Curso 2011/12
CIENCIAS Y TECNOLOGÍAS/25
I.S.B.N.: 978-84-15910-22-0

JÉSICA DE ARMAS ADRIÁN

Problemas de corte: métodos exactos y aproximados para formulaciones mono y multi-objetivo

Directoras
COROMOTO LEÓN HERNÁNDEZ
GARA MIRANDA VALLADARES

SERVICIO DE PUBLICACIONES
UNIVERSIDAD DE LA LAGUNA

SOPORTES AUDIOVISUALES E INFORMÁTICOS
Serie Tesis Doctorales

Agradecimientos

Quiero expresar mi agradecimiento a mi directora de Tesis, Coro, por brindarme la oportunidad de trabajar en el campo que abarca la misma y permitirme recurrir a su capacidad y experiencia científica, siempre dentro de un marco de confianza. Tambien me gustaría agradecer a mi codirectora de Tesis, Gara, por hacerme más ameno el día a día y por su permanente disposicion para ayudar y resolver tantas cuestiones que se han ido planteando durante el desarrollo del trabajo. Asimismo, agradezco a mis compañeros Yanira, Carlos y Eduardo, por ayudarme cuando lo he necesitado.

Especial mención y agradecimiento a mis padres, por enseñarme que la perseverancia y el esfuerzo son el camino para lograr objetivos, y a Alberto por su cariño, paciencia, comprensión y constante estímulo. Muchas gracias.

Además, estoy agradecida al Ministerio Español de Ciencia y Tecnología, por mi beca FPU-AP2007-02414, que me ha permitido llevar a cabo la investigación. Este trabajo ha sido respaldado por el EC (FEDER) y el Ministerio Español de Ciencia y Tecnología como parte del 'Plan Nacional de I+D+i', con números de contrato: TIN2005-08818-C04-04, TIN2008-06491-C04-02 y TIN2011-25448. El Gobierno de Canarias ha financiado parcialmente este trabajo a través del proyecto de investigación PI2007/015. Gracias también al proyecto HPC-EUROPA2 número 228398, respaldado por la "European Community Research Infrastructure Action" bajo el FP7. Muchas gracias a la gente de Centro de Computación Paralela de Edimburgo por su amabilidad. Por último, agradecimientos a la Universidad de La Laguna por el uso de los sistemas computacionales del SAII (Servicio de Apoyo Informático a la Investigación).

Jésica de Armas Adrián

Prefacio

Los problemas de corte y empaquetado son una familia de problemas de optimización combinatoria que han sido ampliamente estudiados en numerosas áreas de la industria y la investigación, debido a su relevancia en una enorme variedad de aplicaciones reales. Son problemas que surgen en muchas industrias de producción donde se debe realizar la subdivisión de un material o espacio disponible en partes más pequeñas.

Existe una gran variedad de métodos para resolver este tipo de problemas de optimización. A la hora de proponer un método de resolución para un problema de optimización, es recomendable tener en cuenta el enfoque y las necesidades que se tienen en relación al problema y su solución. Las aproximaciones exactas encuentran la solución óptima, pero sólo es viable aplicarlas a instancias del problema muy pequeñas. Las heurísticas manejan conocimiento específico del problema para obtener soluciones de alta calidad sin necesitar un excesivo esfuerzo computacional. Por otra parte, las metaheurísticas van un paso más allá, ya que son capaces de resolver una clase muy general de problemas computacionales. Finalmente, las hiperheurísticas tratan de automatizar, normalmente incorporando técnicas de aprendizaje, el proceso de selección, combinación, generación o adaptación de heurísticas más simples para resolver eficientemente problemas de optimización.

Para obtener lo mejor de estos métodos se requiere conocer, además del tipo de optimización (mono o multi-objetivo) y el tamaño del problema, los medios computacionales de los que se dispone, puesto que el uso de máquinas e implementaciones paralelas puede reducir considerablemente los tiempos para obtener una solución. En las aplicaciones reales de los problemas de corte y empaquetado en la industria, la diferencia entre usar una solución obtenida rápidamente y usar propuestas más sofisticadas para encontrar la solución óptima puede determinar la supervivencia de la empresa. Sin embargo, el desarrollo de propuestas más sofisticadas y efectivas normalmente involucra un gran esfuerzo computacional, que en las aplicaciones reales puede provocar una reducción de la velocidad del proceso de producción. Por lo tanto, el diseño de propuestas efectivas y, al mismo tiempo, eficientes es fundamental.

Por esta razón, el principal objetivo de este trabajo consiste en el diseño e implementación de métodos efectivos y eficientes para resolver distintos problemas de corte y empaquetado. Además, si estos métodos se definen como esquemas lo más generales posible, se podrán aplicar a diferentes problemas de corte y empaquetado sin realizar demasiados cambios para adaptarlos a cada uno. Así, teniendo en cuenta el amplio rango de metodologías de resolución de problemas de optimización y las técnicas disponibles para incrementar su eficiencia, se han diseñado e implementado diversos métodos para resolver varios problemas de corte y empaquetado, tratando de mejorar las propuestas existentes en la literatura. Los problemas que se han abordado han sido: el *Two-Dimensional Cutting Stock Problem*, el *Two-Dimensional Strip Packing Problem*, y el *Container Loading Problem*.

Para cada uno de estos problemas se ha realizado una amplia y minuciosa revisión bibliográfica, y se ha obtenido la solución de las distintas variantes escogidas aplicando diferentes métodos de resolución: métodos exactos mono-objetivo y paralelizaciones de los mismos, y métodos aproximados multi-objetivo y paralelizaciones de los mismos. Los métodos exactos mono-objetivo aplicados se han basado en técnicas de búsqueda en árbol. Por otra parte, como métodos aproximados multi-objetivo se han seleccionado unas metaheurísticas multi-objetivo, los *Multi-Objective Evolutionary Algorithm*s, MOEAs. Además, para la representación de los individuos utilizados por estos métodos se han empleado codificaciones directas mediante una notación postfija, y codificaciones que usan heurísticas de colocación e hiperheurísticas.

Tal y como se ha mencionado, algunas de estas metodologías se han mejorado utilizando esquemas paralelos haciendo uso de las herramientas de programación OpenMP y MPI. En el caso de las propuestas exactas, las implementaciones paralelas han hecho posible obtener las soluciones óptimas más rápido, permitiendo abordar instancias más grandes de los problemas. Sin embargo, en el caso de las propuestas aproximadas, los esquemas paralelos no solo han permitido mejorar los tiempos, sino incluso la calidad de las soluciones obtenidas por el algoritmo secuencial.

En resumen, las principales contribuciones de este trabajo son:

- Con respecto al *Two-Dimensional Cutting Stock Problem*:
  - Revisión bibliográfica de las propuestas existentes para resolver el problema.
  - Implementación de un algoritmo exacto y varias paralelizaciones del mismo:
    - Aplicación de reglas de dominancia/duplicación que hacen posible reducir el espacio de búsqueda significativamente.
    - Diseño de dos nuevas propuestas paralelas que permiten acelerar considerablemente la ejecución del algoritmo. La primera propuesta

está basada en un esquema de grano-fino que introduce la generación paralela de meta-rectángulos para el mejor meta-rectángulo del momento. La aproximación de grano-grueso consiste en la ejecución paralela del bucle principal de búsqueda junto con un esquema flexible de sincronización y una estrategia de balanceo de carga. Ambas propuestas paralelas han sido implementadas usando MPI y OpenMP, proporcionando la posibilidad de realizar un estudio de la flexibilidad y eficiencia de estas herramientas de programación paralela.

- Implementación de novedosas propuestas para la resolución del problema con una visión multi-objetivo, (ya que no se conocen trabajos en la literatura que lo aborden como tal) mediante el uso de algoritmos evolutivos, proporcionando:
  - Varios esquemas de codificación directa para la representación de las soluciones.
  - Un esquema de codificación hiperheurística para la representación de las soluciones.

- Con respecto al *Two-Dimensional Strip Packing Problem*:
  - Revisión bibliográfica de las propuestas existentes para resolver el problema.
  - Implementación de un algoritmo exacto (ya que no se conocen trabajos en la literatura que aborden la solución exacta del problema con las restricciones de la utilización de cortes de guillotina y la posibilidad de rotación de las piezas) y varias paralelizaciones del mismo proporcionando:
    - Definición de una nueva cota inferior y una nueva cota superior.
    - Adaptación del algoritmo exacto usado para el problema anterior, permitiendo la rotación de las piezas.
    - Adaptación de las reglas de dominancia/duplicación que hacen posible reducir el espacio de búsqueda, teniendo en cuenta la rotación de las piezas.
    - Aplicación de dos tipos de paralelizaciones que permiten acelerar la ejecución del algoritmo, basadas en los esquemas propuestos para el *Two-Dimensional Cutting Stock Problem*.
  - Implementación de varias propuestas para la resolución del problema con una visión multi-objetivo, mediante el uso de algoritmos evolutivos, mejorando la única aproximación multi-objetivo conocida en la literatura y proporcionando:

        - Un esquema de codificación directa para la representación de las soluciones.
        - Paralelizaciones basadas en islas para dicho esquema.
        - Varios esquemas de codificación hiperheurística para la representación de las soluciones.

- Con respecto al *Container Loading Problem*:

    - Una revisión de las propuestas existentes para resolver el problema.
    - Implementación de una propuesta para la resolución del problema con una visión multi-objetivo, mediante el uso de algoritmos evolutivos, mejorando la única aproximación multi-objetivo conocida en la literatura y proporcionando:
        - Un esquema de codificación que indica una secuencia de colocación de piezas y sus orientaciones. Necesita una rutina de colocación o decodificación que transforma las soluciones codificadas en una distribución completa de piezas.
        - Paralelizaciones basadas en islas usando dicho esquema.

Este trabajo presenta estas contribuciones y algunos fundamentos y antecedentes relacionados. La estructura general de los contenidos se describe a continuación. En primer lugar, el Capítulo 1 recopila la información general indispensable relacionada con los problemas de corte y empaquetado, tal como la tipología y clasificación de los problemas, la revisión bibliográfica, las librerías de instancias de los problemas, etc. Además, el Capítulo 2 introduce los posibles métodos de resolución para este tipo de problemas, cómo medir y comparar la calidad de los resultados obtenidos utilizando estos métodos, y algunas herramientas de apoyo para el desarrollo de los mismos. Algunos de estos métodos se han utilizado para resolver tres problemas representativos de corte y empaquetado. Por esta razón, los siguientes tres capítulos describen los tres problemas y los diversos métodos de resolución experimentalmente aplicados a cada uno de ellos. De este modo, el problema denominado *Two-Dimensional Cutting Stock Problem* se trata en el Capítulo 3, el *Two-Dimensional Strip Packing Problem* se aborda en el Capítulo 4, y el *Container Loading Problem* se detalla en el Capítulo 5. Por último, el Capítulo 6 está dedicado a las conclusiones y a algunas líneas de trabajo futuro.

Índice general

Lista de Algoritmos

Índice de figuras

Índice de tablas

Lista de Acronimos

2DCSP	*Two-Dimensional Cutting Stock Problem*
2DSPP	*Two-Dimensional Strip Packing Problem*
BFDH	*Best-Fit Decreasing Height*
BMVB	*Best-first-search Modified Viswanathan and Bagchi's Algorithm*
CLP	*Container Loading Problem*
CSP	*Cutting Stock Problem*
EA	*Evolutionary Algorithm*
ESICUP	*Euro Special Interest Group on Cutting and Packing*
FFDH	*First-Fit Decreasing Height*
IBEA	*Indicator-Based Evolutionary Algorithm*
MOCLP	*Multi-Objective Container Loading Problem*
MOCSP	*Multi-Objective Cutting Stock Problem*
MOEA	*Multi-Objective Evolutionary Algorithm*
MOP	*Multi-Objective Optimization Problem*
MOSPP	*Multi-Objective Strip Packing Problem*
MPI	*Message Passing Interface*
MVB	*Modified Viswanathan and Bagchi's Algorithm*
NFDH	*Next-Fit Decreasing Height*
NPGA	*Niched Pareto Genetic Algorithm*
NSGA	*Non-Dominated Sorting Genetic Algorithm*
NSGA-II	*Non-Dominated Sorting Genetic Algorithm II*
PMOEA	*Parallel Multi-Objective Evolutionary Algorithm*
SPEA	*Strength Pareto Evolutionary Algorithm*
SPEA2	*Strength Pareto Evolutionary Algorithm 2*
SPP	*Strip Packing Problem*
VB	*Viswanathan and Bagchi's Algorithm*
VEGA	*Vector Evaluated Genetic Algorithm*

Parte I

Fundamentos

Capítulo

1 Corte y empaquetado

Dada la cantidad de aplicaciones prácticas de los llamados problemas de corte y empaquetado (*Cutting and Packing Problems*), y a la diversidad de restricciones que pueden presentar, este capítulo se centra en los fundamentos generales de los mismos. En primer lugar, se definen las propiedades que permiten identificar las diferencias entre los distintos tipos de problemas. Teniendo en cuenta esta distinción de tipos, es posible realizar una clasificación de las variantes más extendidas del problema. El volumen existente de bibliografía relacionada refleja la importancia de estos problemas en muchas áreas de investigación. Mediante la revisión bibliográfica se puede detectar qué tipos de problemas se resuelven con más frecuencia, qué métodos de resolución se han aplicado, e incluso se puede realizar la comparación de las soluciones propias con las ya existentes si se usan las mismas instancias del problema concreto. Por otra parte, la gran variedad de software comercial para llevar a cabo el corte y empaquetado demuestra el fuerte impacto de estos problemas en el ámbito industrial. Es importante destacar que en este capítulo se intenta respetar, en la medida de lo posible, la nomenclatura usada en inglés a la hora de identificar los problemas, dado que no existe una traducción al español formalmente aceptada para algunos nombres y términos referentes al corte y empaquetado.

1.1. Introducción

Los problemas de corte y empaquetado son una familia de problemas de optimización combinatoria que han sido ampliamente estudiados, durante prácticamente medio siglo [29, 107], en numerosas áreas de la industria y la investigación. Son problemas que surgen en muchas industrias de producción donde grandes hojas de material o rollos de materia prima deben cortarse en piezas más pequeñas, o incluso, en el caso del empaquetado de tres dimensiones, se debe determinar la posición de un conjunto de elementos para ser transportados o almacenados.

Los procesos de corte y empaquetado producen *patrones* que no son más que combinaciones geométricas de pequeñas piezas asignadas a objetos de mayor tamaño. Las piezas residuales, es decir, las áreas restantes de los objetos más grandes que no han sido ocupadas por los elementos pequeños, se suelen considerar como desperdicio. Debido al importante papel que juegan los patrones y al tratarse de combinaciones geométricas, se puede decir que los problemas de corte y empaquetado pertenecen al campo de la *geometría combinatoria*. En particular, los problemas de corte y empaquetado trabajan con objetos y piezas definidos por una, dos o tres dimensiones espaciales del espacio Euclídeo. Concretamente en los problemas de corte, los objetos grandes son de materiales sólidos que se cortan en pequeños elementos considerados piezas. Los materiales más comunes son papel o pasta, acero, cristal, madera, plásticos, piel y textiles. En el caso de los problemas de carga y empaquetado, se trata con objetos grandes vacíos, como camiones, furgones o coches, palés, contenedores, cubos, etc. El empaquetado de las pequeñas piezas en dichos objetos se puede considerar como cortes del espacio vacío en partes, algunas de las cuales están ocupadas por pequeñas piezas.

En los problemas de corte, los elementos pequeños son una lista de piezas demandadas, mientras que los objetos de mayor tamaño se conocen como *stock* o materia prima. Los clientes requieren las piezas, y las industrias o fábricas deben producir el conjunto de piezas demandadas usando la materia prima disponible para satisfacer las necesidades de sus clientes. Cuando tratamos con los problemas de empaquetado, los objetos más grandes se conocen como contenedores, mientras que las piezas son los elementos que se deben cargar en esos contenedores, normalmente para su trasporte o almacenamiento.

El objetivo de estos problemas es seleccionar algunas o todas las piezas, agruparlas en uno o más subgrupos y asignar cada uno de los subgrupos resultantes a uno de los objetos grandes de tal manera que se mantengan las condiciones geométricas, es decir, las pequeñas piezas de cada subgrupo tienen que distribuirse en el objeto correspondiente de tal manera que:

- todas las piezas del subgrupo quepan enteras en el objeto de gran tamaño,

- las piezas no se superpongan, y
- se optimice una función objetivo mono o multi-dimensional.

Llegados a este punto, es necesario destacar que una solución del problema puede utilizar algunos o todos los objetos de gran tamaño, y algunas o todas las piezas pequeñas. Considerando estas alternativas, se pueden caracterizar formalmente un conjunto de subproblemas diferentes:

- el problema de selección con respecto a los objetos grandes,
- el problema de selección con respecto a las piezas pequeñas,
- el problema de distribución con respecto a la asignación de los subconjuntos de las piezas pequeñas a los objetos grandes, y finalmente,
- el problema de la disposición con respecto a la distribución de las piezas en cada de los objetos grandes respetando la condición geométrica.

Por supuesto, todos estos aspectos tienen que resolverse simultáneamente para conseguir un óptimo "global". Dependiendo de la definición particular de estos subproblemas y dado el conjunto de restricciones, se tendrá un tipo concreto de problema de corte y empaquetado y por lo tanto, una formulación específica del problema. De cualquier modo, incluso suponiendo el conjunto de restricciones o la formulación del problema más simple, los problemas de corte y empaquetado se clasifican generalmente como problemas NP-duros [61].

1.2. Tipología de los problemas

Una tipología es una organización sistemática de objetos en categorías homogéneas usando unos criterios de caracterización. Así, es posible atribuir a problemas aparentemente heterogéneos fundamentos comunes y reconocer similitudes generales. Es obvio que existe una fuerte relación entre corte y empaquetado, por lo que es común aplicar consideraciones similares a ambas categorías Esta relación es fruto de la dualidad entre material y espacio, la dualidad entre un cuerpo de material sólido y el espacio ocupado por este [63, 108]. Debido a ello, el corte y empaquetado se caracteriza por el hecho de que problemas de prácticamente la misma estructura lógica aparecen bajo nombres diferentes en la bibliografía [63].

Se han propuesto muchos intentos de tipologías y clasificaciones para homogeneizar la nomenclatura usada cuando se hace referencia a problemas de corte y empaquetado [61, 63, 97, 129, 159, 169, 182]. Una tipología ayuda a unificar definiciones y notaciones y, a través de ello, se puede preparar el terreno para trabajos prácticos de investigación. Además, si las publicaciones se categorizan de acuerdo a

una tipología, se proporcionará un acceso más rápido a la literatura relevante. En particular, una tipología en problemas de investigación operativa, proporciona las bases para realizar un análisis estructural de los tipos esenciales, la identificación y definición de los problemas estándar, el desarrollo de modelos y algoritmos, la generación de problemas, etc.

Para proponer una tipología apropiada que permita clasificar de manera fácil la mayoría de los problemas del campo, es necesario analizar los problemas de corte y empaquetado en profundidad de tal manera que se puedan identificar las características principales del problema en sí [140]. Normalmente, estas características principales están relacionadas de alguna manera con los siguientes temas:

- el conjunto de objetos grandes (entrada, suministro),
- el conjunto de piezas pequeñas (salida, demanda),
- los patrones como combinaciones geométricas de pequeñas figuras,
- la asignación de las piezas pequeñas a patrones y de los patrones a los objetos grandes.

En el área del corte y empaquetado, no existe una tipología global con una total aceptación internacional. Sin embargo, la tipología propuesta por Dyckhoff en 1990 [63], ha sido ampliamente usada durante muchos años. Posteriormente, en 2007 se propuso una nueva tipología para los problemas de corte y empaquetado [182]. Esta tipología se basa en la que propuso originalmente Dyckhoff, aunque trata de solventar algunas deficiencias de la misma. Tras una amplia revisión de las tipologías existentes en corte y empaquetado, se ha seleccionado un conjunto de las principales propiedades que definen una especificación dada del problema. Estas propiedades se han usado para clasificar los trabajos relacionados, y están basadas principalmente en las tipologías propuestas en [63, 182]. A continuación, se describen brevemente cada una de las propiedades referentes a los problemas de corte y empaquetado.

1.2.1. Dimensionalidad

Una de las características más importantes es la dimensionalidad. No se trata de una propiedad que se tenga en cuenta por separado para los objetos grandes y las piezas pequeñas; es un atributo del problema o, más concretamente, de los patrones. La dimensionalidad es el menor número de dimensiones que se necesita para describir la geometría de los patrones. Los tipos básicos de problemas según la dimensionalidad son: mono-dimensional (*one-dimensional*), bi-dimensional (*two-dimensional*), tri-dimensional (*three-dimensional*), y multi-dimensional (*multi-dimensional* o *n-dimensional* con $n > 3$).

1.2.2. Medición de cantidad

Otra de las principales características es la manera de medir la cantidad de objetos grandes y de piezas pequeñas. Se pueden distinguir dos casos: medición discreta o entera, es decir, usando número naturales, y medición contínua o fraccionaria, es decir, por medio de números reales. El primer caso se refiere a la frecuencia o el número de piezas de una determinada forma o figura. Por otra parte, las cantidades fraccionarias miden el largo total de varios objetos o piezas que tienen la misma forma con respecto a las dimensiones relevantes, el largo de los objetos o piezas con respecto a otra dimensión no esencial para la geometría de los patrones. En lugar del largo podría ser, por ejemplo, el ancho o el diámetro de los rollos. La combinación de problemas mono-dimensionales con medidas continuas se suele denotar como 1.5-dimensional [65].

1.2.3. Forma de las figuras

La figura de un objeto o una pieza se define como su representación geométrica en el espacio de dimensiones relevantes. Los objetos o piezas de la misma figura tienen la misma representación geométrica, excepto para algunas traslaciones dentro del espacio de dimensiones relevantes. Dejando aparte las traslaciones, una figura se puede determinar únicamente por su forma, tamaño y orientación. En relación con la forma de los objetos, se pueden distinguir dos categorías: regular e irregular. En la mayoría de los casos, las piezas tienen formas regulares, tales como, rectángulos, bloques o cajas, círculos, cilíndros y esferas. El tamaño de los objetos se puede medir por su largo, área o volumen. Con respecto a la orientación de las piezas, se pueden dar tres situaciones diferentes: que se permita cualquier orientación, que se permitan sólo rotaciones de 90 grados, o que la orientación sea fija.

1.2.4. Variedad de piezas pequeñas

Con respecto a la variedad de piezas pequeñas, se pueden distinguir los siguientes casos:

- *Piezas pequeñas idénticas*: todas las piezas tienen la misma forma y tamaño. En algunas formulaciones del problema se puede asumir que se tiene un tipo de pieza individual con una demanda ilimitada.
- *Variedad débilmente heterogénea*: las piezas pequeñas se pueden agrupar en pocas clases (en relación con el número total de piezas), para las que las piezas son idénticas con respecto a la forma y tamaño. La demanda de cada tipo de pieza es relativamente grande, y puede no ser limitada.

- *Variedad fuertemente heterogénea*: sólo muy pocos elementos son de la misma forma y tamaño. Normalmente, si esto ocurre, las piezas se tratan como elementos individuales. Por lo tanto, la demanda de cada pieza es igual a uno.

Para la definición de problemas estándar, es común asumir que el conjunto de piezas pequeñas está uniformemente estructurado, es decir, el conjunto no contiene unas piezas con grandes demandas y otras con pequeñas demandas.

1.2.5. Variedad de objetos grandes

Con respecto a la variedad de objetos grandes, se pueden considerar dos casos diferentes concernientes al número de objetos disponibles:

- *Un objeto grande*: hay sólo un único objeto grande. La extensión del objeto se puede fijar en todas las dimensiones relevantes del problema o sus extensiones pueden ser variables en una o más dimensiones.

- *Varios objetos grandes*: se asume que todas las dimensiones son fijas cuando hay más de un objeto grande disponible. Entre los diferentes objetos disponibles se pueden dar las mismas tres situaciones posibles que en el caso de la variedad de piezas pequeñas, es decir, se pueden considerar todos los objetos grandes idénticos y las variedades débil o fuertemente heterogénea.

1.2.6. Disponibilidad de piezas

En la clasificación de variedad de piezas, sólo se considera el número de objetos disponibles y sus propiedades generales con respecto al tamaño y similitudes. Pero aún existen algunas cuestiones abiertas sobre la disponibilidad de las piezas: su secuencia u orden, así como la fecha o momento en el que un objeto o pieza puede ser o ha sido cortado o empaquetado. Tales restricciones son muy importantes, por ejemplo, en el problema de carga de palés (*pallet loading problems*), donde los objetos más frágiles no se pueden empaquetar en la parte baja del contenedor. Las restricciones en la petición de piezas pueden también aparecer en los problemas de carga de vehículos (*vehicle loading problems*), donde el orden de carga de paquetes en los vehículos viene dada por la secuencia de su posterior distribución.

1.2.7. Tipo de asignación

Normalmente, se analizan dos tipos diferentes de problemas, dependiendo del tipo de asignación de las piezas pequeñas sobre las grandes:

- Maximización de la salida: un conjunto de piezas pequeñas se ha de asignar a un conjunto dado de objetos grandes. El conjunto de objetos grandes no es suficiente para acomodar todas las piezas pequeñas. Todos los objetos grandes se tienen que usar, es decir, no hay problema de selección con respecto a los objetos grandes. El objetivo es asignar una selección (subconjunto) de piezas pequeñas de máximo valor.

- Minimización de la entrada: otra vez, un conjunto de piezas pequeñas ha de asignarse a un conjunto de objetos grandes. Pero en este caso el conjunto de objetos grandes es suficiente para acomodar todas las piezas pequeñas, las cuales han de asignarse a una selección (subconjunto) de grandes objetos de mínimo valor, es decir, no hay problema de selección en cuanto a las piezas pequeñas.

Es necesario destacar aquí, que el "valor" de una selección depende de otras especificaciones. El valor de los objetos puede representarse por costes, ingresos, o cantidades de material. A menudo, se puede asumir que es directamente proporcional al tamaño de la pieza que considera la función objetivo: largo (problemas de una dimensión), área (problemas bi-dimensionales), o volumen (problemas tridimensionales). Pero también se pueden considerar otras alternativas.

1.2.8. Restricciones de los patrones

En algunos problemas hay restricciones asociadas con la combinación geométrica de las figuras pequeñas:

- Distancias mínimas o máximas entre las piezas o entre los cortes. Por ejemplo, cuando se cortan ciertos tipos especiales de material o dependiendo de las propiedades y resolución de las máquinas de corte.

- La orientación de las piezas relativa entre ellas o relativa al objeto en el que se colocan. Por ejemplo, cuando se carga mercancía frágil en palés.

- Pueden haber restricciones con respecto a la frecuencia de una pieza en un patrón. Esto incluye, por ejemplo, una cantidad máxima de desperdicio, así como número máximo de gamas de tamaños o solicitudes.

- El tipo y número de "cortes" permitidos son esenciales particularmente si los objetos y las piezas son rectangulares o tienen forma de bloque.

El tipo y el número de cortes permitidos es uno de los aspectos más importantes cuando se trata con restricciones en los patrones. En la mayoría de los casos, estas

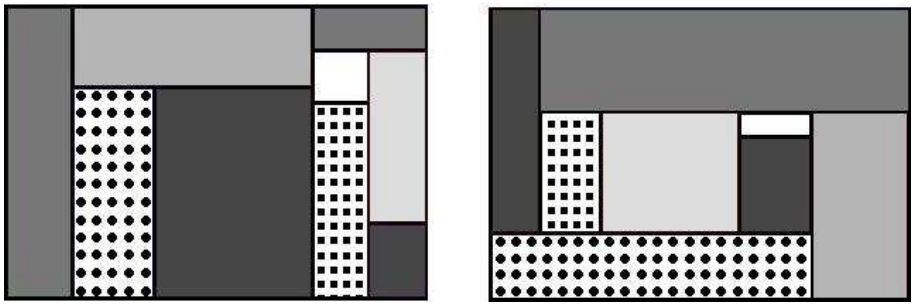

Figura 1.1: Ejemplo de patrón guillotinable (izquierda) y no guillotinable (derecha)

restricciones vienen impuestas por la maquinaria de corte que se usa en el proceso de producción. Los cortes permitidos restringen la manera en la que las piezas pueden combinarse, por lo que se debe prestar una especial atención a los criterios aplicados en la obtención de los patrones adecuados. Esta consideración es especialmente importante cuando se trata con piezas rectangulares o bloques, que, de hecho, son los casos más comunes de problemas de corte y empaquetado. En relación a esto, se ha hecho una distinción entre patrones no-ortogonales y ortogonales. Los cortes ortogonales deben realizarse paralelos a uno de los lados del material rectangular, y pueden emplearse tanto en patrones anidados como de guillotina. El uso de patrones guillotinables significa que cualquier corte debe realizarse desde un lado del material rectangular hasta el otro y además debe ser paralelo a uno de los ejes (Figura 1.1). La complejidad de los patrones guillotinables depende del número de cambios de dirección de corte (etapas) y del número de cortes paralelos por etapa.

1.2.9. Objetivos

No siempre es posible decidir de manera concluyente si una característica es geométrica o combinatoria. Algunas incluyen ambos aspectos y algunas no tienen ninguno de los dos. Los objetivos de los problemas de corte y empaquetado a menudo involucran aspectos geométricos y combinatorios. Además, se le pueden atribuir a los objetos, piezas, patrones o al proceso de asignación. En general, un "objetivo" significa un criterio a maximizar o minimizar. Los criterios que se deben cumplir en un determinado nivel se tratan como restricciones del proceso de corte y empaquetado, por ejemplo, la restricción de que por motivos de gestión sólo las piezas pequeñas pertenecientes al pedido del mismo cliente deberían estar contenidas en un mismo patrón. La diferenciación de los objetivos depende de las propiedades generales del problema en sí. Generalmente, los objetivos de los problemas de corte y

empaquetado dependen de si se refieren a:

- las cantidades de objetos grandes o pequeños y piezas residuales asignadas a los patrones.
- la geometría de los patrones (optimización de la distribución),
- el uso de maquinaria de producción (etapas de corte, número total de cortes,...), o
- la secuencia, combinación o número de patrones.

Tal y como se ha mencionado anteriormente, las funciones objetivo que son lineales con respecto a las cantidades pueden evaluar los objetos o piezas por su tamaño (lo cual afecta al desperdicio de material o a la minimización de la entrada) o, más generalmente, por sus precios. Los objetivos no lineales, por ejemplo, aparecen si el desperdicio relativo se minimiza o si se tienen que tener en cuenta cargas fijas para los patrones. Es típico en muchos problemas de corte tener que considerar más de un objetivo [181]. En la mayoría de los problemas reales que se presentan en la industria, hay muchos objetivos que se deben optimizar simultáneamente. Algo similar ocurre en la industria de corte, donde es imprescindible maximizar el uso de la materia prima mientras simultáneamente se maximiza el uso de la maquinaria industrial. Por ejemplo, si la materia prima es papel (material que no es muy caro), sería recomendable reducir la optimización del material a expensas de obtener patrones que requieran menos cortes, permitiendo producir más rápido la demanda de piezas. Tal compromiso entre los dos objetivos puede ayudar a incrementar la productividad de una fábrica, proporcionando un mejor uso de los recursos involucrados sin un incremento notable del coste asociado con la materia prima necesaria.

1.3. Clasificación de los problemas

En la sección anterior, se han introducido los criterios generales más importantes que describen los problemas de corte y empaquetado. Estos criterios se usan generalmente para clasificar los problemas. Sin embargo, algunas de estas características se superponen, de modo que si se quieren integrar en una clasificación global los distintos tipos de problemas de corte y empaquetado, es necesario realizar un análisis exhaustivo de las propiedades explicadas en la sección previa. Para simplificar la notación en las clasificaciones, se tendrán en cuenta sólo las características más importantes de los problemas. Dyckhoff hizo una propuesta inicial [63] para el conjunto de propiedades más importantes, y más tarde en [182] se propuso una mejora. Un resumen comparativo de ambas propuestas se muestra en la Figura 1.2.

Dyckhoff's C&P typology		Wäscher's improved C&P typology	
Dimensionality			
1	One-dimensional	**1**	One-dimensional
2	Two-dimensional	**2**	Two-dimensional
3	Three-dimensional	**3**	Three-dimensional
N	N-dimensional	**N**	N-dimensional
Kind of assignment			
B	All objects and a selection of items	**OM**	Output value maximisation
V	A selection of objects and all items	**IM**	Input value minimisation
Assortment of large objects			
O	One object	**O**	One object
I	Identical figures		**Oa** all fixed dimensions
D	Different figures		**Oo** one variable dimension
			Om more variable dimensions
		Sf	Several figures
			Si identical figures
			Sw weakly heterogeneous assortment
			Ss strongly heterogeneous assortment
Assortment of small items			
F	Few items (of different figures)	**IS**	Identical small items
M	Many items of many different figures	**W**	Weakly heterogeneous assortment
R	Many items of relatively few different (incongruent) figures	**S**	Strongly heterogeneous assortment
C	Congruent figures		

Figura 1.2: Resumen de las tipologías propuestas por Dyckhoff y Wäscher et al.

Haciendo una combinación de los principales tipos de dimensionalidad, asignación, y variedad se obtienen demasiadas clases de problemas de corte y empaquetado. Si se analizan todas las posibles alternativas una por una no se puede obtener una clasificación adecuada y auto-explicativa. Con el objetivo de proporcionar una clasificación más general para los problemas estándar, se han estudiado las propiedades básicas yendo desde las más generales a las más específicas. De este modo, los tipos básicos de problemas de corte y empaquetado se obtienen realizando una combinación de los criterios de asignación y de la variedad para las piezas grandes y pequeñas.

Considerando el *tipo de asignación* como el criterio global, los tipos básicos de problemas se pueden dividir en dos categorías principales: "maximización de la salida" y "minimización de la salida", como propone Wäscher et al. en [182]. Dentro de cada una de estas dos categorías, se pueden dividir los problemas dependiendo de la *variedad de piezas pequeñas*. Luego, otra vez, cada subproblema se puede subdividir dependiendo de la *variedad de objetos grandes*. Por último, los tipos de problemas

refinados se obtienen aplicando el criterio de *dimensionalidad* y el criterio de *formas de las piezas pequeñas*. Estas últimas subcategorías no se mencionan aquí con detalle, pero en general están caracterizadas por adjetivos que se añaden a los nombres de los tipos de problemas intermedios:

[1, 2, 3]-dimensional [rectangular, circular, ..., irregular] <nombre_problema>

1.3.1. Maximización de la salida

Tal y como ya se ha mencionado anteriormente, una característica común de los problemas de maximización de la salida es que los objetos grandes se suministran sólo en cantidades limitadas que no permiten acomodar todas las piezas pequeñas. Ya que el valor de las piezas acomodadas se ha de maximizar, se usarán todos los objetos grandes. En otras palabras, generalmente hay un problema de selección con respecto a las pequeñas piezas, pero ninguno con respecto a los objetos grandes. Dentro de este tipo de asignación, se pueden reconocer los siguientes problemas básicos [182]:

1.3.1.1. Problema de empaquetado con piezas idénticas (*Identical Item Packing Problem*)

Este tipo de problema consiste en la asignación del mayor número posible de piezas pequeñas idénticas a un conjunto limitado de objetos grandes. Dado el hecho de que todas las piezas son idénticas, el problema se reduce a un problema de distribución, donde se han de colocar las pequeñas piezas en cada uno de los objetos grandes respetando las condiciones geométricas. Esta categoría de problema se puede analizar independientemente de la variedad de objetos grandes, ya que en cualquier caso, se puede dividir en un conjunto de subproblemas independientes, donde cada subproblema consiste en un objeto grande al que se le asignan el mayor número posible de piezas pequeñas.

1.3.1.2. Problema de colocación (*Placement Problem*)

En la bibliografía, los problemas de esta categoría aparecen bajo diferentes nombres. En general, esta categoría representa problemas en los que hay que asignar piezas pequeñas débilmente heterogéneas a un conjunto limitado de objetos grandes. El valor o el tamaño total de las pequeñas piezas colocadas debe maximizarse, o, alternativamente, minimizarse el correspondiente desperdicio. De acuerdo a la variedad de objetos grandes, se pueden considerar los siguientes subproblemas:

- Problema de colocación con un solo objeto grande (*Single Large Object Placement Problem*).

- Problema de colocación con multiples e idénticos objetos grandes (*(Multiple) Identical Large Object Placement Problem*).
- Problema de colocación con múltiples y heterogéneos objetos grandes (*(Multiple) Heterogeneous Large Object Placement Problem*).

1.3.1.3. Problema de la mochila (*Knapsack Problem*)

Este problema es similar al previo, pero considerando una asignación de piezas pequeñas fuertemente, en lugar de débilmente, heterogéneas, es decir, en este caso, las piezas no están agrupadas en relativamente pocas clases, sino que la demanda de cada una de ellas normalmente es igual a uno. De acuerdo con la variedad de objetos grandes, se pueden considerar los siguientes subproblemas:

- Problema de la mochila con un solo objeto grande (*Single Knapsack Problem*).
- Problema de la mochila con múltiples e idénticos objetos grandes (*(Multiple) Identical Knapsack Problem*).
- Problema de la mochila con múltiples y heterogéneos objetos grandes (*(Multiple) Heterogeneous Knapsack Problem*).

1.3.2. Minimización de la entrada

Los problemas de minimización de la entrada se caracterizan por el hecho de que el suministro de objetos grandes es suficiente para acomodar todas las piezas pequeñas. Así, el objetivo de este tipo de problema es minimizar el valor de los objetos grandes necesarios para acomodar todas las piezas pequeñas. Dentro de este tipo de asignación, se encuentran los siguientes tipos básicos de problemas [182]:

1.3.2.1. Problemas de dimensión abierta (*Open Dimension Problem*)

Esta caregoría define problemas en los que un conjunto de piezas pequeñas deben acomodarse completamente en uno o varios objetos grandes. Estos objetos grandes tienen al menos una dimensión que puede considerarse como una variable, es decir, el problema implica una decisión en cuanto a fijar una extensión (o extensiones) de la dimensión (o dimensiones) de los objetos grandes. En general, la mayoría de los problemas de dimensión abierta se considera que sólo hay disponible un objeto grande.

1.3.2.2. Problemas de corte de stock (*Cutting Stock Problem*)

En los problemas de esta categoría se debe colocar por completo una variedad débilmente heterogénea de piezas pequeñas en una selección de objetos grandes de mínimo valor, número o tamaño total. La extensión de los objetos grandes es fija para todas las dimensiones pero se pueden considerar los siguiente subproblemas con respecto a su variedad:

- Problemas de corte con un solo objeto grande (*Single Stock Size Cutting Stock Problem*).
- Problemas de corte con objetos grandes débilmente heterogéneos (*Multiple Stock Size Cutting Stock Problem*).
- Problemas de corte con objetos grandes fuertemente heterogéneos (*Residual Cutting Stock Problem*).

1.3.2.3. Problema de empaquetado de contenedores (*Bin packing problem*)

En este caso, se tiene que asignar una variedad de piezas pequeñas fuertemente heterogéneas a un conjunto de objetos grandes (fuertemente o débilmente heterogéneos), minimizando el valor, el número o el tamaño total de objetos grandes necesarios:

- Problema de empaquetado de un solo objeto grande (*Single Bin Size Bin Packing Problem*).
- Problema de empaquetado de múltiples objetos grandes débilmente heterogéneos (*Multiple Bin Size Bin Packing Problem*).
- Problema de empaquetado con objetos grandes fuertemente heterogéneos (*Residual Bin Packing Problem*).

La clasificación general que se ha seguido aquí fue inicialmente propuesta en [182]. En la Figura 1.3 se muestra un breve esquema de dicha clasificación. Los nombres asignados a cada problema estándar han sido establecidos en [182], pero en otra bibliografía relevante se proponen nombres significativamente diferentes para el mismo tipo de problema [61, 63, 129]. Aunque la clasificación propuesta intenta unificar la notación en los problemas de corte y empaquetado, la falta de acuerdo en esta área de investigación tan importante y de gran alcance, hace imposible unificar completamente la nomenclatura. La tipología presentada al menos facilita la identificación de las propiedades más importantes involucradas en los problemas de corte y empaquetado.

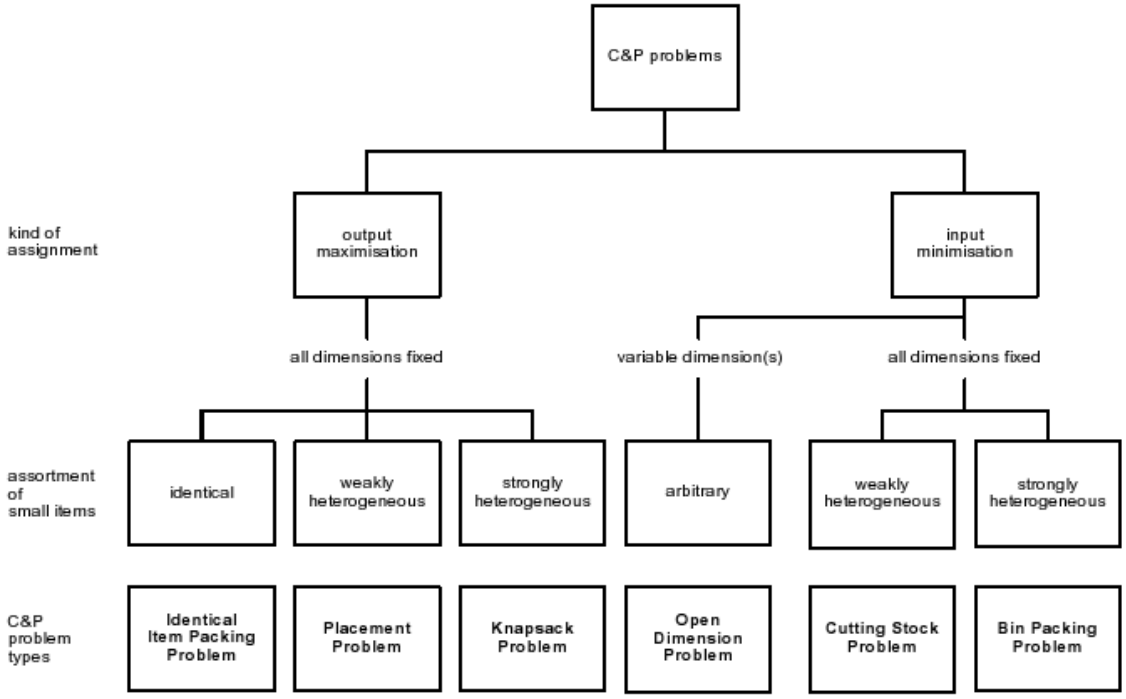

Figura 1.3: Tipos básicos de problemas de corte y empaquetado

1.4. Bibliografía sobre corte y empaquetado

Los problemas de corte y empaquetado se han estudiado ampliamente en disciplinas como: logística, manufactura, ingeniería, administración, informática, matemáticas e investigación operativa. Dada la variedad de aplicaciones y áreas de interés, en este campo ha habido un rápido desarrollo y una gran dispersión de la investigación. Los trabajos científicos en este tipo de problemas empezaron hace más de medio siglo [29, 107]. Desde entonces se ha producido un rápido incremento del número de artículos y trabajos que tratan con aspectos relacionados con ellos. Existen muchas revistas especializadas, congresos y grupos de trabajo que prestan especial atención y centran su interés en estos problemas.

En respuesta a la creciente diversidad y complejidad de investigaciones al respecto, y para facilitar el acceso a las publicaciones relativas a los distintos tipos de problemas y métodos de resolución, se han propuesto varias bibliografías [64, 66, 169]. Para agrupar los problemas en diferentes categorías, la mayoría de las bibliografías siguen consideraciones similares a las presentadas en la sección previa, es decir, se basan en la variedad de piezas, tipo de asignación, dimensionalidad y forma. Con

respecto a las metodologías utilizadas para resolver los problemas, también se pueden considerar diferentes criterios de agrupación, tal y como se verá en el capítulo posterior.

Por otra parte, para proporcionar una acceso fácil a las publicaciones disponibles, las bibliografías también proporcionan una idea general sobre cómo están progresando las investigaciones con respecto a los problemas de corte y empaquetado. Las investigaciones en el campo se han expandido rápidamente en las últimas décadas, y a veces no es fácil saber qué tipo de problemas se estudian más o qué metodologías se aplican con más frecuencia. Una reciente revisión de la bibliografía revela que la mayoría de los problemas estudiados involucran piezas regulares y de dos dimensiones, aunque cada vez se presentan más estudios relativos a los problemas de tres dimensiones. Concretamente dentro de los problemas de tres dimensiones se ha investigado en los últimos años con más frecuencia el caso del *Container Loading Problem* donde se dispone de un contenedor de dimensiones fijas y un conjunto de cajas a colocar en su interior, generalmente intentando maximizar el volumen final. Utilizando la tipología presentada en [182] y en este capítulo, este problema correspondería a un problema de maximización de la salida. En cuanto a la metodología para resolver problemas de corte y empaquetado, las aproximaciones exactas y las heurísticas se han reemplazado en los últimos años por muchos tipos de metaheurísticas.

Particularmente, es digno de mención el *Euro Special Interest Group on Cutting and Packing*, ESICUP [70]. Este grupo reúne a profesionales, investigadores y educadores en investigación operativa con intereses en el área del corte y empaquetado. Su propósito es mejorar la comunicación entre las personas que trabajan en este campo. El ESICUP coordina y promueve la organización de reuniones especializadas. Por otra parte, en un esfuerzo por simplificar el acceso a la investigación en el campo, ESICUP mantiene una bibliografía completa y actualizada de los artículos publicados en ciertas conferencias y revistas relacionadas.

1.5. Instancias

Cuando se trata con una formulación determinada de un problema de corte y empaquetado, es necesario disponer de un conjunto de instancias del problema que proporcionen las propiedades de éste. Por ejemplo, si se trata con un problema de maximización de la salida, es necesario saber el número de objetos grandes y sus dimensiones, mientras que si es un problema de minimización de la entrada, esa información normalmente se obtiene como solución al problema. Por lo tanto, para cada tipo de problema se necesita un tipo específico de datos. En el caso de las aplicaciones reales, estos tests vienen dados por las demandas de usuarios reales,

mientras que en estudios de investigación, es esencial definir un conjunto apropiado de instancias del problema.

Cuando se prueban aproximaciones para resolver un problema en el campo de investigación del corte y empaquetado, las instancias del problema se deben generar de alguna manera. Es recomendable el uso de una gran variedad de instancias del problema concreto que se esté abordando, ya que así se puede comprobar de manera más adecuada la validez de las propuestas. Estas instancias deben incluir piezas de diferentes tamaños y áreas, de modo que permitan a los investigadores determinar si sus algoritmos son correctos para cualquier tipo particular de datos. En las comprobaciones iniciales se pueden usar instancias más sencillas que ayuden a detectar errores y verifiquen la validez de las propuestas. Para estas instancias simples se debe poder calcular la solución exacta, permitiendo así conocer la efectividad de los algoritmos. Sin embargo, estos conjuntos de datos de tamaño pequeño no proporcionan una determinación de si los algoritmos en cuestión podrán afrontar problemas grandes, que normalmente se ajustan mejor a las demandas reales en los problemas de corte y empaquetado. Al mismo tiempo, gestionar estos problemas más largos como pruebas para el algoritmo puede ser bastante complicado: el tiempo de computación necesario podría crecer rápidamente, lo cual se traduce en que a menudo la calidad de la solución sólo es aproximada, ya que puede ser muy duro obtener la solución óptima exacta.

Cuando se propone un algoritmo para solucionar un problema concreto de corte y empaquetado, una manera de determinar su calidad es compararse con otras aproximaciones existentes para resolverlo. Como ya se ha mencionado anteriormente, las tipologías de los problemas de corte y empaquetado y la existencia de bibliografía asociada, hacen posible acceder a otros trabajos que tratan de resolver una formulación del problema similar. Una vez que se localizan esos trabajos, es necesario probar con el mismo conjunto de instancias del problema. De este modo, se pueden comparar las diferentes propuestas directamente.

Para simplificar la elección del conjunto de problemas, se pueden usar los conjuntos de datos ya definidos en los trabajos asociados. Normalmente, estas instancias han sido definidas por otros autores que han trabajado en el problema. Además, para esas instancias se suele conocer la solución exacta y se pueden consultar varios resultados obtenidos por otros algoritmos ya existentes, lo cual proporciona una manera sencilla de comprobar la calidad de nuestras propuestas. Por estas razones, se han puesto a disposición varias librerías de instancias para problemas de corte y empaquetado que sirven como punto de referencia:

- *DEIS* - Operations Research Group Library of Instances [56],
- *PackLib*^{2} [150], y
- *OR-Library* [149].

También ESICUP [70] proporciona un amplio conjunto de instancias, que se encuentran clasificadas por número de dimensiones (1D, 2D o 3D) y por tipo de objetos disponibles (regulares o irregulares). Después ya se puede comprobar el tipo de variedad de piezas pequeñas y objetos grandes disponibles.

Además, también puede ser útil saber cómo se han generado las instancias, lo cual proporciona al usuario la posibilidad de generar nuevas instancias del problema más sencillas o complejas que las disponibles. No todos los autores describen explícitamente los métodos aplicados para generar los conjuntos de datos, pero existen artículos de investigación dedicados específicamente a la generación de tests [24, 136, 179].

1.6. Software

Dada su aplicación a muchos procesos de producción industrial, se ha desarrollado una gran variedad de software de corte y empaquetado. La mayoría de las aplicaciones se desarrollan y distribuyen por las compañías de software, normalmente aplicando licencias propietarias. Suele haber disponibles versiones de prueba de estos programas, pero su funcionamiento está limitado. El número de herramientas no comerciales es muy reducido y sus funcionalidades son mucho más limitadas que las ofrecidas por los productos comerciales.

El software de corte y empaquetado proporciona a los usuarios métodos alternativos para introducir los conjuntos de datos definiendo las propiedades del problema (dimensionalidad, forma de las figureas, tipo de cortes, ...), e incluso para mostrar las composiciones de la solución. A continuación se nombran algunos paquetes de software para corte y empaquetado:

- **CutLogic** (http://www.tmachines.com): es un optimizador 1D y 2D para cortar materiales rectangulares en industrias como la de madera, muebles, metal, cristal, etc. Permite cortes de guillotina y no guillotina, y aplica técnicas basadas en algoritmos genéticos para proporcionar una optimización de alta calidad y confianza para varios escenarios de corte.

- **CutMaster 2D** (http://www.cutmaster2d.com): es un paquete de software profesional. Permite reducir el desperdicio y el coste ya que realiza cortes de alto rendimiento reutilizando los recortes. Utiliza algoritmos avanzados diseñados especialmente para optimizar la disposición de las piezas en el material, que puede ser metal, madera, cristal y similares. Trata de maximizar la productividad al ahorrar tiempo necesario para crear y analizar patrones.

- **Cutting 3** (http://www.cuttinghome.com): usa un algoritmo avanzado y permite un número ilimitados de paneles y piezas para cortar. Se puede interac-

tuar con los materiales moviendo las piezas deseadas a otra hoja de material y permite importar y exportar los resultados desde y para otros programas como el AutoCad.

- **Cutting Optimization Pro** (http://www.optimalprograms.com): es un programa para corte 1D y 2D. Se puede usar para el corte de hojas rectangulares de madera, cristal, metal y otros materiales industriales. También se usa para cortar piezas lineales como barras de madera, tuberías, barras de metal, etc.

- **Smart2DCutting** (http://www.rasterweq.com/specifications.htm): es un paquete de software profesional para la optimización de corte. Usa algoritmos que generan los patrones de corte más optimizados proporcionando la máxima productividad del material. Ha sido probado con trabajos de más de 2000 piezas produciendo resultados rápidamente.

- **Astrokettle Software Products** (http://www.astrokettle.com): proporciona empaquetadores 2D y 3D (*2D/3D Load Packers*) que optimizan el espacio compactando distribuciones de objetos rectangulares de ambas dimensiones y un programa de corte 1D (*1D Stock Cutter*), herramienta que permite optimizar la distribución de un gran dúmero de piezas demandas de diferentes largo, minimizando el desperdicio.

- **CargoWiz** (http://www.softtruck.com): es un software para la planificación de la carga de camiones y contenedores que calcula la carga y el espacio necesarios para un envío.

- **CargoManager** (http://www.packyourcontainer.com/index.htm): es un programa que permite planear y ver la carga de miles de contenedores. Ayuda a maximizar la utilización del espacio en contenedores, palés o cualquier otro espacio rectangular. Tiene en cuenta varias restricciones en la carga, como el peso y la fragilidad de la carga.

- **MaxLoad Pro** (http://www.topseng.com/MaxLoadFeatures.html): es un software de optimización que planifica la carga de contenedores y camiones. Determina cómo se deben cargar los diferentes tipos de productos con distintos tamaños y a varias escalas. Parte de los productos en cajas, luego carga las cajas en palés y finalmente de los palés se pasa a camiones.

- **AutoLoad Pro** (http://www.coptimal.com/products/autoload.htm): es un software de optimización automático que optimiza la carga de camiones, contenedores, cajas y palés. Utiliza una técnica de optimización con dos estados.

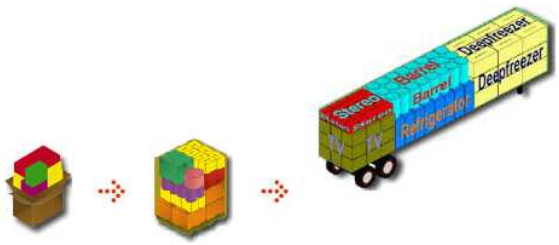

Figura 1.4: Estructura de funcionamiento de MaxLoad Pro

La carga y visualización contempla restricciones como la orientación, la seguridad, las secuencias de carga o la agrupación de piezas.

- **PackVal** (http://www.packvol.com): se trata de un software de planificación de carga, diseñado para el mejor uso del espacio en contenedores y camiones, ayudando a reducir el coste del transporte. Es un programa escrito en C++ pero diseñado para Microsoft Windows. Dispone de un modo automático y un modo manual.

1.7. Maquinaria

Tanto si se ha usado un software específico de corte o empaquetado como si no, una vez que se ha resuelto uno de estos problemas, se puede iniciar el proceso para obtener el conjunto de piezas demandadas o el empaquetamiento deseado. Con este fin, se debe proporcionar como entrada a la máquina la solución obtenida. Existen muchos manuales de máquinas que requieren de una supervisión por parte de los trabajadores, pero hay otras máquinas más modernas que incorporan ya el software y solamente es necesario indicar una simple especificación, por ejemplo, de los patrones de corte a producir. A continuación se exponen algunos ejemplos de distintos tipos de máquinas de corte y empaquetado:

- **CadCam Technology** (http://www.cct-uk.com/laser_cutting_machines.htm): se trata de un proveedor de máquinas de corte mediante láser capaces de producir cortes finos, limpios y con una profundidad exacta. El corte se consigue mediante software que ajusta la velocidad del corte y la potencia del láser. Se puede cortar distintos tipos de material como plástico, papel, madera, cuero, tejidos, cinta adhesiva, etc.

- **OMAX 120X Series** (http://www.omax.com/waterjet-cutting-machines/-model-120X%20Series.php): el modelo 120X Series corta piezas complejas,

usando un chorro de agua, de la mayoría de materiales incluyendo metal, plástico, vidrio, cerámica, piedra y otros compuestos directamente desde un dibujo CAD o un archivo DXF. Puede cortar materiales muy duros, reflectantes y materiales no conductores. El acabado es de buena calidad, eliminando la necesidad de mecanizado secundario.

Figura 1.5: Máquina de corte OMAX 120X Series

- **GKACS (Short Closed Guillotine Knife Assembly)** (http://www.aboxautomation.com/index_files/GKACS.htm): es una manera efectiva y de bajo coste para cortar materiales flexibles en tamaños manejables. Estos cuchillos de guillotina se pueden usar para una gran variedad de materiales, incluyendo: papel, plástico, metales finos, espuma, etc.

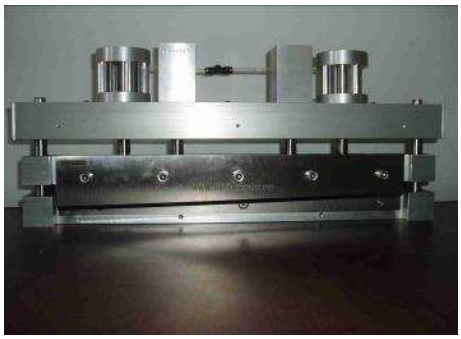

Figura 1.6: Máquina de corte GKACS

- **Five Star Programmable Cutting Machine** (http://www.fivestarmachines.com/cutting1.htm): es una máquina de corte hidráulica, programable y muy avanzada. Está especializada en el corte de papel. Dispone una pantalla

Figura 1.7: Máquina de corte Five Star Programmable

en color para su control, que puede ser de tres tipos: control manual, control programado o modo aprendizaje.

- **SteelTailor Legend Serie** (http://sp.steeltailor.com/sproducts/sheet_matel_plasma_cutting_machine/id/SteelTailor Legend Serie Maquina de Corte para Delgada Metal Por Plasma.html): se trata de una máquina de alta precisión y velocidad de corte, que en lugar de utilizar láser, corta hojas delgadas de metal usando plasma. Es compatible con la mayoría de software relacionado.
- **Inline Ultrasonic Cutting Machine** (http://www.bakon.eu.com/84.html): es una máquina que corta usando un sistema de ultrasonido. Esto significa que la cuchilla corta mediante vibración. Realiza cortes con gran precisión y es usada en sobre todo repostería.
- **Combo Pack Palletizer** (http://www.schneiderequip.com/?page=ctp_rcpp): es una empacadora robótica. Además de empaquetar la mercancía, la distribuye automáticamente en palés.

Figura 1.8: Máquina de empaquetado Combo Pack Palletizer

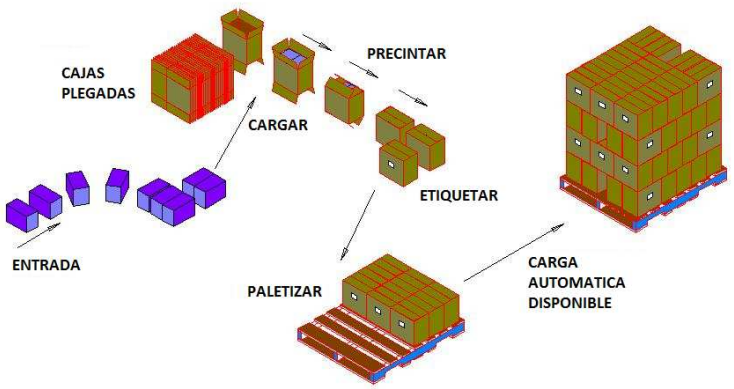

Figura 1.9: Funcionamiento de Combo Pack Palletizer

- **IC Filling Systems** (http://www.icfillingsystems.com/palletisers.php?lang=en): esta compañía dispone de diferentes modelos de maquinas para el empaquetado en palés. Entre ellas se encuentra el "automatic palletizer", utilizado para una gran variedad de contenedores en forma de palés (cajas de cartón, paquetes, bolsas, sacos, etc).
- **ROBOX Palletizer** (http://www.schneiderequip.com/?page=rps): se trata de un robot al que se le deja de entrada un palé vacío y luego se puede recoger a la salida la carga completa del mismo. Dispone de seis ejes de movimiento diferentes.

Figura 1.10: Máquina de empaquetado ROBOX Palletizer

Capítulo

2 Métodos de resolución

Dada la cantidad de características de los problemas de corte y empaquetado que se han visto en el capítulo anterior, y por lo tanto, la gran variedad de problemas de este tipo que se pueden presentar, se necesitan métodos de resolución de diversa categoría que permitan obtener su solución en función de las necesidades. Por esta razón, en este capítulo se verán los distintos métodos de resolución para los problemas de corte y empaquetado, y en general para los problemas de optimización combinatoria, conjunto de problemas a los que pertenecen los primeros. Para comenzar, se hará una breve introducción distinguiendo entre optimización mono-objetivo y multi-objetivo, ya que a la hora de proponer un método de resolución para un problema de optimización, es recomendable tener en cuenta el enfoque y las necesidades que se tienen en relación al problema y su solución. En general, los distintos métodos de resolución se pueden aplicar en ambos tipos de optimización (mono y multi-objetivo), aunque algunos son más viables que otros en cada caso. La clasificación de los métodos utilizados en la solución de problemas de optimización se hará de la siguiente forma: algoritmos exactos, heurísticos, metaheurísticos, hiperherísticos e híbridos o combinaciones de los anteriores. Normalmente, los primeros se consideran prácticos para instancias pequeñas o de poca complejidad en optimización mono-objetivo. Por el contrario, las heurísticas, metaheurísticas e hiperheurísticas son métodos que se consideran prometedores para instancias de grandes magnitudes, imposibles de abordar por métodos exactos en un tiempo aceptable. Es decir, se aplican cuando el número de elementos del conjunto de soluciones es muy elevado, haciendo impracticable la evaluación de todas sus soluciones para determinar el óptimo. Para obtener lo mejor de estos métodos se requiere conocer, además del tipo de optimización y el tamaño del problema, los medios computacionales de los que se dispone, puesto que el uso de máquinas e implementaciones paralelas puede reducir considerablemente los tiempos para obtener una solución.

2.1. Optimización mono y multi-objetivo

Un problema de optimización es aquel para el que no es suficiente encontrar una solución cualquiera, sino que esa solución debe ser la mejor posible: la solución *óptima*. En la mayoría de los problemas reales, la solución óptima está relacionada con los recursos, el tiempo y el coste. Un problema de optimización se compone de tres elementos básicos:

- Una *función objetivo* a minimizar o maximizar.
- Un conjunto de *variables de decisión* que afectan a los valores de la función objetivo.
- Un conjunto de *restricciones* que permiten a las variables tomar ciertos valores y excluir otros.

Considerando estos elementos, se puede definir un problema de optimización en términos generales [140].

Definición 1 *Un **problema de optimización** consiste en encontrar los valores de las variables que minimizan/maximizan la función objetivo satisfaciendo las restricciones.*

En relación a la función objetivo, la mayoría de los problemas de optimización se han formulado en base a una función mono-objetivo [44].

Definición 2 *Un **problema de optimización mono-objetivo** general consiste en minimizar/maximizar $f(x)$ sujeto a $g_i(x) \leq 0,\ i = \{1, ..., m\}$, y $h_j(x) = 0,\ j = \{1, ..., p\}$ $x \in \Omega$. Una solución minimiza/maximiza el escalar $f(x)$, donde x es una variable de decisión en forma de vector n-dimensional $x = (x_1, ..., x_n)$ de algún universo Ω.*

Téngase en cuenta que $g_i(x) \leq 0$ y $h_j(x) = 0$ representan restricciones que deben cumplirse mientras se optimiza $f(x)$. Ω contiene todas las posibles x que se pueden usar para satisfacer una evaluación de $f(x)$ y sus restricciones. Además, x puede ser un vector de variables contínuas o discretas, y f puede ser contínua o discreta. En general, el mínimo global de un problema mono-objetivo se define como [11, 44]:

Definición 3 *Dada una función $f\colon\ \Omega \subseteq \Re^n \to \Re,\ \Omega \neq \emptyset$, para $x \in \Omega$ el valor $f(x^*) > -\infty$ se llama **mínimo global** si y solo si*

$$\forall x \in \Omega\colon\ f(x^*) \leq f(x)$$

x^* es una definición de la solución mínimo global, f es la función objetivo, y el conjunto Ω es la región factible de x. El objetivo de determinar la o las soluciones que son mínimo global se llama ***problema de optimización global*** *para un problema mono-objetivo.*

Sin embargo, en muchos problemas de ingeniería es necesario optimizar varios objetivos diferentes al mismo tiempo. Normalmente, los diferentes objetivos son mutuamente competitivos o exclusivos, es decir, los valores de las variables que optimizan un objetivo pueden hacer distar mucho a los otros objetivos de los valores óptimos. Por lo tanto, no existe una única solución (como en el caso de los problemas de optimización mono-objetivo), sino un conjunto de soluciones. Más concretamente, los problemas de optimización multi-objetivo (*Multi-Objective Optimization Problem*s, MOPs) son aquellos problemas donde el objetivo es optimizar k funciones objetivo al mismo tiempo. Esto supone la maximización de las k funciones, la minimización de las k funciones o la combinación de maximizar y minimizar esas k funciones. La formulación mono-objetivo se extiende para reflejar la naturaleza de los MOPs donde no hay una sóla función objetivo a optimizar, sino muchas [160, 168]:

Definición 4 *Un* ***problema de optimización multi-objetivo o multi-criterio*** *general se define como minimizar (o maximizar) $F(x) = (f_1(x), ..., f_k(x))$ sujeto a $g_i(x) \leq 0,\ i = \{1, ..., m\}$, y $h_j(x) = 0,\ j = \{1, ..., p\}\ x \in \Omega$. Una solución a un MOP minimiza/maximiza los componentes de un vector $F(x)$ donde x es una variable de decisión en forma de vector n-dimensional $x = (x_1, ..., x_n)$ de algún universo Ω. Téngase en cuenta que $g_i(x) \leq 0$ y $h_j(x) = 0$ representan restricciones que deben satisfacerse mientras se minimiza/maximiza $F(x)$, y Ω contiene todas las posibles x que pueden usarse para satisfacer una evaluación de $F(x)$.*

Por lo tanto, un MOP está compuesto por k objetivos reflejados en las k funciones objetivo, $m + p$ restricciones en las funciones objetivo y n variables de decisión. Las k funciones objetivo pueden ser lineales o no lineales y contínuas o discretas. La evaluación de la función, $F\colon \Omega \rightarrow \Lambda$, es una asignación del vector de variables de decisión $(x = x_1, ..., x_n)$ a los vectores de salida $(y = a_1, ..., a_k)$. Por supuesto, el vector de variables de decisión x_i puede ser también contínuo o discreto.

Teniendo varias funciones objetivo cambia la noción de "óptimo", ya que en los MOPs, el objetivo es encontrar un compromiso , no una sóla solución, como en la optimización mono-objetivo. El concepto de "óptimo" más comúnmente adoptado en los MOPs se presenta a continuación [42, 43, 44]:

Definición 5 *Una solución $x \in \Omega$ se dice que es un* ***óptimo de Pareto*** *con respecto a Ω si y sólo si no hay un $x' \in \Omega$ tal que $v = F(x') = (f_1(x'), ..., f_k(x'))$*

domine a $u = F(x) = (f_1(x), ..., f_k(x))$*. La expresión óptimo de Pareto se entiende con respecto al espacio de decisión completo a menos que se especifique lo contrario.*

Normalmente, esta definición dice que x^* es un óptimo de Pareto si no existe un vector x factible que disminuya algún criterio sin causar un aumento simultáneo en el resto de criterios (asumiendo maximización). El concepto de óptimo de Pareto fue formulado por Vilfredo Pareto en el siglo XIX [151] y constituye por si mismo el origen de la investigación sobre optimización multi-objetivo. Desafortunadamente, el óptimo de Pareto casi nunca da una solución única, sino un conjunto de soluciones llamadas soluciones *no-inferiores* o *no-dominadas* [42, 43, 44]:

Definición 6 *Un vector* $u = (u_1, ..., u_k)$ *se dice que* ***domina*** *otro vector* $v = (v_1, ..., v_k)$ *(denotado por* $u \preceq v$*) si y sólo si* u *es parcialmente menor que* v*, es decir,* $\forall i \in \{1, ..., k\}$*,* $u_i \leq v_i \ \wedge \ \exists i \in \{1, ..., k\} : u_i < v_i$*.*

Definición 7 *Para un MOP dado,* $F(x)$*, el* ***conjunto óptimo de Pareto****,* $\mathcal{P}^*$*, se define como:*

$$\mathcal{P}^* := \{x \in \Omega \mid \neg\exists x' \in \Omega \ \ F(x') \preceq F(x)\}$$

Cuando se dibujan en el espacio de objetivos, los vectores no-dominados son conjuntamente conocidos como el *frente de Pareto* [42, 43, 44]:

Definición 8 *Para un MOP dado,* $F(x)$*, y el conjunto óptimo de Pareto,* $\mathcal{P}^*$*, el* ***frente de Pareto*** $\mathcal{PF}^*$ *se define como:*

$$\mathcal{PF}^* := \{u = F(x) \mid x \in \mathcal{P}^*\}$$

Aunque los problemas de optimización mono-objetivo pueden tener una única solución, los MOPs normalmente tienen un conjunto de soluciones en un frente de Pareto. Cada solución asociada con un punto del frente de Pareto es un vector cuyos componentes representan un compromiso en el espacio de decisiones o espacio de soluciones de Pareto. De este conjunto final de soluciones obtenido en forma de frente de Pareto, una persona encargada de tomar las decisiones podrá elegir la solución de compromiso más adecuada.

Dado que los métodos exactos, que se verán a continuación, son prácticamente inabordables para los MOPs, se ha diseñado una gran variedad de algoritmos aproximados. Para simplificar la solución de estos problemas existen dos enfoques comunes: convertir el problema original en un problema de optimización mono-objetivo o traducir alguno de los objetivos en restricciones. Sin embargo, la aplicación de estas aproximaciones requieren de un conocimiento previo del problema que no siempre está disponible. Además, estas técnicas pierden en diversidad de soluciones y pueden

ser sensibles a la forma del frente óptimo de Pareto. Por lo tanto, los MOPs requieren de otras alternativas para su solución. Normalmente, una aproximación más apropiada supone la aplicación de técnicas que traten específicamente con múltiples objetivos y con la complejidad intríseca de estos problemas (espacios de búsqueda muy grandes, incertidumbre, ruido, curvas de Pareto disjuntas, etc.).

En definitiva, tal y como se ha dicho anteriormente, cada tipo de problema de optimización (mono y multi-objetivo) requiere de un método de resolución diferente, adecuado a sus propiedades y a los medios de los que se disponga. Es decir, así como en general es inviable la aplicación de un método exacto a un problema multi-objetivo, existen problemas mono-objetivos que requieren concretamente de una solución exacta. Sin embargo, por razones de tiempo o recursos, en ocasiones se aplican otros métodos que proporcionan unos resultados de buena calidad, pero quizás no de la óptima. Todos estos métodos se detallan a continuación.

2.2. Métodos exactos

Mientras que las heurísticas y las metaheurísticas proporcionan soluciones *suficientemente buenas* en un tiempo *computacionalmente aceptable*, tal y como se verá en las siguientes secciones, los métodos exactos dan la solución óptima global. Sin embargo, el coste de la exactitud suele significar una convergencia lenta. Algunos de los algoritmos exactos más conocidos para resolver problemas de corte y empaquetado se basan en algoritmos de ramificación y acotación (*branch-and-bound*), en búsquedas en profundidad (*depth-first*) o primero-el-mejor (*best-first*).

Los algoritmos de búsqueda se han usado ampliamente para resolver problemas de optimización combinatoria. Se trata de algoritmos que evalúan las soluciones candidatas partiendo de un conjunto finito de posibles soluciones, para encontrar una que satisfaga un criterio específico del problema. En general, un problema de optimización combinatoria puede reformularse como un problema de búsqueda de un camino de mínimo coste en un grafo desde un nodo inicial a otro.

El algoritmo de búsqueda secuencial más adecuado para su aplicación a un espacio de estados depende de si el espacio forma un grafo o un árbol. En un árbol, cada nuevo sucesor conduce a una parte no explorada del espacio de búsqueda, mientras que en un grafo, se puede llegar a un estado por múltiples caminos. Para problemas basados en grafos, cuando se genera un estado, es necesario comprobar si el estado ya se ha generado previamente. Si esta comprobación no se lleva a cabo, entonces el gráfo de búsqueda se está desplegando implícitamente en un árbol en el que un estado se repite para cada camino que conduce a él. Ya que cualquier espacio de búsqueda basado en grafo se puede desplegar en una estructura de árbol, nos centraremos en

los algoritmos de búsqueda basados en árbol. Considerando el caso de un espacio de búsqueda en árbol, el nodo inicial se conoce como *nodo raíz* y un terminal o nodo objetivo como *nodo hoja*. Para explorar sistemáticamente una estructura de árbol, en la literatura se han propuesto algunos métodos básicos de exploración:

- **Búsqueda en amplitud**: siempre expande el nodo del árbol con menor profundidad. Esta búsqueda explora los nodos en el mismo orden en el que se crean, lo cual significa que usa una cola o estructura *FIFO (First-In-First-Out)* para almacenar los nodos que se han generado pero no han sido examinados todavía. Aunque requiere el almacenamiento en memoria del árbol generado hasta el momento, su principal ventaja es que garantiza encontrar la solución de mínima profundidad en el árbol.
- **Búsqueda en profundidad**: Después de expandir un nodo dado, el algoritmo intenta expandir uno de sus nodos recién generados, es decir, siempre selecciona el nodo de mayor profundidad en el árbol para expandirse. Las búsqueda en profundidad explora los nodos en el orden inverso de su creación, empleando una pila o estructura del tipo *LIFO (Last-In-First-Out)* para almacenar los nodos que se han generado pero que aún no se han examinado. Una ventaja de la búsqueda en profundidad es que suele requerir menos almacenamiento que cualquier otra estrategia de búsqueda. Además, esta estrategia también intenta generar una solución inicial rápidamente, de manera que se pueda usar como poda para otras secciones del árbol.
- **Búsqueda primero-el-mejor**: expande el problema de acuerdo a sus valores límites. Los subproblemas con los mejores valores límites se eligen para la expansión antes que los otros. Una ventaja de esta aproximación es que, siempre que los valores límites sean únicos, nunca expandirá un estado del problema que no haya sido expandido por otro método.
- **Búsqueda de ramificación y acotación**: continúa la búsqueda incluso después de encontrar un camino a una solución. Cuando encuentra un nuevo camino para la solución, actualiza la mejor solución actual. De este modo, es posible descartar soluciones parciales inferiores, es decir, caminos hacia soluciones parciales cuya extensión está garantizada que es peor que el actual mejor camino a la solución. Al terminar, la actual mejor solución es la solución óptima global.

De este modo, una estrategia en amplitud encuentra caminos a soluciones con menor profundidad, mientras que un método en profundidad encuentra caminos

que están en la parte más a la izquierda del árbol. Sin embargo, si el método para cuando se encuentra la primera solución, se habrá encontrado una solución factible, pero no necesariamente la óptima. Una posibilidad para encontrar una solución óptima implica la exploración del espacio completo de búsqueda, analizando todas las posibles soluciones antes para finalmente seleccionar la mejor. En los problemas más complejos una exploración exhaustiva es inabordable, así que frecuentemente se usan técnicas que evitan explorar el espacio de búsqueda completo, por ejemplo, la búsqueda primero-el-mejor o los métodos de ramificación y acotación.

Otra posibilidad a la hora de enfrentarse a problemas muy complejos cuando se necesita la solución óptima global, es utilizar técnicas para paralelizar los algoritmos exactos. El cómputo paralelo consiste en la ejecución de más de un cálculo al mismo tiempo usando más de un procesador en una computadora. La meta es reducir al mínimo el tiempo total de cómputo distribuyendo la carga de trabajo entre los procesadores disponibles. Una de las razones principales para utilizar paralelismo, es obtener un alto rendimiento o mayor velocidad al ejecutar un programa. El proceso de paralelizar un programa exige al programador conocer un poco más de la arquitectura de la supercomputadora o *cluster* sobre el cual pretende paralelizar su código, conocer el número de procesadores con los que se cuenta, la cantidad de memoria, espacio en disco, los niveles de memoria disponible, el medio de interconexión, etc. En términos de software el usuario debe conocer qué sistema operativo esta manejando, si los compiladores instalados permiten realizar aplicaciones con paralelismo, si se cuenta con herramientas como PVM o MPI (en sistemas distribuidos), Power C, Power Fortran u OpenMP (en sistemas de memoria compartida), etc. Para evaluar si vale la pena implementar paralelismo, hay que tener en cuenta si existe una necesidad de respuesta inmediata de resultados, y si el algoritmo demanda grandes recursos de cómputo. Es importante que el usuario sepa identificar las tareas independientes, además de localizar las zonas donde se efectúa la mayor carga de trabajo y que consuma la mayor parte de tiempo de ejecución. En el caso de los algoritmos de búsqueda anteriormente mencionados basados en una estructura de árbol, resulta evidente que el cómputo de las diferentes ramas se presta a posibles paralelizaciones.

2.3. Heurísticas

La idea intuitiva de problema *"difícil de resolver"* queda reflejada en el término científico *NP-hard* utilizado en el contexto de la complejidad algorítmica. En términos coloquiales podemos decir que un *problema de optimización difícil* es aquel para el que no podemos garantizar el encontrar la mejor solución posible en un tiempo

razonable. La existencia de una gran cantidad y variedad de problemas difíciles, que aparecen en la práctica y que necesitan ser resueltos de forma eficiente, impulsó el desarrollo de procedimientos eficientes para encontrar buenas soluciones aunque no fueran las óptimas. Estos métodos, en los que la rapidez del proceso es tan importante como la calidad de la solución obtenida, se denominan heurísticos o aproximados. El término *heurística* deriva del griego *heuriskein*, que significa *encontrar* o *descubrir*. Existen muchas definiciones diferentes de *algoritmo heurístico*, entre las que destaca la siguiente:

Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución.

En contraposición a los métodos exactos que proporcionan una solución óptima del problema, los métodos heurísticos se limitan a proporcionar una buena solución no necesariamente óptima. Lógicamente, el tiempo invertido por un método exacto para encontrar la solución óptima de un problema difícil, si es que existe tal método, es de un orden de magnitud muy superior al del heurístico (pudiendo llegar a ser tan grande en muchos casos, que sea inaplicable).

Otras razones para utilizar métodos heurísticos son las siguientes:

- El problema es de una naturaleza tal que no se conoce ningún método exacto para su resolución.
- Aunque existe un método exacto para resolver el problema, su uso es computacionalmente muy costoso.
- El método heurístico es más flexible que un método exacto, permitiendo, por ejemplo, la incorporación de condiciones de difícil modelización.
- El método heurístico se utiliza como parte de un procedimiento global que garantiza el óptimo de un problema, bien porque el método heurístico proporciona una buena solución inicial de partida, o porque participa en un paso intermedio del procedimiento.

