unas variantes (restricciones adicionales) pero intrínsicamente similares.

2. También hemos revisto algunos métodos de resolución del problema de corte y empaquetado en general clasificándolo principalmente por los métodos que resuelven los problemas de optimización combinatoria como son los métodos exactos, heurísticas y las meta heurísticas; y secundariamente por la clasificación más común que se encuentra en la literatura para el problema de corte 2D, los métodos por guillotina y los métodos no guillotina.

3. De los métodos exactos el método de generación de columnas es uno de los clásicos para resolver el problema de corte y empaquetado 2D; sin embargo el más estudiado por su versatilidad para construir patrones guillotinables es el método de Wang [78] y programación dinámica. Otro enfoque nuevo para resolver el problema en forma exacta es utilizando árboles *And/Or* y conceptos de inteligencia artificial para resolver problemas de optimización como el algoritmo AAO* (Aditive AND/OR) presentado por V. Parada y otros [73].

4. La gran mayoría de las heurísticas para el problema de C&E 2D son adaptaciones de las heurísticas FFD y BFD que resuelven el problema de C&E de una dimensión. Estas heurísticas en general trabajan en dos fases, primero empaqueta los ítems en una franja

(*Strip*) considerando solamente una de sus dimensiones y luego corta esta al tamaño de las láminas (cajas) deseadas; esto es, resuelven el problema de corte de dimensión “1+1”. Otras, heurísticas presentadas en este trabajo consideran otros aspectos como: forma de colocar los ítems en las láminas (cajas), como es el de direcciones alternantes; otros, que ordenan los ítems según criterio goloso por el tamaño (área) de las piezas o ítems y empaquetan en forma recursiva, esto es no consideran 2 fases para conseguir patrones, como es el FFD-CUT-$2D_{RG}$ y BFD-CUT-$2D_{RG}$ de Mauricio y Delgadillo [64]. Estas heurísticas intentan mezclar diferentes conceptos para construir mejoras en cuanto al objetivo principal optimizar la pérdida teniendo en cuenta que el tiempo de respuesta es importante.

5. Las meta heurísticas aquí, vistas son conceptualmente sencillas, la simplicidad de la implementación depende en algunos casos de la heurística subordinada que manejen. En el caso del problema de C&E 2D, el problema en sí se dificulta por el hecho de la construcción del patrón de corte. Sin embargo, el problema es posible que sea resuelto por cualquier meta heurística.

6. Los métodos no-guillotinables obtienen mejores resultados respecto a la pérdida residual (merma), pero presentan mayor dificultad en la construcción de los patrones.

7. Todos los métodos presentados se refieren a cortes regulares (rectangulares), existe poca literatura para cortes no regulares, esto es de formas no simétricas, no convexas, aplicables a la industria textil, de cuero y otros. La complejidad del problema de corte y empaquetado 2D asociado a este tipo de corte indudablemente exhiben una mayor complejidad y por esto a la fecha hay pocos estudios referentes al tema.

5.2 Perspectivas futuras

Es factible hacer implementaciones computacionales de las meta heurísticas mostradas en esta investigación, utilizando heurísticas subordinadas que también se muestran en este estudio.

La inclusión de métodos y conceptos de la inteligencia artificial para resolver el problema de corte, ya se ha iniciado, Tabú Search, y algoritmo AAO* son una muestra de ello; pero existen otras metodologías que aún no se han visualizado como son los algoritmos evolutivos, redes neuronales.

Dentro de los métodos de la investigación operativa, los métodos exactos (*lasted fashion*) *Branch and cut* y *Branch and price*; pueden ser estudiados para resolver el problema de C&E 2D.

Estudios referentes a cortes no regulares se hace necesario para resolver los problemas del sector productivo para los cuales los algoritmos de compactación y de computación gráfica deben ayudar a resolver la optimización de la merma. Proyectos de investigación referentes al problema de corte asistido por computador (automatización) se iniciaron hace varios años en España y Brasil. En la actualidad existe software que apoya el cortado de telas en la industria textil de considerable valor de compra, por lo que las pequeñas pymes no pueden acceder a estos beneficios.

BIBLIOGRAFÍA

[1] Alvarez-Valdés Ramón, Parajón Antonio, y Tamarit José Manuel, (2002) "A Tabu Search Algorithm for Large-scale Guillotine (Un)Constrained Two-dimensional Cutting Problems", *Computers & Operations Research* 29, 925-947.