Al abordar el estudio de los algoritmos heurísticos se puede comprobar que dependen en gran medida del problema concreto para el que se han diseñado. En otros métodos de resolución de propósito general, como pueden ser los algoritmos exactos de ramificación y acotación, existe un procedimiento conciso y preestablecido, independiente del problema abordado. En los métodos heurísticos esto no es así. Las técnicas e ideas aplicadas a la resolución de un problema son específicas de éste y

aunque, en general, pueden ser trasladadas a otros problemas, han de particularizarse en cada caso. Así pues, es necesario referirse a un problema concreto para estudiar con detalle los procedimientos heurísticos. Existen muchos métodos heurísticos de naturaleza muy diferente, por lo que es complicado dar una clasificación completa. Además, muchos de ellos han sido diseñados para un problema específico sin posibilidad de generalización o aplicación a otros problemas similares. El siguiente esquema trata de dar unas categorías amplias, no excluyentes, donde ubicar a las heurísticas más conocidas:

- **Métodos de descomposición**: el problema original se descompone en subproblemas más sencillos de resolver, teniendo en cuenta, aunque sea de manera general, que ambos pertenecen al mismo problema.
- **Métodos inductivos**: la idea de estos métodos es generalizar desde versiones pequeñas o más sencillas al caso completo. Las propiedades o técnicas identificadas para estos casos más fáciles de analizar, luego se pueden aplicar al problema completo.
- **Métodos de reducción**: consiste en identificar propiedades que se cumplen mayoritariamente por las buenas soluciones e introducirlas como restricciones del problema. El objetivo es restringir el espacio de soluciones simplificando el problema. El riesgo obvio es dejar fuera las soluciones óptimas del problema original.
- **Métodos constructivos**: consisten en construir literalmente paso a paso una solución del problema. Usualmente son métodos deterministas y suelen estar basados en la mejor elección para cada iteración.
- **Métodos de búsqueda local**: a diferencia de los métodos anteriores, los procedimientos de búsqueda o mejora local comienzan con una solución del problema y la mejoran progresivamente. El procedimiento realiza en cada paso un movimiento de una solución a otra con mejor valor. El método finaliza cuando, para una solución, no existe ninguna solución accesible que la mejore.

Si bien todos estos métodos han contribuido a ampliar el conocimiento para la resolución de problemas reales, los métodos constructivos y los de búsqueda local constituyen la base de los procedimientos metaheurísticos.

2.4. Metaheurísticas

Los procedimientos metaheurísticos se sitúan conceptualmente *por encima* de los heurísticos en el sentido que guían el diseño de éstos. Una metaheurística es

un conjunto de conceptos que pueden ser utilizados para definir métodos heurísticos aplicables a un extenso conjunto de problemas. Por lo tanto, se consideran heurísticas de un nivel más alto, cuyo uso es genérico, es decir, no son específicas para un tipo de problema, aunque deben instanciarse para cada uno de esos tipos.

En definitiva, se trata de estrategias para diseñar procedimientos heurísticos, y por ello, los tipos de metaheurísticas se establecen, en primer lugar, en función del tipo de procedimientos a los que hace referencia. Algunos de los tipos fundamentales son:

- Las *metaheurísticas de relajación*: se refieren a procedimientos de resolución de problemas que utilizan modificaciones del modelo original que lo hacen más fácil de resolver.

- Las *metaheurísticas 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.

- Las *metaheurísticas de búsqueda*: guían los procedimientos que usan transformaciones o movimientos para recorrer el espacio de soluciones alternativas y explotar las estructuras de entornos asociadas.

- Las *metaheurísticas evolutivas*: están enfocadas a los procedimientos basados en conjuntos de soluciones que evolucionan sobre el espacio de soluciones.

Algunas metaheurísticas surgen combinando metaheurísticas de distinto tipo. Otras se centran en el uso de algún tipo de recurso, computacional o formal, especial como las redes neuronales, los sistemas de hormigas o la satisfacción de restricciones y no se incluyen claramente en ninguno de los cuatro tipos anteriores. Entre las metaheurísticas más destacadas se encuentran:

- **Búsqueda local (*Local Search, LS*)**: también llamada mejora iterativa. Los movimientos se realizan sólo si se mejora la solución.

- **Recocido simulado (*Simulated Annealing, SA*)**: es una de las metaheurísticas más antigua que incorpora una estrategia explícita para impedir óptimos locales. Está basada en la física del calentamiento de metales. Se propone una similitud entre una buena estructura cristalina de metales y una buena estructura de soluciones para problemas de optimización combinatoria. Se trata de minimizar una función objetivo que representa la energía del sistema.

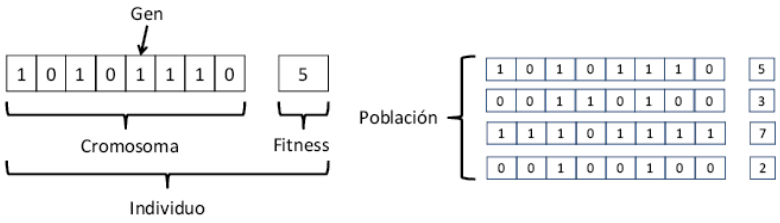

Figura 2.1: Ejemplo de individuo y población

- **Búsqueda tabú (*Tabu Search, TS*)**: utiliza una búsqueda local con memoria a corto plazo que le permite "escapar" de mínimos locales y evitar ciclos. La memoria de corto plazo esta representada por la *lista tabú* la cual registra las últimas soluciones"visitadas" e impide volver a ellas en los próximos movimientos. La *lista tabú* se actualiza normalmente en forma FIFO.

- **GRASP (*Greedy Randomized Adaptive Search Procedure*)**: combina procedimientos constructivos y de búsqueda local. Es un procedimiento iterativo en dos fases, una de construcción de la solución (heurística), y otra de mejora. La construcción de la solución se caracteriza por ser dinámica y aleatoria.

- **Computación evolutiva (*Evolutionary Computation, EC*)**: amplio espectro de técnicas heurísticas que funcionan emulando mecanismos de evolución natural. Trabaja sobre una población de individuos o conjunto de soluciones que evoluciona utilizando mecanismos de selección y construcción de nuevas soluciones candidatas mediante recombinación de características de las soluciones seleccionadas. Las etapas del mecanismo evolutivo son: evaluación, selección, aplicación de operadores evolutivos y reemplazo o recambio generacional. El proceso evolutivo decide en cada iteración qué individuos entrarán en la próxima población, mientras que los operadores evolutivos determinan el modo en que el algoritmo explora el espacio de búsqueda (Figura 2.1).

- **Colonias de hormigas (*Ant colony optimization*)**: heurísticas que se basan en imitar el comportamiento de las hormigas. Si bien cada hormiga tiene individualmente capacidades básicas, la colonia en conjunto logra un comportamiento inteligente. A pesar de ser insectos casi ciegos logran encontrar el camino más corto entre el hormiguero y una fuente de comida y regresar, en base a una comunicación entre ellas de origen químico que se potencia con la presencia simultánea de muchas hormigas. La comunicación es en base a feromonas, que dejan un "rastro" que sirve de referencia a otras hormigas.

- **Algoritmos meméticos (*Memetic Algoritms, MA*)**: son técnicas de optimización que combinan aspectos de otras metaheurísticas como son, el uso de poblaciones en los algoritmos evolutivos y la mejora local utilizada en búsqueda tabú o recocido simulado [141].

Tal y como se ha visto, algunas de estas metaheurísticas basan su funcionamiento en comportamientos de la naturaleza en el mundo real, por ejemplo los algoritmos evolutivos y las colonias de hormigas. Estas metaheurísticas cada vez son más usados para resolver problemas de optimización de diversas áreas. Concretamente, los algoritmos evolutivos (*Evolutionary Algorithm*s, EAs) se han mostrado prometedores para el cálculo de soluciones para problemas de optimización grandes y difíciles, aplicándose con éxito a una gran variedad de problemas reales [12]. De hecho, cuando se aplican a los MOPs, los EAs parecen actuar mejor que otras estrategias de búsqueda a ciegas. El uso de EAs para resolver problemas de esta naturaleza especial viene motivada principalmente porque son capaces de capturar múltiples soluciones óptimas de Pareto en una sola ejecución (lo cual es posible al estar basados en poblaciones) y de explorar similitudes entre las soluciones por recombinación. Los EAs específicamente diseñados para tratar con funciones con múltiples objetivos se conocen como algoritmos evolutivos multi-objetivo (*Multi-Objective Evolutionary Algorithm*s, MOEAs) [44]. Cuando se diseñan MOEAs se deben afrontar dos problemas principales [189]: cómo llevar a cabo la asignación de "fitness" o aptitud y la selección a fin de orientar la búsqueda hacia el conjunto óptimo de Pareto y cómo mantener la diversidad de la población para prevenir convergencias prematuras y conseguir un frente de compromiso bien distribuido. Intentando mantener estos objetivos en el diseño se han propuesto muchas alternativas [41]: *Vector Evaluated Genetic Algorithm* (VEGA) [161], *Niched Pareto Genetic Algorithm* (NPGA) [101], *Non-Dominated Sorting Genetic Algorithm* (NSGA) [55, 167], *Strength Pareto Evolutionary Algorithm* (SPEA) [191, 192], *Indicator-Based Evolutionary Algorithm* (IBEA) [190], etc.

2.5. Hiperheurísticas

Una hiperheurística es un método heurístico de búsqueda que trata de automatizar, normalmente incorporando técnicas de aprendizaje, el proceso de selección, combinación, generación o adaptación de heurísticas más simples (o componentes de esas heurísticas) para resolver eficientemente problemas de búsqueda. Una de las motivaciones para estudiar las hiperheurísticas es construir sistemas que manejen clases de problemas, en lugar de resolver sólo uno. La idea que hay detrás de estos enfoques es que, en el caso perfecto, una vez que se haya desarrollado un algoritmo

hiperheurístico, se puedan abordar varios dominios de problemas e instancias solamente reemplazando las heurísticas de bajo nivel. Por lo tanto, el propósito del uso de una hiperheurística es aumentar el nivel de generalidad al que trabajan la mayoría de las (meta)-heurísticas. Dado que la principal motivación de las hiperheurísticas es diseñar estrategias independientes de los problemas, una hiperheurística no se ocupa de resolver un problema dado directamente como es el caso de la mayoría de las heurísticas. De hecho, la búsqueda se realiza en el espacio de búsqueda de una (meta)-heurística en lugar de en el espacio de soluciones potenciales del problema. Las hiperheurísticas resuelven problemas indirectamente recomendando qué método de resolución aplicar en cada etapa del proceso. En general, el objetivo de aumentar el nivel de generalidad se consigue a expensas de reducir la calidad de la solución (siendo aún aceptable) cuando se compara con métodos (meta)-heurísticos hechos a medida. Su fin no es vencer a técnicas construidas a la medida, sino demostrar que son competitivas y capaces de generar resultados de calidad. Además, se ha comprobado que resultan especialmente útiles en problemas de optimización poco conocidos.

Las hiperheurísticas se pueden clasificar en dos grupos en función de las características de las metaheurísticas de bajo nivel que se utilicen [35]: las que operan con técnicas constructivas y las que lo hacen con técnicas de mejora. Las técnicas constructivas se usan para construir soluciones partiendo de cero. En cada paso, determinan una subparte de la solución [170]. Las técnicas de mejora son aproximaciones iterativas que toman una solución inicial, y la modifican con el propósito de mejorar el valor del objetivo [119]. Algunas hiperheurísticas han sido diseñadas para operar específicamente con un tipo de metaheurísticas de bajo nivel, mientras que otras, pueden usar ambos tipos, constructivas y de mejora.

Aunque el término hiperheurística se introdujo en los primeros años del siglo XXI para describir "heurísticas que eligen heurísticas", la idea de automatizar el proceso de diseño de una heurística, no es nueva. Se remonta a los años 60. Algunas notas históricas y una breve revisión de las primeras aproximaciones se pueden encontrar en [33] y [31] respectivamente.

Las investigaciones más recientes tienden hacia aproximaciones hiperheurísticas para generar automáticamente nuevas heurísticas apropiadas para un problema o clase de problema dado, mediante combinación [32].

2.6. Paralelización

En el procesamiento en secuencial los programas consisten en un conjunto de instrucciones que se ejecutan una detrás de otra, de manera que sólo se puede estar

ejecutando una instrucción en cualquier instante de tiempo. Por el contrario, las técnicas de computación paralela están basadas en el uso simultáneo de múltiples recursos computacionales para resolver el problema dado [19]. Es decir, el programa se tiene que poder dividir en una serie de partes diferentes e independientes que se puedan resolver concurrentemente.

Las razones principales para utilizar el paralelismo en el diseño de software son obtener un alto rendimiento o una mayor velocidad al ejecutar un programa. En el caso de los diferentes métodos de resolución para problemas de optimización, introducir paralelizaciones en los algoritmos que los implementan supone una serie de ventajas sobre la ejecución secuencial de los mismos, las cuales dependen del propósito de la paralelización. El primer propósito puede ser obtener la misma solución que con el algoritmo secuencial en menos tiempo, gracias a la división de trabajo entre los diferentes procesadores. Esto es común cuando se trata con métodos exactos, ya que deben explorar todo el espacio de búsqueda para asegurar el encuentro de la solución óptima, y, por lo tanto, requieren tiempos considerables. Sin embargo, cuando se trata en general con métodos aproximados puede darse el caso de que realmente el objetivo de paralelizar no sea disminuir el tiempo de ejecución, sino obtener una mejor solución en el mismo tiempo empleado por el algoritmo secuencial. En cualquier caso, se consigue una mejora con respecto al método aplicado secuencialmente.

Por otra parte, cuando se aplica paralelismo, la manera en la que se realiza el trabajo concurrentemente depende del modelo de programación que se esté aplicando. Los modelos de programación paralela existen como una abstracción sobre el hardware y las arquitecturas de memoria. Aunque estos modelos están asociados normalmente con un cierto tipo de máquina o arquitectura de memoria, no son específicas ni están restringidas a una en particular. Algunos ejemplos son los *modelos de memoria compartida*, donde las tareas comparten un espacio de direcciones común en el que escriben y leen asíncronamente; los *modelos de paso de mensajes*, donde durante las ejecuciones las tareas intercambian datos a través del envío y recepción de mensajes; los *modelos de hilos*, donde un solo procesador puede tener múltiples ejecuciones concurrentes; y los *modelos híbridos*, que mezclan diversos modelos. Además, cuando se utilizan métodos de resolución basados en poblaciones, como es el caso de algunas metaheurísticas (por ejemplo, los algoritmos evolutivos), la paralelización se suele realizar utilizando los denominados *modelos basados en islas*. Así se divide la población original en un conjunto de sub-poblaciones independientes. Cada sub-población está asociada a una isla, y sobre cada isla se ejecuta una configuración del algoritmo de forma independiente durante un cierto tiempo. Una configuración está constituida por un algoritmo de optimización con sus respectivos parámetros. Generalmente, cada procesador constituye una isla, de forma

que cada isla evoluciona en paralelo de forma independiente. Sin embargo, dado que los esquemas colaborativos suelen alcanzar mejores resultados, se incluye una fase de migración que permite el intercambio de individuos entre islas. Este comportamiento colaborativo añade al esquema basado en islas la posibilidad de obtener un mejor comportamiento. Existen cuatro modelos basados en islas diferentes [44]: todas las islas ejecutan la misma configuración (*homogéneo*), todas las islas ejecutan una configuración diferente (*heterogéneo*), cada isla evalúa un subconjunto diferente de funciones objetivo y cada isla representa una región distinta en los dominios del fenotipo o del genotipo. Además, el modelo heterogéneo puede agregar propiedades auto-adaptativas. Estas propiedades auto-adaptativas permiten conceder dinámicamente más recursos computacionales a los algoritmos más prometedores. Es decir, aplicando una hiperheurística, se permite el cambio automático y dinámico de MOEAs y/o parámetros que se ejecutan en cada una de las islas [119].

2.7. Herramientas de apoyo para la optimización

En muchas ocasiones, cuando se está tratando un problema de optimización concreto, se desea obtener soluciones utilizando diferentes métodos de resolución y configuraciones de los mismos, con el propósito de comprobar cuál se ajusta mejor a las necesidades, demandas o recursos disponibles. En estos casos, aunque el usuario tenga un amplio conocimiento del problema, puede no conocer en profundidad o tener sólo un conocimiento general de cada uno de los métodos para resolverlo. Para proporcionar ayuda a estos usuarios y reducir el esfuerzo implicado en este proceso, existe una serie de herramientas específicas que dan soporte a diferentes técnicas de resolución. Este tipo de herramientas puede venir dado en forma de librerías, entornos, o esqueletos algorítmicos. Tal software aporta una ventaja importante en comparación con una implementación directa desde cero, no sólo en términos de reusabilidad, sino también en lo relativo a la metodología y claridad de conceptos. El mayor inconveniente con la mayoría de estas herramientas es la complejidad de uso, principalmente por el lenguaje de programación o por su estructura lógica interna. Puesto que además los usuarios requieren herramientas que no sólo sean fáciles y flexibles, sino también eficientes, algunas propuestas diseñadas incorporan esquemas paralelos con el objetivo de mejorar la eficiencia.

Hay disponibles muchos entornos y esqueletos algorítmicos para diferentes tipos de técnicas algorítmicas: algoritmos de búsqueda local [36, 79], ramificación y acotación y búsquedas generales en árbol [67, 106, 114, 115, 172], o computación evolutiva [10, 74, 137]. En [176] se presenta una revisión bibliográfica al respecto. La mayoría de estas herramientas están diseñadas para tratar con problemas de

optimización mono-objetivo. Algunos de ellas son:

- *ABACUS* [106]: el nombre proviene de "*A Branch-And-CUt System*". Se trata de una herramienta software escrita en C++ que proporciona un entorno para la implementación de algoritmos de ramificación y acotación usando programación lineal.

- *DREAM* [10]: son las siglas de "*Distributed Resource Evolutionary Algorithm Machine*". es un proyecto que pretende desarrollar un entorno completamente distribuido para algoritmos evolutivos. Implica un motor de red, librerías y diferentes interfaces de usuario, todo esto en Java.

- *MALLBA* [1, 2, 3]: diseñado en C++, proporciona un entorno más general para el uso de metaheurísticas paralelas y aproximaciones algorítmicas estándar. Se trata de una librería donde cada esqueleto implementa un método de optimización y proporciona tres implementaciones diferentes para él: una secuencial, otra paralela para redes de área local y una paralela para redes de área extensa.

- *Templar* [105]: entorno escrito en C++, permite crear objetos e integrarlos en otras aplicaciones en computación distribuida, ya que está orientado específicamente a la hibridación y la cooperación. Implementa métodos tales como la búsqueda tabú, algoritmos genétivos y también algoritmos exactos.

No obstante, en la última década también se han propuesto varios entornos específicos basados en metaheurísticas para resolver problemas de optimización multi-objetivo:

- *PISA* [20]: son las siglas de "*A Platform and Programming Language Independent Interface for Search Algorithms*". Es una de las plataformas más conocidas para la optimización multi-objetivo. El principio subyacente de PISA es separar la parte específica de los algoritmos de la parte específica de las aplicaciones. La comunicación entre ambas parte se lleva a cabo usando mecanismos de compartición de ficheros, de tal modo que la ejecución de las aplicaciones desarrolladas usando PISA tienen un tiempo de penalización asociado.

- *Open BEAGLE* [75]: es una herramienta escrita en C que proporciona un entorno software de alto nivel para cualquier tipo de computación evolutiva. Soporta programación genética basada en árboles; algoritmos genéticos de cadena de bits, vector de números enteros y vector de números reales; y estrategias evolutivas.

- *jMetal* [62]: son las siglas de "*Metaheuristic Algorithms in Java*". Se trata de un entorno orientado a objetos basado en Java para el desarrollo, experimentación y estudio de metaheurísticas para resolver problemas de optimización multi-objetivo. Proporciona un conjunto de clases que se pueden usar como bloques de construcción de metaheurísticas multi-objetivo, con la ventaja de la reutilización de código, lo cual facilita el desarrollo de nuevos algoritmos multi-objetivos.

- *ParadisEO* [124]: es un entorno basado en plantillas, compatible con ANSI-C++, dedicado al diseño flexible de metaheurísticas, incluyendo metaheurísticas basadas en la solución (búsqueda local, recocido simulado, búsqueda local iterativa, etc.) y metaherísticas basadas en población (algoritmos genéticos, estrategia evolutiva, etc.). Además, proporciona herramientas para el diseño de metaheurísticas para la optimización multi-objetivo.

- METCO [119]: son las siglas de "*Metaheuristics-based Extensible Tool for Cooperative Optimization*". Se trata de un entorno paralelo para optimización multi-objetivo escrito en C++ y basado en plugins. Su objetivo es minimizar el esfuerzo requerido a los usuarios finales para incorporar sus propios problemas y algoritmos evolutivos. Se pueden especificar una gran variedad de configuraciones para adaptar el comportamiento del software a muchos modelos paralelos diferentes.

Estos entornos para optimización multi-objetivo proporcionan implementaciones de algunos de los MOEAs más conocidos y también soporte para la implementación de nuevos algoritmos multi-objetivo. Sin embargo, sólo ParadisEO y METCO soportan esquemas paralelos. En particular, ParadisEO ofrece soporte para las implementaciones de modelos paralelos basados en islas [124]. Para realizar una ejecución basada en islas con ParadisEO, los usuarios deben especificar en el código la parte que va a cada isla en la ejecución paralela. Para probar los diferentes modelos paralelos basados en islas, los usuarios deben implementar un código diferente para cada modelo, lo cual complica el uso de la herramienta cuando se están realizando comparaciones de comportamiento entre grandes conjuntos de modelos. Además, los modelos basados en islas no son fáciles de personalizar en ParadisEO. El resultado es que los usuarios finales tienen que realizar un esfuerzo considerable para incorporar nuevas ideas en los modelos paralelos, dificultando el uso de los algoritmos evolutivos multi-objetivo paralelos (*Parallel Multi-Objective Evolutionary Algorithm*s, PMOEAs) para los usuarios que no son expertos. Desde nuestro conocimiento, la interfaz proporcionada no permite ejecuciones donde los algoritmos sean dinámicamente asignados a las islas, por lo que para incorporar los principios de las

hiperheurísticas, los usuarios deben modificar la estructura interna de la herramienta.

Sin embargo, METCO sí permite realizar muchas personalizaciones de los modelos basados en islas, de manera que los usuarios expertos puedan adaptarla para obtener incluso mejores optimizadores. Aunque incorpora una selección de los MOEAs más conocidos en la literatura, se pueden integrar otros a través del uso de plugins [121, 123]. Para personalizar los archivos de configuración y desarrollar nuevos plugins, los usuarios pueden adaptar el entorno a sus propios requisitos. El software ha sido diseñado de tal manera que el usuario pueda especificar fácilmente los requisitos del problema y personalizar la configuración de los MOEAs que participarán en la solución del problema. Por ejemplo, a la hora de introducir un nuevo problema es tan sencillo como especificar: cómo se va a representar un individuo, cómo se va a generar la población inicial, qué operadores de mutación y cruce se van a utilizar, y cómo se van a evaluar los objetivos. Incluso, las configuraciones paralelas se pueden especificar simplemente a través del uso de un archivo de configuración, evitando la necesidad de entender la parte interna de la herramienta o el funcionamiento de las estrategias de optimización. Además, incluye un nuevo modelo paralelo basado en islas, el modelo adaptativo [122], cuyo objetivo final es aumentar el nivel de generalidad al que trabajan la mayoría de algoritmos evolutivos. Esto le permite dirigirse a una gran variedad de problemas, ya que las debilidades de un algoritmos se pueden compensar por los puntos fuertes de otro. El esquema intenta detectar y asignar más recursos a los algoritmos y operadores más adecuados. Este modelo adaptativo es un algoritmo híbrido que combina un esquema paralelo basado en islas con una aproximación hiperheurística [118].

2.8. Métricas de rendimiento

Las métricas o indicadores de rendimiento se utilizan para evaluar el comportamiento de un algoritmo. En el caso de los algoritmos mono-objetivo, para contrastar la calidad de dos algoritmos secuenciales diferentes y saber cuál de los dos se comporta mejor, es suficiente comparar los valores proporcionados por cada uno de ellos. Es decir, por comparación directa de los resultados que proporciona cada algoritmo, tanto en relación al tiempo y recursos que necesitan como en la solución a la que llegan, en general se debería saber cuál se ajusta más a nuestras necesidades. Así, para los algoritmos exactos mono-objetivo se optará por el que proporcione la solución más rápido o utilice menos recursos, mientras que para los algoritmos aproximados también habrá que tener en cuenta el valor de la solución proporcionada por los mismos.

Sin embargo, cuando se trata de algoritmos paralelos la medida del rendimiento se complica más. En primer lugar, se pueden tener distintos propósitos al paralelizar: tratar de obtener el mismo resultado que el algoritmo secuencial en un tiempo menor, o bien, en el mismo tiempo tratar de obtener una solución mejor que la obtenida por el secuencial. Cuando se dispone de un algoritmo exacto el propósito siempre será el del primer caso, mientras que la finalidad más común cuando se dispone de un algoritmo aproximado es la segunda opción. De cualquier modo, para cualquier problema dado, el algoritmo paralelo óptimo puede ser radicalmente diferente del algoritmo secuencial óptimo. El objetivo cuando se diseña cualquier algoritmo paralelo es dividir la tarea global en sub-tareas independientes que requieran de poca sincronización y comunicación. En general, los algoritmos paralelos eficientes se obtienen mediante el uso eficiente de los recursos para procesar y la maximización de la relación cómputo-comunicación. Con el fin de medir el rendimiento o la eficiencia de los programas paralelos se han propuesto muchas métricas [177]. Una de las más usadas y útiles es la ***aceleración***. El término *aceleración* se define como el cociente del tiempo requerido para completar el proceso con el algoritmo secuencial más rápido usando un procesador entre el tiempo requerido para completar el mismo proceso con el algoritmo paralelo utilizando p procesadores. El concepto está directamente relacionado con la *Ley de Amdahl*. La *Ley de Amdahl* establece que la aceleración que se puede obtener de un programa se determina mediante la fracción del código del programa (P) que se puede paralelizar:

$$aceleracion = \frac{1}{1-P} \tag{2.1}$$

Por lo tanto, si no hay secciones del código que se puedan paralelizar ($P = 0$), la *aceleración* que se obtiene es 1, es decir, no es posible acelerar la ejecución del programa. Si todo el código se puede paralelizar ($P = 1$), entonces, teóricamente, la *aceleración* es infinita. Si el 50 % del código se puede paralelizar ($P = 0.5$), la *aceleración* máxima que se puede obtener es 2, lo cual significa que la ejecución será dos veces más rápida. La *aceleración* también depende del número de procesadores (p) que participan en la ejecución, así como de la fracción secuencial del código (S). Considerando ambos factores, la *aceleración* máxima que se puede obtener con un computador de p procesadores se define como:

$$aceleracion \leq \frac{1}{S + \frac{P}{p}} = \frac{1}{S + \frac{1-S}{p}} \tag{2.2}$$

Esta fórmula claramente muestra los límites que afectan a la escalabilidad en los programas paralelos. El aumento del tamaño de la parte secuencial del problema

provoca que la *aceleración* se sature, por lo que cuando se diseñan algoritmos paralelos eficientes las operaciones secuenciales se deben reducir, minimizando así el tiempo de inactividad de cada procesador.

Por otra parte, cuando se trata con problemas de optimización multi-objetivo, distinguir qué algoritmo tiene mejor comportamiento, tanto en secuencial como en paralelo, no es tan sencillo como cuando se tratan problemas mono-objetivo. En este caso se dispone de conjuntos de soluciones que se deben comparar, por lo que se necesitan métricas que simplifiquen las comparativas aportando valores de calidad únicos a frentes de soluciones, que puedan, en cierto modo, globalizar la calidad de cada conjunto de soluciones. Cuando se diseña una buena métrica para problemas multi-objetivo se suelen tener en cuenta tres cuestiones:

- Minimizar la distancia del frente de Pareto producido por nuestro algoritmo con respecto al verdadero frente de Pareto (suponiendo que se conoce su ubicación).

- Maximizar la distribución de las soluciones encontradas, de forma que podamos tener una distribución de las soluciones no dominadas tan suave y uniforme como sea posible.

- Maximizar la cantidad de elementos del conjunto de óptimos de Pareto encontrado.

Existen diversas propuestas de métricas en la literatura (*tasa de error, distancia generacional, dispersión, espacio cubierto, cobertura, hipervolumen, indicadores* ϵ*, etc.*), pero ninguna de ellas captura realmente en un sólo valor numérico los tres elementos anteriores. De hecho, intentar hacerlo puede ser infructuoso ya que las tres cuestiones se refieren a aspectos del rendimiento muy distintos. Por lo tanto, su fusión en un único valor puede dar pie a una métrica que no indique correctamente el comportamiento de un algoritmo multi-objetivo. Lo más recomendable es usar diferentes métricas para evaluar los distintos aspectos del rendimiento de un algoritmo. Las métricas de rendimiento para los problemas multi-objetivo, utilizadas en los estudios que se presentan en los siguientes capítulos han sido el ***indicador*** ϵ y el ***hipervolumen***.

El *indicador* ϵ es un indicador unario propuesto por Zitzler [194] y hace uso directo de los principios de la dominancia de Pareto, consiguiendo que su entendimiento sea muy intuitivo. Considérense dos conjuntos aproximados A y B, donde A domina a B. El *indicador* ϵ es una medida de la menor distancia necesaria para traducir cada punto de B de manera que domine a A. Si el conjunto A se escoge como conjunto de referencia, tal que domina a los conjuntos B y C, entonces B y C se pueden

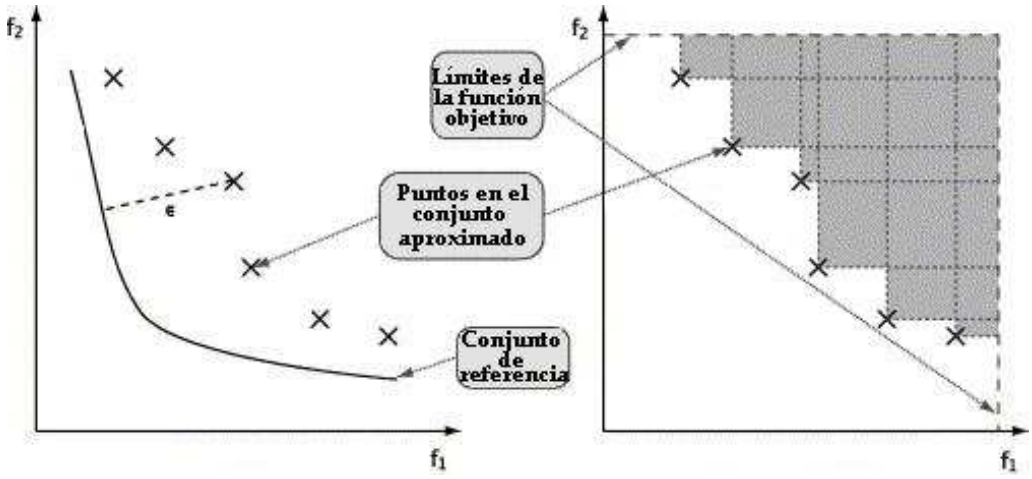

Figura 2.2: Indicador ϵ e hipervolumen

comparar directamente sobre la base del *indicador* ϵ con respecto al conjunto de referencia (Figura 2.2).

Por otra parte el *hipervolumen* fue propuesto por Zitzler and Thiele [193] y sus propiedades se discuten en profundidad en [194]. El caso bi-dimensional se puede visualizar y entender fácilmente. Cada punto en el conjunto de aproximación forma un rectángulo con un área determinada con respecto a un punto de referencia que está fuera de los límites del conjunto de aproximación. El *hipervolumen* es el área resultante de la unión de todos estos rectángulos. Este concepto se puede extender sin dificultades a dimensiones mayores. De hecho, el *hipervolumen* mide el hiperespacio encerrado por el conjunto de aproximación (Figura 2.2).

Con los problemas de optimización multi-objetivo, además del uso de métricas o indicadores de rendimiento, resulta de utilidad visualizar el dibujo de los resultados de las ejecuciones ya que [110]:

- Muchos indicadores de rendimiento en optimización multi-objetivo no son capaces de distinguir relaciones de orden entre los conjuntos de aproximación de, incluso, un solo par de ejecuciones.

- Las personas encargadas de tomar las decisiones podrían tener preferencias sobre ciertas regiones del frente de Pareto, que no pueden expresar generalmente antes de la optimización, pero que se usan en última instancia para juzgar la calidad de los conjuntos aproximados.

- Algunos indicadores de rendimiento no expresan adecuadamente la cantidad

por la que un conjunto de aproximación debería ser valorado mejor que otro.

- Observando la forma del conjunto aproximado se puede conseguir información sobre los puntos fuertes y débiles de un optimizador, o sobre cómo está funcionando.
- Los métodos de visualización pueden proporcionar una "prueba de cordura" para validar cualquier indicador de rendimiento que se use.
- Cuando se conoce el verdadero frente de Pareto, observar la distancia hasta él y la cobertura que se ha conseguido también proporciona un complemento añadido a los indicadores usados.

Por estas razones, complementar el uso de métricas de rendimiento con el dibujo de los conjuntos de aproximación es una práctica común. Sin embargo, cuando se ejecutan uno o más algoritmos varias veces, el dibujo de los conjuntos de aproximación mostrando todos los puntos es confuso y puede resultar incluso engañoso. En particular, es difícil separar los conjuntos individuales y entender la distribución y la extensión de los diferentes conjuntos entre múltiples ejecuciones. Esto ocurre sobre todo cuando se tiene el dibujo de, por ejemplo, treinta ejecuciones de uno o más algoritmos. Por ello, se utilizan las ***superficies de alcance*** (*attainment surfaces* [73, 110]), como una alternativa para visualizar, de una forma clara, los resultados obtenidos en un cierto número de ejecuciones. Una *superficie de alcance* es el límite representado por la familia de objetivos más cercanos al óptimo que ha sido alcanzada durante una ejecución del optimizador (Figura 2.3). Estas superficies facilitan la identificación de las zonas no cubiertas en el espacio de objetivos. Otra de las ventajas de las superficies de alcance se hace evidente cuando queremos visualizar de forma simultánea el resultado de las ejecuciones de distintos optimizadores (Figura 2.4). En este caso, en lugar de dibujar un frente de puntos para cada ejecución, se puede representar la superficie de alcance s. Suponiendo que se han realizado n ejecuciones diferentes, la superficie de alcance s domina débilmente a las superficies de alcance $s+1, s+2, \ldots n$. La ventaja está en que estas superficies son más fáciles de interpretar que los frentes que resultan de cada ejecución, ya que las superficies de alcance nunca se cruzan entre ellas.

Dado que en los estudios de los problemas de optimización multi-objetivo que se presentan en los próximos capítulos, se han utilizado algoritmos evolutivos para su resolución, es necesario destacar que, si en lugar de fijar como condición de parada una determinada calidad de las soluciones, se fija un número de evaluaciones, hay que tener en cuenta que se debe analizar qué es lo que ocurre realmente con el tiempo y con la calidad final de las soluciones obtenidas.

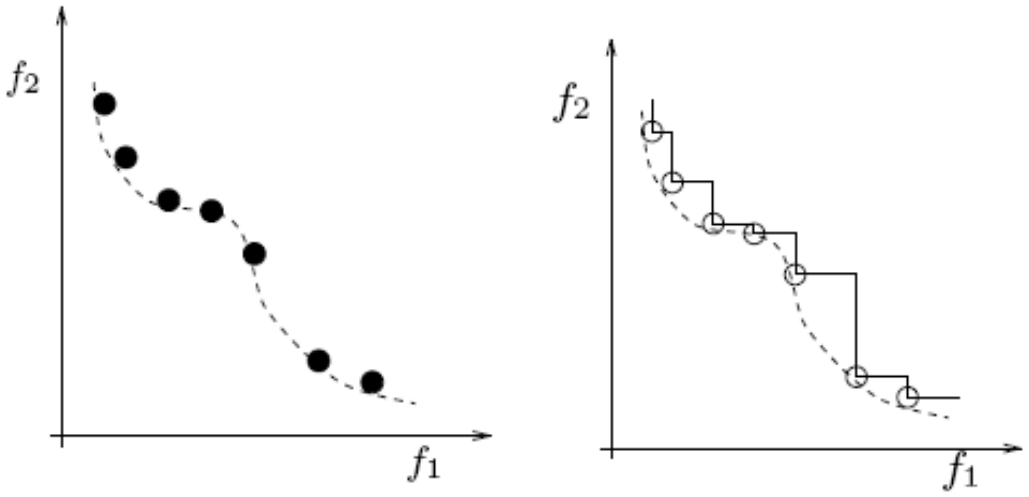

Figura 2.3: Frente de Pareto y su familia de objetivos más cercanos al óptimo

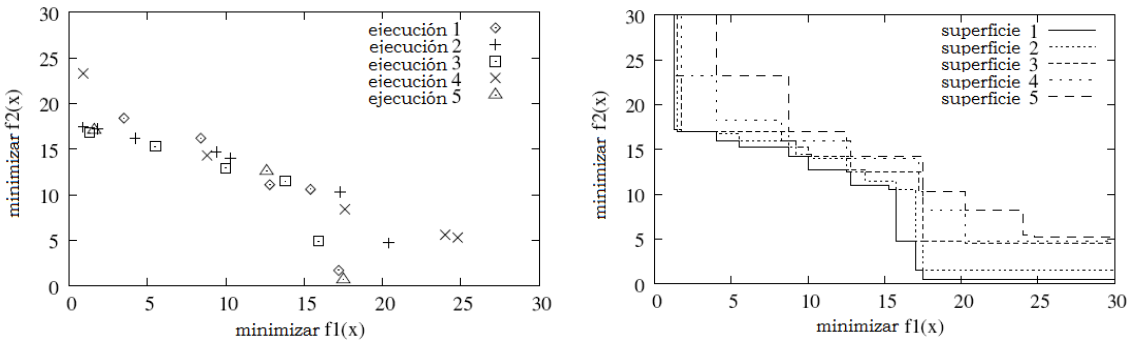

Figura 2.4: Múltiples frentes de Pareto y sus superficies de alcance

Parte II

Problemas

Capítulo

3

2D Cutting Stock Problem

Este capítulo presenta el *Two-Dimensional Cutting Stock Problem*, 2DCSP, un problema que surge en muchas industrias reales de producción, donde hojas grandes de material (cristal, tela, papel, metal, plástico, madera, etc.) se deben cortar en piezas más pequeñas. Las características de estas piezas demandadas pueden variar drásticamente dependiendo de las necesidades de los clientes. En la mayoría de los casos, incluso si el material se corta cuidadosamente, el desperdicio resultante puede ser superior al 10 %. Dependiendo del tipo de material, los gastos asociados con el desperdicio pueden ser inaceptables, con un impacto económico negativo para el crecimiento de la empresa. Sin embargo, la experiencia demuestra que el desperdicio de material se puede reducir al 1 % o menos si se mejora el modo de realizar los cortes para obtener las piezas demandadas. Debido a que el empleo de los cortes óptimos es crucial, el problema se ha estudiado en profundidad y como consecuencia se han desarrollado muchas aproximaciones. Por lo tanto, dada la importancia de optimizar el uso de la materia prima, este capítulo se centra en el estudio de diversas técnicas de resolución para este problema. Se presentan mejoras para un algoritmo exacto y paralelizaciones del mismo, que hacen disminuir los tiempos empleados en obtener la solución. Por otra parte se afronta una novedosa visión multi-objetivo del problema no conocida en la literatura relacionada, mediante el uso de MOEAs. Esta visión multi-objetivo, además de optimizar el uso del material, trata de minimizar el número de cortes necesarios, con el consecuente menor desgaste de las cuchillas y ahorro de tiempo a la hora de cortar las piezas.

3.1. Definición del problema

El 2DCSP es una de las variantes más interesantes de los problemas de corte y empaquetado y consiste en el corte de un rectángulo S de dimensiones $L \times W$ en un conjunto de pequeñas piezas rectangulares (Figura 3.1). En la variante aquí analizada, el proceso de corte debe realizarse usando cortes ortogonales de tipo guillotina, es decir, las piezas se han de obtener usando exclusivamente cortes ortogonales de guillotina. Esto significa que cualquier corte debe poder realizarse desde un lado del rectángulo hasta el opuesto, yendo paralelamente a los otros dos bordes. La producción de cortes no guillotinables puede implicar un funcionamiento más complejo de la maquinaria. Por esta razón, y para abarcar un rango más amplio de máquinas industriales, el trabajo se ha centrado en la formulación guillotinable del problema. De este modo, todas las soluciones se pueden cortar de ambos modos, guillotina y no guillotina, asumiendo además que normalmente todos los cortes son infinitamente finos. El ejemplo de aplicación real más común donde normalmente se requiere cortes de guillotina es la industria del papel.

Después de realizar los cortes, en este problema los rectángulos resultantes deben pertenecer a uno de los conjuntos de piezas dadas $\mathcal{D} = \{T_1 \dots T_n\}$, donde el i-ésimo tipo T_i es de dimensiones $l_i \times w_i$. Todas las piezas tienen una orientación fija, es decir, la pieza de largo l y ancho w es diferente de la pieza de largo w y ancho l (cuando $l \neq w$). Asociado con cada tipo T_i hay un beneficio p_i y una demanda b_i. El beneficio o valor de una pieza dada no tiene por qué ser proporcional a su área. La demanda establece un número máximo de piezas del tipo T_i que pueden aparecer en un patrón de corte factible (existe también una formulación sin restricciones del problema en la que no hay limitación con respecto al número de piezas disponibles de cada tipo). Bajo esta consideración, el objetivo es encontrar un patrón de corte factible, es decir, que no supere las dimensiones del material y donde las piezas no se

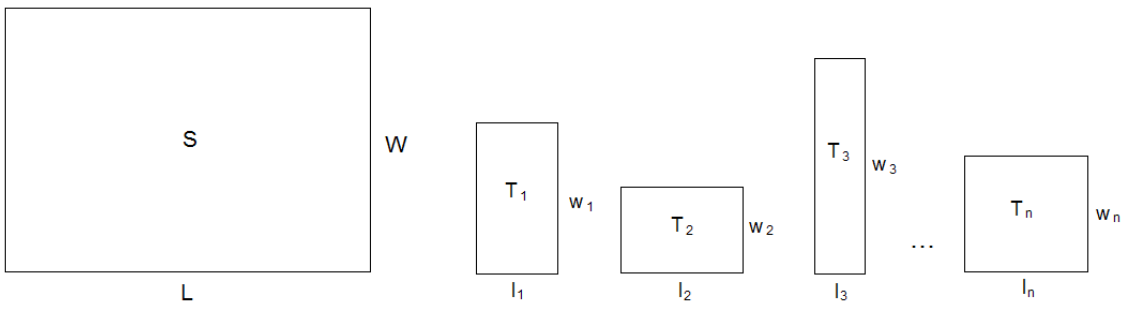

Figura 3.1: Superficie disponible y piezas demandadas

Nombre de la Propiedad	Valor de la Propiedad
Dimensionalidad	Dos dimensiones (2D)
Medición de cantidad	Discreta
Forma de las figuras	Regular y orientación fija
Variedad de piezas pequeñas	Variedad heterogénea
Variedad de piezas grandes	Un objeto grande
Disponibilidad de piezas	No se establece orden entre las piezas aunque piezas diferentes están asociados con beneficios o valores diferentes
Restricciones de los patrones	Ortogonales y guillotinables
Tipo de asignación	Maximización de la salida
Objetivos	Maximizar el beneficio total obtenido del objeto grande

Tabla 3.1: Propiedades del 2DCSP

solapen, con x_i piezas de tipo T_i de modo que se maximice el beneficio total obtenido de la materia prima disponible:

$$maximizar \sum_{i=1}^{n} x_i p_i \text{ sujeto a } x_i \leq b_i \text{ y } x_i \in \mathbb{N} \tag{3.1}$$

Las características principales del 2DCSP se resumen en la Tabla 3.1, de acuerdo con la tipología general que se ha expuesto para los problemas de corte y empaquetado en la Sección 1.2. Teniendo en cuenta este conjunto de propiedades y siguiendo las tipologías más recientes en corte y empaquetado [182] o la clasificación dada en la Sección 1.3, el problema se denotaría como un *Placement Problem* o *Knapsack Problem*. Normalmente, en el llamado *Placement Problem*, la variedad de piezas pequeñas es débilmente heterogénea, mientras que en el *Knapsack Problem* se consideran fuertemente heterogéneas. Sin embargo, en el problema que aquí se trata, no se fijan restricciones en cuanto al número n de rectángulos diferentes ni en la cantidad de piezas disponibles de cada tipo. Además, en la clasificación sugerida el término "*Cutting Stock Problem*" está asociado con un problema completamente diferente donde se debe colocar una variedad de piezas pequeñas débilmente heterogéneas en una selección de objetos grandes de mínimo valor (minimización de la entrada). Sin embargo, en la literatura se ha usado comúnmente este término para identificar al tipo de problema que aquí se aborda.

Algunas aplicaciones reales de corte y empaquetado, pueden tener en cuenta otros criterios de optimización. En algunos campos de la industria, la materia prima es muy barata o se puede reciclar fácilmente, por lo que en tales casos, un criterio más importante para la generación de los patrones podría ser la velocidad a la que

se obtienen las piezas, maximizando el uso de la maquinaria de corte. La velocidad de corte está específicamente limitada por las características de la maquinaria disponible, pero, en general, viene determinada por el número de cortes implicados en el patrón de corte. Además, el número de cortes requerido en el proceso de corte, o proceso de obtención de las piezas pequeñas, es crucial para la vida de la maquinaria industrial. Una metodología de optimización debería tener en consideración el número de cortes, ya que es un aspecto importante para determinar el coste y la eficiencia del proceso de producción. Por lo tanto, en la Sección 3.6, se tomará el número de cortes como un segundo objetivo de optimización. De este modo, el problema se podrá plantear como un problema de optimización multi-objetivo, intentando optimizar la distribución de las piezas rectangulares de modo que se maximice el beneficio al mismo tiempo que el número de cortes necesarios para obtener las piezas demandadas. Esta formulación del problema se denotará como *Multi-Objective Cutting Stock Problem*, MOCSP.

En la mayoría de los problemas de optimización del mundo real, los objetivos implicados suelen estar en conflicto entre ellos. Es decir, por regla general no existe una solución única que optimice simultáneamente todos los objetivos. En lugar de un solo óptimo, existe más bien un conjunto de soluciones alternativas, de entre las cuales una persona encargada de tomar las decisiones debe seleccionar una solución de compromiso.

En el caso del MOCSP, podría parecer, a priori, que las soluciones mejores contendrán más piezas y, por tanto, más cortes (Figura 3.2). Pero no siempre es así, porque hay piezas que dan mayor beneficio ocupando una mayor parte de la superficie disponible, e implicando, así un menor número de cortes que el que se necesitaría si se rellenara la superficie con más piezas (Figura 3.3).

Por lo tanto, se puede decir que en general, para este problema, los objetivos se encuentran, al menos, en cierto grado de conflicto. De este modo, se obtendrá un conjunto de soluciones no dominadas en lugar de una única solución óptima.

3.2. Trabajos relacionados

En la literatura, existen muchas variedades diferentes del 2DCSP. Se pueden encontrar trabajos donde se contemplan restricciones en el número mínimo y máximo de veces que debe aparecer una pieza en el patrón de corte [95], o restricciones en el número de etapas para la realización de los cortes [164], y otros en los que incluso se considera que la superficie de material tiene zonas defectuosas [145]. En general, la mayoría de las contribuciones que tratan con el 2DCSP se centran en el problema con la orientación de las piezas fija. Esto refleja el gran número de contextos prácticos

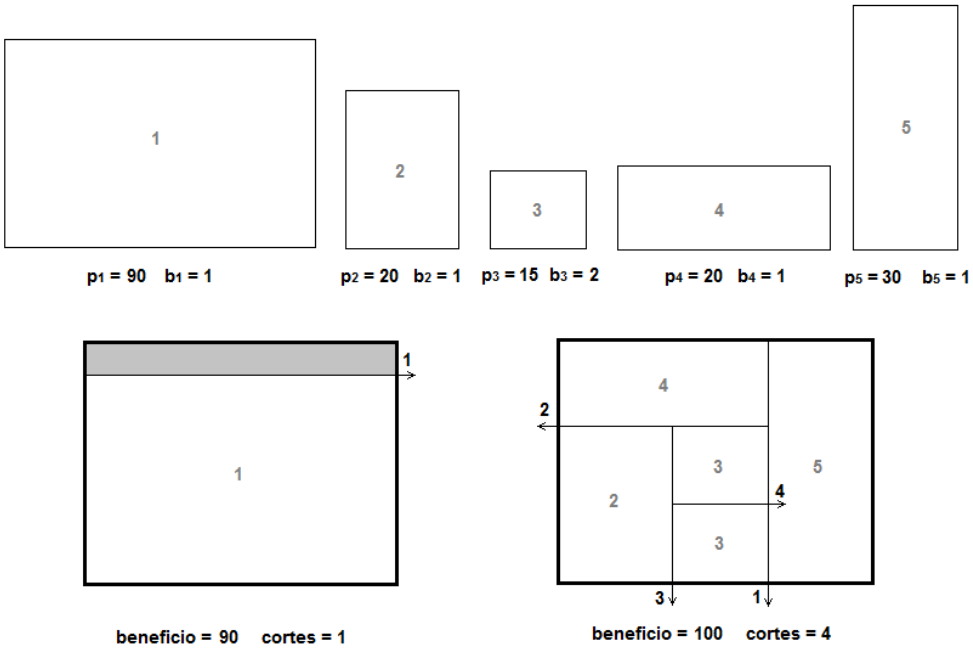

Figura 3.2: Ejemplos de objetivos en conflictivo

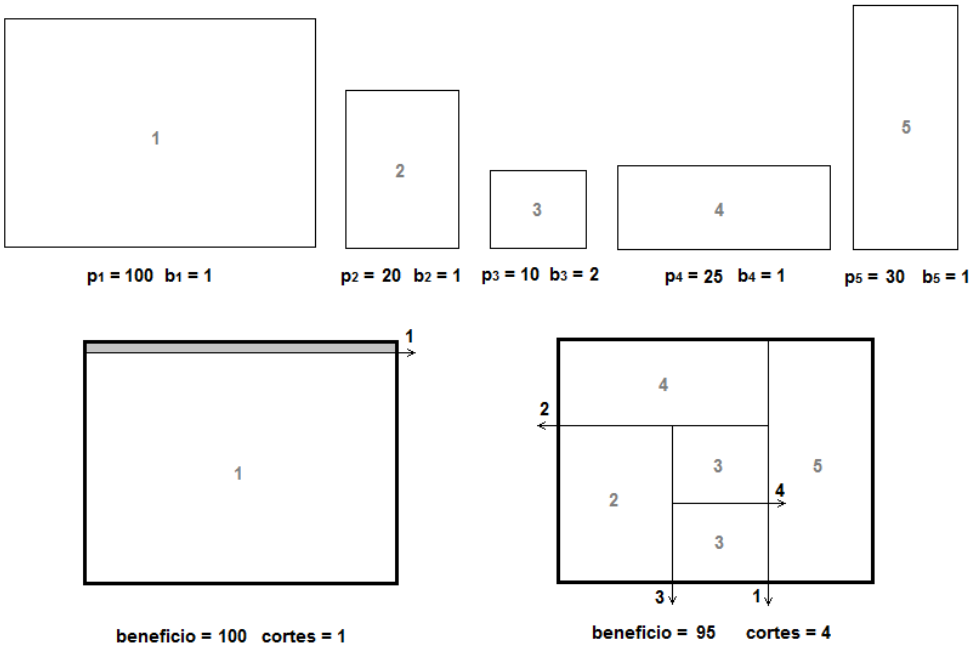

Figura 3.3: Ejemplos de objetivos sin conflicto

en los que se ha de tener en cuenta esta restricción, tales como el corte de un material ondulado o decorado, las páginas de los periódicos, etc. Otras variantes, como las que contemplan la rotación de 90 grados de las piezas, los cortes de tipo guillotina, o el problema de corte por etapas [16, 84, 130], reflejan diferentes tipos de aplicaciones reales. En [129] y [182] se pueden encontrar análisis generales de los problemas de corte y empaquetado.

El diseño de una aproximación para resolver el 2DCSP depende generalmente del entorno particular de su aplicación, y de los medios computacionales disponibles. Como ocurre en la mayoría de los problemas de corte y empaquetado, el problema de encontrar la solución óptima para el 2DCSP es NP-duro. Esto supone que, aunque los algoritmos exactos aseguren la obtención de las soluciones óptimas, no pueden tratar con instancias del problema grandes y reales. Por ello el número de algoritmos exactos en la literatura no es demasiado extenso. Sin embargo, dado que el beneficio obtenido de una hoja de material es crucial para la mayoría de industrias de producción, el estudio y el diseño de las propuestas exactas puede ser un campo de investigación interesante.

Como ocurre con otros problemas de optimización combinatoria, la mayoría de las aproximaciones exactas para el 2DCSP están basadas en la enumeración del espacio de búsqueda de soluciones. Generalmente, estos algoritmos representan el espacio de búsqueda usando árboles. Existe una función de evaluación asociada con los nodos del árbol, y se aplica un procedimiento de búsqueda ordenado para ir visitando los nodos. En una formulación de ramificación y acotación, el orden en el que se recorre el árbol es normalmente el mismo que en la búsqueda en profundidad, así que no se necesita una función de evaluación para establecer el orden entre los nodos [39, 96]. Sin embargo, en este caso, la función de evaluación se usa para comparar los nodos recién creados (soluciones potenciales) con la mejor solución existente. De este modo, el árbol se poda evitando la exploración de ramas que no van a llevar a mejores soluciones que la actual. Por otra parte, en una búsqueda primero-el-mejor, el orden en el que se recorre el grafo de búsqueda lo determina una función de evaluación [49, 92, 175]. Además, la búsqueda para tan pronto se encuentre la solución completa (nodo objetivo). Si la función de evaluación satisface ciertas condiciones, entonces este esquema garantiza la obtención de soluciones óptimas.

Ambos procedimientos, búsquedas en árbol y ramificación y acotación, se han aplicado con éxito al 2DCSP y también a problemas donde se consideran patrones restringidos de tres etapas [46, 47, 48]. Sin embargo, ya que las aproximaciones primero-el-mejor han mostrado un comportamiento más eficiente [175], parte del trabajo realizado con el 2DCSP se ha centrado en el algoritmo original de Viswanathan y Bagchi (*Viswanathan and Bagchi's Algorithm*, VB).

El algoritmo VB [175] está basado en la propuesta de Wang para la construcción

de soluciones que se obtienen como combinaciones de patrones en vertical y horizontal [178]. La propuesta de Wang estudia una formulación menos general del problema donde los valores de los rectángulos demandados son directamente proporcionales a sus áreas. El método es heurístico y no garantiza la obtención de soluciones óptimas, aunque se han propuesto varias mejoras [147, 173, 174]. El algoritmo VB usa la solución de programación dinámica de Gilmore y Gomory [85] (con la versión ilimitada del problema) para construir una cota superior. Más tarde, Hifi [92] y Cung et al. [49] propusieron una versión modificada del algoritmo VB (llamada *Modified Viswanathan and Bagchi's Algorithm*, MVB) introduciendo una cota inferior inicial, una estructura de datos bi-dimensional para manejar una lista (que disocia el bucle de generación en dos: uno para las combinaciones horizontales y otro para las verticales), una cota superior reducida, y algunas reglas para encontrar en tiempo constante patrones duplicados y dominados. Los trabajos de los últimos años basados en el algoritmo VB [76, 77, 116, 117, 138, 139], incluyen varias mejoras sobre el algoritmo secuencial (nuevas cotas superiores e inferiores, estructuras de datos eficientes para el manejo de las construcciones, nueva reformulación del problema, etc.) y también algunas propuestas paralelas.

Ante la imposibilidad de tratar problemas grandes o reales con los algoritmos exactos, la mayoría de las investigaciones en la literatura se han centrado en desarrollar heurísticas que puedan proporcionar soluciones rápidas y de buena calidad (aunque no necesariamente las óptimas) para instancias que de otro modo serían intratables. La mayoría de estas heurísticas resuelven la versión del problema de no guillotina [14, 34, 100]. Sólo unas pocas tratan con el caso de guillotina [72, 146, 155, 178]. Como métodos más generales y sofisticados, se han considerado diferentes tipos de algoritmos híbridos y meta-heurísticas [7, 100, 170]. Son muchos los algoritmos genéticos que se han empleado para resolver este problema [8, 27, 98, 155, 156]. Por otra parte, no se conocen referencias de trabajos que aborden la resolución multi-objetivo para la formulación del 2DCSP que se trata en este trabajo [144, 158, 181, 185].

3.3. Representación de las soluciones

Wang [178] fue la primera en hacer la observación de que todos los patrones de corte de guillotina se pueden obtener por medio de combinaciones de construcciones en horizontal o en vertical. En esos patrones todas las piezas deben colocarse ortogonalmente sobre el material de tal modo que no se superpongan. Tal y como ya se ha mencionado, usando patrones de corte de guillotina cualquier corte debe poder realizarse desde un lado del rectángulo hasta el opuesto yendo paralelamente a los otros dos bordes. Así, estas construcciones se pueden representar utilizando una notación

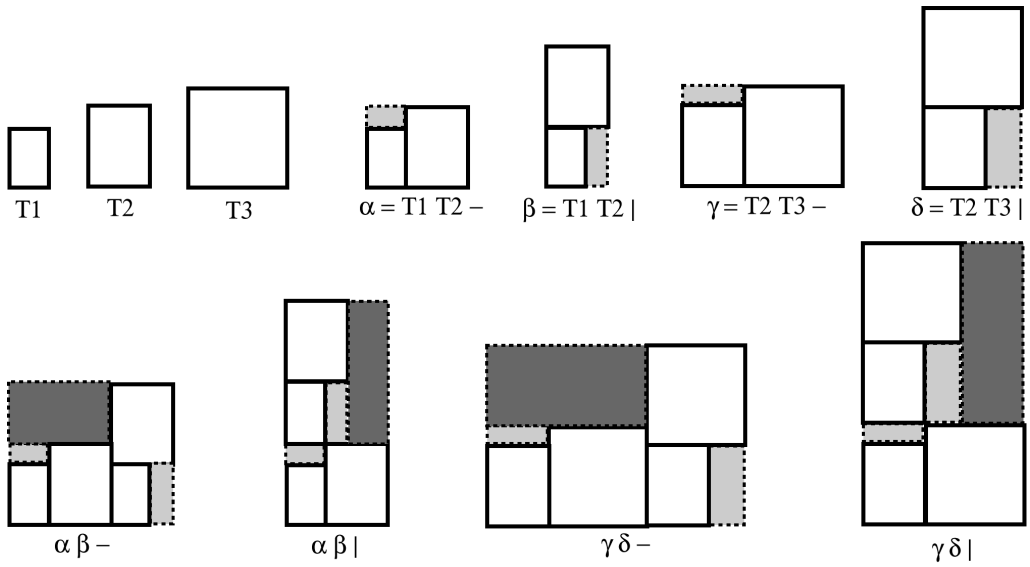

Figura 3.4: Ejemplos de construcciones horizontales y verticales

postfija. De este modo, la búsqueda de la solución se lleva a cabo directamente en el espacio de todas las posibles disposiciones guillotinables de piezas.

Dados dos meta-rectángulos α y β de dimensiones (l_α, w_α) y (l_β, w_β), la construcción vertical $\alpha|\beta$ es un meta-rectángulo de dimensiones $(\text{máx}\{l_\alpha, l_\beta\}, w_\alpha + w_\beta)$ que resulta de colocar β sobre α. La construcción horizontal $\alpha - \beta$ es un meta-rectángulo de dimensiones $(l_\alpha + l_\beta, \text{máx}\{w_\alpha, w_\beta\})$ que resulta de colocar β al lado de α. Usando esta idea, una solución factible puede ser representada por una fórmula como $(T_2|T_3)|(T_1 - T_2)$. Incluso se pueden utilizar expresiones postfijas que evitan la necesidad de usar paréntesis (es decir, $T_2T_3|T_1T_2 - |$), y proporcionan una representación compacta del árbol de sintaxis. La Figura 3.4 muestra ejemplos de expresiones postfijas, como $\alpha\ \beta\ |$ y $\alpha\ \beta\ -$. Estos ejemplos denotan las construcciones verticales y horizontales de los rectángulos α y β.

La Tabla 3.2 presenta una gramática libre del contexto G y las reglas semánticas asociadas para definir el beneficio (g), largo (l), ancho (w), y número de patrones utilizados (T_i), ejemplos de atributos asociados con los meta-rectángulos[1] $\alpha \in L(G)$, siendo $L(G)$ el lenguaje generado por la gramática G. S^{T_i} se inicializa a 0.

Para que la notación postfija sea válida, se ha de mantener para cada operador

[1]En el resto del trabajo los meta-rectángulos también se denominarán patrones de corte o combinaciones/construcciones de piezas

la siguiente condición:

$$1 \leq n_o \leq n_p - 1 \tag{3.2}$$

siendo n_p el número de piezas a su izquierda y n_o el número de operadores a su izquierda más él mismo. Es necesario destacar que en ocasiones la notación puede variar ligeramente para hacerla más legible. Por ejemplo, el operador horizontal $-$ puede ser sustituido por H y y el vertical $|$ por V.

En general, cuando se investiga y se trabaja con problemas de corte y empaquetado se desarrollan algoritmos cuya salida se encuentra en formato de texto, con la representación la solución que se haya escogido. En este caso las soluciones se encuentran representadas con la notación postfija explicada. Lo ideal sería contar con una representación gráfica de la misma de forma que se pueda visualizar la disposición final de las piezas en el material [163]. Dado que el número de herramientas no comerciales para trabajar con la visualización de los problemas de corte y empaquetado es muy reducido y sus funcionalidades son mucho más limitadas que las ofrecidas por los productos comerciales, se ha optado por el desarrollo de una interfaz gráfica de usuario [53] desde la que se pueden introducir los datos de un problema de corte, como la definición y la solución en formato de texto, y obtener la solución del mismo de manera gráfica (Figura 3.5).

Sintaxis	**Semántica**
$S \rightarrow S_1 S_2 \vert$	$S^g = S_1^g + S_2^g$ $S^l = \text{máx}\{S_1^l, S_2^l\};\ S^w = S_1^w + S_2^w$ $S^{T_i} = S_1^{T_i} + S_2^{T_i} \quad \forall T_i$
$S \rightarrow S_1 S_2 -$	$S^g = S_1^g + S_2^g$ $S^l = S_1^l + S_2^l;\ S^w = \text{máx}\{S_1^w, S_2^w\}$ $S^{T_i} = S_1^{T_i} + S_2^{T_i} \quad \forall T_i$
$S \rightarrow T_i$ para cada $T_i \in \mathcal{D}$	$S^g = T_i^p$ $S^l = T_i^l;\ S^w = T_i^w$ $S^{T_i} = S^{T_i} + 1$

Tabla 3.2: Definición dirigida por la sintaxis para un problema de corte

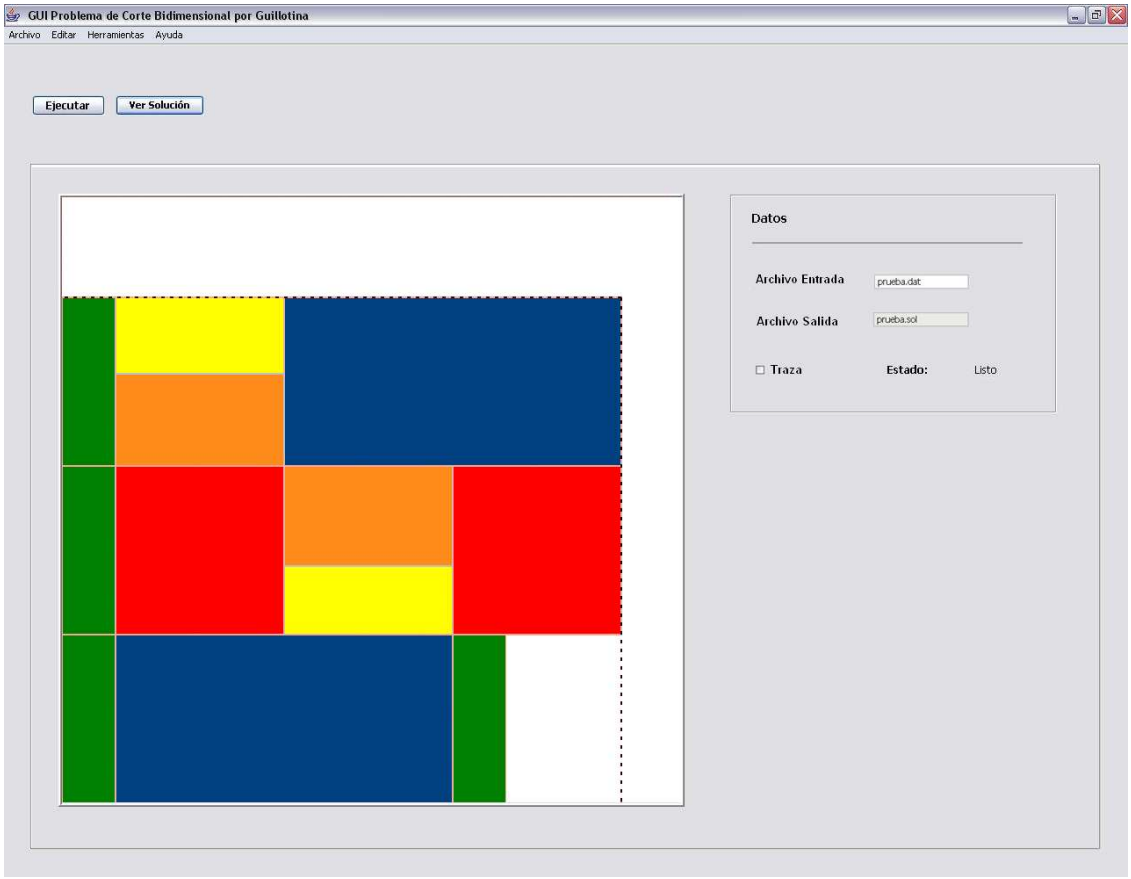

Figura 3.5: Interfaz gráfica de usuario

3.4. Solución exacta

Obtener la solución exacta del 2DCSP es de vital interés sobre todo en industrias que manejan materiales con un elevado coste, y que, por tanto, no pueden admitir más desperdicio del estrictamente necesario. La aplicación de métodos exactos también es interesante para aquellos casos en los que no hayan restricciones excesivas en cuanto al tiempo necesario para obtener las soluciones. Por estas razones, el primer estudio realizado para este problema se ha centrado en la implementación de un algoritmo exacto y en el posterior refinamiento del mismo para conseguir aumentar en la medida de lo posible su eficiencia.

Así, para resolver de manera exacta el 2DCSP el trabajo realizado se ha basado en el algoritmo original VB y en su posterior modificación, MVB, dado su comportamiento eficiente [49, 92, 140, 175]. El Algoritmo 1 muestra el funcionamiento del

Algoritmo 1 Versión modificada del algoritmo de Viswanathan y Bagchi

```
1: OPEN = {T_1, T_2, ..., T_n};
2: CLIST = ∅;
3: f' = cotaSuperior();
4: bestSol = cotaInferior();
5: bestSolValue = g(bestSol);
6: finished = false;
7: repeat
8:     elegir meta-rectángulo R de OPEN con mayor valor de f';
9:     if (h'(R) == 0) o ((f'(R) − bestSolValue) ≤ 0) then
10:        finished = true;
11:    else
12:        transferir R desde OPEN hasta CLIST;
13:        construir todo rectángulo de guillotina Q tal que:
14:            i. Q es una construcción horizontal o vertical de R con cualquier rectángulo R' de CLIST;
15:           ii. las dimensiones de Q son ≤ (L, W);
16:          iii. Q satisface todas las restricciones del problema;
17:        for cada elemento Q do
18:            if (g(Q) > bestSolValue) then
19:                bestSol = Q;
20:                bestSolValue = g(Q);
21:            end if
22:            if (f'(Q) > bestSolValue) then
23:                insertar Q en OPEN;
24:            end if
25:        end for
26:        eliminar todos los elementos R de OPEN que tengan f'(R) ≤ bestSolValue;
27:        if (OPEN está vacío) then
28:            finished = true;
29:        end if
30:    end if
31: until finished
32: return bestSol como solución al problema;
```

MVB. Éste usa dos listas - OPEN y CLIST - para producir el conjunto de soluciones factibles. La lista OPEN almacena las nuevas construcciones generadas durante las combinaciones previas. Consiste en un vector de punteros a listas enlazadas de subproblemas, que comienza en el valor de la heurística inicial para la cota inferior y termina en la cota superior del problema inicial. Los subproblemas con la misma cota superior irán en la misma lista enlazada (Figura 3.6). Mientras, la lista CLIST almacena los meta-rectángulos que ya han sido combinados con otros meta-rectángulos. Se trata de una estructura de datos bi-dimensional limitada por las dimensiones (L, W) de la hoja de material S, donde cada elemento es una lista enlazada de subproblemas con el mismo valor de largo y ancho (Figura 3.7). Inicialmente, OPEN contiene una copia de cada uno de los n rectángulos demandados de tipo T_i, y CLIST está vacía. En cada paso, se elige el meta-rectángulo más prometedor de OPEN y se mueve a CLIST. El elemento seleccionado, R, de tamaño $l_R \times w_R$, se combina horizontal y verticalmente con los elementos en CLIST (incluyendo R) para producir las construcciones horizontales y verticales. Todas las soluciones factibles obtenidas

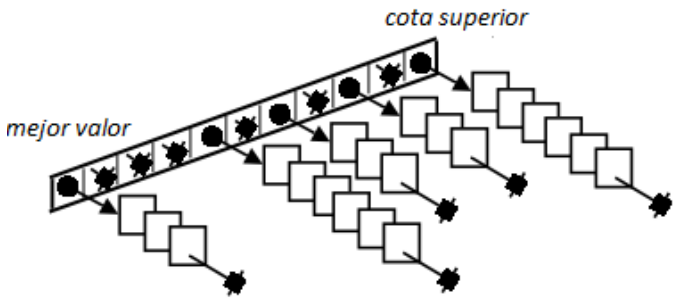

Figura 3.6: Estructura de datos para la lista OPEN

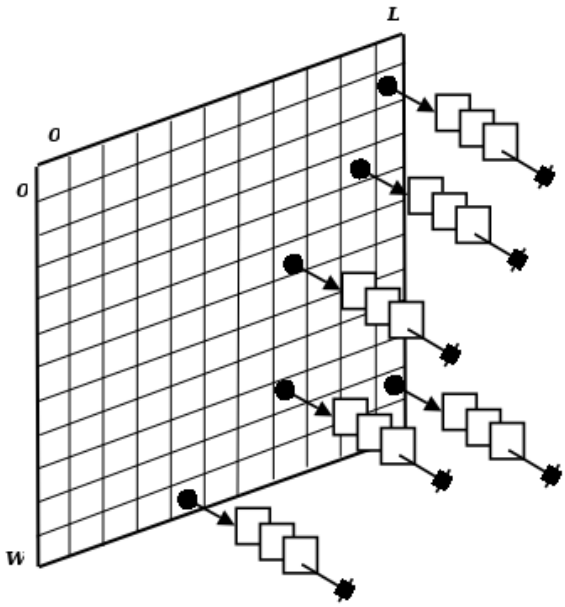

Figura 3.7: Estructura de datos para la lista CLIST

de los nuevos meta-rectángulos producidos se alojan en OPEN. La solución factible Q es un meta-rectángulo que satisface las siguientes condiciones:

1. Las dimensiones de Q no exceden las dimensiones de S, es decir, $l_Q \leq L$ y $w_Q \leq W$.

2. Para cada i, $1 \leq i \leq n$, el número de unidades del rectángulo demandando T_i en Q no excede la restricción de demanda b_i.

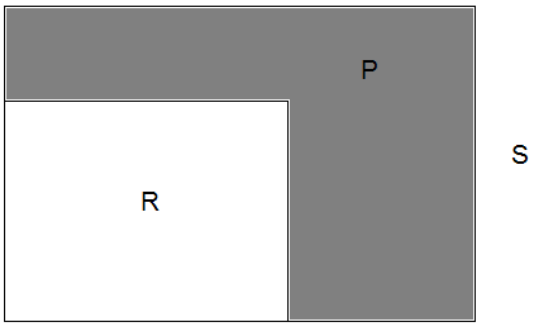

Figura 3.8: Superficie S, meta-rectángulo R, y área restante P

El resto del funcionamiento del algoritmo involucra la exploración del área no ocupada P en el rectángulo inicial (Figura 3.8), usando las cotas superiores apropiadas.

Para reducir el espacio de búsqueda, el algoritmo emplea una cota inferior [116] que permite definir la mejor solución inicial evitando comenzar la búsqueda desde cero. Además, para determinar qué construcciones son méjores y cuáles peores, se debe definir la cota superior del algoritmo [116]. Así pues, dada una instancia del problema y un rectángulo interno R, el objetivo es buscar una cota superior para el valor del mejor patrón de corte de guillotina en S que debe incluir R. Sea $g(R)$ el *beneficio acumulado de la construcción* R o *el valor interno de* R, es decir, la suma de los beneficios de todas las piezas contenidas en la construcción R. Sea $h(R)$ el *beneficio o valor máximo* que se puede obtener, del área restante P de la superficie S, sin violar las restricciones de demanda. Entonces, el *máximo beneficio o valor total* de un patrón de corte factible que incluye R se define como:

$$f(R) = g(R) + h(R) \tag{3.3}$$

En general, no es fácil determinar el valor exacto de $h(R)$, pero se puede encontrar una estimación superior $h'(R)$ sin demasiado esfuerzo. Usando este valor estimado, la cota superior del algoritmo o *beneficio total estimado* se define como:

$$f'(R) = g(R) + h'(R) \tag{3.4}$$

Los pasos de búsqueda descritos en el Algoritmo 1 se repiten hasta que el mejor meta-rectángulo actual R no tenga más posibilidades de insertar piezas en el área restante P ($h'(R) = 0$). Dado que la función h' es una estimación superior de h y no

hay más elementos en OPEN con un valor más alto f', cuando el siguiente elemento R a ser explorado satisfaga $h'(R) = 0$, el proceso de búsqueda puede parar porque se ha obtenido la solución óptima.

En algunos trabajos [76, 116], ya se ha demostrado la validez de esta propuesta secuencial para la solución exacta del 2DCSP cuando se compara con las originales. Por ello aquí el estudio se centra en reducir aún más el tamaño del espacio de búsqueda, introduciendo mejoras sobre las propuestas previas de otros autores. Las mejoras están basadas en la detección de patrones de cortes dominados y duplicados. De este modo, se descartan algunas construcciones redundantes o no óptimas, con lo cual el espacio de búsqueda se reduce significativamente. Tal reducción supone un decrecimiento del esfuerzo computacional asociado al proceso de búsqueda. A continuación se exponen las reglas empleadas para el descarte de construcciones.

3.4.1. Patrones de corte duplicados y dominados

El funcionamiento del algoritmo involucra la exploración de todas las ramas prometedoras del árbol de búsqueda, asegurando la obtención de la solución óptima. El uso de cotas inferiores y superiores posibilita una reducción significativa del espacio de búsqueda [116], aunque aún sigue siendo demasiado grande para la mayoría de instancias reales. Para reducir el espacio de búsqueda más, otra posibilidad es detectar las ramas duplicadas o dominadas del árbol. Para el 2DCSP, las ramas *duplicadas* representan patrones de corte equivalentes, es decir, físicamente representan el mismo uso del material, aunque el mismo conjunto de piezas se pueda obtener mediante procesos de corte diferentes (Figura 3.9). Las ramas *dominadas*, por otra parte, representan patrones de corte poco prometedores, es decir, existe una construcción similar que mejora de algún modo a la dominada. La Figura 3.10 muestra un patrón de corte dominado, γ_1, que tiene las mismas dimensiones que γ_2 pero incluye menos piezas. Todas las construcciones dominadas deben ignorarse porque ninguna de ellas llevará a la solución óptima. Para mejorar la eficiencia del algoritmo es esencial encontrar y eliminar los patrones duplicados y dominados, de manera que no se hagan exploraciones redundantes o infructuosas. Existen diferentes alternativas para introducir reglas de detección de patrones duplicados o dominados: reglas que se pueden aplicar en el proceso de generación (*reglas pre-generación*) o después (*reglas post-generación*).

3.4.1.1. Reglas pre-generación

Antes de hacer una combinación horizontal o vertical de patrones, es recomendable comprobar ciertas condiciones para evitar la creación de construcciones repetidas

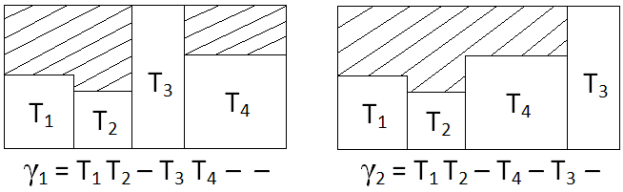

Figura 3.9: Patrones duplicados

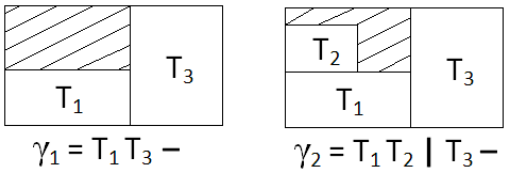

Figura 3.10: Patrón dominado γ_1

o poco prometedoras. Aquí se proponen tres tipos diferentes de reglas de duplicados/dominados. Están basadas en las propuestas presentadas en [49], aunque se han introducido algunas mejoras y se ha propuesto una nueva formulación para simplificar su entendimiento.

Tipo D1: patrones dominados en la generación

A veces la combinación de dos patrones diferentes produce una nueva construcción que puede introducir un desperdicio considerable del material. Cuando el espacio obtenido trás una combinación de dos construcciones es suficientemente grande para acomodar una o más piezas disponibles, se puede impedir la generación de tal construcción. Resulta obvio que esta construcción no llevará a la solución óptima porque existe una construcción posible que tiene las mismas dimensiones pero emplea mejor el espacio, introduciendo piezas extra en el espacio libre, y obteniendo un beneficio mayor. Todas las combinaciones que introducen este tipo de desperdicio deben ser descartadas, ya que representan zonas dominadas del espacio de búsqueda.

Sean α y β dos patrones diferentes. Sean γ_H y γ_V los patrones viables obtenidos por combinación horizontal y vertical de α y β. Sean Θ_H y Θ_V los espacios vacíos resultantes de la construcción horizontal y vertical respectivamente (Figura 3.11). Considerando que $x_i(\alpha)$ y $x_i(\beta)$ representan el número de piezas usadas del tipo T_i en la construcciones α y β, respectivamente, y que b_i es el número de piezas máximo de tipo i. Entonces γ_H o γ_V son patrones dominados si:

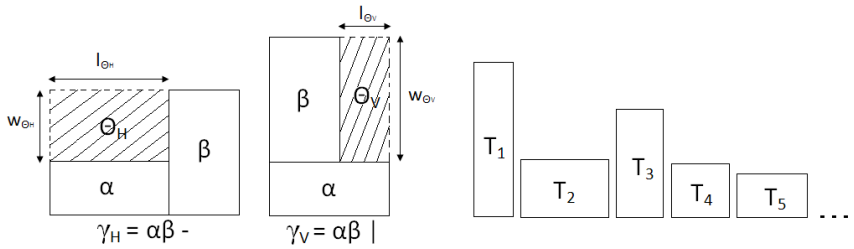

Figura 3.11: Patrones dominados (γ_H, γ_V) y piezas disponibles

$$\gamma_H : \ \exists i/T_i \in \mathcal{D} \ \ x_i(\alpha) + x_i(\beta) < b_i \text{ y } (l_{\Theta_H}, w_{\Theta_H}) >= (l_i, w_i)$$
$$\gamma_V : \ \exists i/T_i \in \mathcal{D} \ \ x_i(\alpha) + x_i(\beta) < b_i \text{ y } (l_{\Theta_V}, w_{\Theta_V}) >= (l_i, w_i)$$

donde $w_{\Theta_H} = |w_\alpha - w_\beta|$ y $l_{\Theta_H} = l_\alpha$ si $w_\alpha < w_\beta$ o $l_{\Theta_H} = l_\beta$ en otro caso. En el caso de la construcción vertical, $l_{\Theta_V} = |l_\alpha - l_\beta|$ y $w_{\Theta_V} = w_\alpha$ si $l_\alpha < l_\beta$ o $w_{\Theta_H} = w_\beta$ en otro caso.

Para demostrar esto matemáticamente considérese que $w_{\Theta_H} = |w_\alpha - w_\beta|$, $l_{\Theta_H} = l_\alpha$ y k es el índice de la pieza candidata para la que $(l_{\Theta_H}, w_{\Theta_H}) >= (l_k, w_k)$. Recuérdese que $g(\alpha)$ y $g(\beta)$ denotan respectivamente el valor interno (cotas inferiores) de α y β. Sea α' la construcción vertical entre α y la k-ésima pieza, con un valor interno $g(\alpha') = g(\alpha) + p(k\text{-ésima pieza})$. Recuérdese que p hace referencia al beneficio asociado. Sea γ'_H otra construcción horizontal entre α' y β, con un valor interno $g(\gamma'_H) = g(\alpha) + g(\beta) + p(k\text{-ésima pieza})$. Entonces, claramente los dos patrones γ_H y γ'_H tienen las mismas dimensiones. Por otra parte, $g(\gamma_H) < g(\gamma'_H)$ ya que:

$$g(\gamma_H) = g(\alpha) + g(\beta) < g(\alpha') + g(\beta) = g(\alpha) + g(\beta) + p(k\text{-ésima pieza})$$

donde $p(k\text{-ésima pieza}) > 0$. Por lo tanto, γ_H es dominado por el patrón γ'_H y γ_H representa un área duplicada del espacio de búsqueda que no es capaz de proporcionar una mejor solución que la obtenida de γ'_H. El algoritmo va a producir ambos patrones, pero es suficiente analizar uno de ellos: γ'_H. Para la construcción vertical de los patrones α y β se puede aplicar una comprobación equivalente.

La Figura 3.11 muestra un ejemplo práctico de dos construcciones, γ_H y γ_V, obtenidas por combinación horizontal y vertical de los patrones α y β. Las nuevas construcciones, γ_H y γ_V, son dominadas porque producen un espacio vacío que podría ser rellenado con algunas de las piezas disponibles. En el caso de γ_H, el espacio vacío resultante Θ_H puede acomodar cualquiera de las piezas T_2, T_4, o T_5, las cuales podrían haberse concatenado verticalmente con α antes de crear la construcción horizontal γ_H. En el caso de γ_V, el espacio vacío resultante Θ_V puede acomodar la

pieza T_3, es decir, la pieza T_3 podría haberse concatenado horizontalmente con β antes de realizar la construcción vertical γ_V. Hay que tener en cuenta que en ambos casos, se asume que hay suficiente disponibilidad de piezas T_2, T_4, T_5 y T_3.

Esta regla de dominancia se debe comprobar durante el proceso de búsqueda, en el paso dedicado a la combinación de la actual mejor construcción α con las construcciones ya exploradas β. Si se detecta que la nueva construcción γ_H o γ_V es un patrón dominado, no se insertará en la lista OPEN, evitando la exploración de patrones de corte no prometedores. Para cada par (α, β), la regla de dominancia debe comprobar si hay disponible alguna pieza que quepa en el hueco interno.

Tipo D2: patrones simétricos en direcciones opuestas

Sean α y β patrones compuestos exclusivamente por combinaciones horizontales de piezas. Estos patrones que contienen sólo concatenaciones horizontales se denotan como *patrón-H*. Sean α' y α'' los últimos subpatrones combinados horizontalmente para obtener α, y β' y β'' los últimos subpatrones combinados horizontalmente para obtener β. Considerando que $d = l_\alpha - l_\beta$ y $d \geq 0$, es decir, $l_\alpha \geq l_\beta$. Entonces, las combinaciones de patrones $\gamma_1 = \beta'\beta'' - \alpha'\alpha'' - |$ y $\gamma_2 = \beta'\alpha'|\beta''\alpha''|-$ se dice que son simétricas (Figura 3.12).

Si $l_{\alpha'} \geq l_{\beta'}$ y $l_{\alpha''} \geq l_{\beta''}$, es decir, $0 \leq l_{\alpha'} - l_{\beta'} \leq d$, γ_1 representa un patrón dominado y puede ser descartado. Como muestra la Figura 3.12, cuando $l_{\alpha'} \geq l_{\beta'}$ y $l_{\alpha''} \geq l_{\beta''}$, γ_2 domina a γ_1. Ambas contienen el mismo número de piezas, por lo que tienen el mismo beneficio, pero γ_2 hace un mejor uso del espacio. Tanto en γ_1 como en γ_2 la longitud viene dada por las longitudes de los subpatrones α' y α'', es decir, $l_{\gamma_1} = l_{\gamma_2} = l_{\alpha'} + l_{\alpha''}$. Sin embargo, los anchos de γ_1 y γ_2 son diferentes:

$$\begin{aligned} w_{\gamma_1} &= max\{w_{\beta'}, w_{\beta''}\} + max\{w_{\alpha'}, w_{\alpha''}\} \\ w_{\gamma_2} &= max\{(w_{\beta'} + w_{\alpha'}),\ (w_{\beta''} + w_{\alpha''})\} \end{aligned}$$

El ancho de γ_1 será siempre el máximo posible, mientras que γ_2 puede dar lugar a un ancho menor. De este modo, γ_1 y γ_2 son patrones simétricos (usan exactamente el mismo conjunto de piezas) pero γ_1 necesita un ancho mayor para distribuir las piezas. Como el objetivo del problema supone la maximización del uso del material, γ_1 representa una solución no prometedora que siempre es dominada por su patrón simétrico γ_2.

Este tipo de dominancia se comprueba dentro del bucle de búsqueda, y es evaluada cada vez que se va a realizar una concatenación vertical entre la actual mejor construcción α y un elemento de la lista OPEN, llamado β. Primero, es necesario comprobar si ambas construcciones, α y β, son *patrones-H*. En el caso de que ambas sean *patrones-H*, entonces si las condiciones $l_{\alpha'} \geq l_{\beta'}$ y $l_{\alpha''} \geq l_{\beta''}$ se cumplen, la nueva

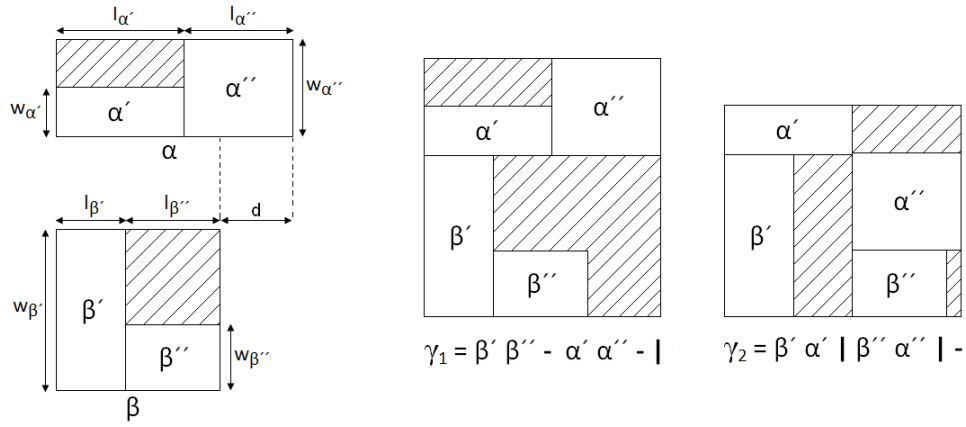

Figura 3.12: Patrones α y β y construcciones γ_1 y γ_2

construcción vertical $\gamma = \alpha\beta|$ (que corresponde a la previamente denominada γ_1) no se crea ni se inserta en la lista OPEN, porque durante el proceso de búsqueda un patrón (γ_2) se creará y tendrán mejores propiedades que γ.

Una vez comprobado que las construcciones α y β son *patrones-H*, se consideran todas las posibles divisiones de los patrones α y β en subpatrones α', α'', β', y β''. Aunque de esta manera se incrementa el coste asociado a la comprobación de la regla, en relación con el coste que tendría si sólo se comprobara la última división de los *patrones-H*, esto hace posible detectar más nodos dominados. Además, durante la búsqueda no se exploran *patrones-H* muy largos ya que los patrones de corte más prometedores normalmente involucran ambas combinaciones, horizontales y verticales.

Tipo D3: patrones simétricos en la misma dirección

Sean α y β patrones obtenidos sólo por combinaciones horizontales de piezas, es decir, *patrones-H* (o solo por combinaciones verticales de piezas, es decir, *patrones-V*). La construcción horizontal (el caso vertical puede hacerse del mismo modo) entre esos dos patrones, α (tomado de OPEN) y β (tomado de CLIST), no se crea si uno de los siguientes casos se verifica:

1. α está compuesto por diferentes tipos de piezas (Figura 3.13):

$$\exists i, j / T_i, T_j \in \mathcal{D},\ x_i(\alpha) \geq 1,\ x_j(\alpha) \geq 1,\ i \neq j$$

2. α está compuesto por un solo tipo de pieza, β está compuesto por diferentes tipos de piezas, y una de las piezas de β es la pieza que compone α (Figura 3.14):

$$\exists i/T_i \in \mathcal{D},\ x_i(\alpha) \geq 1 \ \text{ y } \ \nexists j/T_j \in \mathcal{D},\ x_j(\alpha) \geq 1,\ i \neq j$$
$$\exists k,l/T_k,T_l \in \mathcal{D},\ x_k(\beta) \geq 1,\ x_l(\beta) \geq 1,\ k \neq l$$
$$\exists m/T_m \in \mathcal{D},\ x_m(\beta) \geq 1,\ m = i$$

3. α está compuesta por un solo tipo de pieza, β está compuesta por un solo tipo de pieza, esta pieza es la misma para α y β, y la diferencia entre el número de piezas en α y β es mayor 1 (Figura 3.15):

$$\exists i/T_i \in \mathcal{D},\ x_i(\alpha) \geq 1 \ \text{ y } \ \nexists j/T_j \in \mathcal{D},\ x_j(\alpha) \geq 1, i \neq j$$
$$\exists k/T_k \in \mathcal{D},\ x_k(\beta) \geq 1 \ \text{ y } \ \nexists l/T_l \in \mathcal{D},\ x_l(\beta) \geq 1, k \neq l$$
$$|x_i(\alpha) - x_k(\beta)| > 1,\ i = k$$

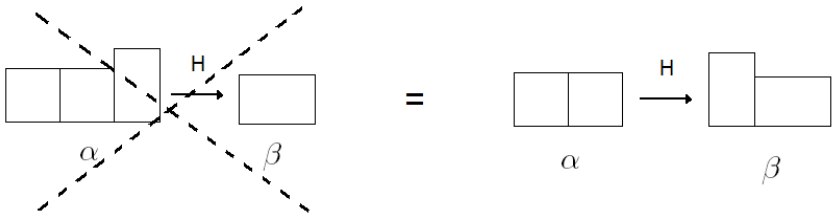

Figura 3.13: Patrones simétricos en la misma dirección: caso 1

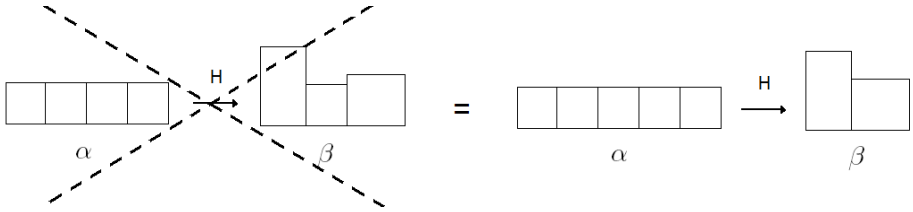

Figura 3.14: Patrones simétricos en la misma dirección: caso 2

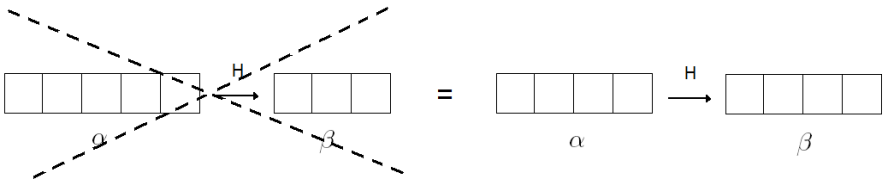

Figura 3.15: Patrones simétricos en la misma dirección: caso 3

Como se muestra en la Tabla 3.3, cada uno de los tres casos previos genera un patrón duplicado, el cual es, de cualquier modo, creado en otro paso del proceso de búsqueda. La Tabla 3.3 muestra el contenido de las listas OPEN y CLIST durante el proceso de búsqueda. La posición de los elementos en OPEN viene dada por sus valores f'. Para simplificar la traza, se han considerado sólo las combinaciones horizontales, y se ha supuesto que todas esas nuevas construcciones son patrones factibles (satisfacen las restricciones del problema sobre la disponibilidad de piezas y las dimensiones de la materia prima). Después del paso de búsqueda 2, el siguiente elemento que se extrae de OPEN es $\alpha = T_3T_2-$. Como está compuesto por diferentes tipos de piezas (T_3 y T_2), se verifica el caso 1, y entonces no es necesario combinarlo con los elementos de CLIST. Para demostrar esta regla de duplicados, la tabla muestra una descripción sobre cómo esas combinaciones descartadas son creadas de otro modo más adelante en el proceso de búsqueda. Después del paso de búsqueda 3, $\alpha = T_2T_2-$ se debe combinar con los elementos de CLIST. Antes de combinar α con $\beta = T_3T_2-$ se detecta una situación del caso 2, por lo que un patrón equivalente a $\alpha\beta-$ será creado en pasos futuros de la búsqueda. Yendo un paso más adelante en la búsqueda, $\alpha = T_2T_2 - T_2T_2 - -$. Cuando se intenta combinar α con los elementos de CLIST T_2 y T_2T_2-, se verifica el caso 3, así que la combinación horizontal se descarta.

Estos tres casos de patrones duplicados se comprueban siempre en cada paso de la búsqueda, antes de realizar la concatenación de los patrones α y β. Si ambas construcciones, α y β, son *patrones-H*, entonces si cualquiera de las tres condiciones se satisface, la nueva construcción $\alpha\beta-$ no se genera. Si ambas construcciones, α y β, son *patrones-V*, entonces si cualquiera de las tres condiciones se satisface, la nueva construcción $\alpha\beta|$ no se genera.

3.4.1.2. Reglas post-generación

Una vez que los nuevos patrones han sido generados se mantienen en una de las dos estructuras de datos disponibles: OPEN o CLIST. Los elementos en OPEN representan construcciones generadas pendientes de ser exploradas. Se encuentran agrupadas por su valor f'. Sin embargo, los elementos en CLIST ya han sido explorados, es decir, han sido extraídos de OPEN y combinados con los elementos en CLIST.

Tipo D4: duplicados/dominados en OPEN

Antes de insertar un patrón γ en OPEN, se comprueba si OPEN ya contiene un patrón γ' con el mismo conjunto de piezas y dimensiones menores o iguales:

$$\forall i/T_i \in \mathcal{D},\ x_i(\gamma') = x_i(\gamma)\ \text{ y } l_{\gamma'} \leq l_\gamma,\ w_{\gamma'} \leq w_\gamma$$

Inicialización:	OPEN = $\{T_2, T_3, T_1\}$	CLIST = $\{\ \}$
Paso 1:	OPEN = $\{T_3, T_2T_2-, T_1\}$	CLIST = $\{T_2\}$
Paso 2:	OPEN = $\{T_3T_2-, T_2T_2-, T_3T_3-, T_1\}$	CLIST = $\{T_2, T_3\}$
Paso 3:	OPEN = $\{T_2T_2-, T_3T_3-, T_1\}$	CLIST = $\{T_2, T_3, T_3T_2-\}$
CASO 1	Descartado	Obtenido por combinación de
	$T_3T_2 - T_2-$	T_2T_2- de OPEN y T_3 de CLIST
	$T_3T_2 - T_3-$	T_3T_3- de OPEN y T_2 de CLIST
	$T_3T_2 - T_3T_2 - -$	T_3T_3- de OPEN y T_2T_2- de CLIST (paso siguiente)
Paso 4:	OPEN = $\{T_2T_2 - T_2T_2 - -, T_2T_2 - T_3-, T_2T_2 - T_2-, T_3T_3-, T_1\}$ CLIST = $\{T_2, T_3, T_3T_2-, T_2T_2-\}$	
CASO 2	Descartado	Obtenido por combinación de
	$T_2T_2 - T_3T_2 - -$	$T_2T_2 - T_2-$ de OPEN y T_3 de CLIST
Paso 5:	OPEN = $\{T_2T_2 - T_2T_2 - -T_3-, T_2T_2 - T_3-, T_2T_2 - T_2-, T_3T_3-,$ $T_1, T_2T_2 - T_2T_2 - -T_2T_2 - T_2T_2 - --\}$ CLIST = $\{T_2, T_3, T_3T_2-, T_2T_2-, T_2T_2 - T_2T_2 - -\}$	
CASO 3	Descartado	Obtenido por combinación de
	$T_2T_2 - T_2T_2 - -T_2-$	$T_2T_2 - T_2-$ de OPEN y T_2T_2- de CLIST
	$T_2T_2 - T_2T_2 - -T_2T_2 - -$	$T_2T_2 - T_2-$ de OPEN y $T_2T_2 - T_2-$ de CLIST (pasos siguientes)
	$T_2T_2 - T_2T_2 - -T_3T_2 - -$	Pertenece al Caso 2

Tabla 3.3: Traza de la detección de patrones simétricos en la misma dirección

En este caso γ es dominado (o duplicado) si sus dimensiones son mayores o coinciden con las de γ' por lo que no se insertará en la lista. Por el contrario, si γ domina algunos de los elementos de OPEN, estos deben borrarse de la lista. De esta manera se evita la inserción de un elemento en OPEN, ahorrando la futura exploración de un patrón de corte.

Cuando γ se compara con los elementos en OPEN, no se realiza la comprobación con todos los patrones que contiene la lista OPEN, puesto que esto supondría demasiado esfuerzo computacional. Sólo se compara con los elementos en OPEN con el mismo valor f', es decir, el mismo beneficio total estimado.

Tipo D5: patrones duplicados/dominados en CLIST

Cada vez que un patrón α se va a insertar en CLIST, se comprueba si ya hay un patrón α' en CLIST que utilice al menos el mismo conjunto de piezas que contiene α. Gracias a la estructura de CLIST, que almacena los elementos por largo y ancho, α solamente se compara con los elementos en CLIST que tienen las mismas dimensiones de largo y ancho:

$$\forall i/T_i \in \mathcal{D},\ x_i(\alpha') \geq x_i(\alpha) \ \text{ y } l_{\alpha'} = l_\alpha,\ w_{\alpha'} = w_\alpha$$

Así, si un elemento α' con las mismas dimensiones que α usa las mismas piezas, α es un duplicado. Si α' usa el mismo conjunto de piezas que α y alguna otra más, entonces α es dominado por α'. En cualquier caso, α no se insertará en la lista. Por el contrario, si α usa las mismas piezas y alguna más, que alguno de los elementos en CLIST con las mismas dimensiones que α, estos son dominados por α y por lo tanto se eliminan de la lista. Evitando la inserción de un elemento en CLIST se ahorran dos combinaciones (horizontal y vertical) para cada uno de los patrones que se exploren posteriormente.

Es necesario destacar que cuando las reglas de pre-generación no están activas, la regla D5 es capaz de detectar y eliminar los patrones que hubieran sido detectados por las reglas D1 y D3. Dado que las parejas de patrones no-dominados y dominados, y no-duplicados y duplicados analizados por las reglas D1 y D3 respectivamente, tienen las mismas dimensiones, se introducirán tarde o temprano en la misma posición de la estructura CLIST, así que, podrán de ser descartados por la regla D5.

Estas dos reglas de post-generación se aplican solo para un subconjunto de elementos en OPEN y CLIST, respectivamente, tal y como se ha explicado. En el caso de la regla D4, se podría aplicar a toda la lista OPEN buscando construcciones con el mismo conjunto de piezas, dimensiones menores o iguales y diferente f', y en el caso de la regla D5, se podría aplicar a otras construcciones con otras dimensiones en CLIST que contuvieran el mismo conjunto de piezas. Se podría pensar que sería mejor aplicar la comparación de los patrones dados con todos los elementos en OPEN o CLIST, respectivamente. Sin embargo, en ese caso la comprobación de la regla no compensa para la mejora obtenida por la reducción de nodos. Por esta razón, se ha decidido diseñar las reglas de manera que se apliquen las comparativas en un grupo más reducido de patrones.

3.4.2. Resultados computacionales

Para analizar el impacto de las reglas de dominancias y duplicados, se han introducido en el algoritmo secuencial y se ha realizado un estudio computacional usando algunas instancias disponibles en [56, 90], ampliamente utilizadas en muchos estudios relacionados [7, 37, 59, 116]. Las pruebas se han realizado en un nodo SMP NovaScale 6320, que soporta hasta 32 procesadores Intel® Itanium 2 a 1.5GHz, y se ha usado el compilador *gcc 3.4.6*. Para cada experimento se han realizado diez ejecuciones y se han considerado los valores medios. Los tiempos se muestran en segundos.

En algunos trabajos [76, 116], ya se ha demostrado la validez de esta propuesta secuencial cuando se compara con las originales. Por ello aquí el estudio se centra

Problema	**Tiempo de Búsqueda**			
	Sin Dom.	Pre-gen.	Post-gen.	Todas Dom.
ATP33s	2522.22	66.44	3.45	3.46
ATP36s	80.18	12.93	1.23	1.22
ATP37s	341.85	5.45	2.67	2.59
ATP39s	17.28	4.3	2.78	2.72
CL_07_25.08	140.25	20.70	6.79	5.71
CL_07_25.10	996.27	87.01	23.94	16.41
CL_07_50.09	184.55	8.24	4.22	2.78
CL_07_100.08	115.26	12.99	6.21	4.58
CW6	132.83	24.10	5.91	5.69
Hchl2	1752.04	329.92	143.24	100.86
Hchl5s	180.31	22.62	5.87	5.52
Hchl5s_	193.76	24.29	8.97	7.93

Tabla 3.4: Efecto de las reglas de dominancias y duplicados

en cómo afecta a la propuesta secuencial la detección y eliminación de las ramas dominadas y duplicadas del árbol de búsqueda. La Tabla 3.4 presenta los resultados obtenidos cuando las reglas de detección de dominados y duplicados se incorporan en el algoritmo MVB. Se muestra el tiempo total de ejecución empleado en la búsqueda de la solución cuando: no se aplican las reglas de detección, se aplican sólo las reglas de pre-generación, se aplican sólo las reglas de post-generación, y se aplican ambos tipos de reglas. Los resultados demuestran que ambos conjuntos de reglas de dominancias permiten la reducción del espacio de búsqueda, aunque la comprobación de las de pre-generación parece ser más costosa en el proceso de búsqueda global. Las reglas de post-generación se comprueban con menos frecuencia (sólo en los etapas de inserción o computación) y no son demasiado costosas computacionalmente. Además, son capaces de descartar una cantidad considerable de nodos, incluyendo también algunas de las detecciones de las reglas de pre-generación. A pesar de que la comprobación de las reglas de pre-generación se hace de manera más frecuente, parece que el coste de la comprobación no compensa tanto en cuanto al número de nodos que permite descartar.

3.4.3. Conclusiones

Se ha presentado un conjunto de reglas de detección de dominancias y duplicados que hace posible la reducción significativa del espacio de búsqueda, permitiendo mejorar la eficiencia del algoritmo primero-el-mejor para el 2DCSP. Las reglas de detección de dominancias y duplicados se basan en las propiedades internas de las soluciones del problema, es decir, en la combinación de piezas para generar los pa-

trones de corte. Algunas reglas se pueden comprobar antes del proceso de generación de patrones y otras, por el contrario, se comprueban después de la creación. Aunque se aplican menos frecuentemente, las reglas de post-generación han mostrado un mayor efecto en la reducción del tiempo total de búsqueda. No son tan costosas computacionalmente y son capaces de detectar construcciones redundantes que también son detectadas por algunas de las reglas de pre-generación.

3.5. Paralelizaciones de la solución exacta

Se ha propuesto un conjunto de reglas de dominancias/duplicados que hacen posible reducir considerablemente el espacio de búsqueda del problema, y que, por lo tanto, reducen también el tiempo de computación requerido por el algoritmo de búsqueda secuencial. Sin embargo, la complejidad intrínseca del algoritmo MVB aún dificulta computacionalmente la solución de las instancias más complejas del problema. Por esta razón, se han estudiado las posibilidades de paralelización del algoritmo. Un análisis del Algoritmo 1 muestra su naturaleza intrínsecamente secuencial. Cada paso requiere que se compute el meta-rectángulo no explorado más prometedor. Esa computación consiste en combinar en horizontal y vertical el meta-rectángulo seleccionado con *todos los ya explorados*. Todas esas combinaciones son necesarias para asegurar la obtención de la mejor solución exacta. Si algunas de estas combinaciones se pierde, se puede perder la mejor solución. Teniendo en cuenta este tipo de dependencias entre las tareas del problema, se han diseñado dos tipos de aproximaciones paralelas que aseguran la obtención de la solución óptima. La primera propuesta es una aproximación de grano fino y supone la paralelización del paso donde se realiza la generación de las construcciones. La segunda alternativa es una aproximación de grano grueso que se basa en la paralelización del bucle completo de búsqueda. El estudio de estos esquemas paralelos muestra que cualquier intento de paralelizar el algoritmo MVB debe considerar su estructura computacional especialmente irregular.

3.5.1. Esquema de grano fino

Esta implementación está basada en la generación paralela de meta-rectángulos a partir de la combinación del actual mejor meta-rectángulo con los ya explorados y almacenados en CLIST. El funcionamiento general de este esquema sigue la misma estructura que el esquema secuencial presentado en el Algoritmo 1. La principal diferencia aparece en el bucle donde se realiza la generación de nuevas construcciones. Cada procesador involucrado trabaja en una sección de la estructura de datos bi-dimensional CLIST, combinando el actual mejor meta-rectángulo con un

subconjunto de elementos en CLIST. Cada procesador mantiene una copia replicada de CLIST. Sin embargo, la lista OPEN está distribuida y solo contiene las construcciones locales generadas por su propio procesador. Este manejo de las estructuras permite a los procesadores trabajar independientemente en la generación de nuevos meta-rectángulos. Sin embargo, después de cada combinación del actual mejor meta-rectángulo con el subconjunto de elementos en CLIST, cada procesador debe identificar su propia "mejor" construcción, es decir, el meta-rectángulo local con cota superior más alta. Para determinar cuál es el mejor meta-rectángulo global actual que se expandirá después, cada procesador comunica su mejor construcción actual. Desde este subconjunto de mejores construcciones se determina la mejor global y se le comunica a todos los procesadores. Esta comunicación de todos con todos se muestra en la Figura 3.16. Una vez que los procesadores tienen la nueva mejor construcción, pueden comenzar con el trabajo de generación. El mismo punto de reducción se usa para actualizar el valor de la mejor solución global, de manera que los patrones de corte no prometedores se puedan descartar.

Inicialmente se ha escogido OpenMP para la implementación de la aproximación de grano fino [76] debido a su simplicidad y su facilidad para trabajar con bucles en paralelo. El Algoritmo 2 muestra el pseudocódigo en OpenMP para k hilos. Está basado principalmente en la paralelización de los dos bucles *for* junto con el punto de reducción para la actualización de las variables: *mejor meta-rectángulo siguiente* y *mejor solución actual*. Usando OpenMP es fácil personalizar la distribución del trabajo. Los elementos de CLIST se comparten usando cláusula *schedule* de OpenMP. Se han probado diferentes opciones para esta cláusula, como *static* y *dynamic*, pero no se han notado diferencias relevantes entre ellas. El principal obstáculo en la im-

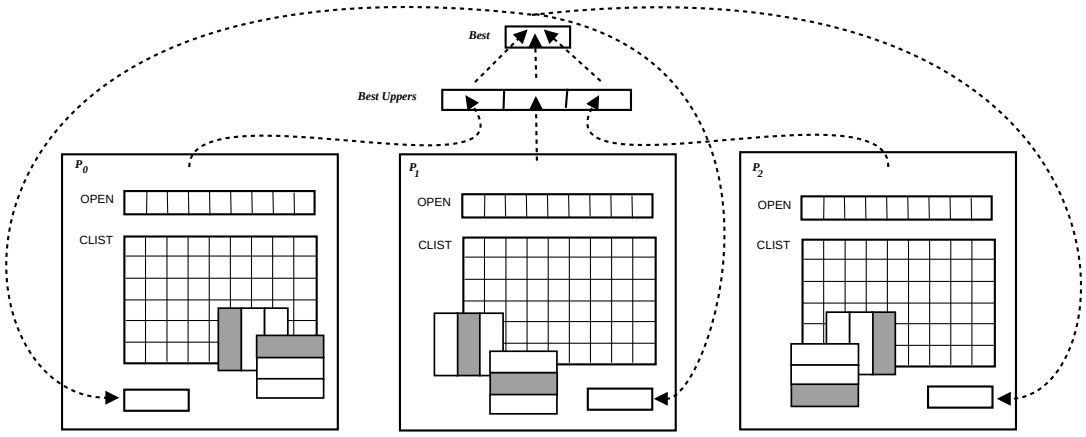

Figura 3.16: Punto de reducción

Algoritmo 2 Paralelización de grano fino para la resolución exacta del 2DCSP

```
1: if (k == MASTER) then
2:     OPEN_k = {T_1, T_2, ..., T_n};
3:     sharedBestUpper[k] = staticMap(elemento en OPEN_k con mayor f');    sharedBestUpId = k;
4: else
5:     OPEN_k = ∅;
6: end if
7: CLIST = ∅;    bestSol = lowerBound();    bestSolValue = g(bestSol);
8: while (∃i/ OPEN_i ≠ ∅) do
9:     if (k == sharedBestUpId) then
10:        eliminar meta-rectángulo α de OPEN_k con mayor f';
11:    else
12:        α = dynamicMap(sharedBestUpper[sharedBestUpId]);
13:    end if
14:    insertar α en CLIST;
15:    # pragma omp parallel for
16:    for (todo β en CLIST / x_i(α) + x_i(β) ≤ b_i ∀i, l_β + l_α ≤ L) do
17:        Γ_H = αβ−;      /* construcción horizontal */
18:        l_{Γ_H} = l_α + l_β;  w_{Γ_H} = máx(w_α, w_β);  g(Γ_H) = g(α) + g(β);
19:        x_i(Γ_H) = x_i(α) + x_i(β) ∀i ∈ [1, n];
20:        if (g(Γ_H) >bestSolValue) then
21:            vaciar OPEN_k desde bestSolValue a g(Γ_H);
22:            bestSolValue = g(Γ_H);    bestSol = Γ_H;
23:        end if
24:        if (f'(Γ_H) >bestSolValue) then
25:            insertar Γ_H en OPEN_k en la entrada f'(Γ_H);
26:        end if
27:    end for
28:    # pragma omp parallel for
29:    for (todo β en CLIST / x_i(α) + x_i(β) ≤ b_i ∀i, w_β + w_α ≤ W) do
30:        Γ_V = αβ|;      /* construcción vertical */
31:        l_{Γ_V} = máx(l_α, l_β);  w_{Γ_V} = w_α + w_β;  g(Γ_V) = g(α) + g(β);
32:        x_i(Γ_V) = x_i(α) + x_i(β) ∀i ∈ [1, n];
33:        ...
34:    end for
35:    sharedBestUpper[k] = staticMap(element en OPEN_k con mayor f');
36:    sharedBestSolValue[k] = bestSolValue;
37:    # pragma omp flush (sharedBestUpper, sharedBestSolValue)
38:    if (k == MASTER) then
39:        sharedBestUpId = i/ ∄j sharedBestUpper[j] > sharedBestUpper[i];
40:        sharedBestSolId = i/ ∄j sharedBestSolValue[j] > sharedBestSolValue[i];
41:    end if
42:    # pragma omp flush (sharedBestUpId, sharedBestSolId)
43:    if (sharedBestSolValue[sharedBestSolId] != bestSolValue) then
44:        vaciar OPEN_k desde bestSolValue hasta sharedBestSolValue[sharedBestSolId];
45:        bestSolValue = sharedBestSolValue[sharedBestSolId];    bestSol = NULL;
46:    end if
47: end while
48: if (bestSol != NULL) then
49:    return bestSol;
50: end if
```

plementación es el uso de estructuras dinámicas como las listas enlazadas (OPEN y CLIST). No se han hallado mecanismos disponibles que aseguren la integridad de los datos dinámicos cuando se ha tratado de encontrar una aproximación más inmediata donde estas listas dinámicas sean compartidas, y puedan ser modificadas por cada hilo. En OpenMP existe un pragma que asegura la integridad de una variable estática, *#pragma flush (nombre_de_la_variable_estática)*, pero no se puede hacer un *flush* a una variable asignada en la pila. Por lo tanto se requieren algunas operaciones adi-

cionales para compartir los datos dinámicos, como comunicar la mejor construcción actual en cada iteración.

Este algoritmo paralelo se puede implementar fácilmente en un esquema de memoria distribuida. En este caso, los procesadores pueden intercambiar las variables *mejor construcción* y *valor de la mejor solución* a través de las funciones MPI en lugar de usar estructuras de datos con memoria estática. Más específicamente, las líneas 35-36 del Algoritmo 2 se reemplazan por un empaquetado del mejor meta-rectángulo y el valor de la mejor solución en un buffer de comunicación. Para la comunicación de esta información, la línea 37 se reemplaza por una operación *MPI_Allgatherv*. Luego las líneas 38-46 se reemplazan por lo siguiente: cada procesador desempaqueta la información reunida, actualiza el valor de la mejor solución y el mejor meta-rectángulo actual que será analizado. En esta implementación, cualquier distribución dinámica de las iteraciones del bucle es más compleja que las que se consiguen con la cláusula *schedule* de OpenMP. El usuario debe manejar explícitamente todas las variables compartidas, puntos de sincronización y cierres, disminuyendo la simplicidad de la aproximación.

3.5.2. Esquema de grano grueso

La aproximación de grano grueso consiste en la ejecución paralela del bucle principal de búsqueda junto con un esquema de sincronización flexible y una estrategia de balanceo de carga [116]. El Algoritmo 3 muestra el código para el procesador k. El esquema paralelo replica CLIST y distribuye OPEN entre los procesadores disponibles, p. Cada procesador de manera independiente sigue los mismos pasos que en el algoritmo secuencial: selecciona el mejor meta-rectángulo de OPEN local para su combinación con los elementos en CLIST. La comunicación entre los procesadores se realiza al menos cada $tPeriod$ segundos o $iPeriod$ iteraciones de búsqueda, siendo $tPeriod$ e $iPeriod$ parámetros de configuración especificados por el usuario. Este punto de comunicación es estrictamente necesario para generar el conjunto completo de soluciones factibles. En este punto, los procesadores llevan a cabo una comunicación de todos con todos (línea 5) para intercambiar información sobre: el mejor valor de la solución, el número de elementos en OPEN y los conjuntos de construcciones que se explorarán desde el último paso de comunicación, y que deben ser combinados entre sí para asegurar la generación del conjunto completo de posibles soluciones. Con la información de las mejores soluciones locales, la mejor solución global actual se actualiza, permitiendo a los procesadores evitar, durante los siguientes pasos de búsqueda, la exploración innecesaria de las áreas del espacio de búsqueda. Por otra parte, el número de elementos en cada estructura OPEN también se intercambia para tener una idea aproximada de la carga de trabajo de asociada a cada procesador,

Algoritmo 3 Paralelización de grano grueso para la resolución exacta del 2DCSP

```
1: OPEN_k = {T_{k+j*p} / k + j * p < n};    CLIST = ∅;
2: bestSol = lowerBound();    bestSolValue = g(bestSol);
3: while (∃i/ OPEN_i ≠ ∅) do
4:     if (OPEN_k == ∅) o (time ≥ tPeriod) o (iters == iPeriod) then
5:         (λ, C, R) = allToAll (bestSolValue, sizeof(OPEN_k), PC);
6:         if (max(λ) > bestSolValue) then
7:             vaciar OPEN_k desde bestSolValue hasta max(λ);
8:             bestSolValue = max(λ);    bestSol = NULL;
9:         end if
10:        CLIST = CLIST ∪ (R - PC);    PC = ∅;
11:        Π = {π_0, ..., π_{p-1}} partición de {S ⊗ T / S, T ∈ R; S ≠ T};
12:        realizar las combinaciones verticales y horizontales de π_k;
13:        if (balanceRequired(C, minBalThresh, maxBalThresh)) then
14:            loadBalance(C, maxBalanceLength);
15:        end if
16:        iters = time = 0;
17:    else
18:        elegir meta-rectángulo α de OPEN_k con mayor f';
19:        insertar α en CLIST y en PC;
20:        for (todo β en CLIST / x_i(α) + x_i(β) ≤ b_i∀i, l_β + l_α ≤ L) do
21:            Γ_H = αβ−;      /* construcción horizontal */
22:            l_{Γ_H} = l_α + l_β;  w_{Γ_H} = máx(w_α, w_β);  g(Γ_H) = g(α) + g(β);
23:            x_i(Γ_H) = x_i(α) + x_i(β) ∀i ∈ [1, n];
24:            if (g(Γ_H) >bestSolValue) then
25:                vaciar OPEN_k desde bestSolValue hasta g(Γ_H);
26:                bestSolValue = g(Γ_H);    bestSol = Γ_H;
27:            end if
28:            if (f'(Γ_H) >bestSolValue) then
29:                insertar Γ_H en OPEN_k en la entrada f'(Γ_H);
30:            end if
31:        end for
32:        for (todo β en CLIST / x_i(α) + x_i(β) ≤ b_i∀i, w_β + w_α ≤ W) do
33:            Γ_V = αβ|;      /* construcción vertical */
34:            l_{Γ_V} = máx(l_α, l_β);  w_{Γ_V} = w_α + w_β;  g(Γ_V) = g(α) + g(β);
35:            x_i(Γ_V) = x_i(α) + x_i(β) ∀i ∈ [1, n];
36:            ...
37:        end for
38:        iters = iters + 1;
39:    end if
40: end while
41: if (bestSol != NULL) then
42:     return bestSol;
43: end if
```

necesaria para aplicar estrategias de balanceo de carga (línea 14).

El comportamiento del algoritmo dependerá de cómo se introduzcan los puntos de comunicación. Si los pasos de comunicación se hacen sólo cuando todos los procesadores no tienen trabajo, el algoritmo paralelo no obtendrá aceleración. Los procesadores pasarían la mayoría de su tiempo haciendo combinaciones innecesarias y esperando a que otros procesadores intercambien información. Así que la comunicación entre los procesadores debe realizarse frecuentemente durante el proceso de búsqueda. La idea más simple es introducir esos puntos de comunicación después de cierto número de pasos de búsqueda, *iPeriod*. El inconveniente de esta propuesta es que la carga de trabajo asociada con cada iteración de búsqueda puede diferir considerablemente. La Figura 3.17 muestra la irregularidad de la carga de trabajo

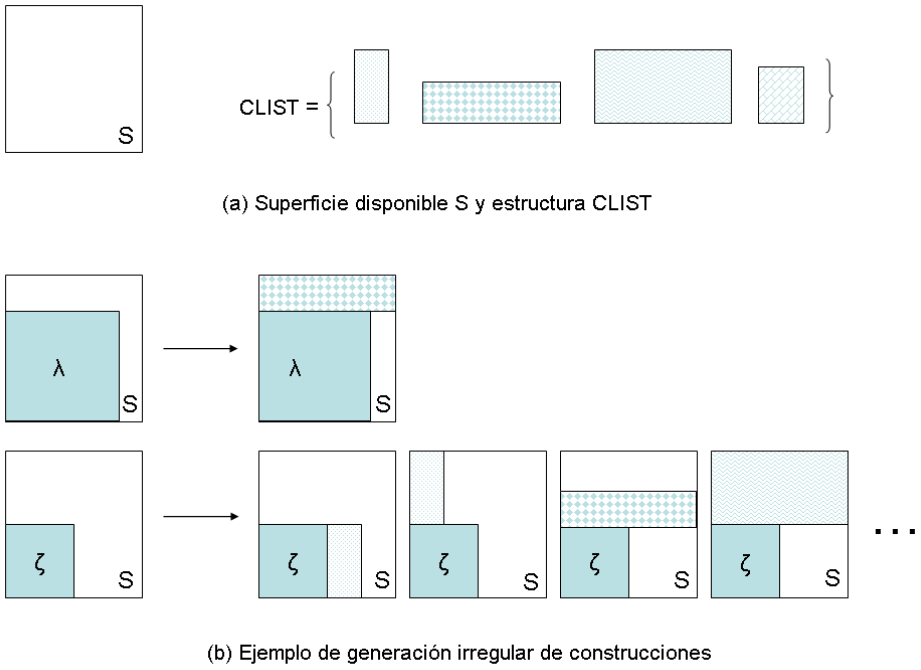

Figura 3.17: Generación irregular de meta-rectángulos

asociada con cada meta-rectángulo diferente. El meta-rectángulo λ sólo genera dos nuevas construcciones, mientras ζ genera al menos una nueva construcción cuando se combina con cada elemento T_i. Si las comunicaciones se hacen cada cierta cantidad de iteraciones de búsqueda, muchos procesos pasarán la mayoría del tiempo sin hacer nada mientras que unos poco trabajan intensamente. La otra posible alternativa es introducir comunicaciones cada intervalo de tiempo fijo $tPeriod$. La idea más sencilla para implementar tal propuesta involucra la comprobación de un temporizador cada iteración del bucle de búsqueda. El inconveniente de esta aproximación es que, como se ha mencionado anteriormente, la carga de trabajo varía considerablemente de una iteración a otra, así que las comprobaciones del temporizador no se producirán por igual entre los procesadores. Si la carga de trabajo asociado con cada iteración es altamente variable, el tiempo invertido en cada iteración variará también considerablemente. Por lo tanto, la frecuencia con la que cada procesador comprueba su temporizador para decidir si comunicar o no, puede ser muy irregular. Sin embargo, incrementar el periodo de tiempo de manera que se hayan hecho varias iteraciones de búsqueda antes de que se alcance el $tPeriod$, ayudará a compensar la carga entre iteraciones. Además, se ha probado otra alternativa de esquema de

sincronización basada en tiempo, independiente de la estructura de las iteraciones de búsqueda [116, 117], pero su complejidad no ha compensado la efectividad de la sincronización. Por esta razón, aquí se propone el uso de varios criterios juntos (por intervalos de tiempo e iteraciones) para hacer posible la mejora del balanceo de trabajo ejecutado por cada procesador entre los puntos de sincronización.

Considerando que en el algoritmo paralelo se llevan a cabo muchos pasos de comunicación, se ha diseñado un esquema dinámico y paramétrico de balanceo de carga, para balancear el trabajo equitativamente entre los procesadores. El tamaño de las estructuras OPEN no es un indicador exacto del esfuerzo computacional requerido para resolver el trabajo pendiente. No obstante, proporcionando una buena distribución de las tareas pendientes entre procesos se consigue un mejor comportamiento, porque cada proceso tiene suficiente trabajo pendiente para estar ocupado hasta el siguiente paso de comunicación. Este esquema de balanceo de carga o de carga compartida pretende que no hayan procesos sin hacer nada, redistribuyendo los elementos de las estructuras OPEN cuando algún proceso tiene menos trabajo que un cierto límite. El trabajo de redistribución normalmente ocurre antes de que cada proceso se desocupe, es decir, el esquema de carga compartida predice si algún proceso se está quedando sin trabajo y actúa redistribuyendo. Para llevar a cabo este balanceo se necesitan tres parámetros de configuración: *MinBalThreshold*, *MaxBalThreshold* y *MaxBalanceLength*. En cada paso del balanceo, el conjunto de procesadores se ordena por el tamaño de su OPEN. El procesador en posición i se asocia con otro que se encuentra en la posición $p - i - 1$. Así se une el procesador con más elementos en OPEN con el procesador con menos elementos en OPEN, el segundo con más elementos con el segundo con menos elementos, y así sucesivamente. Los procesadores asociados hacen un intercambio si el que tiene más elementos en la lista tiene más de *MaxBalThreshold* elementos y el otro tiene menos de *MinBalThreshold* elementos. El número de elementos que se intercambia es la mitad de la diferencia entre los dos tamaños de las listas OPEN, pero nunca puede ser mayor que *MaxBalanceLength*. El esquema intenta minimizar los nodos transferidos entre procesos, redistribuyendo el trabajo sólo cuando sea necesario, intentando minimizar el tiempo que un proceso pasa sin trabajo pendiente.

Como el algoritmo paralelo está basado en un esquema completamente distribuido donde se necesita un punto de comunicación cada cierto intervalo de tiempo o iteraciones, la propuesta inicial ha sido implementada en MPI. Sin embargo, es beneficioso comparar la propuesta inicial en MPI con una basada en OpenMP. Para una implementación con OpenMP se sabe que la comunicación de los datos debe realizarse mediante el uso de estructuras estáticas. Así que, la estructura general del Algoritmo 3 se mantiene, pero la comunicación de datos de sincronización se realiza a través de memoria compartida. Para la comunicación de todos con todos

(línea 5), el empaquetado de datos, envío y recepción, y desempaquetado se reemplaza por la escritura, en una estructura estática en memoria compartida, de los datos siguientes: mejor valor de la solución, tamaño de la lista OPEN, y conjunto de mejores construcciones analizadas. Para compartir el conjunto de construcciones analizadas, éstas deben ser mapeadas en una estructura de datos estática.

3.5.3. Resultados computacionales

Las implementaciones paralelas se han realizado partiendo del algoritmo secuencial con las reglas de detección de patrones dominados y duplicados. Para analizar su comportamiento, el estudio se ha llevado a cabo con las mismas instancias usadas para probar el algoritmo secuencial, disponibles en [56, 90] y empleadas en muchos estudios relacionados [7, 37, 59, 116]. Las pruebas se han realizado en un nodo SMP NovaScale 6320, que soporta hasta 32 procesadores Intel® Itanium 2 a 1.5GHz. Los compiladores usados han sido: *gcc 3.4.6*, *icc 9.1*, y *mpicc* para MPI Bull *1.6.5*. Para cada experimento se han realizados diez ejecuciones y se han considerado los valores medios. El tiempo computacional se muestra siempre en segundos.

Dado que ya se ha demostrado la validez de la propuesta secuencial, ahora el estudio se centrará en el comportamiento de las distintas aproximaciones paralelas, comparando además el efecto de las reglas de detección de patrones dominados y duplicados.

La Tabla 3.5 muestra el tiempo de ejecución obtenido cuando ambos algoritmos paralelos, de grano fino (Sección 3.5.1), y de grano grueso (Sección 3.5.2), se ejecutan con 1, 2, 4, 8 y 16 procesadores. En este experimento inicial, se ha usado la implementación original de cada algoritmo, es decir, la implementación OpenMP para el de grano fino y MPI para el de grano grueso. Para las ejecuciones de grano grueso el parámetro de configuración $tPeriod$ se ha fijado a 0.1 segundos, el $iPeriod$ a 50, $MaxBalThreshold$ a 100, $MinBalThreshold$ a 10, y $MaxBalLength$ a 45. Se han usado los mismos parámetros de configuración para todas las instancias del problema, es decir, el algoritmo no se ha configurado individualmente para cada instancia. Además, ya que la detección de nodos dominados y duplicados afecta seriamente a la estructura del espacio de búsqueda, las dos versiones de los algoritmos paralelos se han comprobado sin considerar la detección de dominancias e incluyéndolas todas.

Se ha podido comprobar una vez más que cuando se introducen las comprobaciones de las dominancias, los tiempos de ejecución para la mayoría de los problemas se reduce drásticamente. Incluso así, en los casos en los que el tiempo de ejecución no es demasiado corto, los algoritmos paralelos son capaces de obtener una aceleración aceptable. Comparando las dos aproximaciones paralelas, se observa que la de grano grueso es capaz de escalar mejor cuando el número de procesadores se

Proc.	Sin Dominancias		Todas Dominancias	
	Fino OpenMP	Grueso MPI	Fino OpenMP	Grueso MPI
ATP33s				
1	4190.82	4420.71	19.3	19.53
2	3361.66	2617.68	9.29	10.82
4	2141.4	1410.84	5.43	5.88
8	1919.88	789.67	7.04	3.4
16	1678.36	394.66	5.61	1.99
Hchl2				
1	2733.7	3256.23	198.86	167.56
2	2463.14	2004.98	164.37	103.54
4	1867.8	1273.17	141.08	54.44
8	1610.41	1050.85	169.35	33.25
16	1622.06	767.03	149.59	18.2
CW6				
1	782.65	907.19	42.25	44.36
2	238.53	476.89	30.18	26.46
4	66.3	239.18	25.2	15.33
8	32.75	131.8	45.32	9.76
16	22.89	73.23	45.69	6.0
CL_07_50_09				
1	264.48	249.37	3.82	4.16
2	170.99	137.05	3.28	2.58
4	123.94	79.17	2.79	1.31
8	138.9	45.99	2.99	0.87
16	110.11	23.68	2.84	0.58

Tabla 3.5: Dominancias en los algoritmos paralelos de grano fino y grueso

incrementa, incluso cuando las ejecuciones no son demasiado largas. Sin embargo, la implementación de grano fino muestra dificultades cuando el número de procesadores se incrementa y cuando las ejecuciones son demasiado cortas. Este hecho está directamente relacionado con el grano asociado a cada propuesta paralela y a la distribución de carga de trabajo obtenida durante las paralelizaciones.

Dentro de las implementaciones de los algoritmos se le asigna un nombre diferente a los nodos dependiendo de su estado. Así, los *nodos generados* son los nodos que representan cualquier construcción creada durante el proceso de búsqueda. Los nodos que se borran de la lista OPEN para combinarse con los mejores subproblemas previos ya explorados son los *nodos computados*. Finalmente, los *nodos insertados* representan los nodos generados que se insertan en OPEN. De este modo, la Figura 3.18 muestra la distribución de los nodos generados para la aproximación de grano fino en OpenMP, puesto que son los nodos que se encuentran implicados en el cómputo del bucle paralelizado donde se lleva a cabo la generación de nuevas construcciones. Además, también se muestra el balanceo de los nodos computados

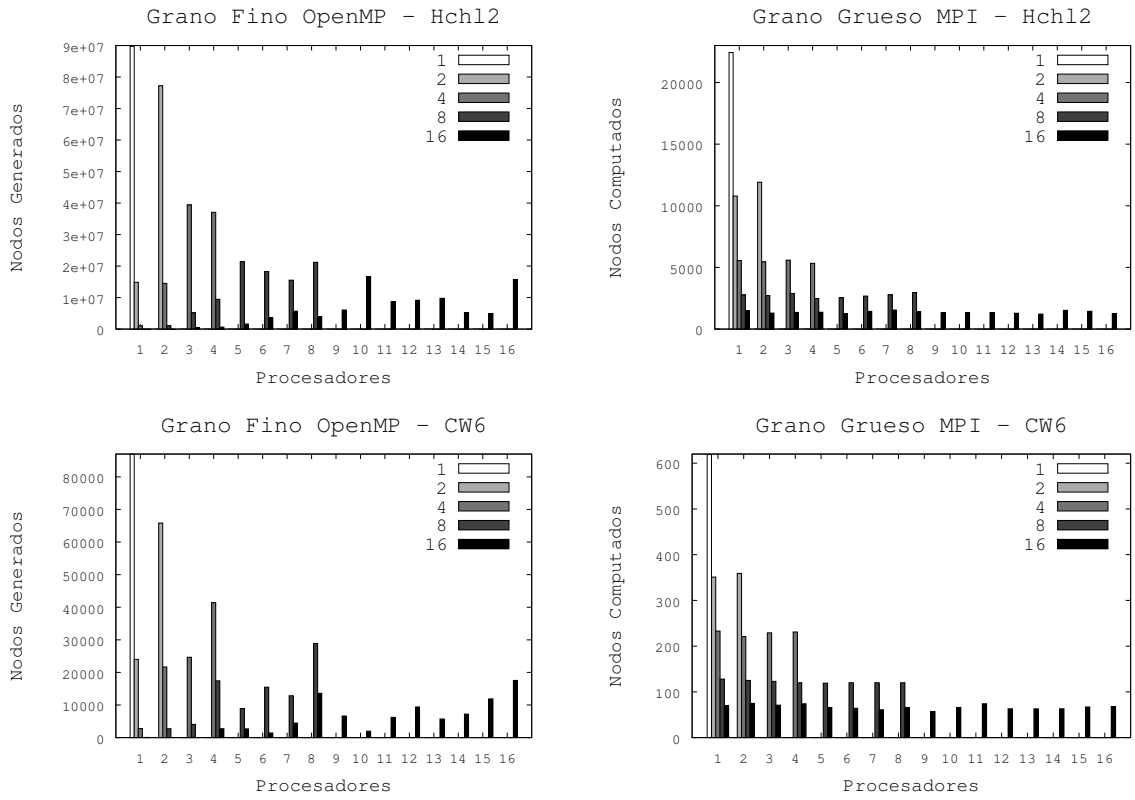

Figura 3.18: Balanceo de carga: grano fino OpenMP vs. grano grueso MPI

para el esquema de grano grueso en MPI, ya que son los nodos implicados en esta paralelización. Así es posible hacernos una idea de la distribución de carga en cada esquema. Los resultados pertenecen a dos instancias del problema (**Hchl2** y **CW6**) ejecutadas con 1, 2, 4, 8 y 16 procesadores. Se puede observar que el esquema de balanceo de carga introducido en la paralelización de grano grueso permite una mejor distribución del trabajo entre los procesadores. En las gráficas de la derecha las construcciones computadas son más homogéneas para los dos procesadores involucrados en la ejecución de dos procesadores, para los cuatro procesadores involucrados en la ejecución de cuatro procesadores, y así para el resto. Sin embargo, la distribución de la carga para la aproximación de grano fino depende de las características de los patrones de corte, que pueden ser considerablemente diferentes de una iteración a la otra. Es más difícil obtener una distribución justa de la carga de trabajo cuando colaboran en la búsqueda un número mayor de hilos, porque no hay suficiente trabajo para distribuir y los subproblemas no están uniformemente colocados por toda la estructura CLIST.

Dado que ha mostrado un comportamiento más prometedor, se ha estudiado en profundidad el algoritmo de grano grueso. Se han desarrollado dos implementaciones para el algoritmo, la original basada en MPI y otra usando OpenMP. Estas dos implementaciones del mismo algoritmo se han comparado, considerando las sincronizaciones por tiempo y por iteraciones, *tPeriod* y *iPeriod*. Los parámetros del balanceo de carga se han fijado a los mismos valores usados en los tests iniciales.

Aunque el comportamiento de cada una de las aproximaciones depende de los parámetros de sincronización, también varía según la instancia del problema. Es decir, la mejor configuración para la sincronización en la aproximación OpenMP depende del problema dado, y generalmente es diferente de la mejor configuración para la implementación MPI para un problema particular. Además, como se ha mencionado en la Sección 3.5.2, el comportamiento óptimo del algoritmo se consigue cuando se combinan ambos parámetros: tiempo e iteraciones. Para la combinación de estos parámetros se ha realizado un amplio ajuste en un análisis inicial. En las Tablas 3.6 y 3.7 se muestran algunos subconjuntos de las mejores combinaciones de parámetros para las implementaciones en OpenMP y MPI, respectivamente. Para cada ejecución, se presenta el tiempo total invertido en las sincronizaciones y el tiempo total de búsqueda. La mejor configuración del intervalo de iteración es la misma para ambas propuestas, pero con respecto al intervalo de sincronización, OpenMP se comporta mejor con intervalos menores que MPI.

En general, las ejecuciones OpenMP involucran un tiempo de búsqueda menor, mejorando el comportamiento obtenido con la implementación MPI. Sólo cuando el problema es simple y el número de hilos se incrementa, la aproximación OpenMP no es capaz de mejorar los resultados obtenidos con MPI. Inicialmente se pensaba que esto se podía deber a la carga adicional introducida por las sincronizaciones, pero considerando los tiempos de sincronización mostrados en las columnas "*Sinc.*" de las tablas, el tiempo medio que cada proceso pasa realizando sincronizaciones es muy similar para ambas aproximaciones. Sin embargo, en un análisis más profundo de los algoritmos, se ha comprobado que en la propuesta OpenMP todos los procesos normalmente llegan a los puntos de sincronización al mismo tiempo, por lo que la desviación entre los tiempos de sincronización es casi insignificante cuando se compara con los de la implementación MPI.

3.5.4. Conclusiones

Partiendo del algoritmo secuencial ya implementado (basado en el algoritmo MVB) con las reglas de detección de patrones dominados y duplicados, se han propuesto dos esquemas paralelos para afrontar las instancias más complejas del

Iter.	Tiempo	Procesadores							
		1		2		4		8	
		Sinc.	Búsqueda	Sinc.	Búsqueda	Sinc.	Búsqueda	Sinc.	Busqueda
ATP33s									
10	0.3	0.000	19.264	1.246	10.097	0.994	5.240	0.990	3.526
	0.6	0.000	19.092	1.315	10.229	1.012	5.298	1.765	4.363
15	0.3	0.000	19.028	0.292	9.346	0.244	4.666	0.320	2.767
	0.6	0.000	19.081	1.115	9.968	1.129	5.514	1.800	4.573
Hchl2									
10	0.3	0.006	183.719	17.329	101.984	11.225	47.638	8.530	30.890
	0.6	0.005	184.880	17.114	97.595	11.437	48.402	8.032	30.397
15	0.3	0.004	184.865	15.616	93.245	10.762	47.951	7.638	29.179
	0.6	0.004	183.119	15.479	92.043	11.581	49.041	8.233	30.273
CW6									
10	0.3	0.000	43.183	0.723	22.953	0.713	13.088	0.929	8.774
	0.6	0.000	43.184	1.309	23.690	1.243	14.245	1.387	9.806
15	0.3	0.000	43.133	0.848	23.429	0.710	12.697	0.797	8.235
	0.6	0.000	43.213	0.471	22.912	0.454	13.483	1.177	9.432
CL_07_50_09									
10	0.3	0.000	4.037	0.321	2.084	0.189	1.103	0.182	0.769
	0.6	0.000	4.030	0.312	2.065	0.189	1.093	0.146	0.749
15	0.3	0.000	4.024	0.247	2.004	0.209	1.200	0.149	0.768
	0.6	0.000	4.012	0.189	1.913	0.168	1.163	0.198	0.828

Tabla 3.6: Algoritmo de grano grueso: implementación OpenMP

Iter.	Tiempo	Procesadores							
		1		2		4		8	
		Sinc.	Búsqueda	Sinc.	Búsqueda	Sinc.	Búsqueda	Sinc.	Búqueda
ATP33s									
10	1.5	0.005	19.048	0.670	10.766	0.729	6.010	0.681	3.625
	2.0	0.005	18.953	0.680	10.859	0.730	6.007	0.652	3.595
15	1.5	0.004	18.929	0.740	10.638	0.827	6.247	0.573	3.338
	2.0	0.004	18.956	0.769	10.732	0.858	6.322	0.556	3.307
Hchl2									
10	1.5	0.145	162.108	12.108	111.075	12.607	63.422	9.322	35.926
	2.0	0.146	162.011	12.086	111.326	12.837	64.514	9.324	35.872
15	1.5	0.116	166.109	12.870	115.838	11.851	63.919	7.773	33.496
	2.0	0.117	162.728	12.815	115.474	11.620	63.400	7.805	33.572
CW6									
10	1.5	0.005	43.163	2.841	26.811	3.196	17.856	3.541	11.909
	2.0	0.005	42.880	2.852	26.897	3.193	17.813	3.533	11.886
15	1.5	0.004	43.143	2.053	26.642	2.304	18.261	2.357	11.028
	1.0	0.004	42.918	2.064	26.852	2.308	18.280	2.596	11.347
CL_07_50_09									
10	1.5	0.015	4.068	0.129	2.101	0.136	1.198	0.102	0.650
	2.0	0.014	4.086	0.130	2.095	0.151	1.290	0.103	0.650
15	1.5	0.013	4.034	0.121	2.096	0.110	1.270	0.092	0.672
	2.0	0.013	4.024	0.123	2.102	0.109	1.270	0.093	0.674

Tabla 3.7: Algoritmo de grano grueso: implementación MPI

problema. La primera propuesta está basada en un modelo de grano fino que ejecuta en paralelo los bucles de generación de construcciones. La segunda está basada en un modelo de grano grueso que realiza la ejecución paralela del bucle de búsqueda e introduce esquemas de sincronización eficientes y un sistema para el balanceo de carga. Los diferentes esquemas de sincronización propuestos están basados en iteraciones de búsqueda, en intervalos de tiempo o en la combinación de ambos.

Con el esquema de grano fino, la cantidad de trabajo que se puede paralelizar es pequeña. Además, la carga real de trabajo depende del número de construcciones almacenadas en cada posición de la matriz CLIST, y normalmente las construcciones no están uniformemente distribuidas en la estructura. Por lo tanto, los resultados obtenidos con la aproximación de grano fino no son suficientemente buenos debido al poco trabajo que se paraleliza en relación al cómputo global y a la distribución irregular de la carga de trabajo. Sin embargo, con el esquema de grano grueso el cómputo paralelizado es mayor, y a pesar de esta estructura computacional tan irregular y de las dificultades de romper la naturaleza secuencial de las búsquedas primero-el-mejor, la combinación de la metodología de sincronización con el uso de estrategias de balanceo de carga proporciona una distribución del trabajo justa y una aceleración lineal.

Por otra parte, la comparativa entre las implementaciones en OpenMP y MPI del algoritmo de grano grueso muestra que, en general, las ejecuciones con OpenMP involucran un tiempo de búsqueda menor, mejorando el comportamiento obtenido con la implementación MPI. Sólo cuando el problema es simple y el número de hilos se incrementa, la aproximación OpenMP no es capaz de mejorar los resultados obtenidos con MPI.

De cualquier modo, se ha podido demostrar que incluso para los algoritmos con una estructura secuencial inherente es posible diseñar un esquema paralelo adecuado capaz de mejorar la eficiencia de la aproximación. Finalmente, es importante destacar que la eficiencia de los esquemas paralelos principalmente depende del grano de trabajo y de la distribución de la carga, aunque la selección de una herramienta de programación paralela adecuada también es decisiva.

3.6. Solución multi-objetivo

Tal y como ya se ha mencionado, en algunos campos de la industria, el material a cortar puede ser muy barato o fácilmente reciclable, por lo que otro criterio importante a tener en cuenta durante el proceso de generación de los patrones podría ser la velocidad a la que se pueden obtener los piezas, minimizando el tiempo de producción. Normalmente esta velocidad viene dada por las características de la

maquinaria industrial de corte, pero, en general, está determinada por el número de cortes necesarios para generar las piezas. Además, el número de cortes es crucial para la vida de la maquinaria industrial. Por estas razones aquí se ha tomado como segundo objetivo minimizar el número de cortes necesarios para obtener las piezas.

Aplicar un método exacto para la resolución de un problema multi-objetivo es inviable debido a la magnitud y complejidad del árbol de búsqueda. Además, si se optara por abordar un problema multi-objetivo tratando de unificar los distintos objetivos en una única función objetivo, se deberían conocer a priori las demandas específicas en cada momento en cuanto a los dos objetivos, para poder combinarlos de la mejor manera. Si las demandas cambian, se necesitarían nuevas ejecuciones. Usando estrategias que afronten el problema multi-objetivo como tal, sólo se requiere una ejecución, y después la persona encargada de tomar las decisiones puede seleccionar la solución más adecuada, dependiendo de las demandas en cada momento. Si entre las metaheurísticas para optimización multi-objetivo se selecciona un MOEA, simplemente hay que emplear un esquema general del algoritmo y definir las características del problema: cómo se representa el individuo, cómo se genera la población inicial, cómo se evalúan los objetivos, y qué operadores evolutivos se utilizarán. Normalmente, el sujeto interesado en resolver el problema tiene profundos conocimientos acerca del problema, y no tantos sobre las implementaciones de métodos de resolución, por lo que esta alternativa supone una buena elección para resolver un problema multi-objetivo.

Por lo tanto, en este caso, se ha optado por la utilización de MOEAs para resolver el MOCSP. En general, tanto para optimización mono-objetivo como para optimización multi-objetivo, cuando se diseñan propuestas metaheurísticas para la solución de un problema dado, hay muchas decisiones de diseño que son cruciales para el impacto del método de resolución. En este sentido, una de las decisiones principales concierne a una de las características del problema que hay que definir: la representación de las soluciones que conforman la población. Según [24], se pueden diferenciar distintos grupos de propuestas metaheurísticas para los problemas de corte y empaquetado, dependiendo del tipo de codificación elegida para la representación de las soluciones:

- Es típico encontrar una solución codificada que indica una secuencia de colocación de piezas. La búsqueda metaheurística se lleva a cabo dentro del espacio de soluciones codificadas y normalmente involucra operadores independientes del problema. Una rutina de colocación o decodificación transforma las soluciones codificadas en una distribución completa de piezas.

- Una alternativa inmediata consiste en combinar la codificación de la solución con el uso de heurísticas de decodificación [112]. Mientras las soluciones

codificadas ya contienen, hasta cierto punto, la distribución o información geométrica, se requiere una rutina adicional de colocación para el posicionamiento final. La codificación de las soluciones normalmente es específica del problema, que suele estar basado en grafos, y el uso de los correspondiente operadores específicos del problema.

- El último tipo de algoritmo no utiliza codificación. La búsqueda se lleva a cabo directamente en el espacio completo de distribuciones, que son manipuladas usando operadores específicos [21, 157].

A continuación, se exponen dos tipos diferentes de esquemas de codificación implementados para la resolución del MOCSP mediante MOEAs: un esquema directo utilizado por el tercer grupo de metaheurísticas, y varios esquemas de codificación basados en hiperheurísticas, es decir, utilizados por el segundo grupo.

3.6.1. Esquemas de codificación directa

Se ha hecho uso de una representación que implica una definición completa de la distribución guillotinable de piezas. De este modo, la búsqueda se lleva a cabo directamente en el espacio de todas las posibles distribuciones y los operadores de variación deben ser diseñados específicamente para manipular tal representación. Se han planteado dos esquemas de codificación directa. Aunque ambos se basan en una notación postfija, la manera de manejar la generación es diferente en cada caso. La primera propuesta se llamará esquema de codificación de *tamaño controlado* y la segunda, esquema de codificación de *tamaño completo*.

3.6.1.1. Representación

Se ha usado la notación postfija introducida en la Sección 3.3 para representar las soluciones candidatas [148, 171]. Los operandos son los identificadores de las piezas, mientras que los operadores son 'V' y 'H' haciendo referencia a las concatenaciones en vertical y horizontal (Figura 3.19). Tal y como se ha mencionado anteriormente, usando esta representación todas las distribuciones de piezas que se obtienen pueden cortarse usando cortes del tipo guillotina.

En este problema, para constituir un cromosoma válido cada pieza de tipo T_i no puede aparecer más de b_i veces. Para la generación inicial de individuos, se ha establecido un orden aleatorio de piezas y se ha aplicado una probabilidad uniforme para determinar los operadores.

Usando el esquema de codificación de *tamaño controlado*, se introducen piezas en la solución hasta que la aplicación de un operador produzca un patrón que no quepa

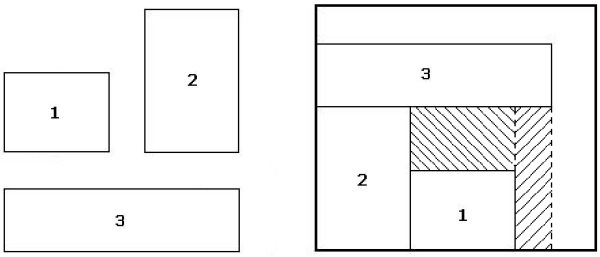

Figura 3.19: Distribución de patrones para el cromosoma '2 1 H 3 V'

en la hoja de material, es decir, hasta que el cromosoma no satisfaga las restricciones de ancho y largo de la materia prima. Cuando se produce esta situación, ese operador que ha producido un patrón que no cabe en el material se cambia por el opuesto, es decir, si es un operador 'H' se cambia por 'V' y viceversa. Esto se realiza con la intención de que quepan más piezas en el material y se mejore la solución. Si incluso con este cambio la combinación de piezas no encaja en el material, la última pieza que se ha intentado colocar se intercambia por la siguiente del cromosoma y se prueban de nuevo los dos operadores. Este proceso de intercambio de la última pieza con las siguientes se repite hasta que se obtenga una solución válida o hasta que se haya aplicado un número máximo de cambios. Si incluso así no se consigue una solución válida, se aplica el mismo método con la pieza anterior a la última que se había intentado colocar. Finalmente, si este procedimiento no funciona, el cromosoma se corta por el tamaño correcto y se aplica un operador de relleno.

Usando el esquema de codificación de *tamaño completo*, el tamaño final del cromosoma se determinará en la evaluación de los objetivos, donde se comprobará hasta dónde encaja la construcción en el material. De este modo, para la generación inicial de individuos, cada individuo se crea del máximo tamaño posible (considerando que todas las piezas se van a colocar en el material).

En la Figura 3.20 se muestra un ejemplo de generación de un cromosoma para la codificación directa de *tamaño controlado*, donde se fija el tamaño del cromosoma cuando ya no caben más piezas en el material; y otro ejemplo para la de *tamaño completo*, donde el cromosoma se crea hasta el final aunque el patrón no quepa en el material y ya luego en la evaluación se verá hasta dónde encaja.

3.6.1.2. Evaluación de los objetivos

Como ya se ha comentado, cada cromosoma representa una cierta distribución de piezas sobre la materia prima. La codificación usada proporciona la información de cómo se deben combinar o colocar las piezas sobre el material. En base a esta infor-

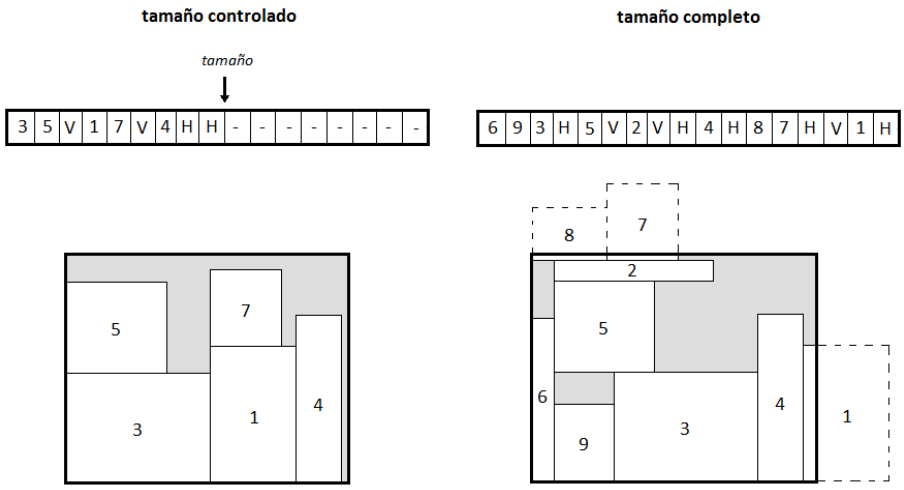

Figura 3.20: Ejemplo de codificaciones de tamaño controlado y completo

mación, se pueden evaluar ambos objetivos de optimización, beneficio total (como la suma de los beneficios de todas las piezas que se colocan en el material) y número de cortes necesarios. Con este propósito, los métodos que se han aplicado aquí se basan en el uso de pilas y de la notación postfija que representa al cromosoma [148].

Para la evaluación del segundo objetivo (número de cortes requeridos) se ha aplicado un método iterativo. El cromosoma se recorre de izquierda a derecha, interpretando cada elemento y creando las construcciones indicadas para calcular los anchos y largos parciales. Por cada combinación horizontal o vertical implicada se necesita al menos un corte. Si los rectángulos combinados no coinciden en largo (para las construcciones verticales) o en ancho (para las construcciones horizontales), se requiere un corte extra para la construcción. Al final del proceso, se obtiene el patrón final completo. En este caso, el valor del primer objetivo (beneficio total) viene dado inmediatamente por el beneficio del patrón final resultante. Este procedimiento se detalla en el Algoritmo 4.

Para la aproximación de *tamaño completo*, el algoritmo debe incorporar las correspondientes comprobaciones para asegurar que la evaluación de los cromosomas siempre satisfaga las restricciones de ancho y largo del material. Cuando el ancho o el largo se excede, el operador correspondiente se intercambia por el operador contrario. Si incluso así, la construcción excede las dimensiones de la materia prima, el tamaño del cromosoma se fija al tamaño de la construcción previa que sí cabe en el material. Se trata de un intento de encajar más piezas dentro del material, mediante un proceso más simple que el aplicado en la generación inicial de individuos con la aproximación de *tamaño controlado*. Estas comprobaciones no son necesarias para

Algoritmo 4 Evaluación de los objetivos del 2DCSP

```
1: cutsNum = 1;
2: for (i = 0 hasta sizeOfChromosomeVbles) do
3:    if (isOperator(chromosome(i)) == false) then
4:       stack.push(chromosome(i));
5:    else
6:       cutsNum = 1 + cutsNum;
7:       pieceA = stack.pop();
8:       pieceB = stack.pop();
9:       profit(newPiece) = profit(pieceA) + profit(pieceB);
10:      if (chromosome(i) == H) then
11:         length(newPiece) = length(pieceA) + length(pieceB);
12:         width(newPiece) = max(width(pieceA), width(pieceB));
13:         if (width(pieceA) != width(pieceB)) then
14:            cutsNum = cutsNum + 1;
15:         end if
16:      else
17:         length(newPiece) = max(length(pieceA), length(pieceB));
18:         width(newPiece) = width(pieceA) + width(pieceB);
19:         if (length(pieceA) != length(pieceB)) then
20:            cutsNum = cutsNum + 1;
21:         end if
22:      end if
23:      stack.push(newPiece);
24:   end if
25: end for
26: finalPiece = stack.pop();
27: totalProfit = profit(finalPiece);
28: if (width(finalPiece) != width(motherSheet)) then
29:    totalCuts = cutsNum + 1;
30: else
31:    totalCuts = cutsNum;
32: end if
```

la evaluación de los objetivos con la aproximación de *tamaño controlado*, puesto que esta codificación asegura que el tamaño del cromosoma es adecuado en el momento en que se van a evaluar los objetivos.

3.6.1.3. Operadores

Se ha usado una codificación que representa implícitamente las soluciones del problema, por lo que el tipo de operadores que se aplique debe tratar con sus características específicas. Como operador de cruce, se han testeado varias alternativas y trás un análisis del comportamiento finalmente se ha seleccionado el *Partially Map-*

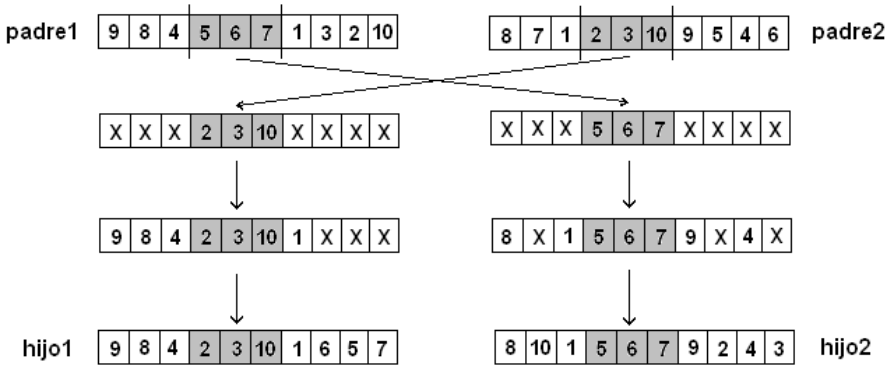

Figura 3.21: Ejemplo para Partially Mapped Crossover

ped Crossover (PMX) [86]. La técnica se basa en la recombinación de dos cadenas de cromosomas donde sólo se tiene en cuenta la información de las piezas, es decir, los operadores 'H' o 'V' no se consideran para la aplicación de este tipo de cruce. En primer lugar, se eligen aleatoriamente dos puntos de cruce dentro de cada padre (Figura 3.21). Luego, los segmentos de los padres entre los puntos de cruce se intercambian para generar la descendencia. El resto de las cadenas de los hijos se obtienen por medio de mapeo entre los dos padres. Si un valor del cromosoma fuera del segmento de intercambio no está contenido en el segmento intercambiado, permanece igual, pero si ya está contenido, debe ser reemplazado por el valor contenido en el segmento original del cromosoma que no esté contenido en el nuevo segmento en consideración.

Por otra parte, la mutación aplicada [148, 171] funciona de la siguiente manera (Figura 3.22): Primero se eligen aleatoriamente dos elementos del cromosoma p_1 y p_2. Ambos elementos representan números de piezas u operadores y p_1 está más cerca de la izquierda en el cromosoma. Si ambos son números de piezas u operadores, o p_1 es un operador y p_2 es una pieza, se intercambian. Si p_1 es un número de pieza y p_2 es un operador, sólo se intercambian cuando, después de realizar el cambio, se siga manteniendo para cada operador la condición de la fórmula 3.2 para que la notación sea válida. Luego, se elige aleatoriamente un operador del cromosoma y se intercambia de 'H' a 'V' o viceversa, siempre bajo la probabilidad de mutación.

Cuando se usa el esquema de codificación de *tamaño controlado*, después de la aplicación del cruce y la mutación, se aplica un operador de reparación para asegurar que sólo se van a generar nuevas soluciones válidas. Este operador corta el cromosoma por el tamaño adecuado. Además, dependiendo de una probabilidad, el

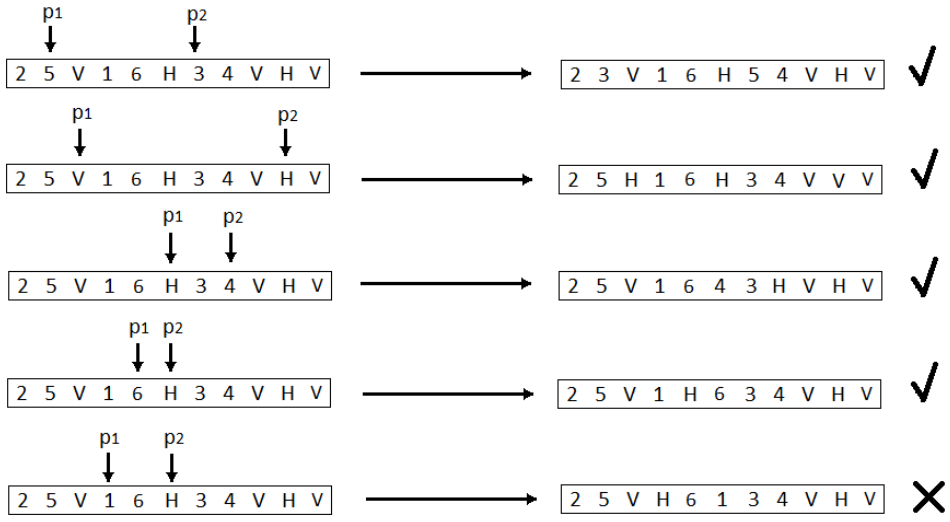

Figura 3.22: Ejemplos de mutación en la codificación directa

cromosoma se recorre de izquierda a derecha intercambiando cada operador y cada pieza (por las piezas aún disponibles), comprobando que en cada paso la combinación de piezas encaja en el material y proporciona mejor beneficio y menor número de cortes que el cromosoma original. En tal caso, el cromosoma original se reemplaza por el nuevo mejorado. Por último, como en la generación inicial, se aplica un operador de relleno intentando llenar el espacio sobrante añadiendo piezas en horizontal o vertical al final del patrón.