[2] Aparecido Constantino Ademir e Mendes Gomes Junior Augusto, (2002) "Um Algoritmo Genético Híbrido para o Problema de corte Industrial Bidimensional", *Acta Scientiarum Maringá* v.24, n.6, p. 1727-1731.

[3] Beasley J.E., (1985) "Algorithms for Unconstrained Two-dimensional Guillotine Cutting", *Journal of the Operation Research Society,* 36/4, 297-306.

[4] Beltrán Cano, Calderón J.E., Cabrera R.J. y Moreno Vega, (2001) " Procedimientos Constructivos Adaptativos (GRASP) para el Problema de empaquetado Bidimensional", CEPEIA Departamento de Estadística, Investigación Operativa y Computación, Universidad de la Laguna.

[5] Berkey J.O. and Wang P.Y., (1987) "Two Dimensional Finite Bin parking Algorithms", *Journal of the Operation Research Society* 38,423-429.

[6] Campello R., y Maculan N., (1994) Algoritmos e Heurísticas, Desenvolvimento e Avaliaçao da Perfomance, Editora da Universidade Federal Fulmínense, Niteroi-RJ.

[7] Caprara A., Toth P. (1998) "Lower Bounds and Algorithms for the 2-constrained Bin Packing Problem", Tech. Report, DEIS OR-98-7, University of Bologna.

[8] Carazo S. M. y Hurtado E. A. (1998) "La industria del Cuero y Calzado en el Perú: Innovando para Competir, serie Cadenas Productivas", MITINCI, Lima-Perú.

[9] Coffman E.G., Garey M.R. and Jhonson D.S., (1978) "An Application of Bin-Packing to Multiprocessor Scheduling", *SIAM Computing* 7, 1-17.

[10] Coffman E.G., Garey M.R., Jhonson D.S. and Tarjan R.E. (1980), "Performance Bounds for Level-oriented Two-dimensional Packing Algorithms", *SIAM Journal on Computing* 9,801-826.

[11] Christofides N. and Whitlock C. (1977), “An Algorithm for Two-dimensional Cutting Problems”, *Operations Research* 33/1, 49-64.

[12] Chung F.K.R., Garey M.R., Johnson D.S., (1982) “On Packing Two-dimensional Bins”, *SIAM Journal on Algebraic and Discrete Methods* 3(1) 66-76.

[13] Chvátal Vasek, (1983) Linear Programming, W.H. Freeman and Company, New York/San Francisco.

[14] Dantzig G.B., (1957) “Discrete-variable Extremum Problems”, *Operational Research* 5, 266-277.

[15] De Jong, K.A. (1985) “Genetics Algorithms: A 10 Years Perspective” *Proceedings of an International Conference on Genetic Algorithms and their Applications,*169-177.

[16] Dowsland K.A., (1990) “Some Experiments with Simulated Annealing techniques for packing problems”, Technical report, University de Wales, Swansea UK.

[17] Dowsland K.A., Dowsland W.B., (1992) “Packing problems”, *European Journal of Operation Research.* 56 (1) 2-14.

[18] Dowsland Kathryn A. y Diáz Berlamino Adencio,(2003) “Diseño de Heurísticas y Fundamentos del Recocido Simulado”, *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, Nº 19, 93-102.

[19] Dyckhoff H., (1990) “A Typology of Cutting and Packing Problems”, *European Journal of Operation Research* 44, 145-159.

[20] Dyson R.G., Gregory A.S., (1976) “The Cutting Stock Problem in the Flat Glass Industry”, *Operational Research Quartely* 25, 41-53.

[21] Eilon E., (1960) “Optimising the Shearing of Steel Bars”, *Journal of Mechanical Engineering Science* 2, 129-142.

[22] Eilon S. and Christofides N., (1971) “The Loading Problem”, *Management Science* 17, 259-268.

[23] Faina Loris, (1999) “An Application of Simulated Annealing to the Cutting Stock Problem”, *European Journal of Operation Research* 114, 542-556.

[24] Farley A., (1983) “Trim Loss Pattern Rearrangement and it’s Relevance to the Flat-glass Industry”, *European Journal of Operation Research* 14, 386-392.

[25] Farley A., (1988) “Mathematical Programming Model for Cutting Stock Problems in the Clothing Industry”, *Journal of the Operation Research Society* 39, 41-53.

[26] Farley A., (1990) “The Cutting Stock Problem in the Canvas Industry”, *European Journal of Operation Research* 44, 247-255.

[27] Feo T. and Resende M.G.C., (1989) “A Probabilistic Heuristic for a Computational Difficult Set Covering Problems”, *Operations Research Letters* 8, 67-71.