El operador de relleno (Figura 3.23) concatena piezas verticalmente desde la más larga hasta la más corta (área y) para rellenar el hueco a la derecha del patrón final p. Luego, concatena horizontalmente piezas de mayor a menor ancho (patrón x) para rellenar el hueco por arriba del partrón final p. Por último, el patrón resultante se obtiene por concatenación de p con x en vertical, y después en horizontal con y, es decir, el patrón final es $p\ x\ V\ y\ H$.

Si se usa el esquema de *tamaño completo*, el final del cromosoma se determinará en la evaluación de los objetivos, que comprueba hasta qué punto encaja la construcción en el material. Por lo tanto, después de la aplicación de los operadores de mutación y cruce, no es necesario controlar el tamaño del cromosoma. Así que no se llama al operador de reparación ni de relleno.

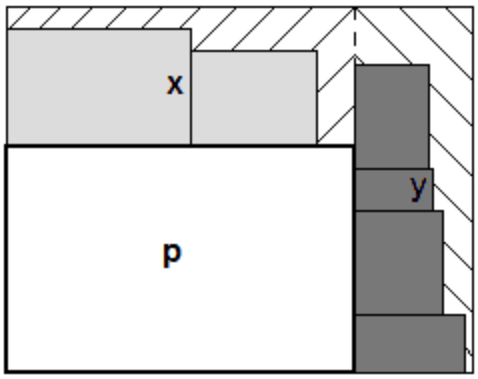

Figura 3.23: Operador de relleno

3.6.2. Esquemas de codificación basados en hiperheurística

Como se comentó al inicio de la sección, otra posibilidad a la hora de representar un individuo de la población consiste en combinar la codificación de la solución con el uso de heurísticas de decodificación. Así mientras las soluciones codificadas contienen la información de la distribución de las piezas, se requiere un procesado de dicha solución aplicando las correspondientes heurísticas para determinar la construcción final.

Tal y como ya se ha mencionado, se puede encontrar una amplia variedad de heurísticas para resolver el 2DCSP en la bibliografía, pero sólo unas pocas tratan con la restricción de producir patrones guillotinables. En este caso, el interés se centrará en las heurísticas de colocación para proponer un esquema de codificación hiperheurístico que las elija y combine de la mejor manera [33]. Se trata por lo tanto de una codificación indirecta que determina el tipo de heurística de colocación de piezas (o heurística de bajo nivel) a aplicar y el número de piezas que cada heurística debe distribuir.

3.6.2.1. Heurísticas de bajo nivel

La primera decisión necesaria para implementar el enfoque basado en hiperheurística es la definición del conjunto de heurísticas de colocación que se van a utilizar. Tal y como se ha mencionado anteriormente, la mayoría de las heurísticas propuestas en la literatura para resolver el 2DCSP tratan la versión no guillotina del mismo [14, 34, 100]. Sólo unas pocas tratan la versión guillotina, y los resultados que aportan no presentan la calidad de los relativos a la versión no guilloti-

na [72, 146, 155, 178] en cuanto a aprovechamiento de la materia prima. Sin embargo, existe un conjunto de heurísticas constructivas para la colocación de piezas, orientadas a niveles, que se pueden aplicar fácilmente al 2DCSP aquí tratado, debido a la manera en la que operan. Su funcionamiento se basa en la construcción de capas o niveles de piezas, es decir, las piezas se van colocando en el material en niveles dependiendo del espacio disponible en cada uno de ellos y del tamaño de las mismas. De este modo, como el problema consiste en hacer el mejor uso del material, se irán creando capas o niveles de piezas hasta llegar al límite correspondiente del mismo. Concretamente se han implementado las siguientes:

- *Next Fit Decreasing Height* (NFDH) [45]: la lista de piezas a obtener se ordena de mayor a menor largo. Las que tengan igual largo conservan su orden original entre ellas en la lista. Esta ordenación implica que el primer rectángulo colocado en un nivel determina el largo de ese nivel. Un rectángulo se coloca en el nivel actual alineado a la izquierda si cabe. Si no cabe, entonces se crea un nuevo nivel a la derecha del actual (que se convierte en el actual) y el rectángulo se coloca ahí. El proceso se realiza de izquierda a derecha para los niveles y de abajo a arriba dentro de un nivel. Para esta heurística nunca se revisan niveles anteriores al actual.

- *First Fit Decreasing Height* (FFDH) [45]: la lista de piezas también se ordena de mayor a menor largo y las que tengan igual largo conservan su orden original. Un rectángulo se coloca alineado a la izquierda en el menor nivel en que quepa. Para ello, se recorren los niveles creados de izquierda a derecha, y si la pieza actual no cabe en ninguno de los existentes, se crea un nuevo nivel a la derecha del último. La pieza se coloca en el nuevo nivel, convirtiéndose éste en el nivel actual. Por lo tanto, la diferencia entre esta heurística y la anterior es que ésta siempre mira los niveles anteriores antes de colocar una pieza, mientras que la anterior no lo hace.

- *Best Fit Decreasing Height* (BFDH) [143]: su funcionamiento es análogo al de la heurística FFDH, excepto que en este algoritmo cuando se recorren todos los niveles creados de izquierda a derecha para colocar cada pieza, se comprueba en cuáles de ellos cabe, y se coloca en aquel nivel en el que quepa dejando el menor espacio residual horizontal.

Para comenzar, se han utilizado estas tres heurísticas orientadas a niveles en la creación del esquema hiperheurístico. Posteriormente, se ha introducido en el esquema una heurística más, que utiliza otro concepto diferente al de capas o niveles, tratando de comprobar si se produce alguna mejora en el resultado final. Dicha

heurística ya fue creada para la solución exacta del 2DCSP, utilizándose como cota inferior del algoritmo. Esta heurística, a diferencia de las anteriores, no sólo tiene en cuenta el tamaño de las piezas, sino el beneficio asociado a cada una de ellas, ya que no necesariamente éste debe ser proporcional a su área. Imita el algoritmo de programación dinámica de Gilmore y Gomory [85], pero se sustituyen las combinaciones verticales y horizontales sin límites por otras factibles dentro de los límites. Sea $R = (r_i)_{i=1...n}$ y $S = (s_i)_{i=1...n}$ los conjuntos factibles de soluciones usando $r_i \leq b_i$ y $s_i \leq b_i$ rectángulos de tipo T_i. El producto cruzado $R \otimes S$ de R y S se define como el conjunto de soluciones factibles construidas partiendo de R y S sin violar las restricciones de los límites: es decir, $R \otimes S$ usa $(\text{mín}\{r_i + s_i, b_i\})_{i=1...n}$ rectángulos de tipo T_i. El resultado viene dado por el valor $H(L, W)$ computado por las siguientes ecuaciones:

$$H(x,y) = \text{máx} \begin{cases} \text{máx}\{g(S(x,y_1) \otimes S(x,y-y_1)) & \text{tal que } 0 < y_1 \leq \lfloor \frac{y}{2} \rfloor \} \\ \text{máx}\{g(S(x_1,y) \otimes S(x-x_1,y)) & \text{tal que } 0 < x_1 \leq \lfloor \frac{x}{2} \rfloor \} \\ \text{máx}\{c_i & \text{tal que } l_i \leq x \text{ y } w_i \leq y\} \end{cases}$$

siendo $S(x, y) = S(\alpha, \beta) \otimes S(\gamma, \delta)$ uno de los conjuntos de cruce, - o una construcción vertical $S(x, y_0) \otimes S(x, y - y_0)$ o una construcción horizontal $S(x_0, y) \otimes S(x - x_0, y)$ - de donde se obtiene el máximo previo, es decir, $H(x, y) = g(S(\alpha, \beta) \otimes S(\gamma, \delta))$.

3.6.2.2. Representación

Se ha diseñado una representación del cromosoma basada en hiperheurística para resolver el 2DCSP multi-objetivo. De este modo, se aplicará una secuencia de heurísticas de bajo nivel a unos conjuntos de piezas. Un individuo se representa mediante una secuencia de 3-tuplas (h, p, o), donde h es el identificador de la heurística de colocación de bajo nivel que se va a aplicar, p es el número de piezas que la heurística debe colocar en el material, y o determina el criterio de ordenación de las piezas (Figura 3.24). Los cromosomas tienen una longitud variable j, que va desde $j = 1$ (solo hay una 3-tupla (h, p, o) de manera que la misma heurística coloca todas las piezas) a $j = n$ (las n piezas se colocan independientemente, es decir, $\forall i \in [1, n], p_i = 1$). Los criterios básicos de ordenación usados son: largos decrecientes, anchos decrecientes, áreas decrecientes y perímetros decrecientes. Tal y como se ha explicado, originalmente las heurísticas NFDH, FFDH y BFDH utilizan el criterio de ordenación por largos decrecientes, ya que de esta manera a medida que se van colocando las piezas éstas pueden caber en el largo de los niveles ya creados. Alterar el criterio original modifica el funcionamiento de las heurísticas, pero esto puede proporcionar una mayor variedad de soluciones que enriquezcan el proceso de evolución.

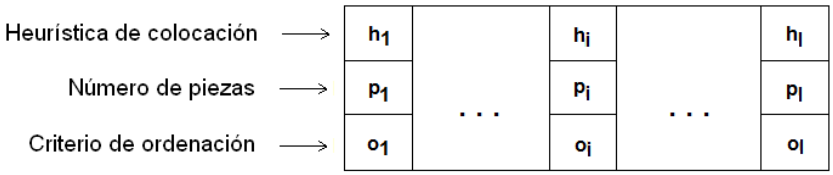

Figura 3.24: Representación basada en hiperheurística

Para la generación inicial de individuos, se crean 3-tuplas (h, p, o) hasta que se acaben las piezas. h se selecciona aleatoriamente entre las cuatro heurísticas de colocación disponibles, p se selecciona aleatoriamente en el intervalo $[1, a_p]$ donde a_p es el número de piezas aún disponibles, y finalmente o es uno de los criterios de ordenación generado aleatoriamente entre los cuatro disponibles.

3.6.2.3. Evaluación de los objetivos

En la codificación directa presentada en la sección previa, los cromosomas representan explícitamente la solución del problema, es decir, la distribución de las piezas sobre el material. Sin embargo, en esta codificación indirecta, los MOEAs utilizan una población de individuos representados por una hiperheurística, por lo que el cromosoma debe ser interpretado para obtener la solución del problema. La representación se analiza de izquierda a derecha, aplicando para cada 3-tupla (h, p, o), la heurística h para posicionar las siguientes p piezas. Estas piezas se distribuyen de acuerdo con sus correspondientes términos de ordenación. Cuando las piezas se colocan en la parte superior de niveles existentes, se crean construcciones verticales. Cuando se abre un nuevo nivel a la derecha de los existentes, se genera una combinación horizontal entre los niveles. La decodificación del cromosoma proporciona una solución del problema representada mediante una notación postfija de cortes verticales y horizontales. Entonces, esa solución se puede evaluar del mismo modo que en la sección previa, para obtener los valores de los dos objetivos: beneficio total y número de cortes necesarios. Antes de realizar dicha evaluación, se aplica el operador de relleno utilizado anteriormente con la codificación directa de *tamaño controlado* y explicado en la Sección 3.6.1.3. Es decir, se intenta rellenar el espacio que sobre con las piezas que no han sido colocadas por las heurísticas.

3.6.2.4. Operadores

Se han diseñado y probado varios operadores de cruce y mutación con la representación previa. Finalmente, se ha seleccionado un cruce o recombinación de

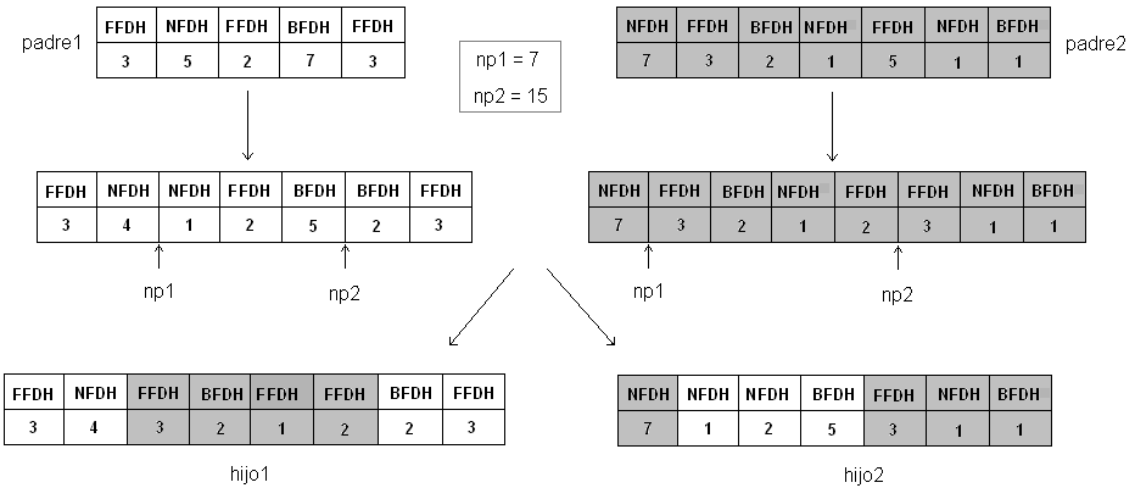

Figura 3.25: Operador de cruce de dos puntos

dos puntos que considera el número de piezas acumuladas dentro de la representación. Para ello se generan aleatoriamente dos números de piezas np_1 y np_2 , tal que $1 \leq np_1 < np_2 < n$. Para cada padre, es necesario encontrar las posiciones del cromosoma donde el número de piezas acumuladas sumen np_1 y np_2. Cuando la suma de las piezas no coincida con el valor generado np, la primera 3-tupla (h_i, p_i, o_i), que verifique $\sum_{m=1}^{i} p_m > np$ se divide en dos partes (h_j, p_j, o_j) y (h_k, p_k, o_k), tal que $h_j = h_k = h_i$, $o_j = o_k = o_i$, $p_k = (\sum_{m=1}^{i} p_m) - np$, y $p_j = p_i - p_k$ (Figura 3.25). Una vez se tienen identificados los dos puntos de cruce dentro de ambos padres, los pares centrales de los mismos se intercambian para generar los dos nuevos hijos.

De los operadores de mutación probados, el que ha mostrado mejor comportamiento está basado en la aplicación de tres tipos diferentes de movimientos en el cromosoma:

- *Añadir*: aleatoriamente selecciona una 3-tupla (h_i, p_i, o_i) dentro de la representación. Entonces se genera una nueva 3-tupla (h, p, o) donde h es una heurística seleccionada al azar, o es un criterio de ordenación seleccionado al azar, y p es un número aleatorio del intervalo $[1, p_i]$. Luego, p_i se actualiza con el valor $p_i - p$. Si después de actualizar, $p_i \neq 0$, las 3-tuplas de las posiciones $i, \ldots, j$ se desplazan una posición a la derecha, incrementando el largo total del individuo $(j = j + 1)$. Finalmente, la nueva 3-tupla se introduce en la posición i (Figura 3.26). En caso contrario, la nueva 3-tupla sustituye a la previa y no es necesario realizar el desplazamiento.

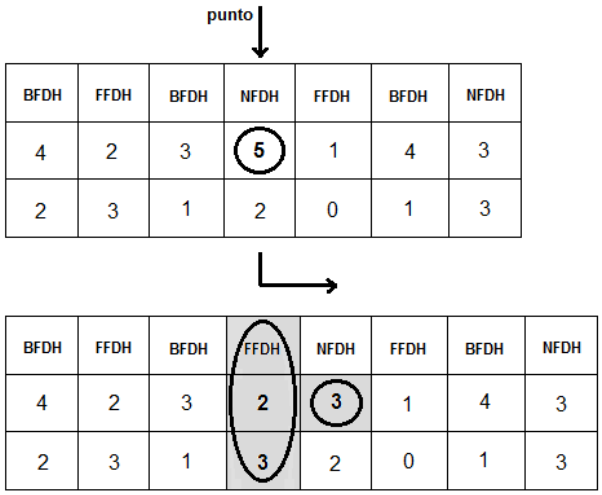

Figura 3.26: Operador de mutación *Añadir*

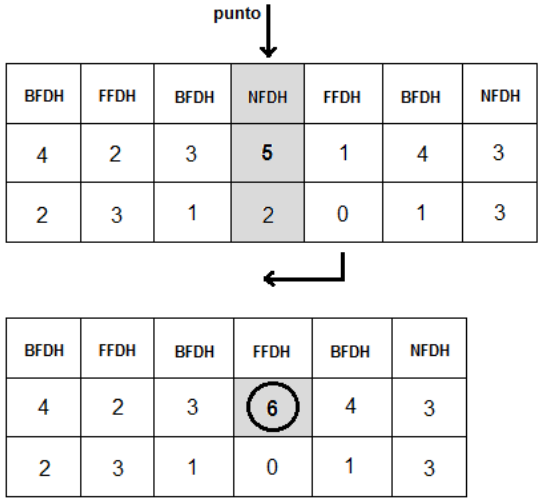

Figura 3.27: Operador de mutación *Eliminar*

- *Eliminar*: aleatoriamente selecciona una 3-tupla (h_i, p_i, o_i) de la representación. Si la 3-tupla seleccionada es la última de la representación $(i = j)$, entonces $p_{i-1} = p_{i-1} + p_i$. En otro caso, $p_{i+1} = p_{i+1} + p_i$ y las 3-tuplas de las posiciones $i + 1, \ldots, j$ se desplazan una posición a la izquierda. En ambos casos, la longitud se actualiza $(j = j - 1)$. Esta operación puede ser aplicada sólo si inicialmente $j > 1$ (Figura 3.27).

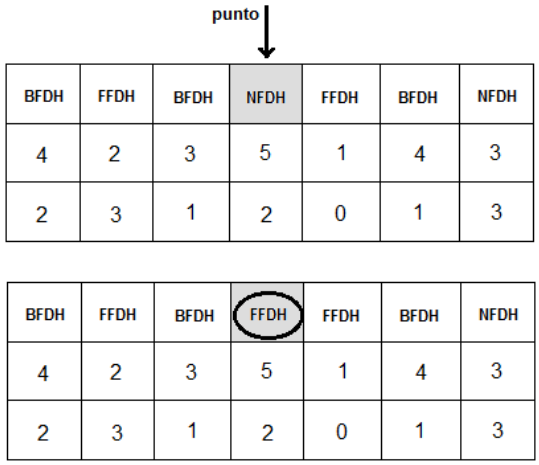

Figura 3.28: Operador de mutación *Reemplazar*

- *Reemplazar*: selecciona aleatoriamente una 3-tupla (h_i, p_i, o_i) de la representación. h_i se fija aleatoriamente a una de las heurísticas de bajo nivel definidas (Figura 3.28).

Cada uno de los movimientos se aplica bajo la probabilidad de mutación p_m, es decir, se generan tres probabilidades - una por movimiento - y se realizan aquellos cuya probabilidad sea inferior a p_m.

3.6.3. Resultados computacionales

Para evitar la implementación desde cero de los MOEAs más conocidos, se ha hecho uso de METCO, entorno explicado en el Capítulo 2. La evaluación experimental se ha llevado a cabo usando un cluster dedicado de 20 nodos de doble núcleo con un sistema operativo Debian GNU/Linux. Cada nodo consiste en dos Intel® Xeon 2.66 GHz, tiene 1GB de RAM y una red de interconexión Gigabit Ethernet. Las tres aproximaciones del problema (utilizando dos codificaciones directas y una codificación basada en hiperheurística) se han implementado en C++ y compilado con *gcc 4.1.3* y MPICH *1.2.7*. Para el estudio computacional, se han usado algunas instancias disponibles en la bibliografía [56, 90].

Tal y como se vio en el Capítulo 2, para introducir un nuevo problema en METCO sólo es necesario especificar: cómo se representa un individuo, cómo se genera la población inicial, cómo se evalúan los objetivos, y qué operadores evolutivos se van a utilizar. Por ello, en primer lugar se han definido tres individuos diferentes para la solución del 2DCSP. Por una parte, se han implementado dos

Esquema	Algoritmo	Cruce	Mutación	Población
Tamaño Controlado	NSGA-II	0.7	0.3	50
Tamaño Completo	NSGA-II	0.9	0.3	50
Hiperheurística	NSGA-II	0.6	0.4	50

Tabla 3.8: Parámetros de configuración

codificaciones directas basadas en la representación postfija de la disposición de los patrones: el esquema de codificación de *tamaño controlado* y el de *tamaño completo*. Por otra parte, se ha implementado una codificación basada en hiperheurística. Para todas las aproximaciones se han implementado las representaciones correspondientes, las evaluaciones, las generaciones iniciales y los operadores involucrados. Las propuestas se han evolucionado usando diferentes MOEAs. Finalmente el algoritmo *Non-Dominated Sorting Genetic Algorithm II*, NSGA-II [55] ha mostrado el mejor comportamiento, por lo que se ha utilizado con cada propuesta ajustando los parámetros que se muestran en la Tabla 3.8. Se trata de un algoritmo evolutivo multi-objetivo basado en *clasificación no dominada*, es decir, para evaluar la adaptación se clasifican las soluciones en frentes de soluciones no dominadas. El procedimiento que sigue es el siguiente: las dos poblaciones, la de padres y la de descendientes, se ordenan de acuerdo a su rango, y se seleccionan las mejores soluciones para crear la nueva población. En el caso de tener que seleccionar individuos del mismo rango, se utiliza una estimación de la densidad basada en la distancia de acumulación de los individuos que los rodean perteneciendo al mismo rango, para obtener las soluciones más prometedoras. Dos de las características más importantes que diferencian al NSGA-II del *Non-Dominated Sorting Genetic Algorithm*, NSGA y de otras propuestas basadas en clasificación no dominada son las siguientes. Primero, es una aproximación rápida con una complejidad computacional reducida ($O(mN^2)$). Segundo, utiliza un operador de selección que combina las poblaciones previas con las nuevas poblaciones generadas, asegurando el elitismo en la propuesta.

En definitiva, haciendo uso del algoritmo NSGA-II, para cada esquema de codificación, se han llevado a cabo treinta ejecuciones. La evaluación de los esquemas de codificación tiene distintos costes asociados, por lo que, si se quiere realizar una comparativa justa, debe hacerse por tiempo. Por esta razón, el criterio de parada para las ejecuciones se ha fijado a diez minutos.

La solución óptima mono-objetivo (maximización del beneficio total) se conoce para las instancias seleccionadas [52]. Partiendo de dichas soluciones, hemos evaluado el número de cortes requeridos para obtener las piezas. Estos resultados se han tomado como referencia para medir la calidad de nuestras soluciones multi-objetivo. En la Tabla 3.9 se puede ver para cada instancia, la solución óptima mono-objetivo

	Sol. Mono-objetivo		Sol. Multi-objetivo					
			Tamaño Controlado		Tamaño Completo		Hiperheurística	
Instancia	**Beneficio**	**Cortes**	**Beneficio**	**Cortes**	**Beneficio**	**Cortes**	**Beneficio**	**Cortes**
ATP33s	236611	34	230072.2	17.2	219720.0	15.1	215027.7	30.5
			29488.5	2.0	30051.0	2.0	60633.6	3.0
ATP37s	387276	39	376826.7	18.1	355970.3	15.1	357343.0	34.0
			47004.8	2.0	47888.0	2.0	133168.8	4.0
ATP39s	268750	39	256807.9	17.0	249770.2	14.9	247349.4	24.4
			33384.0	2.0	33384.0	2.0	130022.2	5.9
ATP36s	130744	45	126703.5	17.4	124954.0	12.2	124736.0	16.0
			19019.0	2.0	19019.0	2.0	76076.0	5.0
CW6	12923	26	11780.7	17.7	11059.1	16.5	4933.0	8.0
			989.2	2.0	933.0	2.0	993.0	2.0
CL_07_100_08	22443	30	21721.5	7.7	21632.2	6.1	18964.0	25.0
			9818.5	2.0	9999.0	2.0	9999.0	2.0
Hchl2	9954	21	9125.7	16.1	9080.6	15.2	8992.0	18.0
			912.0	2.0	925.0	2.0	1800.0	3.0
Hchl5s	45410	31	42603.1	15.1	41989.1	13.5	38280.0	17.0
			4968.0	2.0	4968.0	2.0	9936.0	3.0
Hchl5s_	45361	31	42871.8	16.3	42022.5	13.6	38490.0	18.0
			4968.0	2.0	4968.0	2.0	9936.0	3.0
CL_07_50_09	22088	14	21752.2	5.6	21949.1	6.0	13129.0	12.0
			9345.7	2.0	9506.0	2.0	9506.0	2.0
CL_07_25_08	21915	18	21134.9	10.0	21763.8	10.0	14879.0	14.0
			9933.0	2.0	9999.0	2.0	9999.0	2.0
CL_07_25_10	21915	18	21160.0	10.0	21303.9	11.0	16962.0	18.0
			9946.2	2.0	9999.0	2.0	9999.0	2.0

Tabla 3.9: Soluciones mono-objetivo y multi-objetivo

y los resultados obtenidos por las aproximaciones multi-objetivo.

La tabla muestra dos soluciones para cada par instancia-aproximación multi-objetivo: la media de la mejor solución cuando se considera el objetivo "beneficio", es decir, la media del mayor beneficio de los frentes de Pareto, y la media de la mejor solución cuando se considera el objetivo "número de cortes", es decir, el menor número de cortes en los frentes de Pareto. Si se mira atentamente los máximos beneficios obtenidos con las propuestas multi-objetivo y el valor del beneficio óptimo en la mono-objetivo, se puede comprobar que las aproximaciones multi-objetivo no son capaces de alcanzar los valores óptimos. Sin embargo, las propuestas multi-objetivo proporcionan valores de beneficio bastante cercanos a los óptimos, usando un número de cortes mucho menor que los obtenidos con la solución correspondiente al beneficio óptimo, es decir, las aproximaciones multi-objetivo proporcionan soluciones con un claro compromiso entre los dos objetivos.

Para comparar las propuestas multi-objetivo entre sí, se comenzará teniendo en cuenta en primer lugar ambas codificaciones directas, de *tamaño controlado* y *tamaño completo*. Se ha utilizado la métrica del hipervolumen [193] para comparar los resultados obtenidos por ambas codificaciones. Las Figuras 3.29 y 3.30 muestran el

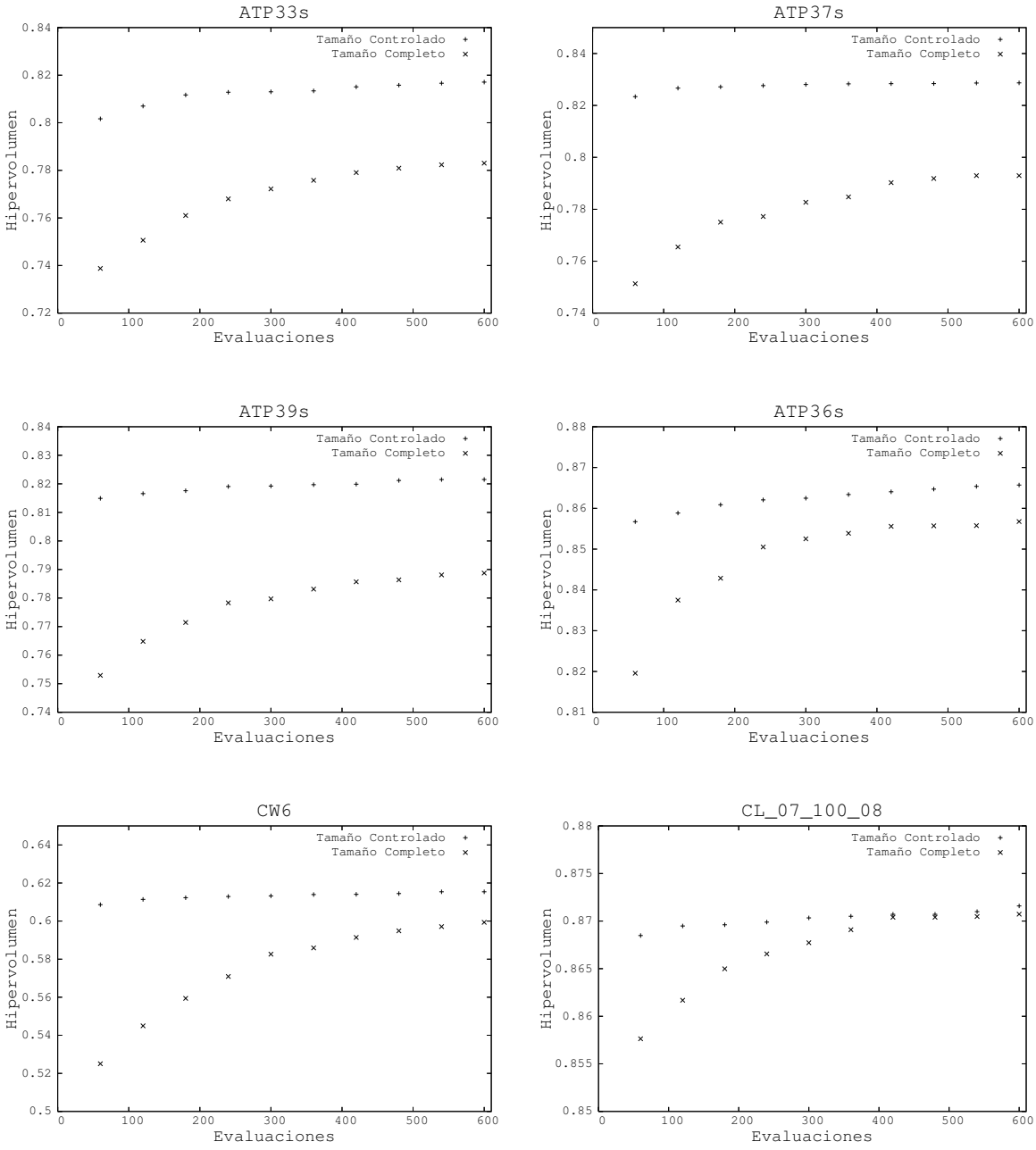

Figura 3.29: Instancias con mejor hipervolumen para *tamaño controlado*

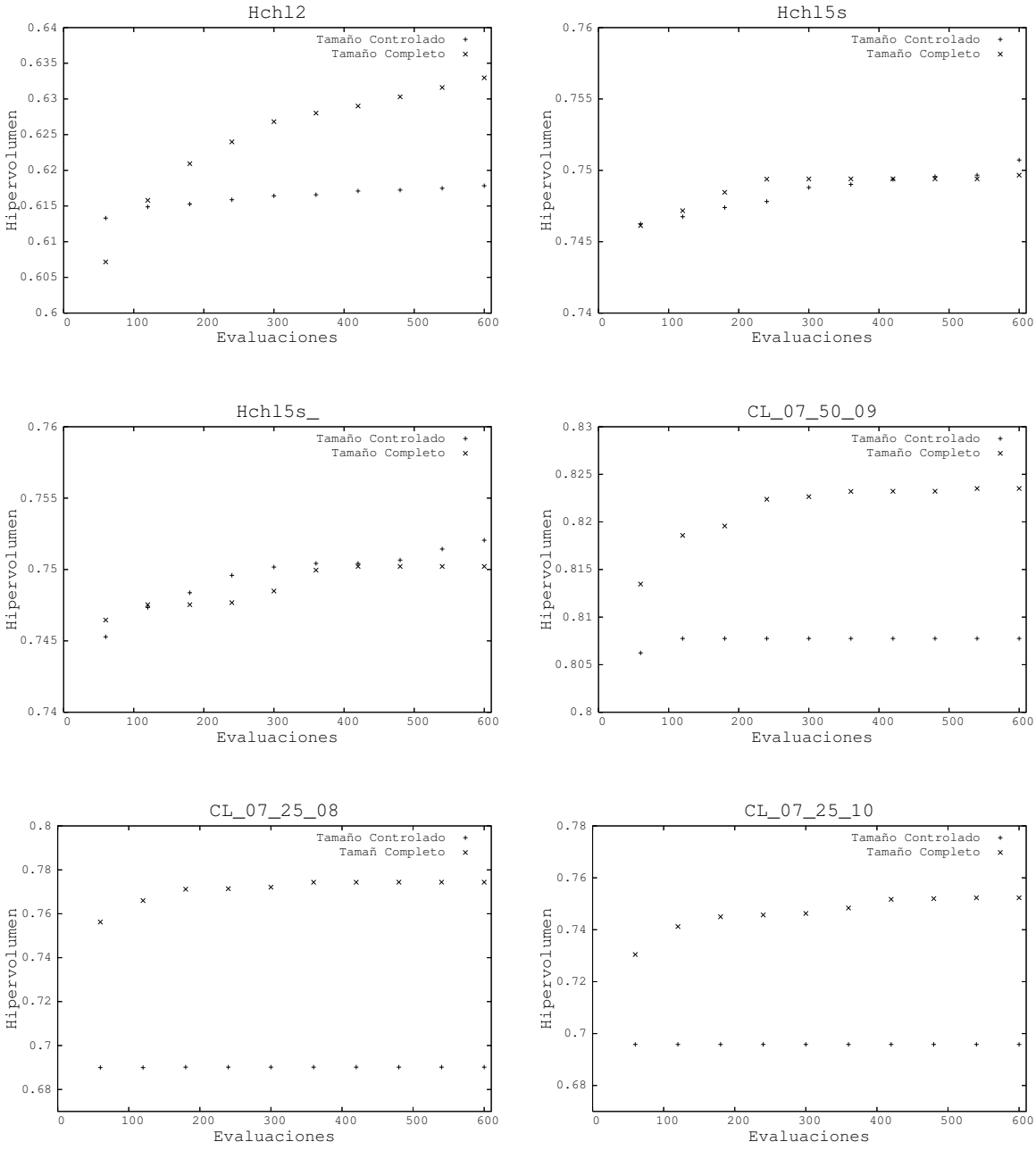

Figura 3.30: Instancias con mejor hipervolumen para *tamaño completo*

Número de piezas disponibles en las instancias mayores					
ATP33s	**ATP37s**	**ATP39s**	**ATP36s**	**CW6**	**CL_07_100_08**
224	222	163	153	149	100
Número de piezas disponibles en las instancias menores					
Hchl2	**Hchl5s**	**Hchl5s_**	**CL_07_50_09**	**CL_07_25_08**	**CL_07_25_10**
75	60	58	50	25	25

Tabla 3.10: Número total de piezas para cada instancia del problema

hipervolumen para todas las instancias usadas tanto para la codificación de *tamaño controlado* como para la de *tamaño completo*. En el conjunto de instancias de la Figura 3.29, el esquema de *tamaño controlado* proporciona un mejor comportamiento que el esquema de *tamaño completo*. Sin embargo, en el conjunto de instancias de la Figura 3.30, ocurre lo opuesto, aunque en algunos casos ambas aproximaciones tienen un comportamiento similar. Si se analizan las características de las instancias usadas, y en particular el número total de piezas demandadas (Tabla 3.10), nos damos cuenta de que, en general, la aproximación de *tamaño controlado* proporciona mejores resultados para las instancias más grandes, que son las de la Figura 3.29. Por el contrario, la aproximación de *tamaño completo* proporciona mejores resultados con el conjunto de instancias de menor número de piezas, que son las de la Figura 3.30. Por lo tanto, no se puede decir que ninguna de las dos aproximaciones sea mejor que la otra.

Después de analizar y comparar ambos esquemas de codificación directa, queda por profundizar en el comportamiento del esquema de codificación basado en hiperheurística. En la Tabla 3.11 se muestran los valores de los dos objetivos de optimización cuando las heurísticas de bajo nivel se aplican individualmente a las instancias y cuando se aplica la aproximación multi-objetivo con la codificación basada en hiperheurística. En el caso de la aproximación hiperheurística, se muestran dos resultados. El primero de ellos corresponde con la media de los mejores valores de las 30 ejecuciones, considerándose el mejor valor aquel que proporciona un mayor valor para el objetivo "beneficio". El segundo corresponde con la media de los mejores valores de las 30 ejecuciones, considerándose el mejor valor aquel que proporciona el menor valor para el objetivo "número de cortes". Como se puede observar, para cada instancia del problema las tres heurísticas mono-objetivo proporcionan la misma solución. Esto puede deberse a que los huecos superiores de cada capa o nivel de piezas que se van creando nunca puedan albergar las piezas que se intentan colocar más tarde debido a que son más pequeños o nulos, con lo que el comportamiento de las tres heurísticas de colocación sea el mismo para las

Solución	**ATP33s**		**ATP37s**		**CL_07_100_08**		**Hchl5s**	
Aproximación	beneficio	cortes	beneficio	cortes	beneficio	cortes	beneficio	cortes
NFDH	166780	15	271275	12	9506	2	27738	8
FFDH	166780	15	271275	12	9506	2	27738	8
BFDH	166780	15	271275	12	9506	2	27738	8
Codificación	215027.70	30.50	357343.00	34.00	18964.00	25.06	38280.00	17.00
Hiperheurística	60633.60	3.03	133168.80	4.00	9999.00	2.00	9936.00	3.00

Tabla 3.11: Valores de los objetivos usando las heurísticas mono-objetivo y el esquema multi-objetivo basado en hiperheurística

instancias usadas cuando se aplican al conjunto completo de piezas. Sin embargo, el primer resultado de la hiperheurística, mejor en cuanto al objetivo "beneficio", siempre supera el beneficio que obtienen las heurísticas por separado, aunque este beneficio tiene asociado un número de cortes más elevado. El segundo resultado, mejor en cuanto al objetivo "número de cortes", siempre consigue reducir el número de cortes con respecto a las heurísticas de colocación, pero casi siempre tiene asociado un beneficio menor que el de las heurísticas. Es decir, en general se pueden obtener mejores resultados combinando las heurísticas de colocación mediante la hiperheurística, que aplicándolas individualmente, al menos teniendo en cuenta los objetivos por separado. Pero hasta el momento sólo se han tenido en cuenta los valores extremos para cada objetivo (beneficio y cortes). Ahora es necesario comprobar el comportamiento del conjunto completo de soluciones obtenidas por el esquema de codificación basado en hiperheurística.

Dado que para comparar la calidad de las codificaciones directas se ha empleado el hipervolumen, no se ha visualizado el comportamiento de dichas aproximaciones sobre el espacio de soluciones. Por ello, en este caso, con la intención de identificar las áreas del espacio de búsqueda que están siendo exploradas, se utilizarán las superficies de alcance (explicadas con detalle en la Sección 2.8 del Capítulo 2). De este modo además se podrán comparar los comportamientos del esquema basado en hiperheurística y los esquemas de codificación directa. Para simplificar y visualizar mejor los resultados se usará sólo un esquema de codificación directa, por ejemplo, el esquema de *tamaño completo*.

La Figura 3.31 muestra, para cuatro instancias diferentes, las superficies de alcance 1, 15 y 30, obtenidas usando el esquema de codificación directa de *tamaño completo* y el esquema basado en hiperheurística. Simplemente comparando los valores de la Tabla 3.9 ya se intuye que las codificaciones directas son mejores que la hiperheurística. Ahora además, se puede observar que los resultados para la codificación directa dominan a los de la hiperheurística, al contrario de lo que ocurre con

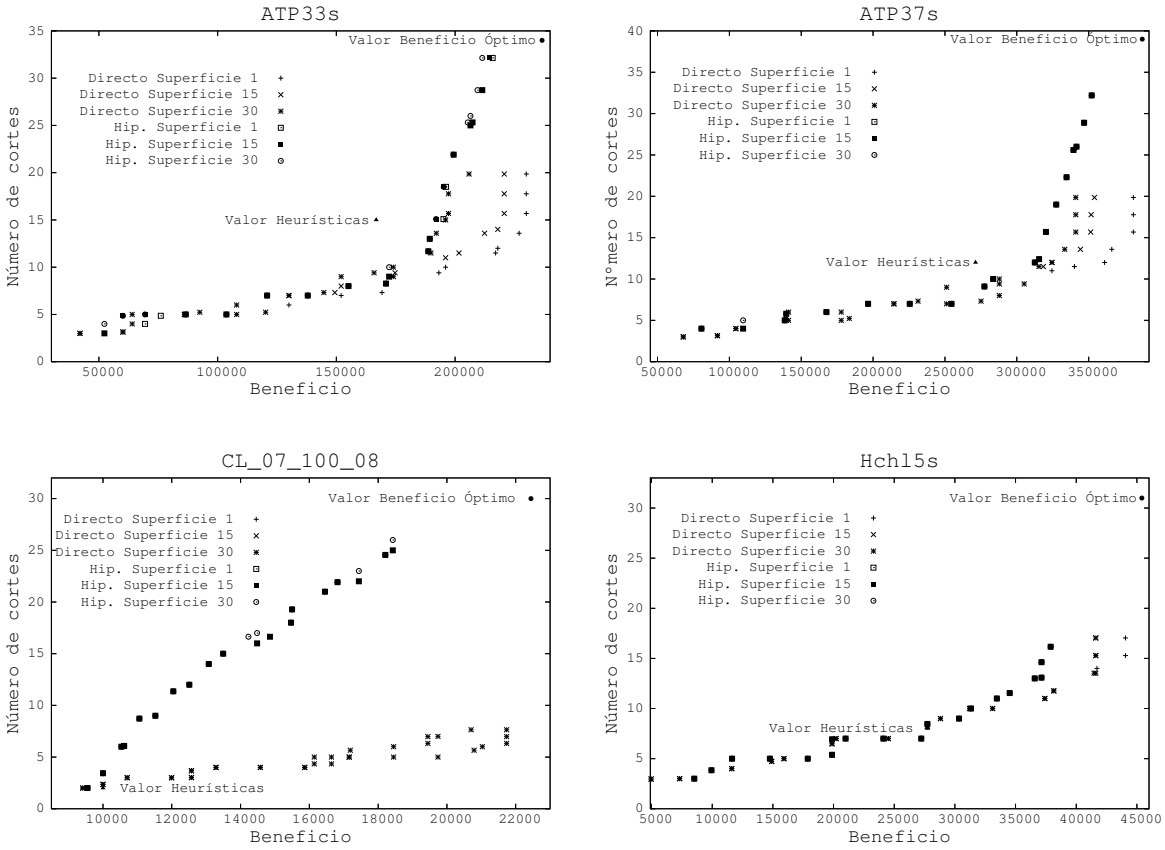

Figura 3.31: Superficies de alcance para el esquema directo de *tamaño completo* y el basado en hiperheurística

el *Two-Dimensional Strip Packing Problem*, como se verá en el próximo capítulo[2].

Es importante destacar que la hiperheurística está basada en heurísticas mono-objetivo que no proporcionan resultados de gran calidad porque originalmente han sido creadas para tratar con problemas de corte orientados a niveles y no tienen en cuenta que el beneficio asociado a cada pieza no tiene por qué ser proporcional a su área. Por esta razón, se ha querido comprobar si añadiendo al esquema hiper-heurístico una heurística diseñada específicamente para este problema se mejoran los resultados.

[2]Es necesario tener en cuenta que la complejidad o dificultad de ambos problemas no es la misma, ya que, aunque ambos manejan instancias con un número similar de elementos, el 2DCSP involucra un espacio de búsqueda menor que el *Two-Dimensional Strip Packing Problem*, 2DSPP al no permitir la rotación de las piezas

Como para la solución exacta del 2DCSP se ha necesitado desarrollar una cota inferior, aquí se ha reutilizado la heurística implementada, que no sólo tiene en cuenta el tamaño de las piezas, sino el beneficio asociado a cada una de ellas, ya que no necesariamente éste debe ser proporcional a su área. Esta heurística ha sido explicada con detalle en la Sección 3.6.2.1. Se trata de un algoritmo cuya complejidad aumenta en función del número de piezas de la instancia del problema. Por lo tanto, para instancias grandes es inviable su uso en la hiperheurística, ya que sólo para evaluar una vez toda una población de individuos que contengan esa heurística se necesita demasiado tiempo. Sin embargo, para instancias del problema con relativamente pocas piezas (menos de 100) la evaluación de la población se realiza en un tiempo razonable, por lo que se puede llevar a cabo una comparativa justa con el esquema hiperheurístico sin esta heurística. El esquema hiperheurístico proporciona un comportamiento más pobre para las instancias pequeñas por lo que además se podrá ver más claro la influencia de la nueva heurística. Así, utilizando los mismos parámetros y el mismo tiempo, se ha ejecutado el algoritmo multi-objetivo con la nueva heurística añadida al esquema hiperheurístico. En la Figura 3.32 se muestra el hipervolumen para todas las instancias pequeñas, con y sin la nueva heurística. Como se puede observar, el hipervolumen mejora considerablemente al introducirla, confirmando que, efectivamente, es necesaria una heurística que tenga en cuenta el beneficio asociado a las piezas independientemente de su área.

En cualquier caso, todas las aproximaciones multi-objetivo proporcionan mejores resultados que las heurísticas mono-objetivo con respecto al beneficio, y reducen considerablemente el número de cortes necesarios para obtener las piezas. Además, los resultados obtenidos con las aproximaciones multi-objetivo se encuentran muy cerca de los valores óptimos del beneficio. Como ventaja adicional, las aproximaciones multi-objetivo proporcionan un conjunto de soluciones entre las cuales la persona encargada de tomar las decisiones podrá seleccionar la que muestre el compromiso más adecuado para cada caso en concreto.

3.6.4. Conclusiones

Dado que en muchas aplicaciones reales el 2DCSP se presenta con múltiples objetivos se ha tratado de resolver intentando maximizar el beneficio y minimizar el número de cortes para obtener las piezas. Existen muchas aproximaciones en la bibliografía para resolver el problema mono-objetivo, sin embargo, se desconocen propuestas multi-objetivo que traten con la formulación del problema aquí utilizada. Como resulta inviable aplicar un método exacto para resolver este problema multi-objetivo, se han aplicado MOEAs (concretamente el NSGA-II) y tres esquemas de codificación: dos codificaciones directas basadas en una notación postfija (*tamaño*

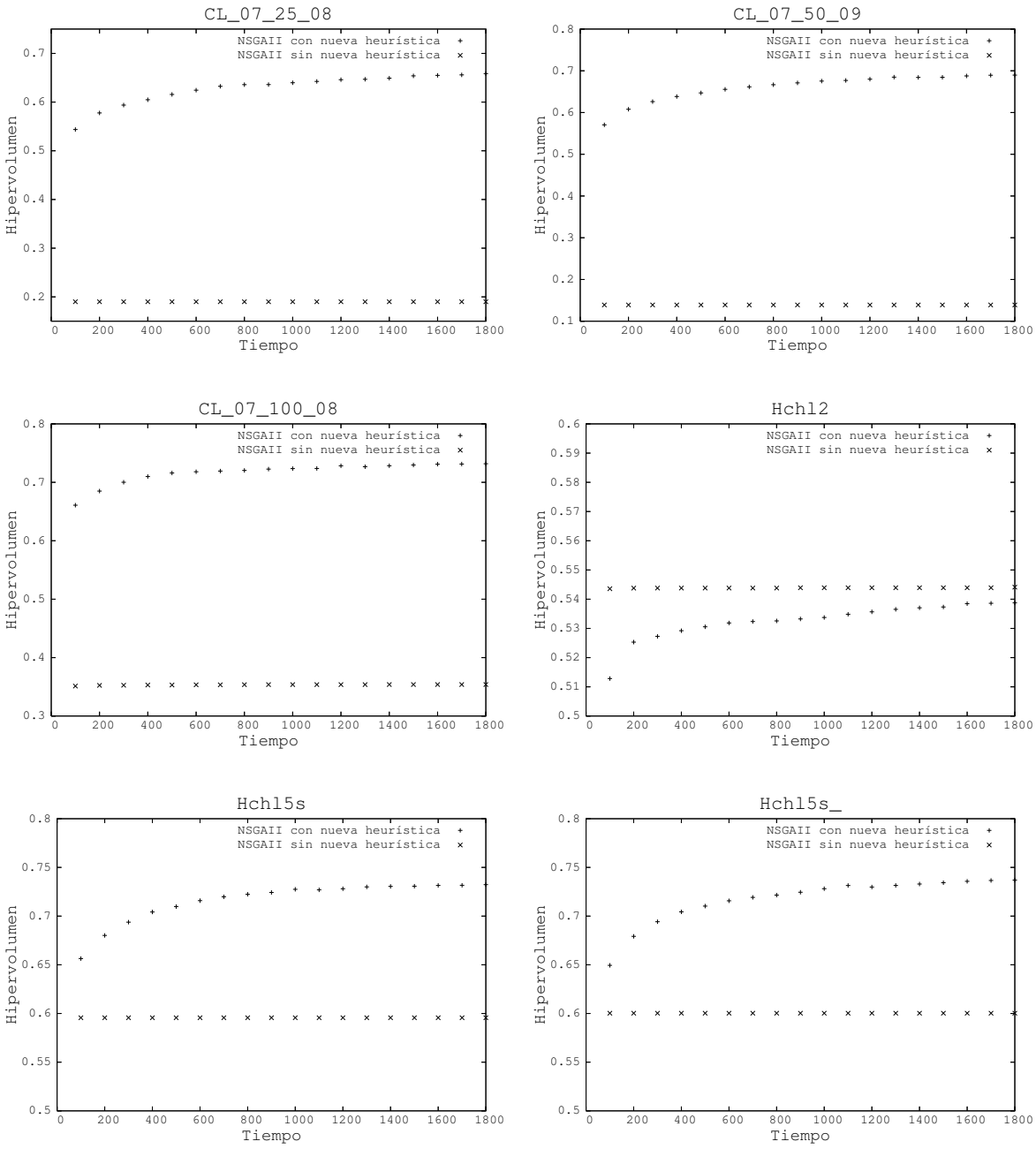

Figura 3.32: Hipervolumen para el esquema hiperheurístico con y sin la nueva heurística

controlado y *tamaño completo*) y una codificación basada en una hiperheurística. La diferencia entre los esquemas directos es la manera en la que se controla la obtención de soluciones válidas. En todos los casos se ha mostrado la efectividad de aplicar MOEAs a este tipo de problemas. Aunque las propuestas no alcanzan el valor de la solución de máximo beneficio obtenida por métodos exactos mono-objetivo, las soluciones se encuentran bastante cercanas y proporcionan un buen compromiso entre ambos objetivos. Por lo tanto, de acuerdo con los dos criterios de optimización se han diseñado tres propuestas que dan como resultado en una sola ejecución un conjunto de soluciones de compromiso entre los dos objetivos, de entre las cuales los clientes podrán elegir de acuerdo a sus necesidades. Incluso se han obtenido soluciones con un número muy bajo de cortes. En la mayoría de los casos el número de cortes se reduce al 40 % o 50 % de los obtenidos con la solución mono-objetivo. Además, hay que tener en cuenta que la implementación de un algoritmo exacto lleva asociada una gran dificultad y coste, y por si esto fuera poco, el algoritmo exacto se centra en un solo objetivo sin considerar los posibles efectos negativos en otras características de las soluciones. Sin embargo, el disponer de estas soluciones exactas mono-objetivo resulta útil para realizar comparativas y comprobar la calidad de las soluciones multi-objetivo.

Por otra parte, se han analizado las distintas codificaciones para ver las diferencias entre ellas. En primer lugar, si se comparan ambos esquemas de codificación directa se obtiene que uno de ellos se comporta mejor con las instancias mayores del problema y el otro con las instancias de menor número de piezas. En cambio los resultados demuestran que con el esquema de codificación basado en hiperheurística, aunque se obtienen soluciones competitivas si se compara con las heurísticas individuales, no se alcanza el comportamiento proporcionado con los esquemas de codificación directa. Esto puede deberse a que la hiperheurística está basada en heurísticas mono-objetivo que no proporcionan resultados de gran calidad para este problema, ya que originalmente fueron diseñadas para tratar con problemas de corte orientados a niveles. Así, se ha podido comprobar que al añadir al esquema una nueva heurística que tiene en cuenta los beneficios asociados a cada una de las piezas, el comportamiento de la hiperheurística mejora notablemente.

Capítulo

4

2D Strip Packing Problem

Este capítulo presenta la definición y formulación del *Two-Dimensional Strip Packing Problem*, 2DSPP, con el que se ha tratado, así como los trabajos relacionados existentes en la literatura que lo han abordado en sus diferentes variantes. Hay una multitud de aproximaciones que tratan de resolver el problema mono-objetivo utilizando métodos heurísticos, metaheurísticos e hiperheurísticos. Sin embargo, es mucho menor el número de algoritmos exactos que se han propuesto a lo largo del tiempo. Igualmente, el número de propuestas para la resolución multi-objetivo del problema es bastante escaso, a pesar de ser una visión más real del mismo. En el resto del capítulo se expone la bibliografía relacionada, así como los distintos métodos que se han utilizado en este trabajo para su resolución. Desde los exactos mono-objetivo y sus paralelizaciones, hasta los algoritmos evolutivos multi-objetivo y sus paralelizaciones, utilizados para resolverlo en su enfoque más real. En cuanto a la solución exacta, se ha tratado de reutilizar el algoritmo exacto empleado para resolver el 2DCSP, de manera que sea útil para resolver este nuevo problema. Permitir la rotación de las piezas es uno de los cambios introducidos en esa parte del código reutilizada de manera que incluso las reglas de detección de patrones duplicados y dominados se han tenido que adaptar. Además, se definen nuevas cotas para el nuevo algoritmo y se proponen distintas paralelizaciones para acelerar la ejecución del mismo. Por últmo, se proponen diversas codificaciones para abordar el problema multi-objetivo mediante algoritmos evolutivos y algunas paralelizaciones basadas en islas, tratando de mejorar los trabajos previos presentes en la literatura.

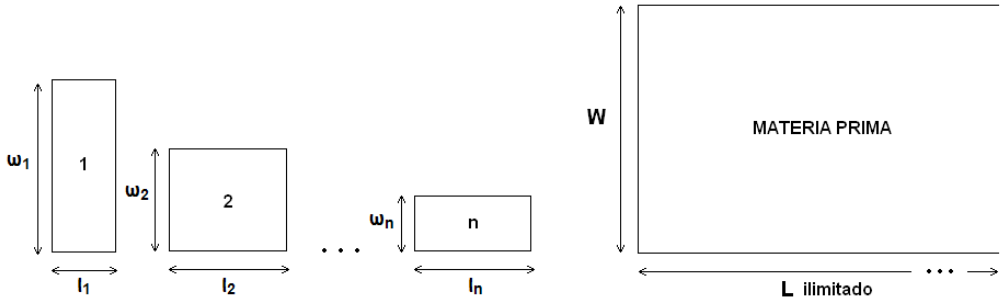

Figura 4.1: Material y piezas demandadas

4.1. Definición del problema

Los 2DSPPs se presentan en muchas industrias de producción donde la materia prima viene dada en forma de rollos. Algunos ejemplos son las industrias que tratan con papel, metal o textiles. Estos materiales aparecen muchas veces en forma de rollos que se pueden considerar de longitud infinita para propósitos prácticos. El objetivo principal de estos procesos de producción es encontrar una disposición del conjunto de piezas demandadas que minimice el largo de la materia prima usada. Ya que el uso óptimo del material disponible puede suponer importantes beneficios para las empresas, se han estudiado en profundidad y se han desarrollado muchas aproximaciones para este problema.

Dependiendo de la definición del conjunto de restricciones asociadas, se tendrá un tipo de problema y una formulación específica diferente. En particular, el problema que aquí se va a tratar supone el corte de un conjunto de n piezas demandas a partir de un rollo largo u hoja de materia prima (Figura 4.1). La hoja de material tiene un ancho fijo W y un largo ilimitado L. Dada esta condición de largo ilimitado, la hoja de material se suele conocer como "strip". Cada pieza rectangular demandada i tiene unas dimensiones fijas (l_i, w_i), aunque se pueden rotar 90 grados, permitiendo también las dimensiones (w_i, l_i). Todas las piezas demandadas deben colocarse ortogonalmente sobre el material de tal modo que no se superpongan y que se generen sólo construcciones horizontales y verticales de piezas. Esta última restricción hace que siempre sea posible obtener las piezas mediante cortes de guillotina, tal y como se vio en la Sección 3.3 del capítulo anterior.

Formalmente, sea $\mathcal{D} = \{1, 2, \ldots, n\}$ un conjunto de n rectángulos, cada uno con su largo l_i y su ancho w_i. El objetivo del problema es colocar ortogonalmente esos n rectángulos sin superposición en el material de ancho W de manera que se minimice el largo requerido L del mismo. La esquina inferior izquierda del material se considera el origen del plano xy, siendo el eje x la dirección del largo del material, y el eje y la

dirección del ancho. La localización de cada rectángulo i en el material se representa por las coordenadas (x_i, y_i) de su esquina inferior izquierda. De este modo, el 2DSPP se formula del siguiente modo:

$$\text{minimizar } L \tag{4.1}$$

$$\text{sujeto a } \; x_i + l_i \leq L \;\; \text{y} \;\; y_i + w_i \leq W \;\; \forall i \in \mathcal{D} \tag{4.2}$$

$$x_i \geq 0 \;\; \text{y} \;\; y_i \geq 0 \;\; \forall i \in \mathcal{D} \tag{4.3}$$

$$x_i + l_i \leq x_j \;\; \text{o} \;\; x_j + l_j \leq x_i \;\; \text{o}$$
$$y_i + w_i \leq y_j \;\; \text{o} \;\; y_j + w_j \leq y_i \;\; \forall i, j \in \mathcal{D}, \; i \neq j \tag{4.4}$$

Las restricciones 4.2 y 4.3 requieren que todos los rectángulos estén dentro del material con ancho W y largo L. La restricción 4.4 evita que los rectángulos se superpongan. Una distribución de los n rectángulos es factible sólo si satisface todas las restricciones mencionadas.

El problema formulado se conoce por varios nombres, aunque el que se empleará aquí (*Strip Packing Problem*) es el más relevante en la literatura. Sin embargo, siguiendo la clasificación dada en la Sección 1.3, el problema planteado pertenece a una subclase de problemas de corte y empaquetado denotados como *Problemas de Dimensión Abierta* [182]. Estos problemas a su vez pertenecen al tipo de minimización de la entrada, es decir, el suministro de objetos grandes es suficientemente grande para acomodar todas las piezas pequeñas, así que el objetivo es minimizar el valor de los objetos grandes necesarios para que quepan todas las piezas pequeñas. Cuando una de las dimensiones del objeto grande se considera variable, de modo que el problema supone una decisión para fijar la extensión de esta dimensión, la minimización de la entrada se denota como un *Problema de Dimensión Abierta*. Teniendo en cuenta las propiedades presentadas en la Sección 1.2, se han resumido las principales características del 2DSPP en la Tabla 4.1.

El objetivo más común en el 2DSPP consiste en minimizar el largo de la materia prima necesaria para cortar todo el conjunto de piezas demandadas. Sin embargo, en algunas aplicaciones reales de corte y empaquetado, se pueden tener en cuenta otros criterios de optimización. Por las mismas razones explicadas para el MOCSP en el capítulo anterior, donde se ha hecho referencia a que el número de cortes es un aspecto importante a la hora de determinar el coste y la eficiencia del proceso de producción, para el caso de 2DSPP el número de cortes se tomará como un segundo objetivo en la Sección 4.5. De este modo, el problema se podrá plantear como un problema de optimización multi-objetivo, optimizando la disposición de las piezas rectangulares intentando minimizar el largo del material al mismo tiempo que se maximiza el número de cortes necesarios para obtener las piezas demandadas. Esta

Nombre de la Propiedad	Valor de la Propiedad
Dimensionalidad	Dos dimensiones (2D)
Medición de cantidad	Discreta
Forma de las figuras	Regular y 90° orientación
Variedad de piezas pequeñas	Variedad heterogénea
Variedad de piezas grandes	Un objeto grande con una dimensión abierta
Disponibilidad de piezas	No se establece orden entre las piezas
Restricciones de los patrones	Ortogonal y guillotinable
Tipo de asignación	Minimización de la entrada
Objetivos	Minimizar el largo necesario del objeto grande

Tabla 4.1: Propiedades del 2DSPP

formulación del problema se denotará como *Multi-Objective Strip Packing Problem*, MOSPP.

Tal y como se ha mencionado en el capítulo anterior, en los problemas de optimización reales, los distintos objetivos implicados se encuentran normalmente en conflicto el uno con el otro, es decir, no existe una única solución para optimizar simultáneamente todos los objetivos. En lugar de un solo óptimo, existe más bien un conjunto de soluciones alternativas, entre las cuales una persona encargada de tomar las decisiones debe seleccionar una solución de compromiso.

En el MOSPP propuesto, se trata con patrones de corte ortogonales y guillotinables. En tal caso, se podría pensar a priori que la obtención de patrones de corte compactos (que permiten minimizar el largo del material) de alguna manera supone un número reducido de cortes. Se espera que las construcciones compactas contengan poco desperdicio interno, y por lo tanto el número de cortes requerido se supone que será bajo. Como se muestra en la Figura 4.2, la combinación horizontal de piezas **1 2 H 3 4 H H** implica ocho cortes de guillotina para obtener las piezas individuales. Sin embargo, la segunda construcción **1 2 V 3 4 V H** claramente proporciona un patrón de corte más compacto: requiere la mitad del largo que el patrón previo y, al mismo tiempo, el número de cortes de guillotina implicados también es la mitad. Este segundo patrón de corte es mucho mejor que el primero puesto que es capaz de mejorar ambos objetivos simultáneamente.

Esto no ocurre con todas las instancias, ya que en muchos casos ambos objetivos se encuentran en conflicto. Por ejemplo, la Figura 4.3 muestra dos patrones de corte diferentes, cada uno optimizando uno de los dos objetivos. El primer patrón **1 2 H 3 H** implica un mayor largo pero sólo requiere de cuatro cortes de guillotina. En contraposición, el segundo patrón **1 2 V 3 V** reduce el largo necesario a expensas de incrementar el número de cortes a seis. Para este conjunto de piezas demandadas, no

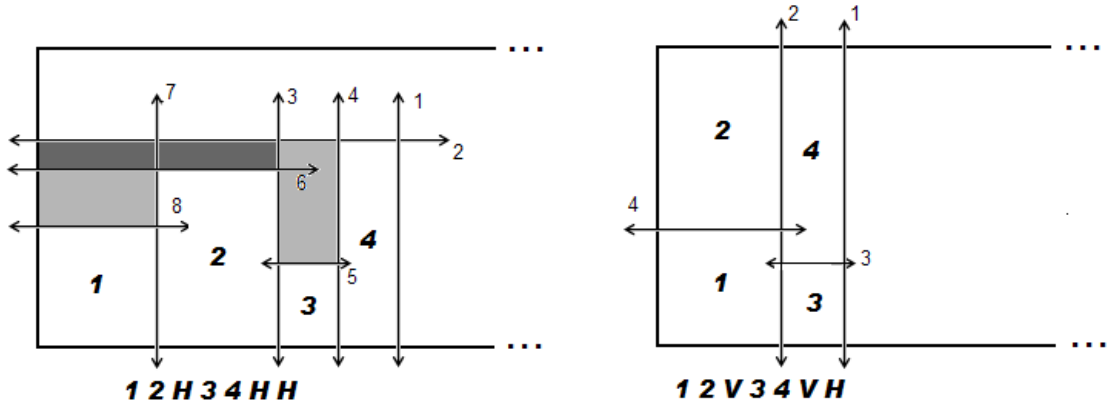

Figura 4.2: Ejemplo de objetivos no conflictivos

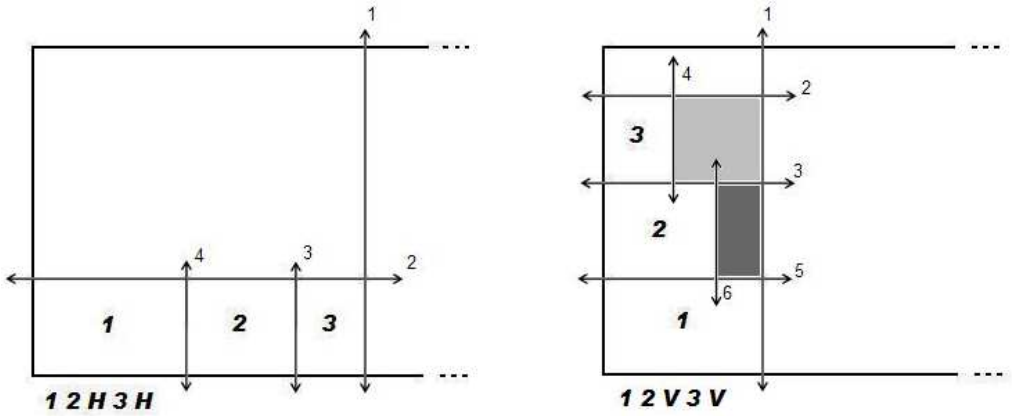

Figura 4.3: Ejemplo de objetivos conflictivos

hay un único patrón de corte factible que optimice ambos objetivos simultáneamente. Por lo que en general, se puede decir que ambos objetivos están al menos en cierto grado de conflicto.

Tal y como se puede observar, para este problema se utiliza la misma notación postfija explicada en el capítulo anterior para el 2DCSP. Sin embargo, en este caso se permite la rotación de las piezas, por lo que se incluirá al final de la notación una cadena indicando si cada pieza se ha rotado o no. Esa cadena estará compuesta por ceros y unos, correspondiendo el primer dígito a la primera pieza, el segundo a la segunda, y así sucesivamente. Un cero indicará que no se ha rotado; por el contrario, un uno significará que esa pieza se ha rotado.

4.2. Trabajos relacionados

Igual que en el caso del 2DCSP, el diseño de una aproximación para resolver el 2DSPP depende generalmente del entorno particular de su aplicación, y de los medios computacionales disponibles. Como ocurre en la mayoría de los problemas de corte y empaquetado, el problema de encontrar la solución óptima para el 2DSPP es NP-duro. Esto supone que, aunque los algoritmos exactos aseguren la obtención de las soluciones óptimas, no pueden tratar con instancias del problema grandes y reales. Por ello el número de algoritmos exactos en la literatura no es muy extenso. No obstante, dado que el material puede ser muy caro en algunas industrias de producción (por ejemplo, algunos tipos de tela como la piel), el estudio y el diseño de propuestas exactas que determinen el largo óptimo del material puede ser un campo de investigación interesante. Actualmente no se conocen trabajos que traten con la solución exacta para la definición particular del problema que aquí se ha utilizado, es decir, con la restricción de cortes de guillotina y permitiendo al mismo tiempo la rotación de las piezas. Sin embargo, estas restricciones son muy comunes para la mayoría de los materiales existentes y la maquinaria industrial.

Para resolver este problema sin la restricción de cortes de guillotina, Martello et al. [134] propuso un método de ramificación y acotación que hace uso de las cotas superior e inferior propuestas en el mismo trabajo. Alvarez-Valdes et al. [5] describe un algoritmo de ramificación y acotación donde la estructura del árbol de búsqueda es igual a la utilizada por Martello et al. [134], pero el rendimiento se mejora usando una nueva cota inferior y nuevas comprobaciones de dominancias. Kenmochi et al. [109] también propone un método de ramificación y acotación para la versión no-guillotina del problema. Recientemente, Boschetti y Montaletty [28] han propuesto procedimientos de reducción, una cota inferior y una cota superior, y un algoritmo exacto para resolver esta misma versión no-guillotina del problema. El 2DSPP con cortes de guillotina fue introducido por Hifi [93], proponiendo dos algoritmos basados en ramificación y acotación. También, Cui et al. [184] utiliza un algoritmo de ramificación y acotación recursivo para obtener una solución aproximada. Bekrar et al. [15] propone una nueva cota inferior y un nuevo algoritmo de ramificación y acotación. Finalmente, Cintra et al. [40] usa un método de generación de columnas y programación dinámica para resolver otra variante del problema. Sin embargo, solo unos pocos trabajos que tratan con métodos exactos para resolver el 2DSPP consideran la rotación de las piezas [109]. Además, el número de algoritmos exactos paralelos es extremadamente reducido. Desde nuestro conocimiento, solo Bak et al. [13] presenta un algoritmo de ramificación y acotación para resolver la versión guillotina del 2DSPP, y no tiene en cuenta la rotación de las piezas.

Tal y como ya se ha comentado, todos estos algoritmos exactos aseguran la obten-

ción de las soluciones óptimas, pero no pueden tratar con instancias grandes y reales del problema. Por ello, en general, las investigaciones se han centrado principalmente en desarrollar heurísticas que proporcionen soluciones de buena calidad (aunque no necesariamente óptimas) en un tiempo computacionalmente aceptable, para problemas que de otro modo serían intratables. En la literatura se puede encontrar una enorme cantidad de heurísticas para el 2DSPP de tipo no-guillotina [6, 87, 98, 120], pero el número de propuestas para el caso guillotinable no es demasiado extenso. Existe una gran variedad de estrategias heurísticas que definen la localización exacta de las piezas u objetos, es decir, dónde se debe colocar una pieza particular dentro de la materia prima. Algunas de ellas funcionan creando y rellenando diferentes niveles, pero otras no dividen el área en sub-espacios, sólo colocan las piezas siguiendo algunas reglas. La existencia de diferentes ordenaciones de la lista de rectángulos, cómo posicionarlos dentro de un nivel, etc., produce un gran número de posibles algoritmos heurísticos para el 2DSPP. Muchas de estas alternativas han sido estudiadas, aunque dado el número tan extenso de artículos publicados es difícil determinar el nivel de las investigaciones actuales en este área [100, 143]. Algunos de los algoritmos orientados a niveles más conocidos capaces de tratar con cortes de guillotina son [146]: *First-Fit Decreasing Height* [45], *Next-Fit Decreasing Height* [45], *Best-Fit Decreasing Height* [143], y *SPLIT* [143]. Muchos artículos proponen mejoras para estas heurísticas de colocación de bajo nivel, o las combinan con otras aproximaciones, como los algoritmos genéticos [21].

Por otra parte, para superar la principal desventaja de los algoritmos de búsqueda local, cuya debilidad es su incapacidad para escapar de mínimos locales, se han diseñado estrategias de búsqueda heurística más sofisticadas. Esto implica la aceptación temporal de estados de baja calidad. Así, los algoritmos metaheurísticos se pueden considerar hasta cierto punto como estrategias de búsqueda local, que incluyen un medio para escapar de los mínimos locales. En este sentido, como métodos más generales y sofisticados se han considerado diferentes tipos de algoritmos híbridos y metaheurísticas [21, 78, 99, 148]. Además, existen algunos trabajos que aplican los principios de las hiperheurísticas con el fin de determinar la forma de combinar las heurísticas individuales de bajo nivel [9, 103].

En general, se han diseñado y evaluado una gran variedad de heurísticas, metaheurísticas, esquemas de codificación, etc., para la formulación mono-objetivo del 2DSPP. Sin embargo, pese al gran número de trabajos que intentan resolver el problema, sólo unos pocos tratan con la visión real multi-objetivo [103, 166, 171]. En [166] se analizan varios criterios secundarios de optimización. La formulación multi-objetivo del 2DSPP que implica la minimización del largo del material y el número de cortes se trata en [51, 103, 171]. Estas propuestas están basadas en la aplicación de MOEAs. Los patrones de corte proporcionados por las aproximaciones

presentadas en [51, 171] siempre permiten cortes de guillotina, mientras que los de [103] son no-guillotinables. Los trabajos que tratan con cortes de guillotina [51, 171], están basados en una codificación de soluciones donde la distribución de piezas se representa directamente a través de una notación post-fija. Además, la propuesta presentada en [51] claramente mejora la propuesta previa presentada en [171].

4.3. Solución exacta

Este problema, al igual que el 2DCSP, se puede presentar en algunos campos de la industria donde la materia prima es muy cara, de manera que la obtención del patrón óptimo es muy importante. Por esta razón, a veces desarrollar un algoritmo exacto para el 2DSPP es crucial. En este caso se ha implementado un algoritmo para obtener la solución exacta de la formulación de este problema aquí tratada, es decir, teniendo en cuenta cortes de guillotina y permitiendo la rotación de las piezas. Las principales ideas de la propuesta secuencial están basadas en una búsqueda primero-el-mejor similar al algoritmo VB [175] y su modificación MVB [92]. Este algoritmo fue diseñado originalmente para resolver otro problema de corte, el 2DCSP expuesto anteriormente en el Capítulo 3, donde la materia prima tiene un ancho y un largo fijos y se necesita maximizar el beneficio total. Sin embargo, dicho algoritmo se ha adaptado para resolver el 2DSPP.

La implementación se ha basado en la propuesta de Hifi [93], que define una aproximación exacta llamada *Best-first-search Modified Viswanathan and Bagchi's Algorithm*, BMVB. Para esta propuesta se parte del rectángulo de mayor largo (L_{max}, W) y se intenta construir la distribución de piezas de largo menor o igual a L_{max}. En cada paso el algoritmo trata de construir un rectángulo más pequeño que contenga todas las piezas. Si dicho rectángulo existe, entonces se decrementa L_{max} y otra iteración del algoritmo intenta construir un rectángulo menor conteniendo todas las piezas. La propuesta se puede resumir del siguiente modo:

1. Sea (L_{max}, W) el rectángulo mayor obtenido aplicando una heurística. Esta solución se considera como una cota superior para el 2DSPP y todos los sub-rectángulos $(x, y) \leq (L_{max}, W)$, $x \geq 0$, $y \geq 0$.

2. Se aplica la función de Gilmore y Gomory [85] a (L_{max}, W). Este paso se utiliza para estimar la cota inferior del 2DSPP.

3. Se aplica el algoritmo MVB con algunas modificaciones para resolver de manera óptima el 2DSPP.

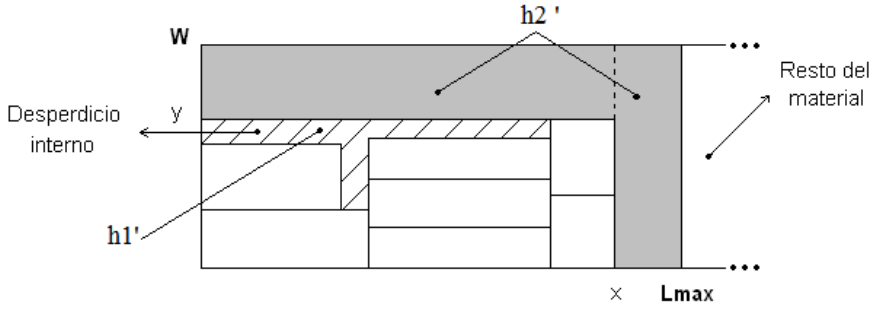

Figura 4.4: Distribución de piezas sobre el material

Por supuesto, el proceso de búsqueda del MVB trata un problema de maximización y por lo tanto, el propósito es transformar este aspecto para tratar un problema de minimización. La función $f = g + h'$ se usa considerando que g define el valor interno, y h' se define como la suma de h'_1 y h'_2, donde h'_1 representa el desperdicio interno de la construcción y h'_2 representa la estimación del resto del área no usada por (L_{max}, W) (Figura 4.4). Por lo tanto, el objetivo pasa a ser la construcción de la mejor cota inferior f para la que h'_2 es 0 y la distribución de las piezas es factible, es decir, el problema consiste en encontrar el mínimo valor de f.

Como novedad sobre el algoritmo BMVB, se ha usado la heurística BFDH* [21] para obtener la cota superior del largo del material, es decir, el primer rectángulo (L_{max}, W). Esta heurística considera la rotación de las piezas, ya que la definición particular del problema que se va a tratar considera dicha rotación, e intenta mejorar la heurística BFDH explicada en la Sección 3.6.2.1, de modo que cuando el algoritmo va a colocar el objeto actual en un espacio prueba ambas orientaciones. Como cota inferior para el largo del material se ha considerado el largo necesario si todas las piezas encajaran en el material sin dejar huecos, es decir, se ha calculado la suma de las áreas de todas las piezas y se ha hallado el largo correspondiente conociendo el ancho del material. Además, para mejorar el comportamiento del algoritmo MVB, se han adaptado y utilizado las reglas de dominancias y duplicados anteriormente utilizadas en el algoritmo exacto para el 2DCSP (Sección 3.4.1). Dicho algoritmo también ha sido adaptado para permitir la rotación de las piezas. De este modo, basta con recorrer mediante un bucle los posibles largos entre la cota inferior y superior, y realizar llamadas al algoritmo adaptado (Algoritmo 5).

Algoritmo 5 Solución exacta del 2DSPP

```
1: cotaSuperior = BFDH*;
2: cotaInferior = ∑_{i=1}^{n} l_i * w_i / W;
3: for (L = cotaInferior) hasta (L = cotaSuperior) do
4:     sol = MVB_Adaptado(L, W);
5:     if (sol ⊂ todas las piezas) then
6:         L_opt = L;
7:         return sol;
8:     end if
9: end for
```

4.3.1. Resultados computacionales

Los experimentos secuenciales se han realizado en una máquina con cuatro procesadores Intel Xeon a 1.4Hz y se ha usado el compilador *gcc 4.3.2*. Se han seleccionado 25 instancias de diferentes tamaños ampliamente usadas en la literatura [91] y se han realizado 30 ejecuciones secuenciales para cada instancia. En la Tabla 4.2 se muestran los tiempos medios (en segundos) invertidos en las ejecuciones para cada una de las instancias del problema. Como se puede observar, los resultados para la mayoría de las instancias se obtienen rápidamente, a excepción de dos casos, en los que se necesitan varias horas. En general, con menos de 15 piezas los resultados son inmediatos. Sin embargo, un breve aumento en el número de piezas puede suponer un gran crecimiento en el número de combinaciones a realizar entre piezas o disposiciones de piezas, considerando que, además, se han de comprobar ambas orientaciones posibles. Es necesario tener en cuenta que, como ya se ha mencionado, en el algoritmo exacto se han introducido todas las dominancias presentadas en el capítulo anterior, con lo cual los tiempos secuenciales ya se encuentran optimizados y, por lo tanto, reducidos.

4.3.2. Conclusiones

Se ha desarrollado un algoritmo secuencial para la resolución del 2DSPP teniendo en cuenta cortes de guillotina y permitiendo la rotación de las piezas. Las soluciones aportadas por dicho algoritmo exacto resultan útiles en aquellas industrias donde la materia prima es muy cara y se necesita desperdiciar la menor cantidad posible de la misma. Con lo cual, en ese caso el principal objetivo es minimizar el largo del material a usar, y un algoritmo exacto que proporcione el resultado con el mínimo largo es el apropiado. Sin embargo, en otras industrias donde el material no es tan caro, existen otros criterios a tener en cuenta a la hora de cortar las piezas. En estos casos, como se verá más adelante, es más apropiado utilizar un algoritmo multi-

Instancia	N.Piezas	Tiempo
SCP1	10	0.016
SCP2	11	0.054
SCP3	15	26.774
SCP4	11	0.175
SCP5	8	0.003
SCP6	7	0.065
SCP7	8	0.019
SCP8	12	8.525
SCP9	12	0.024
SCP10	8	0.068
SCP11	10	0.520
SCP12	18	5.385
SCP13	7	0.063
SCP14	10	0.341
SCP15	14	0.046
SCP16	14	62.752
SCP17	9	0.303
SCP18	10	4.080
SCP19	12	60.755
SCP20	10	0.509
SCP21	11	1.175
SCP22	22	75779.898
SCP23	12	11.777
SCP24	10	3.339
SCP25	15	7045.585

Tabla 4.2: Tiempos medios de las ejecuciones secuenciales

objetivo, pero la solución exacta mono-objetivo sigue siendo importante, ya que puede servir de referencia para conocer la calidad de las soluciones que aporte el algoritmo multi-objetivo.

En cuanto a los resultados, tal y como se ha comprobado, el tiempo que invierte el algoritmo exacto en la obtención de la solución óptima es satisfactorio para la mayoría de las instancias usadas. Sin embargo, cuando el número de piezas aumenta, el tiempo puede incrementarse demasiado, por lo que para instancias con un número de piezas elevado es aconsejable llevar a cabo alguna paralelización del algoritmo.

4.4. Paralelización de la solución exacta

Tal y como se ha visto en la sección anterior, el coste asociado a la implementación exacta para el 2DSPP es demasiado alto e implica demasiado tiempo para las instancias de mayor tamaño. Debido a ello, es recomendable una implementación paralela del algoritmo.

Haciendo un análisis del algoritmo secuencial utilizado, se ha podido comprobar que la mayor parte del tiempo necesario para la resolución de cada instancia se invierte en las llamadas al algoritmo MVB adaptado a este problema (teniendo en cuenta las orientaciones y algunas modificaciones en las dominancias). Por lo tanto, el objetivo aquí es mejorar el esquema secuencial y reducir el tiempo computacional llevando a cabo dos tipos de paralelizaciones de dicho algoritmo. Como es muy similar al utilizado para resolver el 2DCSP exacto, se han utilizado las mismas paralelizaciones diseñadas para resolver ese problema. La primera implementación paralela se puede considerar de grano fino, ya que se distribuye entre los procesos la combinación de una construcción con las de la lista CLIST. La segunda aproximación es de grano grueso, de modo que cada proceso realiza, de forma independiente, la combinación de las construcciones más prometedoras con las que ya han sido exploradas.

Es necesario recordar que, tal y como se dijo en el capítulo dedicado al 2DCSP, los *nodos generados* son los nodos que representan cualquier construcción creada durante el proceso de búsqueda. Los nodos que se borran de la lista OPEN para combinarse con los mejores subproblemas previos ya explorados son los *nodos computados*. Finalmente, los *nodos insertados* representan los nodos generados que se insertan en OPEN. Igual que en el caso del 2DCSP, la distribución de la carga entre los procesos es un aspecto importante. Una distribución justa de trabajo entre procesos es difícil de obtener, ya que no sólo se necesita una distribución justa de subproblemas a generar, sino también de los subproblemas a insertar en la lista de subproblemas usada por el algoritmo, y de los subproblemas que finalmente se van a computar. Esto es complicado, ya que antes de realizar una combinación no se puede saber si será válida o no para ser insertada.

En definitiva, del mismo modo que para el 2DCSP, se han utilizado dos implementaciones del algoritmo de grano fino, una con memoria compartida y otra con memoria distribuida, y dos implementaciones del algoritmo de grano grueso, una con memoria compartida y otra con memoria distribuida. Así se podrán realizar comparativas entre ambos tipos de algoritmos y además entre las distintas implementaciones de memoria compartida y distribuida.

4.4.1. Resultados computacionales

La evaluación experimental se ha llevado a cabo en un supercomputador Cray XE6, que ofrece un total de 1856 XE6 nodos de computación. Cada nodo contiene dos procesadores de 12 núcleos AMD a 2.1 GHz. Hay disponibles 32 GB de memoria principal por nodo, que es compartida entre sus 24 núcleos. Los procesadores están conectados con una interconexión de alto ancho de banda usando chips de comunicación Cray Gemini. Los chips Gemini están dispuestos en una red torus 3D. Se ha hecho uso del compilador *gcc 4.5.1*, y el módulo *xt-mpt 5.1.2*. Para el estudio computacional se han utilizado algunas instancias ampliamente usadas en la literatura [93, 100, 134] y disponibles en [91, 100, 133]. En primer lugar, se han seleccionado las instancias empleadas con la implementación secuencial [91] para las que se necesitaban varias horas de ejecución para obtener la solución exacta (SCP22 y SCP25). Luego, se han escogido las instancias de Hopper y Turton [100] y de Martello et al. [133] con un número similar de piezas que las instancias anteriores. Para estas instancias elegidas la versión secuencial del algoritmo no es capaz de obtener resultados en menos de 12 horas.

Tal y como ya se ha mencionado, se ha analizado la versión secuencial del algoritmo y para la paralelización se ha seleccionado la sección del código donde más esfuerzo computacional se invierte. Esta sección corresponde con la comprobación de si todas las piezas encajan dentro de la hoja de material para un largo específico, es decir, la llamada al MVB adaptado al 2DSPP. Así pues, para esta sección del código se han utilizado dos implementaciones de un algoritmo paralelo de grano fino, una con memoria compartida y otra con memoria distribuida, y otras dos implementaciones de un algoritmo paralelo de grano grueso, una con memoria compartida y otra con memoria distribuida. Se ha usado OpenMP para las implementaciones con memoria compartida y MPI para las implementaciones con memoria distribuida. Para las ejecuciones de grano-grueso el parámetro de configuración $tPeriod$ se ha fijado a 0.05 segundos, el $iPeriod$ a 150. $MaxBalThreshold$ a 100, $MinBalThreshold$ a 10, y $MaxBalLength$ a 45. Se han utilizado los mismos parámetros de configuración para todas las instancias del problema, es decir, el algoritmo no se ha configurado individualmente para cada instancia.

En el análisis del comportamiento secuencial del algoritmo exacto para el 2DSPP, una vez se han conocido los largos óptimos para las instancias iniciales, se ha podido comprobar que cuando el largo utilizado para llamar al algoritmo MVB adaptado no es suficiente para albergar todas las piezas de la instancia, el tiempo invertido en la llamada es irrelevante en comparación con el invertido cuando se llama con el largo óptimo. Por esta razón, con las instancias escogidas para analizar el comportamiento de los algoritmos paralelos se mostrarán los tiempos implicados en la llamada al

N	SCP22	SCP25	C2P1	C2P2	C2P3	HT04	HT05	HT06	BENG01	NGCUT03
2	6915.7	-	-	-	-	-	-	-	9554.0	-
4	3147.1	-	-	-	-	-	-	-	2616.7	-
8	1571.5	-	3647.7	-	-	2648.5	-	-	671.5	31850.0
16	806.7	31818.5	29817.3	-	-	-	-	-	2504.6	-
24	405.5	34249.5	-	-	-	-	-	-	6147.7	-

Tabla 4.3: Grano fino OpenMP

N	SCP22	SCP25	C2P1	C2P2	C2P3	HT04	HT05	HT06	BENG01	NGCUT03
2	794.4	2580.1	10737.7	1075.8	1687.6	9348.3	879.4	2537.5	1632.3	-
4	254.6	4760.6	-	-	-	-	-	41623.8	13213.6	-
8	178.0	-	-	-	-	-	-	-	814.3	-
16	137.4	26677.7	-	-	-	-	-	-	3348.6	-
24	2302.7	22374.6	-	-	-	-	-	-	7626.9	-
32	2287.4	18009.5	-	-	-	-	-	-	6955.4	-

Tabla 4.4: Grano fino MPI

N	SCP22	SCP25	C2P1	C2P2	C2P3	HT04	HT05	HT06	BENG01	NGCUT03
2	610.3	18999.1	-	-	-	-	-	-	-	-
4	46.5	-	40.6	34.6	2.0	62.8	45.8	6.8	675.8	493.2
8	1.2	257.9	178.5	134.0	586.1	154.3	2087.7	263.1	138.0	427.0
16	8.9	-	334.5	370.5	868.7	563.4	5502.4	1376.4	85.5	18.9
24	3.0	9.3	7.7	60.6	7.0	237.1	2482.9	1963.7	11.2	621.2

Tabla 4.5: Grano grueso OpenMP

N	SCP22	SCP25	C2P1	C2P2	C2P3	HT04	HT05	HT06	BENG01	NGCUT03
2	3500.8	25719.2	-	-	-	-	-	-	23.3	-
4	0.8	-	206.0	232.1	5.9	147.1	116.0	4.9	616.4	30.7
8	1.5	12157.2	3515.2	460.4	81.7	425.5	1326.1	333.1	15.3	17.7
16	10.6	10549.1	2518.3	126.1	31.8	5.2	52.2	52.5	3.1	81.6
24	1.5	1466.6	20.5	77.5	134.1	46.2	85.2	3.4	3.7	320.3
32	9.9	148.0	3.5	62.9	56.8	11.7	22.1	73.5	5.9	1.7

Tabla 4.6: Grano grueso MPI

algoritmo MVB adaptado con el largo óptimo para cada instancia. Así, se han llevado a cabo ejecuciones con 2, 4, 8, 16 y 24 procesadores para los algoritmos de memoria compartida (Tabla 4.3 y Tabla 4.5), y con 2, 4, 8, 16, 24 y 32 procesadores usando memoria distribuida (Tabla 4.4 y Tabla 4.6). Los resultados se muestran en segundos, y el guión indica que se han necesitado más de 12 horas de ejecución.

Si se comparan los resultados obtenidos usando los algoritmos de grano grueso con los resultados usando los de grano fino, se puede concluir que los esquemas de grano grueso son capaces de escalar mejor. De hecho, la mayoría de las veces no

se consigue una solución en menos de 12 horas, utilizando los esquemas de grano fino. El grano es demasiado fino para obtener buenos resultados cuando el número de procesadores se incrementa. Sin embargo, cuando los esquemas de grano fino proporcionan resultados usando pocos procesadores, las soluciones se obtienen en menos tiempo que el que necesitan los algoritmos de grano grueso con el mismo número de procesos. Comparando ambos algoritmos de grano fino se puede ver que cuando se incrementa el número de procesadores, las propuestas de memoria compartida proporcionan soluciones en menos tiempo porque se invierte menos tiempo en las comunicaciones entre los procesos. Sin embargo, cuando el número de procesos es pequeño, las aproximaciones de memoria distribuida proporcionan resultados en menos tiempo.

De cualquier modo, para obtener los resultados exactos en menos tiempo que la versión secuencial del algoritmo, la mejor opción es usar los algoritmos de grano grueso. Si se tiene la posibilidad de usar una máquina que permita 24 procesos de memoria compartida, entonces es recomendable usar el algoritmo de grano grueso con memoria compartida, pero si en la máquina se pueden usar 32 o más procesos MPI, la mejor opción es usar el algoritmo de grano grueso con memoria distribuida.

Sin embargo, los resultados obtenidos usando los algoritmos de grano grueso no presentan una aceleración constante al aumentar el número de procesadores. Esto se debe a que la estructura de árbol que usa la búsqueda es altamente irregular, y cuando sus nodos se distribuyen entre los procesadores para realizar la búsqueda, en algunos casos el nodo con la solución óptima puede ser analizado por un proceso al principio de la búsqueda, y en otros casos el nodo con la solución puede estar demasiado profundo en la rama del árbol que va a analizar el proceso. Incluso cuando el nodo con la solución óptima no se encuentra demasiado profundo en el árbol, si otro proceso tiene demasiado trabajo acumulado y se realiza una sincronización, el esquema de balanceo de carga puede provocar un retraso en el análisis del nodo correcto. En definitiva, lo que ocurre se debe a un cambio en el orden de exploración de los nodos del árbol de búsqueda. Además, la forma de balancear afecta bastante a este comportamiento, y quizás habría que hacer un estudio más a medida del balanceo para las instancias usadas. En cambio, en las implementaciones de grano fino nunca puede ocurrir que se cambie el orden de exploración.

En general, el tiempo invertido en resolver una instancia del problema con las propuestas de grano grueso está relacionado con el número de nodos insertados en la lista CLIST usada por el algoritmo, es decir, el número de nodos que se borran de la lista OPEN para combinarse con los mejores subproblemas previos ya explorados (*nodos computados*), y esto, tal y como se ha mencionado, es impredecible dada la estructura irregular del árbol y el esquema de balanceo de carga. Para verificar esta relación, durante la ejecución paralela de grano grueso con memoria compar-

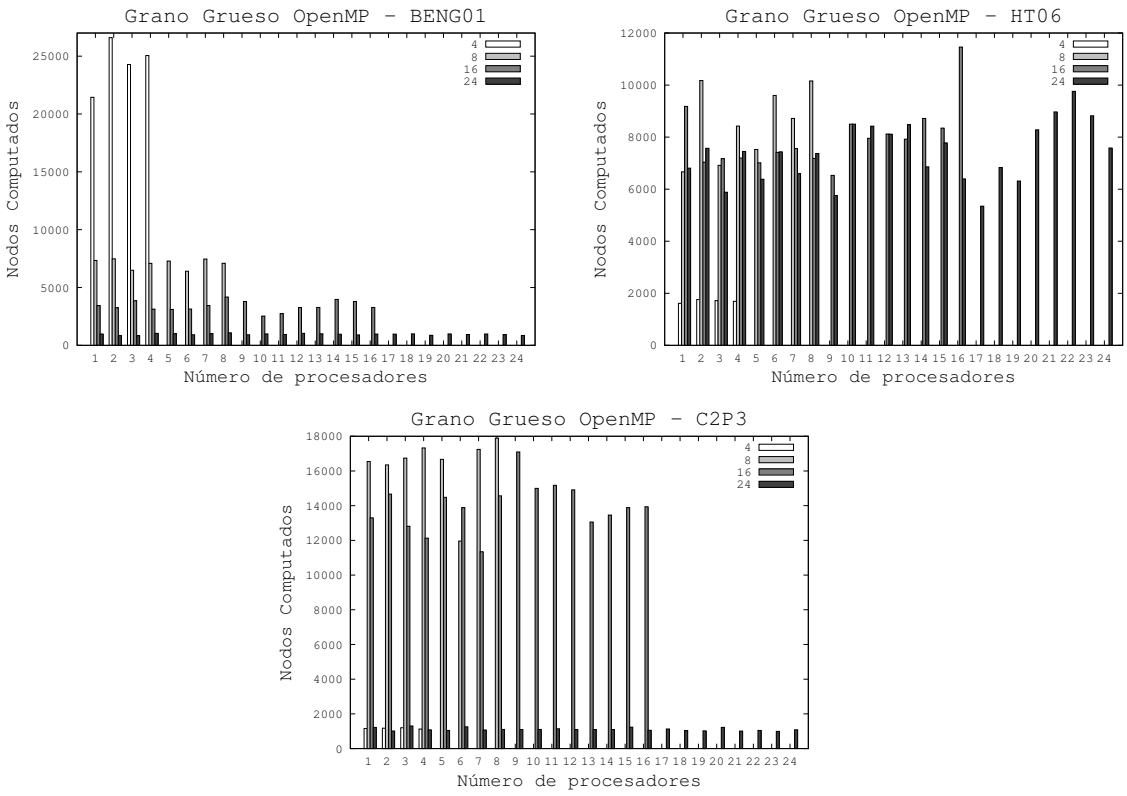

Figura 4.5: Distribución de nodos computados entre procesadores

tida se han calculado los nodos computados para tres instancias del problema, una con buena aceleración (BENG01), otra con mala aceleración (HT06), y otra con una aceleración irregular (C2P3), En la Figura 4.5 se muestra la distribución de los nodos computados entre los procesadores para las ejecuciones con 4, 8, 16 y 24 procesadores. Como se puede observar, para la instancia con buena aceleración el número de nodos computados va descendiendo a medida que se aumenta el número de procesadores, produciendo por tanto la bajada progresiva en el tiempo de ejecución. Para la instancia con mala aceleración, ocurre prácticamente lo contrario, el número de nodos computados va creciendo con el número de procesadores, produciendo el aumento en el tiempo de ejecución. Por último, con la instancia de aceleración irregular se producen altibajos en el número de nodos computados, comenzando con un número de nodos bajo para las ejecuciones de cuatro procesadores, incrementándose posteriormente y volviendo a descender para el número más alto de procesadores, produciendo las variaciones en el tiempo empleado en la ejecución. Es decir, se confirma la relación directa entre el número de nodos computados y el

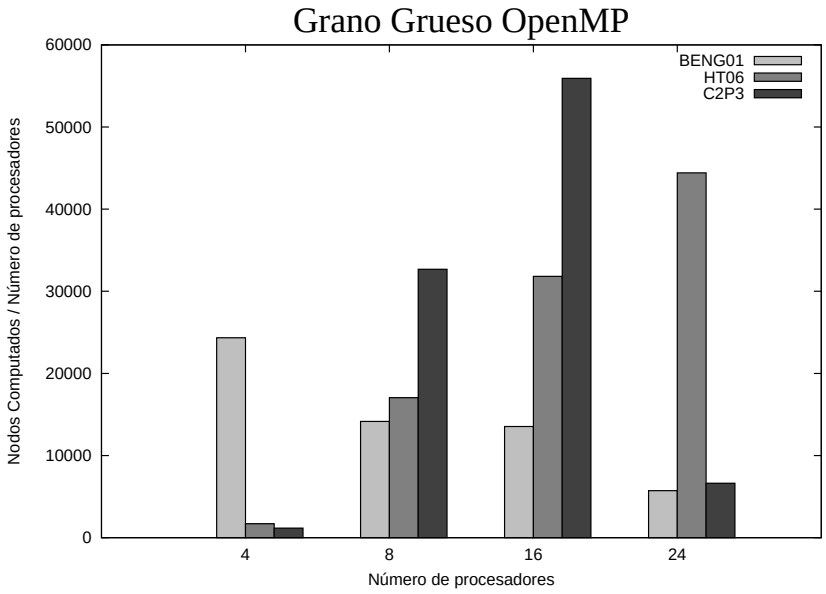

Figura 4.6: Trabajo asociado con un proceso para tres instancias

tiempo de ejecución. La Figura 4.6 muestra la relación *nodos computados : número de procesadores* para las mismas tres instancias del 2DSPP, permitiendo ver con más claridad que la distribución de nodos computados por cada procesador se decrementa para la instancia BENG01, se incrementa para la instancia HT06 y es irregular para la instancia C2P3.

Además, se puede verificar que el reparto del trabajo entre los procesadores durante una ejecución no es tan equilibrado como ocurría con el 2DCSP. Esto puede deberse a que la resolución de estas instancias es más compleja ya que se permite la rotación de las piezas y hay que colocarlas todas, a diferencia del 2DCSP, con lo cual hay que probar entre todas las posibles combinaciones de piezas. En este sentido, las heurísticas implementadas que se usan como cotas para el 2DCSP ayudan en cierto modo a escoger las piezas que mejores soluciones podrían dar, en función de la disponibilidad y del beneficio asociado a cada pieza. Pero para el 2DSPP no existe esa guía en la búsqueda ya que al final se han de colocar todas las piezas en el material. A modo de ejemplo, la Tabla 4.7 muestra, para instancias de diferente tamaño del 2DCSP y del 2DSPP, el número total de piezas de cada instancia y el número de piezas que se han distribuído en el material para ser cortadas en la solución óptima. Como se puede observar, aunque las instancias del 2DCSP constan de más piezas que las instancias del 2DSPP, se colocan en el material pocas piezas y además no se tiene en cuenta su orientación, con lo que la dificultad del problema es menor. Incluso en algunas instancias del 2DCSP la solución final coincide con la encontrada por

Instancia	Total Piezas	Piezas Cortadas	N. Generados	N. Computados	N. Insertados
ATP33s	224	21	482113	809	1816
CW6	149	16	190095	913	3405
CL_07_50_09	50	7	2326030	2527	4362
BENG01	20	20	70455190	26819	129644
C2P3	25	25	156472357	52137	283933
HT06	25	25	391805322	83405	470003

Tabla 4.7: Piezas cortadas y nodos en la solución óptima

la heurística usada como cota inferior. Además, se puede comprobar que el número medio de nodos generados, computados e insertados para las instancias del 2DCSP es mucho más bajo que para el 2DSPP, y dado que se ha comprobado la relación entre los nodos computados y el tiempo necesario para encontrar la solución óptima, es lógico que se tarde más en resolver las instancias del 2DSPP.

4.4.2. Conclusiones

Como se vio anteriormente, cuando el número de piezas a colocar en el material aumenta, el tiempo empleado en la búsqueda de la solución exacta puede incrementarse demasiado, por lo que para instancias con un número de piezas elevado es aconsejable llevar a cabo alguna paralelización del algoritmo. Por ello se han utilizado dos paralelizaciones del algoritmo exacto para el 2DSPP de guillotina donde se permite la rotación de las piezas: un algoritmo de grano fino y un algoritmo de grano grueso. Para cada algoritmo se han utilizado dos implementaciones, con memoria compartida y con memoria distribuida. Los resultados obtenidos usando algunas de las instancias del problema disponibles en la literatura indican que la mejor opción para obtener la solución exacta en menos tiempo que la versión secuencial, es usar los algoritmos de grano grueso. Las implementaciones de grano fino no proporcionan buenas aceleraciones, debido a que el trabajo que se puede paralelizar en cada computación de un nodo está limitado, pero en ningún momento se cambia el orden de exploración de los nodos del árbol de búsqueda. Con las implementaciones de grano grueso, en ocasiones la aceleración no es realmente buena o no es constante, pero esto se debe a que en este caso sí se cambia el orden de exploración de los nodos y la estructura del árbol de búsqueda es altamente irregular. Además el esquema de balanceo de carga también afecta, pudiendo provocar un retraso en la obtención de la solución. En este sentido, se ha podido comprobar la relación directa entre los nodos computados y los tiempos empleados en las ejecuciones.

La principal diferencia con respecto a los resultados obtenidos con el 2DCSP es la irregularidad en la aceleración para las implementaciones de grano grueso. Sin embargo, como ya se ha dicho, en este caso la búsqueda es aún más irregular ya que hay que introducir todas las piezas en el material, con lo que hay que probar toda

las combinaciones posibles de piezas. En ese proceso las heurísticas implementadas como cotas en el 2DCSP no aportan una guía, puesto que éstas ayudaban a escoger las piezas que mejores soluciones podían dar en función de la disponibilidad y del beneficio asociado a cada pieza, pero para el 2DSPP esto no resulta útil, ya que hay que colocarlas todas.

4.5. Solución multi-objetivo

Igual que en el caso del 2DCSP, en este problema se ha tenido en cuenta como segundo objetivo a minimizar el número de cortes necesarios para obtener las piezas demandadas, ya que en las industrias en las que el material es barato, el objetivo de minimizar el largo de la materia prima pasa a tener un poco menos de relevancia. Sin embargo, el criterio de minimizar el número de cortes sigue siendo importante, puesto que cuantos menos cortes se hagan menos se deteriorará la maquinaria industrial y el proceso de corte se realizará con mayor velocidad.

Como ya se ha mencionado anteriormente, aplicar un método exacto para la resolución de un problema multi-objetivo es inviable debido a la magnitud y complejidad del árbol de búsqueda. Además, si se optara por abordar un problema multi-objetivo tratando de unificar los distintos objetivos en una única función objetivo, se deberían conocer a priori las demandas específicas en cada momento en cuanto a los dos objetivos, para poder combinarlos de la mejor manera. Si las demandas cambian, se necesitarían nuevas ejecuciones. Usando estrategias que afronten el problema multi-objetivo como tal, sólo se requiere una ejecución, y después la persona encargada de tomar las decisiones puede seleccionar la solución más adecuada, dependiendo de las demandas en cada momento. Por lo tanto se ha decidido utilizar metaheurísticas multi-objetivo para resolver el MOSPP. Entre ellas se han seleccionado los MOEAs ya que simplemente es necesario emplear un esquema general del algoritmo concreto y definir las características del problema. Tal y como se comentó para el 2DCSP multi-objetivo, tanto para optimización mono-objetivo como para optimización multi-objetivo, cuando se diseñan propuestas metaheurísticas para la solución de un problema dado, hay muchas decisiones de diseño que son cruciales para el impacto del método de resolución. Una de las decisiones principales es la relativa a la representación de los individuos de la población. Igual que se explicó en la introducción de la solución multi-objetivo para el 2DCSP (Sección 3.6), aquí se pueden diferenciar los mismos tres grupos de propuestas metaheurísticas para el 2DSPP, dependiendo del tipo de codificación elegida para las soluciones.

A continuación se exponen dos tipos diferentes de esquemas de codificación implementados para la resolución del 2DSPP multi-objetivo mediante MOEAs: un

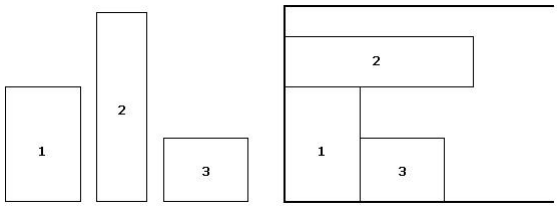

Figura 4.7: Distribución de patrones para el cromosoma '1 3 H 2 V 010'

esquema directo correspondiente al tercer grupo de metaheurísticas, y varios esquemas de codificación basados en hiperheurística, es decir, pertenecientes al segundo grupo. De este modo se podrá ver en qué medida influye el tipo de codificación en la calidad de las soluciones obtenidas.

4.5.1. Esquema de codificación directa

El enfoque inicial está basado en una representación que implica una definición completa de la distribución guillotinable de piezas. Igual que en la codificación directa del 2DCSP, la búsqueda se lleva a cabo directamente en el espacio de todas las posibles distribuciones de piezas y los operadores de variación deben ser diseñados específicamente para manipular tales representaciones.

4.5.1.1. Representación

Se ha usado la notación postfija introducida en el Capítulo 3 para representar las soluciones candidatas [148, 171], al igual que en el 2DCSP. Por lo tanto, de nuevo los operandos son los identificadores de las piezas, mientras que los operadores son 'V' y 'H' haciendo referencia a las concatenaciones en vertical y horizontal (Figura 4.7). Si las restricciones del problema se violan al concatenar en vertical dos patrones, el operador 'V' responsable pasa a ser un operador 'H'. Esto nunca ocurrirá a la inversa ya que el operador 'H' siempre se puede aplicar sin que se produzca una violación de las restricciones dado que la materia prima se considera de largo ilimitado. En este problema, para constituir un cromosoma válido cada pieza debe aparecer una vez. Al final del cromosoma se ha agregado una cadena de bits del mismo tamaño que el número de piezas, determinando si cada pieza debe ser rotada.

Para la generación inicial de individuos, se ha establecido un orden aleatorio de piezas y se ha aplicado una probabilidad uniforme para determinar los operadores y la orientación de las piezas. Cada individuo se crea del máximo tamaño posible, dado que, a diferencia del 2DCSP, aquí se considera que se deben obtener todas las piezas demandas.

4.5.1.2. Evaluación de los objetivos

Como en el caso del 2DCSP, cada cromosoma representa una cierta distribución de piezas sobre la materia prima. La codificación usada proporciona la información de cómo se deben combinar o colocar las piezas sobre el material. En base a esta información, se pueden evaluar ambos objetivos de optimización, largo total y número de cortes necesarios. Con este propósito, los métodos que se han aplicado aquí se basan en el uso de pilas y de la notación postfija que representa al cromosoma [148].

Para la evaluación del segundo objetivo (número total de cortes necesarios) se han considerado dos tipos de evaluaciones: para cortes de tipo guillotina y cortes de tipo no-guillotina. La representación del cromosoma usada permite realizar ambas operaciones. Esto se ha tenido aquí en cuenta para poder realizar comparativas con otros trabajos relacionados que llevan a cabo los dos procedimientos. Cuando no sea necesario realizar cortes de guillotina, el número de cortes requerido para llevar a cabo el trabajo se puede reducir significativamente si las piezas ya cortadas se eliminan del material y se hacen cortes combinados para el resto de bordes alineados. Si se tiene en cuenta la restricción de que los cortes sean de tipo guillotina, el número de cortes involucrados cambia. La Figura 4.8 muestra cómo la misma distribución de piezas supone un número diferente de cortes cuando se aplica un método de guillotina o no-guillotina.

En el caso de usar cortes de tipo guillotina, se ha aplicado un método iterativo para calcular el número de cortes realizados. El cromosoma se recorre de izquierda a derecha, interpretando cada elemento, creando la construcción indicada y calculando los anchos y largos parciales. Para cada combinación vertical u horizontal de piezas, se necesita al menos un corte. Si los rectángulos combinados no coinciden en largo (para las construcciones verticales) o en ancho (para las construcciones horizontales), se necesita un corte extra. Al final del proceso se obtiene el patrón final completo. En este caso, el valor del primer objetivo (largo total requerido) viene dado inmediatamente por el largo del patrón final resultante. Este proceso se describe en el Algoritmo 6. El algoritmo incorpora las comprobaciones que aseguran que la evaluación de los cromosomas siempre satisfaga la restricción del ancho del material. Para simplificar el pseudocódigo, se han omitido las consideraciones relativas a la orientación de las piezas.

Por otra parte, para la evaluación de los objetivos en modo no-guillotina, se necesita un análisis inicial del cromosoma para calcular las coordenadas inferior izquierda (x_1, y_1) y superior derecha (x_2, y_2) de cada pieza colocada sobre la materia prima. El procedimiento empieza con la distribución o combinación de piezas final, que luego se divide en los últimos subpatrones que lo han constituido. Después, el mismo método se invoca recursivamente hasta que todas las piezas individuales estén

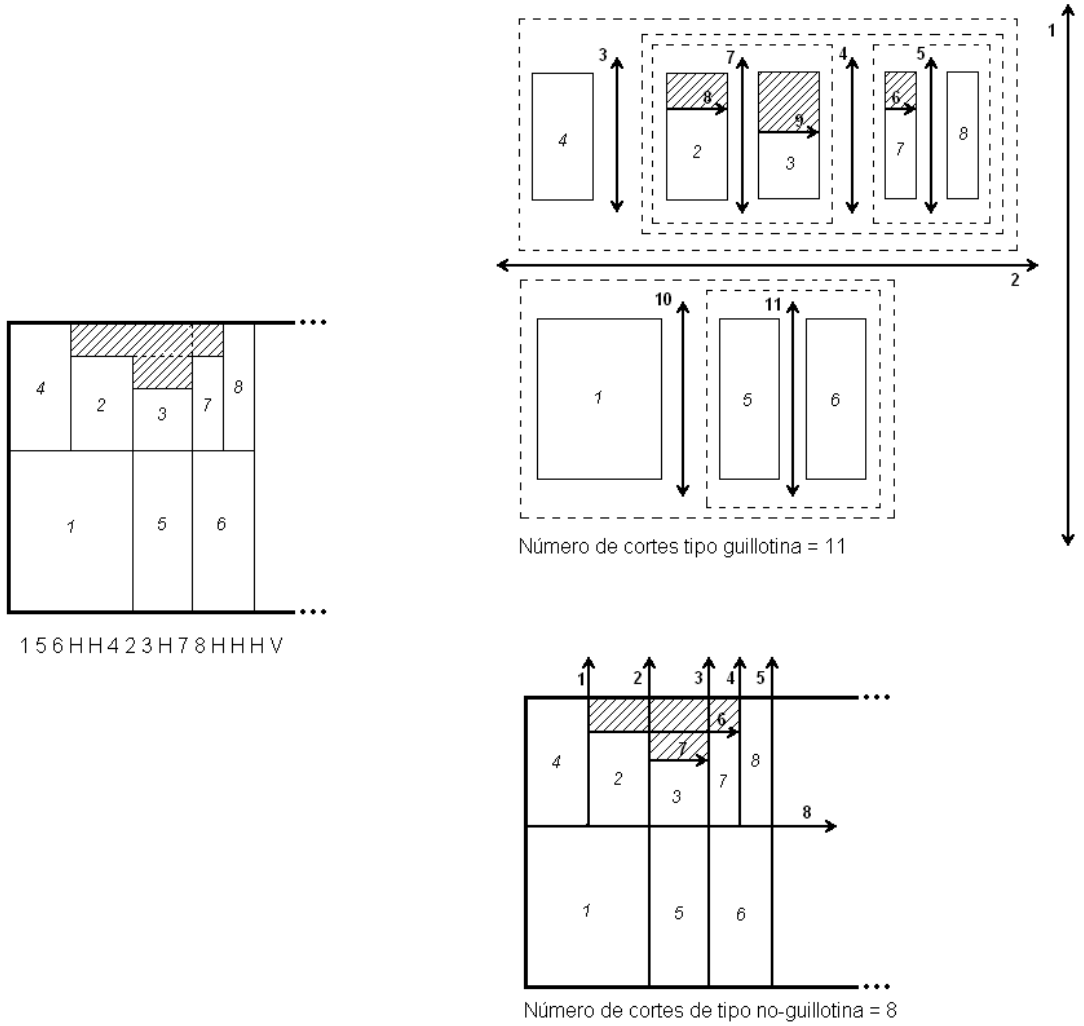

Figura 4.8: Comparativa entre el proceso de corte de guillotina y el de no-guillotina

separadas y con sus coordenadas calculadas. Como en el caso de cortes de guillotina, el largo del material se obtiene de las dimensiones del rectángulo o composición final. Para evaluar el número de cortes, se asume que todas las piezas están dispuestas con sus ejes ortogonales a los ejes del material, sin superposición entre ellas. En esta situación, el número máximo de cortes requeridos en un patrón de corte compuesto por n piezas es $4n$, ya que cada pieza tiene cuatro ejes. Entonces, se comprueba si cada eje de cada pieza coincide con uno de los ejes del material. Si esto ocurre, ese corte no es necesario, así que el número total de cortes puede decrementarse en uno. Después, el resto de ejes que no coincidan con los ejes del material se comparan con

Algoritmo 6 Evaluación de los objetivos del 2DSPP: modo guillotina

```
1: cutsNum = 1;
2: for (i = 0 hasta sizeOfChromosomeVbles) do
3:    if (isOperator(chromosome(i)) == false) then
4:       stack.push(chromosome(i));
5:    else
6:       cutsNum = 1 + cutsNum;
7:       pieceA = stack.pop();
8:       pieceB = stack.pop();
9:       if (chromosome(i) == H) then
10:         length(newPiece) = length(pieceA) + length(pieceB);
11:         width(newPiece) = max(width(pieceA), width(pieceB));
12:         if (width(pieceA) != width(pieceB)) then
13:            cutsNum = cutsNum + 1;
14:         end if
15:      else
16:         length(newPiece) = max(length(pieceA), length(pieceB));
17:         width(newPiece) = width(pieceA) + width(pieceB);
18:         if (width(newPiece) ≥ width(motherSheet)) then
19:            chromosome(i) = H;
20:            deshacer cambios hechos en la iteración i;
21:            repetir la iteración i;
22:         end if
23:         if (length(pieceA) != length(pieceB)) then
24:            cutsNum = cutsNum + 1;
25:         end if
26:      end if
27:      stack.push(newPiece);
28:   end if
29: end for
30: finalPiece = stack.pop();
31: totalLength = lenght(finalPiece);
32: if (width(finalPiece) != width(motherSheet)) then
33:    totalCuts = cutsNum + 1;
34: else
35:    totalCuts = cutsNum;
36: end if
```

los ejes del resto de piezas para encontrar alineaciones. Para cada par de ejes sólo se pueden dar dos situaciones: los dos rectángulos se tocan (en las esquinas o a lo largo de sus ejes) o no se tocan. Si se tocan, entonces sólo se necesita un corte para obtener el corte de ambos ejes, por lo que se decrementa el número de cortes en uno. Si los ejes no se tocan, entonces es necesario encontrar una línea que una los dos ejes sin que haya otra pieza en medio cruzando esa línea. Si ningún rectángulo cruza esa línea, se requiere un sólo corte para ambos ejes, por lo que de nuevo el número

Algoritmo 7 Evaluación de los objetivos del MOSPP: modo no-guillotina

```
construir finalPiece;
totalLength = length(finalPiece);
calcular coordenadas x1, y1, x2, y2 de finalPiece;
calcular coordenadas x1, y1, x2, y2 de cada pieza dentro de finalPiece;
totalCuts = 4 * numPieces;
for (i = 0 hasta numPieces) do
    numAxes = bordes de la pieza i que coinciden con los bordes superior, inferior o izquierdo
    de la hoja de material;
    totalCuts = totalCuts - numAxes;
end for
for (cada par de piezas) do
    if (dos bordes horizontales o verticales están alineados) then
        if (las piezas no se tocan una con la otra) then
            if (hay otra pieza entre ambas) then
                if (la pieza en medio no corta los bordes alineados) then
                    totalCuts = totalCuts - 1;
                end if
            else
                totalCuts = totalCuts - 1;
            end if
        else
            totalCuts = totalCuts - 1;
        end if
    end if
end for
```

de cortes se decrementa en uno. En todo este proceso hay que tener en cuenta que para cada alineación de ejes al menos se necesita computar un corte. El Algoritmo 7 describe brevemente el método de evaluación para los cortes que no son de tipo guillotina. Una vez más, para simplificar el pseudocódigo, las consideraciones de las orientaciones se han omitido.

4.5.1.3. Operadores

Como se ha usado una codificación que representa implícitamente las soluciones del problema, el tipo de operadores que se aplique debe tratar con las características específicas del problema. Para el operador de cruce, se han testeado varias alternativas y tras analizar los resultados obtenidos, se ha decidido utilizar el *Partially Mapped Crossover* (PMX) [86], dado su buen comportamiento cuando se aplica al 2DSPP [148, 171]. La técnica se basa en la recombinación de dos cadenas de cromosomas donde se considera sólo la información de las piezas, es decir, los operadores y los bits de las orientaciones no se tienen en cuenta para la aplicación de este ope-

rador. Su funcionamiento es el mismo que el detallado en la Sección 3.6.1.3. para el 2DCSP multi-objetivo.

El operador de mutación aplicado [148, 171] también es similar al aplicado en el 2DCSP multi-objetivo y funciona de la siguiente manera. Primero se eligen aleatoriamente dos elementos del cromosoma p_1 y p_2. Ambos elementos representan números de piezas u operadores y p_1 está más cerca de la izquierda en el cromosoma. Si ambos son números de piezas u operadores, o p_1 es un operador y p_2 es una pieza, se intercambian. Si p_1 es un número de pieza y p_2 es un operador, sólo se intercambian cuando, después de realizar el cambio, se siga manteniendo para cada operador la condición de la fórmula 3.2 para que la notación sea válida. Luego, se elige aleatoriamente un operador del cromosoma y se intercambia bajo la probabilidad de mutación. Finalmente, se selecciona de manera aleatoria un bit de orientación y se intercambia de '0' a '1' o viceversa, siempre bajo la probabilidad de mutación. Ambos operadores, cruce y mutación, aseguran que sólo se van a generar nuevos cromosomas válidos.

4.5.2. Esquemas de codificación basados en hiperheurística

Tal y como se ha mostrado en la Sección 4.2, en la bibliografía se puede encontrar una gran variedad de heurísticas de colocación de piezas para el 2DSPP que permiten obtener soluciones de alta calidad en un tiempo computacionalmente aceptable. La existencia de tal variedad de heurísticas, unido al hecho de que cada tipo de heurística se comporta mejor para un determinado conjunto de instancias del problema, dificulta considerablemente la selección a priori de una heurística para resolver una instancia en particular. En este sentido, las hiperheurísticas son un procedimiento más general que trata de elegir y/o combinar las heurísticas correctas de forma que las ventajas de unas compensen por las desventajas de otras y viceversa [33]. La motivación que hay detrás de este enfoque, es que una vez que se ha desarrollado un algoritmo hiperheurístico, los dominios de varios problemas e instancias podrían abordarse sólo sustituyendo las (meta)-heurísticas de bajo nivel. Por lo tanto, el propósito de usar una hiperheurística aquí es elevar el nivel de generalidad en el que la mayoría de los actuales sistemas de (meta)-heurísticas funcionan.

Aquí se ha propuesto una codificación indirecta que determina una secuencia de tipo de heurística de colocación de piezas (o heurística de bajo nivel) a aplicar y número de piezas que cada heurística debe distribuir.

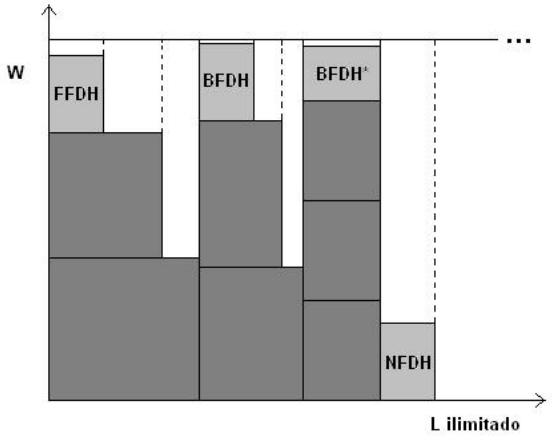

Figura 4.9: Colocación de una pieza según cada heurísticas de bajo nivel

4.5.2.1. Heurísticas de bajo nivel

La primera decisión necesaria para implementar el enfoque basado en una hiperheurística es la definición del conjunto de heurísticas de colocación que se va a utilizar. En la literatura se puede encontrar una enorme cantidad de heurísticas para el 2DSPP de tipo no-guillotina, pero el número de propuestas para el caso guillotinable no es demasiado extenso. Las heurísticas constructivas orientadas a niveles, anteriormente utilizadas para el 2DCSP, se aplican en algunos trabajos para el 2DSPP [143] mostrando buenos resultados. Por ello se han utilizado algunas para introducirlas en la hiperheurística. Las heurísticas que se han empleado son (Figura 4.9): *Next Fit Decreasing Height* (NFDH) [45], *First Fit Decreasing Height* (FFDH) [45], *Best Fit Decreasing Height* (BFDH) [143] y *Best Fit Decreasing Height** (BFDH*) [21]. Las tres primeras son las mismas heurísticas orientadas a niveles que se utilizaron para el 2DCSP y actúan de igual modo, salvando dos principales diferencias. Primero, en este problema existe la posibilidad de orientar las piezas, por lo que todas ellas fijan la orientación de los objetos de modo que su ancho sea mayor que su largo, y una vez fijada la orientación, las piezas se ordenan de mayor a menor largo. En segundo lugar, en este problema el material tiene una longitud considerada "infinita" y no se para de colocar piezas hasta que todo el conjunto de piezas esté distribuido sobre el material. La última heurística BFDH* trata de mejorar la heurística BFDH permitiendo la rotación de los objetos, de modo que cuando el algoritmo intenta colocar el objeto actual en un espacio prueba ambas orientaciones. Ésta no se ha podido utilizar en el 2DCSP debido a que en ese problema no se permite la rotación de las piezas.

4.5.2.2. Representaciones

Se han diseñado cuatro nuevas representaciones del cromosoma basadas en hiperheurísticas para resolver el 2DSPP multi-objetivo. Así se podrá comprobar cuál de ellas proporciona los resultados más adecuados para el problema y en qué medida influye el tipo de codificación elegida en la calidad de las soluciones obtenidas. Todas las codificaciones diseñadas aplican una secuencia de heurísticas de bajo nivel a unos conjuntos de piezas. El orden y la orientaciones de las piezas puede venir dado por la propia heurística o indicarse explícitamente en la representación, por lo que en la explicación de cada representación del cromosoma se explicarán estos aspectos.

Esquema de codificación basado en hiperheurística *(h, p)*

En este caso, un individuo se representa mediante una secuencia de 2-tuplas (h, p), donde h es el identificador de la heurística de colocación que se va a aplicar, y p es el número de piezas que esa heurística debe posicionar en el material disponible (Figura 4.10). Los cromosomas tienen una longitud variable j, que va desde $j = 1$ (representación con una 2-tupla (h, n), la misma heurística distribuye todas las piezas disponibles) hasta $j = n$ (los n objetos disponibles se distribuyen de manera independiente, es decir, $\forall i \in [1, n], p_i = 1$). Nótese que en una representación válida todos los objetos demandados deben ser distribuidos por las heurísticas, es decir, $(\sum_{i=1}^{j} p_i) = n$. Para la generación inicial de individuos, se producen 2-tuplas (h, p) hasta que no queden más piezas por colocar. Para cada par, h es seleccionado aleatoriamente entre las cuatro heurísticas usadas y p se genera aleatoriamente en el intervalo $[1, a_p]$, donde a_p es el número de piezas aún disponibles.

Esquema de codificación basado en hiperheurística *(h, p) + or + ot*

En el primer esquema de codificación, el orden y la orientación de las piezas viene dado por las heurísticas de colocación de bajo nivel. Fijan la orientación de los objetos de tal manera que sus anchos no sean menores que sus largos, y luego se ordenan de mayor a menor largo. Sin embargo, el orden y la orientación pueden venir dados externamente. Es decir, un individuo se puede representar por una secuencia de 2-tuplas (h, p), como en el primer esquema de codificación, pero al final del cromosoma se puede adjuntar la secuencia de piezas ordenadas y una cadena de bits del tamaño del número de piezas. Esta cadena de bits determina si cada pieza debe de rotarse o no (Figura 4.11). De este modo, el cromosoma tiene una longitud variable j que va desde $j = 2n+1$ a $j = 3n$. Para la generación inicial de individuos, se generan las 2-tuplas (h, p) como en el primer esquema de codificación basado en hiperheurística. Luego, se generan aleatoriamente la secuencia de piezas ordenadas y las orientaciones.

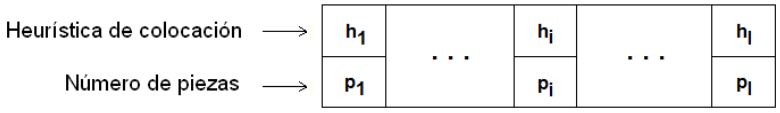

Figura 4.10: Representación basada en hiperheurística ***(h, p)***

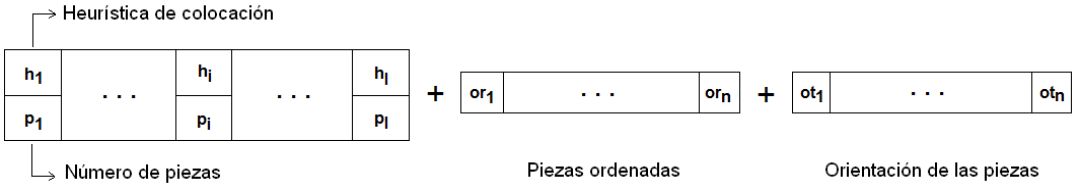

Figura 4.11: Representación basada en hiperheurística ***(h, p) + or + ot***

Esquema de codificación basado en hiperheurística *(h, p, o) + ot*

Si se quisiera ordenar los objetos siguiendo algún criterio, solamente sería necesario especificar el tipo de ordenación para cada heurística de bajo nivel que forme parte de la hiperheurística. De este modo, la secuencia de piezas ordenadas que se usa en la segunda representación se puede eliminar, es decir, un individuo se representará por una secuencia de 3-tuplas (h, p, o), donde o es el criterio de ordenación, y una cadena de bits del mismo tamaño que el número de piezas indicando la orientación de las piezas (Figura 4.12). Así el cromosoma tiene una longitud variable j que va desde $j = n+1$ a $j = 2n$. Los criterios básicos de ordenación que se usan son: largos decrecientes, anchos decrecientes, áreas decrecientes y perímetros decrecientes. Para la generación inicial de individuos, se crean secuencias de 3-tuplas (h, p, o), donde h y p se obtienen como en el primer esquema de codificación, y el criterio de ordenación se genera aleatoriamente entre las cuatro posibilidades disponibles. Luego, se generan al azar las orientaciones.

Esquema de codificación basado en hiperheurística *(h, p, o, r)*

Siguiendo la representación previa, se puede querer orientar los objetos siguiendo algún criterio también. De este modo la cadena de bits que indica la orientación de las piezas se puede eliminar, es decir, un individuo se representará mediante una secuencia de 4-tuplas (h, p, o, r), donde r es el criterio de rotación (Figura 4.13). Así, el cromosoma tiene una longitud variable j que va desde $j = 1$ a $j = n$, como en el primer esquema de codificación basado en hiperheurística. Los criterios de rotación que se usan son; anchos mayores o iguales que largos, largos mayores

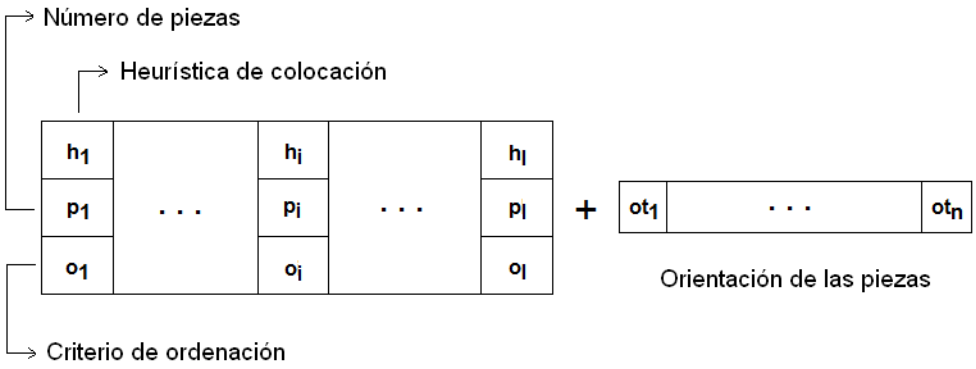

Figura 4.12: Representación basada en hiperheurística ***(h, p, o) + ot***

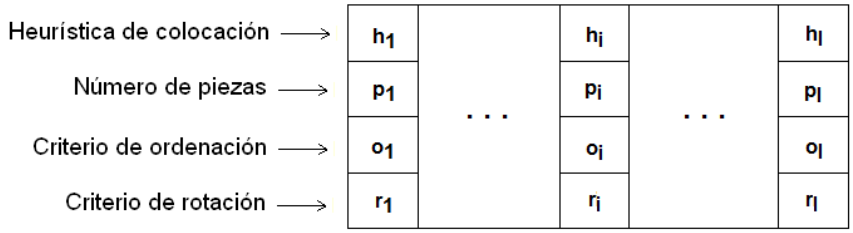

Figura 4.13: Representación basada en hiperheurística ***(h, p, o, r)***

o iguales que anchos, no rotar ningún objeto y rotarlos todos. Para la generación inicial de individuos, se generan secuencias de 4-tuplas (h, p, o, r), donde h, p y o se obtienen como en el esquema de codificación previo, y el criterio de rotación se genera aleatoriamente entre las cuatro posibilidades.

4.5.2.3. Evaluación de los objetivos

En la codificación directa presentada en la sección previa, los cromosomas representan explícitamente la solución del problema, es decir, la distribución de las piezas sobre el material. Sin embargo, en estas codificaciones indirectas, los MOEAs utilizan poblaciones de individuos que representan hiperheurísticas, por lo que el cromosoma debe ser interpretado para obtener la solución del problema. La representación se analiza de izquierda a derecha, aplicando para cada tupla (h, p), (h, p, o) o (h, p, o, r), la heurística h para posicionar las siguientes p piezas. Estas piezas se distribuyen de acuerdo con sus correspondientes términos de ordenación y orientación. Cuando las piezas se colocan en la parte superior de niveles existentes, se crean construcciones verticales. Cuando se abre un nuevo nivel a la derecha de los

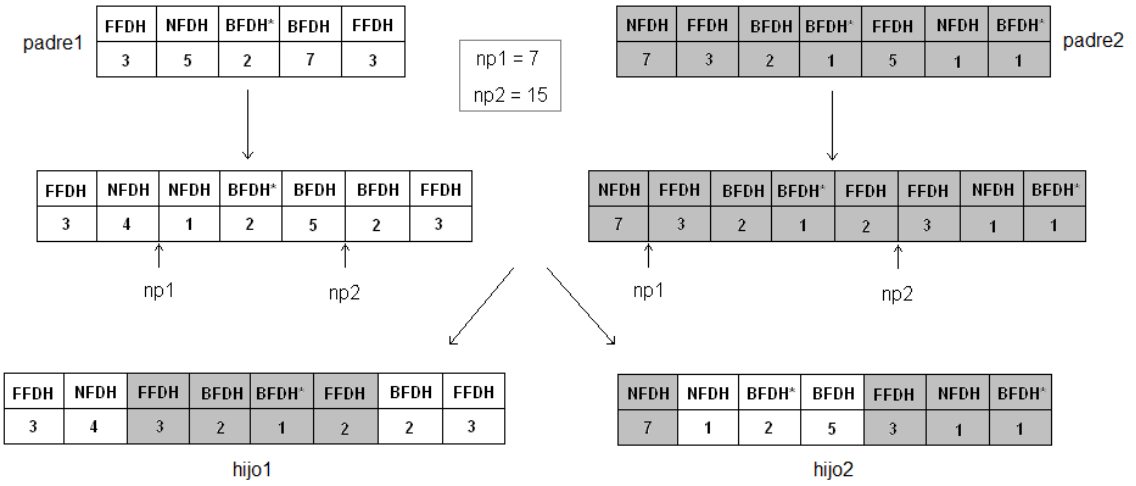

Figura 4.14: Operador de cruce de dos puntos

existentes, se genera una combinación horizontal entre los niveles. La decodificación del cromosoma proporciona una solución del problema representada mediante una notación postfija de cortes verticales y horizontales. Entonces, esa solución se puede evaluar del mismo modo que en la Sección 4.5.1.2, para obtener los valores de los dos objetivos: largo total y número de cortes necesarios.

4.5.2.4. Operadores

Se ha diseñado, implementado y analizado un conjunto de operadores de cruce y mutación para cada una de las representaciones anteriores. Algunos operadores son más generales, y otros más específicos para poder tratar con ciertas restricciones de las representaciones.

Después de testearlos, se ha seleccionado el cruce o recombinación de dos puntos que considera el número de piezas acumuladas dentro de la representación. Para ello se generan aleatoriamente dos números de piezas np_1 y np_2 , tal que $1 \leq np_1 < np_2 < n$. Para cada padre, es necesario encontrar las posiciones del cromosoma donde el número de piezas acumuladas sumen np_1 y np_2. Cuando la suma de las piezas no coincida con el valor generado np, la primera tupla (h_i, p_i), (h_i, p_i, o_i) o (h_i, p_i, o_i, r_i), que verifique $\sum_{m=1}^{i} p_m > np$, se divide en dos partes (h_j, p_j) y (h_k, p_k), (h_j, p_j, o_j) y (h_k, p_k, o_k) o (h_j, p_j, o_j, r_j) y (h_k, p_k, o_k, r_k), tal que $h_j = h_k = h_i$, $p_k = (\sum_{m=1}^{i} p_m) - np$, y $p_j = p_i - p_k$. Una vez se tienen identificados los dos puntos de cruce dentro de ambos padres, las tuplas centrales de los mismos se intercambian para generar los dos nuevos hijos (Figura 4.14). Si se está usando una codificación

en la que el orden y la orientación de las piezas vienen dados de manera explícita, es decir, no mediante criterios (segundo y tercer esquema de codificación basado en hiperheurística), se necesitan más operaciones para terminar el cruce. En ese caso, se aplicará la recombinación *Partially Mapped Crossover* [86] a las piezas ordenadas, y un cruce de un punto a la cadena de bits de la orientación.

De los operadores de mutación probados, el que ha mostrado mejor comportamiento está basado en la aplicación de tres tipos diferentes de movimientos en el cromosoma. Cada uno de los movimientos se aplica bajo la probabilidad de mutación p_m, es decir, se generan tres probabilidades - una por movimiento - y se realizan aquellos cuya probabilidad sea inferior a p_m.

- *Añadir*: aleatoriamente selecciona una tupla (h_i, p_i), (h_i, p_i, o_i) o (h_i, p_i, o_i, r_i) dentro de la representación. Entonces se genera una nueva tupla (h, p), (h, p, o) o (h, p, o, r) donde h es una heurística seleccionada al azar, o es un criterio de ordenación seleccionado al azar, r es un criterio de rotación seleccionado al azar, y p es un número aleatorio del intervalo $[1, p_i]$. Luego, p_i se actualiza con el valor $p_i - p$. Si después de actualizar, $p_i \neq 0$, las tuplas de las posiciones $i, \ldots, j$ se desplazan una posición a la derecha, incrementando el largo total del individuo ($j = j + 1$). Finalmente, la nueva tupla se introduce en la posición i (Figura 4.15).

- *Eliminar*: aleatoriamente selecciona una tupla (h_i, p_i), (h_i, p_i, o_i) o (h_i, p_i, o_i, r_i) de la representación. Si la tupla seleccionada es la última de la representación ($i = j$), entonces $p_{i-1} = p_{i-1} + p_i$. En otro caso, $p_{i+1} = p_{i+1} + p_i$ y las tuplas de las posiciones $i + 1, \ldots, j$ se desplazan una posición a la izquierda. En ambos casos, la longitud se actualiza ($j = j - 1$). Esta operación puede ser aplicada sólo si inicialmente $j > 1$ (Figura 4.16).

- *Reemplazar*: selecciona aleatoriamente una tupla (h_i, p_i), (h_i, p_i, o_i) o (h_i, p_i, o_i, r_i) de la representación. h_i se fija aleatoriamente a una de las heurísticas de bajo nivel definidas (Figura 4.17).

Si se está usando una codificación en la que el orden y la orientación de las piezas vienen dados de manera explícita, es decir, no mediante criterios (segundo y tercer esquema de codificación basado en hiperheurística), se necesitan más operaciones para terminar la mutación. En el caso de la orientación, cada elemento de la cadena de bits se intercambiará de 0 a 1 o viceversa, dependiendo de la probabilidad de mutación p_m. En el caso del orden, cada pieza de la secuencia ordenada se intercambiará por otra pieza bajo la probabilidad de mutación p_m.

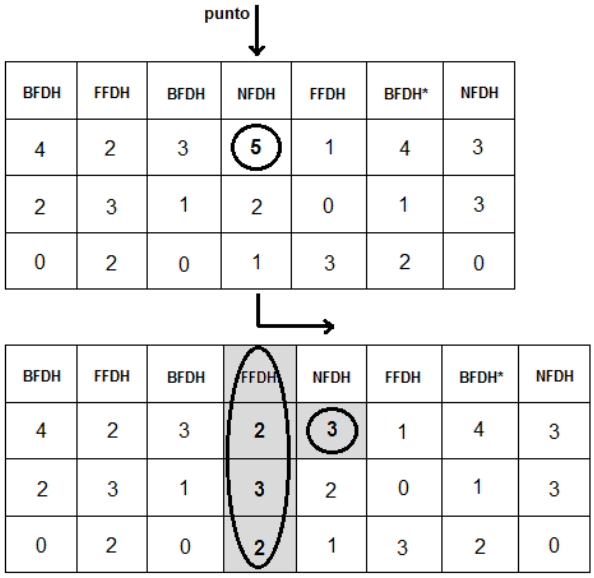

Figura 4.15: Operador de mutación *Añadir*

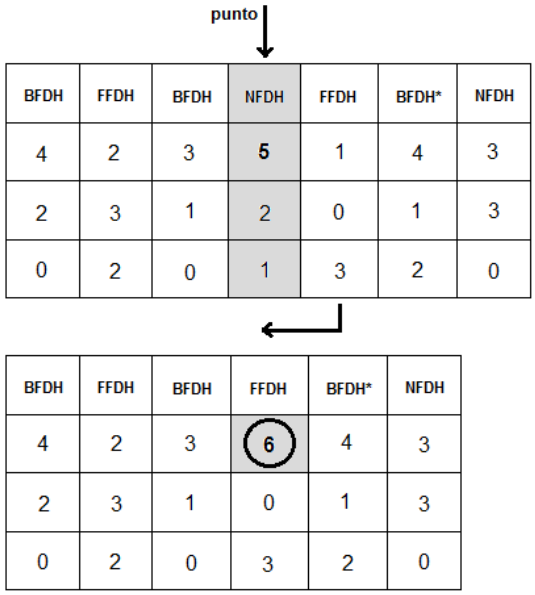

Figura 4.16: Operador de mutación *Eliminar*

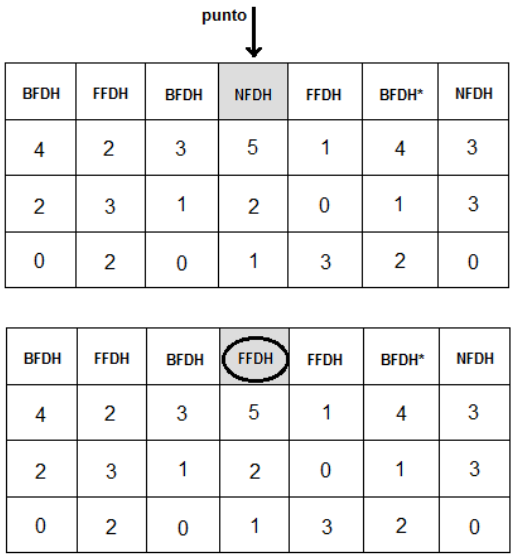

Figura 4.17: Operador de mutación *Reemplazar*

4.5.3. Resultados computacionales

Para evitar la implementación desde cero de MOEAs, tal y como se hizo para el 2DCSP se ha utilizado METCO, entorno explicado en el Capítulo 2. La evaluación experimental se ha llevado a cabo usando un cluster dedicado de 20 nodos de doble núcleo con un sistema operativo Debian GNU/Linux. Cada nodo consiste en dos Intel® Xeon 2.66 GHz, tiene 1GB de RAM y una red de interconexión Gigabit Ethernet. Las dos aproximaciones del problema (utilizando una codificación directa y una codificación basada en hiperheurística) se han compilado con *gcc 4.1.3* y MPICH *1.2.7*.

Para el estudio computacional inicial, se han usado las pequeñas instancias propuestas en [171]. Posteriormente se han usado los conjuntos de instancias Nice y Path [179], puesto que son ampliamente referenciadas en la literatura relacionada. Para las instancias de tipo `Nice.n`, los valores de largo y ancho de las n piezas se encuentran entre 1/4 y 4, siendo su área máxima de valor 7. Para el conjunto de instancias `Path.n`, el rango de valores de largo y ancho de las piezas es 1/100 y 100, siendo su área máxima 100. El problema con estas instancias (Nice y Path) se presenta a la hora de conocer la solución exacta mono-objetivo. Se sabe que el largo óptimo es de valor 100 para todas ellas. Sin embargo, en la literatura relacionada no se conocen las soluciones que indican la disposición de las piezas para ese largo, con lo cual no es posible evaluar el número de cortes para las mismas. Además, se trata

Esquema	Algoritmo	Cruce	Mutación	Población
Directa	NSGA-II	0.7	0.3	50
Hiperheurística	NSGA-II	0.6	0.4	50

Tabla 4.8: Configuración de las codificaciones

de instancias con muchas piezas, y los valores de ancho y largo de las piezas vienen dados por números con cinco cifras decimales. Estas características hacen que los tiempos de ejecución del algoritmo exacto paralelizado, anteriormente presentado en la Sección 4.3, se disparen de manera desorbitada, de tal manera que no se ha podido conocer la solución exacta mono-objetivo. Por lo tanto, a la hora de realizar comparativas de resultados con métodos mono-objetivo, se realizarán con respecto a heurísticas para las que sí se conoce el resultado aproximado.

Dentro de METCO se han definido varios tipos de individuos diferentes para resolver el 2DSPP: uno con la codificación directa basada en una representación postfija de la distribución de los patrones y cuatro con las codificaciones basadas en hiperheurística. Para cada aproximación, se ha definido la representación correspondiente, la evaluación de los objetivos, la generación de nuevos individuos y los operadores de cruce y mutación. Para cada implementación se han usado diferentes MOEAs. Después de realizar un ajuste inicial, se ha seleccionado el algoritmo y los parámetros que mejores resultados proporcionan para cada una de las codificaciones (Tabla 4.8). Nótese que, igual que en otros trabajos relacionados [171], el NSGA-II es el que mejor comportamiento muestra entre los algoritmos seleccionados. Así pues, se ha aplicado este algoritmo evolutivo para todos los experimentos siguientes.

4.5.3.1. Estudio del esquema de codificación directo

Inicialmente, el propósito de resolver el 2DSPP utilizando algoritmos evolutivos y el esquema de codificación directo es comprobar si los resultados obtenidos mejoran los únicos resultados que se conocen para resolver el problema como multi-objetivo y utilizando cortes guillotinables [171]. En dicho trabajo sólo se definen cinco instancias diferentes. Las tres primeras son irrelevantes porque son extremadamente sencillas y se resuelven de manera inmediata. Por esta razón, en esta primera evaluación experimental solo se han usado las dos instancias más largas propuestas en [171]. Estas instancias se denominan *problema 4* y *problema 5*. El problema 4 considera una materia prima de 10 unidades de ancho y 10 rectángulos diferentes de varias dimensiones. El problema 5 está compuesto por 20 piezas de diferentes dimensiones y usa un material de 10 unidades de ancho.

Se han llevado a cabo treinta ejecuciones con ambas instancias utilizando los

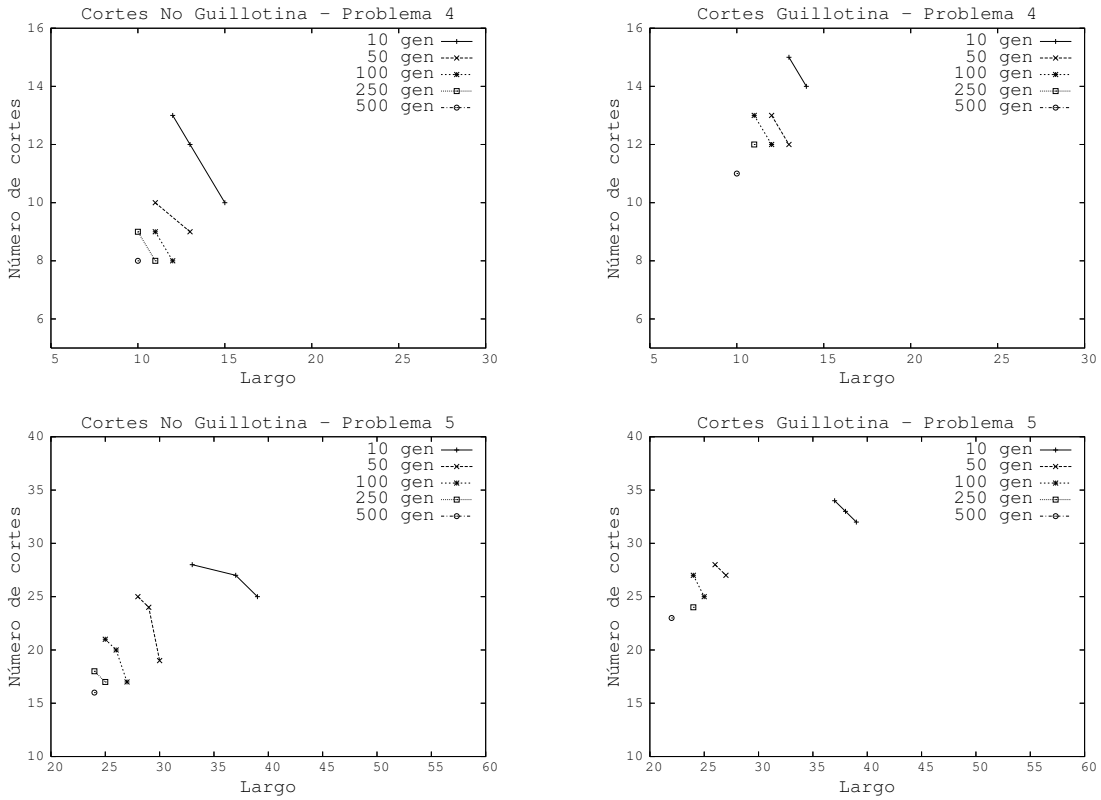

Figura 4.18: Evolución de los frentes de Pareto para una ejecución media

parámetros de la Tabla 4.8 y la codificación directa. Se han considerando ambos procesos de corte, guillotina y no-guillotina para las dos instancias. La Figura 4.18 muestra la evolución de los frentes, desde 10 generaciones (500 evaluaciones) hasta 500 generaciones (25000 evaluaciones), para una ejecución media usando NSGA-II. En este caso, como ejecución media se entiende una ejecución elegida de entre las 30 con un comportamiento intermedio, ni con el peor comportamiento, ni con el mejor. Nótese que el problema 4 es mucho más simple que el 5, y por ello converge más rápido e implica frentes de Pareto más pequeños. Para ambos problemas, la solución final no dominada se consigue antes de las 500 generaciones, demostrando la buena convergencia de esta aproximación evolutiva. Además. los resultados demuestran que la aplicación de cortes de guillotina requiere un número mayor de cortes que los necesarios para obtener las piezas con cortes de no-guillotina.

La Figura 4.19 muestra los mejores frentes de Pareto obtenidos para el problema 4 y 5. En este caso, los frentes de Pareto son menores que los obtenidos en una ejecución

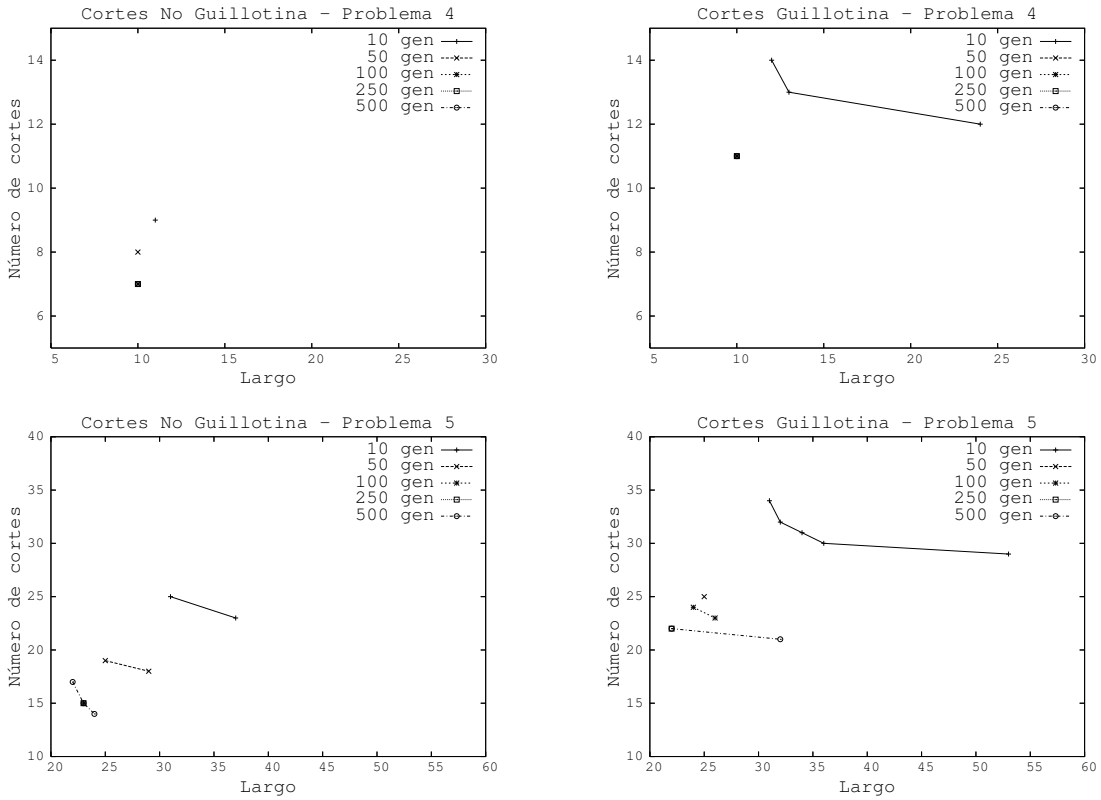

Figura 4.19: Evolución de los frentes de Pareto para la mejor ejecución

media. Las soluciones de los frentes medios se mejoran, aunque la diferencia no es demasiado notable. Como ejemplo de disposición sobre el material, la Figura 4.20 muestra la distribución de las piezas para las dos soluciones obtenidas en el mejor frente de Pareto para el problema 5 cuando se aplican cortes de guillotina.

Para validar la calidad de las aproximaciones que se proponen, los resultados se van a comparar con los presentados en [171], que es la única aproximación que se conoce multi-objetivo y utilizando construcciones guillotinables para el 2DSPP. La representación del cromosoma y los operadores genéticos aplicados en el trabajo referenciado son los mismos que aquí se han usado. La principal diferencia entre las dos propuestas es el algoritmo aplicado. Aquí se han usado diferentes MOEAs proporcionados por METCO, y se ha seleccionado el NSGA-II, mientras que en [171] la estrategia de optimización empleada es una modificación del NSGA-II, aunque no se dan más detalles sobre el algoritmo o los parámetros usados en ese trabajo.

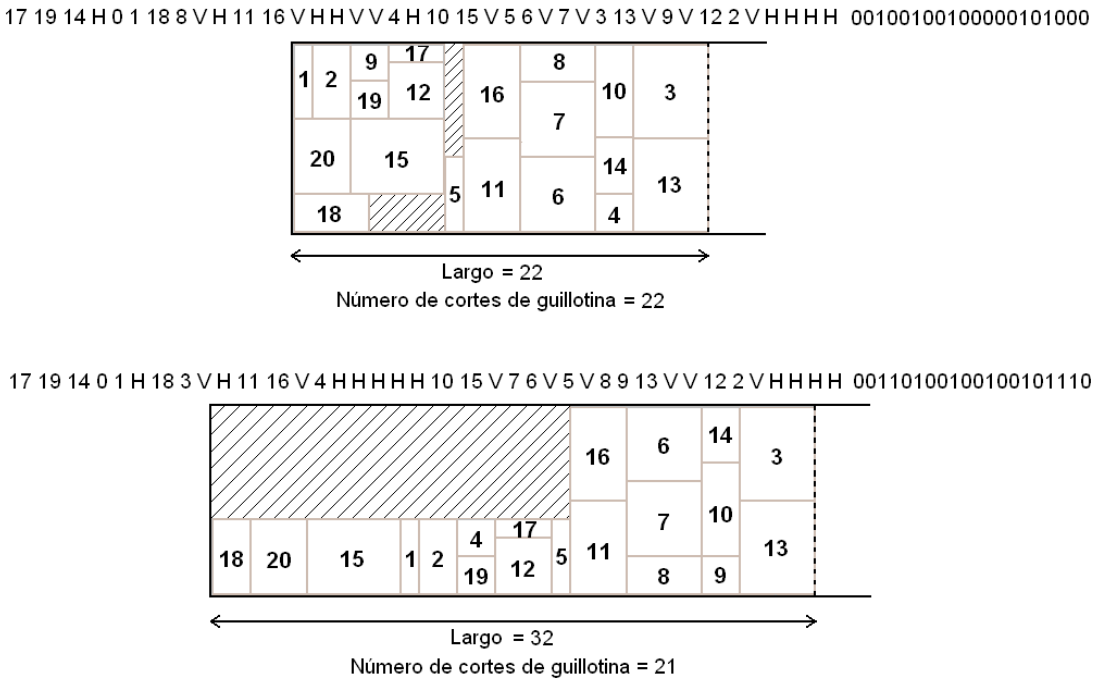

Figura 4.20: Distribución guillotinable del mejor frente de Pareto del problema 5

	Cortes No-Guillotina				**Cortes Guillotina**			
	Problema 4		*Problema 5*		*Problema 4*		*Problema 5*	
	Largo	Cortes	Largo	Cortes	Largo	Cortes	Largo	Cortes
Tiwari y Chakraborti [171]	12	10	26	28	12	13	26	34
	14	9	27	25	14	12	27	30
			28	23			28	29
			29	22			29	28
			30	21			38	27
							46	26
							63	25
NSGA-II	10	7	22	17	10	11	22	22
			23	15			32	21
			24	14				

Tabla 4.9: Soluciones de las aproximaciones multi-objetivo para el 2DSPP

La Tabla 4.9 permite comparar ambas aproximaciones guillotinables para el 2DSPP. En la tabla, se muestran las mejores soluciones para ambas aproximaciones ([171] y la que se propone aquí). Los resultados se especifican para ambos procesos de cortes, guillotina y no-guillotina. Las soluciones obtenidas se representan como un largo total y el número de cortes asociados. La mejor solución en el artículo de referencia se obtiene después de 100000 generaciones, mientras que las mejores soluciones obtenidas aquí sólo requieren de 500 generaciones. Aunque la aproximación previa proporciona frentes de Pareto más grandes, las soluciones obtenidas aquí son claramente superiores, demostrando el mejor comportamiento de esta propuesta.

La aplicación de MOEAs que generan soluciones para el 2DSPP teniendo en cuenta dos criterios diferentes de optimización tiene mayor ventaja para el cliente potencial: tales aproximaciones proporcionan un conjunto de soluciones que ofrecen un rango de compensación o compromiso entre los dos objetivos, y así los clientes pueden elegir de acuerdo a sus necesidades, por ejemplo, coste asociado con la materia prima o incluso tiempos impuestos por el proceso de producción. Sin embargo, tratar con más de un objetivo de optimización no necesariamente implica una reducción de la calidad de la solución a expensas de la posibilidad de optimizar múltiples objetivos. Con esta primera aproximación multi-objetivo para el 2DSPP se obtienen soluciones con niveles de desperdicio similares a las aproximaciones previas que sólo trataban de optimizar el largo total. De hecho, la Tabla 4.10 demuestra la validez de las aproximaciones multi-objetivo para resolver el 2DSPP. Dicha tabla muestra, para algunas de las instancias del 2DSPP disponibles en la literatura [179], el mejor largo obtenido por tres heurísticas de colocación (considerando distintas ordenaciones de piezas) y el largo medio para un algoritmo genético, ambos propuestos en [143]. Todas esas aproximaciones o enfoques tratan la formulación mono-objetivo del 2DSPP.

En [143], los autores mencionan que el tiempo de ejecución para la heurísticas NFDH, FFDH y SPLIT fue de menos de un segundo - para el conjunto de datos aquí analizados - en un ordenador Pentium III. Para las tres heurísticas de colocación, se muestra la mejor solución conseguida considerando tres diferentes órdenes iniciales de piezas demandadas. El GA se ejecuta hasta que no se haya mejorado la mejor solución para 100 generaciones. Cuando era posible, es decir, cuando el tiempo de ejecución era viable, se llevaban a cabo cinco ejecuciones replicadas para el GA y se hacía un promedio de las soluciones. Los tiempos de ejecución para el GA, usando un tamaño de población de 1000, fueron de solo unos pocos minutos para los problemas más pequeños de hasta 100 rectángulos. No obstante, son impredecibles debido a la condición de parada, por lo que puede tomar varias horas o incluso días para los problemas de mayor tamaño. Nótese que las ejecuciones del GA para los problemas de 500 rectángulos y para Nice.200 no son posibles debido a su

Problema	NFDH	FFDH	SPLIT	GA	NSGA-II
Nice.25	133	118	138	108	106
Nice.50	120	119	134	108	109
Nice.100	112	111	137	111	111
Nice.200	110	108	138	-	113
Nice.500	108	107	139	-	126
Path.25	132	120	136	109	101
Path.50	143	131	154	108	103
Path.100	120	109	137	112	108
Path.200	128	116	138	123	121
Path.500	112	105	141	-	147

Tabla 4.10: Comparación de aproximaciones mono y multi-objetivo

largo tiempo de ejecución. La última columna de la tabla representa las soluciones obtenidas para la aproximación de codificación directa. La aproximación fue ejecutada usando los parámetros presentados en la Tabla 4.8. Para problemas pequeños el criterio de parada fue fijado a 10000000 evaluaciones mientras que para instancias grandes se fijó a 25000000 evaluaciones. Los tiempos de ejecución para las instancias más grandes estaban en el rango de dos minutos a dos horas. Para cada problema, se han llevado a cabo 30 ejecuciones. Para cada ejecución se ha considerado el mejor valor para el objetivo "largo total". Luego, con estos valores se ha calculado la media de los mejores valores de las 30 ejecuciones.