[28] Feo T. and Resende M.G.C., (1995) “Greedy Randomized Adaptive Search Procedures”, *Journal of Global Optimization* 2, 1-27.

[29] Festa P. and Resende M.G.C., (2001) “GRASP: An Annotated Bibliography” *AT&T Labs Research Tech. Report.*

[30] Garey, M.R., and Jhonson, D.S., (1979) Computers and Intractability: Guide to the Theory of NP-completeness, Freeman, San Francisco CA.

[31] George, J., and Lamar, B., (1995) “Packing Different Sized Circles into a Rectangular Container”, *European Journal of Operational Research* 84, 693-712.

[32] Gilmore P., and Gomory R., (1961) “A linear Programming Approach to the Cutting–Stock Problem (part 1)”, *Operations Research* 9, 849-859.

[33] Gilmore P., and Gomory R., (1963) “A linear Programming Approach to the Cutting–Stock Problem (part 2)”, *Operations Research* 11, 863-888.

[34] Gilmore P., and Gomory R., (1965) “Multistage Cutting Problems of Two and More Dimensions”, *Operation Research* 13, 94-119.

[35] Gilmore P., (1979) “Cutting Stock Linear Programming, Knapsacking, Dynamic Programming an Integer Linear Programming; Some Interconnections”, *Annals of Discrete Mathematics* 4, 217-235.

[36] Glover F., (1977) “Heuristics for Integer Programming Using Surrogate Constraints”, *Decision Science* 8, 156-166.

[37] Glover F., (1986) “Future Paths for Integer Programming and Links to Artificial Intelligence”, *Computers and Operations Research* 5, 553-549.

[38] Glover F., (1989) “Tabu Search: Part I”, *ORSA Journal on Computing* 1, 190-206.

[39] Glover F. and Laguna M., (1997) Tabu Search, Ed. Kluwer, London.

[40] Golden B.L., (1976) “Approaches to the Cutting Stock Problem”, *AIIE Transactions* 6, 265-274.

[41] Goldberg David E., (1989) Genetics Algorithms in Search, Optimization and Machine Learning, ADDison Wesley Longman, USA.

[42] Haessler R.W. and Talbot F.B., (1983) “A 0-1 Model for Solving the Corrugator Trim Problem”, *Management Science* 29, 200-209.

[43] Herz J.C., (1972) “A Recursive Computing Procedure for Two Dimensional Stock Cutting”, *IBM Journal of Research and Development* 16, 462-469.

[44] Hifi Mhand, V. Zissimopoulos, (1996) “A Recursive Exact Algorithm for Weighted Two-dimensional Cutting”, *European Journal of Operational Research* 91, 553-564.

[45] Hifi Mhand, (1998) “Exact Algorithms for the Guillotine Strip Cutting/Packing Problem”, *Computers Ops Res*. Vol 25, Nº11, 925-940.

[46] Hinxman A.I., (1980) “The trim-loss and assortment problems: a survey”, *European journal of Operation Research* 5, 8-18.

[47] Holland Jhon, (1973) “Genetics Algorithms and the Optimal Allocations of Trials”, *SIAM Journal of Computing* 2(2), 88-105.

[48] Holland J.H., (1992) “Genetics Algorithms”, *Scientific American* 267, 66.

[49] Johnson D.S., Aragon C.R., McGeoch L.A. and Schevon C., (1989) “Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning”, *Operations Research* 37, 24-35.

[50] Kantorovich, L.V. (1939, 1960) “Mathematical Methods of Organizing and Planning Production”, *Management Science* 6, 366-422.

[51] Kirkpatrick S., Gelatt C.D. and Vecchi P.M., (1983) “optimization by Simulated Annealing”, *Science* 220, 671-680.

[52] Kroger, B., (1995) “Guillotineable Bin Packing: A Genetic Approach”, *European Journal of Operational Research* 84, 645-661.

[53] Lacerda, E.G.M. y Carvalho, A:C.P.L.F., (1999) “Introduçao aos Algoritmos Genéticos”. *In: Congesso Nacional da Sociedade Brasileira de Computaçao, Anais* Rio de Janeiro, v.2,p.51-126.

[54] Li, Zhenyu, y Milenkovic, V., (1995) “Compaction and Separation Algorithms for Non-convex Polygons and their Applications”, *European Journal of Operations Research* 84, 539-561.

[55] Liton C.D., (1977) “A frequency approach to the one dimensional cutting problem for carpet rolls”, *Operation Research Quarterly* 28, 927-938.