Para instancias pequeñas, la aproximación multi-objetivo proporciona mejores soluciones que otras estrategias mono-objetivo. Sólo para las instancias mayores nuestra aproximación multi-objetivo no es capaz de proporcionar soluciones mejores o similares a las conseguidas por las técnicas mono-objetivo. En general, se puede afirmar que, cuando el tamaño de las instancias crece, las aproximaciones específicas mono-objetivo tienden a comportarse mejor, gracias a su simplicidad y a su orientación hacia un solo objetivo. En el caso de la aproximación multi-objetivo, se aplica un algoritmo evolutivo a individuos que, en este caso, están codificados por una representación directa de los patrones de corte. Tal codificación directa permite representar cualquier solución posible, por lo que se trata de una aproximación que es capaz de usar el espacio completo de búsqueda de soluciones. El tamaño de las codificaciones, $2 * n - 1 + n$, es proporcional al número de piezas disponibles n. Para problemas pequeños - con un número reducido de piezas - el espacio de búsqueda de soluciones es mucho más reducido y, por lo tanto, el MOEA es capaz de realizar una exploración exhaustiva del espacio, obteniendo soluciones de alta calidad que en muchos casos mejoran a las obtenidas con las aproximaciones mono-objetivo. Para problemas grandes, donde hay una gran cantidad de piezas para distribuir, el

espacio de búsqueda de soluciones es demasiado grande y, por lo tanto, las aproximaciones mono-objetivo que tratan de optimizar el largo total obtienen mejores resultados. Por esta razón, a continuación se ha tratado de paralelizar la aproximación intentando que se mejore la calidad de las soluciones y sean comparables a las proporcionadas con las aproximaciones mono-objetivo.

Paralelización usando el esquema de codificación directo

Tal y como ya se ha visto, los resultados secuenciales parecen prometedores, por lo que cabría esperar mayores mejoras si se hace uso de las aproximaciones paralelas basadas en islas que aporta el entorno METCO. Por ello, como propuesta inicial, se han probado varios modelos homogéneos basados en islas con el problema 5, ya que es más complejo que el problema 4. En estos modelos homogéneos todas las islas (o procesadores) ejecutan idénticos MOEAs o parámetros. En este caso, para cada uno de los dos algoritmos que mejor rendimiento han mostrado en el estudio inicial secuencial (Figura 4.21), el NSGA-II y el SPEA2, se ha definido un esquema homogéneo. Cada uno de los esquemas paralelos se ha ejecutado con 2 y 4 esclavos. La Figura 4.22 representa la evolución promedia del hipervolumen conforme el número de evaluaciones avanza durante la ejecución. Los resultados muestran que cuando se introduce paralelismo, el algoritmo *Strength Pareto Evolutionary Algorithm 2*, SPEA2 se comporta mejor que el NSGA-II. Esta diferencia se acentúa cuando se usa un mayor número de islas esclavas en el modelo. Por otro lado, también se puede apreciar que el número de evaluaciones requeridas para converger a un cierto nivel de calidad aumenta según se incluyen más islas en el modelo. Esto se debe a que el número total de evaluaciones que se realizan en el modelo secuencial se reparten entre el número de islas utilizadas en el modelo paralelo, de manera que cuantas más islas se utilicen, menos evaluaciones de la población hará cada isla, con lo cual cada isla podrá avanzar menos en la exploración. Sin embargo, esta situación se compensa gracias a la aceleración obtenida debido al paralelismo incorporado al esquema. Es decir, si se analiza el tiempo que necesita cada isla para alcanzar un cierto nivel de calidad entonces se obtiene una aceleración casi lineal (Figura 4.23).

Se han probado además otros dos modelos basados en islas: un modelo heterogéneo estándar (etiquetado como *Heterogéneo*) y un modelo heterogéneo cooperativo y auto-adaptativo. (etiquetado como *Heterogéneo Adaptativo*). En los modelos de islas heterogéneos las diferentes islas ejecutan diferentes MOEAs, mientras que los modelos heterogéneos auto-adaptativos evalúan los algoritmos en función de sus resultados y de esta manera, tratan de asignar más recursos computacionales a los algoritmos más prometedores. La idea de este modelo es evitar tener que hacer pruebas con muchos y distintos algoritmos y/o parámetros, para, posteriormente, ser capaces de elegir el mejor de ellos. Los modelos auto-adaptativos o equipos de

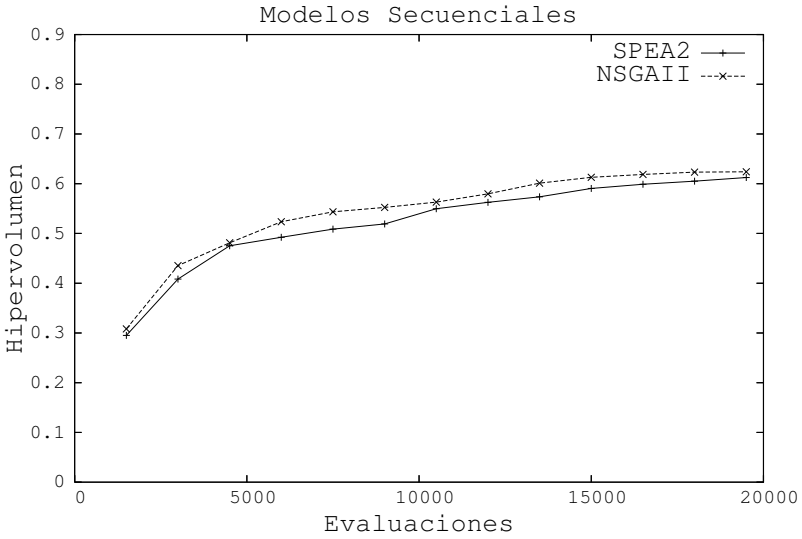

Figura 4.21: Evolución del hipervolumen para la implementación secuencial

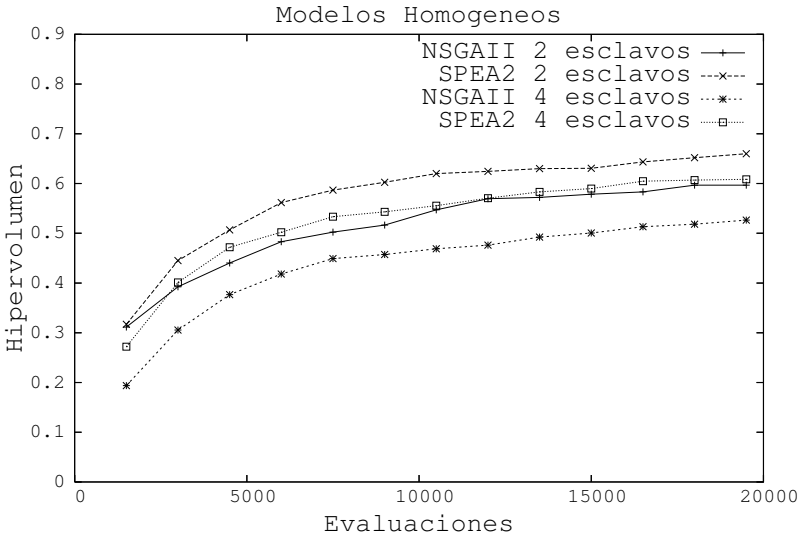

Figura 4.22: Evolución del hipervolumen para el modelo homogéneo

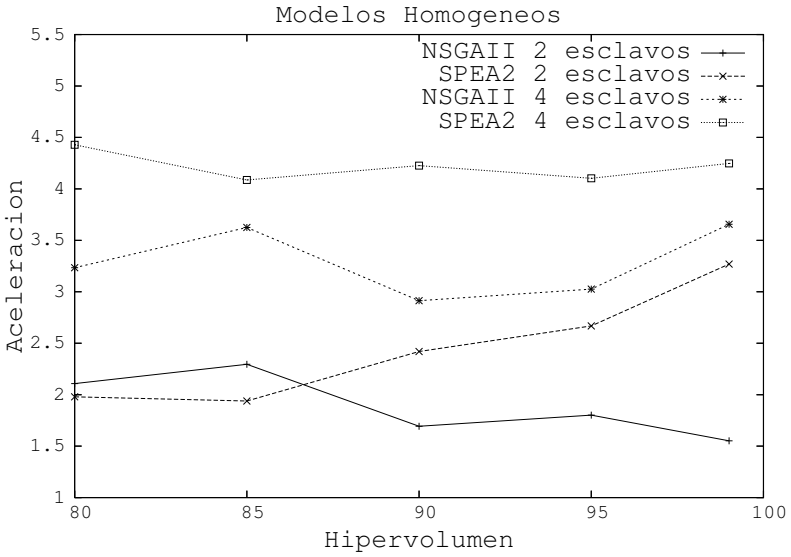

Figura 4.23: Aceleración para el modelo homogéneo

algoritmos permiten obtener, de forma eficiente y sencilla, soluciones de buena calidad, a costa de reducir (pero manteniendo aceptable) la calidad de la solución si se compara con el mejor modelo basado en islas. Como se muestra en la Figura 4.24, las ejecuciones heterogéneas estándar y las heterogéneas auto-adaptativas aportan soluciones mejores que el peor esquema homogéneo, pero peores que el mejor. No obstante, como ya se ha indicado, es difícil conocer a priori cuál es el algoritmo más apropiado para resolver un problema dado. Esto induciría al usuario a probar varios algoritmos antes de tomar la decisión final. En cambio, los modelos heterogéneos y los heterogéneos auto-adaptativos alcanzan soluciones cercanas a las mejores, evitando así que el usuario tenga que probar individualmente una variedad de MOEAs para, posteriormente, seleccionar el que mejor se ajuste a la resolución del problema.

Finalmente, la Figura 4.25 presenta el hipervolumen alcanzado por las implementaciones secuenciales y homogéneas. Considerando que los resultados heterogéneos y heterogéneos auto-adaptativos se encuentran próximos a los homogéneos, entonces, se puede concluir de la figura que las implementaciones paralelas proporcionadas por METCO mantienen la calidad de las soluciones secuenciales, llegando a mejorarla en ocasiones ligeramente, al mismo tiempo que logran obtener una mejora en el rendimiento.

De cualquier modo, con los modelos paralelos no se ha podido mejorar la calidad de las soluciones obtenidas por el modelo secuencial. Para ello probablemente se

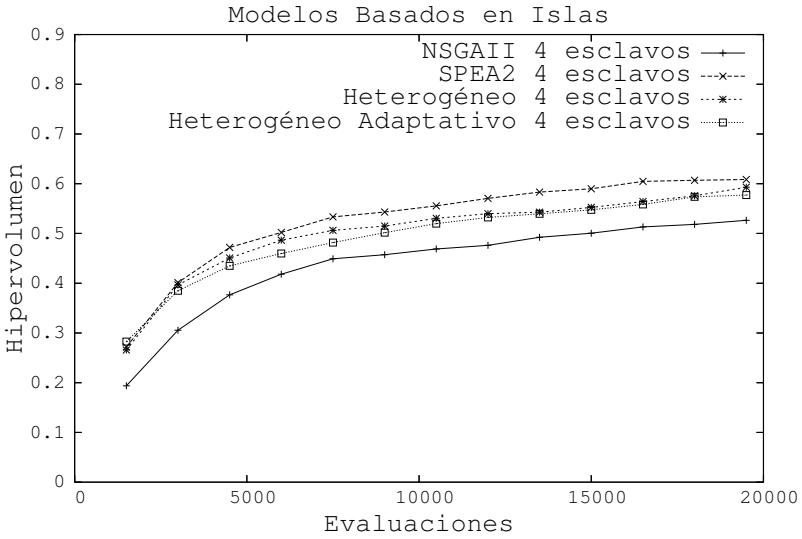

Figura 4.24: Evolución del hipervolumen para todos los modelos basados en islas

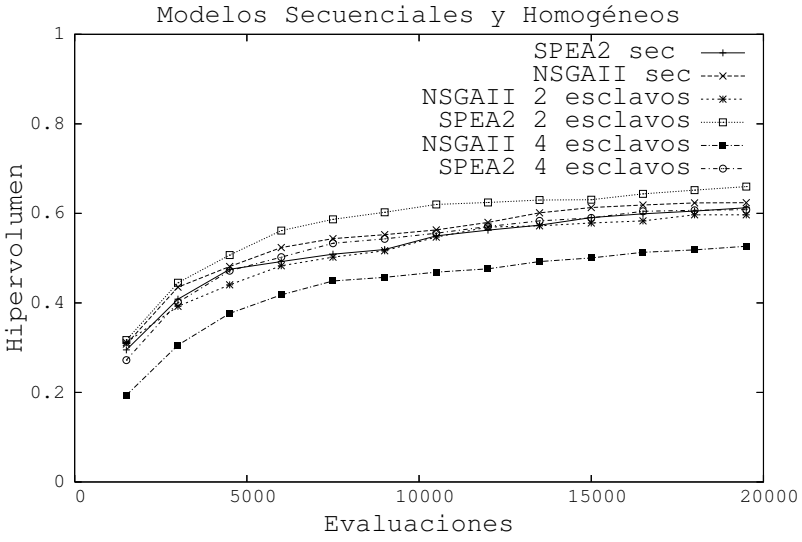

Figura 4.25: Evolución del hipervolumen para el modelo secuencial y homogéneo

tendría que aumentar el número de evaluaciones. Por lo tanto no se han obtenido soluciones mejores que las obtenidas por las técnicas mono-objetivo para instancias grandes del problema. Por esta razón, el principal objetivo de la siguiente sección está centrado en el diseño de una codificación que sea capaz de acotar adecuadamente un espacio de búsqueda tan grande, de tal modo que se obtengan soluciones con calidad comparable a las proporcionadas con las aproximaciones mono-objetivo.

4.5.3.2. Estudio de los esquemas de codificación hiperheurística

Tal y como ya se ha comentado, se han diseñado codificaciones basadas en hiperheurística que combinan diferentes heurísticas de bajo nivel para mejorar las soluciones obtenidas cuando los métodos heurísticos se aplican individualmente. La Tabla 4.11 muestra los valores de los dos objetivos de optimización cuando las heurísticas mono-objetivo se aplican individualmente y cuando se aplican las cinco propuestas multi-objetivo presentadas. Para cada codificación multi-objetivo, se han realizado 30 ejecuciones de 30 minutos cada una, usando los parámetros de configuración de la Tabla 4.8. Para cada ejecución, se han seleccionado dos puntos: el de menor largo y el de menor número de cortes. La media de los valores de esos puntos para las 30 ejecuciones se muestra, respectivamente, en la primera y segunda fila de la codificación. Nótese que para las aproximaciones mono-objetivo sólo hay una solución (que minimiza el largo), aunque de esta solución (disposición de piezas) también conocemos el valor para el objetivo "número de cortes". Los resultados de la tabla reflejan que la implementación mediante codificación directa consigue valores de corte considerablemente menores que los obtenidos por las heurísticas, pero no es capaz de llegar a sus valores de largo. Sin embargo, la aproximación mediante hiperheurística **(h, p)** es capaz de mejorar ambos objetivos cuando se compara con

Solución	**nice200_40**		**path200_40**		**nice500_5**		**path500_5**	
Aproximación	largo	cortes	largo	cortes	largo	cortes	largo	cortes
NFDH	110.90	371.00	127.30	350.00	108.80	943.00	119.61	896.00
FFDH	107.01	374.00	114.32	364.00	106.17	946.00	112.84	904.00
BFDH	107.01	374.00	114.25	392.00	106.17	946.00	112.84	904.00
BFDH*	106.96	373.00	114.10	354.00	106.08	946.00	111.17	905.00
Esquema	123.51	337.80	128.89	331.86	148.84	908.03	187.19	886.70
Cod. Directa	154.67	**306.80**	174.70	**297.80**	186.46	861.40	235.51	849.93
(h, p)	**104.83**	373.60	103.94	353.13	**103.36**	944.36	**103.65**	904.63
Cod. Hiperheurística	107.92	368.93	107.68	334.00	104.76	938.06	106.06	883.13
(h, p) + or + ot	120.54	397.50	126.88	394.76	121.08	996.46	124.79	994.50
Cod. Hiperheurística	155.92	387.60	227.28	384.13	155.18	986.00	239.53	981.46
(h, p, o) + ot	105.63	375.06	104.46	381.80	104.47	925.33	104.84	936.66
Cod. Hiperheurística	110.92	350.66	126.52	346.16	107.14	887.40	130.11	881.16
(h, p, o, r)	106.53	356.56	**103.51**	348.66	104.61	854.73	104.63	881.23
Cod. Hiperheurística	114.33	336.30	115.81	318.90	109.07	**816.55**	135.11	**830.96**

Tabla 4.11: Heurísticas mono-objetivo de bajo nivel y aproximaciones multi-objetivo

las heurísticas mono-objetivo de colocación de piezas. La segunda hiperheurística **(h, p) + or + ot** no es capaz de mejorar ningún valor de los objetivos obtenidos con las heurísticas de bajo nivel, es decir, no proporciona resultados de buena calidad. Finalmente, la tercera y la cuarta propuesta, **(h, p, o) + ot** y **(h, p, o, r)**, mejoran ambos objetivos con respecto a las heurísticas mono-objetivo.

Por lo tanto, los esquemas hiperheurísticos **(h, p)**, **(h, p, o) + ot** y **(h, p, o, r)** diseñados, son capaces de mejorar los resultados dados por las heurísticas de bajo nivel, es decir, mediante el uso de los principios de las hiperheurísticas y las aproximaciones multi-objetivo, se ha conseguido mejorar las soluciones obtenidas por algoritmos hechos a medida para el 2DSPP mono-objetivo. Pero si comparamos las diferentes codificaciones hiperheurísticas aplicadas por el MOEA, nos damos cuenta de que en general, la aproximación **(h, p)** proporciona los valores más bajos de largo del material usado, y la propuesta **(h, p, o, r)** da los valores más bajos de número de cortes. Sin embargo, se están analizando sólo las soluciones con los mínimos largos y cortes. Ahora es necesario comparar el conjunto completo de soluciones obtenidas mediante los esquemas de codificación.

Si se quieren comparar dos aproximaciones multi-objetivo diferentes [111] se puede usar alguna métrica multi-objetivo como el hipervolumen [193] o el indicador ϵ [194]. Pero en este caso, interesa identificar claramente las áreas del espacio de búsqueda que se están siendo exploradas por cada una de las implementaciones. Por ello, se han representado las superficies de alcance (véase la Sección 2.8 del Capítulo 2), como una alternativa para resumir los resultados obtenidos en las 30 ejecuciones.

La Figura 4.26 muestra las superficies de cubrimiento 1, 15 y 30, para los cuatro problemas de mayor dimensión seleccionados. En cada gráfica se pueden ver las superficies de cubrimiento para el esquema de codificación directo y el **(h, p)** esquema de codificación basado en hiperheurística. Como se puede observar, el primer esquema de codificación basado en hiperheurística cubre la zona superior izquierda del espacio de soluciones (valores altos de número de cortes - valores bajos de largo del material), es decir, se centra más en la optimización del largo. Mientras, la codificación directa se centra en la zona media/baja derecha (valores bajos de número de cortes - valores medios/altos de largo del material), es decir, se centra más en optimizar el número de cortes. Sólo hay una pequeña región en tres problemas de los mostrados, donde los puntos más bajos de la aproximación hiperheurística dominan completamente a los puntos más altos de la codificación directa. De cualquier modo, en general los esquemas de codificación están cubriendo regiones diferentes del espacio de soluciones. Además, el hueco entre las dos regiones cubiertas tiende a crecer para los problemas de mayor tamaño. Definitivamente, algunas soluciones parciales no se pueden obtener ni con la codificación directa, ya que el espacio de búsqueda es demasiado amplio para llegar a alcanzarlas, ni usando la primera codi-

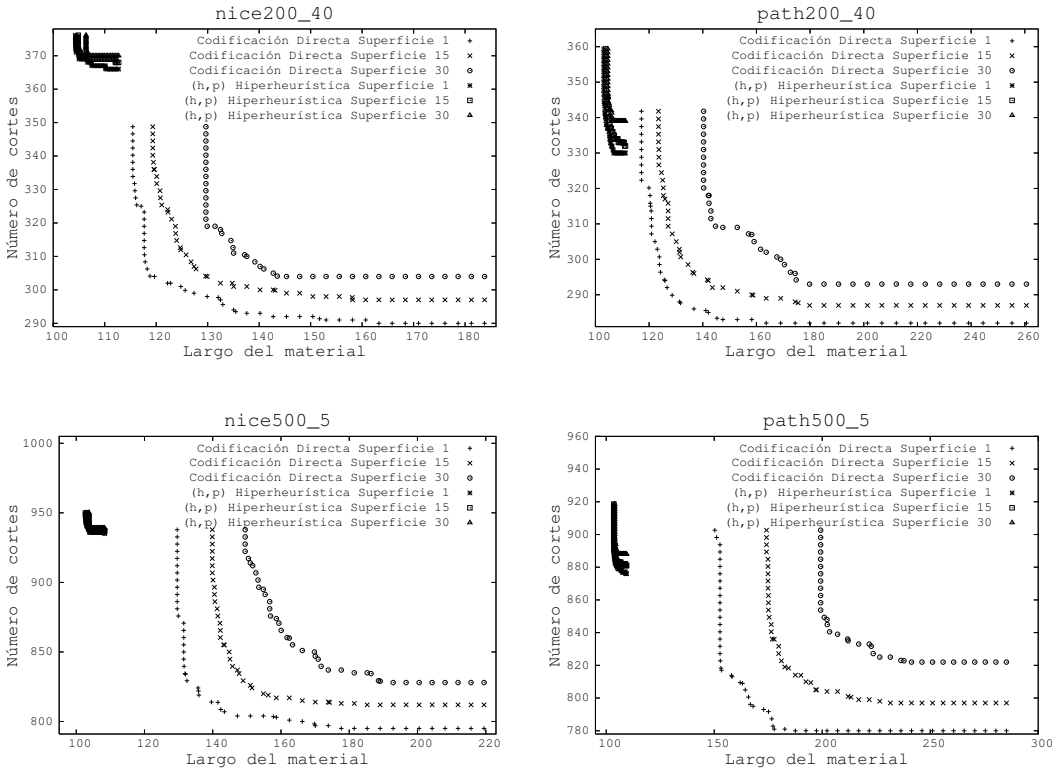

Figura 4.26: Superficies de cubrimiento para el esquema directo y el esquema hiperheurístico ***(h, p)***

ficación hiperheurística, ya que las heurísticas de bajo nivel fijan la orientación y el orden de los objetos, evitando así la generación de algunas soluciones factibles.

Si se especifica el orden y la orientación de las piezas fuera de las heurísticas de bajo nivel para expandir el espacio de búsqueda, se obtiene el **(h, p) + or + ot** esquema de codificación hiperheurística. La Figura 4.27 muestra las superficies de cubrimiento 1, 15 y 30, para los mismos cuatro problemas de gran tamaño. En cada gráfica se pueden ver las superficies de cubrimiento para el esquema de codificación directa y el **(h, p) + or + ot** esquema de codificación basado en hiperheurística. Como se puede observar, para algunos problemas los resultados obtenidos usando la codificación directa incluso dominan a los resultados obtenidos con la segunda codificación hiperheurística. Al igual que ocurría con el esquema de codificación directa, parece que el espacio de soluciones a explorar es demasiado grande para esta codificación hiperheurística. Así pues, para reducir el espacio de búsqueda de la segunda aproximación hiperheurística, una opción sería introducir los criterios de ordenación, en lugar de permitir cualquier ordenación de las piezas. Otra posibilidad

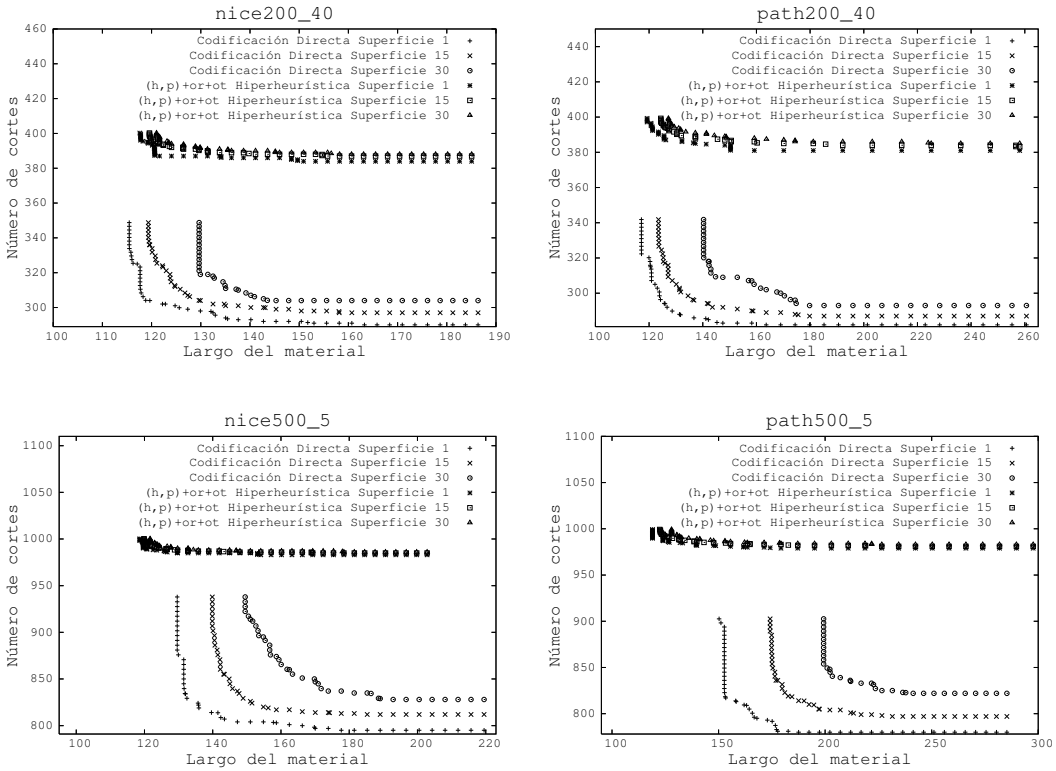

Figura 4.27: Superficies de cubrimiento para el esquema directo y el esquema hiperheurístico ***(h, p) + or + ot***

podría ser introducir los criterios de rotación también, en lugar de permitir cualquier orientación de las piezas. En esto se basan las ideas del tercer y cuarto esquema de codificación hiperheurística, **(h, p, o) + ot** y **(h, p, o, r)**.

La Figura 4.28 y la Figura 4.29 muestran las superficies de cubrimiento 1, 15 y 30 para el esquema de codificación directo y los esquemas de codificación basado en hiperheurística **(h, p, o) + ot** y **(h, p, o, r)**, en cuatro problemas de los más complejos. En ambos casos se consigue reducir el espacio de búsqueda y además las soluciones están posicionadas en el área deseada, es decir, a la izquierda de las soluciones obtenidas por el esquema de codificación directo. En ese área el largo del material es menor, objetivo que con la aproximación multi-objetivo no llegaba a la calidad de las soluciones proporcionadas por las aproximaciones mono-objetivo. El cuarto esquema de codificación hiperheurística tiene el mejor comportamiento, ya que proporciona números de cortes menores asociados a los largos, es decir, muestra un buen compromiso entre los diferentes objetivos. Por esta razón, se ha escogido la cuarta aproximación hiperheurística para realizar estudios futuros.

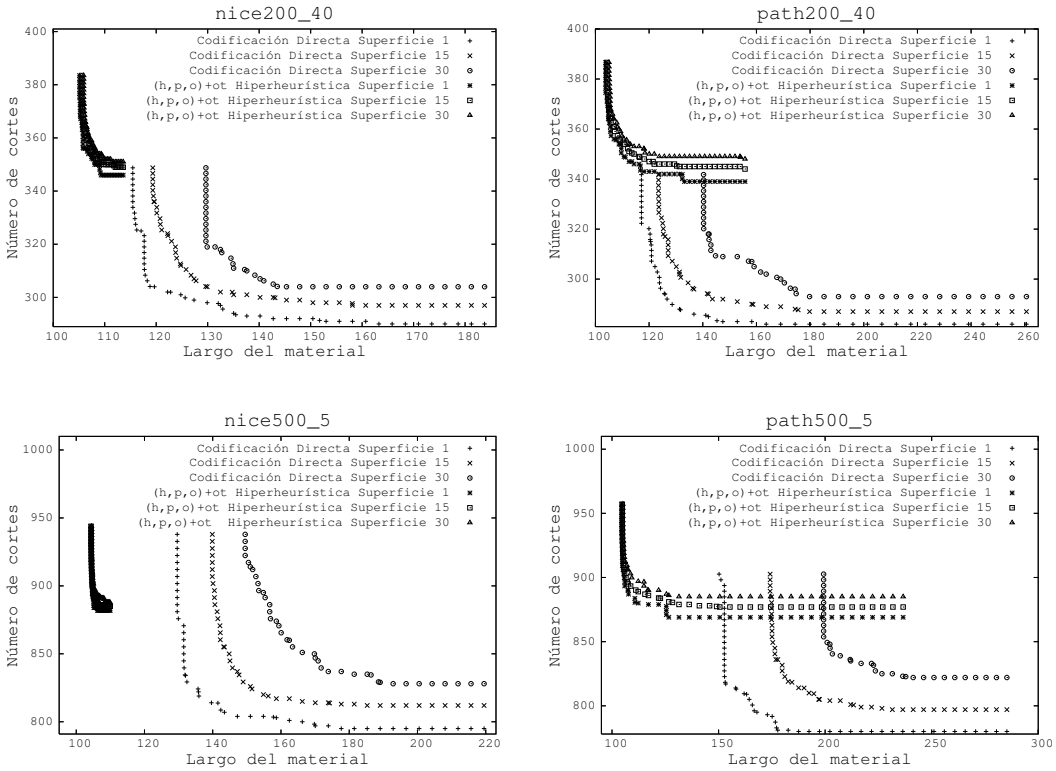

Figura 4.28: Superficies de cubrimiento para el esquema directo y el esquema hiperheurístico ***(h, p, o) + ot***

Finalmente, la Figura 4.30 muestra las superficies de cubrimiento 15 para los cinco esquemas de codificación en dos problemas diferentes. Como se puede observar, la primera aproximación hiperheurística cubre una pequeña área superior izquierda. Las soluciones del segundo esquema de codificación están en una área completamente dominada. La tercera codificación hiperheurística proporciona soluciones que mejoran ligeramente el objetivo de número de cortes (cuando se compara con el primer esquema de codificación hiperheurístico). Sin embargo, el cuarto esquema de codificación hiperheurístico es capaz de cubrir una amplia área del espacio de soluciones, incluso mejorando el número de cortes. Aunque el esquema de codificación directa proporciona el número de cortes más bajo, no es capaz de llegar a valores bajos del largo del material, tal y como se ha mencionado anteriormente. Así pues, para las instancias con mayor número de piezas la codificación directa da soluciones que no minimizan adecuadamente el largo de la materia prima, es decir, cuanto más complejo sea el problema, peor solución dará, ya que el espacio de búsqueda es mucho mayor. No obstante, la mayoría de las codificaciones basadas en hiperheurística

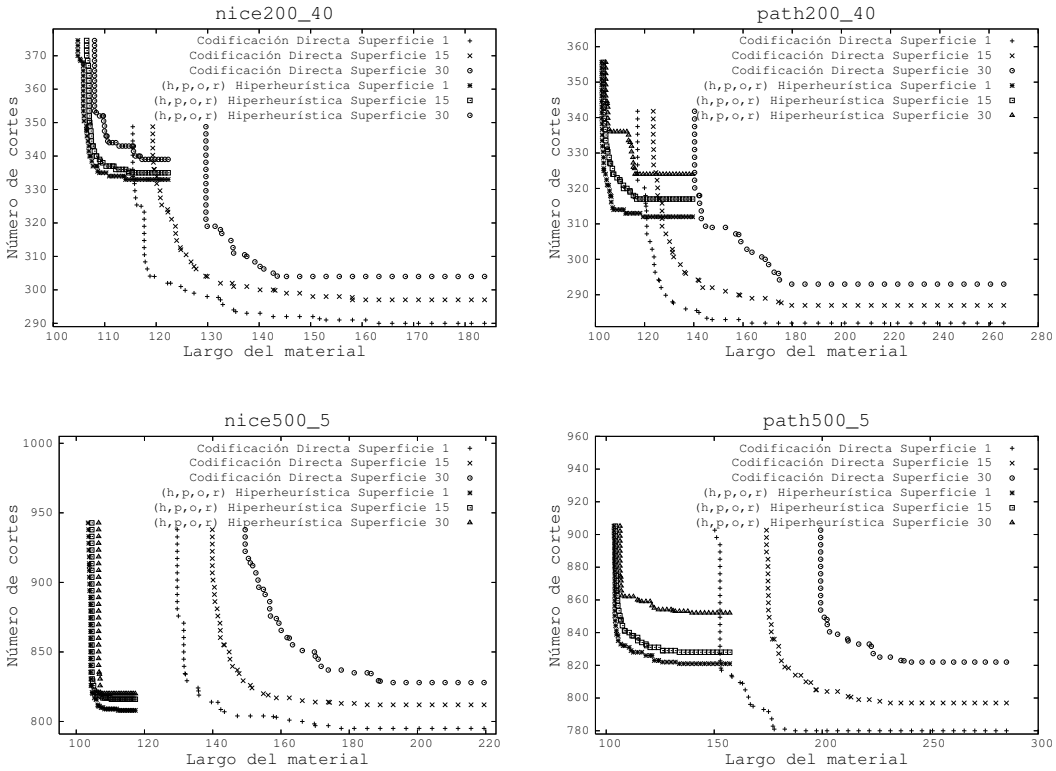

Figura 4.29: Superficies de cubrimiento para el esquema directo y el esquema hiperheurístico ***(h, p, o, r)***

se comportan adecuadamente para estos problemas, siendo el cuarto esquema, **(h, p, o, r)**, el que muestra el comportamiento más prometedor. Por lo tanto, los esquemas de codificación basados en hiperheurística son capaces de obtener soluciones de alta calidad, incluso cuando tratan con instancias del problema con muchas piezas.

De este modo, se cuenta con un esquema de codificación hiperheurístico, **(h, p, o, r)**, que mejora los resultados obtenidos por las heurísticas de bajo nivel, tal y como se puede apreciar en la Figura 4.31, donde los puntos relativos a la solución de cada aproximación mono-objetivo son completamente dominados por el esquema hiperheurístico. Por otra parte, también se cuenta con un esquema de codificación directa que no mejora el objetivo del largo que proporcionan las aproximaciones mono-objetivo, pero proporciona los valores más bajos de número de cortes. Sin embargo, la ventaja de tener diferentes codificaciones eficientes cubriendo distintas áreas del espacio de soluciones es el hecho de que la persona que toma las decisiones puede elegir el tipo de codificación dependiendo del área del espacio de soluciones más interesante para el problema en cuestión.

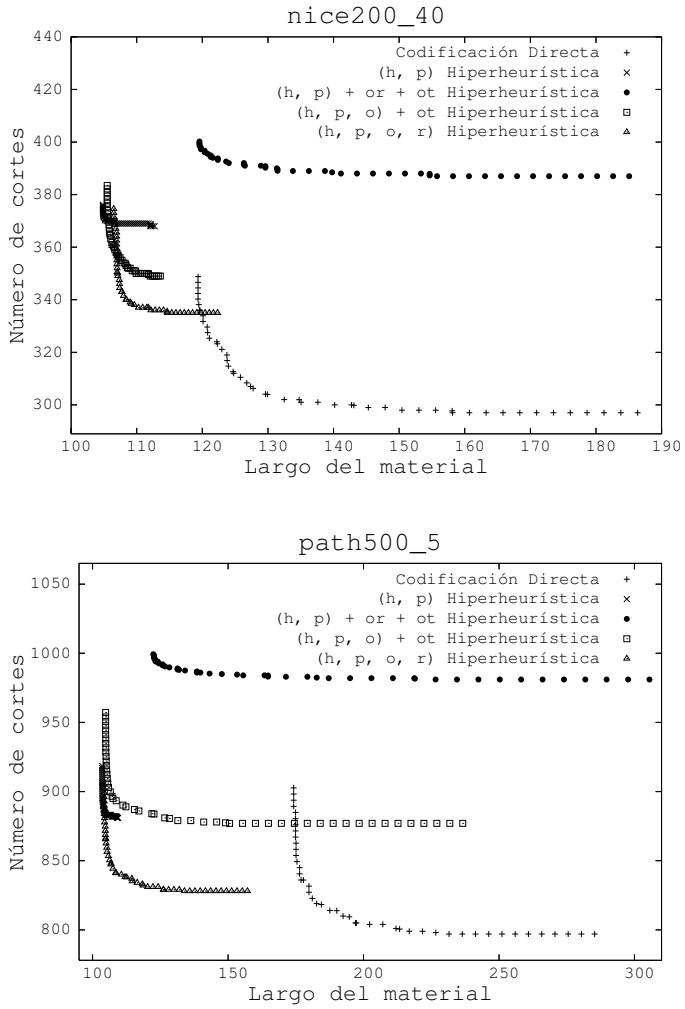

Figura 4.30: Superficies de cubrimiento para todos los esquemas de codificación

4.5.4. Conclusiones

Se ha planteado la resolución del 2DSPP multi-objetivo, considerando los dos objetivos siguientes: minimizar el largo total del material a usar y minimizar el número total de cortes necesarios para obtener el todas las piezas demandadas. Esta formulación es muy adecuada para los problemas reales que aparecen en las industrias de producción y manufactura. La consideración de ambos objetivos puede implicar importantes ahorros para las fábricas, puesto que si además de tratar de minimizar el largo del material que se usa, se minimiza también el número de cortes necesarios, la maquinaria sufrirá menos deterioro y, por lo tanto se podrán reducir los

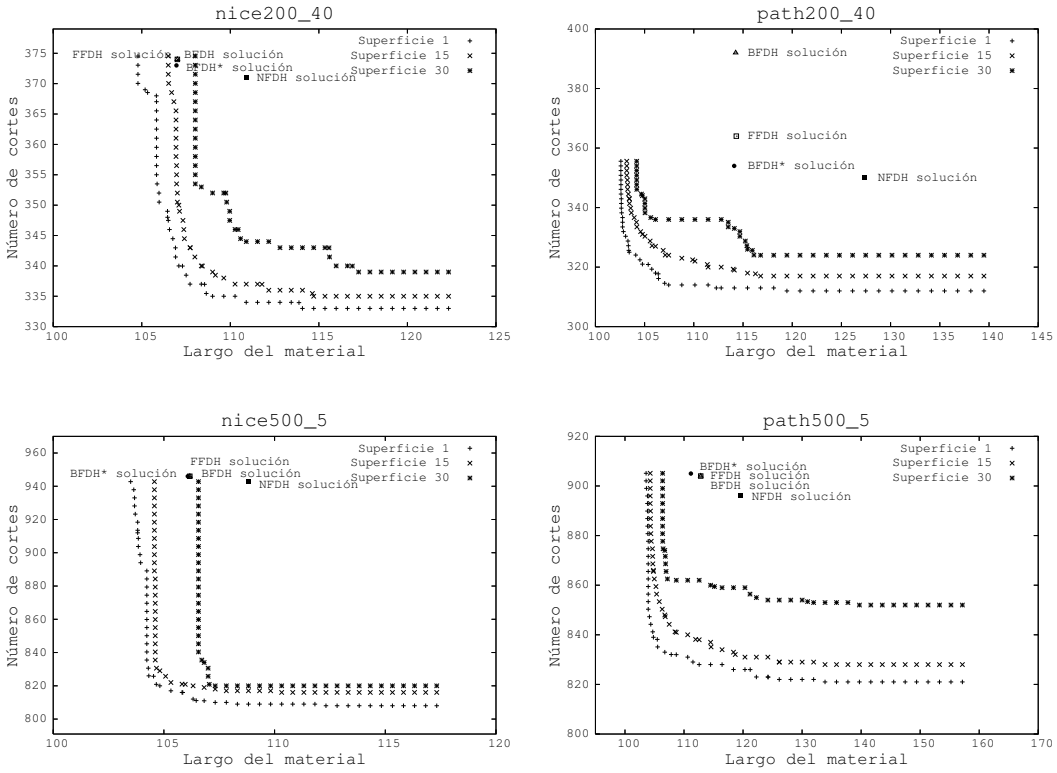

Figura 4.31: Superficies de cubrimiento esquema hiperheurístico ***(h, p, o, r)*** y heurísticas de bajo nivel

costes de mantenimiento de la maquinaria. Además, cuando la materia prima es suficientemente barata, una solución que minimice el número de cortes y, por tanto, maximice la productividad de la maquinaria, puede ser más conveniente. Sin embargo, si el material es caro, puede ser mejor opción una solución que minimice el largo del material requerido. Usando algoritmos que traten ambos objetivos simultáneamente, sólo se requiere una ejecución, y después la persona encargada de tomar las decisiones puede seleccionar la solución más adecuada, dependiendo de las demandas específicas en cada momento. Cuando se aplican aproximaciones mono-objetivo, esas demandas se deben conocer a priori para combinar los diferentes objetivos de la mejor manera. Si las demandas cambian, se necesitarán nuevas ejecuciones. Esta disponibilidad de una diversidad de soluciones es el beneficio más importante de las propuestas multi-objetivo que se han hecho.

A pesar de que existe abundante bibliografía para la formulación mono-objetivo del problema, los trabajos sobre aproximaciones multi-objetivo son escasos. Sin embargo, dada la complejidad inherente a la mayoría de los MOPs, las técnicas di-

señadas específicamente para manejar múltiples objetivos son las más adecuadas para resolverlos. Ya que los algoritmos evolutivos se han usado con éxito en muchos MOPs del mundo real y en muchas aproximaciones mono-objetivo del problema, se han elegido para abordar el 2DSPP multi-objetivo, usándose algunos de los MOEAs más extendidos en la literatura. En lugar de desarrollarlos desde cero, se ha hecho uso de METCO y se ha seleccionado el NSGA-II dado que ha presentado un mejor comportamiento para este problema.

Inicialmente, se ha planteado una implementación de los individuos basada en la representación directa de patrones de corte. Los resultados obtenidos mejoran claramente los únicos resultados conocidos para el 2DCSP multi-objetivo y con cortes guillotinables. Como la representación del cromosoma que se ha usado permite ambos procesos de corte, guillotina y no-guillotina, el número de cortes puede variar dependiendo de las características de la maquinaria disponible. El estudio computacional demuestra que siempre que sea posible realizar cortes de tipo no-guillotina, es decir, cuando la maquinaria permita un proceso más sofisticado de corte, el número de cortes requerido para llevar a cabo el trabajo se puede ver reducido significativamente.

Por otra parte, en un intento de mejorar aún más el comportamiento del esquema directo se han obtenido resultados para distintos modelos paralelos basados en islas. Así se ha podido comprobar que el rendimiento de las aproximaciones evolutivas se puede incrementar un poco gracias a algunos de los modelos paralelos basados en islas proporcionados por el entorno METCO.

De cualquier modo, para las instancias pequeñas del problema, la aproximación directa proporciona soluciones comparables a las obtenidas por aproximaciones que se centran en optimizar el largo total. Sin embargo, para las instancias de tamaño mayor, la codificación tiene que tratar con espacios de búsqueda muy grandes, por lo que, ni siquiera mediante paralelizaciones es capaz de producir soluciones competitivas en cuanto a largo total. Con el objetivo de mejorar estos resultados, se han propuesto cuatro nuevos tipos de codificación basados en la combinación de diversas heurísticas de bajo nivel diseñadas para optimizar sólo el objetivo del largo del material. Cada esquema de codificación se centra en un espacio de búsqueda diferente, por lo que los resultados se encuentran en distintas áreas del espacio de soluciones. Los resultados computacionales demuestran que dos de los cuatro esquemas de codificación hiperheurística son capaces de proporcionar soluciones competitivas comparadas con las heurísticas mono-objetivo individuales. Además, una de las hiperheurísticas, **(h, p, o, r)**, da soluciones que dominan completamente a las obtenidas usando las heurísticas mono-objetivo individuales. Esta codificación puede buscar en regiones donde la optimización de ambos objetivos está mejor equilibrada. De este modo, se tiene un esquema de codificación directa que no puede mejorar

el objetivo del largo para instancias del problema con muchas piezas, comparado con las aproximaciones mono-objetivo, pero proporciona valores bajos de número de cortes. En contraposición, se tiene un esquema de codificación hiperheurística que mejora ambos objetivos, comparado con las heurísticas de bajo nivel, pero proporciona números de cortes mayores que los que aporta el esquema de codificación directa. Usando diferentes esquemas de codificación y mediante la aplicación de métodos basados en hiperheurísticas, se ha conseguido cubrir una gran zona del espacio de soluciones, permitiendo que la persona encargada de tomar las decisiones pueda elegir el tipo de codificación dependiendo del área del espacio de soluciones que sea más interesante para el problema dado.

Tal y como se ha demostrado, el diseño de esquemas de codificación para la representación de soluciones en un MOEA es una cuestión muy importante. La definición de espacios de búsqueda grandes dificulta la obtención de soluciones de alta calidad, pero cuando los espacios de búsqueda se acotan, algunas regiones del espacio de soluciones pueden quedar sin explorarse. Por lo tanto, es importante diseñar codificaciones que permitan cubrir regiones del espacio de búsqueda que ofrezcan un mejor equilibrio entre los dos objetivos de optimización.

Por último, es necesario destacar que no se ha podido aplicar el algoritmo exacto para realizar comparativas con el modelo multi-objetivo, ya que las instancias de interés utilizadas con este modelo son demasiado grandes para ser abordadas de forma viable por un método exacto.

Capítulo

5 Container Loading Problem

En este capítulo se estudia el *Container Loading Problem*, CLP, o problema de carga de contenedores. Este problema surge en las industrias reales de transporte y distribución. Mientras que los problemas de corte se centran en el mejor uso posible de los materiales, los problemas de empaquetado lo hacen en la mejor utilización posible del espacio. Un transporte eficiente de la carga tiene un alto impacto económico en la logística, de modo que si se lleva a cabo una carga óptima se reducen los costes en todo el sistema de distribución. Existen muchas variantes del problema dada la diversidad de aplicaciones del mismo. En base a la aplicación particular, el problema puede presentar diferentes restricciones, como la orientación de las piezas, la distribución de la carga, la estabilidad, las prioridades de introducción de las piezas, el límite de peso de los contenedores, el tipo de colocación de las piezas, etc. Por otra parte, se puede considerar la carga de un sólo contenedor o de varios contenedores, así como contenedores de dimensiones fijas o de una dimensión abierta. Concretamente, este capítulo se centra en el estudio de las técnicas de resolución para una de las formulaciones del problema según la cual se intenta cargar un sólo contenedor de dimensiones fijas maximizando el volumen usado. Con esto se reduce el número de contenedores necesarios para hacer la distribución y, por tanto, se reducen los costes en transporte. Debido al espacio de búsqueda tan grande que presenta este problema aplicar métodos exactos es muy inusual, por lo que en este caso los métodos de resolución más comunes que se han aplicado en la literatura a la visión mono-objetivo del mismo son los heurísticos y metaheurísticos. Teniendo en cuenta que generalmente en el ámbito real se presentan otros criterios de optimización para este problema, en este capítulo, a la hora de resolver el problema, se ha considerado directamente una visión multi-objetivo del mismo. Así se han aplicado MOEAs para resolverlo y se han realizado diversas paralelizaciones.

5.1. Definición del problema

El CLP pertenece a un área de investigación activa y tiene numerosas aplicaciones en el mundo real, particularmente en el transporte de contenedores y en las industrias de distribución. El aumento constante del coste del combustible es una gran motivación para los transportistas a la hora de maximizar el espacio utilizado en los contenedores, reduciendo así al mínimo el número global de viajes necesarios [153]. Además, minimizar los espacios vacíos en los contenedores no sólo es un requisito económico, sino también un problema ecológico debido a las consecuencias negativas del aumento del tráfico en el medio ambiente [180].

De entre los problemas de corte y empaquetado, los de carga de contenedores han sido muy estudiados. En general, estos problemas consisten en distribuir un conjunto de piezas rectangulares (cajas) en uno o más objetos rectangulares grandes (contenedores) de manera que se maximice el volumen total de las cajas empaquetadas. Esto es, se trata de obtener un patrón de empaquetado que utilice el espacio del contenedor lo máximo posible. Dependiendo de la definición del conjunto de restricciones asociadas, se tendrá un tipo de problema y una formulación específica diferente. De este modo, algunas aplicaciones reales del CLP tienen en cuenta criterios adicionales de optimización. En ocasiones, para empaquetar todas las cajas en un contenedor se debe cuidar la orientación de las mismas, por ejemplo, cuando el contenido de éstas es frágil. Otras veces, para un correcto transporte, se debe realizar una buena distribución de la carga en el contenedor, o es necesario un orden específico de introducción de piezas para su posterior extracción.

En el problema concreto que aquí se va a tratar se dispone de un contenedor de dimensiones conocidas ($W = ancho, L = profundidad, H = altura$), y un conjunto de N cajas rectangulares. Esas cajas pertenecen a uno de los conjuntos de tipos de cajas $\mathcal{D} = \{T_1 \dots T_m\}$, donde el i-ésimo tipo T_i tiene dimensiones $w_i \leq W$, $l_i \leq L$ y $h_i \leq H$. Asociado a cada tipo T_i existe un volumen v_i, una demanda b_i, y un número de orientaciones o rotaciones permitidas $o_i \in [1, 6]$ (Figura 5.1). Normalmente, como mínimo siempre habrá dos orientaciones posibles para cualquier caja (la 0 y la 1), puesto que no se altera el contenido de la misma, ya que se trata de la rotar la base 90 grados. Además, en las instancias conocidas de este problema [17, 131] no se permiten las orientaciones 4 y 5, donde el frente de la caja se usa como base, si no se permiten la 2 y la 3, donde el lateral de la caja se usa como base. Pero se pueden permitir la 2 y la 3 y no la 4 y la 5. Es decir, se permitirán dos, cuatro o seis orientaciones posibles para cada tipo de caja. Estas restricciones en cuanto a las orientaciones se deben al tipo de contenido de las cajas, ya que pueden albergar, por ejemplo, un líquido, con lo que se presentarían problemas en determinadas posiciones. En definitiva, lo que se pretende es encontrar un empaquetamiento sin superposiciones de las cajas, y que

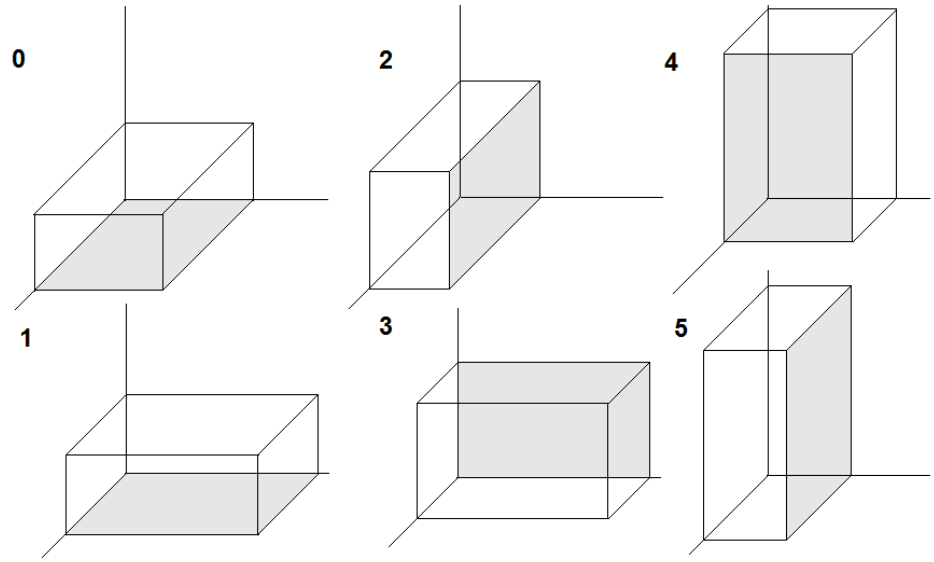

Figura 5.1: Posibles orientaciones para una pieza

utilice x_i cajas de tipo T_i para maximizar el volumen total utilizado del contenedor:

$$\text{máx} \sum_{i=1}^{m} x_i v_i \quad \text{sujeto a } x_i \in [0, b_i] \tag{5.1}$$

El problema se resolverá siguiendo los siguientes supuestos:

- Cada caja se coloca en el suelo del contenedor o encima de otra caja.
- La estabilidad de la distribución de las cajas no se considera, ya que se asume el uso de material para rellenar los huecos entre cajas y evitar así posibles problemas de estabilidad.

Las características principales del CLP aquí tratado se resumen en la Tabla 5.1, de acuerdo con la tipología general que se ha expuesto para los problemas de corte y empaquetado en la Sección 1.2. Teniendo en cuenta este conjunto de propiedades y siguiendo las tipologías más recientes en corte y empaquetado [182] o la clasificación dada en la Sección 1.3, el problema se denotaría como un problema de tres dimensiones de un solo objeto grande de colocación (*3D Rectangular Single Large Object Placement Problem*).

Nombre de la Propiedad	Valor de la Propiedad
Dimensionalidad	Tres dimensiones (3D)
Medición de cantidad	Discreta
Forma de las figuras	Regular y con orientaciones
Variedad de piezas pequeñas	Variedad heterogénea
Variedad de piezas grandes	Un objeto grande
Disponibilidad de piezas	No se establece orden entre las piezas
Restricciones de los patrones	Guillotinables y orientaciones limitadas para algunas piezas
Tipo de asignación	Maximización de la salida
Objetivos	Maximizar el volumen ocupado y el peso del objeto grande

Tabla 5.1: Propiedades del CLP

Entre las restricciones asociadas a este problema, un aspecto bastante común es el límite de peso de los contenedores, ya que normalmente estos no pueden superar un determinado peso para su transporte y se debería aprovechar al máximo sin sobrepasar ese límite. Los vehículos para transportar el cargamento suelen ser alquilados y se pagan de acuerdo al peso total que puede transportar con independencia del volumen total. Por lo tanto, sería conveniente transportar un cargamento con peso total elevado, respetando siempre el límite o tara del vehículo. Por esta razón, en este trabajo se tomará el peso total del contenedor como un segundo objetivo. Así, siendo el peso máximo que puede albergar el contenedor P_{max}, y teniendo cada caja un peso asociado $p_i < P_{max}$, el problema se podrá plantear como un problema de optimización multi-objetivo, intentando optimizar la distribución de las piezas dentro del contenedor de modo que se maximice el volumen al mismo tiempo que el peso total empaquetado sin superar el límite:

$$\text{máx} \sum_{i=1}^{m} x_i v_i \quad y \quad \text{máx} \sum_{i=1}^{m} x_i p_i \quad \text{sujeto a } x_i \in [0, b_i] \quad y \quad \sum_{i=1}^{m} x_i p_i \leq P_{max} \tag{5.2}$$

Esta formulación del problema se denotará como *Multi-Objective Container Loading Problem*, MOCLP.

Tal y como se ha mencionado en los capítulos anteriores, en la mayoría de los problemas de optimización del mundo real, los objetivos implicados suelen estar en conflicto entre ellos, es decir, no hay una sola solución para optimizar simultáneamente todos los objetivos. En lugar de un solo óptimo, existe más bien un conjunto

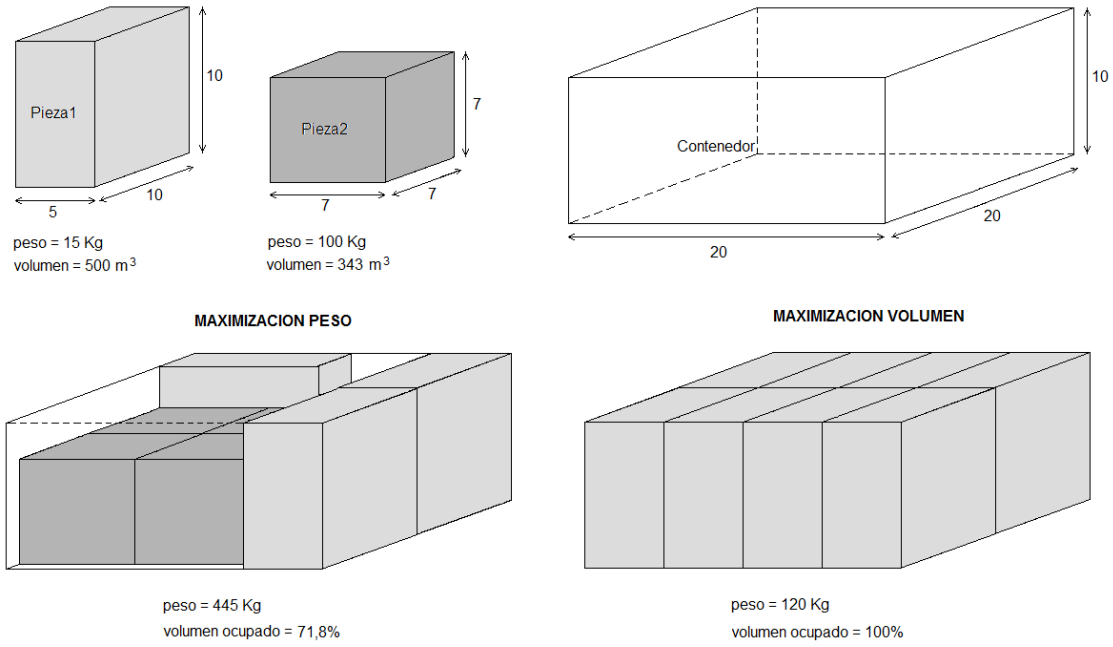

Figura 5.2: Ejemplo de objetivos contradictorios

de soluciones alternativas, de entre las cuales una persona encargada de tomar las decisiones debe seleccionar una solución de compromiso. En el caso del CLP multi-objetivo, se podría pensar a priori que la maximización del volumen total implica una maximización del peso. Sin embargo, muchas veces el tamaño de las piezas o cajas a empaquetar en el contenedor no es proporcional a su peso. Es decir, una caja puede tener un tamaño grande y el contenido de la misma puede ser más ligero que el de una caja de menor tamaño (Figura 5.2). Esto puede ocurrir por ejemplo en una compañía que suministre material de oficina, donde un pisapapeles ocupa muy poco y es más pesado que una papelera, y sin embargo, la papelera es más grande y pesa menos. Otro caso puede darse en el transporte de alimentos, donde una caja de cereales ocupa un volumen superior al de una lata de conservas, pero una lata de conservas puede tener un peso mayor que el de la caja de cereales. No obstante, en otras ocasiones los pesos de los objetos que se van a empaquetar son proporcionales a sus volúmenes, con lo cual si una caja es grande tiene asociado un peso grande y si es pequeña tiene un peso asociado bajo. Por ejemplo, el transporte de detergente y pasta dentífrica. Por lo tanto, una vez más se tienen dos objetivos con, al menos, cierto grado de conflicto.

5.2. Trabajos relacionados

Computacionalmente, el CLP es un problema NP-duro [162], es decir, no se puede obtener una solución exacta en un tiempo polinomial. Por ello, aunque existe algún trabajo aislado para la formulación del problema vista en la sección anterior que lo aborda usando un algoritmo exacto [94], la mayoría de los estudios se centran en proporcionar soluciones mediante heurísticas y metaheurísticas. Las heurísticas se pueden desarrollar teniendo en cuenta diferentes distribuciones específicas: construcción de muros o *wall-building* [18, 154], disposición de bloques o *cuboid arrangement* [22, 69], construcción de pilas o *stack-building* [80, 102], y cortes de guillotina [60]. En la literatura relacionada existe una gran cantidad de algoritmos heurísticos para la formulación del problema aquí estudiada [18, 82, 83, 89, 154, 180, 188]. Sin embargo, en los últimos años se ha incrementado la atención hacia las metaheurísticas como los algoritmos genéticos [80, 88, 126, 186], el recocido simulado [30, 71, 104], la búsqueda tabú [22, 127], y los algoritmos híbridos [23, 58, 128, 142]. En ocasiones, estos procesos pueden incluir paralelizaciones, como es el caso de algunos algoritmos genéticos paralelos [81], búsquedas tabú paralelas [25], o búsquedas locales híbridas paralelas [132].

Otras formulaciones del problema consideran la posibilidad de disponer de varios contenedores. Igual que en el caso de la formulación seleccionada, en la literatura se proponen pocos algoritmos exactos [135], muchos algoritmos heurísticos [50, 69, 102, 113, 152, 165, 187] y algún híbrido [125]. En otros casos, se permite que una dimensión del contenedor sea infinita o abierta de tal modo que se han de colocar todas las piezas en el mismo [4, 26, 183]. El presente trabajo no se ha centrado en este tipo de formulación porque normalmente las empresas tienen un conjunto de camiones o contenedores que no son de cualquier largo, sino que tienen unas dimensiones específicas e inamovibles. Por otra parte, en la literatura también se pueden encontrar formulaciones del problema que tienen en cuenta piezas con formas irregulares [38, 68]. Sin embargo, la mayoría de los artículos a transportar se empaquetan en cajas rectangulares y no en paquetes irregulares.

Generalmente, a la hora de resolver un CLP, la maximización del espacio usado es el objetivo más importante, aunque hay casos en los que existen otras cuestiones que también se deberían tener en cuenta. En [18] se describen muchos requerimientos prácticos, tales como las restricciones de orientación, la distribución de la carga, la estabilidad, las prioridades de introducción, el límite de peso de los contenedores, etc. A pesar de ello, sólo se conoce un trabajo de la literatura que trata este problema como multi-objetivo [57], utilizando para ello un algoritmo de recocido simulado hibridizado con una heurística de relleno. Este trabajo se centra en los mismos objetivos que aquí se van a tratar.

5.3. Solución multi-objetivo

Tratar de resolver el CLP mediante un algoritmo exacto se ha descartado debido a la complejidad del problema (aún mayor que la del resto de problemas tratados en este trabajo), pues existen más posiciones y orientaciones posibles para las piezas. Esto implica que obtener soluciones para instancias un poco complejas o reales no resulte viable. Por lo tanto, se ha optado por desarrollar una propuesta aproximada. Además, como el problema suele plantearse en la realidad con más de un objetivo se ha propuesto un método que lo afronte directamente como multi-objetivo.

Los MOEAs se han mostrado prometedores a la hora de resolver problemas multi-objetivo del mundo real. Concretamente han proporcionado soluciones competitivas para otro tipo de problemas de corte y empaquetado, como se ha podido ver en los capítulos anteriores, por lo que también se va a comprobar su efectividad en este problema en particular. Como ya se ha comentado, en la literatura sólo se conoce una aproximación multi-objetivo [57] para este problema, y tiene en cuenta los mismos objetivos que en este trabajo: maximización del peso y maximización del volumen usado del contenedor. Por lo tanto, se pretende realizar una comparativa con esta aproximación para conocer la competitividad de los MOEAs con este problema concreto. Además, para mejorar el rendimiento y obtener resultados de buena calidad en menor tiempo se llevará a cabo un estudio de las paralelizaciones usando algunos modelos basados en islas que proporciona METCO.

Tal y como se ha visto para los anteriores problemas de corte, cuando se diseñan propuestas metaheurísticas para la solución de un problema dado, hay muchas decisiones de diseño que son cruciales para el impacto del método de resolución. Una de las decisiones principales es la relativa a la representación de las soluciones de la población. Igual que se ha explicado para los problemas anteriores, aquí se pueden diferenciar los mismos tres grupos de propuestas metaheurísticas, dependiendo del tipo de codificación elegida para las soluciones (Sección 3.6).

Para este problema se ha descartado el uso de una representación directa de los individuos mediante una notación postfija, ya que, a diferencia de los problemas de corte anteriormente tratados, en este caso existen más operadores de posición y un máximo de seis orientaciones posibles para cada caja, con lo que el espacio de búsqueda se incrementa demasiado. Obtener soluciones factibles en un tiempo aceptable con el uso de esta representación no es viable. Por lo tanto, se ha descartado el tercer tipo de metaheurísticas y se ha optado por una codificación para el segundo tipo, que indica el orden y la orientación en que se deben colocar las cajas y que requiere de una heurística de colocación para conocer la posición final de las mismas dentro del contenedor. A continuación se presenta dicha representación y el resto de información que necesita METCO sobre el problema para poder resolverlo.

		G_1	G_2	G_3		G_{s-2}	G_{s-1}	G_s
tipo de piezas	t_i	t2	t4	t1	···	t2	t3	t1
número de piezas	n_i	4	1	2	···	2	2	3
orientación de las piezas	r_i	1	0	1	···	4	0	2

Figura 5.3: Ejemplo de la representación de una solución

5.3.1. Representación

Se ha utilizado una representación de las soluciones candidatas con la estructura que se puede ver en la Figura 5.3. Siendo $s \in [1, N]$ el tamaño del cromosoma, que varía en función de la agrupación de piezas de cada tipo, el individuo estará formado por una secuencia de genes $G_1, \ldots, G_s$ donde cada gen consta de tres elementos (t_i, n_i, r_i), con $t_i \in [t_1, t_m]$, $n_i \in [1, b_i]$ y $r_i \in [0, o_i)$. Se trata por lo tanto de una secuencia de 3-tuplas formadas por: tipo de pieza, número de piezas de ese tipo y rotación para esas piezas. De esta manera se determina el orden en el que se van a colocar todas las piezas en el contenedor y sus orientaciones correspondientes. Para que el cromosoma sea válido, debe contener la totalidad de las piezas:

$$\forall j \in [1, m] \sum_{k=1}^{s} z_{jk} = b_j \left\{ \begin{array}{lll} z_{jk} = n_k & \text{si} & t_k = t_j \\ z_{jk} = 0 & \text{si} & t_k \neq t_j \end{array} \right\} \tag{5.3}$$

5.3.2. Generación de la población inicial

Para la generación inicial de individuos en primer lugar se ha de generar aleatoriamente un tipo de pieza t_i para el que aún queden piezas por colocar, es decir, que el número de piezas de tipo i colocadas en el contenedor, x_i, sea menor que el número total de piezas disponibles de ese tipo, b_i $(x_i < b_i)$. Luego se genera la cantidad de piezas de ese tipo, n_i, que se van a considerar en el gen actual. Esta cantidad debe ser menor o igual al número de piezas de ese tipo que queden por colocar $(n_i \leq b_i - x_i)$. Finalmente, se genera una rotación permitida para ese conjunto de piezas $r_i \in [0, o_i)$. Estos pasos se han de realizar hasta que se haya introducido la totalidad de las piezas en el cromosoma, es decir, $\forall i \in [1, m]$ $x_i = b_i$.

5.3.3. Evaluación de los objetivos

Cada cromosoma representa un cierto orden en el que las piezas se van a colocar en el contenedor y con qué orientación deben situarse. Pero para obtener la distribución

concreta de las piezas dentro del contenedor se utiliza una heurística de colocación. Al aplicar la heurística de colocación sobre un individuo, se obtienen intrínsecamente los valores de ambos objetivos de optimización, volumen y peso total, ya que se sabrá qué piezas se han colocado en el contenedor. Dicha heurística se basa en el almacenamiento de las piezas dentro del contenedor por capas o niveles. Una vez creada la secuencia de piezas $q_1, \ldots, q_N$ indicada por el cromosoma con las orientaciones que señale, la heurística funciona del siguiente modo (Algoritmo 8):

- Se introduce en la primera zona libre en el sentido positivo de los ejes de coordenadas la primera pieza de la secuencia que quepa en el ancho libre del contenedor, de manera que no se supere el peso límite. Esta pieza determinará el ancho de la capa, cuyo largo y alto coincidirán con los del contenedor. Así se generarán dos huecos, uno en el *eje z* (sentido positivo) y otro en el *eje y* (sentido positivo), tal y como se puede ver en la Figura 5.4.

- Se recorre el resto de la secuencia de piezas intentando introducir la siguiente pieza en la capa actual, respetando siempre el límite de peso del contenedor. Si no se consigue colocar en la capa actual, se pasa a intentar colocar la siguiente pieza de distinto tipo de la secuencia. Cuando se haya recorrido toda la secuencia y ninguna pieza haya encajado en la capa actual, se creará una capa adyacente a la previa en el sentido del *eje x*, y se volverá a recorrer la secuencia de piezas para intentar colocar las que no se hayan podido colocar anteriormente. El ancho de la nueva capa vendrá dado por el ancho de la primera pieza que se coloque en la misma, y su largo y alto serán los del contenedor. Para realizar la colocación de piezas en una capa se siguen las siguientes pautas:

  - Se intentan llenar en primer lugar los huecos que se han creado en el *eje y*. Si no se encuentra ningún hueco en el *eje y* que pueda acoger la pieza, se pasa a comprobar si cabe en un hueco del *eje z*. Si tampoco cabe, se comprueba si cabe en un hueco del *eje x*. Si finalmente no cabe, entonces se descarta la pieza para la capa actual, y se intentará introducir cuando se cree la siguiente.
  - Cuando se encuentra un hueco con dimensiones suficientes para albergar la pieza, se comprueba, en primer lugar, si las dimensiones del hueco coinciden exactamente con las de la pieza. Si es así, se elimina el hueco, pues es ocupado en su totalidad por la pieza en cuestión. De lo contrario, se sustituye el hueco por la colocación de la pieza y por la generación de un máximo de tres nuevos huecos: uno por el espacio libre que haya quedado en cada uno de los ejes.

Algoritmo 8 Evaluación de los objetivos del CLP

1: $pieces = \{q_1, q_2, ..., q_N\}$;
2: $freeWidth = W$;
3: $totalVolume = 0$;
4: $totalWeight = 0$;
5: $list_x = \emptyset$; // lista de huecos en el eje x
6: $list_y = \emptyset$; // lista de huecos en el eje y
7: $list_z = \emptyset$; // lista de huecos en el eje z
8: **while** ($pieces \neq \emptyset$) y ($\exists q_x \subset pieces,\ q_x \in T_i,\ i \in [1, m]$ / ($freeWidth \geq w_i$) y ($totalWeight + p_i \leq P_{max}$)) **do**
9: crear una capa $layer$ / $width(layer) = w_i$, $length(layer) = L$, $height(layer) = H$;
10: $freeWidth = freeWidth - w_i$;
11: $totalVolume = totalVolume + v_i$;
12: $totalWeight = totalWeight + p_i$; // Se introduce la pieza en el contenedor
13: **if** ($length(layer) - l_i \neq 0$) **then**
14: crear hueco en el eje y $hole_y$ e introducirlo en la lista de huecos $list_y$;
15: $length(hole_y) = L - l_i$;
16: $width(hole_y) = w_i$;
17: $height(hole_y) = h_i$;
18: **end if**
19: **if** ($height(layer) - h_i \neq 0$) **then**
20: crear hueco en el eje z $hole_z$ e introducirlo en la lista de huecos $list_z$;
21: $length(hole_z) = L$;
22: $width(hole_z) = w_i$;
23: $height(hole_z) = H - h_i$;
24: **end if**
25: eliminar q_x de $pieces$;
26: **while** ($\exists q_x \subset pieces,\ q_x \in T_i,\ i \in [1, m]$ / $hole \in list_y$ o $hole \in list_z$ o $hole \in list_x$, $l_i \leq length(hole)$, $w_i \leq width(hole)$, $h_i \leq height(hole)$) y ($totalWeight + p_i \leq P_{max}$) **do**
27: **if** ($l_i == length(hole)$) y ($w_i == width(hole)$) y ($h_i == heigth(hole)$) **then**
28: borrar hueco $hole$ de la lista a la que pertenezca;
29: **else**
30: borrar hueco $hole$ de la lista a la que pertenezca;
31: **if** ($width(hole) - w_i \neq 0$) **then**
32: crear nuevo hueco en el eje x $hole_x$ e introducirlo en la lista de huecos $list_x$;
33: $length(hole_x) = l_i$;
34: $width(hole_x) = width(hole) - w_i$;
35: $height(hole_x) = h_i$;
36: **end if**
37: **if** ($length(hole) - l_i \neq 0$) **then**
38: crear nuevo hueco en el eje y $hole_y$ e introducirlo en la lista de huecos $list_y$;
39: $length(hole_y) = length(hole) - l_i$;
40: $width(hole_y) = width(hole)$;
41: $height(hole_y) = h_i$;
42: **end if**
43: **if** ($height(hole) - h_i \neq 0$) **then**
44: crear nuevo hueco en el eje z $hole_z$ e introducirlo en la lista de huecos $list_z$;
45: $length(hole_z) = length(hole)$;
46: $width(hole_z) = width(hole)$;
47: $height(hole_z) = height(hole) - h_i$;
48: **end if**
49: **end if**
50: $totalVolume = totalVolume + v_i$;
51: $totalWeight = totalWeight + p_i$; // se introduce la pieza en el contenedor
52: eliminar q_x de $pieces$;
53: **end while**
54: **end while**

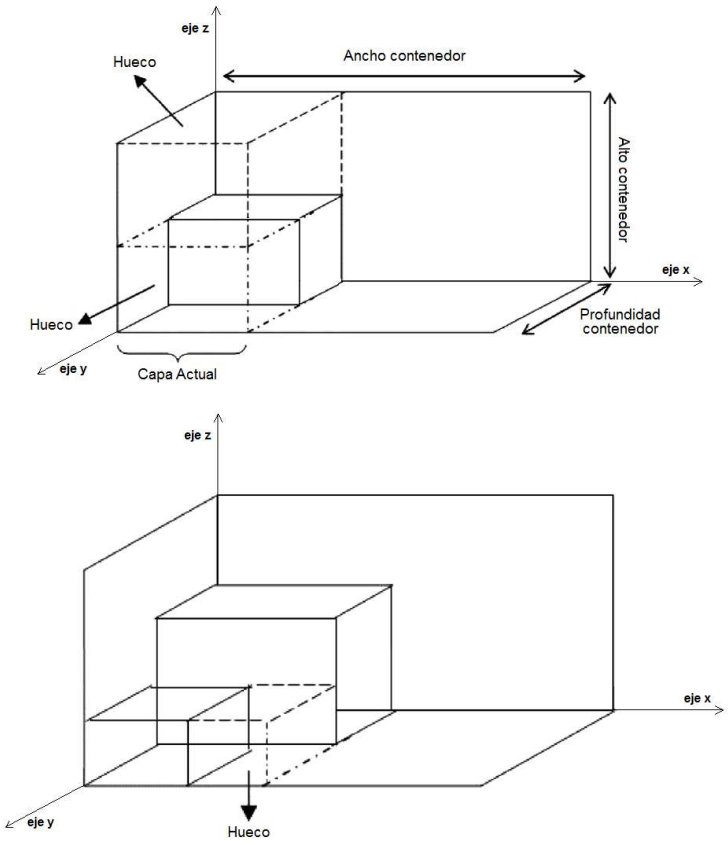

Figura 5.4: Creación de huecos

- Cuando no se puedan crear más capas porque no quede espacio suficiente o porque se haya superado el peso límite, se habrán introducido todas las piezas posibles, y el valor de los objetivos (volumen y peso) se podrá calcular fácilmente sumando los volúmenes y pesos de todas las piezas introducidas en el contenedor.

5.3.4. Operadores

Los distintos tipos de operadores que se apliquen deben respetar la integridad de los cromosomas, es decir, después de aplicar los operadores, cada cromosoma ha de seguir conteniendo todas las piezas demandadas de cada tipo, y cada pieza debe tener asociada una orientación permitida para su tipo. Teniendo en cuenta un individuo representado por el cromosoma $C1 = (G1_1, \ldots, G1_{s1})$ y otro individuo representado por el cromosoma $C2 = (G2_1, \ldots, G2_{s2})$, se ha implementado un cruce de un punto que funciona del siguiente modo (Figura 5.5):

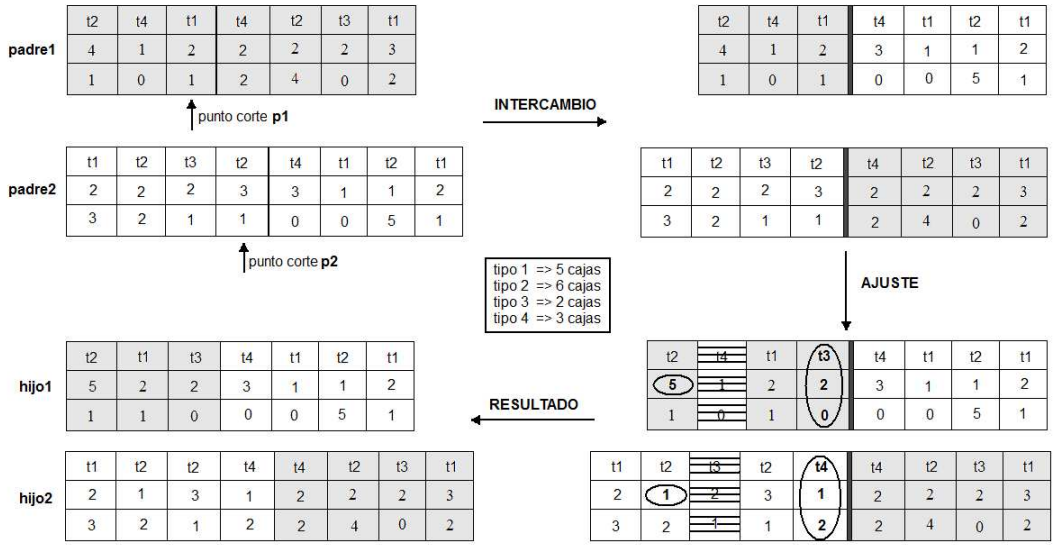

Figura 5.5: Operador de cruce

- En primer lugar, para cada individuo se selecciona un punto al azar, $p1 \in [1, s1]$ y $p2 \in [1, s2]$.

- Se intercambian las partes de cada individuo después de los puntos seleccionados de manera que $C1 = (G1_1, \ldots, G1_{p1}, G2_{p2+1}, \ldots, G2_{s2})$ y $C2 = (G2_1, \ldots,$-$G2_{p2}, G1_{p1+1}, \ldots, G1_{s1})$.

- Se modifica la primera parte de cada individuo, $G1_1, \ldots, G1_{p1}$ y $G2_1, \ldots, G2_{p2}$, para que se respete el número total de piezas de cada tipo presente en el cromosoma. Es decir, se comprueba si la condición de la ecuación 5.3 se sigue cumpliendo. Si no se cumple:

  - Suponiendo que cada gen del cromosoma $C1$ es una 3-tupla de la forma (t_x, n_x, r_x), $\forall t_x \in [t_1, t_m] / \sum_{i=1}^{s2} n_x > b_x$, se recorre la primera parte $G1_1, \ldots, G1_{p1}$ buscando el primer gen $G1_y = (t_y, n_y, r_y)$ que utilice el mismo tipo de pieza, es decir, $t_x = t_y$, y se elimina el número de piezas en exceso $e_x = \sum_{i=1}^{s2} n_x - b_x$ de manera que $n_y = n_y - e_x$ y $e_x = e_x - n_y$, si $n_y > e_x$. Si $n_y \leq e_x$, entonces $e_x = e_x - n_y$ y se borra el gen $G1_y$ completo, desplazándose los siguientes genes una posición a la izquierda. Si después de haber eliminado dichas piezas siguen faltando piezas de ese tipo por eliminar, $e_x > 0$, se seguirá haciendo lo mismo con el resto de genes de la primera parte del cromosoma, $G1_z = (t_z, n_z, r_z) / z \leq p1$ y $t_x = t_z$, hasta que $e_x = 0$.

- Si cada gen del cromosoma $C1$ es una 3-tupla (t_x, n_x, r_x), $\forall t_x \in [t_1, t_m]$ $\sum_{i=1}^{s2} n_x < b_x$, el número de piezas que faltan de ese tipo será $d_x = b_x - \sum_{i=1}^{s2} n_x$. Entonces, se recorre la primera parte del cromosoma $G1_1, \ldots, G1_{p1}$ buscando el primer gen $G1_y = (t_y, n_y, r_y)$ con $t_y = t_x$ para sumarle esas piezas que faltan, $n_y = n_y + d_x$. Si no existe un gen de ese tipo t_x en $G1_1, \ldots, G1_{p1}$, se agregará uno al final de la primera parte del gen $G1_{p1+1} = (t_x, d_x, r_x)$, con una orientación $r_x \in [0, o_x)$. Los genes de la segunda parte del cromosoma se desplazan una posición a la derecha.

Se actúa de igual modo para el cromosoma $C2$.

Para la mutación se han introducido tres tipos diferentes de movimientos u operaciones sobre el cromosoma, aplicado cada uno bajo la probabilidad de mutación:

- *Agregar un gen*: se selecciona de forma aleatoria un tipo de pieza $t_x \in [t_1, t_m]$. A continuación, se selecciona aleatoriamente un gen $G_y = (t_y, n_y, r_y)$ tal que $t_y = t_x$ y $n_y > 1$. Del gen seleccionado se escoge aleatoriamente un número de piezas $n_x < n_y$ de manera que queden repartidas entre ese gen $(t_y, n_y - n_x, r_y)$ y el nuevo que se va a agregar (t_x, n_x, r_x). La orientación se selecciona tal que esté entre las permitidas para ese tipo de pieza $r_x \in [0, o_x)$. Entonces, se escoge la posición del cromosoma en la cual se va a insertar el nuevo gen desplazando el resto hacia la derecha (Figura 5.6).

- *Eliminar un gen*: se selecciona aleatoriamente una posición x dentro del cromosoma, tal que el gen en esa posición es $G_x = (t_x, n_x, r_x)$. A continuación se selecciona aleatoriamente una posición y con un gen $G_y = (t_y, n_y, r_y)$ tal que $t_y = t_x$ (si no existe en el cromosoma ningún y tal que $t_y = t_x$, entonces no se llevará a cabo la operación). Para poder eliminar el gen G_x, el número de piezas n_x se transfiere al gen G_y, tal que $n_y = n_y + n_x$ Como puede suceder que ambos genes no posean el mismo tipo de orientación de piezas $(r_x \neq r_y)$, se selecciona uno de los dos de forma aleatoria. Finalmente, se elimina el gen realizando compactación hacia la izquierda (Figura 5.7).

- *Cambiar un gen*: se selecciona una posición aleatoria x del cromosoma y se le cambia al gen $G_x = (t_x, n_x, r_x)$, también de forma aleatoria, el tipo de orientación r_x dentro de las posibles para el tipo de pieza t_x (Figura 5.8).

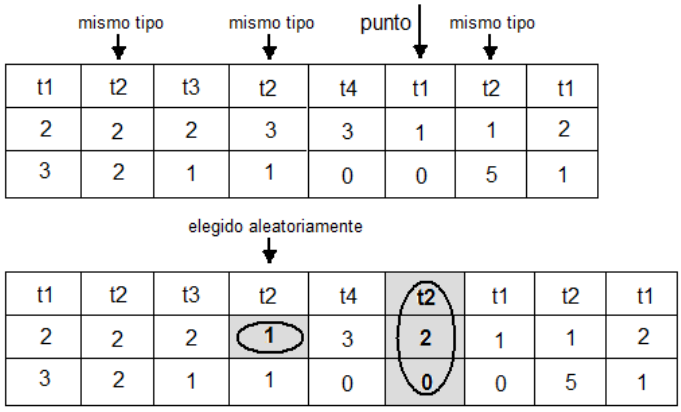

Figura 5.6: Operador de mutación *Agregar t2*

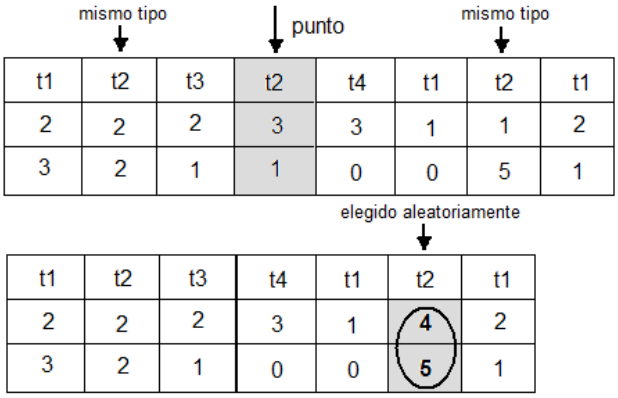

Figura 5.7: Operador de mutación *Eliminar*

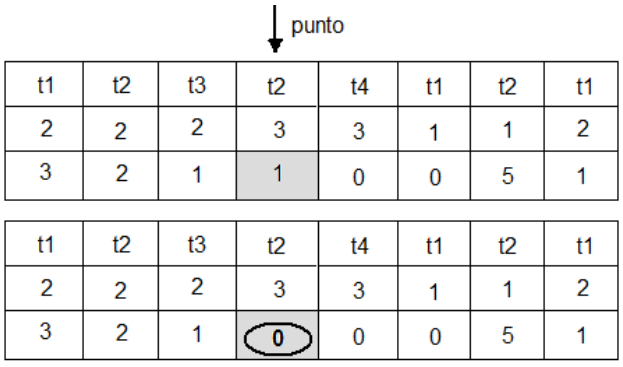

Figura 5.8: Operador de mutación *Cambiar*

ID	Descripción	ancho/largo/alto(cm)	N.cajas	Peso(kg)	V.total(cm^3)
1	Detergente	40/36/28	325	20.00	13,104,000
2	Lejía	54/28/30	25	22.00	1,134,000
3	Cuidado Personal	54/28/30	75	2.35	3,402,000
4	Detergente	39/29/32	75	20.00	2,714,400
5	Producto Afeitar	15/10/20	10	1.47	30,000
6	Cuidado Bebés	42/37/25	150	2.80	5,827,500
7	Pasta Dientes	36/18/18	3	3.72	34,992
8	Champú	18/17/22	25	4.97	168,300
9	Champú	22/17/22	50	5.00	411,400
10	Champú	12/11/16	5	1.33	10,560
11	Lejía	30/27/40	20	17.60	648,000
12	Producto Afeitar	19/7/21	3	0.12	8,379
Total			766	9905.37	27,493,531

Tabla 5.2: Instancia de referencia

5.4. Resultados computacionales

Tal y como se ha comentado en la sección anterior, el CLP suele plantearse en la realidad con más de un objetivo por lo que aquí se ha afrontado directamente una solución multi-objetivo del mismo. Los MOEAs han proporcionado resultados competitivos cuando se han aplicado al resto de problemas de corte y empaquetado aquí tratados, por lo que una vez más se va a comprobar su efectividad. Como sólo se conoce una aproximación multi-objetivo [57] para este problema, que además tiene en cuenta los mismos objetivos que en este trabajo, se utilizará de referencia para comprobar la competitividad de la actual propuesta. Gracias al uso de METCO, se evita la implementación desde cero de MOEAs, ya que proporciona implementaciones de algunos de los MOEAs más utilizados en la literatura.

Para comenzar con la nueva propuesta multi-objetivo, se ha llevado a cabo una evaluación experimental usando un cluster dedicado de 20 nodos de doble núcleo con un sistema operativo Debian GNU/Linux. El framework y la aproximación del problema se han implementado en C++ y compilado con *gcc 4.1.3* y MPICH *1.2.7*. Para el estudio computacional, se ha utilizado la instancia real propuesta en [57]. Se trata de un pedido con 12 tipos de productos diferentes que distribuye una empresa, junto con sus cantidades, dimensiones y pesos. La empresa usa sus propios vehículos para el transporte. Estos vehículos tienen una capacidad de $(530 \times 220 \times 210)$ cm^3 y pueden albergar un peso máximo de 7200 kg (Tabla 5.2). En el artículo que se ha tomado como referencia no se da información acerca de las orientaciones de las cajas. Por ello y dado que algunas de ellas contienen líquidos, se ha supuesto que sólo es posible rotar la base de las cajas, con lo que sólo serían posibles las orientaciones 0 y 1 de la Figura 5.1.

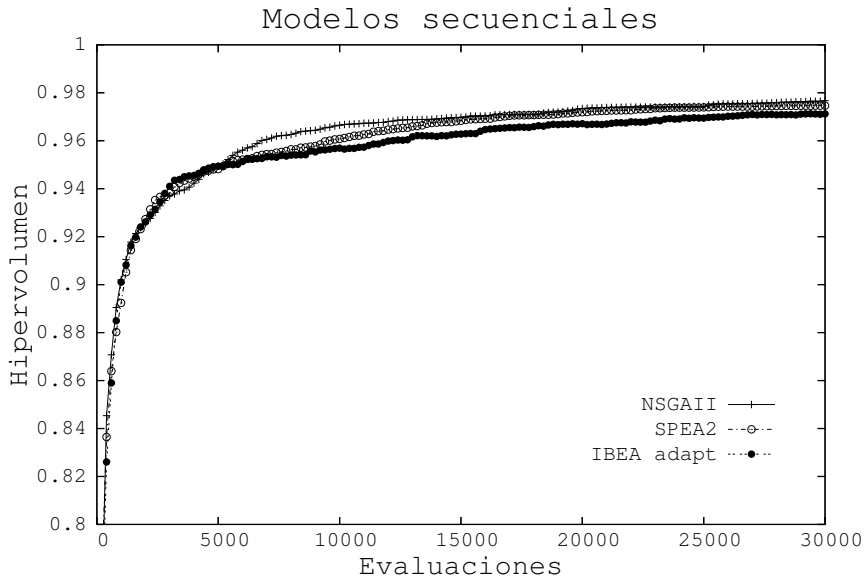

Figura 5.9: Hipervolumen para distintos algoritmos evolutivos

En dicho trabajo, el problema se resuelve utilizando un algoritmo de recocido simulado junto a una heurística de relleno, y asignando diferentes pesos, desde 0 hasta 1, a cada objetivo. Los parámetros del algoritmo de recocido simulado utilizados para el problema son: temperatura inicial $T_{in} = 5000$, temperatura final $T_f = 0.0001$, constante $\alpha = 0.987$ y número de iteraciones que debe de llevarse a cabo para cada temperatura $il_{max} = 12$.

En el presente estudio, se han utilizado distintos MOEAs, SPEA2, NSGA-II e *Indicator-Based Evolutionary Algorithm*, IBEA adaptativo, para realizar un análisis del comportamiento de los mismos. Después de realizar una serie de pruebas se han establecido los valores de los parámetros para los algoritmo: la probabilidad de mutación se ha fijado a 0.3, la probabilidad de cruce se ha fijado a 0.9 y el tamaño de población a 20. Para la versión adaptativa del IBEA, el *fitness scaling factor* se ha fijado a 0.001. Para saber cuál de ellos proporciona resultados de mejor calidad se ha utilizado la métrica hipervolumen. En la Figura 5.9 se muestra el hipervolumen para cada algoritmo. El IBEA adaptativo se encuentra un poco por debajo, pero tanto el NSGA-II como el SPEA2 presentan buenos resultados. Sin embargo, el NSGA-II se encuentra ligeramente por encima del SPEA2 y, como además, se ha demostrado su buen comportamiento en otros problemas de corte y empaquetado [51, 54] ya tratados en este trabajo, ha sido seleccionado para realizar el presente estudio.

En primer lugar, se ha llevado a cabo un estudio secuencial, realizando 30 ejecucio-

	Media Volumen	Media Peso	Valor 1	Valor 2	Valor 3	Valor 4
Volumen (%)	91.13 %	90.87 %	87.51 %	86.13 %	86.09 %	85.96 %
(cm^3)	22315526	22250896	21427698	21089791	21079997	21048165
Peso (kg)	7197.39	7199.52	6713.37	7157.06	7150.41	7151.37

Tabla 5.3: Comparativa de objetivos

nes de 30000 evaluaciones (1500 generaciones) cada una. El tiempo medio empleado en cada ejecución se encuentra en torno a los 10 minutos. Si se calcula la media del mayor volumen obtenido en cada uno de los 30 frentes finales, se puede comprobar que el valor expresado en porcentaje de volumen usado del contenedor (91.13 %), supera claramente el mejor porcentaje (87.51 %) de los resultados aportados por el artículo de referencia [57]. Lo mismo ocurre si se calcula la media del mayor peso en cada uno de los 30 frentes finales, obteniéndose un valor de 7199.52 kg frente a los 7157.06 kg de referencia.

La Tabla 5.3 muestra las dos medias anteriormente halladas con el correspondiente valor del objetivo contrario y las cuatro soluciones diferentes conseguidas en [57] para la instancia tratada. Claramente, los resultados obtenidos mejoran los valores de referencia, obteniéndose mayor volumen empaquetado que el mejor de los cuatro valores de volumen de referencia, incluso cuando nos centramos en los resultados con mayores pesos asociados (segunda columna).

Hasta el momento se han visto las medias de los valores extremos para cada objetivo. Pero sería interesante identificar el área del espacio de búsqueda que se está explorando y ver el comportamiento general del algoritmo evolutivo aplicado a este problema. Si se dibujan directamente los 30 frentes de Pareto puede ser muy confuso, por lo que aquí se han utilizado las superficies de alcance [73]. La Figura 5.10 muestra las superficies de alcance 1, 15 y 30 para la instancia tratada y los puntos que se proporcionan en el artículo utilizado de referencia. Como se puede observar, se ofrece un conjunto de soluciones de compromiso entre los dos objetivos, de manera que la persona encargada de tomar las decisiones podrá elegir la opción que más convenga en cada caso: si se necesita una solución que proporcione un volumen lo más grande posible, dicha solución tendrá asociado un peso menor que otra solución que proporcione un volumen de empaquetado más bajo, y viceversa. Sin embargo, las diferentes soluciones propuestas en [57] quedan por debajo de las superficies de alcance, siendo completamente dominadas por las nuevas soluciones. El primer punto de referencia (21427698, 6713.37) no está representado ya que se sale muy por debajo de la gráfica (el valor del peso es mucho menor que lo representado, pero dado su volumen asociado, también es una solución dominada. De este modo se demuestra la validez de los algoritmos evolutivos multi-objetivo para hacer frente a

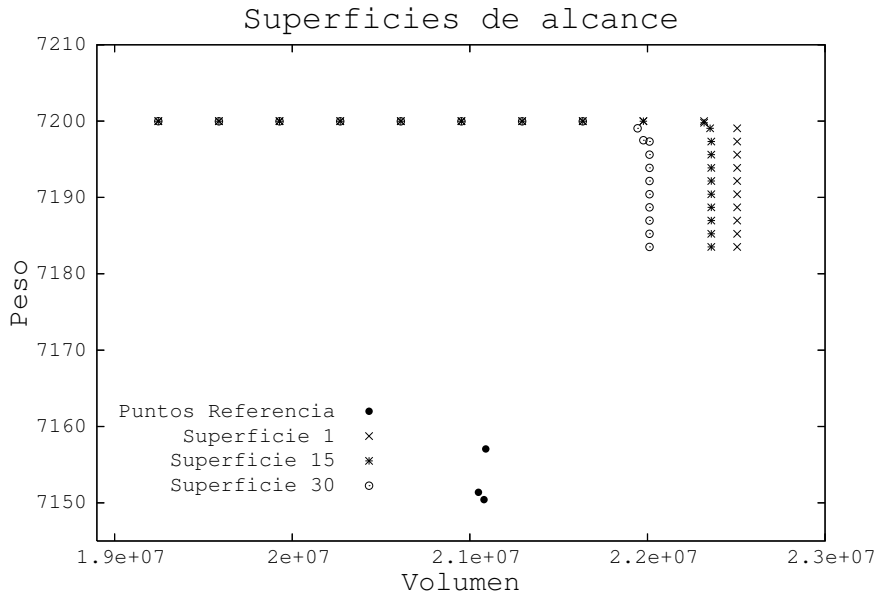

Figura 5.10: Superficies de alcance para las 30 ejecuciones

este problema, superando los resultados obtenidos mediante un algoritmo de recocido simulado. Además se obtiene el conjunto de soluciones de compromiso en una sola ejecución, mientras que el algoritmo aplicado en [57] asigna distintos pesos a cada objetivo, y para cada una de las combinaciones de pesos se requiere de una ejecución distinta.

Es necesario destacar que la forma que presentan las superficies de cubrimiento puede resultar extraña porque parece que existen muchos valores iguales para el peso y para el volumen. Sin embargo, es un efecto de las escalas de los ejes de la gráfica. Por ejemplo, parece que existen muchas soluciones con peso 7200 kg, pero si se aumenta la escala se puede observar que los valores son diferentes, aunque muy cercanos: 7199.99, 7199.90, 7199.50, etc.

A diferencia de los problemas de corte anteriormente tratados, en este caso carece de sentido realizar una comparativa entre los resultados multi-objetivo y resultados mono-objetivo, ya sean exactos o aproximados. No sería una comparativa justa puesto que cuando se trata este problema con múltiples objetivos se está estableciendo un límite de peso que puede alcanzar el contenedor, con lo cual las soluciones están acotadas. Mientras que en una propuesta mono-objetivo, como no se tiene en cuenta el peso, los volúmenes alcanzados serán claramente superiores, puesto que se ha podido comprobar que caben más cajas en el contenedor que las introducidas en

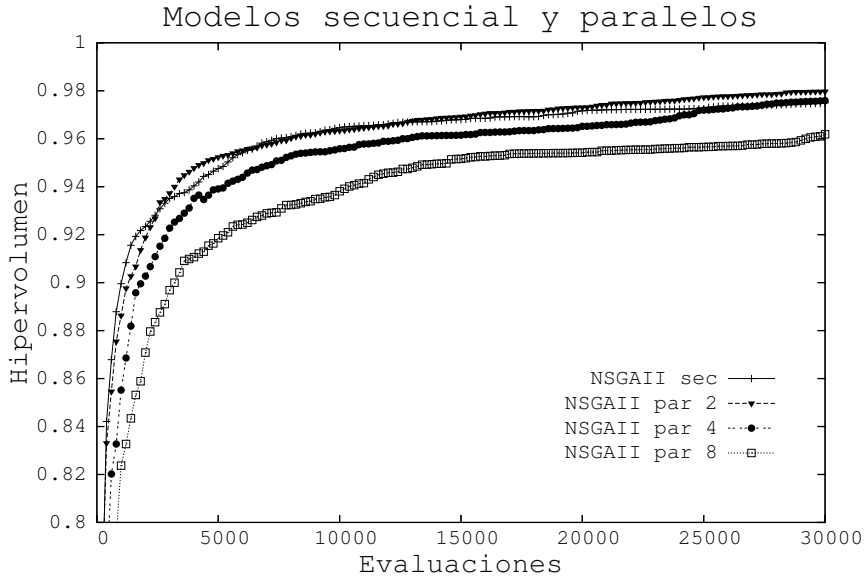

Figura 5.11: Evolución del hipervolumen

la solución de máximo peso obtenida por la aproximación multi-objetivo. Esto no ocurría en los problemas de corte anteriormente tratados, donde dicha comparativa cobraba sentido, ya que el número de cortes, que se tomó como segundo objetivo, no estaba limitado a una cantidad específica. Por lo tanto la aproximación multi-objetivo era capaz de llegar a valores cercanos e incluso mejores del primer objetivo con respecto a los aportados por aproximaciones mono-objetivo.

En definitiva, una vez que se ha verificado que utilizando MOEAs se obtienen mejores resultados que con el algoritmo de recocido simulado tomado como referencia, sería interesante ver si los modelos paralelos basados en islas que ofrece METCO pueden mejorar el comportamiento del modelo secuencial. El modelo más sencillo que se puede probar es el modelo de islas *homogéneo*, donde todas las islas ejecutan la misma configuración de un MOEA, el NSGA-II en este caso.

Así, se han realizado ejecuciones usando el modelo homogéneo con el algoritmo NSGA-II y los mismos parámetros aplicados en la ejecución secuencial, empleando dos, cuatro y ocho islas En la Figura 5.11 se muestra el hipervolumen [193] para cada una de las versiones. Los resultados para las ejecuciones paralelas de dos islas consiguen mejorar los resultados secuenciales especialmente cuando va aumentando el número de evaluaciones y, por tanto, el tiempo. Las ejecuciones con cuatro islas sólo consiguen alcanzar la calidad de las secuenciales al final, mientras que las de

HIPERVOL.	Secuencial		Paralelo 2		Paralelo 4		Paralelo 8	
	Eval.	Tiempo	Eval.	Tiempo	Eval.	Tiempo	Eval.	Tiempo
80 %	333	7.598	150	1.811	130	0.796	65	0.207
85 %	493	11.249	240	2.897	210	1.286	155	0.494
90 %	800	18.255	516	6.229	388	2.376	291	0.928
95 %	2880	65.718	1170	14.125	836	5.121	789	2.516
99 %	7125	162.584	5051	60.979	3071	18.811	2175	6.937

Tabla 5.4: Número de evaluaciones y tiempo por porcentaje de hipervolumen

ocho islas no son capaces de mejorar la calidad de las soluciones secuenciales en el mismo número de evaluaciones totales. Esto es debido a que cuantas más islas se utilicen menor es el número de evaluaciones o generaciones que realiza cada una de ellas por separado, es decir, las mismas evaluaciones son repartidas entre más procesadores, con lo cual cada uno avanza menos en la exploración. Si se realiza un análisis en profundidad de cuántas evaluaciones en promedio necesita cada procesador para alcanzar un determinado porcentaje de hipervolumen (Tabla 5.4) se puede comprobar que cada procesador en las ejecuciones paralelas lleva a cabo menos evaluaciones que el procesador en la ejecución secuencial. Además, se observa que al aumentar la calidad de la solución el número de evaluaciones que necesita cada procesador va aumentando cada vez en mayor proporción. Por otra parte, al disminuir el número de evaluaciones realizadas por cada procesador en las ejecuciones paralelas con respecto al secuencial, disminuye el tiempo necesario para alcanzar soluciones de una determinada calidad. En este sentido, es notable el beneficio que aportan las ejecuciones paralelas, ya que el tiempo para alcanzar soluciones de una buena calidad cercana a la más óptima se reduce drásticamente.

5.5. Conclusiones

Se ha abordado la resolución de una versión multi-objetivo del CLP aplicando MOEAs para comprobar su efectividad con este problema, dado que aplicados a anteriores problemas de corte se han obtenido resultados competitivos. Los dos objetivos que se han tenido en cuenta son: maximizar el volumen usado del contenedor, y maximizar el peso total empaquetado dentro de un límite. En la literatura sólo se conoce la referencia de un trabajo que aborda este problema con múltiples objetivos y para ello utiliza un algoritmo de recocido simulado. Como trata los mismos dos objetivos, se ha tomado como referencia para conocer la calidad de los resultados obtenidos. En primer lugar, tras analizar varios MOEAs, se ha seleccionado el NSGA-II dado su buen comportamiento. Este algoritmo ha proporcionado un

conjunto de soluciones de compromiso entre ambos objetivos, sin necesitar varias ejecuciones con diferentes pesos para cada objetivo como ocurre en el caso del algoritmo de recocido simulado tomado como referencia. Además, dichas soluciones dominan completamente los resultados de referencia, con lo cual se ha demostrado una vez más el buen comportamiento de la estrategia evolutiva ante este tipo de problemas.

En un intento de mejorar aún más el comportamiento del algoritmo secuencial, se han propuesto algunas paralelizaciones homogéneas basadas en islas. Concretamente, se han realizado ejecuciones con dos, cuatro y ocho islas utilizando el algoritmo NSGA-II en todas ellas. Como resultado, se ha comprobado que con todas las paralelizaciones del algoritmo evolutivo se obtienen soluciones de calidad en menor número de evaluaciones y, por tanto, en menor tiempo. Algunas paralelizaciones son capaces incluso de mejorar la calidad, como es el caso de la ejecución con dos islas. Sin embargo, el modelo paralelo de ocho islas no consigue alcanzar la calidad final alcanzada por la propuesta secuencial, dado que al aumentar el número de islas el número evaluaciones se reparte entre más procesadores, de modo que cada isla puede avanzar menos en cada exploración. Es decir, cada isla tiene menos tiempo para explorar, lo cual se puede comprobar observando el tiempo que emplea la ejecución paralela de ocho islas para alcanzar el 99 % del hipervolumen de la ejecución secuencial. Este tiempo es 23 veces inferior al empleado por la ejecución secuencial.

Por lo tanto, se puede concluir que el algoritmo NSGA-II secuencial mejora los únicos resultados conocidos para este problema multi-objetivo, y además, sus paralelizaciones consiguen reducir el tiempo invertido e incluso mejorar la calidad de las soluciones. Con esto se demuestra que la estrategia evolutiva, al igual que en los anteriores problemas de corte, proporciona buenos resultados para este tipo de problemas.

Parte III

Conclusiones y trabajo futuro

Capítulo

6 Conclusiones y trabajo futuro

Los problemas de corte y empaquetado son una familia de problemas de optimización combinatoria que han sido extensamente estudiados en numerosas áreas de la industria y la investigación, debido a sus amplias aplicaciones prácticas. Por ello, el principal objetivo de este trabajo consiste en el diseño e implementación de métodos efectivos y eficientes para resolver distintos problemas de corte y empaquetado. Además, si estos métodos se definen como esquemas generales, se podrán aplicar a diferentes problemas de corte y empaquetado sin realizar demasiados cambios para adaptarlos a cada uno. De este modo, teniendo en cuenta el amplio rango de metodologías de resolución de problemas de optimización y las técnicas disponibles para incrementar su eficiencia, se han diseñado e implementado diversos métodos para resolver varios problemas de corte y empaquetado, tratando de mejorar las propuestas existentes en la literatura. Como resumen, este capítulo presenta las conclusiones obtenidas tras realizar estos estudios con cada problema tratado, así como el trabajo futuro que queda por llevar a cabo.

6.1. Conclusiones

En concreto, los problemas de corte y empaquetado que se han abordado en este trabajo han sido: el *Two-Dimensional Cutting Stock Problem*, el *Two-Dimensional Strip Packing Problem*, y el *Container Loading Problem*. Para cada uno de estos problemas se ha realizado una amplia y minuciosa revisión bibliográfica, y se ha obtenido la solución a los mismos aplicando diferentes métodos de resolución: métodos exactos mono-objetivo y paralelizaciones de los mismos, así como métodos aproximados multi-objetivo y paralelizaciones de los mismos. Los métodos exactos mono-objetivo utilizados se han basado en exploraciones en árbol mediante búsquedas primero-el-mejor. Como métodos aproximados multi-objetivo se han seleccionado un tipo de metaheurísticas multi-objetivo, los MOEAs. Además, para la representación de los individuos utilizados por estos métodos se han empleado codificaciones directas mediante una notación postfija, y codificaciones que usan heurísticas de colocación e hiperheurísticas. Tal y como se ha mencionado, Algunas de estas metodologías se han mejorado utilizando esquemas paralelos haciendo uso de las herramientas de programación OpenMP y MPI. En el caso de las propuestas exactas, las implementaciones paralelas han permitido obtener las soluciones óptimas más rápido, permitiendo abordar instancias más grandes de los problemas. Sin embargo, en el caso de las propuestas aproximadas, los esquemas paralelos no solo han permitido mejorar los tiempos, sino incluso la calidad de las soluciones obtenidas por el algoritmo secuencial.

En primer lugar, se ha abordado el *Two-Dimensional Cutting Stock Problem*. Este problema consiste en distribuir un conjunto de piezas con un beneficio asociado, sobre un material de dimensiones fijas, de tal manera que el beneficio total obtenido sea el máximo posible. Las piezas se tienen que poder obtener utilizando cortes de tipo guillotina, esto es, los cortes deben realizarse desde un lado del rectángulo hasta el opuesto yendo paralelamente a los otros dos bordes. En la literatura existen algunas referencias de trabajos que se centran en encontrar las soluciones óptimas para este problema. Esto es de vital interés sobre todo en industrias que manejan materiales con un elevado coste, y que, por tanto, no pueden admitir más desperdicio del estrictamente necesario. Estos métodos también son interesantes para aquellos problemas en los que no hayan restricciones excesivas en cuanto al tiempo necesario para obtener las soluciones. Por todo ello, el primer estudio realizado para este problema se centra en la implementación de un algoritmo exacto y en el posterior refinamiento del mismo para conseguir aumentar en la medida de lo posible su eficiencia. Este algoritmo está basado en el algoritmo MVB al que se le han agregado una serie de reglas de detección de dominancias y duplicados que hacen posible la reducción significativa del espacio de búsqueda, permitiendo mejorar la eficiencia del

algoritmo primero-el-mejor. Las reglas de detección de dominancias y duplicados se basan en las propiedades internas de las soluciones del problema, es decir, en la combinación de piezas para generar los patrones de corte. Algunas reglas se pueden aplicar antes del proceso de generación de patrones (pre-generación) y otras, por el contrario, se aplican después de la creación (post-generación). Se ha podido comprobar que aunque las reglas de post-generación se aplican con menos frecuencia, tienen más efecto en la reducción del tiempo total de búsqueda. Estas reglas no son tan costosas computacionalmente y además son capaces de detectar construcciones redundantes que también detectan las reglas de pre-generación.

De cualquier modo, incluso con la detección de patrones duplicados y dominados, el espacio de búsqueda es demasiado grande para afrontar las instancias más complejas del problema. Por ello, se han propuesto dos esquemas paralelos partiendo del algoritmo exacto más eficiente, es decir, con las reglas de detección de patrones dominados y duplicados. La primera propuesta está basada en un modelo de grano fino que ejecuta en paralelo los bucles de generación. El segundo esquema está basado en un modelo de grano grueso que realiza la ejecución paralela del bucle de búsqueda e introduce un esquema de sincronización eficientes y un sistema para el balanceo de carga. Los diferentes esquemas de sincronización propuestos para este esquema están basados en iteraciones de búsqueda, en intervalos de tiempo o en la combinación de ambos. Ambas propuestas, de grano fino y de grano grueso, se han implementado utilizando las herramientas OpenMP y MPI, es decir, usando memoria compartida y memoria distribuida, en un intento de saber cuál de las dos herramientas proporciona mejores resultados en cada caso.

Debido a la distribución irregular de la carga de trabajo, los resultados obtenidos con la aproximación de grano fino no son suficientemente buenos. Además, el trabajo que se puede paralelizar en una iteración de cómputo con esta aproximación está muy limitado. No obstante, a pesar de la estructura computacional tan irregular y de las dificultades de romper la naturaleza secuencial de las búsquedas primero-el-mejor, la combinación de la metodología de sincronización con el uso de estrategias de balanceo de carga proporciona una distribución del trabajo justa y una aceleración lineal para el algoritmo de grano grueso. Por otra parte, la comparativa entre las implementaciones en OpenMP y MPI del algoritmo de grano grueso muestra que, en general, las ejecuciones con OpenMP involucran un tiempo de búsqueda menor, mejorando el comportamiento obtenido con la implementación MPI. Sólo cuando el problema es simple y el número de hilos se incrementa, la aproximación OpenMP no es capaz de mejorar los resultados obtenidos con MPI.

De cualquier modo, se ha demostrado que incluso para los algoritmos con una estructura secuencial inherente es posible diseñar un esquema paralelo adecuado capaz de mejorar la eficiencia de la aproximación. Es importante destacar que la

eficiencia de los esquemas paralelos depende principalmente del grano de trabajo y de la distribución de la carga, aunque la selección de una herramienta de programación paralela adecuada también es decisiva.

Al margen de esto, en una visión más realista de este problema, se deberían tener en cuenta otros criterios a la hora de optimizar. En las industrias reales normalmente se presentan otros objetivos, que pueden ser secundarios o tener igual prevalencia que el objetivo más común de maximizar el beneficio. Por esta razón, se ha realizado una propuesta multi-objetivo que intenta maximizar el beneficio y minimizar el número de cortes necesarios para obtener las piezas. Si se minimiza el número de cortes se tarda menos en obtener las piezas, la maquinaria sufre menos y en consecuencia el resultado es más rápido y con menos costes asociados al mantenimiento de las máquinas o al consumo eléctrico. Existen muchas aproximaciones en la literatura relacionada para resolver el problema mono-objetivo, sin embargo, se desconocen propuestas multi-objetivo que traten con cortes de tipo guillotina.

Aplicar un método exacto para la resolución de un problema multi-objetivo es inviable debido a la magnitud y complejidad del árbol de búsqueda. Además, si se optara por abordar un problema multi-objetivo tratando de unificar los distintos objetivos en una única función objetivo, se deberían conocer a priori las demandas específicas en cada momento en cuanto a los dos objetivos, para poder combinarlos de la mejor manera. Si las demandas cambian, se necesitarían nuevas ejecuciones. Usando estrategias que afronten el problema multi-objetivo como tal, sólo se requiere una ejecución, y después la persona encargada de tomar las decisiones puede seleccionar la solución más adecuada, dependiendo de las demandas en cada momento. Si entre las metaheurísticas para optimización multi-objetivo se decide utilizar MOEAs se puede hacer uso de METCO, un entorno paralelo basado en plugins para la optimización multi-objetivo. Su objetivo es minimizar el esfuerzo requerido a los usuarios finales para incorporar sus propios problemas y algoritmos evolutivos. Además, cuenta con una selección de los MOEAs más conocidos en la literatura.

Utilizando METCO para resolver el 2DCSP multi-objetivo, simplemente hay que emplear un esquema general del algoritmo seleccionado y definir las características básicas del problema: cómo se representa el individuo, cómo se genera la población inicial, cómo se evalúan los objetivos, y qué operadores evolutivos se utilizarán. Por lo tanto, en este caso, se ha optado por la utilización de MOEAs a través de METCO para resolver el problema multi-objetivo. Así se han probado algunos de los MOEAs más conocidos, centrando los estudios en algunos de los que mejores resultados han presentado. Para la representación de los individuos de la población utilizada por esos MOEAs se han implementado tres esquemas de codificación, tratando de encontrar el más adecuado: dos codificaciones directas basadas en una notación postfija (*tamaño controlado* y *tamaño completo*) y una codificación basada

en una hiperheurística. La diferencia entre los esquemas directos es la manera en la que se controla la obtención de soluciones válidas, pero ambas usan la misma notación para representar las soluciones. Sin embargo, para la codificación basada en hiperheurística ha sido necesario implementar algunas heurísticas de colocación que se combinen entre sí para dar lugar a la hiperheurística. En la literatura hay muy pocas referencias de heurísticas específicas que resuelvan el 2DCSP de tipo guillotina. Sin embargo, existe un conjunto de heurísticas constructivas, orientadas a niveles, para la colocación de piezas, que se pueden aplicar fácilmente al 2DCSP aquí tratado debido a la manera en la que operan. Algunas de estas heurísticas (NFDH, FFDH y BFDH) se han implementado en este trabajo, ya que pueden aportar algo diferente aplicadas conjuntamente. Además, en un paso posterior, se ha agregado otra heurística implementada previamente para obtener la cota inferior del algoritmo exacto. Ésta no sólo tiene en cuenta el tamaño de las piezas, sino el beneficio asociado a ellas, ya que no necesariamente éste debe ser proporcional a su área.

Para ver las diferencias entre las distintas codificaciones usadas, se ha analizado su comportamiento dentro de las aproximaciones multi-objetivo. En primer lugar, si se comparan ambos esquemas de codificación directa se obtiene que uno de ellos se comporta mejor con las instancias complejas del problema y el otro con las instancias de menor número de piezas. En cambio los resultados demuestran que con el esquema de codificación basado en hiperheurística, aunque se obtienen soluciones competitivas si se compara con las heurísticas individuales, no se alcanza el comportamiento proporcionado con los esquemas de codificación directa. Esto puede deberse a que la hiperheurística está basada en heurísticas mono-objetivo que no proporcionan resultados de gran calidad para este problema, ya que originalmente fueron diseñadas para tratar con problemas de corte orientados a niveles. Así, se ha podido comprobar que al añadir al esquema la nueva heurística, implementada previamente para obtener la cota inferior del algoritmo exacto, que tiene en cuenta los beneficios asociados a cada una de las piezas, el comportamiento de la hiperheurística mejora notablemente.

De cualquier modo, con todas las codificaciones se ha mostrado la efectividad de aplicar MOEAs a este tipo de problemas. Aunque las propuestas no alcanzan el valor de la solución de máximo beneficio, obtenida aplicando el método exacto propuesto, las soluciones se encuentran bastante cercanas y proporcionan un buen compromiso entre ambos objetivos. Por lo tanto, de acuerdo con los dos criterios de optimización se han diseñado tres propuestas. Estas propuestas proporcionan en una sola ejecución un conjunto de soluciones de compromiso entre los dos objetivos, de entre las cuales los clientes podrán elegir de acuerdo a sus necesidades. Además, se han obtenido soluciones con un número muy bajo de cortes. En la mayoría de los casos el número de cortes se reduce al 40 % o 50 % de los obtenidos con la

solución mono-objetivo. Además, hay que tener en cuenta que la implementación de un algoritmo exacto lleva asociada una gran dificultad y coste, y por si esto fuera poco, el algoritmo exacto se centra en un solo objetivo sin considerar los posibles efectos negativos en otras características de las soluciones.

En definitiva, con la propuesta mono-objetivo exacta se ha conseguido mejorar el tiempo necesario para obtener las soluciones con respecto a los trabajos de referencia, y con sus paralelizaciones han reducido aún más esos tiempos. Además, mediante MOEAs se ha tenido en cuenta una visión multi-objetivo del problema no tratada anteriormente en la literatura, que ha proporcionado buenos resultados si se comparan con los resultados exactos mono-objetivo.

El segundo problema tratado ha sido el *Two-Dimensional Strip Packing Problem*. Este problema consiste en distribuir un conjunto completo de piezas sobre un material que, a efectos prácticos, se considera de largo infinito, de tal manera que se minimice el largo del material necesario. Igual que en el problema anterior se deben poder utilizar cortes de guillotina para obtener las piezas, puesto que realizar cortes no guillotinables puede implicar un funcionamiento más complejo de la maquinaria. Por esta razón, y para abarcar un rango más amplio de máquinas industriales, el trabajo se ha centrado en la formulación guillotinable del problema. Además, para este problema se permite la rotación de las piezas cuando se colocan en la materia prima, introduciendo algo más de complejidad en el mismo.

Siguiendo el mismo esquema que para el 2DCSP, para resolver este problema se ha desarrollado en primer lugar un algoritmo exacto. En la literatura no se conocen algoritmos exactos que aborden este problema con cortes de guillotina y permitiendo la rotación de las piezas. No obstante, cuando el número de piezas aumenta el tiempo puede incrementarse demasiado, por lo que para instancias con un número de piezas elevado es aconsejable llevar a cabo alguna paralelización del algoritmo. Igual que para el 2DCSP, en este caso se han utilizado dos paralelizaciones distintas del algoritmo exacto: un algoritmo de grano fino y otro de grano grueso. Ambos algoritmos se han implementado haciendo uso de las herramientas OpenMP y MPI. Los resultados obtenidos usando algunas de las instancias del problema disponibles en la literatura indican que la mejor opción para obtener la solución exacta en menos tiempo que la versión secuencial es usar las implementaciones de grano grueso. Los implementaciones de grano fino no presentan un buen comportamiento, dado que, igual que ocurre con el 2DCSP, el trabajo que se puede paralelizar en cada computación está mucho más limitado que en el caso del grano grueso. En ocasiones, la aceleración con las implementaciones de grano grueso no es realmente buena o no es constante, pero esto se debe a que la estructura del árbol de búsqueda es altamente irregular, y además el esquema de balanceo de carga puede provocar un

retraso en la obtención de la solución. En definitiva, los resultados para el 2DCSP y 2DSPP exactos son hasta cierto punto similares, (peor comportamiento con grano fino, mejor comportamiento con grano grueso). Pero para el 2DSPP se produce una mayor irregularidad en la aceleración debido a que se trata de instancias de mayor complejidad al tener en cuenta la rotación de las piezas. Además se han de colocar todas las piezas, por lo que las heurísticas utilizadas como cotas en el algoritmo inicial para el 2DCSP no son de gran ayuda. Es decir, esas heurísticas ayudan a escoger las piezas que mejores soluciones pueden dar, en función de la disponibilidad y beneficio asociado a cada pieza, pero para el 2DSPP, como hay que colocar todas las piezas en el material no se guía de igual manera la búsqueda y hay que probar entre todas las posibles combinaciones. Tampoco se han utilizado las mismas máquinas para las paralelizaciones, con lo cual también puede estar influyendo la arquitectura particular de cada una.

Por otra parte, igual que para el 2DCSP, en una visión más realista del 2DSPP se deberían tener en cuenta otros criterios a la hora de optimizar. Por esta razón, se ha realizado una propuesta multi-objetivo considerando los dos objetivos siguientes: minimizar el largo total del material a usar y minimizar el número total de cortes necesarios para obtener el conjunto completo de piezas demandadas. Esta formulación es muy adecuada para los problemas reales que aparecen en las industrias de producción y manufactura. La consideración de ambos objetivos puede implicar importantes ahorros para las fábricas, puesto que, si además de tratar de minimizar el largo del material que se usa, se minimiza el número de cortes necesarios, la maquinaria sufrirá menos deterioro, por lo que no se tendrán que hacer tantas inversiones en mantenimiento, y se ahorrará en consumo eléctrico y en mano de obra. Además, cuanto menor sea el número de cortes la productividad será mayor.

Tal y como ocurre con el 2DCSP, resulta inviable aplicar un método exacto para resolver este problema multi-objetivo debido a la magnitud del espacio de búsqueda. Además, si se optara por abordar este problema multi-objetivo tratando de unificar los distintos objetivos en una única función objetivo se deberían conocer a priori las demandas específicas en cada momento en cuanto a los dos objetivos, para poder combinarlos de la mejor manera. Si las demandas cambiasen, se necesitarían nuevas ejecuciones. Usando estrategias que afronten el problema multi-objetivo como tal, sólo se requiere una ejecución, y después la persona encargada de tomar las decisiones puede seleccionar la solución más adecuada, dependiendo de las demandas en cada momento. Esta disponibilidad de una diversidad de soluciones es el beneficio más importante de las propuestas multi-objetivo que se han hecho. Existe abundante bibliografía para la formulación mono-objetivo del problema, sin embargo, el número de aproximaciones multi-objetivo es muy reducido.

Los algoritmos evolutivos se han usado con éxito en muchos MOPs del mundo

real, y en concreto en el 2DCSP tratado anteriormente. Además, también ha proporcionado buenos resultados en muchas aproximaciones mono-objetivo del 2DSPP. Por todo ello, estos algoritmos se han elegido para abordar el 2DSPP multi-objetivo, usándose algunos de los MOEAs más extendidos en la literatura. En lugar de desarrollarlos desde cero, se ha hecho uso de METCO, igual que en el caso del 2DCSP. Tras un estudio inicial con varios MOEAs de los más conocidos, se ha seleccionado el NSGA-II para llevar a cabo el estudio en profundidad, puesto que ha mostrado un buen comportamiento para este problema.

Inicialmente, se ha planteado una implementación de los individuos basada en la representación directa de patrones de corte. Esta representación es similar a las codificaciones directas usadas para el 2DCSP. En este caso no existe la posibilidad de proponer dos codificaciones directas (tamaño fijo o controlado), puesto que para este problema es necesario colocar todas las piezas, con lo cual el cromosoma será siempre de tamaño fijo. Además, el individuo incorpora información para indicar las orientaciones de las piezas, ya que a diferencia del 2DCSP, para este problema las rotaciones están permitidas.

Los resultados obtenidos usando esta representación, mejoran claramente los únicos resultados conocidos para el 2DSPP multi-objetivo y con cortes guillotinables. Como la representación del cromosoma que se ha usado permite ambos procesos de corte, guillotina y no-guillotina, el número de cortes puede variar dependiendo de las características de la maquinaria disponible. El estudio computacional demuestra que siempre que sea posible realizar cortes de tipo no-guillotina, es decir, cuando la maquinaria permita un proceso más sofisticado de corte, el número de cortes requerido para llevar a cabo el trabajo se puede ver reducido significativamente.

De cualquier modo, para las instancias más pequeñas del problema, la aproximación directa proporciona soluciones comparables a las obtenidas por aproximaciones que se centran en optimizar el largo total. Sin embargo, para las instancias de tamaño mayor, la codificación tiene que tratar con espacios de búsqueda muy grandes, por lo que no es capaz de producir soluciones competitivas en cuanto a largo total. Es necesario destacar que no se han podido realizar comparativas con las soluciones exactas mono-objetivo como en el caso del 2DCSP puesto que se desconocen para las instancias usadas. Es decir, en la literatura relacionada se indica que el largo óptimo para el grupo de instancias empleadas es de valor 100, pero no se dispone de las soluciones explícitas que proporcionan ese valor, para poder utilizarlas en la evaluación del número de cortes que implican dichas soluciones. Además, el algoritmo exacto desarrollado, incluso con paralelizaciones, no es capaz de aportar soluciones debido a la complejidad de las instancias utilizadas para el análisis multi-objetivo.

En un intento de mejorar los resultados dados por la aproximación multi-objetivo con la codificación directa, se han obtenido resultados para distintos modelos pa-

ralelos basados en islas. Así se ha podido comprobar que el rendimiento de las aproximaciones evolutivas se puede incrementar un poco gracias a algunos de los modelos paralelos basados en islas proporcionados por el entorno METCO. Aún así la calidad de las soluciones sigue sin incrementarse lo suficiente para alcanzar los valores deseados.

Por lo tanto, con el objetivo de mejorar los resultados para las instancias de mayor tamaño, se han propuesto cuatro nuevos tipos de codificación basados en la combinación de diversas heurísticas de bajo nivel diseñadas para optimizar sólo el objetivo del largo del material. La razón por la que se han propuesto cuatro esquemas, es ver en qué medida influye el tipo de codificación en la calidad de las soluciones obtenidas, y así de paso poder encontrar la codificación hiperheurística con mejor comportamiento para este problema En la literatura se puede encontrar una enorme cantidad de heurísticas para el 2DSPP de tipo no-guillotina, pero el número de propuestas para el caso guillotinable no es demasiado extenso. De las heurísticas existentes se han implementado cuatro de las más conocidas (la mayoría ya usadas anteriormente para resolver el 2DCSP): NFDH, FFDH, BFDH y BFDH*. Cada esquema de codificación se centra en un espacio de búsqueda diferente, por lo que los resultados se encuentran en distintas áreas del espacio de soluciones. Los resultados computacionales demuestran que dos de los cuatro esquemas de codificación hiperheurística son capaces de proporcionar soluciones competitivas comparadas con las heurísticas mono-objetivo individuales. Incluso, una de las hiperheurísticas, **(h, p, o, r)**, da soluciones que dominan completamente a las obtenidas usando las heurísticas mono-objetivo individuales. Esta codificación puede buscar en regiones donde la optimización de ambos objetivos está mejor equilibrada. Además, coincide con el esquema hiperheurístico utilizado para el 2DCSP, con la única diferencia de que para este problema se necesita la información de la rotación de las piezas. Sin embargo, este esquema hiperheurístico proporciona un mejor comportamiento, ya que mejora los resultados dados por la codificación directa para problemas grandes, mientras que en el 2DCSP se queda siempre por detrás de las codificaciones directas. Esto puede deberse a que las heurísticas constructivas mono-objetivo son más adecuadas para este problema, puesto que trabajan por niveles, y como cada nivel afecta al largo del material, al optimizarlos tiene mayor influencia en el primer objetivo de este problema: minimizar el largo del material utilizado.

De cualquier modo, para el 2DSPP se tiene un esquema de codificación directa que no puede mejorar el objetivo del largo para instancias del problema de gran tamaño, comparado con las aproximaciones mono-objetivo, pero proporciona valores bajos del número de cortes. En contraposición, se tiene un esquema de codificación hiperheurística que mejora ambos objetivos, comparado con las heurísticas de bajo nivel, pero proporciona números de cortes mayores que los que aporta el esquema

de codificación directa. Usando diferentes esquemas de codificación y mediante la aplicación de métodos basados en hiperheurísticas, se ha conseguido cubrir una gran zona del espacio de soluciones, permitiendo que la persona encargada de tomar las decisiones pueda elegir el tipo de codificación dependiendo del área del espacio de soluciones que sea más interesante para el problema dado. Tal y como se ha demostrado, el diseño de esquemas de codificación para la representación de soluciones en un MOEA es una cuestión muy importante. La definición de espacios de búsqueda grandes dificulta la obtención de soluciones de alta calidad, pero cuando los espacios de búsqueda se acotan, algunas regiones del espacio de soluciones pueden quedar sin explorarse. Por lo tanto, es importante diseñar codificaciones que permitan cubrir regiones del espacio de búsqueda que ofrezcan un mejor equilibrio entre los dos objetivos de optimización.

En definitiva, para el 2DSPP, dado que no se han encontrado en la literatura algoritmos exactos que tengan en cuenta la rotación de las piezas, se ha propuesto también un algoritmo exacto que igualmente incluye la detección de patrones dominados y duplicados y paralelizaciones. Se ha reutilizado y adaptado el código desarrollado para el 2DCSP de manera que resultara útil para este problema. En cuanto a la versión multi-objetivo, se han conseguido mejorar las únicos resultados que se conocen de un trabajo que lo aborda como tal, y además se han propuestos varios esquemas de codificación que cubren diferentes áreas del espacio de soluciones.

Finalmente, se ha abordado la resolución del *Container Loading Problem*. Este problema consiste en la distribución de un conjunto de cajas dentro de un contenedor de dimensiones fijas, de tal manera que se maximice el volumen utilizado del mismo. En general, estas cajas tendrán una serie de orientaciones permitidas y sin embargo, otras no serán posibles debido, por ejemplo, al contenido de las mismas, que puede ser frágil o simplemente ser un líquido.

En cuanto a métodos de resolución, solamente se conoce la referencia de un trabajo que aborda la solución exacta del problema mono-objetivo. La complejidad de este problema es superior incluso a la de los problemas de corte anteriormente tratados, ya que no solo se contempla un mayor número de ubicaciones de las cajas, sino también más orientaciones que en el caso de dos dimensiones. Como consecuencia, con un método exacto sólo se podrán abordar instancias muy pequeñas del mismo en un tiempo razonable. Además, este problema en la realidad se presenta frecuentemente como multi-objetivo. En base a la aplicación particular, puede presentar diferentes restricciones, como la orientación de las piezas, la distribución de la carga, la estabilidad, las prioridades de introducción de las piezas, el límite de peso de los contenedores, el tipo de colocación de las piezas, etc. Por lo tanto, se ha abordado la resolución de una versión multi-objetivo del CLP tratando de maximi-

zar el volumen total a empaquetar y el peso total empaquetado. Se ha escogido este segundo objetivo dado que es bastante común que los vehículos que transportan los contenedores, tengan asociado un peso máximo que pueden alcanzar. Como dichos vehículos suelen ser alquilados para transportar el cargamento y se pagan de acuerdo al peso total que puede transportar con independencia del volumen total, sería conveniente transportar un cargamento con peso total elevado, respetando siempre el límite o tara del vehículo.

En la literatura sólo se conoce la referencia de un trabajo que borda este problema con múltiples objetivos, y precisamente tiene en cuenta estos dos mismos objetivos. Para ello utiliza un algoritmo de recocido simulado hibridizado con una heurística de relleno. Éste requiere de múltiples ejecuciones aplicando diferentes pesos a cada objetivo, para obtener el conjunto de soluciones. Sin embargo, tal y como se ha comprobado con los problemas de corte anteriormente tratados, los MOEAs permiten obtener un conjunto de soluciones en una sola ejecución, y además han proporcionado resultados prometedores. Por todo ello, este trabajo ha tratado de mejorar las soluciones conocidas utilizando MOEAs.

Después de un estudio inicial en el que se han probado varios MOEAs, se ha seleccionado el NSGA-II debido al buen comportamiento presentado para este problema. En este caso no se ha utilizado una codificación directa, ya que el espacio de búsqueda con una notación postfija sería demasiado grande, al disponer de más operadores y orientaciones posibles que en el caso de dos dimensiones. Por consiguiente, se ha empleado una solución codificada que indica la secuencia de colocación y orientación de las cajas que una heurística utilizará para ubicarlas en el contenedor. Los resultados obtenidos demuestran el buen comportamiento de la estrategia evolutiva ante este problema, obteniéndose mejores valores que los conocidos.

En un intento de comprobar si utilizando las paralelizaciones que ofrece METCO se obtienen resultados de mayor calidad en el mismo o en menor tiempo, se ha comprobado que, con todas las paralelizaciones homogéneas del algoritmo evolutivo, se obtienen soluciones de igual calidad en menor número de evaluaciones y, por tanto, en menor tiempo. Algunas paralelizaciones son capaces incluso de mejorar la calidad de las soluciones secuenciales, como es el caso de la ejecución con dos islas. Sin embargo, el modelo paralelo de ocho islas no consigue alcanzar en el mismo número de evaluaciones la calidad final alcanzada por la propuesta secuencial, dado que al aumentar el número de islas el número evaluaciones se reparte entre más procesadores, de modo que cada isla puede avanzar menos en cada exploración. En definitiva, para resolver el CLP se ha proporcionado una aproximación multiobjetivo que mejora los resultados presentes en el único trabajo de la bibliografía que tiene en cuenta más de un objetivo.

Por lo tanto, como conclusión, en este trabajo se ha abordado la resolución de tres variantes de los problemas de corte y empaquetado: el 2DCSP, el 2DSPP, y el CLP. Como demuestra la literatura asociada, existen muchas propuestas de resolución para los problemas seleccionados, aunque parece que las heurísticas y/o metaheurísticas son los métodos más utilizados, puesto que los exactos no son demasiado viables para este tipo de problemas tan complejos. Sin embargo, en este estudio se analizan algunas propuestas de resolución exactas, como mecanismo para conocer las soluciones óptimas y poder utilizarlas posteriormente para realizar una comparativa con los métodos aproximados. Dado que los algoritmos exactos tienen un coste computacional demasiado alto, ha sido necesario diseñar optimizaciones específicas para reducir los espacios de búsqueda, así como implementaciones paralelas para poder reducir el tiempo invertido en obtener las soluciones a los problemas.

No obstante, cuando las instancias a resolver son más complejas, los métodos exactos dejan de ser viables y es necesario utilizar algún tipo de método aproximado. Además, cuando se quiere analizar los problemas desde un punto de vista más real, en el que también se tengan en cuenta otros objetivos primordiales para optimizar los procesos productivos en las industrias de corte y empaquetado, es necesario afrontar la resolución de estos problemas desde un punto de vista multi-objetivo. Esto implica que, como solución al problema no se obtendrá una solución única, sino un conjunto de soluciones que optimizan de distinta forma los diversos objetivos del problema. En este caso, se ha demostrado que si se utilizan métodos específicos para resolver problemas multi-objetivo, como son los MOEAs, se puede obtener un conjunto de soluciones que proporcionan un compromiso entre los distintos objetivos a optimizar. De esta forma, dependiendo del caso concreto, el usuario podrá seleccionar la solución que más le convenga en función del interés que tenga en optimizar más un objetivo u otro en el momento específico.

Al contrario de lo ocurrido con los algoritmos exactos, el esquema general de este tipo de algoritmos evolutivos no se ha diseñado específicamente para los problemas aquí analizados. Sin embargo, ha sido necesario definir algunos aspectos que sí dependen del problema en concreto y que, como se ha visto, pueden afectar considerablemente en la diversidad y la calidad de las soluciones que se obtengan. Es el caso de la representación escogida para las soluciones, y de los operadores de mutación y de cruce. Existen diversas posibilidades de codificación de las soluciones y de operadores asociados, y dependiendo de lo que se escoja los resultados pueden ser más o menos adecuados a las instancias que se necesitan resolver. Por lo tanto, la definición de la codificación de las soluciones y los operadores asociados, a la hora de utilizar MOEAs son un aspecto tan importante como el propio método de resolución.

Por otra parte, los algoritmos paralelos para este tipo de aproximaciones multi-

objetivo permiten explorar en paralelo el espacio de búsqueda obteniendo las soluciones en un tiempo menor. La diferencia es que, en este caso, la evolución de distintas poblaciones que pueden cooperar y/o interactuar entre sí permite mejorar la calidad de los resultados que se obtienen con las ejecuciones secuenciales.

Por último, se ha comprobado la importancia de la utilización de esqueletos, frameworks o herramientas como METCO, que proporcionen, de forma general, las características globales e internas de los algoritmos, dejando que el usuario se centre únicamente en las cuestiones más específicas de los problemas. Gracias a este tipo de herramientas y al intento de crear representaciones de soluciones generales para los problemas de corte y empaquetado tratados, se han podido extender muchos de los avances, pruebas y estudios realizados para un problema, a los demás problemas estudiados. En cualquier caso, siempre se ha tratado de proponer esquemas generales de resolución de manera que, con pequeñas modificaciones, los métodos se puedan adaptar a diferentes problemas de corte y empaquetado. Para algunos problemas se han utilizado nuevos métodos de resolución que no han sido encontrados aplicados a estos problemas en la literatura asociada. En otros, se ha conseguido mejorar los resultados existentes en la bibliografía agregando mejoras a los métodos aplicados.

6.2. Trabajo futuro

Tal y como ocurre en la mayoría de investigaciones o estudios, en este trabajo han quedado pequeñas ventanas abiertas hacia nuevos focos de interés que se podrían analizar. Así, como trabajo futuro con respecto al 2DCSP, sería interesante diseñar otras heurísticas propias del problema para agregarlas al esquema hiperheurístico que se utiliza en la versión multi-objetivo del mismo. De este modo se podría comprobar si, tal y como ocurrió al añadir la última heurística al esquema, éste mejora su comportamiento y consigue alcanzar o mejorar los resultados obtenidos con los esquemas que utilizan una notación postfija. Una vez se haya fijado el esquema hiperheurístico que presente mejor comportamiento, tanto él como los esquemas de codificación directa que ya han sido analizados en este trabajo, se podrían utilizar para realizar un estudio de los modelos paralelos basados en islas. Para ello se podrían emplear otros MOEAs diferentes, que aunque en comparación con el NSGA-II presentaban un peor comportamiento, quizás en un modelo paralelo heterogéneo puedan aportar alguna mejora.

En el caso del 2DSPP los esquemas hiperheurísticos ya han sido analizados con detalle, y las heurísticas utilizadas en ellos han demostrado ser adecuadas para el problema. Sin embargo, sí se podrían obtener mejoras, no solo con estos esquemas sino también con el directo, diseñando y probando otros operadores de mutación y de

cruce que permitan aumentar la eficiencia. Ya han sido probadas algunas alternativas pero se podría hacer un estudio en profundidad. Una vez que se tengan los esquemas ya refinados, se podrían introducir más MOEAs en los modelos paralelos basados en islas que se han empleado en el trabajo presentado. Además, los MOEAs utilizados se podrían hibridizar con la introducción de búsquedas locales que intenten expandir mejor el espacio de búsqueda hacia ciertas regiones del espacio de soluciones. Como alternativa, la incorporación de algún tomador de decisiones durante la ejecución de los MOEAs podría permitir descartar algunas áreas del espacio de soluciones que no nos interesen, mientras se mantiene la diversidad de soluciones obtenidas en regiones más adecuadas.

En cuanto al CLP, a diferencia de los problemas anteriores, no se ha utilizado una representación directa de los individuos puesto que se ha considerado poco adecuada. Sin embargo, tampoco se ha utilizado una representación basada en hiperheurística, y ésta sí podría resultar factible. Por lo tanto, como siguiente paso sería necesario el diseño de las diferentes heurísticas que formen parte de dicha hiperheurística. Una vez que se haya definido el esquema, habrá que desarrollar los operadores de mutación y cruce que se vayan a utilizar. Entonces, sería interesante realizar un estudio en profundidad sobre dichos operadores, intentando encontrar los más adecuados. Asimismo, para el esquema de codificación que se ha utilizado en este trabajo, se podría llevar a cabo un estudio similar de otros operadores evolutivos. Cuando se encuentren los operadores más adecuados, se podrán realizar pruebas con otros MOEAs además del NSGA-II que han mostrado buen comportamiento en secuencial, tal y como se ha planteado para los anteriores problemas. De este modo, existiría la posibilidad de ejecutar para este problema diferentes modelos paralelos heterogéneos, donde cada isla ejecute un algoritmo distinto. Es necesario destacar que así no solo se pretende ahorrar tiempo a través de la distribución del esfuerzo computacional, sino obtener los beneficios algorítmicos derivados de la cooperación de varias poblaciones. Finalmente, dado que para este problema no se conocen más instancias que la utilizada (que tengan en cuenta el peso de las cajas y del contenedor), se podría tratar de adaptar las instancias usadas por los algoritmos mono-objetivo que hay en la literatura. Así sería posible realizar una comparativa entre los resultados obtenidos con algoritmos mono-objetivo y con los multi-objetivo, y ver la calidad de unos frente a otros.

En definitiva, como aspecto general para todos los problemas tratados, el refinamiento de los esquemas de codificación, y el estudio en profundidad de otros operadores de mutación y cruce en las aproximaciones multi-objetivo es un trabajo que ha quedado pendiente de realizar. Asimismo, la aplicación de nuevos MOEAs para estas propuestas multi-objetivo supondría otra fuente de trabajo importante. Por último, así como en general las codificaciones de los individuos han sido bastante

estudiadas para la versión multi-objetivo de los problemas tratados, el análisis de las paralelizaciones heterogéneas basadas en islas para estas aproximaciones no se ha realizado por completo, por lo que éste es un posible estudio futuro. Al margen de las paralelizaciones heterogéneas donde cada isla ejecuta un MOEA diferente, se podrían llevar a cabo paralelizaciones heterogéneas donde cada isla utilice una codificación diferente de los individuos.

Además de estos trabajos futuros que suponen mejoras para cada una de las propuestas presentadas, dado que se han intentando utilizar esquemas lo más genéricos posible, se podría tratar de resolver nuevos problemas de corte y empaquetado siguiendo las pautas llevadas a cabo en este trabajo.

Por otra parte, sería muy interesante conseguir contactar con empresas reales del sector para ofrecer soluciones a sus problemas y conocer de primera mano las restricciones más comunes que se pueden presentar a la hora de resolverlos, de manera que se puedan ajustar las metodologías a las necesidades reales de las fábricas.

Un caso particular se podría llegar a presentar para los problemas de corte como el 2DCSP y el 2DSPP, si las máquinas industriales capaces de realizar cortes de tipo no guillotina tienen tendencia a ir en crecimiento. En esta situación sería recomendable centrarnos en estos problemas sin considerar sólo construcciones de tipo guillotinable. Esto abriría todo un campo de investigación diferente, ya que se necesitarían representaciones diferentes para los individuos, otros operadores genéticos, otras heurísticas, etc. Habría que explorar de nuevo todos los métodos de resolución.

Parte IV

Apéndices

Apéndice

A Publicaciones y conferencias

Este apéndice presenta las publicaciones que se han producido como resultado de la investigación llevada a cabo durante el curso de esta tesis. Estas publicaciones incluyen artículos en revistas internacionales de relevancia en el área de investigación, contribuciones a conferencias internacionales con comités de revisión para asegurar la calidad y validez de los trabajos seleccionados, así como contribuciones en conferencias nacionales. Finalmente, se expone la participación en otras conferencias importantes en el ámbito del corte y empaquetado, con publicación en libros de resúmenes.

Revistas Internacionales

[1] J. de Armas, C. León, G. Miranda, and C. Segura. Optimisation of a multi-objective two-dimensional strip packing problem based on evolutionary algorithms. *International Journal of Production Research*, 48(7):2011– 2028, 2009. ISSN 0020-7543 print / 1366-588X online.

[2] J. de Armas, C. León, G. Miranda, and C. Segura. Remote service to solve the two-dimensional cutting stock problem: an application to the Canary Islands costume. *International Journal of Grid and Web Services*, 4:342–351, December 2008. ISSN 1741-1114 print / 1741-110 online.

[3] J. de Armas, G. Miranda, and C. León. Improving the Efficiency of a Best-First Bottom-Up Approach for the Constrained 2D Cutting Problem. *European Journal of Operational Research*, 219(2):201–213, 2012. ISSN 0377-2217.

Actas de Conferencias Internacionales

[1] J. de Armas, , Y. González, G. Miranda, and C. León. Parallelization of the multi-objective container loading problem. In *IEEE Congress on Evolutionary Computation (CEC'2012)*, page to appear. IEEE Computer Society, Brisbane, Australia, June 2012.

[2] J. de Armas, C. León, and G. Miranda. Fine and Coarse-grained Parallel Algorithms for the 2D Cutting Stock Problem. In T. G. Marco Danelutto, Julien Bourgeois, editor, *The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP'2010)*, pages 255–262. IEEE Computer Society, Pisa, Italia, February 2010. ISBN 978-0-7695-3939-3.

[3] J. de Armas, C. León, G. Miranda, and C. Segura. Optimization of a Two-Dimensional Cutting Problem through Parallel Cooperation of Multi-Objective Evolutionary Algorithms. In *9th International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA'2008)*, volume 6126, 6127, pages to appear, Spring 2012. Lecture Notes in Computer Science, Trondheim, Norway, May 2008.

[4] J. de Armas, C. León, G. Miranda, and C. Segura. Remote Service to solve the Two-Dimensional Cutting Stock. In *2nd International Conference on Complex, Intelligent and Software Intensive Systems*, pages 971–976. IEEE Computer Society, Barcelona, España, March 2008. ISBN 0-7695-3109-1.

[5] J. de Armas, G. Miranda, and C. León. Hyperheuristic encoding scheme for multi-objective guillotine cutting problems. In *Genetic and Evolutionary Computation Conference (GECCO'2011)*, pages 1683–1690. ACM, Dublin, Ireland, July 2011. ISBN 978-1-4503-0557-0.

[6] J. de Armas, G. Miranda, and C. León. A multi-objective approach for the 2d guillotine cutting stock problem. In *International Work-conference on Artificial and Natural Neural Networks (IWANN'2011)*, pages 292–299. Springer- Verlag on Lecture Notes in Computer Science, Torremolinos, Málaga, June 2011. ISBN 978-3-642-21497-4.

[7] J. de Armas, G. Miranda, and C. León. Two encoding schemes for a multi-objective cutting stock problem. In *IEEE Congress on Evolutionary Computation (CEC'2011)*, pages 27–109. IEEE Computer Society, New Orleans, LA, June 2011. ISBN 978-1-4244-7834-7.

[8] J. de Armas, G. Miranda, and C. León. Hyperheuristic Codification for the Multi-Objective 2D Guillotine Strip Packing Problem. In *IEEE Congress on Evolutionary Computation (CEC'2010)*, pages 2883–2890. IEEE Computer Society, July 2010. ISBN 978-1-4244-6909-3.

Actas de Conferencias Nacionales

[1] J. de Armas, Y. González, G. Miranda, and C. León. El Problema de Carga de Contenedores: una Aproximación Paralela y Multi-objectivo. In *VIII Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'2012)*, pages 399–406. IEEE Computer Society, Albacete, Spain, February 2012. ISBN 978-84-615-6931-1.

[2] J. de Armas, C. León, G. Miranda, and C. Segura. Interfaz gráfica para el corte del Traje de Mago de La Orotava. In *XVII Congreso Español en Informática Gráfica*, pages 275–278. Thomson, Zaragoza, Spain, September 2007. ISBN 978-84-9732-595-0.

[3] J. de Armas, C. León, G. Miranda, and C. Segura. Interfaz gráfica de usuario para el problema de corte bidimensional. Una aplicación al traje típico de las Islas Canarias. In *IX Congreso Internacional de Interacción Persona-Ordenador*, pages 423–432. Albacete, Spain, June 2008. ISBN 978-84-691-3871-7.

[4] J. de Armas, C. León, G. Miranda, and C. Segura. Optimización del Problema de Corte Bidimensional Multi-Objetivo. In *XIX Jornadas de Paralelismo*. Universitat Jaume I, Castellón, Spain, September 2008. ISBN 978-84-8021-676-0.

[5] J. de Armas, G. Miranda, and C. León. Resolución Evolutiva del problema multiobjetivo 2D Strip Packing: Análisis de las Codificaciones. In *VII Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'2010)*, pages 46–53. IEEE Computer Society, September 2010. ISBN 978-84-92812-58-5.

Conferencias sin Actas

[1] J. de Armas, C. León, and G. Miranda. Comparing Encoding Schemes for Multi-objective Two-dimensional Guillotine Cutting Problems. In *9th ESICUP Meeting*, Tenerife, Spain, March 2012. Book of abstracts.

[2] J. de Armas, C. León, and G. Miranda. Multi-objetive Solution for 2D Guillotine Cutting Stock Problem. In *8th ESICUP Meeting*, Copenhagen, Denmark, May 2011. Book of abstracts.

[3] J. de Armas, C. León, and G. Miranda. Codifications in Evolutionary Algorithms for the Multi-Objective 2D Guillotine Strip Packing Problem. In *7th ESICUP Meeting (EURO XXIV)*, Lisboa, Portugal, July 2010. Book of abstracts.

[4] J. de Armas, C. León, G. Miranda, and C. Segura. A Parallel Evolutionary Approach to solve a Multi-Objective Two-Dimensional Strip Packing Problem. In *6th ESICUP Meeting*, Valencia, Spain, March 2009. Book of abstracts.

Apéndice

B Summary and conclusions

This appendix presents a summary in English of the basic fundamentals and the main contributions of this thesis, paying particular attention to the conclusions reached. First, the basic concepts of cutting and packing problems and the different strategies and methodologies for troubleshooting are introduced. Then, the main objectives of this work, and a short analysis of the structure of this thesis, are detailed. Moreover, the contributions made for each of the problems and the approaches considered are briefly described. Finally, some conclusions and a few proposals for future lines of work are presented.

B.1. Introduction

Cutting and packing problems are a family of natural combinatorial optimization problems which have been widely studied - for over half a century [29, 107] - in numerous industrial and research areas. These problems arise in many production industries where large stock sheets of raw material must be cut into smaller pieces, or, in the case of three-dimensional packing problems, where the position of a set of volume elements to be transported or stored must be determined. The cutting or packing process creates patterns resulting in geometric combinations of small items assigned to large objects. Residual pieces, i.e. the remaining areas of the large objects which are not occupied by small items, are usually regarded as *trim loss*. Because of the dominant role played by the patterns and their nature as geometric combinations, it may be said that cutting and packing belong to the field of *geometric combinatorics*. In particular, cutting and packing problems are concerned with objects and items defined by one, two or all three spatial dimensions of the Euclidean space. In cutting (stock) problems, the large objects consist of solid materials that may be divided into smaller elements, which we refer to as pieces. Usual stock materials are paper and pulp, steel, glass, wood, plastics, leather and textiles. In the case of packing and loading problems, it may be said, in the narrow sense, that these are characterized by large objects defined as the empty, useful space of vehicles, cars, pallets, containers, bins, and so on. Packing small material items into these objects may also be considered as cutting the empty space of the large objects into parts of empty spaces, some of which are occupied by small items. In a similar way, cutting stock problems may be viewed as packing the space occupied by small items into the space occupied by large objects.

In the case of cutting problems, the small items are known as order list or demanded pieces, while the large objects are known as stock or raw material. The small items are requested by the clients, and so the industries or companies must produce the set of demanded items from the stock material available in order to satisfy the needs of their clients. When dealing with packing problems, the large objects are known as containers, while the small items are the elements that must be loaded into the containers, usually for transportation or storage purposes.

The goal of these problems is to select some or all of the small items, group them into one or more subsets and assign each of the resulting subsets to one of the large objects such that the geometric condition holds, i.e. the small items of each subset have to be laid out in the corresponding large object such that:

- all small items of the subset lie entirely within the large object,
- the small items do not overlap, and
- a given single or multi-dimensional objective function is optimized.

At this point, it should be noted that a solution to the problem may result in using some or all of the large objects, and some or all of the small items. Considering these possible alternatives, a set of different subproblems can be formally distinguished:

- selection problem involving the large objects,
- selection problem involving the small items,
- allocation problem involving the assignment of the subsets of the small items to the large objects, and finally,
- layout problem involving the arrangement of the small items on each of the selected large objects with respect to the geometric condition.

Of course, all of these problems have to be solved simultaneously in order to achieve a "global" optimum. Depending on the particular definition of these subproblems and the given constraint set, one is faced with a certain type of cutting and packing problem, and thus with a specific problem formulation. In any case, even supposing the simplest constraint set or problem formulation, cutting and packing problems are generally classified as NP-hard problems [61].

Many attempts at typologies and classifications have been proposed in an effort to homogenize the nomenclature used when referring to cutting and packing problems [61, 63, 97, 129, 159, 169, 182]. A typology helps to unify definitions and notations and, by doing so, lays the groundwork for practically oriented research. Moreover, if publications are categorized according to an existing typology, it will provide faster access to the relevant literature. In order to propose an appropriate typology which allows most existing problems in the field to be easily classified, it is necessary to analyze cutting and packing problems in depth so as to identify the main characteristics of the problem itself. Usually, these principal features are related in some way to the following topics:

- the set of large objects (input, supply),
- the set of small items (output, demand),
- the patterns as geometric combinations of small figures,
- the assignments of small items to patterns and of patterns to large objects.

In the area of cutting and packing, there is not a single global typology with complete international acceptance. The typology proposed by Dyckhoff in 1990 [63], however, has been widely used and followed. Later, a new and improved typology for cutting and packing problems [182] was proposed. This typology is based on Dyckhoff's original proposal, although it attempts to fix some underlying deficiencies. After a wide review of the existing typologies on cutting and packing, a set of the main properties which define a given problem specification was selected. These properties are:

- Dimensionality: 1D, 2D, 3D, o n-dimensional.
- Quantity measurement: integer or real.
- Shape of figures: regular or irregular.
- Assortment of small items: many items of a single type, several items of each type, or one or a few items of each type.
- Assortment of large items: a single item available or several.
- Availability of items: sequences or order.
- Type of assignment: output (value) maximization (a set of small items has to be assigned to a given set of large objects, and the set of large objects is not sufficient to accommodate all of the small items, so the aim is to assign a selection (subset) of the small items of maximal value.) or input (value) minimization (a given set of small items is to be assigned to a set of large objects, but in this case the set of large objects is sufficient to accommodate all of the small items, which are to be assigned to a selection (subset) of the large object(s) of minimal value).
- Pattern restrictions: types of cuts, distances between small figures or between cuts, etc.
- Objectives: number and type of optimization functions.

Combining the main type of dimensionality, assignment, and assortment involved yields too many types of cutting and packing problems. Analyzing every possible alternative one by one may not give a proper and self-explanatory classification. In order to provide a more general classification for standard problems, the basic properties were studied by going from general to specific ones. This way, basic types of cutting and packing problems were developed by combining the assignment and assortment criteria for small and large items.

Considering the *kind of assignment* as the global criterion, the basic types of problems can be divided into two main categories: "output maximization types" and "input minimization types", as proposed in [182]. Within each of these two categories, it is possible to divide the problems depending on the *assortment of the small items*. Again, each subproblem can be subdivided depending on the *assortment of the large objects*. In a final step, refined problem types are obtained by applying the *dimensionality* criterion and the *shape of the small items* criterion. These last resulting subcategories are characterized by adjectives which are added to the names of the intermediate problem types:

[1, 2, 3]-dimensional [rectangular, circular, ..., irregular] <problem_name>

Each of these cutting and packing problems can be solved attending to a single objective, or, by taking into account the more realistic multi-objective version of them. However, each type of problem optimization - single and multi-objective - requires a different solution method, suitable to its properties and available resources. That is, just as in general it is not feasible to apply an exact method to a multi-objective problem, there are single-objective problems that specifically require an exact solution. However, sometimes for reasons of time or resources, other methods that provide good quality, but perhaps not optimal, results are applied.

Exact methods give the global optimal solution; however, attaining accuracy usually means slow convergence. Some of the best known exact algorithms for solving cutting and packing problems are based on branch and bound algorithms, depth-first or best-first searches. In contrast to the exact methods that provide an optimal solution, heuristic methods are limited to providing a good quality solution that is not necessarily optimal. Usually, they are highly dependent on the specific problem for which they are designed. Conceptually, metaheuristics procedures are located *above* heuristics procedures, in the sense that they guide their design. A metaheuristic is a set of concepts that can be used to define heuristic methods applicable to a broad set of problems. Therefore, they are considered as higher level heuristics with a generic use. Some of the best known metaheuristic applied to cutting and packing problems are: local search, simulated annealing, tabu search, GRASP, evolutionary computation, and ant colony optimization. Evolutionary algorithms specifically designed to deal with multiple objectives functions are known as multi-objective optimization evolutionary algorithms (MOEAs). Finally, a hyperheuristic is a heuristic search method which tries to automate the process of combining low-level heuristics, usually by incorporating learning techniques, in such a way that a heuristic's strengths make up for the drawbacks of another.

Parallelism can be applied to all of these methods. The main reasons to use parallelism in software design are to obtain high performance or a higher speed when running a program. Introducing parallelizations into the algorithms that implement these methods yields a number of advantages over a sequential execution that depend on the purpose of the parallelization. The first purpose may be to obtain the same solution as the sequential algorithm in less time by dividing the work among different processors. This is common when dealing with exact methods because the entire search space must be explored to ensure finding the optimal solution, and as such they tend to be algorithms which require considerable time. However, when dealing with approximate methods, it is possible that the real goal of parallelize may be not to reduce the execution time, but to obtain a better solution in the same time employed by the sequential algorithm. In any case, an improvement over the sequential method can be obtained. Note that when using approximated so-

lution methods based on populations, the parallelization is usually implemented using the so-called *island-based models*. Four basic island-based models exist [44]: all islands execute identical MOEAs/parameters (homogeneous), all islands execute different MOEAs/parameters (heterogeneous), each island evaluates different objective function subsets, and each island represents a different region of the genotype or phenotype domains. Moreover, the heterogeneous model can add self-adaptive properties. These self-adaptive properties allow for the dynamic allocation of more computational resources to the most promising algorithms. That is, by applying a hyperheuristic, an automatic and dynamic change of MOEAs and/or parameters which are used in each of the islands is allowed.