[56] Lodi A., Martello S., Vigo D., (1999) “Approximation Algorithms for the Oriented Two-dimensional Bin Packing Problem”, *European Journal of Operation Research* 112 158-166.

[57] Lodi Andrea, Martello Silvano, and Vigo Daniele, (1999) “Heuristic and Metaheuristic Approaches for a Class of Two-dimensional Bin Packing Problem”, *INFORMS Journal on Computing* vol. 11, Nº 4, 345-357.

[58] Lorie, J. and Savage L.J., (1955) “Three Problems in Capital Rationing”, *Journal of Business* 28, 229-239.

[59] Madsen O.G.B., (1979) “Glass Cutting in Small Firm”, *Mathematical Programming* 17, 85-90.

[60] Martello S. and Toth P., (1980) “Optimal and Canonical Solutions of the Change Making Problem”, *European Journal of Operational Research* 4, 322-329.

[61] Martello S. and Toth P., (1990) “Lower Bound and Reduction Procedures for the Bin Packing Problem”, *Discrete Applied Mathematics* 28, 59-70.

[62] Martello S. and Vigo D., (1998) “Exact Solution of the Two-dimensional Finite Bin Packing Problem”, *Management Science* 44, 388-399.

[63] Martí Rafael “Metaheurísticos en optimización Combinatoria”, Departamento de Estadística e Investigación Operativa, Universidad de Valencia. E-mail: Rafael.marti@uv.es.

[64] Mauricio D. y Delgadillo R., (2002) “Algoritmos FFD y BFD para el Problema de cortes de Guillotina en 2D” Reporte técnico FISI, UNMSM.

[65] Mauricio D., (2004) “Algoritmos GRASP para el problema de cortes de Guillotina”, Reporte técnico FISI, UNMSM.

[66] Melián Belén, Moreno Pérez José A. y Moreno Vega J. Marcos, (2003) “Metaheuristicas una visión global”, *Revista Iberoamericana de Inteligencia Artificial*, N° 19, 7-28.

[67] Metzger R., (1958) Stock Slitting, Elementary Mathematical Programming, Capitulo 8, Wiley, New York.

[68] Morabito R., Garcia V., (1997) "The cutting stock problem in hardboard industry: a case study", *Computer Operation Research* 25/ 6, 469-485.

[69] Oliveira J.F. and Ferreira J.S., (1990) "An Improved Version of Wang's Algorithm for Two-dimensional Cutting Problems", *European Journal of Operation Research* 44, 256-266.

[70] Oliveira J.F. and Ferreira J.S., (1992) "An Applications of Simulated Annealing to the Nesting Problems" paper present at the 34th *ORSA/TIMS Joint National Meeting*, San Francisco CA.

[71] Osman, I.H. and Kelly J.P., (1996) Meta-Heuristics: Theory and Applications, Ed. Kluwer Academics, Boston.

[72] Paull, A. and Walter J., (1954) "The Trim Problem: An Application of Linear Programming to the Manufacture of Newsprint Paper", *Proceedings of Annual Econometric Meeting*, Montreal, Sept. 10th-13th.

[73] Parada V., Gómes A., De Diego J., (1995) "Exact Solutions for Constrained Two-dimensional Cutting Stock Problems", *European Journal of Operation Research* 84, 633-644.

[74] Parada, V., Sepúlveda M., Solar, M. y Gómes A., (1999) "Solution for the Constrained Guillotine Cutting Problem by Simulated Annealing", *Computers operational Research* 25/1, 37- 44.

[75] Spieksma F.C., (1994) "A Branch-and-Bound Algorithm for the Two-dimensional Vector Packing Problem", *Computers and Operations Research* 21, 19-25.

[76] Toth Paolo, (2000) “Optimization Engineering Techniques for the Exact Solution of NP-Hard Combinatorial Optimization Problems”, *European Journal of Operational Research* 125, 222-238.

[77] Venkateswarlu P., Martyn C.W., (1992) “The trim loss problem in a wooden drums industry”, *OR-92 Proceeding of the Convention of Operation Research Society of India*,

[78] Wang, P.Y. (1983) “Two Algorithms for Constraint Two Dimensional Cutting Stock Problems”, *Operations Research*, 31, 573-586.

[79] Wee, T.S. and Magazine, M.J., (1982) “Assembly Line Balancing as Generalized Bin-Packing”, *Operational Research Letters* 1, 56-58.

[80] Westernlund T., Isaksoon J., Harjunkoski I., (1995) “Solving a production optimization problem in the paper industry”, Report 95-146A, Process Desing Laboratory, Abo Akademi University.