In contrast, when we are dealing with a specific optimization problem, we want to obtain solutions using different solution methods and configurations in order to see which one is best suited to our needs, demands and available resources. In these cases, even if the user has a broad knowledge of the problem, he lacks detailed insight and has only a general knowledge of each of the methods for solving it. There are a number of specific tools that support different solution techniques and provide assistance to these users by reducing the effort involved in this process. Such tools can be libraries, environments, or algorithmic skeletons. This software provides a significant advantage compared to direct implementation from scratch, not only in terms of reusability, but also in terms of methodology and conceptual clarity. The greatest drawback with most of these tools is the complexity of use, mainly because of the programming language or internal logical structure. One of these tools used in this work was METCO, a parallel environment based on plugins for multi-objective optimization. Its objective is to minimize the effort required for end users to incorporate their own problems and evolutionary algorithms. Moreover, it also has a selection of the best known MOEAs in the literature.

B.2. Objectives

Obtaining the best of these solution methods requires knowing the optimization type (single or multi-objective), the size of the problem and the available computational resources, since the use of machines and parallel implementations can significantly reduce the solution time. In addition, the methods to be applied depend mainly on the specific nature of the problem and on the features expected from the solution.

In most of the real-world applications appearing in today's changing and competitive industrial environment, the difference between using a quickly derived solution and more sophisticated approaches to find an optimal solution can determine whet-

her or not a company survives. However, the development of more sophisticated and effective approaches usually involves a greater computational effort, which in many real-world applications may slow down the production process. Therefore, designing effective and, at the same time, efficient approaches is fundamental. As a case study, this paper analyzes the solution to cutting and packing problems, because of their complexity, their extensive study in the literature and their wide-ranging practical applications. Effective and efficient methods have been designed and implemented to solve these kind of problems that take into account single and multi-objective optimization. Moreover, if these methods are defined as generally as possible, they may be easily adapted to suit different cutting and packing problems without requiring too many changes.

Generally, the quality of the solutions depends on the algorithmic method applied, the efficiency of the implementation of the method, and the parametrization. Thus, considering the wide range of methodologies for solving optimization problems and the techniques available to increase their efficiency, several methods for solving certain cutting and packing problems have been designed and implemented in an effort to improve existing proposals in the literature. The three problem types addressed are: the *Two-Dimensional Cutting Stock Problem*, the *Two-Dimensional Strip Packing Problem*, and the *Container Loading Problem*.

The following methods for obtaining solutions to these problem variants were tested: exact algorithms, heuristics, metaheuristics and hyperheuristics. The resulting methodologies were considerably improved thanks to the incorporation of parallel schemes, making use of the OpenMP and MPI programming tools, and also island-based models in multi-objective optimization. In the case of exact approaches, parallel implementations allowed for the achievement of optimal solutions quicker, thus making it possible to afford larger problem instances. In the case of approximate approaches, parallel schemes allowed for improvements to the solution quality.

So, in short, the research conducted as part of this thesis focused on solving cutting and packing problems using different approaches (single and multi-objective) and different algorithmic methods, always for the purpose of improving the proposals known in the literature. The general structure of the contents are briefly described next. First, Chapter 1 compiles general and indispensable information related to cutting and packing problems, e.g. problem typologies and classifications, bibliography reviews, libraries of problem instances, etc. Chapter 2 introduces the possible solution methods for these kinds of problems and some tools which support different solution techniques. Some of these methods have been used to solve three representative cutting and packing problems. As a result, the subsequent three chapters describe the three problems, the related literature, and several solution methods experimentally applied to each one. Thus, the Two-Dimensional Cutting

Stock Problem is addressed in Chapter 3, Chapter 4 deals with the Two-Dimensional Strip Packing Problem, and the Container Loading Problem is detailed in Chapter 5. Finally, Chapter 6 is devoted to conclusions and to future lines of work.

B.3. Contributions

As already mentioned, the research made in this thesis has focused on solving three cutting and packing problems using different approaches and different algorithmic methods, always for the purpose of improving the proposals known in the literature. The main contributions of this work can thus be divided into three groups: those involving the optimization of the Two-Dimensional Cutting Stock Problem, those involving the optimization of the Two-Dimensional Strip Packing Problem, and those involving the optimization of the Container Loading Problem. Then, the contributions made in each of the problems are briefly described.

Two-Dimensional Cutting Stock Problem (2DCSP)

This two-dimensional cutting problem involves the cutting of a large rectangle S of dimensions $L \times W$ into a set of smaller rectangles (see Figure 3.1). The cutting process must be done by means of orthogonal guillotine cuts. The use of orthogonal guillotinable builds means that any cut must run from one side of the rectangle to the other and be parallel to one of the edges (see Figure 1.1). This constraint is usually imposed by the nature of the cutting machines used in the production industry, which in many cases rely on mechanical cutting blades. It is usually assumed that all of the cuts are infinitesimally thin. After the cuts are made, the resulting rectangles must belong to one of a given set of rectangle types $\mathcal{D} = \{T_1 \dots T_n\}$, where the i-th type T_i has dimensions $l_i \times w_i$. Every piece has a fixed orientation, i.e. a piece of length l and width w is different from a piece of length w and width l (when $l \neq w$). Associated with each type T_i is a profit p_i and a demand constraint b_i. The profit or value of a given piece is not restricted to being proportional to its area. The demand constraint establishes the maximum number of pieces of type T_i that can appear in a feasible cutting pattern (in the unconstrained formulation of the problem there is no limitation regarding the number of pieces of each type available). Under such considerations, the goal is to find a feasible cutting pattern with x_i pieces of type T_i so as to maximize the total profit obtainable from the raw material available.

This problem is so complex and so computationally expensive that most of the methods proposed for solving it are mainly based on heuristic or metaheuristic techniques. However, in some industrial fields the raw material is very expensive, so obtaining the optimal pattern is very important. That is why developing an

exact algorithm is so crucial sometimes. In the literature, the exact algorithms for this problem are mainly based on branch-and-bound and/or tree-searches strategies. Out of all of the exact proposals, the VB algorithm, which is based on a tree-search, has been shown to be the most efficient [175]. In fact, thanks to its efficiency, this algorithm has been the subject of several studies and suggestions for improvement. Hifi [92] and Cung et al. [49] propose a new modified version of this algorithm called MVB. As a result, an exact algorithm based on the MVB algorithm has been developed in an effort to improve the time required to obtain a solution.

The algorithm uses two lists - OPEN and CLIST - to produce the set of feasible solutions. The OPEN list stores the new construction generated during the previous combinations. It consists of a vector of pointers to linked lists of subproblems, which starts at the value given by the initial heuristic for the lower bound and ends at the value of the upper bound of the initial problem. Subproblems with the same upper bound are in the same linked list (see Figure 3.6). Meanwhile, the CLIST list stores the meta-rectangles which have been combined with other meta-rectangles. It is a bi-dimensional data structure limited by the (L, W) dimensions of the sheet of material S, where each element is a linked list of subproblems with the same length and width values (Figure 3.7). Initially, OPEN contains a copy of each of the demanded rectangles of each type, and CLIST is empty. At each step, the most promising meta-rectangle of OPEN is chosen and it is moved to CLIST. The selected item is horizontally and vertically combined with elements in CLIST to produce horizontal and vertical constructions. Every feasible solution obtained from the new meta-rectangles produced is accommodated in OPEN. The feasible solution is a meta-rectangle whose dimensions do not exceed the dimensions of the raw material and contains a set of demanded pieces. The rest of the algorithm uses the appropriate upper bounds to explore the area not occupied by the initial rectangle (Figure 3.8). A pseudocode of the algorithm is presented in Algorithm 1.

To improve the performance of this algorithm, certain rules for the detection of dominated and duplicated patterns have been introduced. *Duplicated* branches of the search tree represent equivalent cutting patterns, i.e. physically, they represent the same usage of the stock sheet, although the same set of final pieces can be obtained through different cutting processes (see Figure 3.9). *Dominated* branches, on the other hand, represent non-promising cutting patterns, i.e. there exists a "similar" build which improves somehow on the dominated one (see Figure 3.10). These rules make it possible to significantly reduce the search space, allowing us to improve the efficiency of the best-first algorithm. These rules of dominance and duplicate detection are based on the internal properties of the problem solutions, i.e., on the combination of pieces to generate the cutting patterns. Some rules can be applied before the pattern generation process (*pre-generation rules*) [49] and others are ap-

plied after their creation (*post-generation rules*) using the data structures available. When a composition is generated, the algorithm checks whether the composition is dominated or whether an equivalent composition can be generated during the pattern combination phase. In all of the above cases, the arrangement is ignored. If not, we proceed to its insertion in OPEN. We now compare the pattern to those already inserted in the same entry in the list, so that we can delete the patterns in the sublist which are dominated or duplicate.

Here, three different types of duplication/dominance *pre-generation rules* and two different types of *post-generation rules* have been proposed. The computational results presented (see section 3.4.2) show that these optimizations of the sequential algorithm yield a significant increase in its efficiency. Moreover, it has been shown that although the post-generation rules are applied less frequently, they have more of an impact in reducing the total search time. These rules are not computationally expensive and are also able to detect redundant buildings which the pre-generation rules detect.

But even with the detection of duplicate and dominated patterns, the search space is too large to deal with the largest test problem instances. That is why two parallel schemes based on the efficient exact algorithm have been proposed. The first algorithm is based on a fine-grained scheme (see section 3.5.1) and involves the parallelization of the two building generation loops [76]. It is a master-slave model where the master is responsible for extracting patterns from OPEN, inserting them into CLIST and introducing all of the new combinations into OPEN. The new patterns are generated in a loop where the best current pattern is combined with those stored in CLIST, in a routine that is distributed among all of the slave processes. The second parallel algorithm is based on a coarse-grained model (see section 3.5.2). The processors distribute the initial pieces among their OPEN lists and run the full search loop in parallel [116]. To guarantee the generation of all possible compositions, every certain interval (of time or iterations) the processes are synchronized and the compositions analyzed so far are communicated so as to determine the best overall solution. In addition to this synchronization mechanism, the algorithm introduces a load balancing strategy that allows an equitable distribution of work among processes. For each algorithm two implementations were developed: one implementation based on shared memory (using OpenMP) and another one based on distributed memory (using MPI).

Results demonstrate that the fine-grained algorithm fails to scale well since it is not possible to distribute the work properly. The real work load depends on the number of structures stored in each position of the matrix CLIST, and typically constructs are not uniformly distributed in the structure. Moreover, the work that can be parallelized in one iteration of computation with this approach is very limited. Ne-

vertheless, in spite of the highly irregular computational structure and the difficulties in breaking the sequential nature of best-first search approaches, the combination of a bulk synchronous methodology with the use of a load-balancing strategy resulted in a fair work load balance and in a linear speedup for the coarse-grained algorithm. Moreover, a comparison between the OpenMP and MPI implementations of the coarse-grained algorithm shows that, in general, the executions with OpenMP involve a lower search time, improving the behavior obtained with the MPI implementation. Only when the problem is simple and the number of threads is increased, the OpenMP approximation cannot improve the results obtained with MPI. Anyway, we have demonstrated that, even for algorithms with an inherent sequential structure, it is possible to design suitable parallel schemes capable of improving the efficiency of the approach. Finally, it is important to note that the efficiency of the parallel schemes mainly depends on the work grain and load distribution, although the selection of a suitable parallel programming tool may be also decisive.

Apart from this, a more realistic view of this problem should take into account other factors when optimizing. For this reason, a multi-objective approach that seeks to maximize profit and minimize the number of cuts required to obtain the pieces has been proposed. While there are many approaches in the literature for solving the single-objective problem, multi-objective proposals dealing with guillotine cuts are unknown. Applying an exact method to solve a multi-objective problem is non-viable due to the magnitude and complexity of the search tree. Moreover, tackling a multi-objective problem by trying to unify the various objectives into a single objective function requires knowing the specific demands for the two objectives at all times so as to combine them in the best way possible. If the demands change, further executions would be required. Using strategies that can address the multi-objective problem as such, only one executions is required. The decision maker can then select the most appropriate solution depending on the demands at the time. If we select a MOEA among the meta-heuristics for multi-objective optimization (see section 2.4), we need only use a general scheme of the algorithm and define the characteristics of the problem: how to represent the individual, how to generate the initial population, how to evaluate the objectives, and what evolutionary operators used. Normally, the person interested in solving the problem has an in-depth knowledge of the problem, but not of how to implement solution methods. This alternative, then, is a good choice for solving a multi-objective problem.

Therefore, in this case, we opted to use MOEAs to solve the multi-objective problem. After doing some tests, NSGA-II was selected given its good behavior for this problem. Three encoding schemes have been implemented to represent an individual in the population: two direct encoding schemes based on a postfix notation

(*controlled size* and *complete size*), and one hyperheuristic encoding scheme. The difference between both direct schemes is the way in which the obtaining of valid solutions is controlled, but both schemes use the same notation (see section 3.6.1). However, some placement heuristics have been developed for the hyperheuristic encoding scheme. The placement heuristics are combined and give rise to the hyperheuristic (see section 3.6.2). There are only a few references in the literature on specific heuristics for solving the guillotine 2DCSP. Nevertheless, there is a set of level-oriented placement heuristics for arranging the pieces. These heuristics can be easily applied to the 2DCSP because of the way in which they operate. Thus, the NFDH, FFDH and BFDH heuristics have been implemented (see section 3.6.2.1), because they can provide different advantages when applied together. Moreover, in a later step, another heuristic previously implemented to obtain the lower bound of the exact algorithm will be added. This not only takes into account the size of the pieces, but the profit associated with each of them, as the profit may not necessarily be proportional to the area.

The computational results have demonstrated the effectiveness of MOEAs applied to this kind of problem with all of the encoding schemes (section 3.6.3). Although the proposals do not yield the value of the solution with maximum profit, obtained by applying the exact method explained above, the solutions are fairly close and provide a good compromise between both objectives. Therefore, in keeping with the two optimization criteria, three proposals that result in a series of compromises between the two objectives in a single run have been designed, allowing customers to choose based on their needs. Solutions were obtained even with a very low number of cuts. In most cases the number of cuts was reduced to 40 % or 50 % of those obtained with the single-objective solution. Moreover, it must be realized that the implementation of an exact algorithm involves a great deal of difficulty and cost. As if this were not enough, the exact algorithm focuses on a single objective without considering the possible negative effects on other features of the solutions.

The encoding scheme used in the multi-objective approaches has been analyzed to ascertain the differences among them. Firstly, if we compare both direct encoding schemes, we find that one of them performs better with large instances of the problem, and the other performs better with instances involving fewer pieces. Instead, the results show that while the encoding scheme based on hyperheuristic yields competitive solutions when compared to individual heuristics, it does not approach the performance provided by direct encoding schemes. This may be because the hyperheuristic is based on single-objective heuristics that do not provide high-quality results for this problem because they were originally designed to deal with level-oriented cutting problems. We thus found that by adding a new heuristic to the scheme, one that was previously implemented to obtain the lower bound of the

exact algorithm and which takes into account the profit associated with each piece, the behavior of the hyperheuristic is noticeably improved.

Two-Dimensional Strip Packing Problem (2DSPP)

This two-dimensional cutting problem involves the cutting of a set of n demanded pieces from a large roll or stock sheet of raw material (Figure 4.1), while minimizing the length of stock required. The stock sheet has fixed width W and unlimited length L. Due to its unlimited length, the stock sheet is often known as a "strip". Each rectangular piece i demanded has fixed dimensions (l_i, w_i), although they can be rotated 90 degrees, thus allowing for the dimensions (w_i, l_i) as well. All demanded pieces must be orthogonally arranged on the material in such a way that they do not overlap and that only vertical or horizontal builds of pieces are generated. This last constraint always gives the possibility of producing the packing pattern through guillotine cuts, i.e. the pieces have to be cut with their edges parallel to the edges of the stock sheet, going from one border straight to the opposite side. The production of non-guillotinable cuts may entail a more complex machinery operation. As a result, and in order to encompass a wider range of industrial cutting machines, in the approach proposed only guillotinable patterns are built so that all of the resulting solutions can be cut in both guillotine and non-guillotine modes.

As was done with the 2DCSP, an exact algorithm for solving the 2DSPP has been developed that takes into account guillotine cuts and the rotation of pieces. The main ideas of the sequential proposal are based on a best-first search similar to the VB [175] algorithm and its modification MVB [92]. This algorithm was originally designed to solve another cutting problem, the 2DCSP indicated earlier. However, this algorithm has been adapted to solve the 2DSPP. The implementation is based on Hifi's proposal [93], which defines an exact approach called BMVB. This proposal stars from the largest rectangle (L_{max}, W) and tries to build an arrangement of pieces with a length less than or equal to L_{max}. At each step, the algorithm tries to build a smaller rectangle that contains all of the pieces. If this rectangle exists, then (L_{max}, W) is updated and another iteration of the algorithm tries to build a smaller rectangle with all of the pieces.

As a novelty over the BMVB algorithm, the BFDH* heuristic [21] has been used to obtain the upper bound of the length of the material, i. e., the first rectangle (L_{max}, W). This heuristic considers the rotation of the pieces because the particular definition of the problem being treated considers this rotation. If all of the pieces fit into the material without gaps, the necessary length for this case is taken as the lower bound for the length of the material, i. e., the sum of the areas of all the pieces are calculated, followed by the corresponding length, knowing the width of the material. Moreover, to improve the performance of the MVB algorithm, the dominated

and duplicated rules previously used in the exact algorithm for the 2DCSP were adapted and used here. This algorithm was also adapted to allow for the rotation of pieces. This way, the possible lengths are covered through a loop between the lower bound and the upper bound, and the adapted algorithm is called (see Algorithm 5). As shown in the experimental study, the time spent in obtaining the solution is satisfactory for most of the instances used (Table 4.2). However, when the number of pieces increases the time can be increased too. Thus, for instances with a large number of pieces, it is advisable to parallelize the algorithm.

The sequential version of the algorithm was analyzed and the section of the code involving more computational effort was selected for parallelization. This section corresponds to checking whether all of the pieces fit within the sheet of material for a specific length, i.e., the call to the MVB adapted to the 2DSPP. For this section of code, then, two implementations of a fine-grained parallel algorithm (using shared memory and distributed memory), and other two implementations of a coarse-grained parallel algorithm (using shared memory and distributed memory), are used. OpenMP was employed for the shared memory implementations and MPI was employed for distributed memory implementations. In analyzing the sequential behavior of the exact algorithm for the 2DSPP, once the optimal length for the initial test problem instances is known, it has been shown that when the length used to call the adapted MVB algorithm is not enough to accommodate all of the pieces of the instance, the time spent on call is irrelevant compared to the time spent when called using the optimal length. For this reason, the times spent on call with the adapted MVB algorithm with the optimal length is taken into account for each instance selected to analyze the behavior of the parallel algorithms.

Comparing the results obtained using the coarse- and fine-grained algorithms, it can be concluded that the coarse-grained schemes are capable of better scaling (see Tables 4.3, 4.4, 4.5 and 4.6). In fact, most of the time a fine-grained scheme will not yield a solution in under 12 hours. The grain is too fine to obtain good results when the number of processors increases. However, when the fine-grained schemes provide results using few processors, the solutions are obtained in less time than that required by the coarse-grained algorithms with the same number of processors. At any rate, in order to obtain the exact results in less time than the sequential version of the algorithm, the best option is to use coarse-grained algorithms. Nevertheless, the results obtained using the coarse-grained algorithms do not exhibit constant acceleration if the number of processors is increased. This is because the tree structure that uses the search is highly irregular, and when the nodes are distributed among the processors to perform the search, in some cases the node with the optimal solution can be analyzed by a process at the beginning of the search, and in other cases the node with the solution may be too deep into the tree branch that

will analyze the process. Even when the node with the optimal solution is not too deep in the tree, if another process has accumulated too much work and a synchronization is performed, the load balancing scheme may cause a delay in the analysis of the correct node. In short, what happens stems from a change in the order of exploration of the search tree nodes. This behavior is further affected by how the load is balanced. In contrast, a change in the exploration order can never happen using fine-grained implementations.

In general, the time spent solving a test problem instance using the coarse-grained approaches is related to the number of nodes inserted in the CLIST list used by the algorithm, i.e., the number of nodes that are removed from the OPEN list and combined with the best previous subproblems already explored (*computed nodes*), and this, as mentioned, is unpredictable given the irregular structure of the tree and load balancing scheme. This relation is shown in Figure 4.5 and Figure 4.6. Furthermore, the distribution of work among processors during an execution is not as balanced as was the case with the 2DCSP. This may be because the solution of these instances is more complex, allowing for the rotation of the pieces and distributing all of the pieces throughout the material, unlike the 2DCSP, so it must test among all possible combinations of pieces. In this sense, the heuristics used as a lower bound for the 2DCSP helped in some way to choose the pieces that could give better solutions, depending on the availability and profits associated with each piece, but for the 2DSPP that guide does not exist in the search because in the end it has to put all of the pieces in the material (Table 4.7).

Once the exact solution of the 2DSPP is analyzed, a more realistic multi-objective version of the problem is proposed. As in the case of the 2DCSP, in this problem we consider a second objective, namely that of minimizing the number of cuts required to yield the demanded pieces. This is because if the material is cheap, then this criterion is of greater importance, affecting the industrial machinery and the speed the process. As mentioned earlier, applying an exact method for solving a multi-objective problem is unfeasible due to the magnitude and complexity of the search tree. We have thus decided to use multi-objective metaheuristics to solve the multi-objective 2DSPP. We have opted to use the MOEAs, among others, since by employing METCO, we need only use a general outline of the selected algorithm and define the basic characteristics of the problem: how to represent the individual, how to generate the initial population, how to evaluate the objectives and what evolutionary operators are used.

An important decision is what representation of an individual of the population to use. Here, two different kinds of encoding schemes are used: one scheme based on a direct encoding which does not use codification of solutions, and another one

based on hyperheuristics. For this latter case, four approaches are considered. The reason for studying many hyperheuristic schemes is to find the best hyperheuristic representation, and to see how the encoding scheme affects the solution space, so several problems and instances can be addressed by simply replacing the low-level heuristics. The difference among the hyperheuristic schemes is the way in which the pieces are ordered and oriented when they are placed in the material by the heuristics. The first scheme, called the **(h, p)** hyperheuristic, uses the order and orientation given by the low level heuristics. The second one, denoted as **(h, p) + or + ot**, introduces the specific order and orientation of pieces as part of the chromosome. The third one, called **(h, p, o) + ot**, is equal to the second one, but it uses order criteria in the chromosome instead of the ordered sequence of pieces. Finally, the fourth one, called **(h, p, o, r)**, is equal to the third one, but it uses rotation criteria in the chromosome instead of the sequence of orientations of pieces.

In using METCO for the 2DSPP, different types of individuals to solve the problem have been employed: one for the direct encoding scheme based on a postfix representation of the distribution patterns, and four for the hyperheuristic encoding schemes. So, for each approach, we have defined the corresponding representation, the generation of new individuals, evaluation of objectives, and crossover and mutation operators. However, some placement heuristics had to be developed for the hyperheuristic encoding scheme. The placement heuristics are combined and give rise to the hyperheuristic. There are not many references in the literature to specific heuristics for solving the guillotine 2DSPP. Nevertheless, there is a set of level-oriented placement heuristics for placing the pieces. Thus, the heuristics NFDH, FFDH, BFDH and BFDH* have been implemented because they provide different advantages when applied together. For each implementation, the direct one and hyperheuristic ones, different MOEAs has been used. After an initial tuning, the algorithm and parameters that provide the best behavior are selected. Note that, as in other related references [171], the NSGA-II has the best behavior among the algorithms selected. Thus, this evolutionary algorithm was applied for all of the following sequential experiments.

Firstly, the direct encoding scheme was tested. Initially, the purpose of solving the 2DSPP using evolutionary algorithms and the direct encoding scheme is to check whether the results obtained improve on the only results that are known to solve the multi-objective problem using guillotinable cuts [171]. Both guillotine and non-guillotine cutting processes were taken into account. The solutions obtained are clearly superior and the results show that the application of guillotine cuts requires a greater number of cuts than those necessary to obtain the pieces with non-guillotine cuts (see Table 4.9). Dealing with more than one optimization objective does not necessarily imply a reduction in the quality of the solution at the expense of the

possibility of optimizing multiple objectives. With this first multi-objective approach to the 2DSPP, solutions are obtained with waste levels similar to previous approaches that merely attempted to optimize the total length (see Table 4.10). For the smaller test problem instances, the multi-objective approach provides better solutions than other single-objective approaches. Only for large problem instances is the multi-objective approach unable to provide solutions that are better than or similar to those obtained by single-objective techniques.

Given the promising sequential results, further improvements could be expected if we use the parallel island-based approaches provided by METCO. As an initial proposal, we tested several homogeneous island-based models. For the sequential executions, the second MOEA that exhibits good performance after NSGA-II is SPEA2. A homogeneous scheme was thus defined for each of the two algorithms, NSGA-II and SPEA2 (Figure 4.21). The results show that when parallelism is introduced, the SPEA2 algorithm performs better than NSGA-II. This difference is accentuated when using a larger number of slaves in the island model. Moreover, the number of evaluations required to converge to a certain level of quality increases as more islands are included in the model. Since the evolutionary process and the associated operators are applied considering only subpopulations of the islands, the convergence of the exploration will be slower in the parallel model. However, this situation is compensated by the acceleration obtained due to the parallelism built into the scheme. Two additional island-based models were tested: a standard heterogeneous model and an auto-adaptive heterogeneous model. Heterogeneous executions and auto-adaptive heterogeneous executions provide better solutions than the worst homogeneous scheme, but worse than the best one (see Figure 4.24). Nevertheless, it is difficult to know a priori what the most appropriate algorithm is for solving a given problem. This would induce the user to test several algorithms before making a final decision. In contrast, heterogeneous models and auto-adaptive heterogeneous models achieve near optimal solutions, meaning the user does not have to individually test a variety of MOEAs and then select the one best suited to solving the problem. In light of the similar results provided by the heterogeneous, auto-adaptive heterogeneous and homogeneous algorithms, it may be concluded that the parallel implementations provided by METCO maintain the quality of sequential solutions while yielding improved performance.

In any case, even using parallelism, the solutions obtained by the multi-objective approach are worse than those obtained by single-objective techniques for large test problem instances. The direct encoding scheme yields values for the number of cuts that are significantly lower than those obtained by the heuristics, but it is not capable of reaching length values. Thus, an encoding scheme that is able

to adequately narrow the large search space is necessary. For this reason, the four hyperheuristic encoding schemes were developed (see section 4.5.2.2).

Using the first one, both objectives can be improved with respect to the single-objective heuristics. The second one is not able to improve any objective. Finally, the third and fourth proposals improve both objectives (Table 4.11). The hyperheuristic schemes designed, **(h, p)**, **(h, p, o) + ot** and **(h, p, o, r)** are thus able to improve the results given by the low-level heuristics, i.e., using the principles of hyperheuristics and multi-objective approaches, we have improved the solutions obtained by tailor-made algorithms for the single-objective 2DSPP.

However, each encoding scheme focuses on a different search space, so the results are found in different areas of the solution space. If we compare the complete set of solutions obtained by the direct and hyperheuristic encoding schemes for large test problem instances (Figure 4.30), we can see that the fourth hyperheuristic encoding scheme is the only one capable of covering a wide area of the solution space, improving the number of cuts given by the other hyperheuristic encoding schemes, and the length values given by the direct encoding scheme. Thus, for large instances, the direct encoding scheme gives solutions that do not adequately minimize the length of the raw material, but, the fourth hyperheuristic scheme, **(h, p, o, r)**, is able to adequately minimize the length and provide a reasonable number of cuts.

Therefore, we have a hyperheuristic encoding scheme, **(h, p, o, r)**, which improves the performance of low-level heuristics (Figure 4.31), and a direct encoding scheme which does not improve the length objective provided by the heuristics, but which does provide the lowest number of cuts. However, the advantage of having different efficient codifications covering different areas of the solution space is that the decision maker can choose the type of codification depending on the area of the solution space that is of more interest to the problem at hand.

Container Loading Problem (CLP)

The CLP belongs to an area of active research and has numerous applications in the real world, particularly in the container transportation and distribution industries. In general, these problems involve distributing a set of rectangular pieces (boxes) in one large rectangular object (container) so as to maximize the total volume of packed boxes, i.e., trying to obtain a packing pattern that utilizes as much of the container space as possible. When solving a real CLP, maximizing the used space is normally the most important goal, but there are other issues that may be taken into account. A rather common aspect when considering this problem is the weight limit of the containers, since their transportation normally cannot exceed a certain limit. The rented trucks to transport the shipment are paid according to the total weight they can transport regardless of the total volume. Thus, the decision

maker prefers to load and ship a shipment with high total weight rather than a shipment with low total weight. As a result, in this paper the total weight is treated as a second objective. The problem can then be stated as a multi-objective optimization problem that aims to optimize the layout of the pieces inside the container so that both the volume and weight are maximized without exceeding the weight limit. The problem formulation is thus as follows: we have a container with known dimensions ($W = width, L = length, H = height$), a maximum weight it can hold P_{max}, and a set of N rectangular boxes. These boxes belong to one of the sets of box types $\mathcal{D} = \{T_1 \dots T_m\}$, where the i-th type T_i is of dimensions $w_i \leq W$, $l_i \leq L$ and $h_i \leq H$. Associated with each T_i type is a weight p_i, a volume v_i, a demand b_i, and a number of orientations allowed $o_i \in [1, 6]$ (see Figure 5.1). Ultimately, the aim is to find a container packing solution without overlapping, with x_i boxes of type T_i and maximizing the total container volume used and the total weight of the packages.

The problem will be solved using the following assumptions:

- Each box is placed on the container floor or on top of another box.
- The stability of the distribution of the boxes is not considered, since it is assumed that filler material is used to prevent potential problems.

Since MOEAs have shown promise for solving multi-objective problems in the real world, especially in cutting and packing problems [51, 54], it would be interesting to prove their effectiveness in this particular problem. The purpose is to make a comparison with the only known multi-objective approach [57] that takes into account the same goals: maximizing the weight and available volume of the container. Moreover, to improve performance and deliver quality results in less time, a study of parallelizations using the island-based models provides by METCO will be conducted.

To start the study then, we begin by defining the representation of individuals, the initial generation of individuals, the evolutionary operators, and the evaluation of the objectives (see Section 5.3) needed to obtain results using METCO. In this case, a direct encoding was not used for the representation of individual since the search space with a postfix notation would be too large (there are more operators, more possible orientations of the pieces, etc.). An encoding solution was used. It indicates the placement sequence and the orientation of the boxes that a heuristic must use when placing them in the container.

After defining the required characteristics of the problem, a sequential study was carried out using three different MOEAs: NSGA-II, SPEA2, and an adaptive version of IBEA. As shown in Figure 5.9, the hypervolume for NSGA-II and SPEA2 exhibits a very similar behavior, whereas the hypervolume for the adaptive version

of IBEA is a little smaller. In fact, the hypervolume for NSGA-II seems to be the largest. Moreover, since this MOEA has shown good behavior in other cutting and packing problems, it was selected for the study. Clearly, the results obtained with this MOEA improve the reference values, resulting in higher volume packaging than the best of the four reference values of volume, even when we focus on the results associated with higher weights (see Table 5.3). The results cover a large area of the solution space, offering a set of compromise solutions between the two objectives, so that the decision maker may choose the most appropriate option in each case (see Figure 5.10). However, different solutions given in [57] are completely dominated by the new solutions. That way, the validity of MOEAs to tackle such problems is demonstrated, surpassing the results obtained using a simulated annealing algorithm. Moreover, the set of compromise solutions is obtained in only one execution, whereas the algorithm applied in [57] assigns different weights to each objective and requires a separate execution for each weight combination.

Once this is verified, it would be interesting to see if the island-based parallel models offered by METCO can improve on the performance of the sequential model. The *homogeneous* model, where every island executes the same configuration of a MOEA, was analyzed. The *homogeneous* model with the NSGA-II algorithm and the same parameters applied in the sequential execution were executed with two, four and eight islands. The results for these parallel executions with two islands improve on the sequential results, especially when the number of evaluations, and therefore the time, increases. The executions with four islands only attain the quality of the sequential ones at the end, and the executions with eight islands are not able to improve on the quality of the sequential executions over the same number of total evaluations (Figure 5.11). This is because the more islands used, the lower the number of evaluations or generations each one makes separately, i.e., the same evaluations are divided among more processors, so each processor make less headway in the exploration. Then, if we analyze in depth how many evaluations on average each processor needs to achieve a percentage of the hypervolume, we see that each processor in a parallel execution carries out fewer evaluations than a processor in a sequential execution (see Table 5.4). Moreover, we note that increasing the quality of the solution requires a steadily higher number of evaluations by each processor. In contrast, reducing the number of evaluations performed by each processor in parallel versus sequential executions decreases the time required to reach a solution of a given quality. In this sense, the benefits of parallel executions are remarkable since the time needed to achieve good quality solutions that are almost optimal is drastically reduced.

B.4. Conclusions

Cutting and packing problems are a family of combinatorial optimization problems that have been extensively studied in many areas of industry and research due to their broad practical applications. Of the many types of cutting and packing problems that exist, in this paper we focus on solving three of the most common variants: the 2DCSP, the 2DSPP, and the CLP.

Generally, when faced with solving an optimization problem, taking into account different approaches and solution methods is useful when attempting to adapt the problem to a specific environment. The methods to be applied depend mainly on the specific nature of the problem, on the features expected from the solution, and on the available computational resources. Thus, deciding on the best solution methods requires knowing the complexity of the problem, the size of the test problem instances to solve, the kind of solution expected, the optimization type (single or multi-objective), the maximum time available to obtain the solution, which depends on the environment, and the available computational resources, since the use of machines and parallel implementations can significantly reduce the solution time. Therefore, based on the user's needs, the design of an effective and, at the same time, efficient approach is fundamental. Taking into account the wide range of methodologies for solving optimization problems and the techniques available to increase their efficiency, several methods for solving some cutting and packing problems have been designed and implemented in an effort to improve the proposals found in the literature: single-objective exact methods and their parallelizations, and approximate multi-objective methods and their parallelizations. Thus, some of the contributions of this thesis include improving the existing results in the literature for solving some variants of these problems, and providing new and realistic proposals. An extensive and thorough review of the literature is presented for each of the cutting and packing problems selected.

As shown in the associated literature, there are many approaches for solving the selected problems, although it seems that the heuristics and/or metaheuristics are the most used methods, since the exact methods are not too feasible for this type of complex problems. However, in this study, some exact approaches are analyzed as a mechanism for finding the optimal solutions and, therefore, for using this data to make a subsequent comparison with the results of the approximate methods and thus ascertain their quality. Since the exact algorithms have too high a computational cost, it was necessary to design specific optimizations to reduce the search space, (for example, rules to detect dominated and duplicated patterns) and parallel implementations to reduce the time spent obtaining the problem solutions. Using parallelization of exact algorithms for the 2DCSP and 2DSPP, we have shown that

even for algorithms with an inherent sequential structure, it is possible to design a suitable parallel scheme that can improve the efficiency of the approach. Importantly, the efficiency of parallel schemes depends mainly on the grain of the work and on the distribution of the workload, although the selection of a suitable parallel programming tool is also crucial.

Nevertheless, when the instances to be solved are more complex, the exact methods are no longer viable and we must use some kind of approximate method. Moreover, when we want to analyze the problems from a more realistic standpoint that takes into account other key objectives to optimizing production processes in the cutting and packaging industries, it is necessary to address the solution to these problems from a multi-objective view. This implies that there will not be a single problem solution, but rather a set of solutions that optimize the various problem objectives differently. In this case, it was shown that by using specific methods for solving multi-objective problems, such as the MOEAs, we can obtain a set of solutions that provide a compromise between the different objectives to be optimized. Thus, depending on the case, the user can select the solution that is best suited to one objective or another, depending on the pressing interest at any given time.

Unlike what happened with the exact algorithms, the general scheme for this type of evolutionary algorithms was not designed specifically for the problems discussed here. However, it was necessary to define some aspects which do depend on the particular problem and, as we have seen, can significantly affect the diversity and quality of the solutions obtained. This is the case of the representation chosen for solutions and for mutation and crossover operators. There are several possibilities for coding the solutions and associated operators, and depending on which is chosen, the results can be more or less appropriate to the instances that need to be solved. Therefore, defining how the solutions and associated operators are coded when using MOEAs is as important as the solution method itself. Thus, for each cutting and packing problem addressed, several encoding schemes were provided. It has been demonstrated that each encoding scheme focuses on a different search space, so the results are found in different areas of the solution space. Defining a large search space makes obtaining high quality solutions difficult, but when the search space is limited, some regions of the solution space may be left unexplored. Therefore, it is important to design codifications that allow the search space that provides the best balance between the two optimization objectives to be covered.

As regards the exact algorithms, parallel algorithms for this type of multi-objective approach allow us to explore the search space in parallel by obtaining the solutions in less time. In this case, the difference is that the evolution of different populations that can cooperate and/or interface with each other yields improved results over those obtained with sequential executions. This was clearly demonstrated for the

2DSPP and the CLP.

Finally, we have proven the importance of using skeletons, frameworks and tools like METCO, which provide, in general, global and internal features of the algorithms and allow the user to focus only on the most specific issues of the problems. Thanks to these tools and the attempt to create general representations of solutions to the cutting and packing problems presented, many of the developments, tests and analyses have been extended to other problems. For example, it was possible to use the same implementations of the MOEAs by using METCO for the three problems addressed. Actually, the encoding schemes used for the individuals in the multi-objective approaches were quite similar. The direct encoding schemes are based on the same postfix notation adapted to each problem, whereas the hyperheuristic scheme which provided the best results for the 2DSPP matched the hyperheuristic scheme that was used for the 2DCSP, the only difference being the rotation of pieces. Moreover, the low-level heuristics used by hyperheuristics schemes were also reused. Even for exact proposals, we were able to reuse code, since the efficient and exact algorithm used for the 2DCSP and its parallelizations were reused and adapted to solve the exact 2DSPP.

In any case, an effort was made to propose general solution schemes so that, with minor modification, methods can be adapted to different cutting and packing problems. For some problems, new solution methods were used that have not been applied to these problems in the associated literature. In others, results in the literature were improved by ehnancing the methods applied.

B.5. Future Work

As in most research or studies, this work offers glimpses into new areas that lend themselves to analysis. Thus, as a future line of research into 2DCSP, it would be interesting to design other heuristics and add them to the hyperheuristic scheme used in the multi-objective version of this problem. This way, we could determine whether, as was the case when we added the last heuristic to the scheme, this improves its behavior and achieves or improves on the results obtained with the schemes that use a postfix notation. Once the hyperheuristic scheme that presents the best behavior is determined, both this and the direct encoding schemes that were analyzed in this paper could be used to study parallel island-based models. To this end, we could use different MOEAs, which, though they exhibited worse performance that the NSGA-II, can perhaps offer some improvement in a parallel heterogeneous model.

In the case of 2DSPP, the hyperheuristic schemes have already been analyzed in

detail, and the heuristics used in them have been shown to be suitable for the problem. However, improvements are achievable, not only with these schemes but also with the direct one, by designing and testing other mutation and crossover operators to increase efficiency. Some alternatives were tested but an in-depth study could be conducted. Once these schemes are refined, we could introduce more MOEAs in parallel island-based models, as used herein. Moreover, the MOEAs employed could be hybridized with the introduction of local searches to try to better expand the search space to certain regions of the solution space. As an alternative, the incorporation of a decision maker during the execution of the MOEAs could allow for some areas of the solution space that are not of interest to be discarded, while maintaining the diversity of the solutions obtained in appropriate regions.

With respect to the CLP, unlike the above problems, we did not use a direct representation of individuals because it was considered inadequate. However, a hyperheuristic-based representation was not used either, and this could be feasible. Therefore, as the next step it would be necessary to design the different heuristics that are part of this hyperheuristic. Once we have defined the scheme, we should develop the mutation and crossover operators to be used. Then, it would be interesting to conduct a detailed study of such operators to try to find the most suitable. Also, for the encoding scheme that was used in this paper, we could carry out a similar study of other evolutionary operators. When the most suitable operators are found, tests may be conducted with other MOEAs, besides the NSGA-II, which have shown good sequential behavior, as we have proposed for the above problems. Thus, the possibility of running different parallel heterogeneous models for this problem, where each island runs a different algorithm, will exist. It should be noted that this way we not only aim to save time by distributing the computational effort, but to obtain the algorithmic benefits from cooperation among several populations. Finally, since for this problem we were able to find only one instance that takes into account the weight of both the boxes and of the container itself, we had to adapt the instances utilized in the literature for single-objective algorithms. This makes it possible to draw a comparison between the results obtained with single-objective and multi-objective algorithms, and thus determine the quality of some algorithms over others.

So, as a general consideration for all of the problems addressed, the refinement of the encoding schemes and the in-depth study of other mutation and crossover operators in the multi-objective approaches remains a line of work for future research. Also, the application of new MOEAs for these multi-objective proposals would be another important source of work. Finally, since the encoding of individuals has been widely studied for the multi-objective version of the problems addressed, a full

analysis of the heterogeneous parallelizations in these approaches is another possible area for future study. Apart from the heterogeneous parallelizations where each island runs a different MOEA, heterogeneous parallelizations where each island uses a different encoding of individuals could be carried out.

In addition to these future lines of work to improve on each of the proposals made, new cutting and packing problems could be solved following the guidelines undertaken in this work.

It would also be quite interesting to establish contact with real companies in the industry to offer them solutions to their problems and to learn first-hand of the most common restrictions that may occur when solving them, so that methodologies can be adjusted to the real needs of factories.

One particular example of this for cutting problems such as the 2DSPP and 2DCSP would be if the trend in the industry were toward machines capable of executing non-guillotine cuts. In this situation, it would be advisable to focus on these problems without considering only guillotine cuts. This would open up an entirely new field of research, since different representations of individuals, other genetic operators, etc. would be necessary. This would require exploring all of the solution methods anew.

Bibliografía

[1] E. Alba, F. Almeida, M. Blesa, J. Cabeza, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, J. Luna, L. Moreno, J. Petit, A. Rojas, and F. Xhafa. MaLLBa: A Library of Skeletons for Combinatorial Optimization. In R. Feldmann and B. Monien, editors, *European Conference on Parallel Computing (Euro-Par'2002)*, volume 2400, pages 927–932. LNCS, Springer-Verlag, 2002.

[2] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, G. Luque, J. Petit, C. Rodríguez, A. Rojas, and F. Xhafa. Efficient parallel LAN/WAN algorithms for optimization. The MALLBA project. *Parallel Computing*, 32(5-6):415–440, June 2006.

[3] E. Alba, G. Luque, J. García-Nieto, G. Ordóñez, and G. Leguizamón. MALLBA: a software library to design efficient optimisation algorithms. *International Journal of Innovative Computing and Applications*, 1(1):74–85, 2007. ISSN 1751-648X.

[4] S. Allen, E. Burke, and G. Kendall. A hybrid placement strategy for the three-dimensional strip packing problem. *European Journal of Operational Research*, 209(3):219–227, 2011. ISSN 0377-2217.

[5] R. Alvarez-Valdes, F. Parreño, and J. M. Tamarit. Reactive grasp for the strip-packing problem. *Computers & Operations Research*, 35(4):1065–1083, April 2008. ISSN 0305-0548.

[6] R. Alvarez-Valdes, F. Parreño, and J. Tamarit. A tabu search algorithm for a two-dimensional non-guillotine cutting problem. *European Journal of Operational Research*, 183(3):1167–1182, 2007. ISSN 0377-2217.

[7] R. Alvarez-Valdés, A. Parajón, and J. Tamarit. A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. *Computers & Operations Research*, 29(7):925–947, June 2002. ISSN 0305-0548.

[8] P. Andras, A. Andras, and Z. Szabo. A Genetic Solution for the Cutting Stock Problem. In *In 1st Online Workshop on Soft Computing (WSC1)*, pages 87–92. Nagoya University, Japan, August 1996.

[9] I. Araya, B. Neveu, and M. Riff. *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, chapter An Efficient Hyperheuristic for Strip-Packing Problems, pages 61–76. Springer, 2008. ISBN 978-3-540-79437-0. ISSN 1860-949X (Print) 1860-9503 (Online).

[10] M. Arenas, P. Collet, A. Eiben, M. Jelasity, J. Merelo, B. Paechter, M. Preub, and M. Schoenauer. A framework for distributed evolutionary algorithms. In J. Merelo, P. Adamidis, H. Beyer, J. Fernández-Villacañas, and H. Schwefel, editors, *Seventh International Conference on Parallel Problem Solving from Nature*, pages 665–675, 2002.

[11] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

[12] T. Bäck, D. Fogel, and M. Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing Ltd. and Oxford University Press, 1997.

[13] S. Bak, J. Blazewicz, G. Pawlak, M. Plaza, E. K. Burke, and G. Kendall. A Parallel Branch-and-Bound Approach to the Rectangular Guillotine Strip Cutting Problem. *Informs Journal on Computing*, 23(1):15–25, 2011.

[14] J. Beasley. A Population Heuristic for Constrained Two-Dimensional Non-Guillotine Cutting. *European Journal of Operational Research*, 156:601–627, 2000.

[15] A. Bekrar and I. Kacem. An Exact Method for the 2D Guillotine Strip Packing Problem. *Advances in Operations Research*, 2009, 2009.

[16] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171(1):85–106, 2006. ISSN 0377-2217.

[17] E. Bischoff and M. Ratcliff. Loading Multiple Pallets. volume 46, pages 1322–1336, Nov. 1995.

[18] E. E. Bischoff, F. Janetz, and M. S. W. Ratcliff. Loading pallets with non-identical items. *European Journal of Operational Research*, 84(3):681–692, August 1995.

[19] G. E. Blelloch and B. M. Maggs. Programming parallel algorithms. *Communications of the ACM*, 39:85–97, 1996.

[20] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA - A Platform and Programming Language Independent Interface for Search Algorithms. In *Conference on Evolutionary Multi-Criterion Optimization*, volume 2632 of *LNCS*, pages 494–508. Springer, Faro, Portugal, 2003.

[21] A. Bortfeldt. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, 127 (3):814–837, 2006.

[22] A. Bortfeldt and H. Gehring. A tabu search algorithm for weakly heterogeneous container loading problems. *OR Spectrum*, 20(4):237–250, December 1998. ISSN 0171-6468.

[23] A. Bortfeldt and H. Gehring. A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, 131(1):143–161, May 2001.

[24] A. Bortfeldt and H. Gehring. New Large Benchmark Instances for the Two-Dimensional Strip Packing Problem with Rectangular Pieces. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 30.2. IEEE Computer Society, 2006. ISBN 0-7695-2507-5.

[25] A. Bortfeldt, H. Gehring, and D. Mack. A parallel tabu search algorithm for solving the container problem. *Parallel Computing*, 29:641–662, 2003.

[26] A. Bortfeldt and D. Mack. A heuristic for the three-dimensional strip packing problem. *European Journal of Operational Research*, 183(3):1267–1279, 2007.

[27] A. Bortfeldt and T. Winter. A genetic algorithm for the two-dimensional knapsack problem with rectangular pieces. *International Transactions in Operational Research*, 16(6):685–713, 2009. ISSN 1475-3995.

[28] M. A. Boschetti and L. Montaletti. An exact algorithm for the two-dimensional strip-packing problem. *Operations Research*, 58(6):1774–1791, 2010.

[29] R. Brooks, C. Smith, A. Stone, and W. Tutte. The dissection of rectangles into squares. *Duke Mathematical Journal*, 7:312–340, 1940.

[30] M. Brusco, G. Thompson, and L. Jacobs. A morph-based simulated annealing heuristic for a modified bin-packing problem. *Journal of the Operational Research Society*, 48(4):433–439, April 1997.

[31] E. Burke and G. Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter Chapter 1: Introduction, pages 5–18. Springer, 2005.

[32] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward. Exploring Hyper-heuristic Methodologies with Genetic Programming. In J. Kacprzyk, L. C. Jain, C. L. Mumford, and L. C. Jain, editors, *Computational Intelligence*, volume 1 of *Intelligent Systems Reference Library*, pages 177–201. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-01799-5.

[33] E. K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. *Handbook of Meta-heuristics*. Kluwer, 2003. 457–474 pp.

[34] E. K. Burke, G. Kendall, and G. Whitwell. A New Placement Heuristic for the Orthogonal Stock-Cutting Problem. *Operations Research*, 52(4):655–671, 2004. ISSN 0030-364X.

[35] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, 2007.

[36] M. Cadoli, M. Lenzerini, and A. Schaerf. Local++: A C++ framework for local search algorithms. Technical Report DIS 11-99, University of Rome, La Sapienza, 1999.

[37] A. Caprara and P. Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. Technical Report OR/97/3, University of Bologna, 1997.

[38] D. Chen, J. Liu, Y. Fu, and M. Shang. An efficient heuristic algorithm for arbitrary shaped rectilinear block packing problem. *Computers & Operations Research*, 37:1068–1074, June 2010. ISSN 0305-0548.

[39] N. Christofides and C. Whitlock. An Algorithm for Two-Dimensional Cutting Problems. *Operations Research*, 25(1):30–44, 1977.

[40] G. Cintra, F. Miyazawa, Y. Wakabayashi, and E. Xavier. Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation. *European Journal of Operational Research*, 191(1):61–85, November 2008.

[41] C. A. Coello. An Updated Survey of Evolutionary Multiobjective Optimization Techniques: State of the Art and Future Trends. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 3–13. IEEE Press, 1999. ISBN 0-7803-5537-7 (Microfiche). URL `citeseer.ist.psu.edu/coellocoello99updated.html`.

[42] C. A. Coello. Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: A Survey of the State of the Art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, January 2002.

[43] C. A. Coello and G. B. Lamont, editors. *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, Singapore, 2004. ISBN 981-256-106-4.

[44] C. A. Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic and Evolutionary Computation. Springer, 2007. ISBN 978-0-387-33254-3.

[45] E. Coffman, M. Garey, D. Johnson, and R. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:808–826, 1980.

[46] Y. Cui. An exact algorithm for generating homogeneous two-segment cutting patterns. *Engineering Optimization*, 39:365–380, 2007.

[47] Y. Cui. An exact algorithm for generating homogenous t-shape cutting patterns. *Computers & Operations Research*, 34:1107–1120, 2007.

[48] Y. Cui. Heuristic and exact algorithms for generating homogenous constrained three-staged cutting patterns. *Computers & Operations Research*, 35:212–225, 2008.

[49] V. D. Cung, M. Hifi, and B. Le-Cun. Constrained Two-Dimensional Cutting Stock Problems: A Best-First Branch-and-Bound Algorithm. Technical Report 97/020, Laboratoire PRiSM, Université de Versailles, 1997.

[50] A. de Almeida and M. B. Figueiredo. A particular approach for the Three-dimensional Packing Problem with additional constraints. *Computers & Operations Research*, 37:1968–1976, November 2010. ISSN 0305-0548.

[51] J. de Armas, C. León, G. Miranda, and C. Segura. Optimisation of a multi-objective two-dimensional strip packing problem based on evolutionary algorithms. *International Journal of Production Research*, 48(7):2011– 2028, 2009. ISSN 0020-7543 print / 1366-588X online.

[52] J. de Armas, C. León, and G. Miranda. Fine and Coarse-grained Parallel Algorithms for the 2D Cutting Stock Problem. In T. G. Marco Danelutto, Julien Bourgeois, editor, *The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP'2010)*, pages 255–262. IEEE Computer Society, Pisa, Italia, February 2010. ISBN 978-0-7695-3939-3.

[53] J. de Armas, C. León, G. Miranda, and C. Segura. Remote service to solve the two-dimensional cutting stock problem: an application to the Canary Islands costume. *International Journal of Grid and Web Services*, 4(3):342–351, 2008. ISSN 1741-1114 print / 1741-110 online.

[54] J. de Armas, G. Miranda, and C. León. Two encoding schemes for a multi-objective cutting stock problem. In *IEEE Congress on Evolutionary Computation (CEC'2011)*, pages 27–109. IEEE Computer Society, New Orleans, LA, June 2011. ISBN 978-1-4244-7834-7.

[55] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *VI Conference on Parallel Problem Solving from Nature*, volume 1917 of *LNCS*, pages 849–858. Springer, 2000.

[56] DEIS - Operations Research Group. Library of Instances: Bin Packing Problem. http://www.or.deis.unibo.it/research_pages/ORinstances/2CBP.html.

[57] T. Dereli and G. Sena Das. A Hibrid Simulated Annealing Algorithm for Solving Multi-objective COntainer-Loading Problems. *Applied Artificial Intelligence: An International Journal*, 24(5):463–486, 2010.

[58] T. Dereli and G. Sena Das. A hybrid [']bee(s) algorithm' for solving container loading problems. *Applied Soft Computing*, 11(2):2854–2862, 2011. ISSN 1568-4946.

[59] M. H. Didier Fayard and V. Zissimopoulos. An Efficient Approach for Large-Scale Two-dimensional Guillotine Cutting Stock Problems. *JORS*, 49:1270–1277, 1998.

[60] X. Ding, Y. Han, and X. Zhang. A discussion of adaptive genetic algorithm solving container-loading problem. *Periodical of Ocean University of China*, 34(5):844–848.

[61] K. A. Dowsland and W. B. Dowsland. Packing Problems. *European Journal of Operational Research*, 56(1):2–14, 1992.

[62] J. J. Durillo, A. J. Nebro, F. Luna, B. Dorronsoro, and E. Alba. jMetal: a Java Framework for Developing Multi-Objective Optimization Metaheuristics. Technical Report ITI-2006-10, Universidad de Málaga, 2006. URL `http://jmetal.sourceforge.net`.

[63] H. Dyckhoff. A Typology of Cutting and Packing Problems. *European Journal of Operational Research*, 44(2):145–159, 1990.

[64] H. Dyckhoff and U. Finke. *Cutting and Packing in Production and Distribution: A Typology and Bibliography*. Springer, 1992. ISBN 3790806307, 9783790806304.

[65] H. Dyckhoff, H. Kruse, D. Abel, and T. Gal. Trim loss and related problems. *Omega*, 13(1):59–72, 1985.

[66] H. Dyckhoff, G. Scheithauer, and J. Terno. *Annotated bibliographies in combinatorial optimization*, chapter Cutting and packing, pages 393–414. Wiley, New York, 1997.

[67] J. Eckstein, C. A. Phillips, and W. E. Hart. Pico: An object-oriented framework for parallel branch and bound. Technical report, Rutgers University, Piscataway, NJ, 2000.

[68] J. Egeblad, C. Garavelli, S. Lisi, and D. Pisinger. Heuristics for container loading of furniture. *European Journal of Operational Research*, 200(3):881–892, 2010. ISSN 0377-2217.

[69] M. Eley. Solving container loading problems by block arrangement. *European Journal of Operational Research*, 141(2):393–409, 2002. ISSN 0377-2217.

[70] EURO Special Interest Group on Cutting and Packing. http://paginas.fe.up.pt/∼esicup.

[71] L. Faina. A global optimization algorithm for the three-dimensional packing problem. *European Journal of Operational Research*, 126(2):340–354, October 2000.

[72] D. Fayard and V. Zissimopoulos. An approximation algorithm for solving unconstrained two-dimensional knapsack problems. *European Journal of Operational Research*, 84(3):618–632, August 1995.

[73] C. Fonseca and P. J. Fleming. On the performance assessment and comparison of stochastic multiobjective optimizers. In *Parallel Problem Solving from Nature*, LNCS, pages 584–593. Springer-Verlag, 1996.

[74] C. Gagné and M. Parizeau. Open BEAGLE: a new C++ evolutionary computation framework. In *Genetic and Evolutionary Computation Conference (GECCO'2002)*. New York, USA, 2002.

[75] C. Gagné and M. Parizeau. Genericity in Evolutionary Computation Software Tools: Principles and Case Study. *International Journal on Artificial Intelligence Tools*, 15(2):173–194, 2006.

[76] L. García, C. León, G. Miranda, and C. Rodríguez. A Parallel Algorithm for the Two-Dimensional Cutting Stock Problem. In *European Conference on Parallel Computing (Euro-Par'2006)*, volume 4128 of *LNCS*, pages 821–830. Springer-Verlag, Dresden, Germany, 2006.

[77] L. García, C. León, G. Miranda, and C. Rodríguez. Two-Dimensional Cutting Stock Problem: shared memory parallelizations. In *5th International Symposium on Parallel Computing in Electrical Engineering*, pages 438–443. IEEE Computer Society, Bialystok, Poland, September 2006. ISBN 0-7695-2554-7.

[78] P. Garrido and M. C. Riff. Collaboration Between Hyperheuristics to Solve Strip-Packing Problems. In *Foundations of Fuzzy Logic and Soft Computing*, volume 4529 of *LNCS*, pages 698–707. Springer, 2007. ISBN 978-3-540-72917-4. ISSN 0302-9743.

[79] L. D. Gaspero and A. Schaerf. EasyLocal++: an object-oriented framework for the flexible design of local search algorithms and metaheuristics. In *4th Metaheuristics International Conference*, pages 287–292, 2001.

[80] H. Gehring and A. Bortfeldt. A Genetic Algorithm for Solving the Container Loading Problem. *International Transactions in Operational Research*, 4(5-6): 401–418, 1997.

[81] H. Gehring and A. Bortfeldt. A Parallel Genetic Algorithm for Solving the Container Loading Problem. *International Transactions in Operational Research*, 9(4):497–511, 2002. ISSN 1475-3995.

[82] H. Gehring, K. Menschner, and M. Meyer. A computer-based heuristic for packing pooled shipment containers. *European Journal of Operational Research*, 44(2):277–288, 1990. ISSN 0377-2217.

[83] J. A. George and D. Robinson. A heuristic for packing boxes into a container. *Computers & Operations Research*, 7(3):147–156, 1980. ISSN 0305-0548.

[84] P. C. Gilmore and R. E. Gomory. Multistage Cutting Stock Problems of Two and More Dimensions. *Operations Research*, 13(1):94–120, 1965. ISSN 0030364X.

[85] P. C. Gilmore and R. E. Gomory. The Theory and Computation of Knapsack Functions. *Operations Research*, 14:1045–1074, March 1966.

[86] D. E. Goldberg and J. R. Lingle. Alleles\loci and the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 154–159. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1985. ISBN 0-8058-0426-9.

[87] J.-P. Hamiez, J. Robet, and J.-K. Hao. A Tabu Search Algorithm with Direct Representation for Strip Packing. In *Proceedings of the 9th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'2009)*, pages 61–72. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-01008-8.

[88] D. He, J. Cha, and Y. Jiang. Research on solution to complex container loading problem based on genetic algorithm. *Journal of Software*, 12(9):1380–1385, 2001.

[89] K. He and W. Huang. An efficient placement heuristic for three-dimensional rectangular packing. *Computers & Operations Research*, 38:227–233, January 2011. ISSN 0305-0548.

[90] M. Hifi. 2D Cutting Stock Problem Instances. ftp://cermsem.univ-paris1.fr/pub/CERMSEM/hifi/2Dcutting/.

[91] M. Hifi. 2D Strip Packing Problem Instances. ftp://cermsem.univ-paris1.fr/pub/CERMSEM/hifi/Strip-cutting.

[92] M. Hifi. An Improvement of Viswanathan and Bagchi's Exact Algorithm for Constrained Two-Dimensional Cutting Stock. *Computers & Operations Research*, 24(8):727–736, 1997.

[93] M. Hifi. Exact algorithms for the guillotine strip cutting/packing problem. *Computers & Operations Research*, 25(11):925–940, 1998. ISSN 0305-0548.

[94] M. Hifi. Exact algorithms for unconstrained three-dimensional cutting problems: a comparative study. *Computers & Operations Research*, 31:657–674, April 2004. ISSN 0305-0548.

[95] M. Hifi, R. M'Hallah, and T. Saadi. Approximate and exact algorithms for the double-constrained two-dimensional guillotine cutting stock problem. *Computational Optimization and Applications*, 42(2):303–326, March 2009. ISSN 0926-6003.

[96] M. Hifi and V. Zissimopoulos. Constrained Two-Dimensional Cutting: An Improvement of Christofides and Whitlock's Exact Algorithm. *The Journal of the Operational Research Society*, 48(3):324–331, 1997.

[97] A. I. Hinxman. The trim-loss and assortment problems: A survey. *European Journal of Operational Research*, 5:8–18, 1979.

[98] E. Hopper and B. Turton. A Genetic Algorithm for a 2D Industrial Packing Problem. *Computers & Industrial Engineering*, 37:375–378, 1999.

[99] E. Hopper and B. C. H. Turton. A Review of the Application of Meta-Heuristic Algorithms to 2D Strip Packing Problems. *Artificial Intelligence Review*, 16 (4):257–300, 2001. ISSN 0269-2821.

[100] E. Hopper and C. H. Turton. An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem. *European Journal of Operational Research*, 128/1:34–57, 2000.

[101] J. Horn and et al. A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 82–87, 1994.

[102] W. Huang and K. He. A caving degree approach for the single container loading problem. *European Journal of Operational Research*, 196(1):93–101, 2009. ISSN 0377-2217.

[103] S. Illich, L. While, and L. Barone. Multi-objective strip packing using an evolutionary algorithm. In *IEEE Congress on Evolutionary Computation (CEC'2007)*, pages 4207–4214, September 2007.

[104] Z. Jin, K. Ohno, and J. Du. An efficient approach for the three dimensional container packing problem with practical constraints. *Asia-Pacific Journal of Operational Research*, 21(3):279–295, 2004.

[105] M. Jones, G. McKeown, and V. Rayward-Smith. *Optimization Software Class Libraries*, chapter Distribution, Cooperation, and Hybridization for Combinatorial Optimization, pages 25–58. Kluwer Academic Publishers, 2002.

[106] M. Jünger and S. Thienel. Introduction to ABACUS - a branch-and-cut system. *Operations Research Letters*, 22(2-3):83–95, 1998.

[107] L. Kantorovich. Mathematical methods of organizing and planning production. *Management Science*, 6:366–422, 1960 (Russian version appeared in 1939).

[108] J. Karelahti. Solving the cutting stock problem in the steel industry. Master's thesis, Department of Engineering Physics and Mathematics, Helsinki University of Technology, Helsinki, Finland, 2002.

[109] M. Kenmochi, T. Imamichi, K. Nonobe, M. Yagiura, and H. Nagamochi. Exact algorithms for the two-dimensional strip packing problem with and without rotations. *European Journal of Operational Research*, September 2008. ISSN 03772217.

[110] J. Knowles. A summary-attainment-surface plotting method for visualizing the performance of stochastic multiobjective optimizers. In *Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, pages 552–557. IEEE Computer Society, 2005. ISBN 0-7695-2286-06.

[111] J. Knowles and D. Corne. On metrics for comparing nondominated sets. In *IEEE Congress on Evolutionary Computation (CEC'2002)*, volume 1, pages 711–716. IEEE Service Center, May 2002.

[112] B. Kröger. Guillotineable bin packing: A genetic approach. *European Journal of Operational Research*, 84:645–661, 1995.

[113] D. C. S. J. L., N. Y. Soma, and N. Maculan. A greedy search for the three-dimensional bin packing problem: the packing static stability case. *International Transactions in Operational Research*, 10(2):141–153, 2003.

[114] B. Le-Cun and C. Roucairol. BOB : a unified platform for implementing branch-and-bound like algorithms. Technical Report 95/16, Laboratoire PRiSM, Université de Versailles, June 1995.

[115] C. León, G. Miranda, and C. Rodríguez. *Optimization techniques for solving complex problems*, chapter Tools for tree searches: Branch-and-Bound and A* Algorithms, pages 193–208. Wiley, 2008. ISBN 978-0-470-29332-4.

[116] C. León, G. Miranda, C. Rodríguez, and C. Segura. 2D Cutting Stock Problem: a New Parallel Algorithm and Bounds. In *European Conference on Parallel Computing (Euro-Par'2007)*, volume 4641 of *LNCS*, pages 795–804. Springer-Verlag, Rennes, France, 2007. ISBN 978-3-540-74465-8.

[117] C. León, G. Miranda, C. Rodríguez, and C. Segura. A distributed parallel algorithm to solve the 2D cutting stock problem. In *16th Euromicro Conference on Parallel, Distributed and Netword-Based Processing (PDP'2008)*, pages 429–434. IEEE Computer Society, Toulouse, France, February 2008. ISBN 0-7695-3089-3.

[118] C. León, G. Miranda, and C. Segura. Parallel Hyperheuristic: A Self-Adaptive Island-Based Model for Multi-Objective Optimization. In *Genetic and Evolutionary Computation Conference (GECCO'2008)*, pages 757–758. ACM, Atlanta, U.S.A., July 2008. ISBN 978-1-60558-131-6.

[119] C. León, G. Miranda, and C. Segura. METCO: A Parallel Plugin-Based Framework for Multi-Objective Optimization. *International Journal on Artificial Intelligence Tools*, 18(4):569–588, 2009.

[120] S. C. H. Leung and D. Zhang. A fast layer-based heuristic for non-guillotine strip packing. *Expert Syst. Appl.*, 38(10):13032–13042, September 2011. ISSN 0957-4174.

[121] C. León, G. Miranda, E. Segredo, and C. Segura. Librería Paralela de Algoritmos Evolutivos Multi-Objetivo. In *XIX Jornadas de Paralelismo*, pages 3–8. Universitat Jaume I, Castellón, Spain, September 2008. ISBN 978-84-8021-676-0.

[122] C. León, G. Miranda, E. Segredo, and C. Segura. Equipo de Algoritmos Evolutivos con Mecanismo Auto-adaptativo para Optimización Multi-Objetivo. In E. Alba, F. Chicano, F. Luna, and G. Luque, editors, *VI Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'2009)*,

pages 555–562. Universidad de Málaga, Málaga, Spain, February 2009. ISBN 978-84-691-6813-4.

[123] C. León, G. Miranda, E. Segredo, and C. Segura. Parallel Library of Multi-objective Evolutionary Algorithms. In D. E. Baz, F. Spies, and T. Gross, editors, *17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'2009)*, pages 28–35. IEEE Computer Society, Weimar, Germany, February 2009. ISBN 978-0-7695-3544-9.

[124] A. Liefooghe, M. Basseur, L. Jourdan, and E.-G. Talbi. ParadisEO-MOEO: A Framework for Evolutionary Multi-objective Optimization. In *Evolutionary Multi-Criterion Optimization*, volume 4403 of *LNCS*, pages 386–400. Springer, 2007. ISBN 978-3-540-70927-5. ISSN 0302-9743.

[125] A. Lim, B. Rodrigues, and Y. Yang. 3-D Container Packing Heuristics. *Applied Intelligence*, 22:125–134, March 2005. ISSN 0924-669X.

[126] J.-L. Lin, B. Foote, S. Pulat, C.-H. Chang, and J. Cheung. Hybrid genetic algorithm for container packing in three dimensions. In *Artificial Intelligence for Applications, 1993. Proceedings., Ninth Conference on*, pages 353–359, March 1993.

[127] J. Liu, Z. Dong, and G. Ma. Solving container loading based on tabu search algorithm. *Journal of Shenyang University of Technology*, 31(2):212–216, 2009.

[128] J. Liu, Y. Yue, Z. Dong, C. Maple, and M. Keech. A novel hybrid tabu search approach to container loading. *Computers & Operations Research*, 38:797–807, April 2011. ISSN 0305-0548.

[129] A. Lodi, S. Martello, and M. Monaci. Two-Dimensional Packing Problems: A Survey. *European Journal of Operational Research*, 141(2):241–252, September 2002.

[130] A. Lodi and M. Monaci. Integer linear programming models for 2-staged two-dimensional Knapsack problems. *Mathematical Programming*, 94(2):257–278, 2003. ISSN 0025-5610.

[131] T. H. Loh and A. Y. C. Nee. A packing algorithm for hexahedral boxes. pages 115–126, 1992.

[132] D. Mack, A. Bortfeldt, and H. Gehring. A parallel hybrid local search algorithm for the container loading problem. *International Transactions in Operational Research*, 11(5):511–533, 2004. ISSN 1475-3995.

[133] S. Martello, M. Monaci, and D. Vigo. Two-dimensional strip packing problem. http://www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm.

[134] S. Martello, M. Monaci, and D. Vigo. An Exact Approach to the Strip-Packing Problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003. ISSN 1526-5528.

[135] S. Martello, D. Pisinger, and D. Vigo. The Three-Dimensional Bin Packing Problem. *Operations Research*, 48(2):256–267, 2000.

[136] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons Ltd, 1990.

[137] N. McPhee, N. Hopper, and M. Reierson. Sutherland: an extensible object-oriented software framework for evolutionary computation. In J. Koza, editor, *Proceedings of the Third Annual Conference on Genetic Programming*, page 241. Morgan Kaufmann, University of Wisconsin, Madison, USA, 1998.

[138] G. Miranda and C. León. An OpenMP skeleton for the A* heuristic search. In Springer-Verlag, editor, *High Performance Computing and Communications*, volume 3726 of *LNCS*, pages 717–722. Naples, Italy, September 2005. ISBN 3-540-29031-1. ISSN 0302-9743.

[139] G. Miranda and C. León. OpenMP skeletons for tree searches. In *14th Euromicro Conference on Parallel, Distributed and Network-based Processing (PDP'2006)*, pages 423–430. IEEE Computer Society, Montbeliard-Sochaux, France, February 2006. ISBN 0-7695-2513-X.

[140] G. Miranda Valladares. *Esquemas algorítmicos para optimización combinatoria. Paralelizaciones y aplicaciones*. PhD. thesis, Universidad de La Laguna, Dpto. Estadística, I.O. y Computación, 2009. ISBN 978-84-7756-878-0.

[141] P. Moscato and C. Cotta. An Introduction to Memetic Algorithms. *Revista Iberoamericana de Inteligencia Artificial*, 19(2):131–148, 2003.

[142] A. Moura and J. F. Oliveira. A GRASP Approach to the Container-Loading Problem. *IEEE Intelligent Systems*, 20(4):50–57, July 2005. ISSN 1541-1672.

[143] C. L. Mumford-Valenzuela, J. Vick, and P. Y. Wang. *Metaheuristics: computer decision-making*, chapter Heuristics for large strip packing problems with guillotine patterns: an empirical study, pages 501–522. Kluwer Academic Publishers, 2004. ISBN 1-4020-7653-3.

[144] C. Múñoz, M. Sierra, J. Puente, C. R. Vela, and R. Varela. Improving Cutting-Stock Plans with Multi-objective Genetic Algorithms. In *Bio-inspired Modeling of Cognitive Tasks*, volume 4527, pages 528–537. Springer Berlin/Heidelberg, 2007. ISBN 978-3-540-73052-1. ISSN 0302-9743 (Print) 1611-3349 (Online).

[145] V. Neidlein, A. Vianna, M. Arenales, and G. Wäscher. Two-Dimensional Guillotineable-Layout Cutting Problems with a Single Defect - An AND/OR-Graph Approach. In *Operations research proceedings 2008*, pages 85–90. Springer-Verlag Berlin Heidelberg, Augsburg, 2008. ISBN 978-3-642-00141-3.

[146] N. Ntene and J. Van Vuuren. A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem. *Discrete Optimization*, 6(2): 174–188, January 2009.

[147] J. Oliveira and J. Ferreira. An improved version of Wang's algorithm for two-dimensional cutting problems. *European Journal of Operational Research*, 44: 256–266, 1990.

[148] T. Ono and T. Ikeda. Optimization of two-dimensional guillotine cutting by genetic algorithms. In H. J. Zimmermann, editor, *European Congress on Intelligent Techniques and Soft Computing*, volume 1, pages 7–10, 1998.

[149] OR-Library. http://people.brunel.ac.uk/∼mastjjb/jeb/info.html.

[150] PackLib^2. http://mo.math.nat.tu-bs.de/packlib.

[151] V. Pareto. *Cours d'Économie Politique*, volume I and II. Lausanne, 1896.

[152] F. Parreño, R. Alvarez-Valdes, J. Oliveira, and J. Tamarit. An hybrid GRASP/VND algorithm for two- and three-dimensional bin packing. *Annals of Operations Research*, 179:203–220, 2010. ISSN 0254-5330.

[153] F. Parreño, R. Alvarez-Valdes, J. F. Oliveira, and J. M. Tamarit. Neighborhood structures for the container loading problem: a VNS implementation. *Journal of Heuristics*, 16:1–22, February 2010. ISSN 1381-1231.

[154] D. Pisinger. Heuristics for the container loading problem. *European Journal of Operational Research*, 141(2):382–392, 2002.

[155] L. Pál. A Genetic Algorithm for the Two-dimensional Single Large Object Placement Problem, 2006.

[156] A. Rahmani and N. Ono. An Evolutionary Approach to Two-Dimensional Guillotine Cutting Problem. In *IEEE International Conference on Evolutionary Computing*, volume 1, pages 148–151. Perth, WA , Australia, November 1995. ISBN 0-7803-2759-4.

[157] K. Ratanapan and C. Dagli. An object-based evolutionary algorithm: The nesting solution. In *International Conference on Evolutionary Computation*, pages 581–586. IEEE Computer Society, 1998.

[158] O. M. Saad, M. K. El-Shafei, and L. E. Ezzat. On Treating Multiobjective Cutting Stock Problem in the Aluminum Industry under Fuzzy Environment. *Trends in Applied Sciences Research*, 2(5):374–385, 2007.

[159] H. M. Salkin and C. A. de Kluyver. The Knapsack Problem: A survey*. *Naval Research Logistics Quartely*, (22):127–144, 1975.

[160] Y. Sawaragi, H. Nakayama, and T. Tanino. *Theory of multiobjective optimization*. Academic Press, Orlando, 1985.

[161] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In J. J. Grefenstette, editor, *International Conference on Genetic Algorithms and Their Applications*, pages 93–100, 1985.

[162] G. Scheithauer. Algorithms for the Container Loading Problem. In *Operations Research Proceedings 1991*, pages 445–452. Springer-Verlag, 1992.

[163] B. Shneiderman and C. Plaisant. *Diseño de Interfaces de Usuario. Estrategias para una Interacción Persona-Computador Efectiva*. Pearson Education, 2006.

[164] E. Silva, F. Alvelos, and J. Valério de Carvalho. An integer programming model for two- and three-stage two-dimensional cutting stock problems. *European Journal of Operational Research*, 205(3):699–708, 2010. ISSN 0377-2217.

[165] S.-M. Soak, S.-W. Lee, and M. Jeon. The improved adaptive link adjustment evolutionary algorithm for the multiple container packing problem. *Applied Intelligence*, 33:144–158, October 2010. ISSN 0924-669X.

[166] X. Song, C. Chu, Y. Nie, and J. Bennell. An iterative sequential heuristic procedure to a real-life 1.5-dimensional cutting stock problem. *European Journal of Operational Research*, 175(3):1870–1889, 2006.

[167] N. Srinivas and K. Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, 1994. URL `citeseer.ist.psu.edu/srinivas94multiobjective.html`.

[168] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. John Wiley, New York, 1986.

[169] P. E. Sweeney and E. R. Paternoster. Cutting and Packing Problems: A categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

[170] H. Terashima-Marin, A. Moran-Saavedra, and P. Ross. Forming hyper-heuristics with GAs when solving 2D-regular cutting stock problems. In *IEEE Congress on Evolutionary Computation (CEC'2005)*, volume 2, pages 1104–1110, September 2005.

[171] S. Tiwari and N. Chakraborti. Multi-objective optimization of a two-dimensional cutting problem using genetic algorithms. *Journal of Materials Processing Technology*, 173:384–393, 2006.

[172] S. Tschöke and T. Polzer. Portable Parallel Branch-and-Bound Library - PPBB-LIB, 1996. http://wwwcs.uni-paderborn.de/fachbereich/AG/monien/SOFTWARE/PPBB/ppbblib.html.

[173] F. Vasko. A computational improvement to Wang's two-dimensional cutting stock algorithm. *Computers & Industrial Engineering*, 16(1):109–115, 1989.

[174] F. Vasko and C. Bartkowski. Using Wang's two-dimensional cutting stock algorithm to optimally solve difficult problems. *International Transactions in Operational Research*, 16:829–838, 2009.

[175] K. V. Viswanathan and A. Bagchi. Best-First Search Methods for Constrained Two-Dimensional Cutting Stock Problems. *Operations Research*, 41(4):768–776, 1993.

[176] S. Voss and D. Woodruff, editors. *Optimization Software Class Libraries*, volume 18 of *Operations Research and Computer Science Interfaces*. Kluwer Academic Publishers, 2002.

[177] A. Wai-Sing Loo. *Peer-to-Peer Computing: Building Supercomputers with Web Technologies*. Computer Communications and Networks. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 1846283817.

[178] P. Y. Wang. Two Algorithms for Constrained Two-Dimensional Cutting Stock Problems. *Operations Research*, 31(3):573–586, May-June 1983.

[179] P. Y. Wang and C. L. Valenzuela. Data set generation for rectangular placement problems. *European Journal of Operational Research*, 134(2):378–391, October 2001.

[180] Z. Wang, K. W. Li, and J. K. Levy. A heuristic for the container loading problem: A tertiary-tree-based dynamic space decomposition approach. *European Journal of Operational Research*, 191(1):86–99, November 2008.

[181] G. Wäscher. An LP-based approach to cutting stock problems with multiple objectives. *European Journal of Operational Research*, 44(2):175–184, January 1990.

[182] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3): 1109–1130, December 2007.

[183] Y. Wu, W. Li, M. Goh, and R. de Souza. Three-dimensional bin packing problem with variable bin height. *European Journal of Operational Research*, 202(2):347–355, 2010.

[184] C. Yaodong, Y. Yuli, C. Xian, and S. Peihua. A recursive branch-and-bound algorithm for the rectangular guillotine strip packing problem. *Computers & Operations Research*, 35(4):1281–1291, 2008.

[185] M. Ye and G. Zhou. A local genetic approach to multi-objective, facility layout problems with fixed aisles. *International Journal of Production Research*, 45 (22):5243–5264, November 2006.

[186] L. H. W. Yeung and W. K. S. Tang. A hybrid genetic approach for container loading in logistics industry. *IEEE Transactions on Industrial Electronics*, 52 (2):617–627, 2005.

[187] J. Zhihong, I. Takahiro, and O. Katsuhisa. The Three-Dimensional Bin Packing Problem and Its Practical Algorithm. *JSME International Journal Series C Mechanical Systems, Machine Elements and Manufacturing*, 46(1):60–66, 2003.

[188] W. Zhou-jing and L. Kevin W. Layer-layout-based heuristics for loading homogeneous items into a single container. *Journal of Zhejiang University SCIENCE A*, 8(12), 2007. ISSN 1673-565X, 862-1775.

[189] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.

[190] E. Zitzler and S. Künzli. Indicator-Based Selection in Multiobjective Search. In *VIII Conference on Parallel Problem Solving from Nature*, volume 3242 of *LNCS*, pages 832–842. Springer, 2004.

[191] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimization and Control*, pages 19–26. CIMNE, Barcelona, Spain, 2002.

[192] E. Zitzler and L. Thiele. An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach. Technical Report 43, Computer Engineering and Networks Laboratory (TIK), 1998.

[193] E. Zitzler and L. Thiele. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In *Parallel Problem Solving from Nature*, volume 1498 of *LNCS*, pages 292–301. Springer, 1998.

[194] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.